一般社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report
VLD2018-82,CPSY2018-92,RECONF2018-56(2019-01)

FiCSW 上での部分再構成の評価

山倉　美穂† 　畔上　佳太† 　武者　千嵯†

天野　英晴†

† 慶應義塾大学　〒 223–0061 神奈川県港北区日吉 3–14–1
E-mail: †{miho,azegami,mushak,hunga}@am.ics.keio.ac.jp

あらまし　FiC(Flow-in-Cloud) は安価でスケーラブルな性能をクラウド内で実現するマルチ FPGA システムである．FiC においては複数の FPGA を回線交換ネットワークによってつなぐことによって，HLS(High Level Synthesis) 設計のレベルではひとつの大きな FPGA のように動作させる．しかし，FPGA に載せるアプリケーションを変えようとして一枚のボード全体をコンフィギュレーションし直してしまうと，スイッチ部分の動作も止まってしまい他のボードにも影響を及ぼしてしまう．そこで本報告では FiC のボード上で部分再構成を可能にすることによってスイッチ部分の動作を止めずにアプリケーションを入れ替えることを可能にした．また，このときの実装について設計やコンフィギュレーションに要する時間などを評価した．

キーワード　マルチ FPGA，部分再構成，動的再構成，評価，STDM スイッチ，FPGA-in-Cloud

The Evaluation of Partial Reconfiguration for FiCSW

Miho YAMAKURA†, Keita AZEGAMI†, Kazusa MUSHA†,

and Hideharu AMANO†

† Keio University　3–14–1 Hiyoshi, Kouhoku-ku, Yokohama, Kanagawa, 223–0061 Japan
E-mail: †{miho,azegami,mushak,hunga}@am.ics.keio.ac.jp

1. はじめに

近年，FPGA(Field Programmable Gate Array) は，特定の処理を高速化するアクセラレータとしての役割が注目され，強力な浮動小数演算機能を持つ Intel Stratix-10 や，大規模なオンチップ RAM を持つ Xilinx Virtex-Ultrascale+などが，AI を中心としてそれぞれが得意とする応用分野で利用されている．しかし、これらの FPGA の多くは高額であり，ボードを購入して利用することができるユーザは限定される．そこで，FPGA を GPU や TPU など他のアクセラータ同様，クラウドで利用する試みが盛んであり [1]，Amazon EC2 F1 Instance など、商用での利用も可能になっている．しかし、現在の FPGA-in-Cloud は、単一 FPGA の利用技術が中心に研究開発が行われており，ユーザから見ると，その FPGA のサイズと資源の限界以上のものを利用できず，大きな資源をサポートする強力な FPGA を利用する場合，やはりコストは大きなものとなる．

そこで、新エネルギー・産業技術総合開発機構 (NEDO) の「省電力 AI エンジンと異種エンジン統合クラウドによる人工知能プラットフォーム」プロジェクトでは、ミドルエンドのコスト性能比の高い FPGA を安価なシリアルリンクを多数用いて接続した大規模なマルチ FPGA システムである Flow-in-Cloud(FiC) [2] を開発している．FiC においては，高速シリアルリンクと STDM スイッチによって複数の FPGA 間に回線区間ネットワークを構築することにより，HLS 設計レベルから見てひとつの大きな FPGA のように動作させる．こうすることによって，実質上，無限に近い FPGA 資源を安価かつ高い電力性能で利用可能である。

しかし，この複数の FPGA を一つの FPGA のように動作させるシステムにおいて，動作途中で各ボードに載せるアプリケーションを変更することを考えると問題が生じる．それは，アプリケーション部分を変更しようとしてボード全体をコンフィギュレーションし直すとスイッチ部分の動作が止まってしまうということである．スイッチ部分が動作しなくなるとコンフィギュレーションをしているボードだけでなく，他のボードの動作も止めてしまう．そこで，本報告では FiC システムにおいて部分再構成を行うことによって，スイッチ部分を動作させたままアプリケーション部分を入れ替えることができるようにする．また，部分再構成ができるように設計を行う場合，入れ

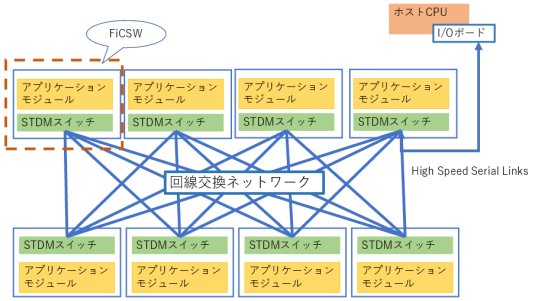

図 1　FiC システムの構成

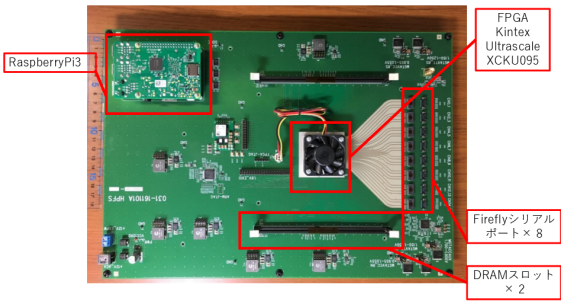

図 2　FiC-SW1 のボード写真

替えるアプリケーション部分以外のスタティックの部分の設計は変更する必要がないので配置配線などの設計にかかる時間を短縮でき，ビットストリームをコンフィギュレーションするときもアプリケーション部分のみコンフィギュレーションし直せばよいので情報量と時間を削減することができる．

FPGA 上で部分再構成を行う，ということについては様々な研究が行われてきている．例えば，[1] においては部分再構成を利用して FPGA をクラウドに統合している．この研究のシステムでは，一つの FPGA を部分再構成によって，独立して動作する領域に分けることによって複数のユーザが同時に利用できるようにしている．[1] においては一つの FPGA を複数のユーザで共有することを考えているのに対して，[3] においては部分再構成を利用して，複数のタスク間で FPGA を共有することを考えている．また，[4] では様々な FPGA において効率的に部分再構成を利用できるようにするツールが紹介されている．他にも，[5] においては FPGA を通信のためのスイッチとして利用し，余ったロジックで GPU などの演算を補助するアクセラレータを実装し，そのアクセラレータを部分再構成を利用して入れ替えられるようにしている．また，マルチ FPGA システムについても様々な研究が行われており，[6] や [7] においてはマルチ FPGA システムのための環境やアルゴリズムが提案されている．

本報告では，まず FiC システムの概要について説明し，つづいて部分再構成を実現するための実装について述べる．そして，設計やコンフィギュレーションに要した時間などについて評価した結果を示し，最後に結論を述べる．

2.　Flow-in-Cloud(FiC)

2.1　FiC プロジェクトの概要

図 1 は FiC システム [2] の構成の概要を示している．FiC を構成する各 FPGA ボードのことは FiCSW と呼び，このボード同士を高速シリアルリンクによって接続する．そして，各ボード上の STDM スイッチモジュールにより回線交換ネットワークを構築している．こうすることによって，各ボード上のアプリケーションモジュールを高速かつ低遅延で相互に接続することができ，全体で一つの大きな FPGA のように動作させることができる．

2.2　FiC-SW1

FiC のプロトタイプボードである FiC-SW1 について説明する．FiC-SW1 のボードの写真を図 2 に，簡単なハードウェア

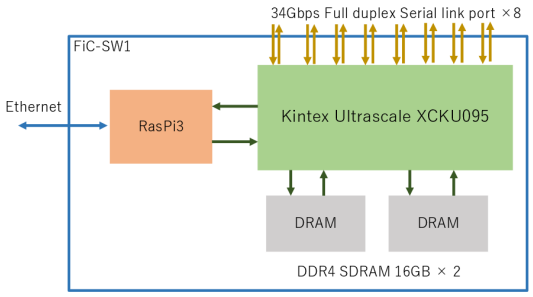

図 3　FiC-SW1 のブロック図

の構成を図 3 に示す．FPGA には Xilinx Kintex Ultrascale XCKU095 を採用することにより，スイッチなどに資源を利用しても，再構成領域内に大規模なアプリケーションの実装に必要な資源を確保できるようにしている。そして，ローカルにデータを格納するために二つの DRAM スロットが装備されている．また，ボード間通信のために X4 双方向の FireFly Micro Flyover システムが 8 ポート (1 ポートあたり，最大 9.9Gbps のフラットケーブル 4 本) 装備されている．そして，FPGA へのコンフィギュレーションのために RaspberryPi3 ボードがドーターボードの形で接続されている．

2.3　STDM スイッチ

FiC システムにおいては，ボード間の通信方式としてサーキットスイッチングである STDM(Static Time Division Multiplexing) を採用していて，この点がこのシステムの大きな特徴の一つである．通常のアプリケーションではパケットスイッチングよりも不利であると考えられている STDM を採用する理由は，STDM は時間ごとに一定の通信を行うため，あらかじめモジュール間の転送にかかる時間が予測可能で，FiC がターゲットとしているニューラルネットワークのような同期が必要かつデータサイズが事前に明らかであるようなアプリケーションでは，この予測可能であるという特性が有効に働くと考えられるからである．図 4 に FiC-SW1 に実装されている STDM スイッチの概略を示す．ただし，この図の例ではポート数とスロット数はともに 3 となっている．それぞれのポートから入ってくるデータの宛先はスロットごとに決まるようになっている．例えばポート 2 に入ったデータは一度 FIFO IP に保存されて，スロット 0 になるとポート 0 へと転送される．このような割り当ては各スイッチ内のルーティングテーブルによって決定され，そのルーティングテーブルは RaspberryPi からコンフィ

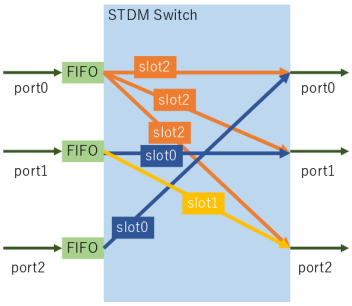

図 4　STDM スイッチの概略図 (ポート数とスロット数がともに 3 の場合)

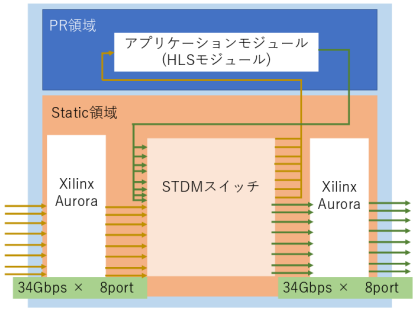

図 5　FiC-SW 上でのスタティック領域と PR 領域

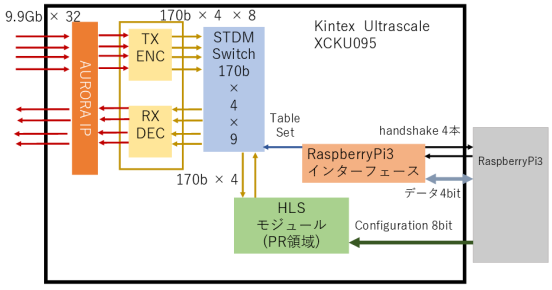

図 6　FiC-SW1 の FPGA の構成

ギュレーションの後に設定ができるようになっており，これによって STDM スイッチはソースノードから宛先ノードへと転送を行うことができる．また，ポート 0 からはスロット 2 ですべてのポートに対してブロードキャストを行う．こうすることによって，最大スロット数以内であればスロット数はパラメータで与えられるようになる．それだけでなく，深層学習のモジュールが複数のノード上に分散して実装される場合にはブロードキャストが必要になるので，そのような場面でも役に立つ．この STDM スイッチの詳細については参考文献 [8] や [9] に示されている．

3.　FiCSW-1 上での部分再構成

3.1　スタティック領域と PR 領域

本研究においては，FiC-SW の FPGA を大まかに述べて二つの領域に分けて実装した．一つは STDM スイッチやその他のボード間の通信に利用する IP モジュール (Xilinx AURORA) が含まれる，プラットフォームを維持するために途中で変更されることなく常に動き続けるスタティック領域である．もう一つは入れ替え可能なアプリケーションモジュールを実装する部分再構成 (Partial Reconfiguration，以後 PR) 領域である．このように実装することにより，ボード間の通信を維持しながら複数のアプリケーションを動的に再構成することができる．FiC では回線交換方式を採用しているため，一つの FPGA 全体を再構成してスイッチモジュールなどのボード間の通信の役割を担うモジュールが経路上から失われると他のボード上のアプリケーションも動作しなくなる可能性がある．そのため，このようにして再構成の間もスタティック領域は動作し続けるようにする必要がある．図 5 はスタティック領域と PR 領域の接続のイメージ図である．STDM スイッチのポートのうち一つ以上をローカルの PR 領域に接続する．こうすることによって，PR 領域内のアプリケーションモジュールが回線交換ネットワークを利用できるようになる．また，ボード上の RaspberryPi とアプリケーションが IP を通して DRAM を利用するために使用するデータ線も PR 領域に接続した．

3.2　ボード間の通信と HLS モジュールの組み込みの方法

FiC-SW1 においては PR 領域に実装するアプリケーションモジュールは Vivado-HLS で記述された HLS モジュールとする．アプリケーションを構成する各 HLS モジュールを，スタティック領域とのインターフェースが記述された wrap モジュールでまとめて，それを PR 領域において入れ替えられるようにした．また，HLS で記述されたアプリケーションはすべて ap_rst，ap_start，ap_done などの標準インターフェースを持つのでこれらを利用して各モジュールの制御を行うようにした．FiC-SW1 の FPGA の構成を図 6 に示す．FiC-SW1 のシリアルリンクは 4 本で一組のフラットケーブル 8 組であり，これが Xilinx のシリアル転送用の IP Aurora を用いて一本当たり最大 9.9 Gbps のシリアル転送を行うことができるようになっている．ただし，64b/66b エンコーディングを行い，ECC を装備しているので IP により 85bit 幅に展開して，100MHz で交換を行う (実際には 1 クロックに 1 回 85bit 幅のデータを転送しているわけではなく，170bit 幅のデータを 2 クロックに 1 回転送している) ことになるので実質的な転送速度は 1 本当たり 8.5Gbps となる．先ほども述べたようにこのリンクは 4 本で 1 組のケーブルを用いて接続するので，実際には図 3 や図 5 に示されているように，ボード間は 1 系統当たり 8.5G × 4=34Gbps のチャネル 8 系統で接続していると言える．このチャネル内の 4 本のリンクについては完全に同期が取れているわけではないため，FiC-SW1 においては 170bit 幅のスイッチを 4 セット持つようにしている．また，PR 領域内の wrap モジュールは STDM スイッチのポート 0 に直接接続している．

3.3　部分再構成 (PR) のデザインフロー

図 7 に部分再構成を可能にするためのデザインフローを示す．部分再構成のためにはまず，HLS 部分を空のダミーのモジュールにして全体を合成し，チェックポイントに保存する．HLS モジュールの一つ (PR 領域の決定に利用するので入れ替えるモジュールの中で一番資源の使用量の多いものを選ぶ) を同様に合成して，チェックポイントに保存する．そして，最初

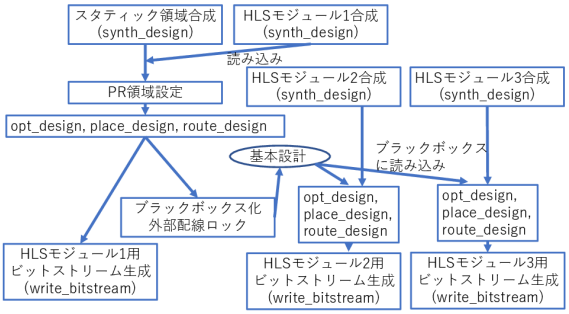

図 7　PR のデザインフロー

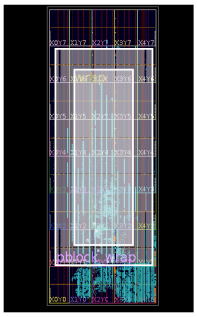

図 8　PR 領域設定の例

に合成した全体の設計の HLS 部分に合成した HLS モジュールをはめ込み，GUI を立ち上げて PR 領域の設定を行う．今回は PR 領域に最大でどれだけの資源を利用できるか評価するために，HLS モジュールに必要な大きさ以上にできる限り大きく PR 領域を設定した (スタティック領域に必要な資源を確保できなくなるほど PR 領域を大きくとってしまうとエラーが出るので，エラーの出ない最大の大きさとした)．このときの様子を図 8 に示す．PR 領域の位置や大きさを変更する必要がなければ入れ替える他のモジュールの設計を行うときにはこの操作を行う必要はない．この設計データに対して通常の Implement 操作 (opt_design，place_design，route_design) を実行し，その結果から Partial Reconfiguration 設定を行って bitmap ファイルを生成すると，スタティック領域も含めた全体のコンフィギュレーションファイル，PR 領域をクリアするためのクリアファイル，PR 領域のみのコンフィギュレーションファイルの 3 種類のコンフィギュレーションファイルを生成することができる．そして，ここまでの設計データの HLS 部分をブラックボックス化し，スタティック領域の配線をロックする．こうすることによって，HLS 部分を様々な HLS モジュールで入れ替えることのできるテンプレートが出来上がる．これを利用すれば，入れ替えを行う他の HLS モジュールについてもそれぞれを合成してチェックポイントに保存し，それをブラックボックス化したモジュールと入れ替えて通常の Implement 操作を行えば 3 種類のコンフィギュレーションファイルを生成することができる．このように，部分再構成の設計は PR 領域を変更しなければ HLS モジュールのみ合成，配置配線をすればよい (一応全体での配置配線を行うがスタティック領域の配線はロックされている) ので後述するように配置配線にかかる時間を大幅に削減することができる．

また，FiC においてアプリケーションを実行するときには以下のような手順になる．

（1） RaspberryPi3 は ap_rst を L にして HLS モジュールの動作を停止する．

（2） PR 領域の構成のみを初期化するクリア用のコンフィギュレーションデータを転送する．

（3） 入れ替える HLS モジュールの PR 領域のみのコンフィギュレーションデータを転送する．

（4） ap_rst と ap_start を実行してアプリケーションをスタートさせる．

4. 部分再構成の評価

4.1 評価環境

PR 領域に入れるモジュールとして簡単な畳み込みニューラルネットワーク (CNN) の推論フェーズの演算を FiC 上で行うためのモジュールを用意して， 3.3 で紹介したデザインフローに沿って各モジュールの設計にかかる時間と生成されたコンフィギュレーションデータをコンフィギュレーションするのに要する時間を計測した．今回用意したモジュールは，ボードを 1 枚のみ使うときのモジュール (シングル)，ボードを複数枚利用するときの CNN の演算を行うモジュール (マルチ演算)，2 枚のボードを利用するときに画像データや結果の管理をするモジュール (マルチ入出力 2 ノード)，4 枚のボードを利用するときに同様の動作をするモジュール (マルチ入出力 4 ノード) の 4 種類である．本研究で実装したアプリケーションよりも大規模なものであるが， [10] において FiC 上で CNN のアプリケーションを動作させた例が紹介されている．また，STDM スイッチのスイッチサイズについても 3 × 3 から 9 × 9 まで変化させてそれぞれについて設計を行った．つまり，スタティック領域について 7 種類のスイッチサイズでそのそれぞれについて PR 領域に入れるモジュールを 4 種類用意して各々の場合について設計とコンフィギュレーションにかかる時間を評価した．また，各スイッチサイズにおいて PR 領域で利用できる資源の最大量も評価した．表 1 に設計や評価の環境をまとめた．コンフィギュレーションにかかる時間については図 9 に示す Prototype-FiC を使って計測した．Prototype-FiC は FiC-SW1 のボード 4 枚とクロックボードから構成されていて，そのうちの一枚にデータをコンフィギュレーションした．また，DRAM は動作させなかった．

表 1　実装と評価の環境

デザインツール	Vivado HLS Version 2017.2(Xilinx)
実装	Vivado Version 2017.2(Xilinx)
FPGA	Kintex UltraScale XCKU095-FFVB2104(Xilinx)
周波数	100MHz

4.2 PR 領域内に確保できる資源の量

3.3 で紹介したデザインフローで，各スイッチサイズについて設計を行ったところ，3 × 3 から 7 × 7 のときは最大でとることのできる PR 領域のサイズがすべて同じであったが，8 × 8 と 9 × 9 のときはそのサイズで PR 領域をとろうとするとエラーが出たため，これらのときは領域のサイズを他のスイッチ

図 9　Prototype-FiC

サイズのときと比べて小さくした (8 × 8 と 9 × 9 のときは同じサイズだった)．つまり，スイッチサイズによって PR 領域内に確保できる資源の量が二通りあった．この二通りの PR 領域内の資源の量と，ボード全体の資源の量 (すべてボード 1 枚あたり) を表 2 に示す．また，二通りの場合それぞれの PR 領域内の資源の量の全体の資源の量に対する割合を表 3 に示す．表 3 を見ると，資源の量の少ないスイッチサイズ 8 ×8 から 9×9 の場合でも PR 領域内に，各資源について全体の 6 割前後を確保できていることがわかる．そして，表 4 は 4 種類の中で一番資源の使用量の多いモジュールであるマルチ演算モジュールが PR 領域の大きさの異なる二つの場合それぞれにおいて，PR 領域内の資源をどれだけ使っているかの割合を示している．これを見ると，今回の簡単な CNN のモジュールであれば余裕をもって実装できるだけの資源を PR 領域内に確保できていることがわかる．また，表 5 に FPGA 上に，より大規模なニューラルネットワークのシステムを実装している他の研究における各資源の使用量を示す．ただし，表中の空欄は文献中に記述のなかったものである． [11] は NullHop というカーネルサイズをフレキシブルに変更でき，一つの層で最大 128 の入力を並列に処理できるような CNN のシステムを FPGA 上に実装した研究で， [12] はフレキシブルで高性能な TNN(Ternary Neural Network) のシステムを FPGA 上に実装した研究である．表 2 と表 5 を見比べると， [11] において使用されている資源の量を 1 枚の FiC-SW1 上の PR 領域で上回っており，また [12] についてもボードを複数使えばこのシステムに必要な資源の量は FiC 上で確保できることがわかる．示したのは一部の研究だけではあるが，関連研究での資源の使用量を考えても，本報告の実装においては部分再構成を可能とすると同時に PR 領域に十分な資源を用意できていることがわかる．

表 2　再構成領域内の各資源.

	スイッチサイズ 3 ×3 から 7 ×7 のときの PR 領域	スイッチサイズ 8 ×8 から 9×9 のときの PR 領域	全体
LUT as logic	353120	318720	537600
LUT as memory	50400	46080	76800
CLB Registers	706240	637440	1075200
BRAM	1260	1152	1680
DSP	560	512	768

表 3　再構成領域内の各資源の全体に対する割合

	スイッチサイズ 3 ×3 から 7 ×7	スイッチサイズ 8 ×8 から 9×9
LUT as logic	65.68%	59.29%
LUT as memory	65.63%	60.00%
CLB Registers	65.68%	59.29%
BRAM	75.00%	68.57%
DSP	72.92%	66.67%

表 4　PR 領域内でのマルチ演算モジュールの資源使用率

	スイッチサイズ 3 ×3 から 7 ×7	スイッチサイズ 8 ×8 から 9×9
LUT as logic	1.75%	2.31%
LUT as memory	0.22%	0.24%
CLB Registers	0.72%	0.8%
BRAM	32.82%	35.89%
DSP	4.29%	4.69%

表 5　関連研究における資源の使用量

	NullHop [11]	TNN [12]
LUT as logic	229000	275042
LUT as memory		84810
CLB Registers	107000	
BRAM	386	2187
DSP	128	

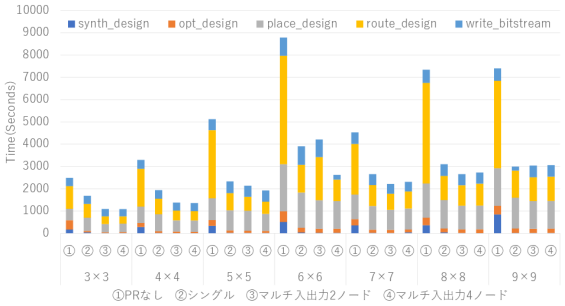

図 10　設計の各フェーズに要した時間

4.3　設計に要した時間

図 10 に，各スイッチサイズにおいて，各モジュールの各設計フェーズに要した時間を示す．ただし，図中の PR なしとされているところについては図 7 のデザインフローの中の HLS モジュール 1 のところを資源の使用量の多いマルチ演算モジュールとして，スタティック領域も含めて全体を設計した場合に要した時間を示しており，ほかのモジュールについてはデザインフローの HLS モジュール 2 や 3 のようにスタティック領域の設計はテンプレートを利用し，主に PR 領域内のみ新しく設計した場合に要した時間を示している．図 10 より，多くのスイッチサイズにおいて部分再構成を行った場合は行わない場合と比べて設計時間を半分程度に抑えられることがわかる．また，部分再構成を行う場合に PR 領域内のアプリケーションが異なると設計に要する時間も多少異なることがわかるが，いずれのモジュールについても PR を行わない場合と比べれば設計に要する時間は大幅に削減できている．

4.4　コンフィギュレーションに要した時間

表 6 に Total(全体)，Partial(PR 用)，Clear(クリア用) の 3 種類のコンフィギュレーションファイルのサイズを示す．ただし，このサイズはスイッチサイズやモジュールが変化しても変

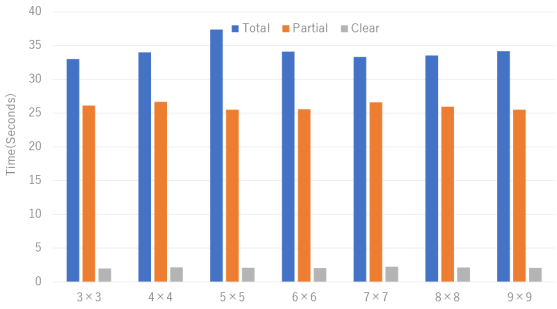

図 11　シングルモジュールの各ファイルのコンフィギュレーションに要した時間 (スイッチサイズごと)

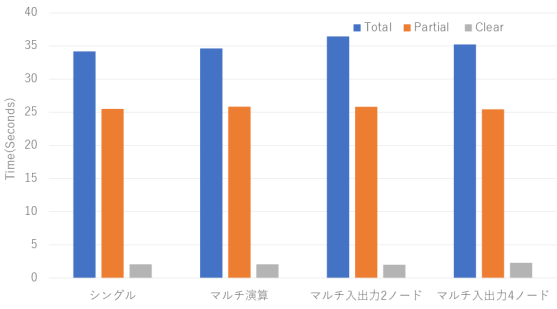

図 12　スイッチサイズが 9 × 9 のときの各モジュールの各ファイルのコンフィギュレーションに要した時間

わらなかった．また，図 11 に，各スイッチサイズで，PR 領域内にシングルモジュールを入れて生成した各コンフィギュレーションファイルのコンフィギュレーションに要した時間を示しており，図 12 にスイッチサイズが 9 × 9 のときに，PR 領域に各モジュールを入れて生成したファイルをコンフィギュレーションするのに要した時間を示している．つまり，図 11 はモジュールの種類を固定してスイッチサイズを変えた結果を示しており，図 12 はスイッチサイズを固定してモジュールを変えた結果を示している．これらを見ると，コンフィギュレーションに要する時間はスイッチサイズやモジュールの種類を変えてもはあまり変わらず，概ねファイルサイズに比例して変化することがわかり，全体をコンフィギュレーションする場合と PR をする場合を比べると要する情報量も時間も 8 割弱に抑えられていることがわかる．

表 6　各コンフィギュレーションファイルのサイズ

	Total	Partial	Clear
Data Size(MB)	34.18	26.60	1.76

5. ま と め

クラウドにおいて安価でスケーラブルな性能を実現するマルチ FPGA システムである FiC では，複数の FPGA を一つの大きな FPGA のように動作させることを考えており，それを可能にするためにスイッチ部分を動作させたままアプリケーションを入れ替える，という部分再構成を行った．スタティック領域の STDM スイッチのサイズや PR 領域に入れるモジュールを変えながら設計を行い，その結果生成したファイルを FiC-SW 上の FPGA にコンフィギュレーションして，設計やコンフィギュレーションにかかる時間や使用する資源の量を評価した．その結果，スイッチサイズを大きくしても PR 領域内に十分量の資源を確保できること，部分再構成を行うと設計に要する時間は半分程度に，コンフィギュレーションに要する情報量と時間が 8 割弱に抑えられることがわかった．また，今後は FiCSW 上の PR 領域に，より大規模なアプリケーションを入れて部分再構成を行うことを目指している．

謝　　辞

この成果は，国立研究開発法人新エネルギー・産業技術総合開発機構 (NEDO) の委託業務の結果得られたものです．

文　　献

[1] S. Byma, J.G. Steffan, H. Bannazadeh, A.L. Garcia, and P. Chow, “FPGAs in the cloud: Booting virtualized hardware accelerators with openstack,” 2014 IEEE 22nd Annual International Symposium on Field-Programmable Custom Computing Machines, pp.109–116, May 2014.

[2] 工藤知宏，高野了成，天野英晴，鯉渕道紘，松谷宏紀，塙　敏博，池上　努，須崎有康，田中　哲，赤沼領大，並木　周，田浦健次朗，“データの流れに着目した異種エンジン統合クラウドシステム Flow in Cloud,” 信学技報　vol. 117　no. 153　CPSY2017-16，pp.1–5，2017.

[3] V.V. Skvortsov, M.I. Zvyagina, and A.A. Skitev, “Sharing resources in heterogeneous multitasking computer systems based on FPGA with the use of partial reconfiguration,” 2018 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus), pp.370–373, Jan. 2018.

[4] C. Beckhoff, D. Koch, and J. Torresen, “Go ahead: A partial reconfiguration framework,” 2012 IEEE 20th International Symposium on Field-Programmable Custom Computing Machines, pp.37–44, April 2012.

[5] 天野英晴，弘中和衛，山倉美穂，“マルチボード FPGA システム FiCSW における部分再構成の実装,” 信学技報 vol. 118 no. 165 CPSY2018-16，pp.65–69，2018.

[6] U. Farooq, I. Baig, and B.A. Alzahrani, “An efficient inter-fpga routing exploration environment for multi-fpga systems,” IEEE Access, vol.6, pp.56301–56310, 2018.

[7] Q. Tang, H. Mehrez, and M. Tuna, “Routing algorithm for multi-FPGA based systems using multi-point physical tracks,” 2013 International Symposium on Rapid System Prototyping (RSP), pp.2–8, Oct. 2013.

[8] 畔上佳太，武者千嵯，弘中和衛，A.B. Ahmed，天野英晴，“マルチ FPGA ボード間通信を行うスイッチの開発,” 信学技報 vol. 118 no. 215 RECONF2018-29，pp.55–59，2018.

[9] K. Musha, T. Kudoh, and H. Amano, “Deep learning on high performance FPGA switching boards: Flow-in-cloud,” Applied Reconfigurable Computing, pp.43–54, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Springer Verlag, Germany, Jan. 2018.

[10] 武者千嵯，A.B. Ahmed，工藤知宏，天野英晴，“FPGA スイッチボードへの深層学習アプリケーションの実装,” 信学技報 vol. 118 no. 340 RECONF2018-40，pp.33–38，2018.

[11] A. Aimar, H. Mostafa, E. Calabrese, A. Rios-Navarro, R. Tapiador-Morales, I. Lungu, M.B. Milde, F. Corradi, A. Linares-Barranco, S. Liu, and T. Delbruck, “Nullhop: A flexible convolutional neural network accelerator based on sparse representations of feature maps,” IEEE Transactions on Neural Networks and Learning Systems, pp.1–13, 2018.

[12] A. Prost-Boucle, A. Bourge, F. Ptrot, H. Alemdar, N. Caldwell, and V. Leroy, “Scalable high-performance architecture for convolutional ternary neural networks on fpga,” 2017 27th International Conference on Field Programmable Logic and Applications (FPL), pp.1–7, Sept. 2017.