17th IEEE Symposium on High Performance Interconnects

FPGA accelerated low-latency market data feed processing

Gareth W. Morris
Celoxica Inc.
Email: gareth.morris@celoxica.com

David B. Thomas and Wayne Luk
Imperial College London
Email: {dt10,wl}@doc.ic.ac.uk

***Abstract*—Modern financial exchanges provide updates to their members on the changing status of the market place, by providing streams of messages about events, called a market data feed. Markets are growing busier, and the data-rates of these feeds are already in the gigabit range, from which customers must extract and process messages with sub-millisecond latency. This paper presents an FPGA accelerated approach to market data feed processing, using an FPGA connected directly to the network to parse, optionally decompress, and filter the feed, and then to push the decoded messages directly into the memory of a general purpose processor. Such a solution offers flexibility, as the FPGA can be reconfigured for new data feed formats, and high throughput with low latency by eliminating the operating system's network stack. This approach is demonstrated using the Celoxica AMDC board, which accepts a pair of redundant data feeds over two gigabit Ethernet ports, parses and filters the data, then pushes relevant messages directly into system memory over the PCIe bus. Tests with an ORPA FAST data feed redistribution system show that the AMDC is able to process up to 3.5M messages per second, 12 times the current real-world rate, while the complete system rebroadcasts at least 99% of packets with a latency of less than 26us. The hardware portion of the design has a constant latency, irrespective of throughput, of 4us.**

I. Introduction

Modern financial instrument exchanges provide updates on the current state of the market place to their members. This is performed by transmitting messages describing change events on a dedicated 'market data feed'. These events include changes such as completed trades, bid/ask prices, and other status information. The event messages are typically aggregated into one large market data feed, containing information about all activities within an exchange or market. Automated trading systems can examine this feed in order to reconstruct the current market state for financial instruments of interest. This can then be used, for example, to perform algorithmic trading, detect instrument arbitrage opportunities, or to re-hedge portfolios. However, this feed is already in the gigabit range, and members face the problem of parsing this huge volume of data, while also supporting a sub-millisecond response to messages of interest.

Existing pure software solutions are no longer able to provide low latency solutions, so there is a need for hardware acceleration of market feed data processing. Field Programmable Gate Arrays (FPGAs) provide a very attractive means of acceleration, as they are a mature technology, with low power and space requirements [1]. However, FPGAs have traditionally been seen as difficult to program, requiring applications to be written in hardware-design languages, and needing specialised engineers to develop, maintain, and extend any FPGA-based solutions.

Application Specific Integrated Circuits (ASICs) may provide similar advantages to FPGA technology with respect to power and space. However ASICs have a far longer design cycle when compared to FPGA based solutions, and don't have the advantage of reconfigurability for resistance to updates in market data feed specifications. Furthermore the cost of an ASIC-based design would be considerably higher for the small to medium production runs needed to address the financial industry.

This paper proposes an FPGA-based hardware acceleration architecture for the processing of high-throughput market data feeds, providing a solution able to operate up to the maximum data-rate of the network connection, while offering a very low latency path from the network interface to the consuming process, irrespective of network load. Our key contributions are:

- A model for the division of market data feed processing, placing line-rate A-B line arbitrage, filtering, and routing in reconfigurable hardware, while keeping non-line-rate consuming processes in software for ease-of-use, and processing ability appropriate for trading algorithms.
- An implementation of this model using the Celoxica AMDC accelerator card, which is able to filter and process a redundant pair of market data feeds at gigabit line rates.
- An evaluation of the AMDC card using the FAST-compressed OPRA market data feed, demonstrating an average latency from packet arrival to message delivery of 4us, at throughputs limited only by the gigabit Ethernet connection.

We first explain the motivation and needs of market data feed processing, in particular the requirements for high message rates, and low-latency. Our high-level approach to FPGA acceleration of feed processing is then described, followed by a description of a concrete implementation of this approach on the AMDC board and its achieved performance.

II. Motivation

Financial market data feeds are used by exchanges to communicate changes in prices and market conditions to traders. For example, an option exchange will facilitate trades in a large

1550-4794/09 $26.00 © 2009 IEEE
DOI 10.1109/HOTI.2009.17

number of quoted options, each of which will have different properties, such as the underlying asset on which the option relies, the strike price of the option, and the expiry date of the option. For each instrument there will be a set of bid orders from traders wishing to buy at a certain price, and ask orders from traders who are willing to sell at a certain price. The job of the exchange is to collect and monitor these bid/ask prices, and match trades when the bid on a particular instrument meets or exceeds the ask price.

Information about changes in bid/ask levels and trades that have been completed is broadcast by the exchanges in real-time, allowing traders to reconstruct the market place, monitor market conditions, and respond when necessary. These automated trading systems may execute a number of strategies:

Algorithmic Trading: Many trading strategies can be expressed algorithmically, from the very simple "stop-loss" approach, up to complex heuristics to detect patterns in the market.

Instrument Arbitrage Detection: When an asset (or two similar assets) are priced differently in two markets, and this difference can be detected in time, a profit can be made by buying in the cheaper market and selling in the more expensive one.

Portfolio Hedging: It is possible to construct a portfolio with a known level of risk, by including specific sets of assets and derivatives. However, as market conditions change the composition of the portfolio must be adjusted to reflect changes in prices.

In almost all such applications, a key requirement is low latency processing, as the market is continually changing, so decisions should be made based on the most up to date data. For example, an opportunity for instrument arbitrage only exists until the first trader is able to exploit it, so it is critical to minimise the time between the information becoming available and the execution of the appropriate response.

Market data feeds intended for low latency trading are made available via UDP multicast IP networks, allowing many users to subscribe to a shared source. These feeds are often duplicated, with two streams (A and B) providing the same information, giving redundancy in the case of packet-losses between the exchange and trader. However, exchange members who wish to use the redundancy feature provided by the A-B feeds must process twice as much data. If the redundancy of the A-B feeds cannot be used, messages can be re-requested from the exchange, which then gives a significant latency disadvantage whilst waiting for the re-requested messages.

Over time the number of messages produced has drastically increased. Figure 1 shows the increase in messages rate for the OPRA data feed. This increase has a number of causes, such as:

Increased range of products: The number and scope of traded instruments have continually increased. For example, options are now quoted at strike price increments of one dollar, rather than five dollars, increasing the number of listed options by five times.

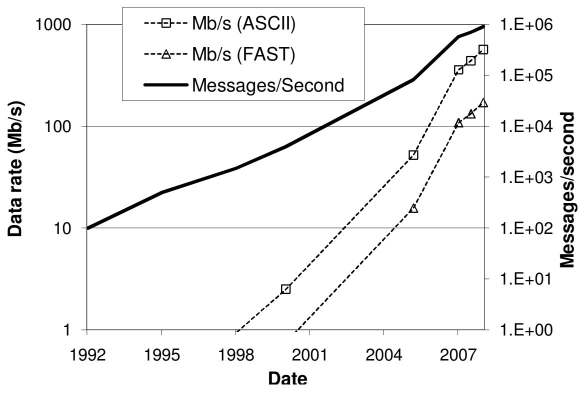

Fig. 1. Increase in the message and data rate for the OPRA data feed.

High resolution monitoring: Changes in prices are now reported to a resolution of one penny ($0.01), rather than a nickel ($0.05), resulting in much more frequent updates due to small price movements.

Algorithmic trading: Algorithms generate and cancel trades at a much higher rate than human traders, leading to an increased message rate from the exchange.

The result is a message rate approaching 1M messages/sec, a huge computational load for any potential data-stream client.

The growth in message rate has also caused the raw data-rate of feeds to increase in proportion. The data-rate required for the ASCII encoded OPRA stream has risen from less than 1 Mb/s to almost 600 Mb/s over the last ten years. Such data-rates place enormous strain on the networks between exchanges and traders, so data-streams are increasingly compressed, using domain-specific encoding schemes.

One such scheme, used on the OPRA feed amongst others, is FAST [2], a binary protocol which replaces the ASCII FIX format. To reduce message length, the format is no longer self-describing, so clients must already know the structure of the messages. A number of compression strategies are also used, such as:

- Variable-length encoding of integers.
- Delta-encoding of values against values transmitted in previous packets.
- Packing of data into seven bit atoms, using the eighth bit to indicate end-of-field.

This compression reduces the data-size by around two-thirds, significantly reducing pressure on networks. However, clients must now decompress the stream, increasing the amount of processing that must be performed before messages can be delivered to the applications that will consume them.

III. Approach

The combination of increased message rate and more complex market data feed formats mean that pure software solutions are often unable to keep up. Even when software solutions can deal with *average* message rates, they may start to develop a backlog during large bursts of trading activity, increasing latency in the situations where it is most important to stay up to date. Our proposal is to use a hardware accelerated stream processor, while still supporting software trading applications. Our key requirements are:

- High-bandwidth message processing, limited only by incoming network bandwidth.
- Minimal latency between arrival of packets at network interface and delivery of the contained packets to software.
- Support for redundant A-B input streams, with automatic recovery of missing messages.

A. *Overview*

Figure 2 gives a high-level overview of the problem to be solved. On the left is shown the market data source, within the exchange. This takes a stream of messages, which may be aggregated from sources within the exchange, and produces a single master stream. This stream may then be compressed, and sent to two network ports which multicast the streams out, via standard network infrastructure such as bridges and switches, to any interested parties. Eventually the two feeds arrive at the user's market data processor server, which contains the applications wishing to consume a particular subset of the messages contained in the feed.

The first step that must be taken in the data processor machine is to reconstruct the original feed - all networking equipment will occasionally lose packets, so the two redundant feeds are designed to minimise the effect of this. A-B line arbitrage requires any missing data from one stream to be replaced with correct data from the other stream. This is a non-trivial job, as even though the two streams left the data source machine at the same time, the intervening network infrastructure is likely to introduce skew. The line arbitrage should also make sure to select the earliest packet to arrive at the two interfaces, to minimise latency.

The second step is to decompress the reconstructed compressed stream, if necessary. This turns the packets into messages that can be consumed in software. Decompression requires intimate knowledge of the stream compression format and the stream payload, and must be updated as standards evolve. The final stage is to filter the stream, identifying which messages are of interest to which applications.

One approach is to use a multi-core processor for this task, assigning parallel threads to the jobs of A-B line arbitrage, decompression, and routing, but it has a number of drawbacks:

1) Software processing is reliant on OS calls to receive packets from network ports, but this introduces significant latency, due to the inefficiency of the OS networking stack, and because of the cost of context-switching between user- and kernel-space.
2) Co-operating threads must pass messages around either using OS level synchronisation primitives which require expensive context switches, or using spin-locks which lock up processors.
3) Many aspects of stream decompression are essentially sequential, so it is difficult to evenly schedule the processing load over multiple cores.

These factors combine to make a pure software solution unacceptable, in particular because there are so many sources of latency, and because the overall latency will vary randomly.

In contrast, we propose placing almost all stages of stream processing directly into hardware. All stages of data-stream handling, right up until the packet is delivered to the application code, is mapped into an FPGA accelerator. This is achieved by connecting the market data ingress ports directly to the FPGA, and by providing the FPGA with direct access to system RAM, allowing the accelerator to push data structures directly into the memory space of threads executing application code. Pushing data directly into RAM removes the need for any calls to OS routines, allowing application threads to detect new messages without any context switching, and so minimising latency.

The use of an FPGA as the main processing element means that incoming Ethernet traffic can be processed at line rate, at every tick of its mother clock. As data is processed at line rate, an FPGA solution can guarantee that no packets are dropped, irrespective of the link saturation.

B. *Stream Processing*

Each data feed arrives over the network as a sequence of distinct packets, with each packet prefixed with layers of headers providing information for different abstraction layers in the network. In the case of a market data feed, these headers start at the standard network layers, from Ethernet up to UDP, but also then extend into the financial protocol itself, where there are encapsulation formats, such as FAST, wrapped around specific payload formats such as different types of FIX packets.

Figure 3 provides an overview of the processing stack, from the lowest level (Ethernet) through to the feed specific payload messages. Because there are two data streams, A and B, the entire processing stack must be replicated, up until the point that the two streams are merged.

Our packet processing approach uses a set of composable packet processing components, which allows packet processors for new types of stream to be rapidly constructed. Packets are initially buffered as byte wide streams, which are transferred at one byte per cycle (shown in Figure 4). Header extraction components then use a shift register to extract the header for a given type of packet, producing a new stream with the same width as the header. This wide stream contains the entire parsed header in the first cycle, followed by the packet payload in successive cycles.

This method may appear inefficient, because headers are typically many bytes wide, but in practice, the header extraction components are connected directly to other components that will filter or route packets based on the header, so the wide stream only travels a short distance. Furthermore, unused fields will be optimised away from the design by the FPGA compilation tool. After filtering, only the payload is retained,

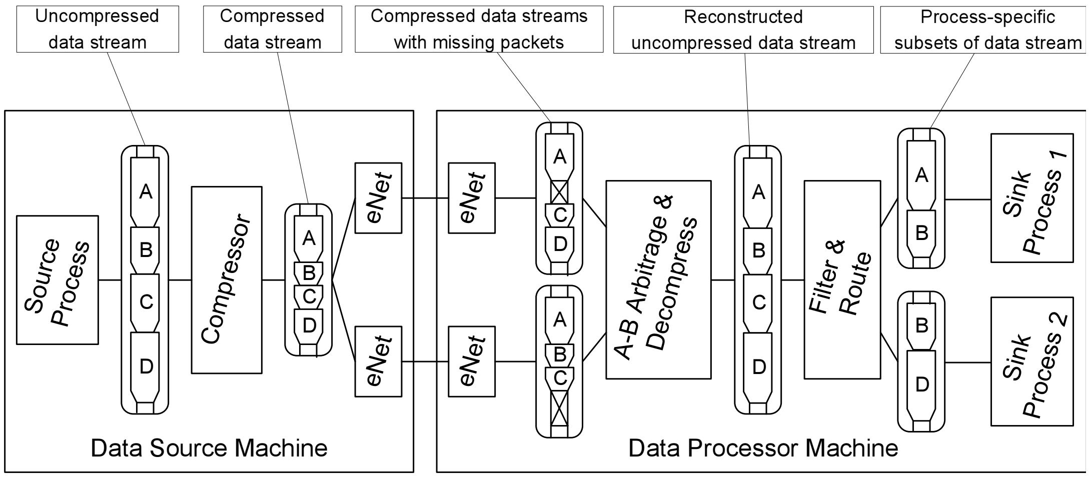

Fig. 2. Overview of stream processing problem

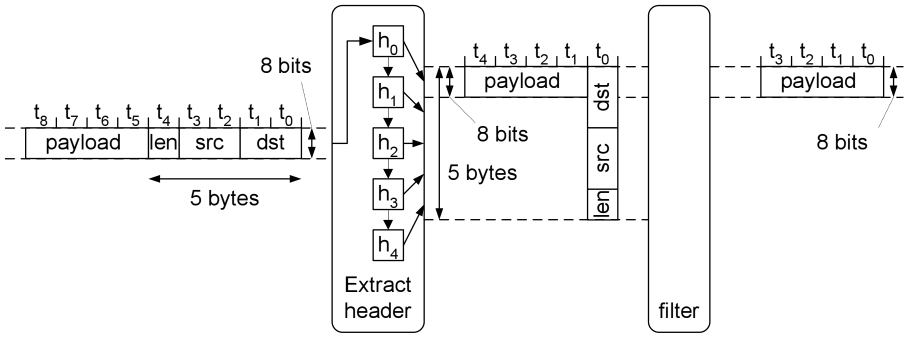

Fig. 4. Expansion of packet headers

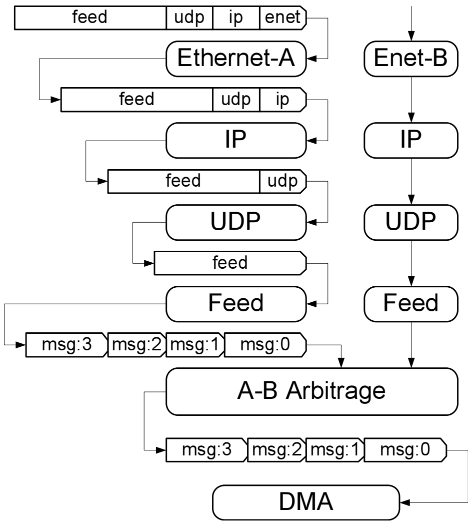

Fig. 3. Progressive filtering of packet headers

resulting in a byte wide stream that can then be efficiently buffered. Using separate header extraction components also has the advantage that multiple processing components can be connected directly to the header extractor, rather than each processing component extracting the header itself; it also promotes modular design and component reuse.

C. A-B Line Arbitrage

Since there is a possibility of packet loss between the exchange producing the market feed, and the member consuming the feed, two identical feeds of data are transmitted. The two streams, known as A and B, use the same message sequence identifiers, so it is possible to identify the same message as it arrives on each stream. In the case of a network error, a given message may never arrive on one of the feeds, but most likely it will still be delivered by the other stream. In very rare cases a message may not arrive on either stream, in which case the loss of the message must be indicated to the application.

In the majority of cases, the message will arrive on both the A and B stream, so in this case the message must be delivered as soon as it arrives on either stream, in order to minimise latency. However, the same message must not be delivered twice, as this would double the message rate seen by applications, and require them to use resources on book-keeping to check whether messages have been seen. This means that A-B line arbitrage (stream merging) must occur in hardware, after the raw stream has been parsed into distinct messages, but before they are filtered and routed to the applications via memory.

At the exchange, the messages sent over the A and B feed are sent at exactly the same time. However, the intervening network infrastructure between exchange and member will introduce delays of variable length, for example due to buffering in the internal buses of switches. For the same reason packets may also arrive out of sequence, either due to switching algorithms, or due to different packets taking different routes. The result is that the relationship between the A and B feeds will change over time, possibly on a per-packet basis, with sometimes one feed ahead, and sometimes the other.

To reconstruct the stream we maintain a window of message identifiers, which extends from the most recent message identifier we have seen, r, down to the bottom of the window $r - w + 1$. The parameter w is chosen to reflect the maximum observed transmission delay. Within this window we maintain a flag associated with each message identifier, indicating whether that message has been forwarded to the application yet. As each message arrives from the A or B channel, the associated flag is checked, and the message is forwarded or dropped depending on whether it has already been seen. When a message identifier i greater than r is encountered, the window is moved forwards – any non-zero flags in range removed from the window indicate completely lost messages, which are reported to the application.

D. Filtering and Delivery

The eventual consumers of messages are implemented as conventional software threads, executing on multi-processor host containing the FPGA board. This allows threads to be developed in conventional languages, such as C, rather than requiring esoteric hardware design languages. Threads interact with the hardware accelerator by first using a software API to specify a message filter, indicating which specific assets in the market they are interested in, and what types of messages about those assets they wish to receive. Once the message filter has been set, the threads continuously poll a message queue, waiting for any messages meeting the filter to be delivered.

Polling is traditionally seen as inefficient, but in this situation it is the best way of minimising latency. A traditional notification mechanism, such as an OS level mutex or event, incurs a context switch on the part of both notifier and notifyee, adding a potentially large and variable amount of latency. However, because messages are pushed directly into the memory space of the thread, a thread polling memory will receive the new packet with a latency determined only by the speed of the memory and cache-coherency protocol.

To further minimise the latency, it is necessary to lock each stream client thread to a specific CPU, and to make sure that it is the *only* thread on that CPU. This means that the OS never moves threads between processors, nor that a thread will be pre-empted by another thread. Modern multi-core processors offer many individual CPU cores, so it is an efficient trade-off. All non-feed processing threads, such as OS processes, are locked onto a single dedicated CPU.

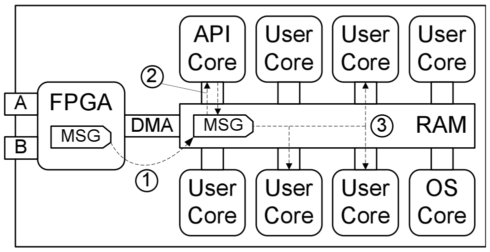

Fig. 5. Broadcast of packets to threads

As well as the stream client and OS cores, another core is dedicated to run-time management of the hardware accelerator. This is responsible for some aspects of routing, such as multicasting packets out to individual threads. Note that the run-time distributor only delivers pointers to structures that have already been placed in memory: messages do not need to be moved around once they have entered system memory.

Figure 4 gives an overview of the software processing of messages: first, the FPGA accelerator DMAs a copy of the entire message into shared memory; second, the run-time management thread detects the new message, and broadcasts a pointer to the message to all interested processing threads; and finally, each thread receives a pointer to the shared message, and can immediately start processing. Since user threads operate on lists of trading symbols, it is simple to process core usage, and balance them symmetrically.

IV. Implementation and Results

In order to verify the architecture, an implementation of the scheme proposed in Section III was carried out. The platform used for this implementation was the Celoxica AMDC accelerator card, which uses a Xilinx Virtex 5 LX110T FPGA device as the main processing element. The FPGA packet processing engine was written in Handel-C [3], using the Hyper-Streams programming model [4]. The AMDC card has been measured to draw less than 15watts of power from its host server.

The Celoxica AMDC card was inserted in a quad core 2.4GHz AMD Opteron server, running Redhat Enterprise Linux 5. The server was configured such that one CPU core was locked to polling the FPGA card for incoming messages, with another locked running a user packet redistribution application. The remaining cores were left unlocked for running OS related processes.

Figure 6 shows the harness used for these tests. Pre-recorded OPRA FAST v2 data is inserted into the system with the same inter-packet gaps as the original data. Further tests artificially accelerate the data transmission rate by reducing the inter-packet gaps by a constant factor. The original capture had an average of 296,177 messages per second over 60 seconds.

Data is streamed into the AMDC card inside the test server, as well as a separate packet sniffer. The Celoxica AMDC card processes the incoming feed, as discussed in Section III, and the data is made available to the user application. The user application repackages the data and broadcasts it out of the server's internal NIC card, using a standard OS socket call.

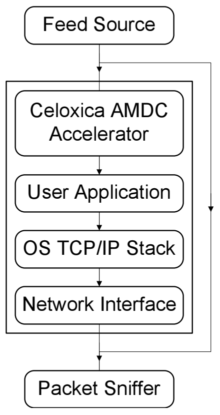

Fig. 6. Test harness for measuring overall system latency

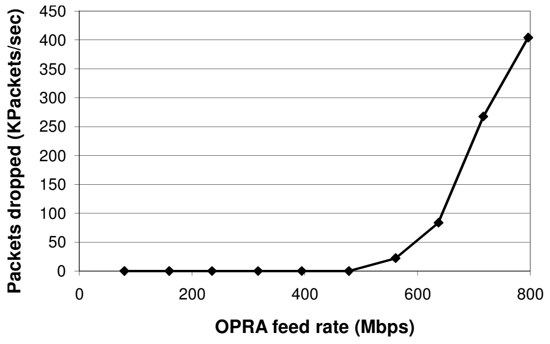

Fig. 7. Graph showing numbers of packets dropped in the software portion of the test environment with increasing throughput

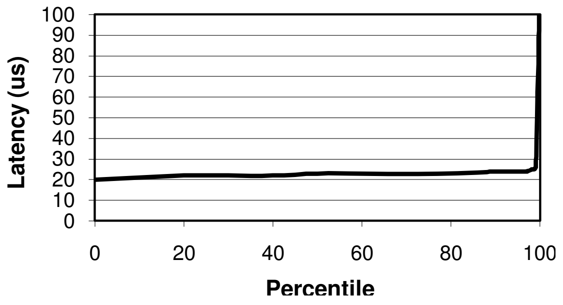

Fig. 8. Graph showing percentiles of packet processing latency

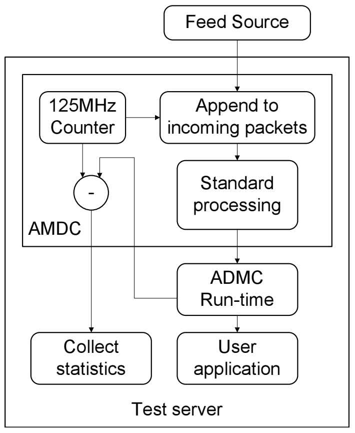

Fig. 9. Test-bench for measuring AMDC latency

On the output of the system, the packet sniffer also records data leaving the test server's NIC card. Comparing the difference in arrival time between packets from the feed source and test server allows latency across the entire test server to be measured. It also allows packets dropped in the test server to be recorded. In order to determine if these packets are dropped by the AMDC card, or software processes, the user application records packets dropped by the AMDC card.

As expected, AMDC did not drop packets until the point where line saturation is reached at around 450Mbps (3,554,119 messages per second, 12 times the original OPRA rate).

However, the combination of user application and OS network stack began to drop packets after six times the original data rate, at 1,777,059 messages per second. This is illustrated in Figure 7, as the throughput increases above six times the original data rate, an increasing number of packets are dropped.

Use of operating system networking stack routines on the output of the system skewed the latency tests. It is estimated that these routines add between 15 to 20us of extra latency, depending on load. Figure 8 shows 99% of packets have less than 26us of latency passing through the test server.

The results so far have concentrated on a complete redistribution system. Here we will consider the latency of the hardware portion of the design residing on the AMDC card, from the time a packet enters the FPGA on the AMDC card, until it becomes available in the memory space of the user application.

To gather these results, the test illustrated in Figure 9 was conducted. A counter running at the gigabit Ethernet mother clock of 125GHz (8ns resolution) runs on the FPGA in parallel with the packet processor. The tail of each packet entering the FPGA is appended with the current state of the 125MHz counter. The packet then passes through normal processing on the FPGA and is then handed over to the API running on one of the test server's CPU cores over the PCIe bus. When the API hands over a packet to the user application, it strips the packet's counter value and returns it to the FPGA over the PCIe bus. A separate module inside the FPGA subtracts this from the current counter value to calculate the latency, and accumulates the mean latency, which is reported at the end of the test. Notice that the latency of the PCIe bus is included twice in the measurement, one more than necessary. This latency has not been taken account of in the results, though doing so would obviously improve these results further.

The results of this test are reproduced in Figure 10. With increasing throughput, the latency of the hardware portion of the design remains roughly constant at around 4us.

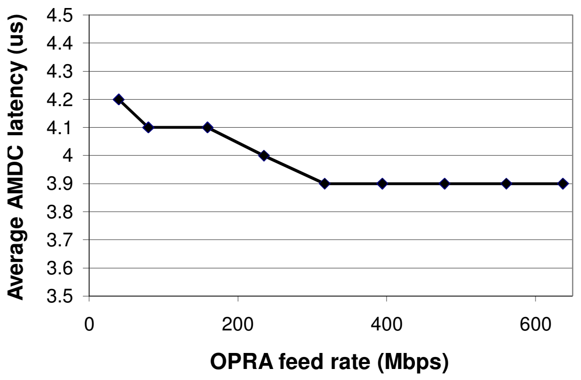

Fig. 10. Graph showing latency of the AMDC portion of the overall system latency with increasing throughput

V. Related Work

An alternative hardware-accelerated market data architecture is the Exegy Ticker Plant [5]. In this approach, incoming market data enters the system via a conventional Ethernet card. Processing is then augmented with an FPGA accelerator on the processor bus. This has the advantage of a higher throughput bus, although this isn't needed even for a saturated gigabit Ethernet link.

Another accelerated feed processing approach is provided by the ActivFeed MPU [6]. This uses an XtremeData FPGA accelerator which is placed in a processor socket of the host system, and communicates using HyperTransport. However, network integration is via an Infiniband bridge, rather than by direct connection to the Ethernet, and processing latencies are quoted as "end-to-end latency surpass[es] 100 us", compared to the 20 us latencies reported here.

The idea of packet processing using graphs of composable nodes has been used in a number of previous systems. In the Click system [7] a graph of processing nodes is defined using a C++ interface in software, which is then compiled into an FPGA design. The Net FPGA project (http://www.netfpga.org/) offers a library of packet processing components, which communicate using a common protocol, but requires the use to manually connect together each of the protocol wires between them. In contrast, the approach developed here allows the graph to be described directly in Handel-C [3], so no extra compiler passes are required. The connections between components are also specified as abstract data-paths – all protocol-specific details of the connection are hidden from the programmer.

VI. Conclusion

This paper presents a method that allows processing of market data feeds using FPGAs, providing the ability to process extremely large numbers of messages per second, while also minimising the latency between arrival of network packets and their delivery to their intended target in software. This is achieved by eliminating the operating system networking stack: all message processing and filtering is applied in an FPGA, which is then able to push messages directly into the memory space of software threads via FPGA-initiated DMA. As well as reducing latency due to the OS stack, this also reduces both the programming burden and performance over-head for software components, as messages are provided as fully decoded memory structures, rather than serialised messages which must be parsed.

This approach has been implemented in the Celoxica AMDC accelerator card, which incorporates two gigabit Ethernet ports and a Xilinx Virtex-5 LX110T FPGA, connected to a host computer over the PCIe bus. Tests performed using the OPRA-FAST compressed data feed format have shown that an AMDC accelerated system can support a message throughput of 5.5 million messages per second, 12 times the current real-world rate, while the complete system rebroadcasts at least 99% of packets with a latency of less than 26us. The hardware portion of the design has a constant latency, irrespective of throughput, of 4us.

Currently the proposed architecture only accelerates incoming market data. In the future accelerated Ethernet transmission will be examined. This will include Uni- and Multi-cast UDP offload, TCP/IP offload, and market order execution.

References

[1] T. J. Todman, G. A. Constantinides, S. J. E. Wilton, O. Mencer, W. Luk, and P. Y. K. Cheung, "Reconfigurable computing: architectures and design methods," *IEE Proc. Computing and Digital Techniques*, vol. 152, no. 2, pp. 193–207, 2004.

[2] K. Houstoun, *FIX Adapted for STreaming - FAST Protocol Technical Overview*, 2006.

[3] *Handel-C Language Reference*, http://www.celoxica.com, Celoxica Ltd., 1999.

[4] G. W. Morris and M. Aubury, "Design space exploration of the European option benchmark using Hyperstreams," in *FPL*, 2007, pp. 5–10.

[5] S. T. A. Center, "Exegy ticker plant with infiniband," STAC Report, July 2007.

[6] "Activefeed MPU: Accelerate your market data," http://www.activfinancial.com/docs/ActivFeedMPU.pdf, 2007.

[7] K. C, G. Brebner, and G. Schelle, "Mapping a domain specific language to a platform FPGA," in *Proc. IEEE Design Automation Conference*, 2004, pp. 924–927.