Vol.2017-HPC-160 No.5
2017/7/26

OpenCLとVerilog HDLの混合記述によるFPGA間Ethernet接続

大畠 佑真[1,a)]　小林 諒平[2]　藤田 典久[2]　山口 佳樹[3]　朴 泰祐[2]

概要：近年，FPGA (Field Programmable Gate Array) が，ハードウェアおよびソフトフェアの進歩により，アクセラレータとして注目を集めてきている．ハードウェア面では，40 Gbps/100 Gbps に対応した通信ポートが搭載され，低レイテンシかつ広帯域な通信ができるようになっている．ソフトウェア面では，高級言語で FPGA のプログラムができる高位合成と呼ばれる技術が発達し，従来に比べ低コストで実装ができるようになっている．これらの背景の元，我々は，FPGA に通信機構に加えアプリケーションに特化した演算機能も組み込むというコンセプトとして，Accelerator in Switch を提唱している．本稿では，高位合成が可能な OpenCL 言語を用いて Accelerator in Switch の実現を目指す．通信規格には，スイッチが使える Ethernet を使用する．初期状態では，OpenCL から Ethernet 通信ができないため，本稿では，Ethernet 通信のための機能を実装し評価を行った．Ethernet 通信のためのコントローラを Verilog HDL で実装し，OpenCL から扱うことで，OpenCL からコントローラを通して Ethernet 通信を行うことができた．FPGA 間での OpenCL カーネルからの通信レイテンシは，860 ns であった．通信バンド幅は 4.9 GB/s で，利用した通信ポートの理論ピーク性能が 5 GB/s であるため，非常に効率の良い通信ができていると言える．回路使用率は，15 %と低いコストで実装できた．しかし，現時点では Ethernet 通信に必要なフロー制御や再送制御などの機能がコントローラに実装されてないため，簡単な通信しか行うことができない．これらの機能を実装していくことが今後の課題である．

1. はじめに

高い演算性能とメモリバンド幅を有する GPU(Graphics Processing Unit) を演算加速装置として搭載する CPU-GPU 構成のクラスタが今日の HPC システムのデファクトスタンダードになりつつある．筑波大学計算科学研究センターでは，演算加速装置間を低レイテンシの通信ネットワークで密に接続する TCA(Tightly Coupled Accelerators) と呼ばれるコンセプトを提唱しており，PEACH2(PCI Express Adaptive Communication Hub Ver.2)[1] という独自開発された通信機構を搭載した HA-PACS/TCA (Highly Accelerated Parallel Advanced System for Computational Sciences/TCA)[2] をこのコンセプトの実証システムとして運用している．

PEACH2 は FPGA (Field Programmable Gate Array) を用いて独自開発された通信機構である．FPGA とは任意の論理回路を電気的にプログラムすることができる集積回路であり，例えば PEACH2 では，低レイテンシの通信を実行する回路に加えて，オンザフライにオフロードされる CPU や GPU が不得手とする処理を実行する回路を FPGA 上に実装することによって，アプリケーション全体の性能を向上させる研究事例が報告されている [3], [4] ．我々は，このような通信機構内に演算機構を組み込むことで，高速なネットワークで接続された複数の FPGA が低レイテンシの並列処理を実行するコンセプトを Accelerator in Switch と呼んでおり，CPU-GPU クラスタ構成である現在の HPC システムの性能を更に向上させる鍵であると睨んでいる．更に，FPGA の演算性能と通信性能は半導体プロセス技術の進歩の恩恵によって，この数年において著しく向上している [5] ため， このコンセプトの実現に FPGA を用いることは合理的である．

この実現に向けてまず課題となるのが，「どのように FPGA 上に高速化したいアプリケーションを処理する回路を実装するか」である．一般的に，FPGA 上に実装する論理回路の設計では，VHDL や Verilog HDL などの HDL(Hardware Description Language) を用いた RTL(Register Transfer Level) での設計手法が採用される．しかしこの設計手法では，実装する回路の挙動を定義するために，どの素子が，どのタイミングで，どのように駆動す

[1] 筑波大学 システム情報工学研究科
[2] 筑波大学 計算科学研究センター
[3] 筑波大学 システム情報系
[a)] oobata@hpcs.cs.tsukuba.ac.jp

るかを設計者が明示しなければならないため，ソフトウェア開発と比較して基本的に長い開発期間を要するという欠点がある [6]．更に，このように回路の挙動を細粒度に定義しなければならないことからコードの移植性が低く，アプリケーションごとに回路を作り込まなければならない．そのため，HDL を用いて HPC アプリケーションに特化した回路を実装することは現実的ではない．

この課題の解決策として注目されているのが，高位合成と呼ばれる技術である．高位合成とは，ソフトウェア開発で用いられる C 言語，C++言語，OpenCL 言語などの高級言語から，VHDL や Verilog HDL などの HDL により記述された FPGA 上に実装可能な論理回路を生成する処理系を用いた設計方式であり [6]，ハードウェアエンジニアのみならずソフトウェアエンジニアも対象に近年において普及が進んでいる．

高位合成の利用を前提として Accelerator in Switch の実現に向けてもう一つ課題となるのが，「どのように FPGA の通信機能をユーザレベルで操作するか」である．このユーザレベルでの操作とは高級言語で記述されたアプリケーションのソースコード中に FPGA の外部 I/O コントローラにアクセスできるような記述が可能であることを意味する．例えば，Intel FPGA SDK for OpenCL では，BSP (Board Support Package) で提供されているペリフェラルコントローラは Channel 機能でアクセスすることができるが，提供されていない場合は設計者が所望のペリフェラルにアクセスできるようなハードウェアコンポーネントおよびライブラリを実装する必要がある [7]．また，40Gb/100Gb Ethernet のような高速な外部リンクを用いて複数の FPGA を接続し，高性能計算を試みた研究 [8] も報告されている．しかし，[8] では FPGA 間の通信プロトコルはポイント・ツー・ポイントであるため，接続数が数台程度ならば問題ないが，数百，数千とスケールすることは困難である．スケーラビリティを維持するため，我々は FPGA 間の通信は，ネットワークスイッチを経由しての Ethernet 通信を前提とする．

以上を踏まえて，本研究の目的は高位合成ツールの一つである Intel FPGA SDK for OpenCL を用いて，高速なネットワークで接続された複数の FPGA が協調して低レイテンシの並列処理を実現できる可能性を示すことである．OpenCL から FPGA 間の Ethernet 通信を行うためには，FPGA 上に Ethernet 通信プロトコルを解釈するコントローラを実装する必要がある．しかし，このスクラッチ実装は開発コストが比較的高いことから，我々は実装を簡単化するために FPGA ベンダーから提供される Ethernet IP コアを活用し，Verilog HDL を用いて OpenCL カーネルと Ethernet IP コアとの接続を記述する．本稿では，アプリケーションに直結する部分や FPGA 間通信に OpenCL を，OpenCL とハードウェアコンポーネント間の接続に Verilog HDL を併せて用いることで上記の目的の実現可能性を検討する．Ethernet IP コアの利用方法や，Verilog HDL による OpenCL カーネルと Ethernet IP との接続における実装方法を示し，OpenCL から FPGA 間おいて Ethernet プロトコルによる PingPong 通信を行い，通信レイテンシおよび通信バンド幅を評価する．

2. FPGA の開発手法

2.1 一般的手法

FPGA のプログラミング言語には，一般的にハードウェア記述言語が使用される．ハードウェア記述言語 (HDL: Hardware Description Language) には，Verilog HDL や VHDL などがある．ハードウェア記述言語で記述する動作レベルを RTL (Register Transfer Level) と呼び，RTL における記述は，1 クロック毎のレジスタ間のデータ移動について記述するものである．ハードウェア記述言語で記述した回路を FPGA ベンダが提供するツールを用いてコンパイルすることによって，FPGA 上で実際に動作する回路が構築される．目的の処理に応じて 1 クロック 1 ビットの粒度で回路を記述するため，非常に効率のよい回路が生成される．しかし，データの移動や変化のタイミングが少しでも間違えていると正しく動作しない，複数のレジスタ上で同時にデータの移動や変化が起こるためどこでエラーが起きているのか見つけ辛い等の要因から，ハードウェア記述言語を用いた実装は難易度が高いとされている．

ハードウェア記述言語から論理回路を生成することを論理合成という．この論理合成には，短くて数十分，回路の規模が大きくなると数時間かかる．そのため，ハードウェア記述言語でのデバッグの際，実機で動作を確認する前に，シミュレータを用いて回路のシミュレーションをすることが一般的である．シミュレーションでは，入力データに対する出力のデータの変化を 1 クロック単位で確認することができる．しかし，シミュレーションで正しい結果を得られても，様々な要因から実機で正しく動作しないことがある．その場合，繰り返し論理合成することになるため，デバッグに非常に時間がかかり，実装コストが高くなる．

2.2 高位合成

1 章で述べたように，高位合成はソフトウェア開発で用いられている高級言語からハードウェア記述言語を生成する技術である．比較的容易に扱える高級言語を使用して FPGA の実装ができ，RTL での記述をする必要がない．加えて，高位合成により生成される回路は，高級言語による記述が正しければ実機上で正しく動作することが保障されている．そのため，ハードウェア記述言語でのデバッグをする必要がない．これらの利点により，高位合成を用いることで大きく実装コストを抑えることができる．

一方で，いくつか欠点もある．一つ目は，ハードウェア

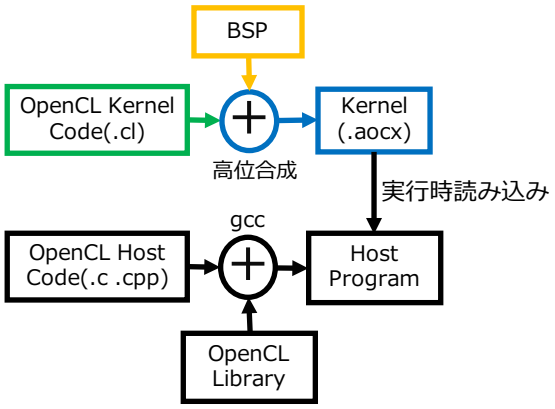

図 1: プログラム実行の流れ

記述言語による実装に比べて制限があることである．高位合成言語で記述できるハードウェアの範囲は，ハードウェア記述言語を用いて記述する場合と比べて狭い．二つ目は，性能が低くなることである．RTL での実装に比べ，回路の動作周波数が低くなったり，回路のリソース使用量が増えたりする場合がある．しかし，これらの欠点はある程度補えると分かっており [7], [9]，高位合成によって得られる利点の方が大きいと考えている．

3. Intel FPGA SDK for OpenCL

3.1 開発の流れ

Intel FPGA SDK for OpenCL は，Intel 社が提供している OpenCL を用いた高位合成ツールである．Intel FPGA SDK for OpenCL を用いて開発する際の流れを，図 1 に示す．OpenCL には，カーネルコードのコンパイル方法として，あらかじめコンパイルしておくオフラインコンパイルと，ホストプログラム実行時にコンパイルするオンラインコンパイルがあるが，高位合成には時間がかかるため，Intel FPGA SDK for OpenCL ではオフラインコンパイルのみ用いられる．

高位合成により生成した FPGA の回路データは，ホストプログラムの実行時に OpenCL の API を用いて FPGA に転送される．Intel FPGA SDK for OpenCL は高位言語のコンパイラだけでなく，ホスト OS で実行される FPGA 制御用のドライバ，ホストプログラムで用いるライブラリが提供されている．ホストプログラムは C 言語や C++言語など，一般的に OpenCL を利用できる言語で記述でき，ホストプログラムから FPGA の制御を行うために，OpenCL の API が提供されている．

3.2 BSP

BSP (Board Support Package) は，OpenCL が利用可能な FPGA ボード毎に，開発元から用意されているハードウェアコンポーネントであり，OpenCL コンパイラによってハードウェア記述言語を生成する際に用いられる．

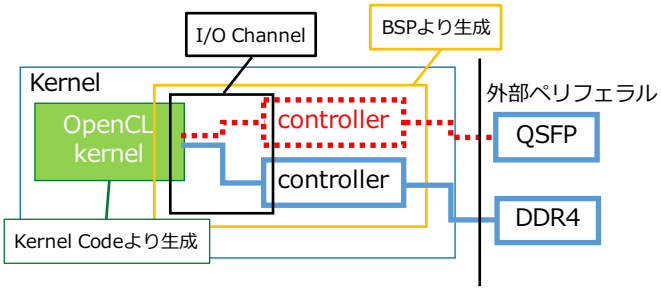

図 2: 高位合成によって生成されるカーネルの概略

```
write_channel_intel(write_channel_id , write_data);
read_data = read_channel_intel(read_channel_id);
```

図 3: Channel 関数

FPGA ボード毎に FPGA チップや，ボード上の外部ペリフェラル (DDR4 メモリ等) への接続方法や制御方法が異なる．BSP によりボード間の差異を吸収できるため，ユーザは同じ OpenCL コードからボードの違いによらず高位合成を用いて FPGA のプログラミングができる．

一般的に，BSP の実装は外部ペリフェラルの制御に関して，PCI Express や DRAM など OpenCL のプログラムを実行するために必要な最低限のインターフェイスについてしか提供されていない．そのため，本研究のように通信ポートを使用したい場合は，図 2 のようにペリフェラルコントローラを追加する必要があり，OpenCL からアクセスするためのインターフェイスも必要となる．

3.3 I/O Channel

Intel FPGA SDK for OpenCL による OpenCL 言語への独自拡張として，Channel 機能がある．これは，OpenCL カーネル間で直接通信を行う機能で，外部メモリを経由しないため高速に通信できる．Channel 機能のひとつとして I/O Channel があり，図 2 のように，OpenCL カーネル間ではなく OpenCL カーネルと外部ペリフェラルコントローラ間の通信機能を提供する．I/O Channel 用いることで，OpenCL カーネルから外部ペリフェラルの制御ができる．

I/O Channel は，Avalon ST (STreaming) インターフェイスでデータの送受信を行う．Avalon ST インターフェイスは，Intel FPGA における標準的なインターフェイスのひとつであり，point to point でデータをやり取りする．Avalon ST インターフェイスは，ペイロードを表す DATA 信号，送信データが有効であることを示す VALID 信号，データの受け取りが可能であることを示す READY 信号の 3 つの信号から成る．VALID 信号と READY 信号が共に 1 の時に，データを受け渡すという仕組みである．OpenCL コードからは，図 3 に示す記述で，データの送受信を行う．

4. Ethernet

4.1 Ethernet フレーム

本研究では， 40 GbE (Gigabit Ethernet) を使用しており，40 GbE のフレームは，図 4 のように構成される．以下にフレームの各要素について述べる．図 4 の Start，Preamble，SFD (Start Frame Delimiter) は，フレームの開始を意味するデータ列で，これらのデータ列はフレームの始まりを意味する．それぞれ，Start = 0xFB，Preamble = 0x555555555555，SFD = 0x5B と定められている．

Ethernet は，フレームの宛先や送信元を特定するために MAC (Media Access Control) アドレスを使用し，図 4 の Destination Address は宛先の MAC アドレス，Source Address は送信元デバイスの MAC アドレスを表す．Type-/Length は，フレームの Payload 部分のプロトコルまたはフレームの長さを表す．

図 4 の Payload に送信するデータを格納する．Ethernet の規格では，Payload の最大サイズは 1500 byte とされているが，それ以上のサイズを送信することも可能である．これはジャンボフレームと呼ばれており，ネットワーク機器によってサイズが異なるが，およそ 9 KiByte のサイズであれば送信できる．

図 4 の PAD (Padding) は，Payload が 46 byte 未満であるときに必要で，Payload と合わせて Ethernet 規格で定められている最小フレームサイズ 64Byte になるようにゼロを挿入する．

図 4 の CRC (Cyclic Redundancy Check) 32 には，誤り検出符号の一種である CRC-32 を用いて求められた Payload のチェックサムが格納され，受信デバイスでこの値を用いて受信データに誤りがないか検査する．

EFD (End Frame Delimiter) はフレームの終わりを示すデータ列で，規格では EFD = 0x5D と定められている．IPG (Inter Packet Gap) は，フレームを送信してない間，継続的に送信しなければならないデータ列で，IPG = 0x7E と定められている．また，複数のフレームを連続で送信する場合も，フレーム間に最低 12 byte 分の IPG 信号を挿入する必要がある．

4.2 Ethernet IP Core

IP (Intellectual Property) Core は，特定の機能についてまとめられた回路記述のことで，高級言語でのライブラリに相当する．本研究では，Ethernet フレームの送受信に Intel 社が提供している “Low Latency 40- and 100-Gbps Ethernet MAC and PHY MegaCore Function’’ [10] を利用する．Ethernet IP は，Ethernet 規格でデータを送受信する際に必要な機能を提供する．

Ethernet における物理層では，信号中にクロックを埋め込むシリアル通信を用いるため，FPGA 内でシリアル/パラレル変換が必要となるが，Ethernet IP はこの処理を自動で行う．また，4.1 節で述べたように，Ethernet のフレームは，図 4 のように構成されるが，図 4 の青色で示されているデータが自動で生成されフレームが構成される．受信時には，その部分が自動で取り除かれ，CRC チェックなども自動で行われる．そのため，ユーザはフレームの内，図 4 に示す赤色のデータ (以降ユーザデータと呼ぶ) を扱うだけで良い．

これらのサポートにより，IP Core を利用するユーザの回路は物理層に関する回路記述を行う必要がなく，Ethernet 規格でのデータの送受信が行える．ただし，Ethernet IP によるサポートはデータリンク層までであるため，それより上のレイヤーのプロトコル (TCP/IP，UDP/IP 等) に関してはサポートしていない．

図 5 に，Ethernet IP を用いてユーザデータを扱うための入出力信号を示す．tx_valid 信号，tx_data 信号，tx_ready 信号は，Avalon ST インターフェイスと同様の意味で，tx_valid 信号と tx_ready 信号が共に有効の時に tx_data 信号が，Ethernet IP に入力される．一度に Ethernet IP に入力できるデータサイズは 256 bit であり，ユーザデータがこれより大きい場合は，データを分割して入力する必要がある．tx_startofpacket 信号，tx_endofpacket 信号は，それぞれユーザデータの先頭と末尾を示すための信号である．tx 信号はユーザが Ethernet IP にデータを入力するための信号であるのに対して，rx 信号はユーザが Ethernet IP からデータを受け取るための信号である．rx 信号のそれぞれの信号の意味は tx 信号と同様であるが，rx 信号にだけあるものとして rx_error 信号があり，これは Ethernet IP に通信ポートから届いたデータのエラーコードを示す．

ユーザは，挿入するユーザデータのサイズをあらかじめ Ethernet IP に設定する必要がなく，Ethernet IP は，tx_startofpacket 信号が有効になってから tx_endofpacket 信号が有効でなくなるまでの間に，挿入されたデータの数から自動でユーザデータのサイズを判断する．tx_empty 信号は，tx_data 信号の一部をマスクする場合に用い，0〜tx_empty∗8 bit の範囲が無効となる．Ethernet IP は，ジャンボフレームに対応しており，ユーザデータが 1500 byte を超えた場合自動でジャンボフレームとして扱う．

5. 実装

5.1 通信ポートの調整

本研究で使用する FPGA ボードには，通信ポートとして QSFP+ (Quad Small Form-factor Pluggable Plus) が搭載されている．QSFP+は，動作周波数が約 10 GHz と非常に高く，動作周波数が高い場合，伝送エラーが発生しやすいという問題がある．

送信パルスは，図 6 左上の波形のような矩形波である．

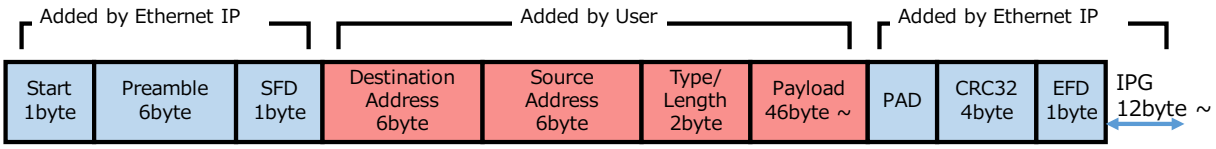

図 4: Ethernet フレームの内訳

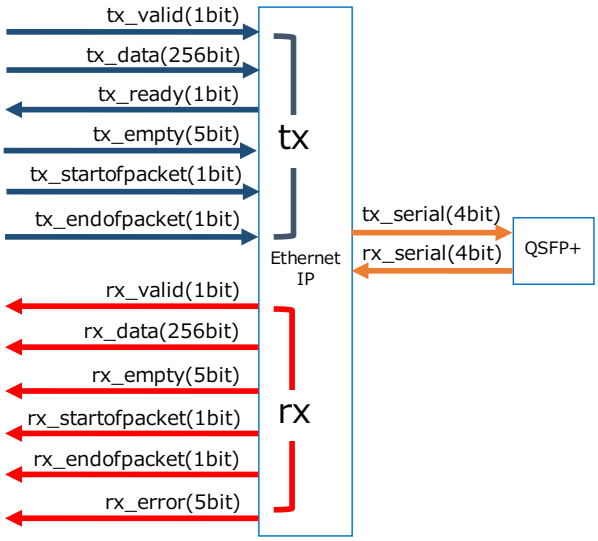

図 5: Ethernet IP の入出力信号

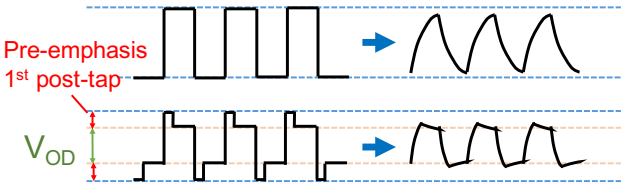

図 6: 左：設定した波形，右：実際の波形

しかし，信号の周波数が高い場合，信号の変化が間に合わず，図 6 右上の波形のようにパルスが乱れてしまうことがある．これにより，受信側で正しく信号を認識できず伝送エラーが発生してしまうことがある．具体的には，図 6 左下のように波形を変化させる．このように波形を変化させることで，実際の出力波形が図 6 右下のように矩形波に近づき，エラー率を減らすことができる．Intel 社は，GUI ベースでパラメータを変化させながらエラー率の確認ができる Transceiver Toolkit [11] というツールを提供しており，本研究ではこのツールを用いて信号を補正している．

5.2 実装方法

本節では，本研究において BSP 内で変更および追加したファイルについて述べる．図 7 は，それぞれのファイルの担当領域を示している．

(1) Ethernet IP の追加
(2) (1) を制御するためのモジュールの追加 (5.3 節で説明)
(3) board_spec.xml
　(2) と OpenCL カーネル間を接続する I/O Channel についてパラメータの追加
(4) freeze_wrapper.v
　I/O Channel に関する宣言の追加
(5) top.v
　(1) と (2) や I/O Channel，外部 I/O 間の接続に関する記述の追加
(6) flat.qsf
　(5) と QSFP ポート間の接続および QSFP ポートのパラメータに関する記述の追加
(7) opencl_bsp_ip.qsf
　(1),(2) のファイルパスの追加
(8) top_post.sdc
　タイミング制約の追加

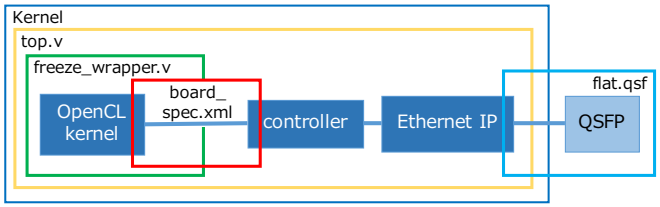

図 7: ファイル役割の概略

Intel FPGA SDK for OpenCL では，論理合成の際，インクリメンタル・コンパイルを行う．インクリメンタル・コンパイルとは，回路の内，変更しない部分をあらかじめ論理合成しておき，変更のあった部分だけを論理合成し部分的に回路を更新する手法である．Intel FPGA SDK for OpenCL を用いて高位合成する場合，BSP に含まれる回路部分は固定であり，OpenCL カーネルから生成される回路だけが更新されるためインクリメンタル・コンパイルの対象となる．

インクリメンタル・コンパイルを用いて正しく回路を生成するには，回路のどの部分を固定するのかについて設定を行わなければならない．BSP は開発元のベンダがその設定を行っており，BSP をそのまま利用する場合，ユーザはインクリメンタル・コンパイルに関する設定を行う必要はない．しかしながら，本研究では BSP に変更を加えているため，インクリメンタル・コンパイルを無効化し開発している．

5.3 Ethernet IP Controller

本節では，本研究で用いる Ethernet IP Controller について述べる．OpenCL からコントローラにアクセスす

るコードの例を図 8 に示す．4.1 節，4.2 節で述べたように Ethernet フレームには，送信元 MAC アドレス，宛先 MAC アドレスが必要で，ユーザデータに含める必要がある．これらを，OpenCL から SET_SOURCE_ADDR，SET_DEST_ADDR_COUNT Channel でコントローラに送る．SET_DEST_ADDR_COUNT Channel で送信データサイズをコントローラに送る (256 bit のデータを何個送るかを示す)．その後，SEND_DATA Channel からデータをコントローラに送信することで，Ethernet IP を通して，QSFP+ポートからデータが送信される．

40GbE IP とコントローラ間が 256bit 幅で接続されているため，OpenCL カーネルから 1 サイクルで送れるデータサイズも 256 bit としている．OpenCL 側から送信するデータサイズが，フレームのユーザデータの最大サイズを超えた場合，コントローラで自動的に複数のフレームに分割される．また，分割したフレーム毎に，送信元 MAC アドレスおよび宛先 MAC アドレスを自動で挿入する．

OpenCL カーネル–コントローラ間と，コントローラ–Ethernet IP 間で動作周波数が異なるため，送信データと受信データをコントローラ内でバッファリングしている．RECEIVE_DATA Channel から，受信用バッファに存在するデータを取り出すことができる．Ethernet IP には，スイッチから定期的に同期信号が送られるが，これはコントローラで破棄し OpenCL 側には渡らない．

受信したフレームの先頭 256 bit のうち 96 bit がアドレスで，本来であれば残りの 160 bit データとして使用できるが，実装が煩雑になるため現段階の実装では使用していない．加えて，送信元アドレス別にデータをバッファリングしたり，フロー制御，再送制御などをしたりする必要があるが，そこまで実装が進んでおらず，現時点では複数の宛先があったとしても，1 つのバッファを共有する実装となっている．

```
1  /*I/O の定義Channel*/
2  channel int SET_SOURCE_ADDR __attribute__((depth
       (0))) __attribute__((io("set_source_addr")));
3  channel int2 SET_DEST_ADDR_COUNT __attribute__((
       depth(0))) __attribute__((io("
       set_dest_addr_count")));
4  channel int8 SEND_DATA __attribute__((depth(0)))
       __attribute__((io("send_data")));
5
6  /*送信側*/
7  write_channel_intel(SET_SOURCE_ADDR, source_addr);
8  write_channel_intel(SET_DEST_ADDR_COUNT, (int2)(
       count, dest_addr));
9  for (int i = 0; i < count; i++){
10   write_channel_intel(SEND_DATA, send_data256[i]);
11 }
12
13 /*受信側*/
14 channel int8 RECEIVE_DATA __attribute__((depth(0))
       ) __attribute__((io("receive_data")));
15 for (int i = 0; i < count; i++){
16   receive_data256[i] = read_channel_intel(
         RECEIVE_DATA);
17 }
```

図 8: I/O Channel を用いて Controller にデータを送受信するための OpenCL コード

6. 評価

6.1 評価環境

評価には筑波大学 計算科学研究センターで運用中の PPX (Pre-PACS-X) システムを用いる．同センターでは，筑波大学で開発が続けられている PACS シリーズ・スーパーコンピュータの次世代機 PACS-X (PACS version 10) の開発を計画しており，PPX はそのプロトタイプである．PPX は Accelerator in Switch のコンセプトの実証実験と実アプリケーションへの適用を目的としている．

PPX の計算ノードの仕様を表 1 に示す．また，ハードウェア構成を図 10 に示す．1 ノードあたり，CPU として Intel Xeon E5-2660 v4 を 2 ソケット，FPGA ボードとして BittWare A10PL4 ボードを 1 枚搭載している．CPU と FPGA ボード間の接続は PCI Express であり，Gen.3 8 レーンを用いて接続されている．また，本稿では利用していないが，2 台の NVIDIA P100 GPU，Mellanox InfiniBand ConnectX-4 をそれぞれ搭載しており，CPU, GPU, FPGA の 3 種のデバイスを用いるヘテロジニアスなプログラミングが可能な環境である．ただし，InfiniBand ネットワークはホストのみを結合し FPGA は接続されておらず，また，40Gbit Ethernet ネットワークは FPGA のみを結合しホストは接続されていない．

A10PL4 ボードには 2 つの QSFP+ポートがあり，それぞれのポートが 40Gbit Ethernet で通信できる．PPX の FPGA 間を結合するネットワークは，スイッチ 1 台を用いたスター型トポロジを構成しており，FPGA ボードにある 2 つの QSFP+ポートは，どちらも同じスイッチに接続されている．

PPX で用いられている Ethernet スイッチは Mellanox MSN2100-CB2R [12] であり，このスイッチは最大で 100Gbit Ethernet までの通信規格をサポートしているが，FPGA ボードの通信性能にあわせて各ポートのリンク速度を 40Gbps に設定している．また，ジャンボフレームを利用するために MTU (Maximum Transmission Unit) をスイッチの上限値である 9216 バイトに設定し，FPGA の Ethernet IP Core においても同様に設定を行なっている．

6.2 評価方法

本研究では，性能評価として通信レイテンシおよび通信バンド幅の測定を行う．今回使用する FPGA ボードの

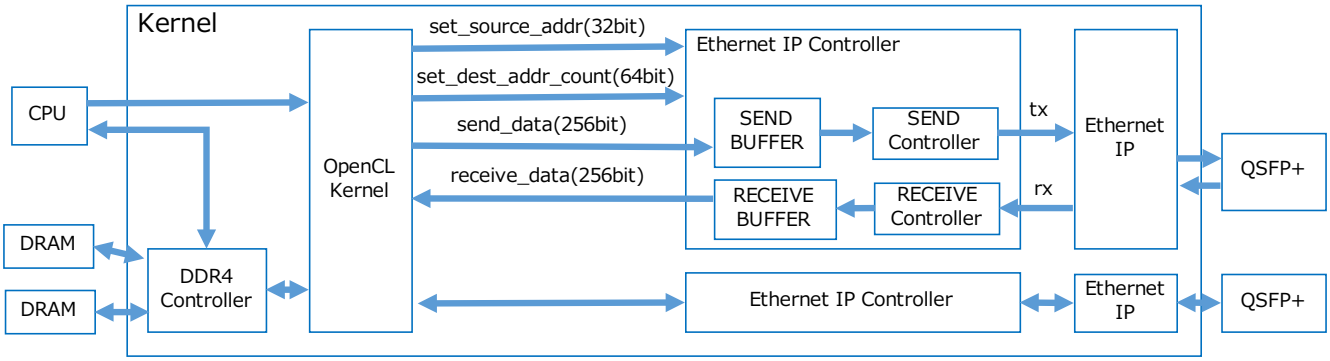

図 9: 実装回路の概略図

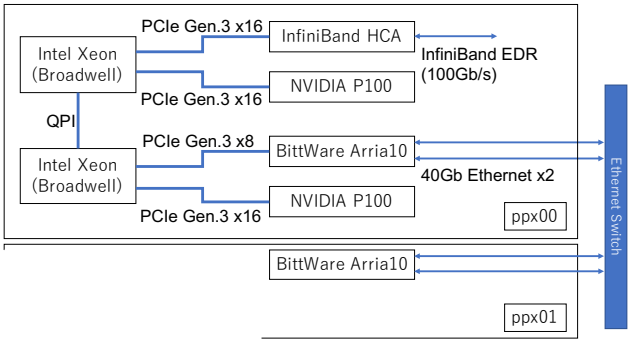

図 10: PPX 計算ノードにおける各コンポーネントの接続図

A10PL4 には QSFP+ポートが 2 つ搭載されており，レイテンシの測定は同一 FPGA ボード上の 2 つの QSFP+ポート間で測定を行う．レイテンシを測定する際に，スイッチを経由しての通信が不安定なため，同一 FPGA ボードの 2 つのポートの間で測定を行っている．通信バンド幅は，CPU 間で FPGA ボードの QSFP+ポートを通して PingPong を行うことで測定した．

6.3 通信レイテンシの評価

図 11 にスイッチの有無を含めた 3 パターンの通信レイテンシの測定結果を示す．“direct” は，スイッチを経由せず同一 FPGA 上の QSFP+ポート間を直接繋ぎ測定し，送信側の Ethernet IP にデータを送信してから，受信側 Ethernet IP からデータを受信するまでの時間を示す．同様に，“via-switch” はスイッチを経由して同一 FPGA 上の QSFP+ポート間を接続した場合の結果である．“via-switch-controller” は “via-switch” と同じ条件で，送信側の Ethernet IP Controller にデータを送信してから，受信側の Ethernet IP Controller からデータを受信するまでの時間を示す．

“direct” は Ethernet IP がデータの送受信の際の処理にかかる時間 (472 ns) , “via-switch”−“direct” (322 ns) は Ethernet スイッチによる遅延，“via-switch-controller”−“via-switch” (62 ns) は Ethernet IP Controller によるオーバーヘッドである．Ethernet IP Controller によるレイテンシが小さい理由は，再送制御やバッファ制御などが未実装であるためである．

表 1: 評価環境

PPX 計算ノード	
CPU	Intel Xeon E5-2660 v4 × 2
CPU Memory	DDR4 2400MHz 64GB (8GB × 8)
GPU	NVIDIA Tesla P100 (PCIe card version)
Infiniband	Mellanox ConnectX-4 EDR
Host OS	CentOS 7.3
Host Compiler	gcc 4.8.5
FPGA Compiler	Intel Quartus Prime Pro, Intel FPGA SDK for OpenCL Version 17.0.0 Build 289
Ethernet Switch	Mellanox MSN2100-CB2R
FPGA ボード (BittWare A10PL4)	
FPGA	Intel Arria10 GX 10AX115N3F40E2SG
LE (Logic Element)	1,150 K
ALMs (Adaptive Logic Module)	427,200
Registers	1,708,800
DSP (Digital Signal Processor)	1,518
M20K memory blocks	2,713
M20K memory size	53 Mbits
Exteral Memory	DDR4 2133MHz 8GB (4GB × 2)
通信ポート	QSFP+ x2 (40 Gbps x2)

6.4 通信バンド幅の評価

図 12 に通信バンド幅の測定結果を示す．最大で 4.97 GB/s のバンド幅が得られた．

本実験環境における理論ピーク性能は、次式より求められる．

$$5GB/s \times \frac{9216 - 57}{9216} = 4.97GB/s \qquad (1)$$

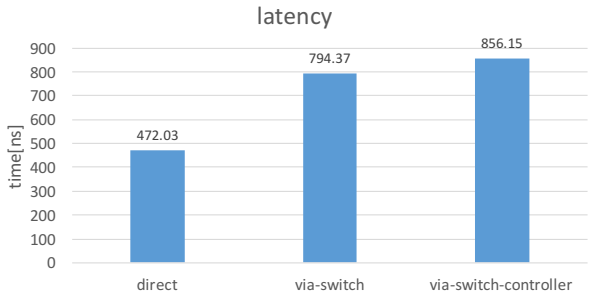

図 11: 通信レイテンシの測定結果

図 12: 通信バンド幅の測定結果

ここで、フレーム最大長の MTU (9216 byte) から，Start (1 byte)，Preamble (6 byte)，SFD (1 byte)，CRC32 (4 byte)，EFD (1 byte)，IPG (12 byte) を引いたデータがユーザデータの最大値で，加えてフレームの先頭のデータの一部 (32 byte) を破棄している．したがって，有効データ部分の理論ピーク性能は 4.97GB/s と算出される．今回の実験で得られた性能が理論ピーク性能とほぼ同じであり，高い実効効率が得られているとわかる．

6.5 FPGA 回路のリソース使用率

表 2 は，今回実装した回路のリソース使用率をモジュール毎に表している．表の ALM (Adaptive Logic Module) は，FPGA で回路を構成する最小の論理素子であり，FPGA では複数の ALM を用いて任意の論理回路を表現する．M20K memory block は，Intel 社の FPGA の中に実装されている大きさが 20K bit の内蔵 SRAM ブロックであり，バッファなどの用途で用いられる．M20K memory size は，前述した M20K メモリの中で実際に利用されているビットの量を示したものであり，全てのビットを利用しているとは限らないため，一般に memory block 使用率より memory size 使用率の方が低くなる．

表 2 の “Others” は，全体で使われている回路リソースから，Ethernet IP，Controller，OpenCL Kernel が使用している回路リソースを引いた値であり，変更を加えてない BSP によって生成される回路によって使われるリソースと考えられる．Ethernet IP や Controller を含め，全体のリソース使用率が 15 %ということは，残りの 85 %を演算などの処理に利用できるということであるため，Accelerator in Switch の実現に十分な演算リソースが残されているといえる．

7. まとめ

本稿では，OpenCL から QSFP+ポートを用いて，Ethernet 規格で FPGA 間通信ができるよう実装を行い評価をした．BSP に，Ethernet 通信を行うための Ethernet IP および，これを制御するための Ethernet IP Controller を追加した．OpenCL から I/O Channel を通して，この Controller とデータの送受信を行うことで，OpenCL から FPGA 間通信ができた．また，現時点では少ないリソースで，低レイテンシ，高効率な通信ができることを確認した．しかし，現時点での実装では Ethernet IP Controller に，フロー制御や再送制御，アドレス毎のバッファリングなどの機能がない．Ethernet 通信に加えて演算などの処理を，大規模なシステムで並列に行うにはこれらの機能が必ず必要となるため，これから，リソースやレイテンシを意識しつつ，これらの機能を追加していく必要がある．また，この通信機能に演算を組み合わせて実際に FPGA を Accelerator in Switch として利用できるか検討する必要がある．

謝辞 本研究の一部は，JST-CREST 研究領域「ポストペタスケール高性能計算に資するシステムソフトウェア技術の創出」研究課題「ポストペタスケール時代に向けた演算加速機構・通信機構統合環境の研究開発」，及び文部科学省研究予算「次世代計算技術開拓による学際計算科学連携拠点の創出」による．また，本研究の一部は，「Intel University Program」を通じてハードウェアおよびソフトウェアの提供を受けており，Intel の支援に謝意を表する．

参考文献

[1] Hanawa, T., Kodama, Y., Boku, T. and Sato, M.: Interconnection Network for Tightly Coupled Accelerators Architecture, *2013 IEEE 21st Annual Symposium on High-Performance Interconnects*, pp. 79–82 (online), DOI: 10.1109/HOTI.2013.15 (2013).

[2] 筑波大学：HA-PACS, 筑波大学 (online), available from ⟨https://www.ccs.tsukuba.ac.jp/research_project/ha-pacs/⟩ (accessed 2017-06-28).

[3] Kuhara, T., Tsuruta, C., Hanawa, T. and Amano, H.: Reduction calculator in an FPGA based switching Hub for high performance clusters, *2015 25th International Conference on Field Programmable Logic and Applications (FPL)*, pp. 1–4 (online), DOI: 10.1109/FPL.2015.7293985 (2015).

[4] Tsuruta, C., Miki, Y., Kuhara, T., Amano, H. and Umemura, M.: Off-Loading LET Generation to PEACH2: A Switching Hub for High Performance GPU Clusters, *SIGARCH Comput. Archit. News*, Vol. 43, No. 4, pp. 3–8 (online), DOI: 10.1145/2927964.2927966 (2016).

表 2: リソース使用率

	ALMs	Registers	M20K memory blocks	M20K memory size [bit]	動作周波数 [MHz]
Ethernet IP x2	25,371 (6.0 %)	46,264 (2.7 %)	26 (1.0 %)	434,176 (0.8 %)	312.5
Controller x2	834 (0.2 %)	1,997 (0.1 %)	56 (2.1 %)	1,050,624 (1.9 %)	312.5，316.67
OpenCL Kernel	9,707 (2.3 %)	23,664 (1.4 %)	89 (3.3 %)	1,250,816 (2.3 %)	316.67
Others	27,741 (6.5 %)	52,852 (3.1 %)	228 (8.4 %)	1,955,688 (3.5 %)	
Total	63,653 (15.0 %)	124,777 (7.3 %)	399 (14.8 %)	4,691,304 (8.5 %)	

[5] 天野英晴， 他：FPGA の原理と構成，オーム社 (2016).
[6] 高前田伸也：高位合成ツールを用いたハードウェア設計 — FPGA を用いた専用コンピュータの開発がお手軽にできる時代が来た! —，電子情報通信学会誌， Vol. 100, No. 2, pp. 103–108 (2017).
[7] 藤田典久，大畠佑真，小林諒平，山口佳樹，朴 泰祐：OpenCL と Verilog HDL の混合記述による FPGA プログラミング，情報処理学会研究報告 (2017).
[8] Antoniette, M., Tomohiro, U., Daichi, T., Kentaro, S. and Satoru, Y.: High-Performance Scalable Stream Computing with Multiple FPGAs, ハイパフォーマンスコンピューティングと計算科学シンポジウム論文集, Vol. 2017, pp. 16–16 (2017).
[9] 大畠佑真，藤田典久，小林諒平，山口佳樹，朴 泰祐：高位合成による FPGA の高性能計算へ適用，ハイパフォーマンスコンピューティングと計算科学シンポジウム論文集，Vol. 2017, pp. 15–15 (2017).
[10] Intel: Ethernet IP, Intel (online), available from ⟨https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/ug/ug_ll_40_100gbe.pdf⟩ (accessed 2017-06-28).
[11] Intel: Transceiver Toolkit, Intel (online), available from ⟨https://www.altera.co.jp/products/design-software/fpga-design/quartus-prime/features/swf-transceiver-toolkit.html⟩ (accessed 2017-06-28).
[12] Mellanox: MSN2100-CB2R, Mellanox (online), available from ⟨http://www.mellanox.com/related-docs/prod_eth_switches/PB_SN2100.pdf⟩ (accessed 2017-06-28).