PAPER

Network-level FPGA Acceleration of Low Latency Market Data Feed Arbitration

Stewart DENHOLM[†], *Nonmember*, **Hiroaki INOUE**[††], **Takashi TAKENAKA**[††], *Members*, **Tobias BECKER**[†], *and* **Wayne LUK**[†], *Nonmembers*

SUMMARY Financial exchanges provide market data feeds to update their members about changes in the market. Feed messages are often used in time-critical automated trading applications, and two identical feeds (A and B feeds) are provided in order to reduce message loss. A key challenge is to support A/B line arbitration efficiently to compensate for missing packets, while offering flexibility for various operational modes such as prioritising for low latency or for high data reliability. This paper presents a reconfigurable acceleration approach for A/B arbitration operating at the network level, capable of supporting any messaging protocol. Two modes of operation are provided simultaneously: one prioritising low latency, and one prioritising high reliability with three dynamically configurable windowing methods. We also present a model for message feed processing latencies that is useful for evaluating scalability in future applications. We outline a new low latency, high throughput architecture and demonstrate a cycle-accurate testing framework to measure the actual latency of packets within the FPGA. We implement and compare the performance of the NASDAQ TotalView-ITCH, OPRA and ARCA market data feed protocols using a Xilinx Virtex-6 FPGA. For high reliability messages we achieve latencies of 42ns for TotalView-ITCH and 36.75ns for OPRA and ARCA. 6ns and 5.25ns are obtained for low latency messages. The most resource intensive protocol, TotalView-ITCH, is also implemented in a Xilinx Virtex-5 FPGA within a network interface card; it is used to validate our approach with real market data. 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.

key words: *data feed arbitration, acceleration, FPGA, low latency, finance*

1. Introduction

Financial exchanges send out market information in form of market data feeds. These data feeds describe market events such as available and completed trades. Financial institutions can subscribe to these data feeds and utilise the information in a number of applications: It is possible to determine the current state of the market and the institution's risk, to search for time-critical arbitrage opportunities, or to trade automatically with algorithmic trading platforms. In the latter two examples, time-critical decisions have 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 and hence, 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.

As message feeds are typically transmitted over Ethernet using UDP, they offer no guarantee that the messages will be received or arrive in order. Ever increasing line rates allow for lower latency transmissions, but little work is being done to tackle messages that are lost in transmission. Financial exchanges address this issue by providing redundant networks that transmit two identical message feeds from the exchange, referred to as A and B feeds. Financial institutions traditionally arbitrate between the two feeds in software to mitigate the effects of lost packets, and provide a single message stream for processing by downstream financial applications. However, general-purpose architectures of CPU systems separate data acquisition and processing, leading to latency penalties when processing external data.

The pipelined and parallel nature of A/B arbitration provides an opportunity for hardware acceleration of time-critical processing before the resulting data is passed to the CPU. While FPGAs are capable of performing high bandwidth, low latency processing, achieving this requires careful consideration of design choices, especially when dealing with the data-path widths necessary to support the ever-increasing demands on latency and throughput. Resource choice, placement and, often times, duplication play a pivotal role in meeting these constraints.

In this work we present an architecture for low latency A/B arbitration, supporting all market data feed protocols. We provide a low latency and a high reliability mode, and support dynamically configurable windowing methods so that downstream applications can alter the arbitrator to respond to changing requirements in real time. The contributions of our work are:

- A new hardware accelerated, low latency A/B line arbitrator which runs two packet windowing modes simultaneously. It supports three dynamic windowing methods, any market data protocol, indepen-

Manuscript received January 1, 2011.
Manuscript revised January 1, 2011.
[†]The author is with Imperial College London, UK
[††]The author is with NEC Corporation, Kawasaki, Japan
DOI: 10.1587/trans.E0.??.1

dent memory allocation per input, and configurable data-path widths (section 3).
- Performance models and design considerations necessary to perform low latency processing of data from multiple inputs. This includes: critical path analysis, guaranteeing deterministic memory access, and creation of a low latency testing framework (section 4 and 5).
- Implementation and evaluation 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. We use our new low latency testing framework and real market data to measure its performance (section 6 and 7).

2. Background and Motivation

Reliable communication protocols such as TCP rely on retransmission of lost packets to provide a reliable message stream, but retransmissions lead to higher latencies and degraded throughput. As an alternative, one can achieve reliability through duplicating packets and sending them through disjoint communication channels. Multiple path communication with duplicated packets is an established method to provide reliable communication for time-critical applications [4]. It has also been demonstrated that dual path communication with FPGA-based duplication and merging can maintain higher bandwidths with lower retransmission rates than a single path solution [5]. However, the focus in previous work on multi-path communication is usually on reducing retransmission rates, maintaining high bandwidths, or providing protection against complete link failure.

A/B arbitration is a form multi-path communication that avoids retransmission and is therefore not fully reliable. Retransmission of packets would result in unacceptable latencies where time-sensitive trading opportunities will be lost. Instead, the goal is usually to balance minimal packet losses and low communication latencies. A/B line arbitration aims to compensate for missing packets within an acceptable time frame and allowing each application to set and adjust this time frame themselves is a key factor in its successful operation and our proposed design.

The importance of A/B arbitration will continue to 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.

NASDAQ TotalView-ITCH 4.1, OPRA and ARCA are market data feed protocols that provide market information. TotalView ITCH is a data-feed provided by NASDAQ and delivers a range of market data in variable length messages [1]. The messages include order book information reflecting the interest of buyers and sellers in a particular financial instrument, trade messages, administrative messages such as paused trading on a security, and event controls such as start and end of the day. The NYSE Arca data feed contains similar market information on depth of book, trades, order imbalance data, and security status messages [3] but the technical implementation and packet size differ from TotalView-ITCH. The Options Price Reporting Authority (OPRA) provides information about transactions in the options markets [2]. Information such as last sale information and current options quotations is featured in a data feed of variable length packets that can contain multiple messages.

Morris [6] 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 [7] 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 [8], 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 [9], and the NASDAQ data feed handler by Pottathuparambil [10]. Other works describe, but do not implement, arbitration, like the high frequency trading IP library from Lockwood [11]. 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.

In our previous work [12] 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. We also lacked the low latency architectures and testing methodologies necessary to achieve the lowest possible latencies, remedied here by the creation of a new low latency design and testing framework.

3. Flexible 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.

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. In the case where market data protocols do not issue packet numbers—such as OPRA, where sequence numbers are assigned to messages—we use the sequence number of the first message in the packet to identify that packet. The next expected packet is then:

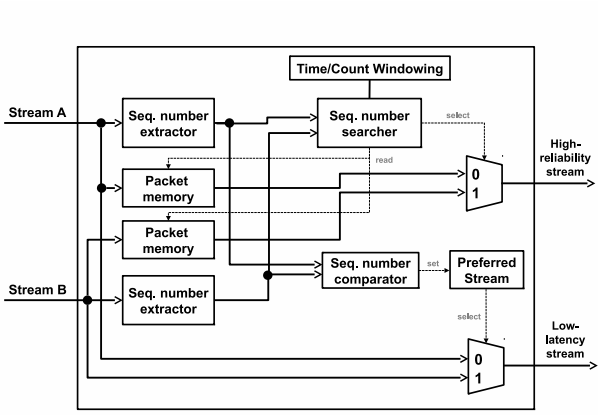

Fig. 1: The layout of our A/B line arbitration design.

$$SN_{pkt+1} = SN_{pkt} + M_{pkt} \quad (1)$$

where SN_{pkt} is the sequence number of the current packet and M_{pkt} is the number of messages it contains.

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.

3.1 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 [6], time & count by [7], while [8] does not detail its windowing approach. This is the first work to support all three methods provide low latency, application-specific parametrisation, and output a high reliability and low latency stream simultaneously. Furthermore, we offer dynamic reconfiguration between the three windowing modes so that downstream applications can modify the arbitration method in real time, a feature not previously available.

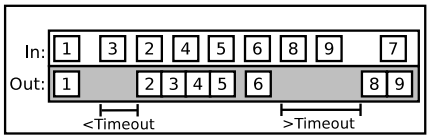

Fig. 2: High reliability time.

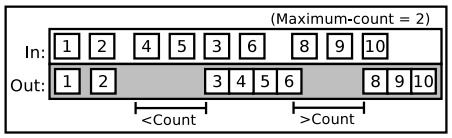

Fig. 3: High reliability count.

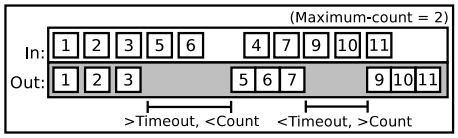

Fig. 4: High reliability time & count.

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 [6], 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.

3.2 Low Latency Mode

The singular arbitration mode in our base design [12] 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 while also minimising 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. This differs from the previous high reliability modes where we have additional cycles available to process and compare the sequence number.

With these simple operations we reduce arbitration to a single cycle. The single arbitration mode in [12] 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.

3.3 Network-level Operation

By choosing to arbitrate between message streams at the network layer we remove the need for each downstream application to arbitrate between the streams themselves, eliminating this processing redundancy. However, 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 [6], [7] and [8] 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.

3.4 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 the following customisations:

The windowing threshold values can be reconfigured at run time, as discussed above. The user, or a monitoring function is able to tailor the time and number of messages delayed, to meet both the changing needs of the market and downstream applications.

Any physical connection for input and output ports can be used. Our arbitrator can connect to any commercial or customised network by translating the connection's interface, e.g. Ethernet or InfiniBand, to a standard FPGA interface. The arbitrator may also be used within a computing node, rather than at the network level.

The size and number of packets stored can be configured at compile time, for both market and non-market packets. The size of the pipeline is adjusted accordingly.

Any market data feed protocol can be adopted, not just those of TotalView-ITCH, OPRA and ARCA. The only protocol-specific information required is the maximum packet size, and the location and bit-width of the sequence number within the packet.

4. Performance Model

Low latency processing within an FPGA has many difficulties, and the high performance requirements of A/B line arbitration specifically, pose a number of challenges. In this section we present a performance model of our high performance, low latency arbitration approach.

Any decision regarding a new packet is dependent on its sequence number, and therefore the number of cycles we must wait to process it. Given the packet byte position where the sequence number begins, Pos_{seq}, its length in bytes, Len_{seq}, and the byte width of our datapath, $Data_{width}$, the number of cycles we must wait to encounter the sequence number is then:

$$C_{seq} = \lfloor \frac{Pos_{seq} + Len_{seq} - 1}{Data_{width}} \rfloor \tag{2}$$

The packet processing latency's lower bound is achieved when we encounter expected packets. No further processing should be needed, but due to the delay from Equation 2 we must still begin storing it in memory and then read it out. The number of cycles for an expected packet is then:

$$C_{exp} = C_{seq} + C_{read} \tag{3}$$

The worst case time delay, C_t, is the number of cycles we wait when the windowing system delays a packet. This acts as the upper bound latency. C_t is potentially infinite for the high reliability count mode, as it is designed to be time-independent. In practice this only occurs when there are no further packets to process, and can be resolved via runtime alteration of the maximum-count threshold. The upper bound cycle delay is given by:

$$C_{max} = C_{exp} + C_{write} + C_t \tag{4}$$

C_t for the time mode is simply the timeout, while for time & count it is the lower of the time and count delays. In finding C_t for the count mode we must take into account the time taken to receive sufficient messages—not packets—to exceed our maximum-count threshold. This is based on: the maximum-count value, MC; the maximum number of packets that can be stored in the buffer, n; the number of cycles required to find the stored packet with the lowest sequence number, $\log_2(n)$; the number of packets per input, per cycle, λ_p; the number of messages per packet, λ_m; and the number of inputs, I. For the count mode, C_t is then:

$$C_t = \log_2(n) + \frac{MC}{I \times \lambda_p \times \lambda_m} \tag{5}$$

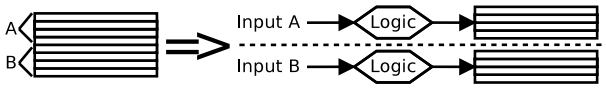

Fig. 5: The address generator automatically splitting a large memory.

5. High-Performance Architecture

Achieving fast, low latency processing in our arbitrator requires careful development of the overall hardware architecture. In this section, we present several architectural considerations to achieve low latency processing of data from multiple input streams.

5.1 Deterministic, Multiple-Input Memory Access

Storing data in DDR memory is unsuitable for low latency applications due its high access times. In addition, DDR memory locations must be refreshed periodically to maintain their state, leading to non-deterministic access patterns. Other memory types, such as Content-Addressable Memory (CAM) or specialised flash memory, are not widely available in commercial systems and their access latencies are still higher than those required by high performance designs. Internal FPGA memory, such as Block RAMs, are the best option to ensure fast and predictable access times.

With multiple input sources and a single, centralised memory within an FPGA, the number of writers to memory will exceed the available memory ports. In reality, a large FPGA memory is comprised of many, smaller ones mapped to any available memory within the FPGA. Many memory ports are then available to us, but micromanaging memory on such a scale is undesirable. The problem can be tackled at a higher, algorithmic level through the use of memory address generators.

For our multiple input system we create a single logical memory, providing the address of a free memory location to the logic for each input. The simplest case of two inputs is shown in Figure 5, where the address range of input A covers the first half of the memory, and input B, the second. When built, the two memory halves will be independent and so will be mapped close to their respective accessors.

To tackle storage inefficiencies, i.e., when data does not appear uniformly across all inputs, we must: **(1)** minimise data duplicated across inputs; **(2)** ensure the initial memory allocation reflects the percentage of total data originating at each input; and **(3)** make sure data is removed from each memory in proportion to that memory's occupancy.

For (1), we have no duplicated data in memory as we guarantee packets are globally unique by checking packet sequence numbers. Given the nature of our duplicated market data feeds, (2) is tackled by evenly allocating memory to each input. Finally, for (3), arbitration provides for a well defined, ordered removal of packets, for which our packet windowing methods establish an upper bound.

We select a memory packet to output by finding the smallest sequence number using a binary search, requiring $\log_2(n)$ cycles for n packets. Binary search is realised in a pipelined data-path that uses Block RAM for storing the packet numbers. The key comparison is realised with a register storing the search key and a comparator. The search index calculation uses two registers for the upper and lower search index, an adder and bitshift for the midpoint calculation, and a comparator. We do not sort packets before writing to memory as binary insertion takes the same amount of time, but does not scale well with multiple inputs.

5.2 Optimising Packet Accesses

Now that data access is predictable and can be easily scaled for multiple writes we must deal with read latency. Block RAMs require two clock cycles for their data to be available, making packet comparison in memory very costly, especially if both packets are stored within a single Block RAM. A small meta-data cache in registers will allow immediate access to packet-specific information, reducing routing latency as fewer links to packet memory are required.

The meta-data cache is shown in Table 1, with example bit widths for the larger, TotalView-ITCH protocol. To be effective it must be small, fast and contain all necessary packet information. A cache line is 128 bits wide and utilises dual-port distributed RAM so it can be written and read in the same clock cycle. It makes efficient use of resources as, even for the large TotalView-ITCH protocol, each cache line fits into a single SLICEM within the FPGA.

Table 1: Meta-data cache contents.

Name	Bits	Description
Nr. of Data	10	Size of packet/data-path width
Sequence Nr.	64	Packet's unique sequence number
Cycles Remaining	32	Cycles until this packet times out
Nr. of Messages	16	Total messages in this packet
Final Byte Enable	4	Enable signal for final packet bytes
Packet Being Input	1	Is this packet still being input
Cache Line Free	1	Cache line available or occupied
Total	**128**	

5.3 Cycle-Accurate Testing

The difficulty in testing low latency designs is that many of the corner cases occur within nanosecond time-frames, so it becomes difficult to arrange packets from multiple feeds to arrive at the arbitrator at precise times. We therefore create a testing framework within the FPGA

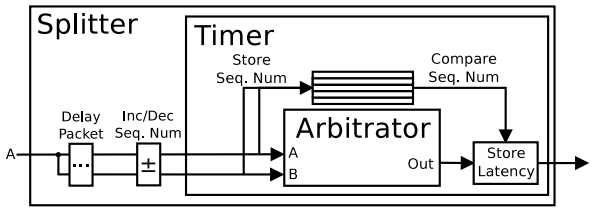

Fig. 6: The cycle-accurate testing framework.

using two wrappers around our design, as shown in Figure 6.

The outer, splitter wrapper, takes a packet from one input and mirrors it on the others, either in the current cycle or delayed one or two cycles. The splitter can increment or decrement the mirrored sequence number so it appears as a new packet. The inner, timer wrapper, notes incoming packet sequence numbers, counts the cycles until it appears on the output, and writes this latency into the packet header.

The testing framework only requires the packet's sequence number location, and is otherwise application and protocol independent. We do not interfere with the arbitrator's operation or affect the critical path as all measurements are performed outside of the arbitration module.

5.4 Improving Throughput and Latency

Improving the throughput and latency of a design can be accomplished by widening the data-path or increasing the clock frequency. A wider data-path requires more resources and routing at each stage of the design, but means fewer clock cycles are required to process packets (see Equation 2).

The critical path is defined by operations with the largest combined logic (T_l) and routing (T_r) delays. The maximum clock frequency is then:

$$F_{max} = \frac{1}{T_l + T_r} \tag{6}$$

Logic delays are reduced by increased parallelism, and using multi-staged pipelines to spread processing over multiple cycles. Our $\log_2(n)$ binary packet searcher and multi-stage input packet processing are targeted at minimising this delay.

Routing delays are reduced by using fewer resources, placing interconnecting resources physically closer together, and using additional data buffering stages. Routing delays are harder to reduce and depend heavily on the FPGA utilisation and the design's interconnections. We tackle this by separating data processing streams so logic and memory for each stream are independent and can be placed together.

Our design's critical path comes from packet storage and sequence number comparisons. Any resource/frequency trade-offs to improve throughput and latency must then be made by modifying these design elements: either storing fewer packets or comparing shorter sequence numbers.

6. 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 2.

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 [8] while maintaining at least 20Gbps throughput, twice that of the 10Gbps Ethernet line rate.

Table 2: 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 by entering the values from Table 2 into our configuration file. Adopting a new protocol in the future requires only that we indicate where the equivalent fields are located within the new packet format.

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 7. 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.

OPRA and ARCA operate on top of UDP, while TotalView-ITCH uses a UDP variant called moldUDP64 [13]. 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 2. Each packet has an associated meta-data

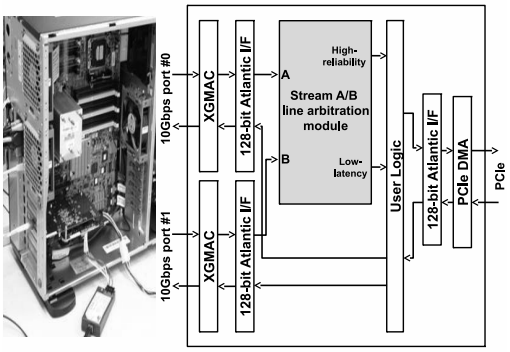

Fig. 7: The layout of our arbitrator module within the FPGA.

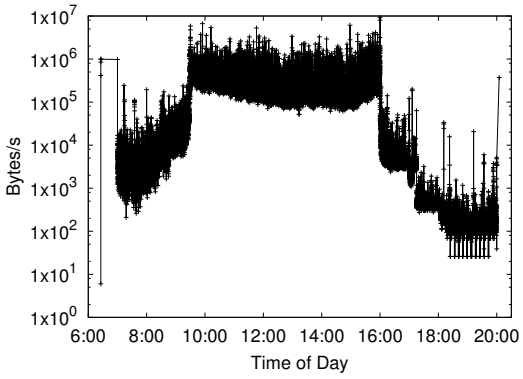

Fig. 8: Market data feed throughput during the day.

cache entry, for which the entire cache will require only 4 slices. With our deterministic, multiple-input memory architecture we can simultaneously write packets from each input feed and read packets out. The high-level nature of the architecture also makes it simple to expand or contract the memory size at compile time to suit our specific market protocol.

7. 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 a maximum throughput of 21.3Gbps, while OPRA and ARCA's $5.25ns$ latency means a 190MHz design and a maximum throughput of 24.3Gbps. Both designs are fast enough to satisfy 20Gbps processing. With financial markets making greater use of higher throughput connections, our design will be well placed to capitalise on this increased throughput capacity. Indeed, the TotalView-ITCH message feed is already available via both 10Gbps and 40Gbps connections. However, only fraction of this throughput is currently used by the message feed. Figure 8 illustrates the market activity for TotalView-ITCH at different times of the day, showing the total throughput per second. The rate of incoming messages changes throughout the day, illustrating the need for run-time configuration of the high reliability mode thresholds. For example, the acceptable delay between the busy $9.30 - 16.00$ market hours will be less than in the pre and post market hours. We find that, at peak times, we must process 73Mbps (or 1000 packets per second). Our implementation is more than capable of meeting existing demand, so we must focus on shortening our processing latency, i.e., the time taken to react to packets and messages. Previous works often emphasise the fact that their designs operate at the 10Gbit Ethernet line rate, but this is not required to meet the rate of arriving current market data feed messages.

TotalView-ITCH's requirement for 9000 byte packets is multiple times that of OPRA (1000 bytes) or ARCA (1400 bytes). Figures 9 and 10 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.

7.1 Latency

We measure our packet processing latency by analysing our design and making use of the formulae we derive in Section 4. We find that for our implementation, C_{write}, the number of cycles required to write to our packet buffer, to equal 2: one cycle to trigger a write operation, and one cycle to write to the memory. C_{read}, takes: one cycle to trigger an output operation, one cycle to specify the memory location, one cycle for the data to appear on the memory output, and one cycle to output the data, for a total of 4 cycles. Each packet is broken up into smaller segments, each of size $Data_{width}$, and subsequent packet segment reads are pipelined.

Plugging the protocol specific values Pos_{seq} and Len_{seq} from Table 2 and $Data_{width}$ of our 128-bit (16 byte) data-path into Equation 2, we find the number of cycles to encounter the sequence number are: $C_{seq} = \lfloor \frac{53+8-1}{16} \rfloor = 3$, $C_{seq} = \lfloor \frac{47+4-1}{16} \rfloor = 3$, and $C_{seq} = \lfloor \frac{46+4-1}{16} \rfloor = 3$, for TotalView-ITCH, OPRA and ARCA respectively. Despite having different message formats, the sequence numbers of all three protocols happen to be visible in the same cycle. The lower bound of the packet processing latency is then found as $C_{exp} = 3 + 4 = 7$ cycles.

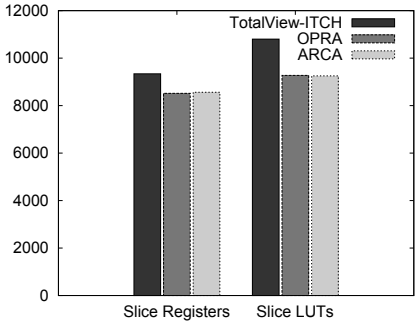

Fig. 9: Slice usage for the three messaging protocols.

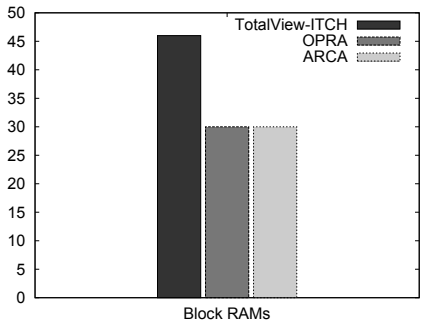

Fig. 10: Block RAM usage for the three messaging protocols.

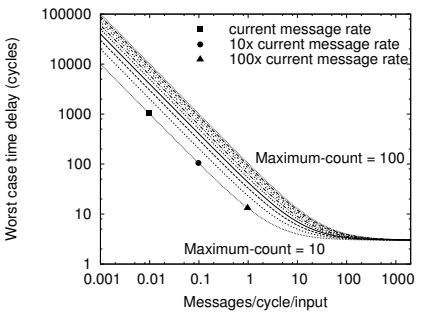

Fig. 11: Worst case time delay as an indicator of arbitrator saturation.

The upper bound latency, C_{max}, requires knowledge of the worst case time delay, C_t. For the high reliability time mode C_t is the user-defined timeout value for a stored packet, therefore, $C_{max} = 3 + 2 + C_t = 5 + C_t$. To delay a packet for a maximum of 50 cycles, for example, we set the timeout to $50 - 5 = 45$ cycles.

The high reliability count mode is not dependent on time, but rather the rate of arriving messages, so its upper bound latency offers little insight. As an example, let us consider our implementation and the peak time performance for TotalView-ITCH messages where we receive around 1000 messages per second. From this we obtain: $n = 8$, $I = 2$, $\lambda_p = 6.5 \times 10^{-6}$ packets per cycle and $\lambda_m = 9000/6 = 1500$ messages per packet. Here, λ_m assumes the worst case scenario of 100% packet utilisation, modelling a fully saturated market feed. Using Equation 5 and setting the maximum-count threshold $MC = 10$ messages, we find $C_t = \log_2(8) + \frac{10}{2 \times 6.5 \times 10^{-6} \times 1500} = 516$ cycles. The upper bound latency is then $C_{max} = 3 + (2 - 1) + 516 = 520$ cycles, where one of the C_{write} cycles is performed in parallel, so the write latency is only 1 cycle.

A worst case time delay of 520 cycles is a long time for TotalView-ITCH messages, but for OPRA we find an even longer 10125 cycles. For $MC = 10$ it is possible for this threshold to be exceeded multiple time over within a single cycle, and indeed, this is the most likely scenario. It is therefore best that C_{max} is not used as an indicator of upper bound latency, but rather as a measure of the incoming message rate, with consistently low values indicating an increasing likelihood of throughput saturation.

7.2 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 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 our latency calculations by making use of our cycle-accurate testing framework. 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. This low latency result also succeeds in demonstrating the resolution of our testing framework, as we are able to measure processes that occur within one cycle.

7.3 Future Performance Scalability

Our new saturation indicator is shown in Figure 11 where the message rate is used to calculate the worst case time delay, C_t, for maximum-count values ranging between 10 and 100 messages. As the rate increases, C_t tends towards $\log_2(8) = 3$ cycles, indicating the arbitrator is becoming saturated and the high reliability mode is now effectively functioning as a higher latency version of the low latency mode. The value we calculated for C_t using our real market data implementation is plotted in the graph as a square. From this we see that current market rates are well within the range that allows the high reliability modes to work effectively.

To model future message rates we plot a circle and triangle in Figure 11, representing over 6 million and 60 million messages per second respectively. We see that 6 million messages is over an order of magnitude away from saturation meaning our arbitrator is well within its operational limits. The triangle's 60 million messages a second is closer to saturation, but has not yet crossed the inflection point. For such high message rates, the effectiveness of arbitration can then be increased using a higher maximum-count.

As even 60 million messages per second is still less than the 10Gbps line rate of Ethernet, it is possible to process 100 times the current market data rate using current technology.

7.4 Performance Improvement

We now measure our new arbitrator design against our basic design from previous work [12]. The new design 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 [14], 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.

8. Conclusion

In this paper we outline an A/B line arbitrator for market data feeds that operates at the network level. We present an architecture that simultaneously produces a high reliability and a low latency output stream. A key novelty in this work is the ability to dynamically reconfigure the high reliability with three windowing methods to adapt to the requirements of downstream financial applications in real time. We also introduce network-level processing which was previously not available. Our architecture supports any market data protocol, and can be configured for different data-path widths.

Furthermore, we present a model for packet processing latencies and discuss architectural considerations of efficient low latency design, opportunities for efficient, high-level configuration, and our impact on downstream applications.

An implementation of our architecture targeting a Xilinx Virtex-6 FPGA achieves lower latencies than our previous work. We now achieve $42ns$ for TotalView-ITCH and $36.75ns$ for OPRA and ARCA, where our previous implementation had a fixed latency of $56ns$ regardless of the protocol on the same target device. Both designs have a lower bound latency of 7 cycles for high reliability processing, while our new low latency mode performs simple arbitration within 1 cycle. This corresponds to latencies of $6ns$ and $5.25ns$ respectively. This mode was not available previously.

Finally, the most resource intensive protocol, TotalView-ITCH, is also implemented on a Xilinx Virtex-5 FPGA within a network interface card using real market data, and verified using our new cycle-accurate testing. We discover latency measurements can indicate message feed saturation, providing a quantifiable method to indicate the point at which the low latency windowing mode becomes the optimal approach. This is used to demonstrate the effectiveness of our design at message rates 100 times their current level. For the three messaging protocols examined, TotalView-ITCH, OPRA and ARCA, 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.

References

[1] "NASDAQ TotalView-ITCH 4.1." `https://www.nasdaqtrader.com/content/technicalsupport/specifications/dataproducts/NQTV-ITCH-V4_1.pdf`, 2013.
[2] "OPRA Participant Interface Specification." `http://www.opradata.com/specs/participant_interface_specification.pdf`, 2011.
[3] "NYSE ARCA Europe Exchange Client Specification." `http://www.nyxdata.com/doc/36868`, 2013.
[4] P. Ramanathan and K.G. Shin, "Delivery of time-critical messages using a multiple copy approach," ACM Trans. Comput. Syst., vol.10, no.2, pp.144–166, May 1992.
[5] Y. Kodama, T. Kudoh, and T. Shimizu, "Dependable communication using multiple network paths on fast long-distance networks," Systems and Computers in Japan, vol.38, no.12, pp.46–54, 2007.
[6] G. Morris, D. Thomas, and W. Luk, "FPGA Accelerated Low-Latency Market Data Feed Processing," High Performance Interconnects, 17th IEEE Symposium on, 2009.
[7] Solarflare, "Solarflare AOE Line Arbitration Brief." `http://www.solarflare.com/Content/UserFiles/Documents/Solarflare_AOE_Line_Arbitration_Brief.pdf`, 2013.
[8] Cisco, "The next generation trading infrastructure." `http://www.cisco.com/c/dam/en/us/products/collateral/switches/nexus-3000-series-switches/white_paper_c11-720080.pdf`.
[9] C. Leber, B. Geib, and H. Litz, "High Frequency Trading Acceleration Using FPGAs," Field Programmable Logic and Applications (FPL), pp.317–322, 2011.
[10] R. Pottathuparambil, J. Coyne, J. Allred, W. Lynch, and V. Natoli, "Low-Latency FPGA Based Financial Data Feed Handler," Field-Programmable Custom Computing Machines (FCCM), pp.93–96, 2011.
[11] 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)," High-Performance Interconnects (HOTI), pp.9–16, 2012.
[12] S. Denholm, H. Inoue, T. Takenaka, and W. Luk, "Application-specific customisation of market data feed arbitration," Field Programmable Technology (FPT), pp.322–325, 2013.
[13] "MoldUDP64 Protocol." `http://www.nasdaqtrader.com/content/technicalsupport/specifications/dataproducts/moldudp64.pdf`, 2009.
[14] 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.