金融商品アルゴリズム取引システムのハードウェアアクセラレーション

小林 弘幸[1,†1,a)]　田中 清史[1,b)]

概要：今日の金融商品の電子取引においては，売買注文の大部分はあらかじめプログラムされた取引アルゴリズムにより自動的に送信されている．売買注文は取引所に到達した順に売り・買い注文のマッチングが行われるため，電子取引に関わるシステムには可能な限り低遅延で通信と演算処理を行うことが要求される．通常の業務アプリケーションのネットワーク処理においては，電気信号による情報の受信時から起算して，情報を解釈・処理し，処理結果を送信する一連のプロセスにおいて，OS の TCP/IP プロトコルスタック及び通信アプリケーションの多階層を経由するために相応の CPU 時間を必要とし，遅延が発生する最大の要因となっている．そこで本研究では金融市場における低遅延性の要求に応えるために，汎用プロセッサ上のソフトウェア処理に代えて，安価な SoC FPGA 上に専用回路を構成しハードロジックにより処理を行うアクセラレータの試作機を開発した．このアクセラレータを用いることで，FPGA 内で発生するレイテンシを 1 マイクロ秒未満に抑えることができ，サーバ側で計測される遅延値も 80 %程度削減されるという結果が得られた．

キーワード：アルゴリズム取引，低遅延取引，FIX プロトコル，TCP/IP，SoC FPGA

Hardware Acceleration for Algorithmic Trading of Financial Products

Hiroyuki Kobayashi[1,†1,a)]　Kiyofumi Tanaka[1,b)]

Abstract: Recently, most financial transactions are executed with orders sent automatically from pre-programmed trading algorithms. Since trading venues match and execute buying and selling orders in order of arrival, electronic trading systems are required to communicate and process as quickly as possible. In the case of commercial network application programs, a process starts in receiving packets as a form of electronic signal. It interprets and handles the received information, and sends out the result. These consecutive events take time on CPU, since they run through multiple layers in TCP/IP protocol stack and application programs, which is the biggest factor to cause the delay. Thus, we developed a hardware accelerator prototype on low-priced SoC FPGA in which the dedicated hardwired logic performs fast processing, instead of the software processing on the CPU, to satisfy the requirement of low latency in the financial markets. The evaluation results show that the latency on the FPGA is less than 1.0 micro second, and that around 80% of the total latency, which is observed at the connected server, can be reduced by using this accelerator.

Keywords: Algorithmic Trading, Low Latency Trading, FIX Protocol, TCP/IP, SoC FPGA

1. はじめに

近年，ネットワーク処理や組込みシステム/自動運転の制御の目的で，リアルタイムで取得される情報に対して瞬間的に判断し低遅延で応答を返す専用ハードウェアを開発する動きがある [1]．ハードウェア処理を行うことで性能が向上するアプリケーション分野は多岐に渡るが，低遅延性を特に要求する領域の一つに金融商品取引がある．

2000 年前後から各国の主要取引所において株式などの金融商品が電子的に取引されるようになり，以降，迅速に取引関連情報を送受信する環境の整備が進み，現在では，主要な取引参加者は取引所のシステムが存在するデータセンター内にサーバを導入しており，情報更新に対して可能な限り高速に応答するシステムの開発競争が進んでいる [2]．

[1] 北陸先端科学技術大学院大学
JAIST, Nomi, Ishikawa 923–1292, Japan
[†1] 現在，株式会社 Sigma Technology
Presently with Sigma Technology Co., Ltd.
[a)] hiroyuki.kobayashi@sigma-technology.biz
[b)] kiyofumi@jaist.ac.jp

本研究は，市場における代表的な発注アルゴリズムである逆指値を対象とし，ハードウェアアクセラレーションにより価格情報の受信から注文情報の送信までに要するレイテンシを低減することを目的とする．通信プロトコルはEthernet，TCP/IP 及び，金融市場で最も一般的に用いられている FIX プロトコル [3] を対象とする．従来のソフトウェアを用いたネットワーク処理は，幾重にもわたる階層（割り込みハンドラ，Ethernet ドライバ，IP 層，TCP 層，ソケットシステムコール，FIX エンジン，アプリケーション）を経由することで必然的に遅延が発生している．本研究においてはFPGA 上の専用回路を用いることで，この遅延を低減し通信を高速化することを意図している．一方，逆指値は非常にシンプルなロジックであり，取引アルゴリズムの演算処理の高速化を意図するものではない．

FPGA を用いた低遅延取引の研究報告は次章で示すように幾つか存在するが, 既存研究においては実現のための具体的な回路構成の記述は十分でなく使用している機材も高額であるため, 専門事業者以外には手が出せないのが現状である. 本研究はハードウェアアクセラレータを安価で容易に入手可能な FPGA デバイスで実現し, また正常な FIX メッセージとしてサーバ側で受理されるために必要なハードウェア及びソフトウェアの構成方法を示している. 低遅延取引による投資家間の公平性に関する懸念が提示される [4] 中, 一般投資家も技術の恩恵を受けられるように低遅延取引の普及に貢献することを本研究の目的としている.

2. 低遅延取引にかかわる研究

ハードウェアアクセラレータを利用した低遅延取引の研究事例は 2007 年頃から存在する[*1]. 代表的な研究としては，複数情報源からの価格情報を集約するサーバのプロセッサとして FPGA を用いることで遅延を 26μs 以下に短縮できることを実証した Gareth W. Morris, David B.Thomas and Wayne Luk（2009） [5]，ネットワークプロトコルのデコード処理を FPGA にオフロードすることで発注判断の前処理部分を高速化できることを示した Christian Leber, Benjamin Geib, Heiner Litz（2011） [6]，FPGA を用いることで平均的に 2.7μs で応答可能で，かつソフトウェア処理に比べてバラつきが小さいことを確認した RobinPottathuparambil et al.（2011） [7] が挙げられる．

ベンダー企業の発行する資料や論文にも重要な研究が存在する．John W. Lockwood, Michaela Blott et al.（2012） [8] は FPGA 搭載 NIC を用いて 1μs（FPGA 内部では 200ns）の遅延で取引が可能であることを報告している．またこの論文の 1〜3 章は低遅延取引における FPGA アクセラレータの利用方法と現状に関するサーベイとして利用できる．Finteligent Trading Technology Community（FTTC）によるリサーチレポート（2012） [9] には，ネットワーク処理に関しての遅延計測の具体的な方法が記載されている．ARGON DESIGN, ARISTA, FTTC によるリサーチレポート（2013） [10] は，価格情報メッセージの到達が完了する前に必要な情報が入手できた時点で注文情報の送信を始めるように回路を構成し，150ns で取引できることを報告している．井上（2012） [11] は日本の証券市場の実取引データを用いて株価の変化点検出等の複合イベント処理について，FPGA 搭載 NIC を用いることでプロセッサよりも 12.3 倍の速度で処理できることを報告している．

「FPGA の原理と構成」天野編著（2016） [12] の 7 章 6 節は NIC（ASIC)+ CPU の通常構成，FPGA NIC ＋ CPU の構成，SoC FPGA のワンチップ構成の比較を行い，低遅延取引における SoC FPGA の優位性を指摘している．

3. ソフトウェア処理

3.1 FIX プロトコル

FIX プロトコルは取引所からの価格情報や取引参加者からの注文情報といった，金融商品の電子取引に必要な種々の情報のやりとりを統合的に扱うプロトコルで，非営利団体の FIX Trading Community により策定され，取引所，金融機関，機関投資家等の市場関係者の間の商取引情報の伝達手段として最も広範囲に用いられている．

FIX メッセージは文字列形式（ASCII コード）で表現され，これに TCP ヘッダ，IP ヘッダ，Ethernet ヘッダ，プリアンブル及び FCS を付加した Ethernet フレームが実際に送受信される．本研究では FPGA のハードロジックで注文を行う場合においても，FIX メッセージはソフトウェア（発注クライアント）で生成することとし，売買タイミング判断及び下位レイヤのヘッダの生成にハードロジックを活用する．

3.1.1 メッセージタイプ

FIX メッセージは管理系メッセージと業務系メッセージに大別できる．管理系メッセージはログオン/ログアウトやハートビート，再送要求などの L5（セッション層）の処理に相当し，業務系メッセージは新規注文や価格情報配信などの実取引に関わる L7（アプリケーション層）のアクションに相当する．

FIX4.4 ではメッセージタイプは 0〜9, A〜Z, a〜z, AA〜AZ，BA〜BH の 96 種類が定義されているが，そのうち本研究に関わるもののみを表 1 に列挙する．“C” はクライアント，“S” はサーバを意味する．取引において特に重要なのは Market Data Snapshot Full Refresh（W）と New Order Single（D）の 2 つのメッセージタイプであり，クライアン

[*1] ソフトウェア側の修正でも遅延を減少させることもある程度の水準までは可能だが,1 マイクロ秒未満の領域ではソフトウェア処理で達成したという報告は確認できず, また OS の再設計は（ネットワーク処理部分に限定しても）FPGA で処理をバイパスする以上に開発期間を要するので今回は検討していない.

表 1　メッセージタイプ
Table 1　Message Type.

コード	方向	名称
0	S → C → S	Heartbeat
A	C → S → C	Logon
5	C → S → C	Logout
V	C → S	Market Data Request
W	S → C	Market Data Snapshot Full Refresh
D	C → S	New Order Single
F	C → S	Order Cancel Request
8	S → C	Execution Report

トに Market Data が到達してから New Order Single を送出するまでをハードウェアアクセラレーションによるレイテンシ削減の対象とする．

3.1.2　FIX エンジン

FIX プロトコルを用いたアプリケーションを開発するにあたり，FIX エンジンと呼ばれる基本機能を備えたクラスライブラリとして，フリーソフトの QuickFIX を用いる [13]．QuickFIX は管理系メッセージを内包しており，管理系メッセージの送出，処理に関しては新たに記述する必要がないため，業務系メッセージの送出と受信時の手続きをコーディングすることとなる．

QuickFIX は C++/Java/.net/Go の 4 バージョンがあるが，本研究において仮想取引所サーバの構築には GUI 開発の容易さから C#（.net）を用いる（バージョンは QuickFIX/n 1.7.0）．発注用クライアントは Linux 環境で動作させることと，レイテンシ比較に用いるために実行速度に優れる C++を用いる（バージョンは QuickFIX 1.14.3）．

3.2　取引所シミュレータ（サーバ）

公設の取引所（証券取引所やデリバティブ取引所）では，取引参加者の買い注文と売り注文を集約し，売り注文価格≦買い注文価格となる注文対（買い及び売りの組）が出現した場合，即座に約定させる．（新規注文に対して複数の既存注文が条件を満たす場合は価格→発注時間の優先順位で約定させる注文を決定する．）この注文マッチングの機能をシミュレートする取引所シミュレータを実装する．取引所シミュレータは FIX 通信のサーバ側の役割で，複数のクライアントと接続することが可能である．

取引所シミュレータは，各クライアントからの New Order Single（新規注文:D），Order Cancel Request（注文取消:F），Market Data Request（価格情報要求:V）のメッセージを受け入れ，新規注文と注文取消に対しては Execution Report （注文受付/約定報告メッセージ:8）を返し，価格情報要求に対しては Market Data Snapshot Full Request（価格情報:W）を返す．加えて，各上場銘柄ごとに板情報（現在の注文状況）のリストを保有し，新規注文，注文取消メッセージを受け付ける際に板情報が更新される度に，価

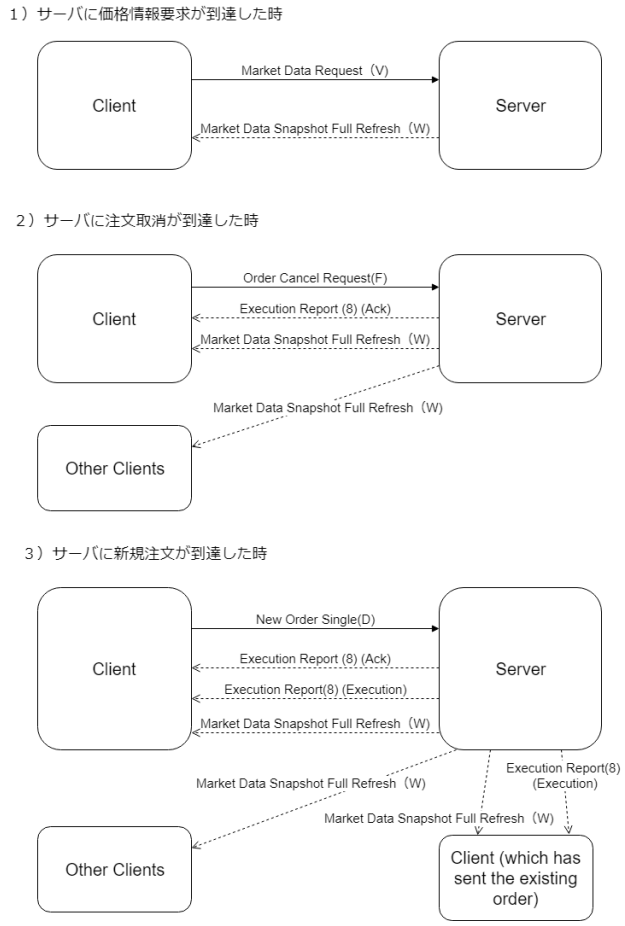

図 1　取引所シミュレータの挙動
Fig. 1　Behavior of exchange simulator.

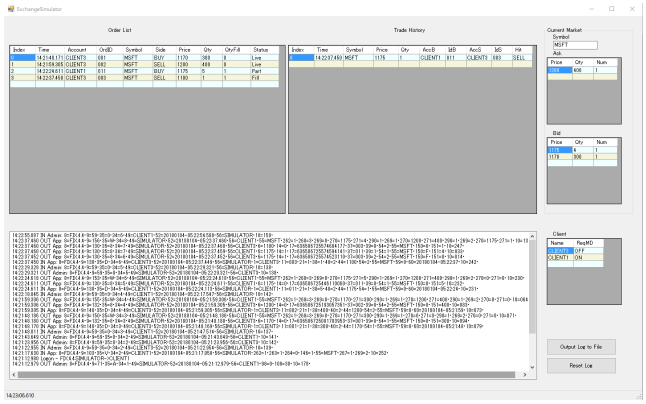

図 2　取引所シミュレータ (GUI)
Fig. 2　GUI on exchange simulator.

格情報要求を送信した全てのクライアントに対して価格情報（W）を送信する．また，新規注文の結果，売買が成立した場合，取引当事者双方のクライアントに対して注文受付/約定報告（8）メッセージを送信する．以上をまとめたものが図 1 である．

取引所シミュレータは注文状況，約定結果や取引のログを表示するために図 2 の GUI を持つ．

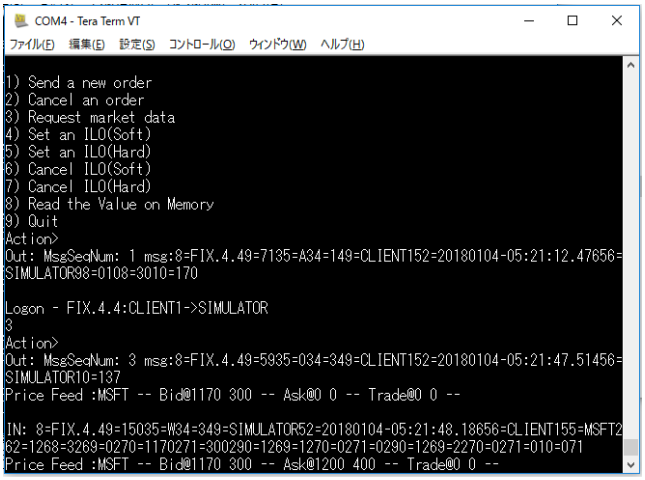

図 3 発注クライアント (CUI)
Fig. 3 CUI on order client software.

3.3 発注クライアントと取引アルゴリズム

発注クライアントは図 3 のような CUI アプリケーションであり，コマンドを入力することで取引所シミュレータに対し新規注文（D），注文取消（F），価格情報要求（V）のメッセージを送信できる．また，取引アルゴリズムを登録することで，取引所シミュレータから価格情報更新（W）メッセージを受信時に，あらかじめ決められた条件を満たした場合，新規注文（D）を自動的に送信することができる（図 4 参照）．

本研究においては，この取引アルゴリズムとして「逆指値」を設定している．逆指値は価格があらかじめ設定された数値以上になった場合に買い注文を出す，あるいは設定された数値以下になった場合に売り注文を出すロジックで，デジタル回路の 1 クロックサイクルで処理できる最もシンプルな取引アルゴリズムである．逆指値はストップオーダーとも呼ばれ，リスクを限定的にする目的で使われる．

金融市場で用いられる他の取引アルゴリズムは逆指値より複雑であるが，それらの複雑な取引アルゴリズムの処理はハードウェア化の対象としていない．その理由として，大多数の取引アルゴリズムは価格変動に起因して注文を送出するが、次の瞬間に起こりうる価格変動のパターンは限られており，その全てのパターンに対してソフトウェア側で発注判断の計算をあらかじめ行うことにより，ハードウェアの処理は「価格が○○円以上（以下）になれば注文を出す」という逆指値と同じ形に帰着できることが挙げられる．[*2] そのため，本研究では取引アルゴリズムではなく通信処理の高速化に焦点を当てている．

[*2] 取引アルゴリズムの処理時間はアルゴリズムの性質によるが，通信に起因する遅延よりも小さくなるように逆算されて取引アルゴリズムが設計されることが多く，また実用化する際はサーバをデータセンターに置くので演算処理の高速化には GPU やメニーコアプロセッサも利用可能であるため，並列可能な取引アルゴリズムの計算に要する遅延はボトルネックにならないと考えられる．

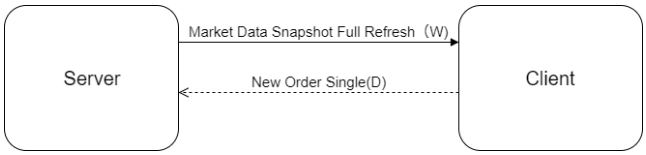

図 4 発注クライアントのアルゴリズム取引
Fig. 4 Algorithmic Trading on order client.

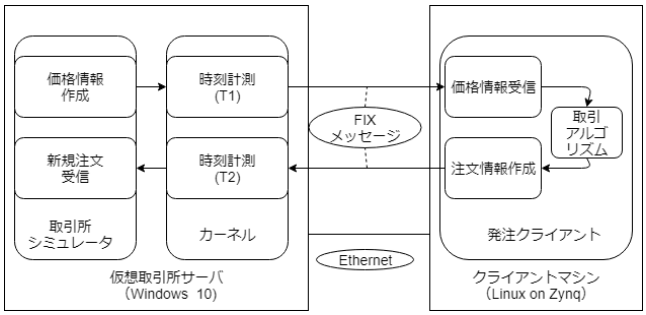

図 5 システム全体像
Fig. 5 Whole image of system.

3.4 システム全体像

取引所シミュレータを起動する仮想取引所サーバ（Windows PC）と発注クライアントを起動するクライアントマシン（PYNQ-Z1 上の Linux）は図 5 のようにイーサネットケーブル（1000BASE-T）で接続され，FIX メッセージをやり取りする．時刻計測はサーバ側で行う．

4. ハードウェア処理

4.1 FPGA（Zynq）

本節ではハードウェア設計に利用する FPGA デバイスとバスについて述べる．

3 節で述べた発注クライアントソフトウェアを用いる方式，およびハードウェアアクセラレータ方式の両方を実行可能な環境として，CPU と FPGA が 1 チップとなった Xilinx 社の Zynq-7020 [14] を搭載する PYNQ-Z1 ボード（Digilent 社） [15] [16] を用いる．これにより実現されるシステムは，2 節で紹介したアクセラレータ群に対し，安価であることに加え，ワンチップ構成であるために低遅延・高帯域幅でプロセッサ・FPGA 間のデータ移動が可能である点が特徴的である．

Zynq の PS 部には CPU（ARM Cortex-A9 のデュアルコア）とギガビットイーサネットコントローラ（Cadence Gigabit Ethernet MAC）が存在し，PYNQ-Z1 ボード上に DDR3 DRAM が存在する．通常の構成において，イーサネットコントローラは受信したフレームを DRAM に転送し，CPU が DRAM 上のフレームを読む．送信時は CPU が送信すべきフレームを DRAM に書き込み，イーサネットコントローラが DRAM 上のフレームを読む．一方，本研究ではドライバを改造し，フレームの転送先を PL 部にある BRAM に変更することで，PL 部のユーザ回路からも

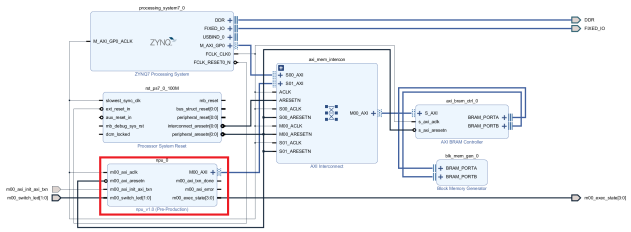

図 6　PL 部回路構成
Fig. 6　Circuit diagram for PL.

送受信するフレームデータにアクセス可能にしている．

4.2　アクセラレータ全体像

本節では本研究で作成するアクセラレータの回路構成，ステートマシン，BRAM のアドレス設定に着目し，アクセラレータの全体像を述べる．

SoC FPGA 形式のハードウェアアクセラレータを用いながら回路構成を工夫し, レイテンシセンシティブな箇所に関してはハードウェアで処理し, 通信遅延の改善に直結しない部分はソフトウェア資産を利用することで, 低遅延性を保ちつつデジタル回路をシンプルに留めるためのシステムアーキテクチャを設計したことが本研究の特徴である．

4.2.1　回路構成

作成した回路の全体図を図 6 に示す．実際に設計する回路（IP）は図 6 の左下の赤枠で囲まれる FIX プロセッシングユニットで，AXI バス経由で BRAM と接続される．PS 部（AXI3）は AXI Interconnect の内部にあるプロトコルコンバータで AXI4 に変換され，PL 部の BRAM と接続される．

4.2.2　ステートマシン

FIX プロセッシングユニットは 3 つの「状態」を持ち，この状態を順序回路の制御に用いている．状態遷移図を図 7 に示す．

4.2.3　BRAM

BRAM Controller を利用し，物理アドレスを割り振ることで，明示的に PS 部や他の IP からアクセスする．本研究では Linux 上のソフトウェア（Ethernet ドライバ及び発注クライアントアプリ）と PL 部のデータの受け渡し用に BRAM の特定アドレス（0x4000 0000 – 0x4000 0FFF）（4KB）を確保している．このアドレス領域にアクセスすることで CPU からも FIX プロセッシングユニットからもデータを Read/Write できる．具体的な仕様を表 2 に示す．

4.3　受信・送信フレーム

本節において，BRAM に書き込まれる受信フレームと送信フレームについて述べる．

価格情報を表すフレームの受信時に，当該フレームから価格情報を抜き出し，設定されている逆指値と比較を行い，

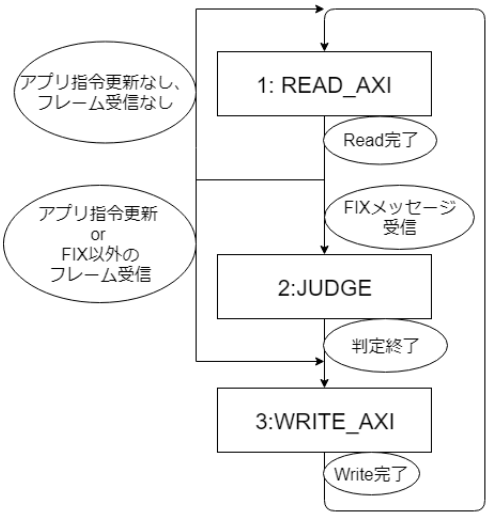

図 7　ステートマシン
Fig. 7　State machine.

表 2　BRAM アドレス設定
Table 2　BRAM address settings

バイト長	内容	ハード	ドライバ	アプリ
256	送信バッファ	W	R	–
128	各種フラグ格納領域	R/W	R/W	R/W
256	アプリからの指令領域	R	–	W
256	受信バッファ	R	W	–
3200	受信バッファ	–	W	–

発注する場合に FIX メッセージに TCP，IP，MAC のヘッダを付加し BRAM 中の送信バッファに書き込む．ここで付加される各ヘッダは受信した価格情報フレームから作成される．以下で具体的な Ethernet フレームを用いて方法を述べる．

4.3.1　L2（MAC）

データリンク層（L2）で付加される L2 ヘッダは Ethernet-II で定義されている．送信フレームの L2 ヘッダは単純に受信フレームの L2 ヘッダに対して宛先と送信元の MAC アドレスを入れ替えて作成できる．Ethernet-II ではプリアンブルと FCS も定義しているが，この 2 項目はイーサネットコントローラが処理し PL 部には届かないため対象としない．

4.3.2　L3（IP）

ネットワーク層（L3）で付加される L3 ヘッダは RFC791 で定義される IP（v4）である．L3 ヘッダの項目中のパケット長とチェックサムに関しては回路上で計算が必要となる．

4.3.3　L4（TCP）

トランスポート層（L4）で付加される L4 ヘッダは RFC793 で定義される TCP である．本研究では Timestamps を無効にし，オプション項目を持たないように設定する．送信パケットに含まれる確認応答番号は受信したシーケンス番号に受信したペイロードの長さ（バイト）を加えたもので

表 3　FIX メッセージ（価格情報）
Table 3　FIX message(Market Data Snapshot Full Refresh)

番号	タグ名	値	文字数
8	BeginString	FIX.4.4	7
9	BodyLength	159	3
35	MsgType	W	1
34	MsgSeqNum	22	1 ～ 6
49	SenderCompID	SIMULATOR	1 ～ 10
52	SendingTime	20180112-23:28:59.432	21
56	TargetCompID	CLIENT1	1 ～ 10
55	Symbol	MSFT	1 ～ 4
262	MDReqID	1	1
268	NoMDEntries	3	1
269	MDEntryType	0	1
270	MDEntryPx	1170	1 ～ 6
271	MDEntrySize	296	1 ～ 4
290	MDEntryPositionNo	1	1
269	MDEntryType	1	1
270	MDEntryPx	1200	1 ～ 6
271	MDEntrySize	497	1 ～ 4
290	MDEntryPositionNo	1	1
269	MDEntryType	2	1
270	MDEntryPx	1170	1 ～ 6
271	MDEntrySize	1	1 ～ 4
10	CheckSum	046	3

あり，回路上で計算する必要がある．トランスポート層でもネットワーク層同様チェックサムの演算装置が必要となる．

4.3.4　FIX（価格情報受信）

価格情報の FIX メッセージは表 3 のフォーマットで与えられる．L2，L3，L4 ヘッダ（54 バイト）を合わせたフレームデータを 256 バイト以内に収めるために，FIX メッセージの長さを 202 バイト以内に抑える必要がある．そこで各項目に文字数の上限を設定している．

この中で重要な項目は Symbol（55）（銘柄コード）と MDEntryPx（270）（価格）である．後者は Bid（買注文），Offer（売注文），Trade（取引）の 3 種類があるが，本研究の実装では FIX メッセージ中の 3 番目（最後）の MDEntryPx の項目が直近の Trade（取引）価格を意味する．本研究では取引価格のみを用いるため，「Symbol」及び「3 番目の MDEntryPx」の 2 項目を対象に回路を構成する．

4.3.5　発注クライアントアプリから渡されるデータ仕様

発注クライアントのソフトウェアからハードに送信すべき FIX メッセージ及び送信条件を渡すためのアドレス仕様が表 4 である．送信すべき FIX メッセージ及び送信対象銘柄，トリガー条件の価格と売買フラグをアプリからハードウェアに渡す．

4.3.6　FIX(新規注文送信)

表 5 に新規注文の FIX メッセージを示す．新規注文に関しては FIX クライアントのソフトウェアが生成して FPGA に渡すため，FIX メッセージを生成するための回路は存在しない．

表 4　アプリからの指令項目
Table 4　Directions from application

バイト長	内容	値
1	ID	F1
4	OrderID	30 31 30 31
4	銘柄名	4d 53 46 54
1	売 or 買	32(Sell)
5	トリガー価格	00 00 11 80 00
1	パディング	00
最大 202	FIX（新規注文）	(次項参照)

表 5　FIX メッセージ（注文情報）
Table 5　FIX message(New Order Single)

タグ番号	タグ名	値
8	BeginString	FIX.4.4
9	BodyLength	140
35	MsgType	D
34	MsgSeqNum	20
49	SenderCompID	CLIENT1
52	SendingTime	20180112-23:28:59.089
56	TargetCompID	SIMULATOR
1	Account	CLIENT1
11	ClOrdID	0102
21	HandlInst	1
38	OrderQty	2
40	OrdType	2
44	Price	1100
54	Side	2
55	Symbol	MSFT
59	TimeInForce	0
60	TransactTime	20180112-23:28:59
10	CheckSum	086

表 6　PL 部資源利用率
Table 6　Resource utilization ratio on PL

Resource	Utilization	Available	%
LUT	11650	53200	21.90
LUTRAM	125	17400	0.72
FF	18737	106400	17.61
BRAM	32	140	22.86

4.4　論理合成・配置配線結果

設計したアクセラレータ（図 6 の回路）を論理合成し，配置配線を行った．各資源の利用率を表 6 に示す．動作周波数 50MHz（20ns/clock）に対して Worst NegativeSlack(WNS) は 3.745ns(MET) であった．

4.5　ソフトウェアの部分的更新

Zynq の PL 部を利用するために，発注クライアントア

プリとネットワークのデバイスドライバを部分的に変更する必要がある．本節ではこのソフトウェアの変更に関して述べる．

4.5.1 発注クライアントアプリの修正 – BRAM へのアクセス

発注クライアントから BRAM にアプリ指令データを書き込むためにソフトウェアを書き換える．ユーザ空間において，C/C++では mmap システムコールを使い物理アドレスを unsigned char 型のポインタに直接マッピングすることができる．

4.5.2 発注クライアントアプリの修正 – FIX メッセージのアップデート

FIX メッセージの作成は発注クライアントアプリに委ねているが，FIX メッセージの中に MsgSeqNum（メッセージ通番）の項目があるために，アプリは一度作成した FIX メッセージを BRAM に書きこむだけでなく，そのメッセージが送信されるか取り消されるまで適宜更新し続ける必要がある．このため，BRAM 上に逆指値が設定されている場合は，FIX メッセージを送信する直前に BRAM の累積送信回数を読み，累積送信回数が+1 されていない（＝設定された逆指値が現在も有効である）場合，新しい MsgSeqNum の値で再度 FIX メッセージを作成し，BRAM を更新するように修正する．

4.5.3 ネットワークドライバの改造 – BRAM へのアクセス

本研究ではイーサネットコントローラが受信したデータを PL 部の BRAM に送り，FIX プロセッシングユニットからアクセスできるようにしている．また，Linux の TCP/IP プロトコルスタックが管理する通常の送信処理に加え，BRAM の指定領域に送信フレームが書き込まれた時にも送信するようにしている．この仕様を実現するためにデバイスドライバを修正する必要がある．本研究では，イーサネットコントローラのデバイスドライバ（xemacps）のソースコード（xilinx_emacps.c）を改造する．

まず，ドライバを初期化する時に呼び出される関数 xemacps_descriptor_init 内において，受信フレームの転送先アドレスを BRAM 内のアドレスに変更し，このアドレスを配列（xmap）にマッピングする．受信時のフレームの転送先メモリアドレスは DRAM に存在する受信バッファディスクリプタによって管理されており，イーサネットコントローラは（受信バッファでなく）受信バッファディスクリプタのリストのベースアドレスを保持している．そこで，受信バッファディスクリプタが参照しているアドレスを BRAM の受信バッファ領域の先頭アドレスに設定する．ドライバはカーネル内で起動しており mmap システムコールは使えないため，ioremap_nochache 関数を用いて BRAM のアドレスにマッピングする．

次に，Ethernet フレームを受信した時に発生するハードウェア割り込みの手続きの中で呼ばれる関数 xemacps_rx 内において，受信直後に FPGA が生成した注文の送信処理を行うように変更する．送信に関しても，送信バッファのアドレスは各送信バッファディスクリプタが管理しており，イーサネットコントローラは送信バッファディスクリプタリストのベースアドレスを保持している．

このようにネットワークドライバを改造することにより，イーサネットコントローラは Ethernet フレームの受信時に BRAM に DMA 転送する．その後の処理はドライバ内で行われ，ソフトウェアの関与はゼロではないが，xemacps_rx が呼ばれるハードウェア割り込みコンテキスト内で完結するため，ネットワーク処理に要する遅延の大部分が削減できる．

5. 評価

5.1 回路遅延

FPGA のデジタル回路で発生する遅延は「受信バッファに価格情報フレームデータが書き込まれた瞬間から送信バッファに新規注文フレームデータが書き込まれる瞬間までのクロックサイクル」に動作周期（動作周波数の逆数）を掛けて算出できる．

PL 部がデータを読んでから書き込むまでのクロックサイクルは基本的には毎回同一値であるが，FIX プロセッシングユニットはパケット処理時以外は常に READ 状態であり，BRAM に対して READ 命令を発行し続ける．一回の AXI READ 命令の開始から終了までに 15 クロックを要するため，その中のどのタイミングで受信バッファ書込み完了フラグがセットされるかによって必要なクロックサイクルに差異が生じる．

実際に 1 クロックずつ書込み完了フラグをセットするタイミングをずらして調べた結果，AXI READ に必要なクロックサイクル数は最小で 12，最大で 26 であった．（一様分布であることから平均は (12+26)/2=19 クロック．）また，発注すべきかを判定する JUDGE フェイズに要するクロックサイクル数は 5 であり，全データの BRAM への書き込みに必要なクロックサイクル数は 8 である．したがって，最小で 12+5+8=25，最大で 26+5+8=39，平均で 19+5+8=32 となる．動作周波数は 50MHz に設定されており，動作周期は 20ns である．したがって，回路遅延は平均で 32*20=640ns，最大で 39*20=780ns と算出できる．

5.2 トータル遅延の計測

通信の相手側が価格情報メッセージを送信してから新規注文メッセージを受信するまでのトータル遅延に関して，アクセラレータを用いた場合と，従来のソフトウェア処理により注文メッセージを送出した場合とを比較し，レイテンシ削減効果を確かめた．計測方法は，サーバ上で動作するパケットキャプチャツール（Wireshark）のタイムスタ

表 7　トータル遅延（μs）
Table 7　Total latency(μs)

統計量	本研究	PYNQ-Z1	Fedora20
サンプル数	52	53	51
平均値	148	1,573	684
中央値	139	1,276	680
最大値	254	6,911	794
最小値	89	1,161	577
標準偏差	41	1,119	56

ンプによるものであり，サーバが価格情報メッセージを送信した時刻と新規注文メッセージを受信した時刻の差として測定できる遅延である．

比較対象として，PYNQ-Z1 ボード以外に，実際の運用環境に近いものとして Linux（Fedora20）でのソフトウェア処理を計測した（表 7）[*3]．

平均値及び中央値を比較すると，本研究で設計したハードウェアアクセラレータを用いることにより，同じ PYNQ-Z1 ボードのソフトウェア処理の遅延の 9 割が削減され，Fedora マシン上のソフトウェア処理と比べても 8 割程度の遅延削減効果があることが確かめられる．この計測値にはサーバマシン内部で生じる遅延も含まれ，クライアントの責任領域で生ずる遅延の削減率に関する下限値と解釈できる．

5.3　考察

アプリケーションソフトウェアを用いた通常のネットワーク通信において，受信したフレームデータはアプリケーションのユーザロジックに届くまでにソフトウェアの階層を経るため，不可避的に遅延が発生する．具体的には，カーネルからユーザランドへの遷移，ユーザランドからカーネルへの遷移が最低 1 回は発生し，コンテキストスイッチのオーバヘッドが見込まれる．加えて，ソフトウェア割り込みで処理されるため，CPU 時間の割り当てに不確実性があり，他のプロセスの影響も受けやすい．

本研究の方法では一部にソフトウェアを介在させているが，カーネル内のハードウェア割り込みコンテキストでの処理に限られるため，ソフトウェアで発生する遅延の大部分を削減できる．

6.　おわりに

本研究では金融商品の低遅延取引の実現をテーマに，ネットワーク経由で発生するデータをリアルタイムで処理するためのハードウェアアクセラレータを設計した．SoC FPGA を利用し，OS の TCP/IP プロトコルスタックや FIX エンジンといったソフトウェア資産を利用しつつ，レイテンシにセンシティブな処理のみを FPGA 上のデジタル回路に配置した．具体的には，逆指値という最もシンプルな取引アルゴリズムを実装し，FPGA 内部で生じる遅延が最大 780ns となった．また，サーバ側で計測されるトータルの遅延に関して，同じ FPGA ボード上のソフトウェア処理と比較し 90%，Linux PC のソフトウェア処理と比較し 80%の遅延削減が確認できた．

[*3] Linux マシンのスペックは CPU:Core i7-3770 3.4GHz，Ethernet Controller:RealTek 8169，OS:Fedora 20 64bit，Kernel:3.11.10.

参考文献

[1] 高性能半導体「データフロープロセッサー(DFP)」，https://www.denso.com/jp/ja/news/media-center/press-kits/tms2017/TMS_DFP_JPN.pdf（2018 年 5 月 7 日閲覧）
[2] 祝迫得夫. 日本における高頻度取引（High Frequency Trading）の現状について．第 1 期 JSDA キャピタルマーケットフォーラム研究論文，pp.27–40, (2017).
[3] FIX Trading Community, http://www.fixtradingcommunity.org/, (2018 年 5 月 7 日閲覧)
[4] 金融庁「高速取引行為者向けの監督指針」(2018 年 7 月 28 日閲覧) https://www.fsa.go.jp/news/29/syouken/20171227-5-2.pdf
[5] Gareth W. Morris, David B. Thomas and Wayne Luk, *FPGA accelerated low-latency market data feed processing*, In 17th IEEE Symposium on High Performance Interconnects, (2009).
[6] Christian Leber, Benjamin Geib, Heiner Litz, *High Frequency Trading Acceleration using FPGAs*, In FPL, (2011).
[7] Robin Pottathuparambil et al., *Low-latency FPGA Based Financial Data Feed Handler*,. IEEE International Symposium on Field-Programmable Custom Computing Machines, (2011).
[8] John W. Lockwood et al., *A Low-Latency Library in FPGA Hardware for High-Frequency Trading (HFT)*, In 2012 IEEE 20th Annual Symposium on High-Performance Interconnects, (2012).
[9] *Finteligent Trading Technology Community (FTTC)*, Research Report: 10GbE Low Latency Networking Technology Review, (2012).
[10] ARGON DESIGN, ARISTA, fnteligent. Research Report: The Arista 7124FX Switch as a High Performance Trade Execution Platform, (2013).
[11] 井上浩明．特集，金融市場における最新情報技術 : FPGA による金融業務アクセラレーション – 複合イベント処理を題材に –．情報処理，vol. 53, no. 9, pp.921-926, (2012).
[12] 天野英晴編，FPGA の原理と構成，オーム社, (2016).
[13] QuickFIX, http://www.quickfixengine.org/ (2018 年 5 月 7 日閲覧)
[14] Louise H. Crockett et al., *The Zynq Book*, Strathclyde Academic Media, (2014).
[15] PYNQ:PYTHON PRODUCTIVITY FOR ZYNQ, http://www.pynq.io/ (2018 年 5 月 7 日閲覧)
[16] Xilinx Python productivity for Zynq(Pynq) Documentation Release 1.01, (2017).