Low Latency FPGA Acceleration of Market Data Feed Arbitration

Stewart Denholm*, Hiroaki Inoue†, Takashi Takenaka†, Tobias Becker* and Wayne Luk*
*Department of Computing, Imperial College London, London, UK
†NEC Corporation, Kawasaki, Japan

***Abstract*—A critical source of information in automated trading is provided by market data feeds from financial exchanges. Two identical feeds, known as the A and B feeds, are used in reducing message loss. This paper presents a reconfigurable acceleration approach to A/B arbitration, operating at the network level, and supporting any messaging protocol. The key challenges are: providing efficient, low latency operations; supporting any market data protocol; and meeting the requirements of downstream applications. To facilitate a range of downstream applications, one windowing mode prioritising low latency, and three dynamically configurable windowing methods prioritising high reliability are provided. We implement a new low latency, high throughput architecture and compare the performance of the NASDAQ TotalView-ITCH, OPRA and ARCA market data feed protocols using a Xilinx Virtex-6 FPGA. The most resource intensive protocol, TotalView-ITCH, is also implemented in a Xilinx Virtex-5 FPGA within a network interface card. We offer latencies 10 times lower than an FPGA-based commercial design and 4.1 times lower than the hardware-accelerated IBM PowerEN processor, with throughputs more than double the required 10Gbps line rate.**

I. Introduction

Market data feeds are used by financial exchanges to provide updates about changes in the market. Messages are multicast to members and describe market events such as available and completed trades. Members, i.e., financial institutions, utilise these feeds in a number of applications to determine the current state of the market and the institution's risk, searching for time-critical arbitrage opportunities, and as triggers for trades within algorithmic trading platforms.

The latter two applications require time-sensitive decisions to be made based on the input data, often by analysing patterns within the data. This decision making is a critical element for electronic trading, so it is vital messages are received and presented in the correct order. Failure to do so will result in the loss of profit-generating opportunities, provide competitors with an advantage, and create a false image of the current state of the market, increasing risk.

Much work has been done demonstrating the benefit of hardware over software when performing this task. The general-purpose architectures of CPU systems separate data acquisition and data processing, leading to latency penalties when processing external data. This disparity provides an opportunity for hardware acceleration of time sensitive processing before the resultant data is passed to the CPU.

In this work we outline a low-latency hardware architecture able to perform market data feed arbitration for any given messaging protocol. Dynamically configurable windowing methods are provided so downstream applications can alter the arbitrator to respond to their realtime requirements. The critical path and performance of the arbitrator changes with the market protocol used and so provides an ideal platform for the exploration of high performance, low latency designs. The contributions of our work are:

- The first hardware accelerated, low latency A/B line arbitrator to run two packet windowing modes simultaneously. It supports three dynamic windowing methods, any market data protocol, independent memory allocation per input, and configurable data-path widths.
- Implementation of A/B line arbitration using the NASDAQ TotalView-ITCH [1], OPRA [2] and ARCA [3] market data feed protocols in a Xilinx Virtex-6 FPGA. TotalView-ITCH is also implemented on a Xilinx Virtex-5 FPGA within a network interface card.
- Evaluation and performance measurements for the TotalView-ITCH, OPRA and ARCA protocols.

In our previous work [4] we looked at fitting multiple basic A/B line arbitrators into a single FPGA, but with only a single arbitration mode we limited the range of downstream applications we could support. This is addressed in this work with the use of three high reliability modes, one of which can be output simultaneously with a new low latency focused windowing method.

II. Background and Motivation

Financial institutions subscribe to both the A and B feeds, traditionally arbitrating between the two feeds in software to provide a single message stream for processing by financial applications. The pipelined and parallel nature of this task is ideally suited to low latency hardware acceleration. However, A/B arbitration is rarely addressed, and its importance will only grow in the future as line rates increase and financial exchanges continue to process an ever growing number of messages. Since exchanges send multiple messages per packet using multicast UDP, any error during transmission will result in the loss of all packet messages. More packets processed every second means more messages bundled together into each packet, increasing its informational value and the chance that it will contain a bit error and be lost.

Mechanisms are in place to retransmit lost packets, but in the time taken to request and receive it, any time-sensitive opportunities will be lost. A/B line arbitration is the only method that can compensate for missing packets within an acceptable time frame. Allowing each application to set this time frame themselves is a key factor in its successful operation and our proposed design.

Morris [5] uses a Celoxica board to process financial messages, achieving a 3.5M messages per second throughput and hardware latency of $4\mu s$. Their trading platform is one of the few including line arbitration, but no details of its performance are given. It uses a single, simple windowing system similar to the high reliability count mode in this work and only supports the OPRA FAST format. The windowing thresholds are not discussed and cannot be changed.

Most stand-alone A/B arbitrators are commercial and their implementation details are usually not presented. They tend to operate within a network interface card (NIC) and communicate with the host via PCI Express.

One such arbitrator from Solarflare [6] uses an Altera Stratix V FPGA. It supports either a low latency mode or a maximum reliability mode; the latter being similar to the high reliability time & count mode in this work. Multiple message protocols are supported, but no processing latency figures are available. Another platform from Enyx [7], also using the Altera Stratix V, does not give any details regarding the windowing method used or possible configuration options. It is non-deterministic, with packet processing latencies ranging from $1050 - 3080ns$ based on 1500 byte packets. Some protocols, like TotalView-ITCH 4.1, specify 9000 byte packets must be supported, so it is unclear how this latency will scale with larger packets.

Recently, a number of FPGA based feed processors have been proposed. The majority do not mention A/B line arbitration, such as the OPRA FAST feed decoder from Leber [8], and the NASDAQ data feed handler by Pottathuparambil [9]. Other works describe, but do not implement, arbitration, like the high-frequency trading IP library from Lockwood [10]. This is a strange omission since line arbitration is an integral part of message feed processing as it increases the amount of available information and actively prevents message loss.

Platforms incorporating some aspects of feed processing and trading within an FPGA are limited in the range of functions they provide, making it difficult to customise desired features. The flexibility to support applications with different data requirements and different time scales is not present in past works. Single trading platforms are therefore unlikely to be deployed within financial organisations unless the design features exactly meet the needs of the organisation, including the market data feed protocol used.

III. Accelerating A/B Line Arbitration

Messages from financial exchanges are transmitted via identical A and B data streams. Due to network infrastructure differences, or errors during transmission, messages may fail to arrive, or may be reordered. By subscribing to both streams, members reduce the likelihood of message loss, but must now merge and order the two streams. This is facilitated by unique identifiers within each message, typically taking the form of an incrementing sequence number.

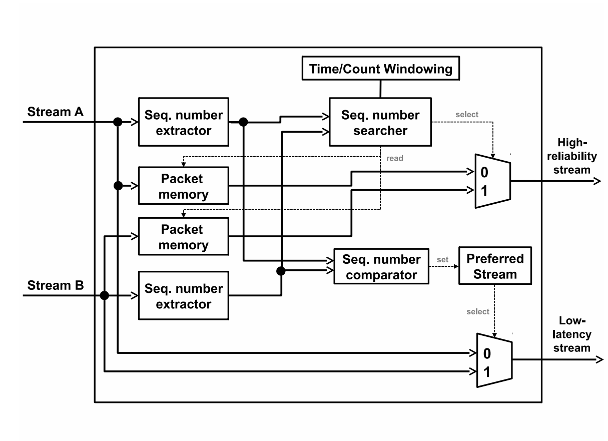

Fig. 1. The layout of our A/B line arbitration design.

Uncertainty regarding the presence and order of messages on the A and B streams give rise to four possibilities. A message may: (**1**) arrive on both streams; (**2**) be missing from one stream; (**3**) be missing from both streams; or (**4**) arrive out-of-order. For the first and second cases we should pass through the earliest message we encounter, and have no expectation of seeing it again. However, the third and fourth cases illustrate the need for an expectation regarding future messages. We require a centralised system to monitor the streams and share state information

It is important to distinguish between market data messages and packets. Exchanges send UDP packets containing one or more messages. This can be viewed as a continuous block of messages, all correctly ordered by sequence number, with no missing messages. When a packet is missing we are in fact dealing with a block of missing messages. This means packets, rather than messages, are the smallest unit of data we process and store.

Figure 1 gives our design layout, showing the high reliability and low latency modes. The windowing module supports three high reliability modes of operation, for which the windowing thresholds can be set at runtime. An operator or monitoring function can adjust these thresholds to meet application or data feed requirements.

A. High Reliability Modes

When we encounter a packet with a sequence number larger than the next expected sequence number, it has arrived out of order. The missing packet, or packets, may be late, or never arrive. A high reliability mode stores these early packets and waits for the missing packets, stalling the output.

We decide how long to wait for missing packets using a windowing system, based on either: the amount of time we have stalled the output, the number of messages delayed, or a hybrid of both time and message count. Within this window we store new packets while waiting for the missing packets. Whatever system used, we must ensure not to delay a valid, expected packet as this is the most likely case.

Count-based windowing is used by [5], time & count by [6], while [7] does not detail its windowing approach. This is the first work to: support all three methods, provide low

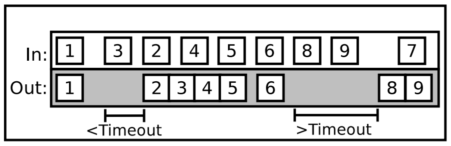

Fig. 2. High reliability time example.

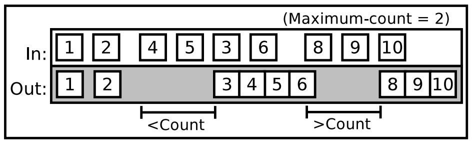

Fig. 3. High reliability count example.

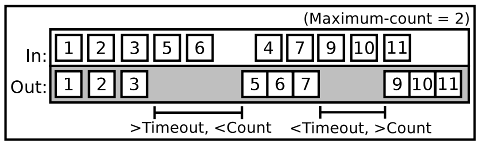

Fig. 4. High reliability time & count example.

latency, application-specific parametrisation, and output a high reliability and low latency stream simultaneously.

High reliability - time: A time-based windowing approach is good when we want to set a hard limit on possible delays and define the maximum processing time of packets.

When we delay a packet, P, we assign it a timeout value, T, the maximum number of clock cycles we will delay it. T is decremented each clock cycle, and when it reaches zero we discard any missing packets and output P. An example with a single input is given in Figure 2, where packet P_2 is late, but arrives before P_3's timeout reaches zero and is able to be output. P_7, however, is too late, so the delayed packets P_8 and P_9 are output, and P_7 is discarded.

Assigning the maximum timeout value to a packet then decrementing it is a more beneficial than incrementing from zero. The timeout check is then simply a zero equality check, and the number of remaining cycles may be used to predict this condition and pre-compute future data values, such as the expected number of buffered messages in the next cycle.

High reliability - count: Time-based windowing sets a packet timeout regardless of how many messages it contains. Counting delayed messages—not packets—more accurately represents processing delay, as the number of messages per packet varies during the day. This time-independent approach better matches the pace of incoming data.

We output a delayed packet when either: the missing packet or packets arrive, or the number of stored messages exceeds the maximum-count threshold. Two examples of this are shown in Figure 3's single input example. Packet P_3 arrives before we exceed maximum-count = 2 buffered messages, so P_3 and the stored packets P_4 and P_5 are output. Packet P_7 does not arrive, so when we receive P_{10} and there are now more than maximum-count = 2 messages buffered, we discard P_7 and output the stored packets in order.

One issue with count-based windowing occurs at the end of the day. With no more input packets to process, we cannot output stored packets. This windowing is used in [5], but residual packets are not addressed. It is solved in this work either by use of the hybrid time & count method's time limit, or by dynamically altering the maximum-count threshold.

High reliability - time & count: Combining the time and count based high modes provides the most robust solution for processing out-of-order packets. We can utilise the count threshold's time-independent ability to follow the incoming packet rate as it fluctuates during the day, whilst still allowing an upper limit on delay times.

In Figure 4's single input example, both the time and count windowing thresholds are used to determine if a stored packet should be output. Packet P_4 takes too long to arrive, therefore exceeding P_5's timeout and resulting in P_4 being discarded. Later, P_8 is also late, but whilst waiting for P_9's timeout, the number of buffered messages exceeds maximum-count = 2, and P_8 is discarded.

B. Low Latency Mode

The singular arbitration mode in our base design [4] lacked the ability to reduce arbitration to its simplest, fastest form: outputting a stream of unique, ordered packets. We present it in this work as the low latency mode.

We treat an input packet as valid based solely on whether its sequence number is larger than or equal to the next expected sequence number. We do not wait for missing packets and hence, do not require resources for packet storage, and minimise transmission latencies.

The Ethernet, IP and UDP packet headers pose a problem when trying to minimise the arbitration latency. The packet's sequence number is only visible after we process these headers, which may take a number of cycles. We solve this by assuming a packet is valid and immediately output it. When we encounter the sequence number and it is not valid—i.e., less than the next expected sequence number—we register an output error, causing the packet to be discarded.

Similarly, when packets arrive on both input streams simultaneously, we must make the choice of which packet we should output without any information on either packet's contents. There is no method that can guarantee a priori which stream to select, so we instead select the last stream on which we encountered a valid packet.

With these simple operations we reduce arbitration to a single cycle. The single arbitration mode in [4] took 7 cycles meaning, with our new arbitrator design, applications can receive an arbitrated stream of packets 7 times faster if they are able to accommodate missing packets. Also, as it does not require many resources, we can output the low latency mode simultaneously with our high reliability mode.

C. Network-level Operation

When processing at the network-level, rather than within a computing node, we must take an active role in routing non-market data packets. Even within a dedicated market data feed network, routers and NICs will transmit information requests to other nodes. We must reserve FPGA resources to route these non-market feed packets. Network identification packets are typically only hundreds of bits, requiring little storage space, and are processed at the lowest possible priority to minimise interference with market packets.

Past works [5], [6] and [7] focus on processing market data feeds on FPGAs situated within computing nodes rather than at the network-level. Data is then passed to the CPU or GPU via low latency DMA transfers. This scales poorly if further nodes are needed as each will need an FPGA for data feed processing. Our packet-based, network-level arbitrator consolidates the node-independent arbitration operations. Only the low latency DMA transfers need be implemented within nodes to create a newly scalable system with the same functionality as past works.

D. Customisation and Extensibility

As our arbitrator deals with the initial stages of storing, processing and identifying market feed packets and messages, it is a simple matter to extend our system to provide additional functionality within the FPGA. We support:

- Runtime configurable windowing thresholds.
- Any physical connection for input and output ports.
- The size and number of both market and non-market packets stored can be configured at compile time.
- Any market data feed protocol can be adopted, not just those of TotalView-ITCH, OPRA and ARCA.

IV. Implementation

We verify our proposed design and low latency architecture by implementing an A/B line arbitrator for each of the TotalView-ITCH, OPRA and ARCA market data feed protocols. For each protocol we require knowledge of the maximum packet size, the sequence number width and the byte position of the sequence number in the packet. This is determined by their specifications, and is given in Table I.

To reduce latency we make use of a wide 128-bit data-path, double the 64-bit width commonly used—as 64-bit multiplied by the 156.25MHz reference frequency = 10Gbps. This can negatively affect the routing delay, but with our low latency architecture we achieve latencies an order of magnitude lower than [7] while maintaining at least 20Gbps throughput, twice that of the 10Gbps Ethernet line rate.

TABLE I. Packet protocol specifications.

Protocol	Max Packet Size	Sequence Number	
		Width	Position
ITCH	9000 bytes	64 bits	53
OPRA	1000 bytes	31 bits	47
ARCA	1400 bytes	32 bits	46

We verify and test our design in two ways. First, we implement an arbitrator for each of our chosen protocols within a Xilinx Virtex-6 LX365T FPGA on an Alpha Data ADM-XRC-6T1 card. As our processing rate is greater than the 10Gbit Ethernet connections used by each protocol, we transfer data via PCI Express. We configure each arbitrator for their respective protocols according to the values from Table I.

Second, we implement our design on a Xilinx Virtex-5 LX330T FPGA within an iD ID-XV5-PE20G network interface card. This card receives a duplicated data feed over two 10Gbit Ethernet connections. The high reliability and low latency outputs are transmitted to the host via PCI Express, with the layout given in Figure 5. We also allow for additional user logic within the FPGA. The TotalView-ITCH protocol is used to test our real world design as it is the most resource and processing intensive, and messages from 9 September 2012 are used to test the system.

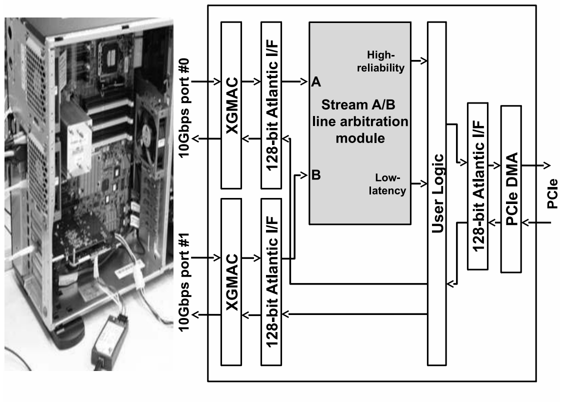

Fig. 5. The layout of our arbitrator module within the FPGA.

OPRA and ARCA operate on top of UDP, while TotalView-ITCH uses a UDP variant called moldUDP64 [11]. Our design stores 8 packets, each with sufficient space for the Ethernet (14 bytes), IP (20 bytes) and UDP (8 bytes) headers, as well as the packet payloads from Table I.

V. Results

Sequence number comparisons are a source of our critical path, so reducing the width of sequence numbers will lower our routing delay and latency. Our Virtex-6 implementation found TotalView-ITCH, with its 64-bit sequence numbers achieved a single cycle latency of $6ns$, whereas the OPRA and ARCA both achieved $5.25ns$ with sequence number widths half that of TotalView-ITCH. This suggests that artificially truncating the sequence numbers of packets can benefit arbitration, at the cost of additional logic to deal with packets that straddle the new sequence number boundary.

TotalView-ITCH's $6ns$ latency results in a 166MHz FPGA design with 21.3Gbps throughput, while OPRA and ARCA's $5.25ns$ latency means a 190MHz design and 24.3Gbps throughput. With financial markets making greater use of higher throughput connections, our design will be well placed to capitalised on this increased throughput capacity. Indeed, the TotalView-ITCH message feed is already available via both 10Gbps and 40Gbps connections.

TotalView-ITCH's requirement for 9000 byte packets is multiple times that of OPRA (1000 bytes) or ARCA (1400 bytes). Figures 6 and 7 show its resource usage does not increase in proportion to this requirement, mainly due to buffering host communications. Buffering plays a larger role in our network interface card design as we implement two bi-directional 10Gbit Ethernet connections. Its operation is therefore an important test of real world performance.

From analysing the messages from our real world implementation, within which we process TotalView-ITCH messages from two redundant 10Gbps Ethernet links, we find

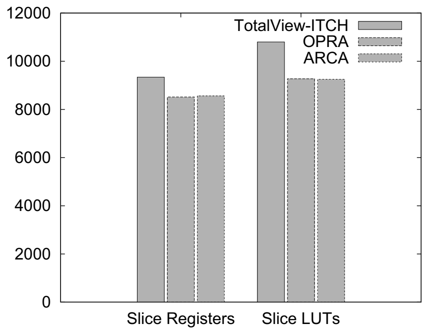

Fig. 6. Slice usage for the three messaging protocols.

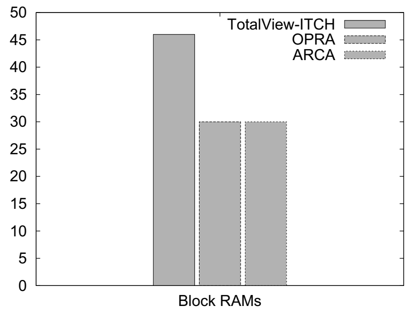

Fig. 7. Block RAM usage for the three messaging protocols.

we process about 322 million messages throughout the day. This would require 29 bits for message sequence numbers, demonstrating that we can safely truncate TotalView-ITCH messages without affecting performance.

The real world implementation also allows us to verify the latency of our two arbitration modes. By inspecting the packets on the host after they have been arbitrated we can easily read out the number of cycles it took for each packet to be processed. For the high reliability mode we find it takes 7 cycles to process expected packets, i.e., packets not needing to be buffered. For the low latency mode we find packets are processed in 1 cycle.

We now measure our new arbitrator design against our basic design from previous work [4]. The new design now supports three high reliability windowing methods and simultaneously outputs a low latency mode with a single clock cycle latency. In high reliability mode, our new design achieves a $42ns$ latency for the resource intensive TotalView-ITCH protocol, and $36.75ns$ for OPRA and ARCA. The previous design achieved only a $56ns$ latency for all packet protocols. In low latency mode, our new design supports latencies of $6ns$ for TotalView-ITCH and $5.25ns$ for OPRA and ARCA. This mode is not available in previous work.

Finally, we compare a software arbitrator using the cutting-edge IBM PowerEN processor [12], with out-of-order packets stored in L2 cache and using a time-based windowing mechanism similar to the high reliability time mode in this work. Arbitration is performed using only the OPRA protocol and takes $150ns$ compared to $36.75ns$ in our design. Thus, our design achieves a 4.1 times lower latency.

VI. Conclusion

In this paper we outline an A/B line arbitrator for market data feeds that helps to mitigate the impact of missing packets on time-critical financial applications. Our scheme provides a high-reliability and low-latency mode simultaneously, supports three windowing methods, and is customisable for other protocols, and varying packet sizes and numbers.

We outline an architecture, and provide implementations for the TotalView-ITCH, OPRA and ARCA protocols on a Xilinx Virtex-6 FPGA. For testing with real market data, we also create a TotalView-ITCH implementation on a network card equipped with a Virtex-5 device. Compared to our past work [4] where the design had a latency of $56ns$ regardless of the protocol, we now achieve $42ns$ for TotalView-ITCH and $36.75ns$ for OPRA and ARCA using the same target device. The new low-latency mode reduces arbitration to a single cycles and supports latencies of only $6ns$ and $5.25ns$ respectively. We offer latencies 10 times lower than an FPGA-based commercial design and 4.1 times lower than the hardware-accelerated IBM PowerEN processor, with throughputs more than double that of the specified 10Gbps line rate.

Acknowledgment

This work is supported by NEC, by the European Union Seventh Framework Programme under the grant agreement number 257906, 287804 and 318521, and by UK EPSRC.

References

[1] "NASDAQ TotalView-ITCH 4.1," 2013. [Online]. Available: https://www.nasdaqtrader.com/content/technicalsupport/specifications/dataproducts/NQTV-ITCH-V4_1.pdf

[2] "OPRA Participant Interface Specification," Tech. Rep. [Online]. Available: http://www.opradata.com/specs/participant_interface_specification.pdf

[3] "NYSE ARCA Europe Exchange Client Specification," Tech. Rep. [Online]. Available: http://www.nyxdata.com/doc/36868

[4] S. Denholm, H. Inoue, T. Takenaka, and W. Luk, "Application-specific customisation of market data feed arbitration," in *Field Programmable Technology (FPT)*, 2013, pp. 322–325.

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

[6] "Solarflare AOE Line Arbitration Brief," 2013. [Online]. Available: http://www.solarflare.com/Content/UserFiles/Documents/Solarflare_AOE_Line_Arbitration_Brief.pdf

[7] "The Next Generation Trading Infrastructure." [Online]. Available: http://www.ts-a.com/attachments/white_paper_c11-720080.pdf

[8] C. Leber, B. Geib, and H. Litz, "High Frequency Trading Acceleration Using FPGAs," in *Field Programmable Logic and Applications (FPL)*, 2011, pp. 317–322.

[9] R. Pottathuparambil, J. Coyne, J. Allred, W. Lynch, and V. Natoli, "Low-Latency FPGA Based Financial Data Feed Handler," in *Field-Programmable Custom Computing Machines (FCCM)*, 2011, pp. 93–96.

[10] J. W. Lockwood, A. Gupte, N. Mehta, M. Blott, T. English, and K. A. Vissers, "A Low-Latency Library in FPGA Hardware for High-Frequency Trading (HFT)," in *High-Performance Interconnects (HOTI)*, 2012, pp. 9–16.

[11] "MoldUDP64 Protocol," 2009. [Online]. Available: http://www.nasdaqtrader.com/content/technicalsupport/specifications/dataproducts/moldudp64.pdf

[12] D. Pasetto, K. Lynch, R. Tucker, B. Maguire, F. Petrini, and H. Franke, "Ultra low latency market data feed on IBM PowerEN," *Computer Science - Research and Development*, vol. 26, pp. 307–315, 2011.