東京工業大学工学部

学士論文

USB3.0 接続の高速でポータブルな FPGA アクセラレータ

指導教員　吉瀬 謙二　准教授

平成 27 年 2 月

提出者

学科　情報工学科

学籍番号　11_03601

氏名　臼井 琢真

指導教員　印	

学科長　認定印	

平成 26 年度　学士論文内容梗概

USB3.0 接続の高速でポータブルな FPGA アクセラレータ

指導教員　吉瀬 謙二　准教授
情報工学科
11_03601　臼井 琢真

汎用プロセッサが不得意とする処理を高速化するために，FPGA にアクセラレータを実装してアプリケーションを高速に実行する研究が盛んに行われている．FPGA を用いたアクセラレータは，アプリケーションに特化した演算パイプラインとデータ供給機構を実現する回路を FPGA 上に実装することにより，CPU や GPU と比較して高い演算能力や電力効率を達成できる．しかし，FPGA アクセラレータを使用するためには，ホスト PC 間とデータの送受信を行うためのインターフェースを選択しなければならない．PCI Express などの高速 IO を選択した場合，FPGA アクセラレータは高い転送速度の恩恵を受けて高い性能を示すが，ラップトップ PC などではこの手法は適用できず，ポータビリティに欠ける．USB2.0 のような外付けインターフェースを選択した場合，様々な環境で使用できるためポータビリティが高いが，通信速度がボトルネックとなるため性能は低くなる．このため，従来の FPGA アクセラレータに高い性能と高いポータビリティを両立させることが困難であった．

本論文では，USB3.0 をインターフェースに選択することで高い性能と高いポータビリティを両立した FPGA アクセラレータを提案する．USB3.0 は理論値最大 5Gbps と非常に高い転送速度を持ち，提案する FPGA アクセラレータはその恩恵を受けて高い性能を持つことが期待される．更に USB3.0 は外付けインターフェースであるため，デスクトップ PC だけでなくラップトップ PC といった様々な環境で使用できる．このため，提案する FPGA アクセラレータは高い性能と高いポータビリティを両立する．

提案手法のケーススタディとして，高速化するアプリケーションにソーティングを選択し，高速にソーティングできる FPGA アクセラレータの設計と実装を行った．提案する FPGA アクセラレータが高い性能と高いポータビリティを両立していることを示すために，デスクトップ PC やラップトップ PC といった様々な計算機環境でソーティングの性能を CPU と比較し評価した．評価の結果から，ホスト PC と USB3.0 にて接続した場合，提案する FPGA アクセラレータが高い性能を持つことを明らかにした．

目次

第 1 章

緒論

1.1 研究の目的

演算性能と電力効率の向上を達成するために，汎用 CPU が不得手とする処理を GPU や FPGA などで高速化する研究が注目を集めている．特に，FPGA を用いたアクセラレータは，アプリケーションに特化した演算パイプラインとデータ供給機構を実現する回路を FPGA 上に実装することにより，CPU や GPU と比較して高い演算性能や電力効率を達成できる．

FPGA アクセラレータを設計するときには，ホスト PC と FPGA アクセラレータ間におけるデータの送受信のためのインターフェースを考慮しなければならない．データの送受信のためのインターフェースとしては，UART, Ethernet, PCI Express, USB などが存在する．高速な IO である PCI Express を選択した場合，FPGA アクセラレータは高い転送速度の恩恵を受け性能向上を達成可能であるが，拡張スロットの存在しないラップトップ PC などにはこの手法を適用できず，ポータビリティは低い．USB などの外付けインターフェースを選択した場合，多くの環境において使用可能なためポータビリティが高いが，通信速度がボトルネックとなるため性能は低くなる．このため，従来の FPGA アクセラレータに高い性能と高いポータビリティを両立させることは困難であった．

本論文では，ホスト PC と FPGA アクセラレータ間におけるデータ送受信のためのインターフェースに USB3.0 を選択する．USB3.0 は外付けのインターフェースであり，

- 転送速度が非常に高く (理論値 5Gbps)，ボトルネックとなりにくい
- ホットスワップに対応しているため接続が容易
- 最大 900mA のバスパワーにより，省電力な機器であれば USB ケーブルを 1 本繋ぐ

だけで動作する

- モダンな計算機の大半は USB3.0 接続に対応している

という利点を有している [1]．これらの利点は CPU と FPGA アクセラレータ間におけるデータの送受信のためのインターフェースとして非常に好ましく，FPGA アクセラレータが高い性能と高いポータビリティを両立することが期待される．以上より，本論文では USB3.0 接続をサポートする FPGA アクセラレータを提案する．

ケーススタディとして，本論文では，FPGA で高速化させるアプリケーションにソーティングを選択する．ソーティングは多くのアプリケーションにとって非常に重要な計算カーネルであり，データベース，画像処理，データ圧縮などで FPGA による高速化手法が研究されてきた [2][3][4].

本論文では，USB3.0 接続をサポートする FPGA アクセラレータによるソーティングの実効性能を評価する．また本論文では，提案する FPGA アクセラレータのポータビリティを示すために，ソーティングの実効性能をデスクトップ PC やラップトップ PC と比較している．

1.2 本論文の構成

本論文の構成は以下の通りである．第 2 章で関連研究や USB3.0 について紹介する．第 3 章でソーティングを高速に実行するポータブルな FPGA アクセラレータを示し，その設計，実装，検証・評価方法に関して述べる．第 4 章で提案する FPGA アクセラレータの評価を行う．最後に，第 5 章で本論文をまとめる．

第 2 章

研究の背景

本章ではまず，FPGA について簡単に述べ，次に関連研究について述べる．最後に，提案する高速でポータブルな FPGA アクセラレータの要となる USB3.0 について述べる．

2.1 FPGA

FPGA(Field Programming Gate Array) とは，ユーザーが必要に応じて機能を書き換えることが可能な LSI である．そのため，ユーザー自身が任意の論理機能を自由に設計できる．

FPGA に所要の機能を実装する方法には，回路図を作成する方法や状態遷移を定義する方法など様々なものがあるが，一般的に，ユーザーは Verilog HDL や VHDL といったハードウェア記述言語を用いて所要の機能を記述し，FPGA ベンダが提供するツールを用いて FPGA 上に実装する．本論文では設計したソーティングロジックをこの手法を用いて FPGA に実装している．

2.2 FPGA アクセラレータ

プロセスの微細化などにより，アクセラレータとして用いた際に達成可能な性能はソフトウェアと比較して十分なものとなっている．そのため，ASIC を開発する場合と比較して低コスト・短期間で FPGA にアクセラレータを実装することで CPU や GPU と比較して高い性能・電力効率を達成できる．以上の理由から，FPGA を用いたアクセラレータの研究が現在盛んに行われている．以下，FPGA アクセラレータを用いてアプリケーションを高速化した例を挙げる．

2.2.1 PageRank

米 Microsoft は，FPGA を用いて PageRank 処理を高速化する技術「Catapult」を開発した [5]．サーバ 1 台につき一つの FPGA を接続し，これらを合計 1632 台束ねてクラスターを構成している．FPGA 上でパイプライン処理をさせることで，PageRank 処理のスループットを 2 倍に増加させた．現在は試験運用中だが，今年初頭に Bing のシステムに導入する予定となっている．

2.2.2 ステンシル計算

[6] では，流体計算や気象計算に利用される計算カーネルであるステンシル計算を高速に実行するアクセラレータを提案している．この FPGA アクセラレータは，ALTERA Stratix Ⅲ を搭載した，Terasic DE3 評価ボードを複数枚使用して実装されており，動作周波数が 3.2GHz の Intel Core i7-3930K に対して，最大 13.7 倍の高速化を達成している．[7] でもステンシル計算を高速に実行するアクセラレータについて報告している．ハイエンド FPGA を利用する [6] とは違い，小容量の FPGA である Xilinx Spartan-6 を多数使用することで FPGA アクセラレータを実装しており，NVIDIA GTX280 グラフィックカードと比較して約 3.8 倍の電力効率を達成している．

2.2.3 ソーティング

FPGA 上に実装することができるソーティングアルゴリズムはさまざまあるが，ロジック効率および実行効率が良いのは，マージソートツリーである [8]．マージソートツリーとは，2 入力 1 出力で比較を行うソートセルで構成され，このソートセルを二分木状に接続してマージソートを実行するデータパスである．

[8] では，FIFO ベースのマージソートとマージソートツリーを組み合わせた FPGA によるソート手法について提案している．これは，まず最初に FIFO ベースのマージソートを用い，その後マージソートツリーを複数回使用してソートを行っている．本論文にて設計を示すソーティングロジックはこの手法を採用している．

[2] では，FPGA 上のソーティングネットワークについて詳しく議論されている．ソーティングネットワークでは，比較器の接続方法を調整することで，Even-Odd Merge, Bitonic Sort, Bubble Sort, Insertion Sort などを実現することができる．しかしこの手法は，いずれのソーティングアルゴリズムにおいても，実装されている入力幅分のデータセッ

トしかソートできず，より大きなデータセットをソートしたい場合には更に CPU でマージを実行する必要があり，FPGA のみでソートを完結できないという欠点を有している．

2.3　USB3.0

ホスト機器に様々な周辺機器を接続するために策定された汎用外付けインターフェースである USB(Universal SerialBus) の比較的新しい規格である [1]．1996 年に USB1.0 が登場したが，その速度は 12Mbps と低速であった．しかし，2000 年に最大転送速度が 480Mbps(理論値) の USB2.0 が登場し，多くのデバイスのインターフェースに採用されるようになった．

USB は以下の特徴を持っている．

- ホットスワップに対応
- バスパワーでの電源供給 (USB2.0 で最大 500mA)
- 根をホスト，節をハブ，葉をデバイスとする木構造を構成
- 必ずホスト側が通信を発行

このような特徴のために，大容量記憶デバイス，画像転送装置で一般的に用いられている．バスパワーのため，省電力なデバイスであれば別途電源を繋げることなく使うことができる．この特徴により，従来一般に用いられたシリアルポートやパラレルポートがこれに取って代わられた．

2008 年，更に高速な USB3.0 が仕様策定された．正式名称は「SuperSpeed USB」である．これは上記に加えて，

- 最大転送速度 5Gbps(理論値)
- 900mA に拡張されたバスパワー

といった特徴を持っている．この高速な転送速度により，従来の USB2.0 ではその通信速度が大きなボトルネックとなっていた用途にも使われるようになった．主な使用用途として，SSD などの高速な記憶装置のインターフェース，高画質な動画配信のためのインターフェースが挙げられる．バスパワーも従来の USB2.0 の 500mA から拡張されており，電源要らずで手軽に利用できる機器の幅も広がった．現在，Intel の最新チップセットの全て，AMD の最新チップセットの殆どが標準で対応している．

本論文では，ホスト PC と FPGA アクセラレータ間のデータ送受信のためのインターフェースに USB3.0 を採用することで，外付けインターフェースを選択した際に問題と

なっていた通信速度のボトルネックの解消を目指す.

第 3 章

高速でポータブルな FPGA アクセラレータ

図 3.1 に提案する FPGA アクセラレータの使用状況を示す．ボード上の FPGA にアクセラレータを実装し，これを USB3.0 にて接続する．ホスト PC から USB3.0 を介してソーティングされるデータ列を受信し，FPGA 内でソーティングを実行し，ソートされたデー

図 3.1: ノート PC に接続された FPGA アクセラレータ

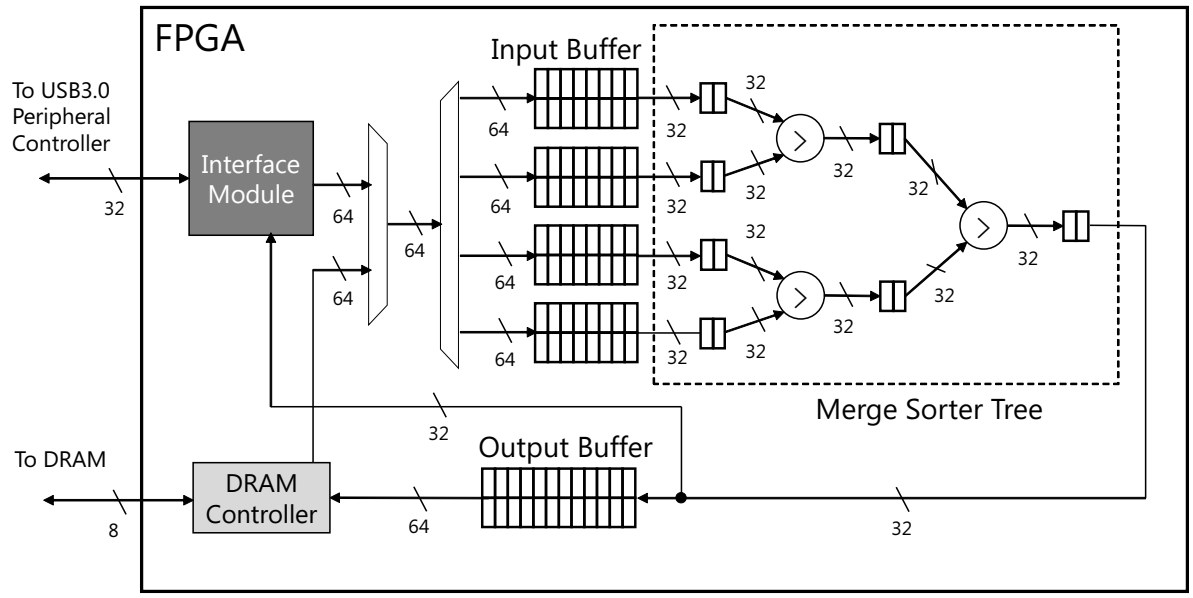

図 3.2: FPGA に実装するソート回路のデータパス

タ列をホスト PC に送り返す．これによってデスクトップ PC だけでなく，ノート PC などといった様々な環境での使用を可能とする．本章では，提案する FPGA アクセラレータの性能を高いものとするための，ソーティングを高速に実行するハードウェアの設計について述べる．次に，その実装を述べ，最後に検証・評価方法を述べる．

3.1 ソーティングロジック

図 3.2 に，FPGA 上に実現されるソーティングロジックのデータパスを示す．

図 3.2 中の破線で囲まれた部分が Merge Sorter Tree であり，このロジックでソーティングが実行される．Merge Sorter Tree 中の四角は FIFO を表し，丸はソートセルを表している．ソートセルは 2 つの入力データを比較し，それらの入力データの比較結果にしたがってどちらかのデータを選択して出力する組み合わせ回路である．Interface Module はホスト PC から送信されるデータ列の受信，FPGA 内でソートされたデータ列の送信を担当する．今回，受信したデータを 2 要素ずつソートする 2-Element Sorter を内蔵するものと内蔵しないものの 2 種類に分けて設計した．ホスト PC から送信されるデータ列を受信する際，Interface Module は，2-Element Sorter が内蔵されている場合は入力データに対して 2 要素ずつソートし，そうでない場合は到着順に 2 要素ずつ並べ，Input Buffer に格納する．Input Buffer は入力幅が 64bit(2 要素)，出力幅が 32bit(1 要素) の非対称 FIFO

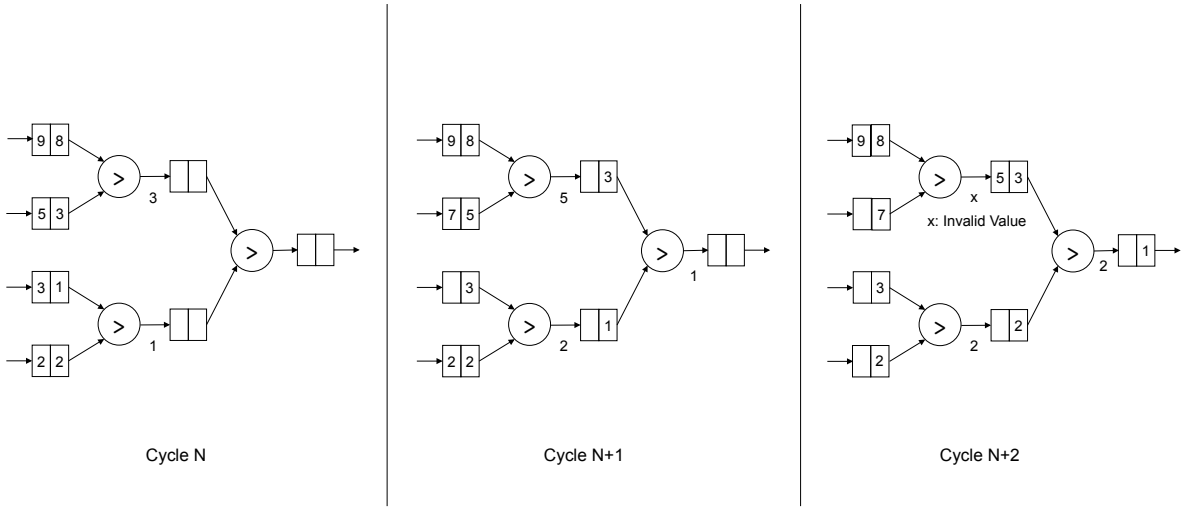

図 3.3: Merge Sorter Tree におけるソーティングプロセス

である．Merge Sorter Tree でソートされたデータは Output Buffer で一時的にバッファリングされた後，DRAM にライトされる．Output Buffer は入力幅が 32bit(1 要素)，出力幅が 64bit(2 要素) の非対称 FIFO である．

図 3.3 に Merge Sorter Tree でどのようにソーティングが実行されるのかを示す．我々は，Merge Sorter Tree の葉の数，つまり Merge Sorter Tree に入力されるデータ列のレーン数を way 数と定義している．そのため，**図** 3.3 の Merge Sorter Tree は 4-way Merge Sorter Tree となる．それでは，4-way Merge Sorter Tree でソーティングがどのように実行されるのかを説明していく．

まず，Cycle N で要素″8″ と″3″, 要素″1″ と″2″ の比較が実行されている．ここで，最左端の FIFO にはソート済みのデータ列が格納されていることに留意してもらいたい．ソートセルは組み合わせ回路であり，出力先の FIFO が full でない場合，比較結果にしたがって小さい方の要素を出力する．Cycle N+1 では要素″3″ と″1″，要素″8″ と″5″，要素″3″ と″2″ の比較が実行される．どのソートセルの出力先の FIFO も full ではないので，各ソートセルはそれぞれの比較結果にしたがって小さい方の要素を出力する．Cycle N+2 においても各ソートセルで比較が実行されるが，要素″8″ と″7″ の比較を実行しているソートセルの出力先の FIFO が full であるので，このソートセルは比較結果による要素を出力せず，無効な値″x″ を出力する．そして，出力先の FIFO が full でなくなったら，要素″7″ を出力する．このように各サイクル毎にソートセルで要素の比較を実行し，比較結果にしたがって出力される要素を FIFO に格納することで，Merge Sorter Tree の root からソー

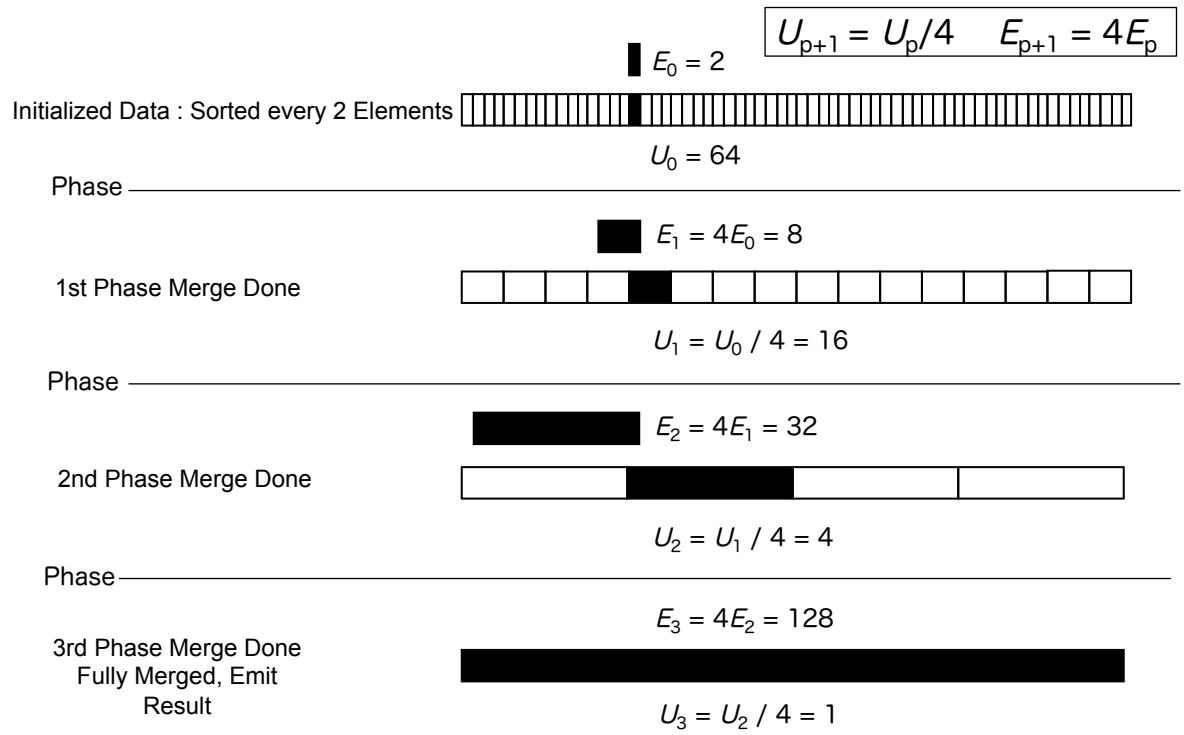

図 3.4: 入力データ列がソートされていく様子

トされたデータ列が出力される．

4-way Merge Sorter Tree では，ソート済みのデータ列 4 個を 1 つのソートされたデータ列にマージすることができる．しかし，データ列が 4 個より多いときは必要な回数分マージを繰り返さなければならない．このプロセスを実行するために，Merge Sorter Tree から出力されるソートされたデータ列は Output Buffer で一時的にバッファリングされた後，DRAM にライトされる．Output Buffer でバッファリングするのは，DRAM に書き込むデータの粒度を上げるためである．

3.2 ソーティングプロセス

図 3.4 は，ホスト PC から送信されてきた未ソートのデータ列 (128 要素) が時間の経過とともにどのようにソートされているかを示している．

我々は，Merge Sorter Tree に入力されるデータ列のうちソート済みであるデータ列を Unit と定義し，Unit がマージされるステップを Phase と定義し，Phase が実行された回数を P と定義する．例えば，**図** 3.4 中の U_0 は Phase が一度も実行されていない ($P = 0$)

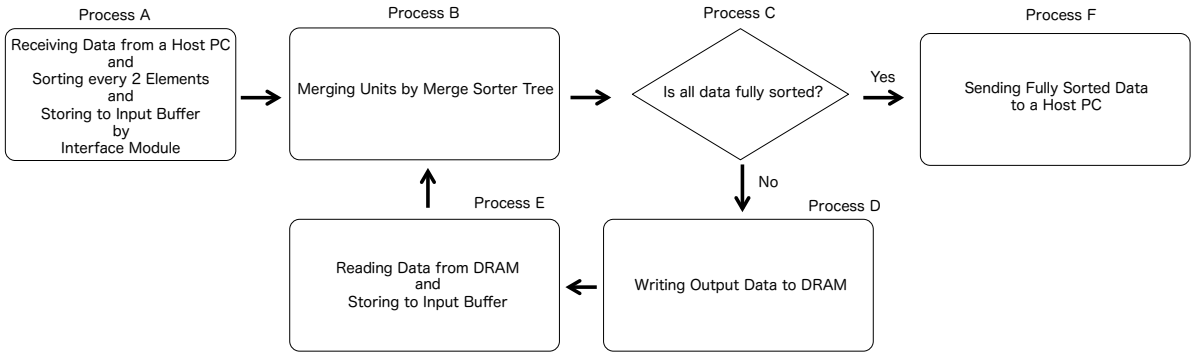

図 3.5: ソーティングの手順

ときの Unit 数，E_0 は U_0 内の要素数を意味しており，U_1 は Phase を 1 回した後 ($P = 1$) の Unit 数，E_1 は U_1 内の要素数を意味している．

図 3.4 で実行されているソーティングの手順を，2-Element Sorter が Interface Module に内蔵されている場合を例に挙げて**図** 3.5 に示す．まず，**図** 3.5 中の Process A でホスト PC から送信されてきた未ソートのデータ列は**図** 3.2 の Interface Module によって 2 要素ずつソートされ，Input Buffer に格納される．この 2 要素ごとにソートされたデータ列の集合が**図** 3.4 中に示されている”Initilized Data” である．このとき，Phase が一度も実行されていないので，$U_0 = 64$, $E_0 = 2$ である．

次に，**図** 3.5 中の Process B で $E_0 = 2$ のユニットが Merge Sorter Tree でマージされる．

図 3.5 中の Process C でデータ列全体のソーティングが完了したかどうか判断する．データ列全体のソーティングが完了していたら，ソートされたデータ列をホスト PC に送信し (Process F)，そうでなければ，Merge Sorter Tree から出力されるデータ列を DRAM に格納する (Process D)．4-way Merge Sorter Tree では，4 つの Unit を 1 つの Unit にマージすることができる．すなわち**図** 3.4 の場合，$U_1 = 16$, $E_1 = 8$ となったデータ列が MergeSorter Tree から出力される．**図** 3.4 から分かるように，データ列全体に対するソーティングは完了していない．そのため，Process D に移行し，$U_1 = 16$, $E_1 = 8$ となったデータ列を DRAM に格納する．

図 3.5 の Process E で，Process D で格納したデータ列をソートするために，DRAM からそれらのデータを読み出して Input Buffer に格納する．

この ProcessB∼E を繰り返すことで，データ列全体のソーティングが完了する ($U_3 = 1$, $E_3 = 128$)．実装では，FPGA アクセラレータの性能を向上させるために，この

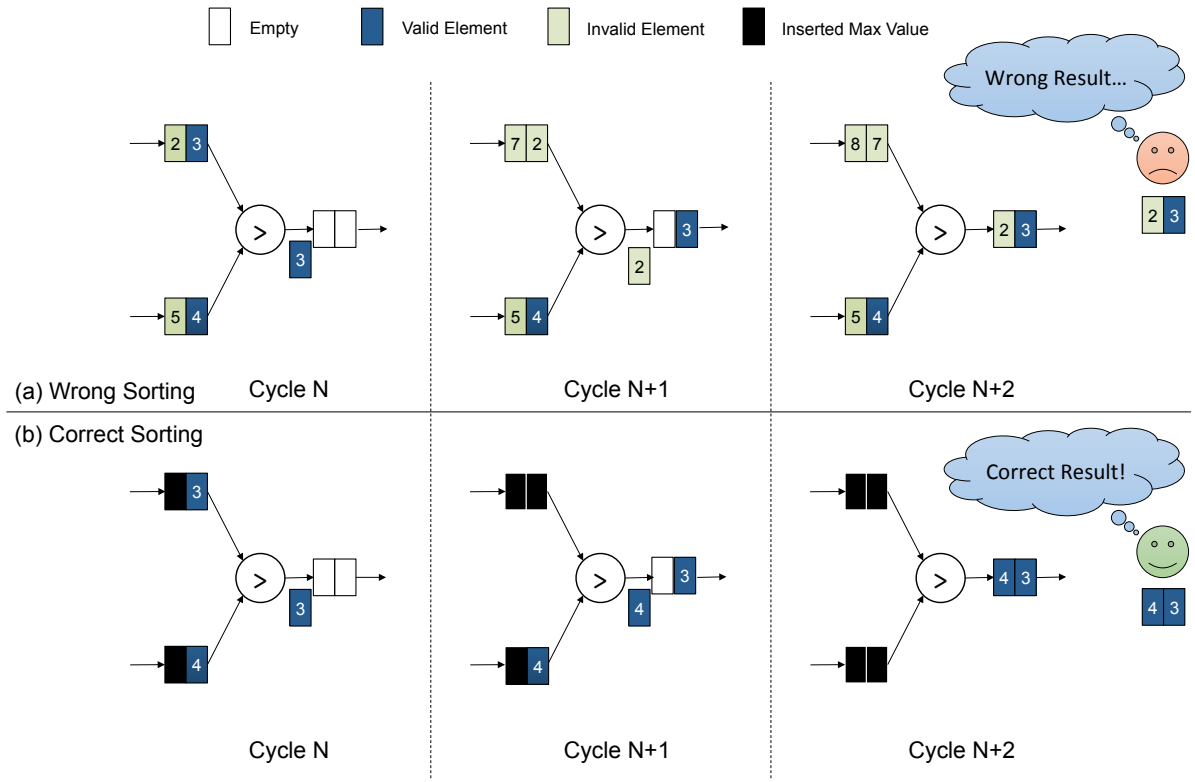

図 3.6: 誤ったソーティングと正しいソーティング

ProcessB∼E の処理をオーバーラップさせている．そして，ソーティングが完了したデータ列は Interface Module を通ってホスト PC に送信される．

この議論から，U_P, E_P のデータ列と K-way Merge Sorter Tree は以下の式で関係づけることができる．

$$U_{P+1} = \frac{U_P}{K} \qquad E_{P+1} = E_P K \tag{3.1}$$

ただし，この式は $U_0 > K$ のときに成り立つ．そして，データ列全体をソートするために必要な Phase の数は $\log_K U_0$ である．

3.2.1 ソーティングロジックの制御

この節ではソーティングを実行する計算ロジックの制御について述べる．このロジックを正しく動作させるためには，ユニット内の要素が正しくソートされていることを保証しなければならない．

FPGA に実装されるソーティングロジックは $U_0 > K$ のとき，ユニットのマージを必要

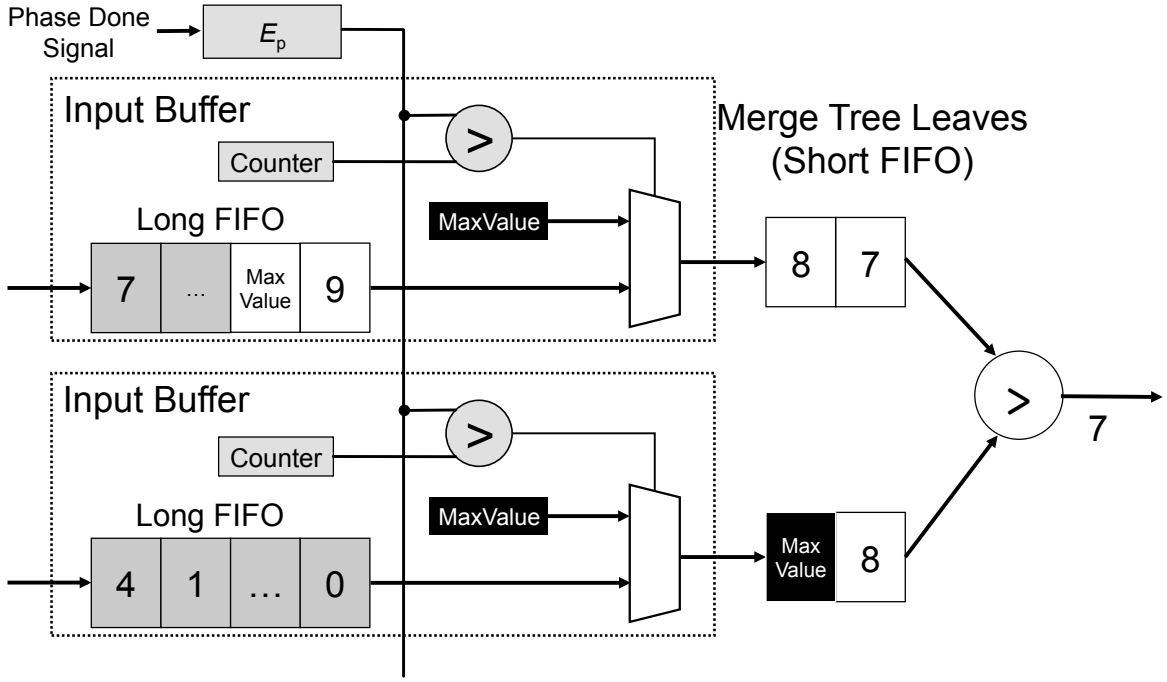

図 3.7: Input Buffer の内部

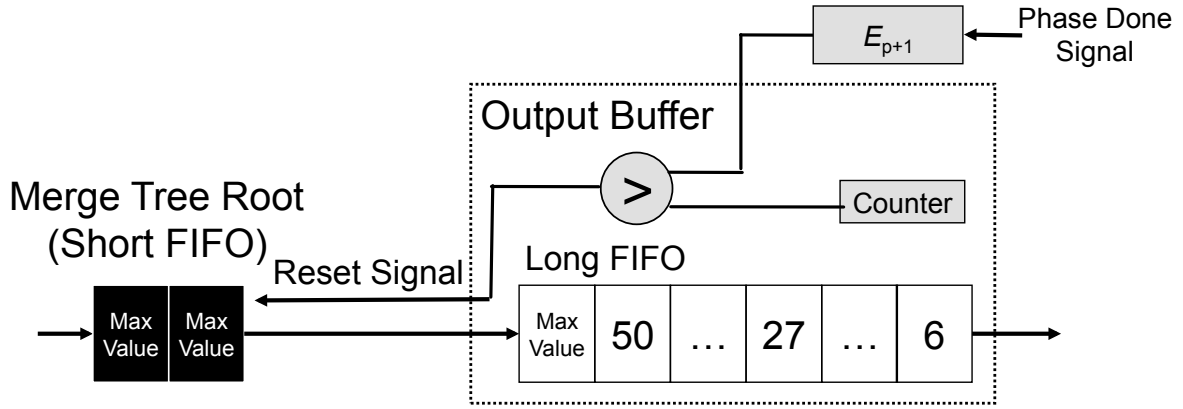

図 3.8: Output Buffer の内部

な回数分実行する．ここで K は way 数を示している．Merge Sorter Tree によるユニットのマージの実行において，Input Buffer から Merge Sort Tree に出力する際に，各 Unit を区切る必要がある．それをしない場合，各 Unit に有効な値でない要素が混入してしまい，ソーティングが正しく実行されない．

図 3.6(a) にその例を示す．**図** 3.6 は Unit 内の要素が 1 個のである Unit のマージ (“3” と “4” のマージ) が正しく実行されない場合 (a)，正しく実行される場合 (b) を示している．Merge Sorter Tree で 1 つのユニットにマージされるべき要素，つまりこの場合では “3” と “4” を”Valid”，それ以外の要素を”Invalid” と定義する．Cycle N+1 において，ソートセルから出力されるのは”Valid” な要素である”4” であるべきだが，(a) では”2” の”Invalid” な要素を出力してしまい，正しいソーティングが実行されない．

これを解決するために，“Valid” な要素の後ろには要素のビット幅に応じた最大値を挿入する (**図** 3.6(b))．そうすることで，Cycle N+1 においてソートセルから”4” が出力され，正しいソーティングが実行される．

図 3.6(b) を実現するため，Input Buffer に各 Unit を区切る (“Valid” と”Invalid” の要素を区別する) ための最大値を生成する回路を接続した (**図** 3.7)．各 Input Buffer は Input Buffer から出力した要素数を記憶するカウンタを保持している．カウンタの値が Phase N における要素数 E_N を上回った場合，要素のビット幅に応じた最大値を出力し，後続の Unit は Input Buffer 内に保持され続ける．ここで，Phase N における要素数 E_N とは，Phase N における各ユニット内の要素数を意味している．

そして，Merge Sorter Tree から出力され Output Buffer に格納される要素数を Output Buffer が保持するカウンタ (**図** 3.8) で記憶し，カウンタの値が E_{N+1} を上回ったら Merge Sorter Tree 内の全ての FIFO と Input Buffer のカウンタ，OutPut Buffer のカウンタをリセットし，Input Buffer に格納されている他のユニットのマージに備える．カウンタの値が E_{N+1} を上回るということは，Phase $N+1$ におけるユニット 1 個分に含まれる要素のソーティングが完了したことを意味している．

Phase N におけるソートが終了 (つまり Phase N における全てのユニットがマージされたら) したら，Input Buffer の E_N が格納されているレジスタ，Output Buffer の E_{N+1} が格納されているレジスタを $log_2 K$bit だけ左シフトし，Phase $N+1$ のソートに備える．

3.3 実装

この節では FPGA アクセラレータの実装について述べる．

FPGA アクセラレータを実装するために，我々は TKDN_ART7_BRD-2[9] を導入した (**図** 3.9)．このボードは，FPGA に Xilinx Artix-7 XC7A100T を採用している．DRAM には Micron 社製の 256MB DDR3 SDRAM を使用しており，FPGA と DRAM 間において，最大 800MB/s のバンド幅でデータを送受信することができる．USB3.0 Peripheral Controller として，Cypress Semiconductor 社製の EZ-USB (FX3CYUSB3014) を使用し

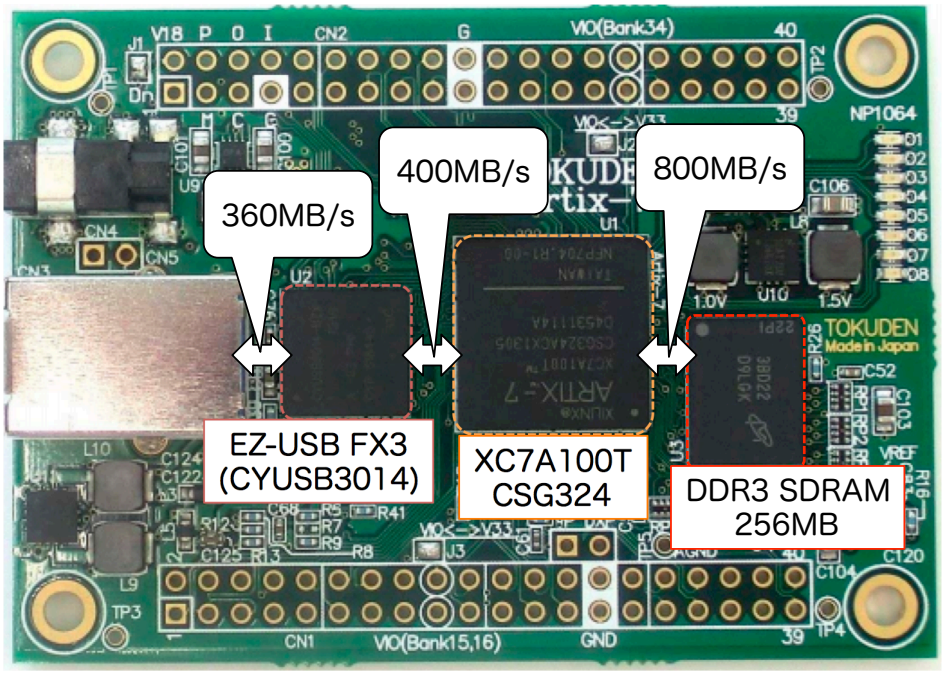

図 3.9: TKDN_ART7_BRD-2

ており，ホスト PC と EZ-USB 間で最大 360MB/s, EZ-USB と FPGA 間で最大 400MB/s のバンド幅でデータを送受信することができる．

我々は，**図** 3.2 示したソーティングロジックを Verilog HDL で記述し，Interface Module の実装では特殊電子回路株式会社 [9] が提供する IP コアを，DRAM Controller の実装では Xilinx 社が提供する IP コア [10] を利用した．FPGA 内の全てのモジュールは 100MHz にて動作し，DRAM は 400MHz で動作する．

3.4　検証

この FPGA アクセラレータの検証と性能評価を以下の手順で行った．

1. データ列と USB デバイスを初期化
2. 時間測定開始
3. FPGA アクセラレータにデータ列を送信
4. FPGA アクセラレータよりソート済みデータ列を受信
5. 時間測定終了

```
 1: #define ELM 32 * 1024 * 1024
 2:
 3: int main(){
 4:
 5:   init(data);                   // (1) Initialize Data Array and USB
 6:
 7:   start = getTime();            // (2) Start The Time Measurement
 8:   USBWrite(data,ELM);           // (3) Send Unsorted Data to FPGA
 9:   USBRead(data,ELM);            // (4) Receive Sorted Data from FPGA
10:   end = getTime();              // (5) Finish The Time Measurement
11:   elapsed_time = end - start;
12:
13:   error_check(data);            // (6) Check whether the Data is Accurate
14:   finalize(data, elapsed_time); // (7) Display elapsed_time and Finalize
15: }
```

(a) FPGA アクセラレータにソーティングを実行させる場合

```
 1: #define ELM 32 * 1024 * 1024
 2:
 3: int main(){
 4:
 5:   init(data);                   // (1) Initialize Data Array and USB
 6:
 7:   start = getTime();            // (2) Start The Time Measurement
 8:   MergeSort(data,0,ELM-1);      // (3)(4) Sort on CPU
 9:
10:   end = getTime();              // (5) Finish The Time Measurement
11:   elapsed_time = end - start;
12:
13:   error_check(data);            // (6) Check whether the Data is Accurate
14:   finalize(data, elapsed_time); // (7) Display elapsed_time and Finalize
15: }
```

(b) ホスト PC にソーティングを実行させる場合

図 3.10: FPGA アクセラレータの検証と性能評価に用いたコード

6. 受信データ列が正しいかをソート結果が記されたファイルを読み込み各要素を比較, エラーがあれば出力
7. データ列を格納していた領域を解放し USB デバイスを安全に閉じる

検証に用いたプログラムコードの概略を**図** 3.10 に示す.

図 3.10a は FPGA アクセラレータにソーティングを実行させるプログラムのコードであり，USBWrite 関数と USBRead 関数については特殊電子回路株式会社 [9] が提供するライブラリを用いている．入力するデータ列の生成には擬似乱数生成器である XorShift を使用しており，生成したデータ列は各測定で全て同一のものである．ソーティングの実行時間は，**図** 3.10a の 7 行目でソーティングの開始時刻を取得し，10 行目でソーティングの終了時刻を取得し，終了時刻から開始時刻を差し引くことで求めた．

図 3.10b はホスト PC 上でソーティングを実行させるプログラムのコードである．入力するデータ列の生成方法及びソーティングの実行時間の取得方法は**図** 3.10a と同様である．

FPGA アクセラレータの性能評価はこの 2 つのソーティングの実行時間を比較して行った．これらのプログラムのコンパイルには Microsoft Visual C++ 2013 Compiler を用いて，最適化オプションは/Ox に設定した．

第 4 章

評価

この章では,

- 高速化目標決定のための予備評価
- 提案する FPGA アクセラレータを実装した際のハードウェア使用量の評価
- 評価対象とするソーティングの要素数
- FPGA ボードとホスト PC 間の USB3.0 を介した通信のスループット
- 提案する FPGA アクセラレータのポータビリティの評価
- 提案する FPGA アクセラレータの性能評価

を述べる.

表 4.1: クイックソートとマージソートの実行時間

# of Elements	32M			64M		
Data Sequence	Random	Sorted	Reverse	Random	Sorted	Reverse
QuickSort	2.754 sec	>2hours	>2hours	5.82sec	Unmeasured	Unmeasured
MergeSort	**4.30**sec	1.67sec	1.66sec	8.51sec	3.25sec	3.13sec

表 4.2: ハードウェア使用率

K	Occupied Slices	Block RAM
8	23 %	14 %
16	30 %	26 %
32	47 %	50 %
64	73 %	97 %

4.1　予備評価

ソーティングアルゴリズムは数多く存在するが，一般的に実用的であるのはクイックソートやマージソートである．**表** 4.1 に，ランダムなデータ列 (“Random”)，ソート済みのデータ列 (“Sorted”)，逆順にソートされたデータ列 (”Reverse”) を入力としたときのクイックソートとマージソートの実行時間を示す．データ列は 32M 個と 64M 個の int 型の要素で構成され，1 要素は 4Bytes である．クイックソートとマージソートを C で記述し，Microsoft Visual C++ 2013 Compiler にて最適化オプションを/Ox に設定してコンパイルした．動作周波数 3.5GHz の Intel Corei7-3770K のシングルスレッドで実行させ，クイックソートとマージソートの実行時間を取得した．実行時間を比較したところ，“Random” はクイックソートの方が高速だが，“Sorted”，“Reverse” ではマージソートの方が遥かに高速であった．これは，ソート済みのデータ列，逆順にソートされたデータ列を入力としたときのクイックソートの計算量が最悪計算量 $O(n^2)$ である一方，マージソートの計算量は $O(n \log n)$ であることに起因している．クイックソートは一般的に最も高速だとされているが，この結果のように，入力のデータ列によって実行時間が大きく変化してしまう．そのため我々は，入力のデータ列に実行時間が影響を受けにくいマージソートの方が実用的なソーティングであると判断し，FPGA アクセラレータによるソーティングは，マージソートより高速になることを目指す．

4.2　ハードウェア使用量の評価

表 4.2 に各 way の Merge Sorter Tree を実装したときのハードウェア使用量の内訳を示した．2-Element Sorter の有無はこれに影響を及ぼさなかったが，ハードウェア使用率

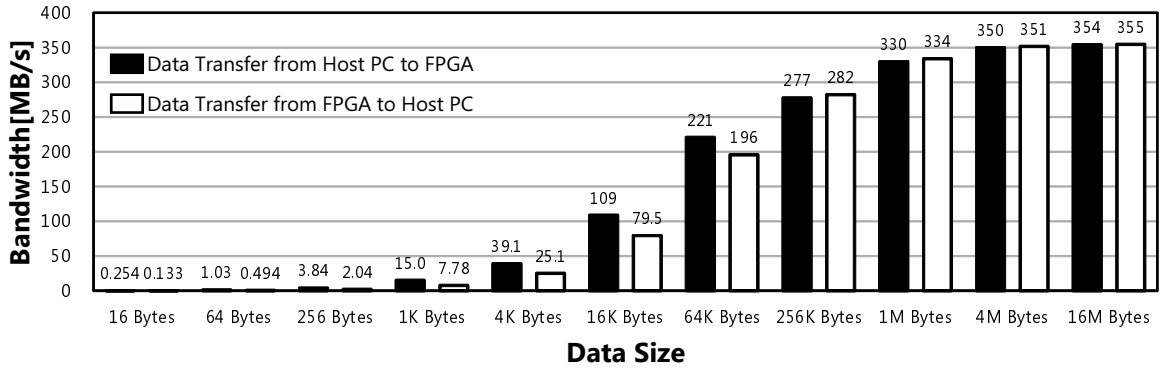

図 4.1: USB3.0 の通信速度

が K に応じてほぼ線形に増大していることが分かる．更に，このうち Block RAM は全て各 Buffer に使われていた．よって，各 Buffer の容量を調整することで最適化が可能であると考える．

4.3 USB3.0 による効率的なデータ通信

図 4.1 にホスト PC と FPGA ボード間におけるデータの通信速度 [MB/s] を計測した．この通信速度 [MB/s] は，それぞれ異なるデータサイズについて，送信と受信を適当な回数繰り返したときの時間を計測し，それらの算術平均で求めた値である．**図** 4.1 に示すように，データサイズが小さいときは通信のオーバーヘッドのため通信速度が非常に低く，データサイズが大きいほど通信速度は向上する．

前章で述べたように，USB3.0 を介してホスト PC から送信されてきた未ソートのデータ列は，Interface Module によって Input Buffer に格納される．これによって，FPGA アクセラレータはホスト PC からデータを受信しながらソーティングを実行できる．そして Final Phase においてはソーティングを実行しながら，Merge Sorter Tree から出力されるソートされたデータ列をホスト PC に送信している．このようにデータの送受信を効率的に行うことによって，ソーティングの実効性能を向上させることができる．

4.4 評価するデータサイズ

表 4.3 に，評価対象となるデータ列の要素数を示した (1 要素: 32bit の int 型整数)．ここで，以下の変数が導入されている．

表 4.3: Merge Sorter Tree の way 数・2-Element Sorter の有無・必要フェーズ数と要素数 (32M 以下) の対応表

Merge Sorter Tree	8-way		16-way		32-way		64-way
o	1	0	1	0	1	0	1
Phase 8	32M	16M					
Phase 7	4M	2M					
Phase 6	512K	256K	32M	16M			
Phase 5			2M	1M		32M	
Phase 4					2M	1M	32M

表 4.4: 評価環境

Host PC	Computer A	Computer B	Computer C	Computer D
Type	Tower	Laptop	Slim Desktop	Laptop
CPU	Intel Core i7 3770K	Intel Core i3 4010U	Intel Core i7 870	Intel Core Duo T2400
CPU Frequency	3.5 GHz	1.70 GHz	2.93 GHz	1.83 GHz
Memory Capacity	16 GB	4.0 GB	4.0 GB	1.0 GB
USB Ports	USB3.0 × 4，USB2.0 × 6	USB3.0 × 2	USB2.0 × 8	USB2.0 × 2

- o: 2-Element Sorter の有無を示す整数で，値の取りうる範囲は $\{0,1\}$．$o = 1$ は Interface Module に 2-Element Sorter が実装されていることを示し，$o = 0$ はそうでないことを示す．

このとき，$\{K, o\}$ は K-way Merge Sorter Tree を実装し Interface Module に 2-Element Sorter を{実装した ($o = 1$)，実装しない ($o = 0$)}FPGA アクセラレータを表す．要素数 N には Merge Sorter Tree の way を Final Phase まで全て用いる必要があるものが対応している．ただし，FPGA ボードの DRAM の容量が 256MB のため，32bit 要素をソーティングする場合その数 N は 32M 以下に限定される．

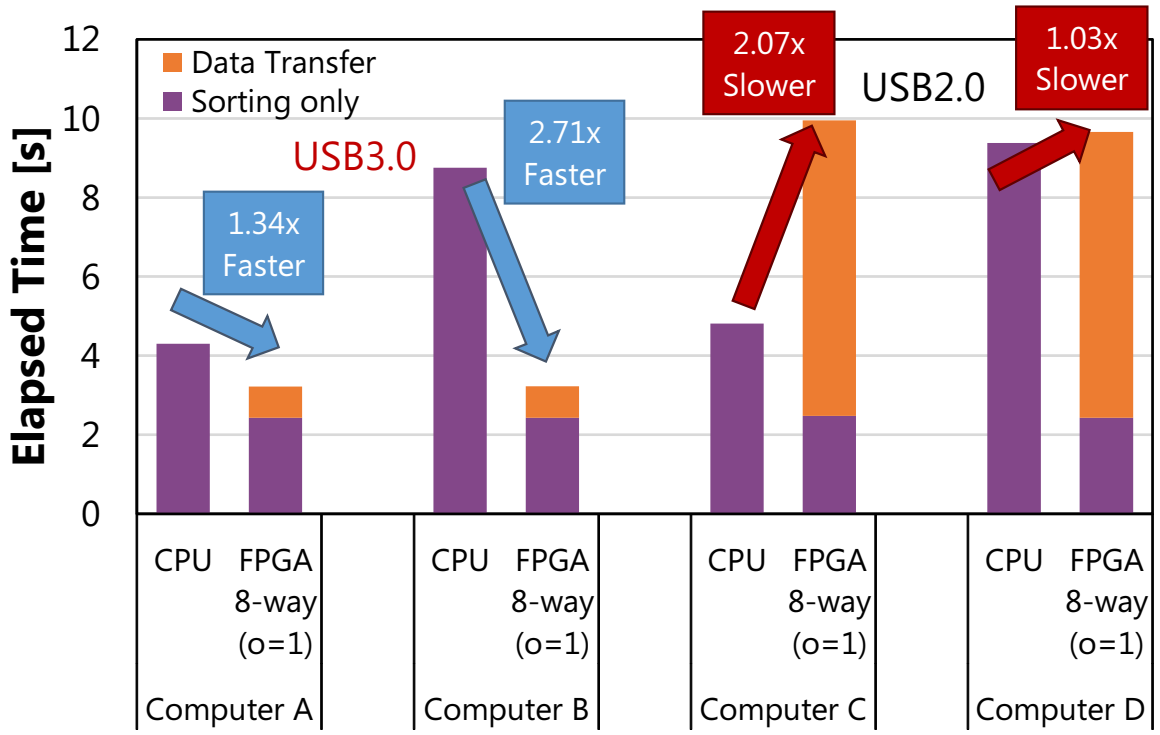

図 4.2: USB3.0 環境と USB2.0 環境間の 32M 個の int 型要素 (1 要素 : 4Bytes) のソーティングの実行時間の比較

4.5 評価環境

提案する FPGA アクセラレータの高いポータビリティを示すため，評価環境として FPGA アクセラレータに接続するホスト PC を**表** 4.4 のように 4 台用意した．Computer A と Computer B を USB3.0，Computer C と Computer D を USB2.0 にて FPGA アクセラレータと接続し，$\{K, o\} = \{8, 1\}$ の FPGA アクセラレータで 32bit 要素 32M 個のソーティングを実行させた場合の実行時間とホスト PC でマージソートをシングルスレッドで実行させた場合の実行時間を比較した．

前述した環境において”Random” 系列の 32M 個の int 型要素 (1 要素 : 4Bytes) のソーティングに要した時間を計測したところ，**図** 4.2 のようになった (**表** 4.1 に示したソーティング時間はこの Computer A にて測定・算出したものである)．Computer C, Computer D においてはホスト PC のみでソーティングを実行したときより実行速度が低下したが，

Computer A, Computer B においてはホスト PC のみで実行したときより実行速度が向上した．特に Computer B においては 2.71 倍の高速化を達成した．

次に，Computer A と Computer B の場合において，FPGA アクセラレータによるソーティングが高速化した要因を調べた．**図** 4.2 において，どの PC においても FPGA アクセラレータによるソーティングの実行時間は同一だが，Computer A と Computer B においては送信時間と受信時間が大幅に短縮されていた．これは，FPGA アクセラレータが USB3.0 の通信速度の恩恵を受けていることを示している．

FPGA アクセラレータに USB2.0 を介して接続した Computer C，Computer D では，ソーティングを FPGA アクセラレータにオフロードしても速度向上が得られず，特に Computer C では 2.07 倍遅くなった．**図** 4.2 において，比較的モダンな CPU を搭載する Computer C においては，ホスト PC-FPGA 間の通信時間がホスト PC 上でのソーティング時間を上回っている．このため，FPGA アクセラレータに USB2.0 を介して接続した場合では，FPGA アクセラレータに処理をオフロードしても，逆にソーティングの実行速度が低下してしまう．

対して，USB3.0 を介して接続した Computer A と Computer B においては，通信時間の大幅な削減により，実行速度がそれぞれ 1.34 倍，2.71 倍となった．このことから，USB3.0 接続をサポートする PC ならば，提案する FPGA アクセラレータによってソーティングの高速化を達成することができる．

4.6　提案する FPGA アクセラレータの性能評価

前節では，USB2.0 環境では通信のボトルネックのためどのような FPGA アクセラレータを設計しても高速化が不可能であることと，USB3.0 環境での高いポータビリティを示した．この節では，USB3.0 環境で様々な系列・要素数のデータのソーティングを FPGA アクセラレータにオフロードしたときの実効性能を，Computer A の CPU と比較する．

次の頁から，4.4 節で示した要素数のそれぞれに対し，対応する $\{K, o\}$ の FPGA アクセラレータにソーティングをオフロードした際の性能を述べる．要素数 256K，512K，1M，2M，4M，16M，32M の順に述べる．

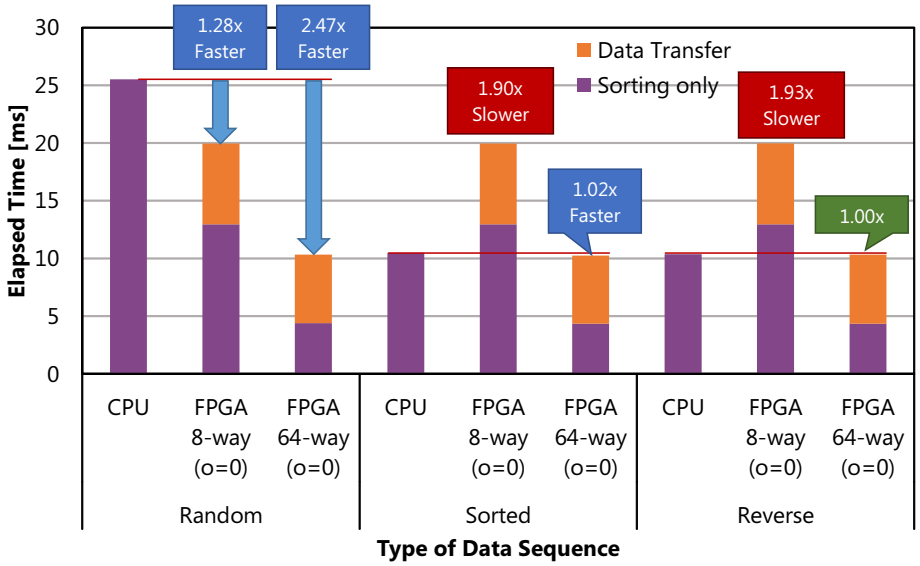

図 4.3: 256K 要素のデータを与えたときの FPGA アクセラレータの性能

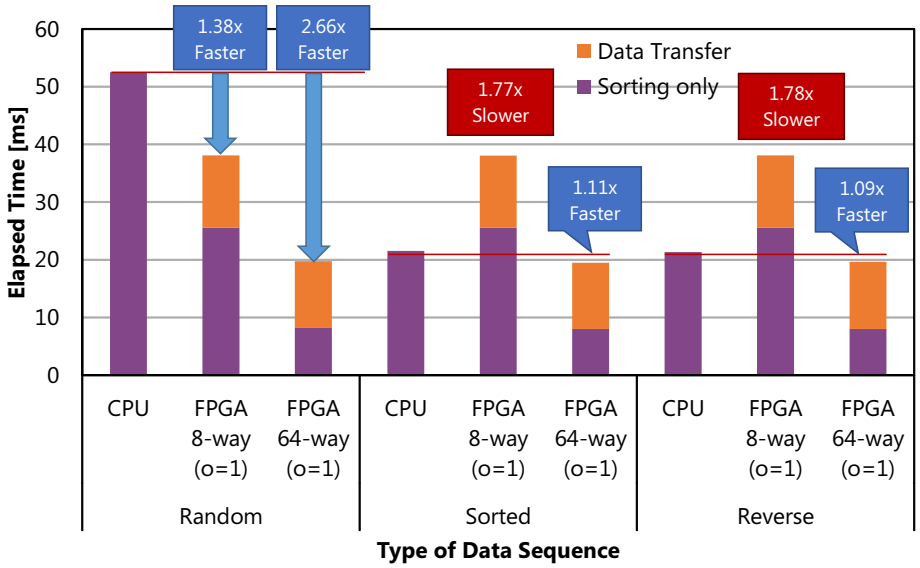

図 4.4: 512K 要素のデータを与えたときの FPGA アクセラレータの性能

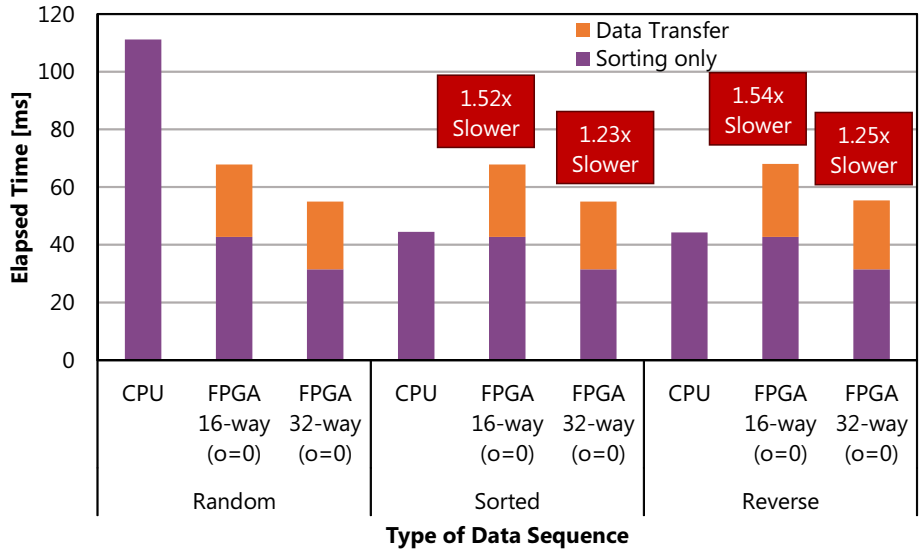

図 4.5: 1M 要素のデータを与えたときの FPGA アクセラレータの性能

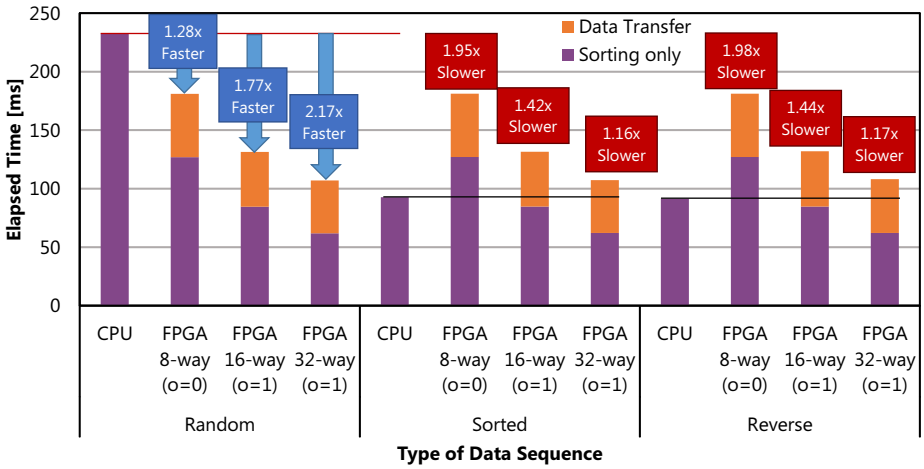

図 4.6: 2M 要素のデータを与えたときの FPGA アクセラレータの性能

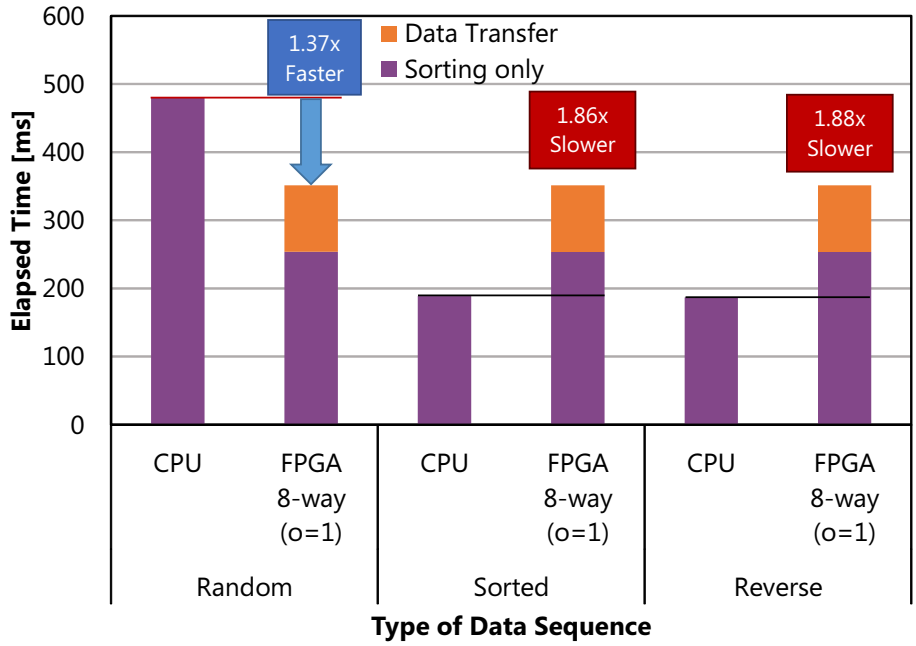

図 4.7: 4M 要素のデータを与えたときの FPGA アクセラレータの性能

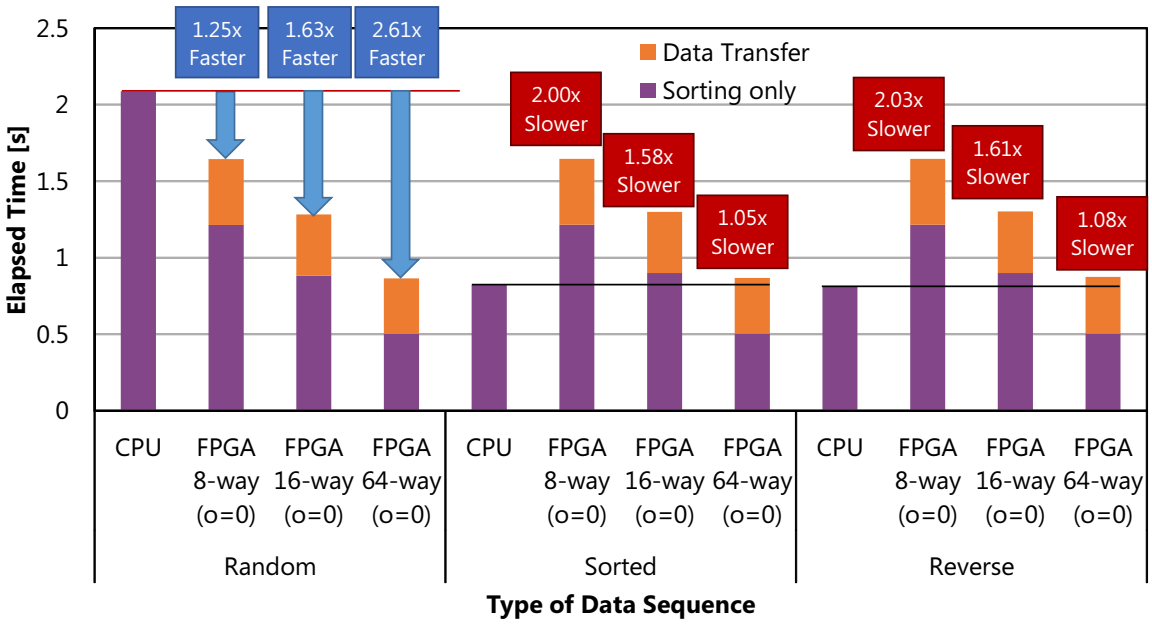

図 4.8: 16M 要素のデータを与えたときの FPGA アクセラレータの性能

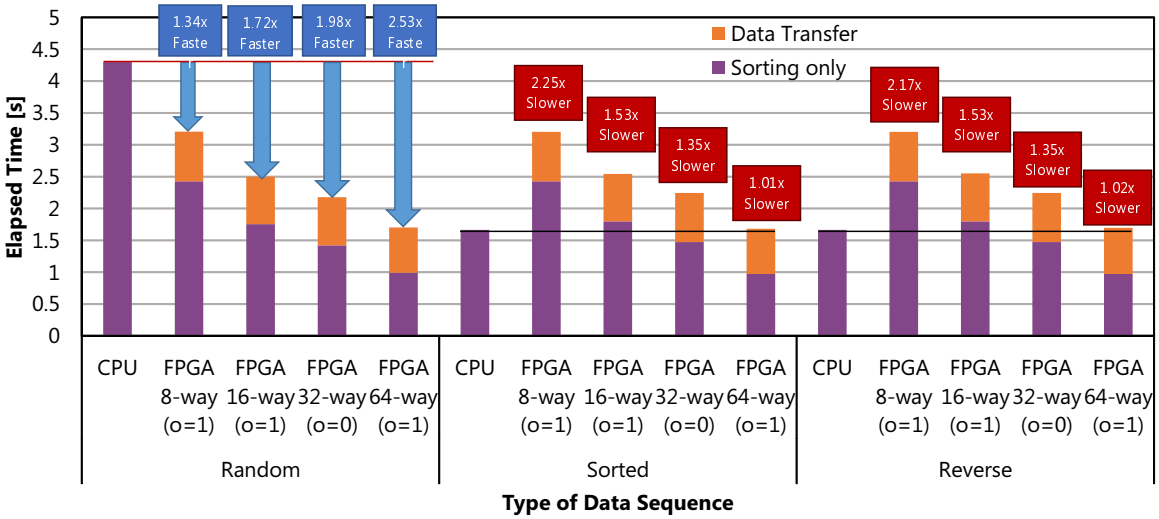

図 4.9: 32M 要素のデータを与えたときの FPGA アクセラレータの性能

以上を見ると，CPU の分岐予測やコンパイラの支援などが有効に働く”Sorted” 系列や”Reverse” 系列を入力した場合は，FPGA アクセラレータにソーティングをオフロードしても，多くの場合で性能が低下した．しかし，$\{K, o\} = \{64, 1\}$ の FPGA アクセラレータのソーティングの性能は，512K 要素を与えたときに CPU 比 1.1 倍程度となった．また，“Random” 系列を与えたときは全ての場合において性能向上し，特に 512K 要素のソーティングにおいて $\{K, o\} = \{64, 1\}$ のものが CPU 比 2.66 倍の高速化を達成した (**図** 4.4).

第 5 章

結論

本論文では，高速な外付けインターフェースである USB3.0 での接続をサポートする高速でポータブルな FPGA アクセラレータを提案し，その設計と実装，ソーティングの実効性能の評価について述べた．

まず，提案する FPGA アクセラレータの高いポータビリティを示すため，USB3.0 環境と USB2.0 環境を用意し評価を行った．その結果，USB2.0 を介して接続した場合では通信速度が低いためにソーティングを高速化できなかったが，USB3.0 を介して接続することによってソーティングの高速化を達成した．これにより，USB3.0 環境における高いポータビリティを示した．

また，提案する FPGA アクセラレータにて様々なデータのソーティングの実効性能を評価した．その結果，高パフォーマンスの Core i7-3770K と比較して，“Sorted” 系列や”Reverse” 系列を入力としたときは最大でも 1.11 倍の高速化に留まったが，“Random” 系列を入力に与えた場合は最大 2.66 倍の高速化を達成した．これにより，提案する FPGA アクセラレータが Core-i7 3770K と比較して比較的高い性能を持つことを示した．

以上より，提案する FPGA アクセラレータが高い性能とポータビリティを両立していることを明らかにした．

今後の課題として，更に高速なソーティングアクセラレータを実装した場合の性能評価が挙げられる．また，特電ライブラリが Windows を対象としていたため，ホスト PC の OS は Windows である必要があった．そのため，提案する FPGA アクセラレータのポータビリティを更に高めるために，このライブラリを他の OS に移植することを検討している．今回，高速化させるアプリケーションはソーティングであったが，画像処理やグラフ処理といった他の実用的なアプリケーションでの評価も検討している．

謝辞

本研究を進めるにあたり，適切な指導を行って下さいました指導教員の吉瀬謙二准教授に深く感謝致します.

また，吉瀬研究室の皆さんには大変お世話になりました．ケーキを提供してくれるなど研究室に華やかにしてくれた小川愛理さん，とある実験からのパートナーである味曽野智礼君，いつも面白い話をして研究室を賑やかにして下さった奥村開里先輩，研究室所属時より諸々の世話をして下さった森悠先輩，全くの異分野からの所属にも関わらず今やギャップを感じさせない松田裕貴先輩，お忙しい中原稿の校正して下さった Thiem Van Chu 先輩，ありがとうございました.

特に，研究に全面的に協力頂くのみならず，本論文含め，原稿にも最後まで深く携わって下さった小林諒平先輩に深い感謝の意をここに表します．先輩の支援無くして本研究は有り得ません.

これからも研究は勿論のこと，様々な面で精進していきたく思いますので，どうか宜しくお願い致します.

最後に，このような私を支えて下さった両親に深く感謝致します.

参考文献

[1] USB-IF. http://www.usb.org/.

[2] Rene Mueller, Jens Teubner, and Gustavo Alonso. Sorting networks on fpgas. *The VLDB Journal*, Vol. 21, No. 1, pp. 1–23, February 2012.

[3] K. Ratnayake and A. Amer. An fpga architecture of stable-sorting on a large data volume : Application to video signals. In *Information Sciences and Systems, 2007. CISS '07. 41st Annual Conference on*, pp. 431–436, March 2007.

[4] J. Martinez, R. Cumplido, and C. Feregrino. An fpga-based parallel sorting architecture for the burrows wheeler transform. In *Reconfigurable Computing and FPGAs, 2005. ReConFig 2005. International Conference on*, pp. 7 pp.–, Sept 2005.

[5] A. Putnam, A.M. Caulfield, E.S. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G.P. Gopal, J. Gray, M. Haselman, S. Hauck, S. Heil, A. Hormati, J.-Y. Kim, S. Lanka, J. Larus, E. Peterson, S. Pope, A. Smith, J. Thong, P.Y. Xiao, and D. Burger. A reconfigurable fabric for accelerating large-scale datacenter services. In *Computer Architecture (ISCA), 2014 ACM/IEEE 41st International Symposium on*, pp. 13–24, June 2014.

[6] K. Sano, Y. Hatsuda, and S. Yamamoto. Multi-fpga accelerator for scalable stencil computation with constant memory bandwidth. *Parallel and Distributed Systems, IEEE Transactions on*, Vol. 25, No. 3, pp. 695–705, March 2014.

[7] R. Kobayashi and K. Kise. Scalable stencil-computation accelerator by employing multiple small fpgas. *IPSJ Transactions on Advanced Computing Systems*, Vol. 6, No. 4, pp. 1–13, oct 2013.

[8] Dirk Koch and Jim Torresen. Fpgasort: A high performance sorting architecture exploiting run-time reconfiguration on fpgas for large problem sorting. In *Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable*

Gate Arrays, FPGA '11, pp. 45–54, New York, NY, USA, 2011. ACM.

[9] Tokushudenshikairo Inc. http://www.tokudenkairo.co.jp/art7/index.html.

[10] Memory Interface Generator (MIG). http://www.xilinx.com/products/intellectual-property/MIG.htm.

[11] Maxeler Technologies. https://www.maxeler.com/.