Build Fast, Trade Fast: FPGA-based High-Frequency Trading using High-Level Synthesis

Andrew Boutros, Brett Grady, Mustafa Abbas and Paul Chow
Department of Electrical and Computer Engineering, University of Toronto
Email: {andrew.boutros, bgrady, mustafas.abbas}@mail.utoronto.ca, pc@eecg.toronto.edu

***Abstract*—High-Frequency Trading (HFT) systems require extremely low latency in response to market updates. This motivates the use of Field-Programmable Gate Arrays (FPGAs) to accelerate different system components such as the network stack, financial protocol parsing, order book handling and even custom trading algorithms. However, the long cycle of developing and verifying FPGA designs makes it challenging for HFT software developers to deploy such highly-dynamic systems, especially with their limited hardware design expertise. We present a complete highly-optimized infrastructure that implements low-latency system components in C++ using High-Level Synthesis (HLS). We also develop a framework that enables HFT algorithm developers to implement their trading algorithms in a high-level programming language and rapidly integrate it to the rest of the system. We implemented our HLS-based system on a Xilinx Kintex Ultrascale FPGA running at 156 MHz. Our on-board measurements show an end-to-end round-trip latency less than 870ns, which is comparable to that achieved by prior RTL-based implementations but requires reduced system development time and effort.**

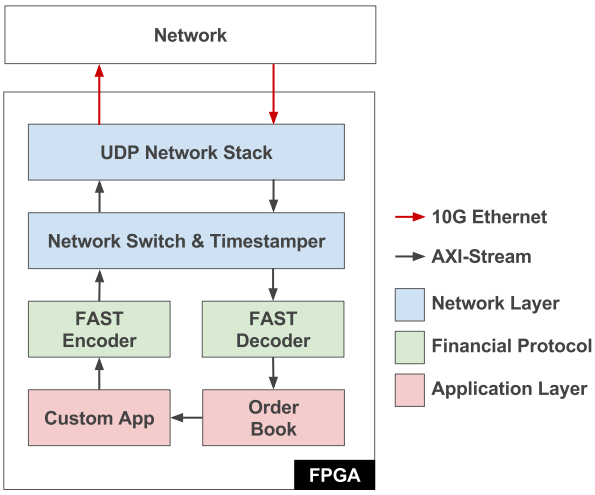

Fig. 1. HFT System block diagram

I. Introduction

Throughout the last decade, financial markets have witnessed a major change from the conventional human-based physical venues and floor-based trading into electronic-based automated trading controlled by sophisticated computer algorithms without any human intervention. A large portion of these automated trades are based on identifying and exploiting market *spread*; the transient differences between top asking and bidding prices of securities. The term *High-Frequency Trading* (HFT) emerged in the mid 2000s to refer to this type of automated trading systems [1]. According to a study carried out in 2010, the Securities and Exchange Commission estimated that the HFT volume in U.S. equity markets in the second half of the decade was greater than 50% of the total trading volume [2].

A typical HFT system consists of four main building blocks: network stack, financial protocol parsing, order book handling and custom application layer. Financial exchanges broadcast market updates along an Ethernet connection at typical line rates of 10 Gb/s [3]. The network stack receives the messages sent by the financial exchange and performs the initial packet processing. The packets are usually compressed in a domain-specific format to save on bandwidth; a prominent example is FAST (FIX Adapted for STreaming), which is an adaptation of FIX (Financial Information Exchange) [4]. The financial protocol parsing block changes the compressed packets into meaningful limited and market orders that are used to build the order book. The order book gives a view of the current market price by ordering bids (buying offers) and asks (selling offers) according to their prices with the highest bidding price and lowest asking price at the top of the book. Finally, the top bid and ask entries are used by the custom application layer to analyze the market and consequently issue buy/sell orders. These orders are then encoded using the same financial protocol and sent back over the network. The time interval between receiving incoming packets of an order into the system and sending out the triggered response packets is defined as the *time-to-trade* or the *round-trip latency*.

Due to the importance of having low-latency HFT systems, traders and hardware vendors have been in an arms race to lower the total round-trip latency. Typical high-end processor-based systems with specialized Network Interface Controllers (NICs) can react to market orders in a few microseconds [5]. But due to the need of further decreasing latency beyond that, designing application-specific hardware accelerators started to gain more attention in the HFT domain, especially FPGA-based accelerators due to their flexibility and re-programmability. FPGA-based systems proved to achieve far lower latency, approaching a four-fold reduction compared to conventional NIC solutions, often with more deterministic response times [6].

However, despite the performance advantages of this approach, there remains a concern about the development time of such systems. This is due to the longer cycle of designing

978-1-5386-3797-5/17/$31.00 ©2017 IEEE

and verifying FPGA systems as well as the rapid change in the implemented trading algorithms. In addition, HFT algorithm developers usually have different skill sets including high-level software programming and machine learning algorithms design that have minimal overlap with the hardware design skills required for this task.

Our work uses advancing FPGA High Level Synthesis (HLS) tools to prove that it is an attractive solution even for low-latency applications such as HFT systems. Offering an easier design and verification flow, this approach can be used by the more software-oriented HFT algorithm developers to achieve comparable latency to that of RTL designs and faster turn-around times. The contributions of this paper can be summarized as follows:

- A complete end-to-end HFT system, shown in Fig. 1, that achieves a round-trip latency less than 869ns measured using on-board time-stamping and monitoring that demonstrates the feasibility of HLS as a competitive solution to custom RTL designs in [3], [6].
- A novel HLS implementation for the order book handling IP core that achieves high-throughput and low-latency insertion, deletion and update operations of 80 ns.
- Demonstrating how HFT algorithm developers can implement their custom trading algorithms in high-level programming languages as C/C++, then use it to generate a hardware IP core and integrate it into our modular streaming infrastructure to generate a complete standalone system with minimal hardware design expertise.

II. Prior Work

As previously mentioned, HFT systems impose very tight constraints on latency, which favors the use of application-specific hardware to minimize the overhead of any unnecessary components when compared to conventional general-purpose CPUs. However, HFT algorithms and financial protocols have numerous variations and are dynamically changing in a way that makes it impossible to use Application-Specific Integrated Circuits (ASICs) in this domain. As a result, FPGAs are regarded as a very attractive solution for accelerating HFT systems due to their flexibility in designing application-specific hardware and reconfigurability in accounting for changes in implemented algorithms.

Most of the prior works in this domain focus on accelerating only parts of the system while having a software-based financial trading algorithm implemented on a host CPU. The reason for this is the large design effort and time required to build the hardware for those rapidly changing algorithms. It also requires hardware design expertise, which is not common among HFT software developers.

As an example for such systems, Leber et al. [6] proposed offloading the network stack as well as implementing a multiple-stream decoding of incoming financial packets on a Virtex-4 FPGA. The decoded packets are then sent to the host server for software-based book handling and processing. They implemented their system in Verilog code and compared it to a top-of-the-line server with a standard Network Interface Controller (NIC) achieving a one-way latency of 2.6 μs which is four times less when compared to the standard NIC solution. Similarly, in [7], the authors presented a library of different financial parsing and packet processing IP cores implemented in HLS. Their system also offloads the decoded market data to the host CPU using the PCIe interface for further processing with a latency of 0.5-1.3 μs.

TABLE I
Summary of Prior Work

Work	FPGA Blocks [a]			Latency (μs)	Design Method	Financial Algorithm [b]
	NT	FP	OB			
[6]	✓	✓	×	2.6	RTL	SW
[8]	×	×	✓	0.25	RTL	–
[9]	×	×	✓	0.4 - 0.7	MaxJ	–
[7]	✓	✓	×	0.5 - 1.3	HLS	SW
[3]	✓	✓	✓	1 [c]	RTL	HW (HDL)
Our Work	✓	✓	✓	0.87	HLS	HW (C++)

[a] Networking (NT), Financial Parsing (FP), Order Book Handling (OB)
[b] Software-based (SW), Hardware-based (HW)
[c] Round-trip latency does not include order book handling.

Another category of prior work focuses on the acceleration of order book handling on the traders' side building the order book from a market feed broadcast such as [8]. They implement an architecture that uses Cuckoo hashing in VHDL and use external SRAM modules to store the book for 119,275 instruments with a latency of 253 ns. On the financial exchanges' side, order book handling requires dealing with larger amounts of data. For instance, Fu et al. [9] use a CPU-FPGA hybrid table that caches MBs of most accessed data on-chip and GBs of data on the host CPU memory with synchronization support at a latency of 400-700 ns.

As an attempt to facilitate the deployment of FPGA-based HFT systems, Lockwood et al. [3] introduced a library of RTL-based IP cores to handle networking, protocol parsing and order book handling so that users can focus on developing the hardware for their custom trading algorithms. The system they implemented on a Virtex-5 FPGA using the IPs from this library achieved an end-to-end latency of 1 μs for the network stack and financial protocol parsing without including the order book handling or the application layer latency. However, using this library still requires extensive hardware design skills to both implement the financial trading algorithms in RTL and interface it to the rest of the system.

Table I summarizes prior works mentioned in this section in comparison to the work presented in this paper. In our work, we investigate the use of HLS to design a complete end-to-end HFT system and show that the latency results achieved are comparable to that of RTL designs. Not only does this allow HFT algorithm developers to implement their algorithms in C/C++ and easily transform them into optimized IP cores using simple pragmas, but also offers them the ability to integrate their IP cores with a complete off-the-shelf infrastructure for networking, financial protocol parsing and order book keeping blocks with minimal effort and hardware design expertise.

III. System Architecture

Our proposed system consists of three different layers as color-coded in Fig. 1: the Network Layer which contains UDP/IP and Ethernet layers as well as a network switch and timestamper block, a FAST protocol decoder/encoder for the Financial Protocol Parsing layer and finally the Application Layer which performs the order book handling and contains the custom trading algorithm as well. The modules of the different layers are arranged as a streaming pipeline using the AXI4-Stream bus protocol [10].

This section will present the functionality and architecture of each block followed by a demonstration of how HFT algorithm developers can take advantage of the presented infrastructure to develop complete end-to-end HFT systems with minimal design effort.

A. Network Layer

Typically, HFT systems are built on top of commodity networking protocols that uses one of the various Ethernet standards (i.e. IEEE 802.3ae-2002) as the physical layer, while the transport and internet layers are commonly UDP/IP [5], [3]. UDP is preferred over other alternatives because it is *connectionless*, fast to encode/decode, and relatively simple. Therefore, a UDP stream can be broadcast to many different destinations with no handshaking or re-transmission, and it is also a good fit for streaming hardware as almost no control flow or check-summing is required.

Since network acceleration on FPGAs is a mature topic and off the scope of this paper, we leverage off-the-shelf Ethernet and UDP/IP cores from Xilinx to implement the network stack [11]. We further added a simple custom network switch for system monitoring and priority based multiplexing of the channel as shown in Fig. 2. The network switch tags all incoming packets with a timestamp that is appended to the corresponding FAST message. Incoming market updates that trigger a trade have their timestamps appended to corresponding outgoing orders, which allows accurate benchmarking of the system's round-trip latency. The network switch also allows time-multiplexed access to the network prioritizing outgoing order packets over other system monitoring packets in case the FPGA is connected to a remote monitoring CPU over the network.

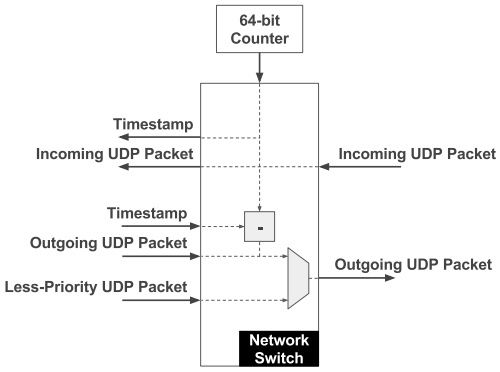

Fig. 2. Block diagram of our custom network switch

TABLE II
FAST Protocol Template Specifications

Field	Offset	Size (Bytes)	Field Description
Presence Map	0	1	Flags fields present in template
Template ID	1	1	Specifies template used
Price (Exponent)	2	1	Specifies the order price in base-10 floating-point format
Price (Mantissa)	3	1-5	
Order Size	4-8	1-2	Specifies size in units of tens
Order ID	9-13	1-5	Specifies a unique order ID
Order Type	14	1	Specifies the order type (Buy/Sell - Market/Limited)

B. Financial Protocol Parsing Layer

The FAST Protocol is a data compression algorithm developed for low-latency transfer of market data from financial exchanges to traders and the other way around [12]. Each FAST message contains multiple-field values that can define a market order such as price, order size, type, etc. The message is decoded by the receiver based on a pre-defined template that specifies the contents and locations of different fields.

Due to a lack of publicly available templates for exchanges, we created our own template for the FAST message fields based on [12] as detailed in Table II. We exploit the reconfigurability of FPGAs by building hardware that supports only the needed parts of the financial protocol according to the pre-defined template instead of sacrificing latency to build a generic encoder/decoder that handles all possible templates.

Fig. 3 shows the block diagram of the implemented FAST decoder and encoder. The decoder receives the FAST message as 64-bit chunks of data, concatenates them, and determines the offsets of each field by detecting the stop bits. Individual sub-decoders then operate on different fields in parallel, providing a scalable architecture similar to that in [4].

The encoder takes a triggered market order from the application layer and encodes all of its variables into the proper FAST-compliant data-types using the minimum number of bytes possible, concatenating them together with the required stop bits. The encoded blocks are then sent to the Network Switch, which bundles the data as UDP packets before sending it over the network.

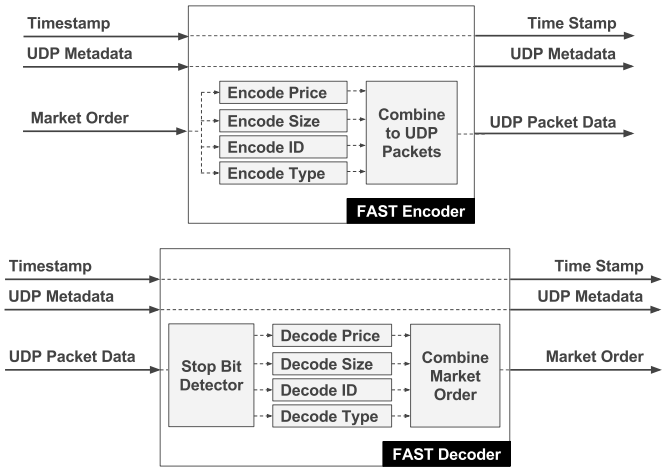

Fig. 3. Block diagram of our FAST decoder/encoder

C. Order Book Handling

We implemented an order book that can handle up to 2^{12} entries based on a heap-like structure with an efficient algorithm for insertion and deletion of nodes presented in [13]. A heap is a binary tree where each node is guaranteed to have less priority than its two children. In our implementation, each node of the heap represents a complete data structure containing the order price, size, ID and type. The priority of nodes is assigned according to the order's price.

Since our main concern is the insertion/deletion latency and we are only interested in the top bid and ask, we adopt a top-down heap structure. This allows us, in most cases, to write the top entry of the books to the output stream directly after the first comparison such that the custom trading algorithm can start execution while the rest of the heap is being sorted simultaneously. This also offers a scalable solution as the insertion and deletion times are not affected by increasing the size of the order book.

Algorithm 1 shows the basic insertion routine for the heap structure in our system where H is the array storing the nodes of the heap and H_o is the array storing indices of nodes that were previously removed from the heap. The insertion routine depends mainly on two observations: Firstly, if we represent the heap structure as an array of values stored in on-chip Block RAMs (BRAMs), for any node at index j, its left and right children are at index $2j$ and $2j+1$ respectively. Secondly, for any node at index j and level i, the path from the root of the heap to this node can be represented as the least significant i-bits of the value $j-2^i$ with 0 indicating *left* and 1 indicating *right*. The deletion routine works similarly but will not be described in detail in this paper for brevity.

Unlike conventional heaps, order book handling requires more features beyond simple insertion and deletion such as: arbitrary delete, modify top and multiple-node delete.

Input: *Heap Array (H), Holes Array (H_o), Input Order (O)*
if *(numItems < heapSize)* **then**
  if *(numHoles == 0)* **then**
    numItems++
    j ← numItems
  else
    numItems++
    numHoles−−
    j ← H_o[numHoles]
  end
  i ← log_2(j)
  path ← j-2^i
  while *(currentNodeIndex ≠ j)* **do**
    currentNodeIndex ← getNextNode(path)
    if *(O.price > H[currentNodeIndex].price)* **then**
      swap(H[currentNodeIndex], O)
    end
  end
end

Algorithm 1: Insertion routine for heap-based order book

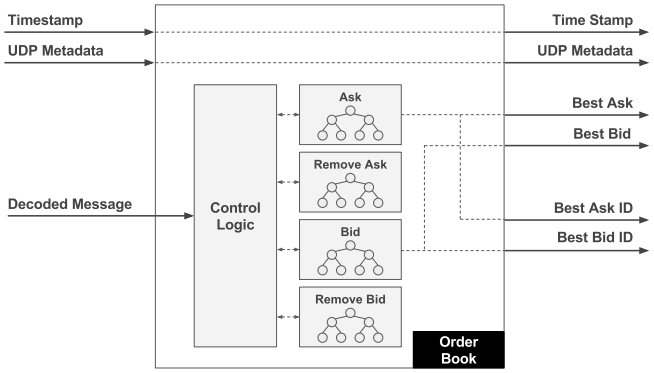

Fig. 4. Block diagram of our order book handling block

1) Arbitrary Delete: The ability to delete any arbitrary node from the heap is necessary in case an order is timed out or is withdrawn from the market regardless of its position in the order book. Scanning the heap to find and delete a specific node is an $O(N)$ operation, where N is the size of the heap, which is practically not affordable in such low-latency applications as HFT. Another solution is to have a hash map that links the order ID to its current position in the heap. However, this would be an inefficient use of on-chip memory and requires updates each time the order changes its position within the order book. Instead, we implemented this feature by storing all the incoming arbitrary delete orders in a separate heap structure that keeps track of the to-be-deleted order with highest priority and each time the top of the original heap is changed, if it matches the top to-be-deleted order, they are both removed.

2) Modify Top: If a market order of size S is received, it means that the best S units are to be removed from the order book. However if S is less than the size of the current best bid/ask (S_{best}), this requires not removing the best bid/ask order but only to modify its size to $S_{best} - S$, which is implemented by adding extra logic to check the size of the incoming market order compared to that of the top order.

3) Multiple-Node Delete: This feature is necessary if the size of the received market order S is more than the size of the current best bid/ask S_{best}. This means that the incoming market order will lead to the removal of more than one order from the order book. This is performed iteratively by subtracting the size of the deleted best order from the required size of the market order until it reaches zero.

The order book is tested to support those features as well as any combinations of them that can result from a complex scenario of received orders. As shown in Fig. 4, the order book takes an input stream of incoming decoded orders along with their time stamps. It outputs the current top bid and ask orders, once ready, as streams to the custom trading algorithm. It also sends the top bid and ask order IDs to a MicroBlaze core instantiated on the programmable fabric for monitoring and debugging the system through two AXI-Lite ports.

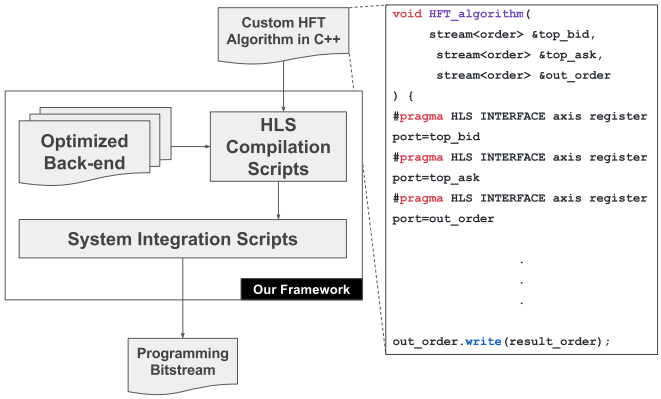

Fig. 5. System Integration Flow

D. Integration with Custom HFT Algorithms

Fig. 5 illustrates the system integration flow that we implemented by developing a set of scripts that automates the HLS compilation of the back-end IP cores for the network, financial protocol parsing and order book handling along with the user provided custom HFT algorithm. After that, system integration is automated to instantiate all the IP cores and handle the interfaces between them to produce a programming bitstream that is used to implement the system on the target FPGA.

Implemented completely using HLS with streaming connections and interfaces, the provided back-end IPs along with the system integration flow allow HFT algorithm developers to write their trading algorithms ranging from simple online algorithms [14] to more complicated financial Monte-Carlo simulations [15], [16] in C++ using a template as shown in Fig. 5. Then, with minimal effort and hardware design expertise, they can optimize their code with HLS pragmas to get a complete end-to-end FPGA-based HFT system.

Although developing a custom trading algorithm is out of the scope of this work, we implemented a simple algorithm to test our system's functionality, integration flow and accurately benchmark the round-trip latency. The implemented algorithm triggers a trade whenever the top bid or ask price crosses a specified threshold.

IV. Experimental Results

All system blocks were implemented in Vivado HLS 2016.3 and integrated using Vivado 2016.3. The target FPGA used in our experiments was a Xilinx Kintex Ultrascale XCKU115-2-FLVA1517E on the Alpha Data 8K5 board. Our system runs at 156.2 MHz frequency to match the 10G Ethernet bit rate (64-bit packet per cycle) and avoid clock domain crossing.

TABLE III
FPGA Resources Utilization Report

Resource	Used	Available	Utilization %
LUT	49,638	663,360	7.48
LUTRAM	2,148	29,3760	0.73
FF	32,718	1,326,720	2.47
BRAM	474	2,160	21.92
DSP	0	5,520	0

TABLE IV
FPGA Timing Results

System	Frequency	Round-trip Cycles	Latency
HFT Subsystem	156 MHz	42 cycles	269ns
HFT and Network	156 MHz	136 cycles	869ns

TABLE V
Latency Breakdown

Module	Latency (Cycles)	Latency (ns)
Network Receive	47	300
Network Transmit	47	300
Network Switch	12	77
FAST Encoder & Decoder	18	115
Order Book & App	12	77
Total	**136**	**869**

A. Resources Utilization

The utilization of different resources is shown in Table III. Our system uses less than 8% of the logic blocks and no DSP blocks which leaves the majority of the FPGA resources available for implementing complex HFT algorithms. The BRAM utilization is around 22% which is mostly due to the extensive partitioning of the order book storage to ensure high throughput. Some of these partitions are only a few records in size, but must be mapped to an entire BRAM on the FPGA. This issue was exacerbated by limited flexibility in HLS memory partitioning directives. Relaxing throughput constraints would allow for more compact partitioning schemes that use less BRAM. We suspect that in practice lower throughput is acceptable, however our design demonstrates that there is no hard performance ceiling with our approach. For future work offloading table entries of less priorities to off chip DRAM, similar to that in [9], would also reduce BRAM utilization. Both [3] and [6] do not disclose the resource utilization of their implementation for comparison.

B. Timing Results

To test the round-trip latency of the system, we developed a Python script that runs on the host CPU and sends 10,000 orders of different types to the FPGA over the network and later receives the system's response packets. The timestamps embedded in outgoing packets by the network switch module were aggregated and averaged to determine the number of cycles. Recalling Section III.A, this timestamp is effectively the delta from a trade-triggering order being received, till the corresponding response reaches the network stack. We achieved an average round-trip latency of 42 cycles (270ns) with best and worst case cycle counts of 36 and 62 respectively. This variability in the round-trip latency is due to the different insertion and removal scenarios that can occur in the order book. Adding the additional *worst case* latency need for transmitting and receiving through the Xilinx UDP and

TABLE VI
LATENCY COMPARISON TO PRIOR WORKS

System	Platform	Freq. (MHz)	Latency (μs)		
			Network Stack	Financial Parsing	Order Book
[6]	Virtex-4 FX100	125	2	0.6	–
[3]	Virtex-5 TX240T	156	0.8	0.2	–
[7]	Kintex-7 325T	156	0.8	0.5 - 1.3	–
Ours	Kintex-U 1517E	156	0.6	0.12	0.08

Ethernet layers (2×300ns as reported by [11]) yields a total system round-trip latency of 869ns as summarized in Table IV.

To gain more insight from the timing results, we performed a latency breakdown among different system modules by implementing variations of the original system to loop-back the received packets after the network and the financial protocol parsing layers. Table V shows the number of cycles consumed by each module as well as the cumulative latency that adds up to the total latency. The order book and network switch latency appear higher than expected from the modules alone due to the overhead of the AXI Stream interconnect used to interface all of the modules.

In Table VI, we compare the latency of different system modules to other prior work. Although prior works are not implemented on the same target FPGA, they are all operating at the same 156 MHz frequency to match the 10G Ethernet bitrate. The work from [6] operates at a lower frequency of 125 MHz which, if scaled to 156 MHz frequency, achieves latency of 1.6 μs and 0.48 μs for the network stack and the financial parsing protocol respectively. The financial protocol parsing layer in our work achieves 1.6 to 10 times lower latency due to implementing only the required part of the protocol instead of a generic encoder/decoder that handles all the protocol variations. This is encouraged by the fact that whenever a different protocol variation is used, the HLS C++ source code of the encoder/decoder can be modified to add this new change or remove any no longer needed components.

Despite implementing order book handling in [3], they did not include its latency in their published results so we were unable to compare it to our latency results. The comparison presented in Table VI aims to show that the HLS-based implementation in our work offers easy system deployment and integration for HFT algorithm developers in addition to achieving a round-trip latency that is in the ballpark of previous RTL designs.

V. CONCLUSION

This paper presented an HLS implementation of a complete end-to-end standalone HFT system implemented on a Xilinx Kintex Ultrascale FPGA. The system consists of the network stack, financial protocol parsing based on the FAST protocol, as well as order book handling to provide an abstract view of the current market state. The system achieves a round-trip latency less than 870ns measured using on-board timestamping. The order book handling block is implemented based on an top-down heap-like structure that takes around 80ns on average for insertion and deletion operations. We also present a framework that allows the more software-oriented HFT algorithm developers to implement their custom trading algorithms in C++ and easily integrate it with our infrastructure. This enables rapid deployment of complete FPGA-based HFT systems with minimal hardware design expertise while achieving comparable round-trip latency to that of prior RTL-based designs.

ACKNOWLEDGMENT

The authors would like to thank Chris Madill from Arches Computing Systems and Dan Ly-Ma from University of Toronto for their insights and recommendation which were very useful for this work.

REFERENCES

[1] P. Gomber *et al.*, "High-Frequency Trading," 2011. Available at SSRN: https://ssrn.com/abstract=1858626.
[2] A. J. Menkveld, "High Frequency Trading and the New Market Makers," *Journal of Financial Markets*, vol. 16, no. 4, 2013.
[3] J. W. Lockwood *et al.*, "A Low-Latency Library in FPGA Hardware for High-Frequency Trading (HFT)," in *IEEE Symposium on High-Performance Interconnects*, 2012.
[4] H. Li *et al.*, "Fast Protocol Decoding in Parallel with FPGA hardware," in *IEEE International Conference on Computational Science and Engineering (CSE)*, 2014.
[5] H. Subramoni *et al.*, "Streaming, Low-Latency Communication in Online Trading Systems," in *IEEE International Symposium on Parallel Distributed Processing (IPDPSW)*, 2010.
[6] C. Leber *et al.*, "High Frequency Trading Acceleration using FPGAs," in *IEEE International Conference on Field Programmable Logic and Applications (FPL)*, 2011.
[7] Q. Tang *et al.*, "A Scalable Architecture for Low-Latency Market-Data Processing on FPGA," in *IEEE Symposium on Computers and Communication (ISCC)*, 2016.
[8] M. Dvorak and J. Korenek, "Low Latency Book Handling in FPGA for High Frequency Trading," in *International Symposium on Design and Diagnostics of Electronic Circuits Systems*, 2014.
[9] H. Fu *et al.*, "A Nanosecond Level Hybrid Table Design for Financial Market Data Generators," in *IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2017.
[10] Xilinx, "AXI4-Stream Infrastructure IP Suite (PG085)," Apr 2017.
[11] D. Sidler *et al.*, "Scalable 10Gbps TCP/IP Stack Architecture for Reconfigurable Hardware," in *IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2015.
[12] D. Rosenberg, "FAST Specification Version 1.1," 2006.
[13] R. V. Nageshwara and V. Kumar, "Concurrent Access of Priority Queues," *IEEE Transactions on Computers*, vol. 37, no. 12, 1988.
[14] J. Loveless, S. Stoikov, and R. Waeber, "Online Algorithms in High-Frequency Trading," *Communications of the ACM*, vol. 56, no. 10, 2013.
[15] X. Tian and K. Benkrid, "Design and Implementation of a High Performance Financial Monte-Carlo Simulation Engine on an FPGA Supercomputer," in *IEEE International Conference on Field-Programmable Technology (FPT)*, 2008.
[16] D. B. Thomas *et al.*, "Hardware Architectures for Monte-Carlo Based Financial Simulations," in *IEEE International Conference on Field Programmable Technology (FPT)*, 2006.