High Frequency Trading Acceleration using FPGAs

Christian Leber, Benjamin Geib, Heiner Litz
University of Heidelberg
68131 Mannheim, Germany
{christian.leber, benjamin.geib, heiner.litz}@ziti.uni-heidelberg.de

***Abstract*—This paper presents the design of an application specific hardware for accelerating High Frequency Trading applications. It is optimized to achieve the lowest possible latency for interpreting market data feeds and hence enable minimal round-trip times for executing electronic stock trades. The implementation described in this work enables hardware decoding of Ethernet, IP and UDP as well as of the FAST protocol which is a common protocol to transmit market feeds. For this purpose, we developed a microcode engine with a corresponding instruction set as well as a compiler which enables the flexibility to support a wide range of applied trading protocols. The complete system has been implemented in RTL code and evaluated on an FPGA. Our approach shows a 4x latency reduction in comparison to the conventional Software based approach.**

Keywords-FPGA, high frequency trading; low latency; FAST; FIX; Ethernet; UDP

I. INTRODUCTION

High Frequency Trading (HFT) has received a lot of attention over the past years and has become an increasingly important element of financial markets. The term HFT describes a set of techniques within electronic trading of stocks and derivatives, where a large number of orders are injected into the market at sub-millisecond round-trip execution times [1]. High frequency traders aim to end the trading day "flat" without holding any significant positions and utilize several strategies to generate revenue, by buying and selling stock at very high speed. In fact, studies show that a high frequency trader holds stock for only 22 seconds in average [2]. According to the Aite Group, the impact of HFT on the financial markets is substantial, accounting for more than 50% of all trades in 2010 on the US-equity market with a growth rate of 70 % in 2009 [3].

High frequency traders utilize a number of different strategies, including liquidity-providing strategies, statistical arbitrage strategies and liquidity detection strategies [2]. In liquidity-providing strategies, high frequency traders try to earn the bid-ask spread which represents the difference of what buyers are willing to pay and sellers are willing to accept for trading stock. High volatility and large bid-ask spreads can be turned into profits for the high frequency trader while in return he provides liquidity to the market and lowers the bid-ask spread for other participants, adopting the role of a market maker. Liquidity and low ask-bid spreads are desirable as they reduce trading costs and improve the informational efficiency of asset price [4]. Traders that employ arbitration strategies [5], on the other hand, try to correlate pricing information between related stocks or derivates and their underlying prices. Liquidity detection comprises strategies that seek to discover large orders by sending out small orders which can be leveraged by the traders. All strategies have in common that they require absolute lowest round-trip latencies as only the fastest HFT firm will be able to benefit from an existing opportunity.

Electronic trading of stocks is conducted by sending orders in electronic form to a stock exchange. Bid and ask orders are then matched by the exchange to execute a trade. Outstanding orders are made visible to the market participants through so-called feeds. A feed is a compressed or uncompressed real time data stream provided by an independent institution like the Options Price Reporting Authority (OPRA). A feed carries pricing information of stocks and is multicasted to the market participants using standardized protocols which are generally transmitted over UDP over Ethernet. The standard protocol that is applied is the Financial Information Exchange (FIX) protocol Adapted for Streaming (FAST) which is used by multiple stock exchanges to distribute their market data [18].

To enable minimal round-trip latencies, a HFT engine needs to be optimized on all levels. The required low latency connection to the feed handler can be achieved through collocation which allows servers to be deployed very close to the stock exchange. In addition, the feed needs to be internally distributed with minimum latency to the servers of the HFT firm. An efficient decoding of the UDP data stream as well as of the FAST protocol is mandatory. Finally, the decision to issue an order as well as its transmission needs to be carried out with lowest possible latency. To achieve these goals we present in this paper a novel HFT trading accelerator engine implemented in Field Programmable Gate Arrays (FPGAs). By using FPGAs we can offload UDP and FAST decoding tasks from the CPU to optimized hardware blocks. Our proposed system implements the complete processing stack except the decision making process in hardware including a highly flexible microcode engine to decode FAST messages. Our approach shows a significant latency reduction of more than 70% compared to the standard software solution while maintaining the flexibility to support new and modified exchange protocols with low efforts in contrast to an Application Specific Integrated Circuit (ASIC) solution.

II. BACKGROUND

The following paragraphs will provide background information about the basic concepts and the actual implementation of a common trading infrastructure. In addition, the basic properties of the FAST protocol will be presented to define the requirements of a discrete hardware implementation.

A. Trading Infrastructure

A typical trading infrastructure consists of different components which are controlled by independent entities. In principle, these are the stock exchange, the feed handler and the market participants as shown in Figure 1. The matching engine, the data center switch as well as the gateway server are controlled by the exchange, while the feed engine is provided by the feed handler. The member access switch and the trading servers are property of the market participant, in our case the HFT firm.

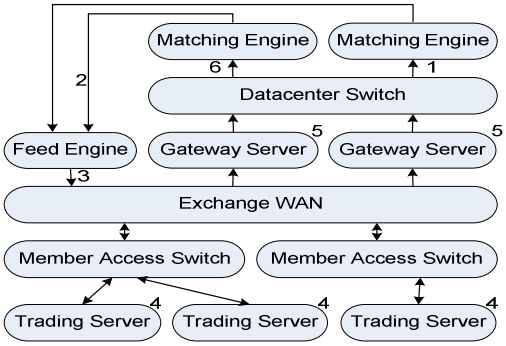

Figure 1. Trading Infrastructure

The following sequence describes how information is propagated in such a system and how orders are processed. (1) An opportunity is created at the matching engine. (2) The matching engine creates an update and sends it to the feed engine. (3) The feed engine multicasts it to all the clients. (4) The client machines evaluate the opportunity and respond with an order. (5) The gateway receives the order and forwards it to the matching engine. (6) The matching engine matches the *first* arriving order against the created opportunity and a trade is executed.

B. Protocol Stack

In current electronic trading deployments many protocol layers need to be traversed to be able to execute trades. Figure 2 illustrates these different layers.

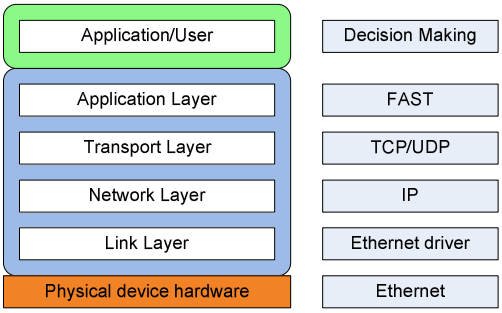

Figure 2. Protocol Stack

1) Ethernet Layer

The lowest layer of the protocol stack is represented by the Ethernet (ETH) layer. It provides the basic functionality for sending packets throughout a network. Therefore, it defines a framing and a Cyclic Redundancy Check (CRC) to ensure data integrity. In addition, ETH enables to identify endpoints within a network by defining Media Access Control (MAC) addresses both for the sender and the recipient of a packet.

2) IP Layer

The Layer above ETH is the Internet Protocol (IP) layer. This protocol is widely used and forms the first medium independent protocol layer. It groups computers into logical groups and assigns a unique address to every end node inside such a group. IP can also be used to send messages to multiple endpoints utilizing multicasts and is therefore used in most exchanges.

3) UDP Layer

The User Datagram Protocol (UDP) is used by application software to send messages between nodes. It offers multiple targets and sources among one node and ensures data integrity through a checksum.

4) FAST

The FAST protocol has been specified to transmit market data from exchanges to market participants using feeds. The protocol defines various fields and operators which are used to identify specific stocks and their pricing. An important aspect of FAST is its compression mechanism which reduces bandwidth, however, introduces significant CPU overhead. In fact, decoding of FAST represents a major bottleneck which makes it particular interesting for offloading to an FPGA. A more detailed description of the protocol will be given in the next section.

5) Decision Making

Decision making can be a very complex and resource consuming task depending on the applied algorithm. Essentially there are a variety of given parameters and incoming variables that are compared using mathematical and statistical approaches. A detailed analysis of the various algorithms that can be applied is out of the scope of this paper. Due to its complexity, the decision making process is not offloaded to the FPGA but kept in software.

C. Pitfalls of the FAST Protocol

The FAST protocol is applied by the feed handler to transfer pricing information to the market participants. To reduce the overhead, multiple FAST messages are encapsulated in one UDP frame. These messages do not contain any size information nor do they define a framing which aggravates decoding. Instead, each message is defined by a template which needs to be known in advance to be able to decode the stream. Most feed handlers define their own FAST protocol by providing independent template specifications. Care has to be taken as a single decoding mistake requires dropping the entire UDP frame. Templates define a set of fields, sequences and groups, where groups are a set of fields that can only occur once and sequences are a set of fields that can occur multiple times.

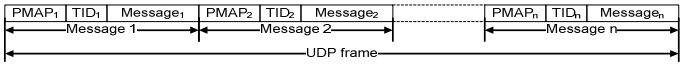

Figure 3. UDP Frame Including Multiple FAST Messages

Each message starts with a presence map (PMAP) and a template identifier (TID) as it is shown in Figure 3. The PMAP is a mask and used to specify which of the defined fields, sequences or groups are actually present in the current stream. Fields can either be mandatory or optional and can in addition have an *operator* assigned to it. It depends on the presence attribute (mandatory or optional) and the assigned operator if a field uses a bit in the PMAP. This adds additional complexity, as it has to be determined in advance whether the PMAP needs to be interpreted or not.

The TID is used to identify the template needed to decode the message. Templates are specified using XML; an example template definition is given below.

```
<template name="This_is_a_template" id="1">
  <uInt32 name="first_field"/>
  <group name="some_group" presence="optional">
    <sint64 name="second_field"/>
  </group>
  <string name="third_field" presence="optional"/>
</template>
```

In this example the TID is 1 and the template consists of two fields and one group. A field can be either a string, an integer, signed or unsigned, 32 or 64 bit wide, or a decimal, which is a set of a 32 bit wide signed integer for the exponent and 64 bit wide signed integer for the mantissa.

Only the bytes containing useful data are transmitted in order to save bandwidth. For example only 1 byte is transmitted for a sint64 (64bit signed integer) with value '1' even if the actual value is of course 64bit wide.

In addition, only the first seven bits of each transmitted byte are used to encode actual data, the eighth bit is used as a stop bit in order to be able to separate the fields. The stop bit needs to be removed and the remaining seven bits need to be shifted if a field is larger than one byte.

Consider the following incoming binary stream:

1̲0000111 0̲0101010 1̲0111111

These three bytes are two fields as it can be seen at the underlined stop bits. In order to receive the actual value of the first field it is sufficient to replace the eighth bit with a 0. The result is:

Binary value: 0̲0111111
Hex value: 0x63

The second field spans over two bytes. To get the actual transmitted value of this field, the first seven bits of each of the two bytes need to be moved together and padded with two 0 bits. The result is:

Binary value: 0̲0̲000011 10101010
Hex value: 0x03 0xAA

To make things even more complicated, fields also have to be decoded differently depending on their presence attribute. An optional integer for example needs to be decremented by one, a mandatory field however doesn't.

Operators trigger the execution of an operation after the field has been decoded or if the field is not present in the stream. The operators available are *constant*, *copy*, *default*, *delta* and *increment*. The *constant* operator for example defines that a field always has the same value and therefore will never be transmitted.

The FAST specification also defines byte vectors which do not use the stop bit encoding, but use all eight bits of a byte for data transmission and have a length field to define the length of the vector. This part of the specification has not been implemented due to its complexity and the fact that it is not used in any template of the target exchange, in our case Frankfurt.

III. Related Work

An increasing volume of work has recently appeared in literature, though probably the largest part of work done by institutions is not published. A very good overview about the acceleration of high frequency trading is given in [6]. Furthermore, a system for accelerating OPRA FAST feed decoding using Myrinet MX hardware is presented. A multi-threaded "faster FAST" processing engine using multiprocessor machines is presented in [7]. While both approaches focus on accelerating FAST decoding, to the best of our knowledge our approach is the first one that deploys FPGA hardware for this purpose. Morris [8] presented an FPGA assisted HFT engine that accelerates UDP/IP Stream handling similar to [9]. Another topic of interest are algorithmic trading techniques like the acceleration of Monte Carlo simulations [10][11][12] using FPGAs. Sadoghi [13] proposes an FPGA based mechanism for efficient event handling for algorithmic trading. Mittal proposes a FAST software decoder that is executed on a PowerPC 405 which is embedded in certain Xilinx FPGAs [13]. Finally, Tandon has presented "A Programmable Architecture for Real-time Derivative Trading" [14].

IV. Implementation

We propose three different approaches to reduce latency in HFT applications. The first is UDP offloading, while the second extends the hardware to enable direct decoding of FAST messages in hardware. The third approach introduces parallel hardware structures to enable simultaneous decoding of multiple streams.

A. UDP Offloading

In a common system, UDP data is received by a NIC in the form of raw ETH packets. The NIC then forwards the packets to the kernel which performs CRC checks and decoding of the packets. The activity flow of this mechanism is depicted in Figure 4. This approach introduces high latency as usually interrupts are used to inform the Operating System (OS) of new packets and as another translation layer in form of the socket interface to the user space is involved. Furthermore, OS jitter in the form of other activities conducted by the OS introduces

additional latency spikes. A detailed list of the different latencies that are involved for processing TCP/IP can be found in [15]. In the UDP case the latencies are very similar as UDP only differs in the upper layers from TCP.

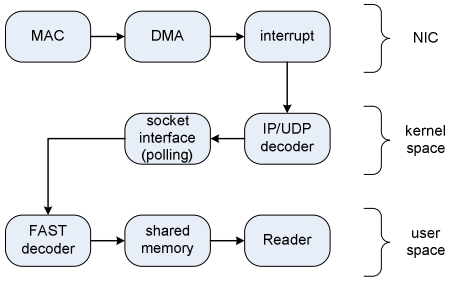

Figure 4. Activity Flow with a Standard NIC

The first and most efficient approach to reduce latency is to bypass the OS kernel and directly decode the received frames in hardware as it is depicted in Figure 5. Therefore, a rudimentary support for the Address Resolution Protocol (ARP) is necessary in order to be able to receive frames and send frames to the correct receiver. ARP is used to map IP addresses to physical MAC addresses of the recipient.

For receiving multicasts it is also necessary to support the Internet Group Management Protocol (IGMP), which is required to join and leave multicast groups. Other protocols that need to be supported are ETH, IP and UDP. All of these protocols are implemented in an efficient pipelined design, which focuses on lowest possible latency for ETH, IP and UDP. IGMP and ARP frames on the other hand are not timing critical for trading, since they are not used to actually deliver data. All header checksums of the different protocols and the ETH CRC are checked in parallel while the results of all these checks are interpreted at the last pipeline stage.

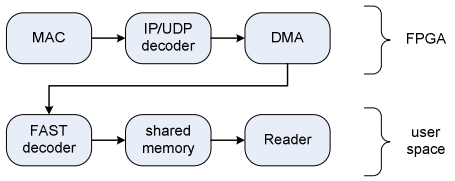

Figure 5. Activity Flow with UDP Offloading

The checked frame is then forwarded to the FAST decoder or the DMA engine, which is able to directly write the raw UDP payload in the address space of the trading software. In the case of FAST offloading shown in Figure 6 a similar DMA engine is utilized to deliver decoded packets into main memory.

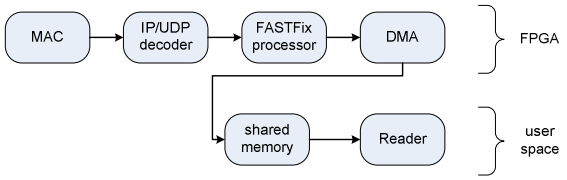

Figure 6. Activity Flow with FAST Decoding in Hardware

B. *FAST decoding in Hardware*

The FAST decoder is composed of three independent units due to the complexity of the FAST protocol. As it is shown in Figure 7, all three units are decoupled by FIFO buffers to compensate for different and sometimes non deterministic latencies of the units. As decompression increases the amount of data that needs to be processed, the FAST processor can be clocked at a higher frequency than the ETH core, which increases throughput and can therefore compensate the higher data volume.

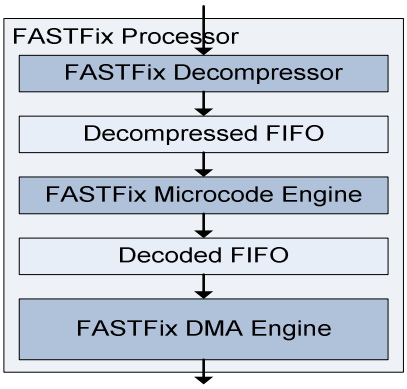

Figure 7. Schematic View of FAST Processor

1) *FAST Decompressor*

The FAST Decompressor detects stop bits and aligns all incoming fields to a multiple of 64 bit. This is done to have fixed size fields, which alleviates decoding for the following units.

2) *FAST Microcode Engine*

As FAST messages differ substantially depending on the applied template, it was chosen to develop a microcode engine which is flexible enough to decode any variation of the FAST protocol. Using partial reconfiguration instead of implementing microcode engine was dismissed due to the high reconfiguration latency of several ms.

The microcode engine runs a program that is loaded into the FPGA on startup with a subroutine for each template. A jump table provides the pointers to be able to jump to the right subroutine depending on the template ID that is defined at the start of each FAST message. All fields in the FAST messages are decoded according to the corresponding subroutine. Depending on the subroutine the content of the fields is either discarded or forwarded to the next unit in the pipeline. The binary code for the Microcode Engine is produced by an assembler from a simple domain specific language that has been designed for this purpose. The assembler code for the template presented in the introduction above would look like the following.

NOP		STORE_PRES	
SET_TID	0	STORE_TEMP	
NOP		JUMP_TEMP	
CON_U32_MAN	1	INCR_PC_DATA	
NOP		JUMP_PRES	1
CON_S64_MAN	2	INCR_PC_DATA	
CON_ASCII	3	INCR_PC_PM_DATA	

As can be seen, the proposed domain specific language defines four columns. The two left most columns describe the data value which is processed in that specific time step; while the two right most columns specify the command that shall be executed by the microcode engine. In particular, the first column defines the field with its presence attribute, while the second column maps the field to a unique identifier such that it can be later interpreted by software. The third column defines the control command, which increments the program pointer, jumps over some commands, shifts out data from the data FIFO or checks the PMAP. The last column is used to specify the jump target.

Using an assembler in collaboration with a microcode engine makes it easy to adapt the FPGA to template changes of an exchange or even adapt the FPGA to different exchanges without the need of developing a whole new design. This speeds up the adaption of the FPGA to protocol modifications significantly.

3) FAST DMA Engine:

To provide the trading software with the decoded FAST stream, a DMA engine has been developed. Each field in the different templates is tagged with an eight bit tag to allow efficient and unique identification by the trading software. The DMA engine forwards the received data into a ring buffer that resides in the address space of the trading software. Each write consists of eight quadwords, as this is the size of a cache-line on the utilized x86 CPU architecture. Seven of this quadwords are used for the content of fields; the eighth quadword contains the seven tags for the identification of the fields presented in column two and additional status information. The status information also incorporates a bit that is inverted every time the ring buffer wraps around. Using this bit the software can efficiently detect if new data has arrived without overwriting the ring buffer.

These implementations allow the lowest possible latency by allowing the software to make use of polling. Polling provides much lower latency than interrupts. The major reason for the high latency in the case of the using interrupts lies in the necessity of performing a context switch. The lower latency significantly outweighs the higher CPU overhead and increased power consumption that may be caused by this approach.

C. Parallelization

The FAST protocol itself is strictly linear because the beginning of each field in the stream depends on the prior field. Even after the FAST Decompressor has aligned the data it is still impossible to process a single stream of FAST messages in parallel. This is a limitation imposed by the protocol.

Fortunately the assembly of data provided by the stock exchange is consisting of multiple FAST streams. Therefore it is possible to improve the throughput and reduce the latency by implementing multiple FAST Processors in parallel. Each of the FAST processors is working on its own FAST stream. By deploying this technique the throughput of the system can be greatly improved. Each FAST Processor uses post place and route 5630 LUTs.

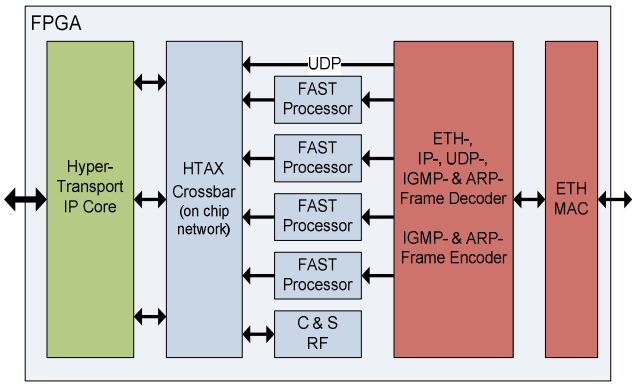

Figure 8. Schematic View of the HFT Accelerator

D. Host Interface

To further reduce the latency in the nanosecond scale the very low latency HyperTransport host interface, as offered by AMDs Opteron processor family, is used. HyperTransport offers much lower latencies compared to PCI Express due to the fact that the FPGA is directly connected to the CPU without intermediate bridging [16] using an HTX slot in AMD Opteron based systems.

E. Final Architecture

Our complete design has been implemented in synthesizable verilog RTL code and tested on an FPGA card based on a Xilinx Virtex-4 FX100.

Figure 8 illustrates the overall architecture of the accelerator. On the far left is the HT-core, which runs at 200 MHz. Followed by the HTAX on chip network connecting the Fast Processors, the UDP bypass and the Register File to the HT-core and the units itself. This part of the design runs at a maximum frequency of 160 MHz. The right hand side in red shows the packet decoding infrastructure connected to a Gigabit Ethernet MAC which runs at 125 MHz. The complete design, including four FAST Processors uses 77 percent of the available slices.

V. Evaluation

To perform our measurements, a recorded data stream provided by the feed handler has been sent to a standard network interface controller (NIC) in a top of the line server to measure the base line for our evaluation. Figure 9 shows the flow of the messages in such a standard system. Measured latencies for the NIC part is 9 us, for the kernel space 1.8 us and for the user space 2 us. This results in an aggregated latency of 12.8 us.

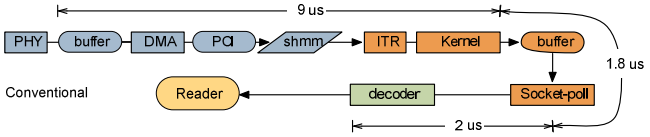

Figure 9. Latency of a the Baseline System

The same recorded stream has then been sent to the FPGA based trading accelerator. In the first run only the kernel bypass option has been enabled as it is shown in Figure 10. The measured latency of the NIC implemented on the FPGA is 2 us followed by a user space application with an additional latency of 2.1 us. Thus the latency already can be reduced to 4.1 us, which represents a 62 percent reduction in respect to the standard implementation.

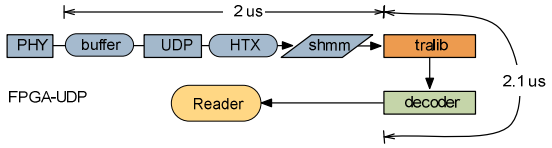

Figure 10. Latency with Kernel Bypass

In a second run the FAST Processor has been enabled, allowing to decode the messages in hardware and to deliver the decoded data directly to the user-level software as shown in Figure 11. The measured latency of the NIC and FAST decoding both on the FPGA shows a further reduction of 28 percent resulting in a latency of only 2.6 us, for reception and decoding of FAST packets. In total, this results in a 4 times latency reduction in respect to the standard NIC solution.

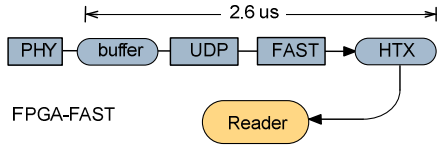

Figure 11. Latency with Kernel Bypass and FASTFix Offloading

VI. CONCLUSION

We have shown the implementation of an HFT accelerator which is able to significantly reduce the time needed to receive and decode FAST based trading information. In particular, the latency has been reduced by a factor of four which leads to substantially reduced round-trip times for executing trades.

The special nature of the FAST protocol which aggravates efficient parsing on a general purpose CPU makes it highly attractive to design FPGA based dedicated logic to decode it with significantly reduced latency. We have successfully shown the feasibility of this approach.

VII. REFERENCES

[1] J.A. Brogaard, "High Frequency Trading and its Impact on Market Quality," *5th Annual Conference on Empirical Legal Studies,* 2010.

[2] M. Chlistalla, "High-frequency trading Better than its reputation?," *Deutsche Bank research report*, 2011.

[3] A. Group, New World Order: The High Frequency Trading Community and Its Impact on Market Structure, 2009.

[4] K.H. Chung and Y. Kim, "Volatility, Market Structure, and the Bid-Ask Spread," *Asia-Pacific Journal of Financial Studies*, vol. 38, Feb. 2009.

[5] J. Chiu, D. Lukman, K. Modarresi, and A. Velayutham, "High-frequency trading," *Stanford University Research Report*, 2011.

[6] H. Subramoni, F. Petrini, V. Agarwal, and D. Pasetto, "Streaming, low-latency communication in on-line trading systems," *International Symposium on Parallel & Distributed Processing, Workshops (IPDPSW), 2010.*

[7] V. Agarwal, D. a Bader, L. Dan, L.-K. Liu, D. Pasetto, M. Perrone, and F. Petrini, "Faster FAST: multicore acceleration of streaming financial data," *Computer Science - Research and Development*, vol. 23, May. 2009.

[8] G.W. Morris, D.B. Thomas, and W. Luk, "FPGA Accelerated Low-Latency Market Data Feed Processing," *2009 17th IEEE Symposium on High Performance Interconnects*, Aug. 2009.

[9] F. Herrmann and G. Perin, "An UDP/IP Network Stack in FPGA," *Electronics, Circuits, and Systems (ICECS)*, 2009.

[10] G.L. Zhang, P.H.W. Leong, C.H. Ho, K.H. Tsoi, C.C.C. Cheung, D. Lee, R.C.C. Cheung, and W. Luk, "Reconfigurable acceleration for Monte Carlo based financial simulation," *IEEE International Conference on Field-Programmable Technology, 2005*.

[11] D.B. Thomas, "Acceleration of Financial Monte-Carlo Simulations using FPGAs," *Workshop on High Performance Computational Finance (WHPCF)*, 2010.

[12] N. a Woods and T. VanCourt, "FPGA acceleration of quasi-Monte Carlo in finance," *2008 International Conference on Field Programmable Logic and Applications*, 2008, pp. 335-340.

[13] M. Sadoghi, M. Labrecque, H. Singh, W. Shum, and H.-arno Jacobsen, "Efficient Event Processing through Reconfigurable Hardware for Algorithmic Trading," *Journal Proceedings of the VLDB Endowment,* 2010.

[14] S. Tandon, "A Programmable Architecture for Real-time Derivative Trading," *Master Thesis, University of Edinburgh*, 2003.

[15] S. Larsen and P. Sarangam, "Architectural Breakdown of End-to-End Latency in a TCP/IP Network," *International Journal of Parallel Programming, Springer*, 2009.

[16] D. Slogsnat, A. Giese, M. Nüssle, and U. Brüning, "An open-source HyperTransport core," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 1, Sep. 2008, pp. 1-21.

[17] G. Mittal, D.C Zaretsky, P. Banerjee, "Streaming implementation of a sequential decompression algorithm on an FPGA," *International Symposium on Field Programmable Gate Arrays – FPGA09*, 2009.

[18] FIX adapted for Streaming, www.fixprotocol.org/fast