Low-latency FPGA Based Financial Data Feed Handler

Robin Pottathuparambil*, Jack Coyne †, Jeffrey Allred †, William Lynch † and Vincent Natoli †
* *Reconfigurable Computing Systems Lab, University of North Carolina at Charlotte, Charlotte, NC 28223*
Email: `rpottath@uncc.edu`
† *Stone Ridge Technology, 2015 Emmorton Rd. Suite 204, Bel Air, MD 21015*
Email: {`jcoyne, jallred, wlynch, vnatoli`}`@stoneridgetechnology.com`

***Abstract*—Financial exchanges provide real time data feeds containing trade, order and status information to brokers, traders and other market makers. ITCH is one such market data feed that is disseminated by the NASDAQ exchange. The work presented in this paper describes an FPGA based ITCH feed handler and processing system. The handler, built on the Stone Ridge RDX-11 hardware platform with a combination of HDL and ImpulseC, accepts and processes ITCH packets at line speed with extremely low latency. Our implementation parses sixteen different stock symbols in the feed and generates an outbound packet on the NASDAQ one second heartbeat. The unit was tested with an artificial feed that could be adjusted in speed to simulate market surges. The system demonstrated a turnaround latency of 2.7μs with very little variation for all tested feed rates. The CPU equivalent demonstrated 38$\pm$22μs (1x rate) with a long tail illustrating the variability inherent to processing by the host O/S. The FPGA solution demonstrated ultra-low, deterministic latency and was able to continue processing data at the line rate limit.**

I. Introduction

Since the mid 1980s financial exchanges have provided information about the market through 'market data feeds'. These data feeds contain information about the participating stocks, orders and executed, canceled or updated order details. The feeds are subscribed by end users to watch market movements and to make decisions. As the number of stock symbols and the order size increases, there has been a steady increase in the data rate already in the range of gigabits per second. In order to use the fast growing feed data and to send out a decision (place a buy/sell order) packet (turnaround packet), the user needs to handle and process the data as fast as possible. To handle the growing data feeds and to process the data feed faster, we need capable computing systems.

Computing systems can be broadly classified as general purpose computing machines and custom computing machines. An example of a general purpose computing machine is a traditional processor-based system. In such a system, the network data (feed data) passes through the network stack and through the PCI bus to the cache for processing. The processor can handle data at line rate however non-deterministic latency is added by the operating system layer.

In a custom computing machine, for example in a Field Programmable Gate Array (FPGA) based system the 'market data feed' packet can be routed to the FPGA fabric with less or very low latency without involving the PCI bus or the operating system. Low-latency and on-chip buffering allow processing of the data feed at line speed in a well bounded number of clock cycles. This bounded number of clock cycles for a specific packet length makes the whole feed handling and processing application deterministic in an FPGA. Application Specific Integrated Circuits (ASICs) also fall into the category of custom computing machines with low latency processing, however the design path and the cost involved in fabricating an ASIC is large. A change in the design requires a new ASIC design followed by a significant re-fabrication cost. In an FPGA based computing system, a change in the design leads to re-configuration (re-programming), which does not incur any cost.

A financial expert requires computing platforms with low (order of few micro-seconds) and deterministic turnaround latency and a familiar coding environment for the development of proprietary algorithms. Considering the design advantages and deterministic latency for an FPGA, the coding of the 'market data feed' handler and processing algorithm for a finance expert on an FPGA is a complex task. Alternatively, if a platform was provided to write the feed handler and feed processing algorithm in 'C', the coding complexity is reduced to a great extent. The coded 'C' design can be then converted into a HDL design using a C-to-HDL conversion tool.

The work discussed in this paper describes an ultra low-latency FPGA-based architecture to handle and process a NASDAQ ITCH 4.0 'market data feed'. In order to reduce the coding complexity a C-to-HDL conversion tool, Impulse C, has been customized to be used with the RDX-11 FPGA board. The processing algorithm discussed in the paper generates a low-latency and deterministic out-bound packet with statistics related to the order prices and volume for sixteen different stock symbols. The FPGA design turnaround latency numbers are compared with a CPU based feed handler. The rest of this paper is organized as follows. The following section discusses related work and provides information on similar attempts at handling and processing 'market data feeds'. The design section describes the setup and design of the FPGA and CPU-based feed handler. The paper concludes with results.

978-0-7695-4301-7/11 $26.00 © 2011 IEEE
DOI 10.1109/FCCM.2011.50

II. Related work

This section briefly introduces previous work similar in nature to that presented here. A low latency and high throughput market data regeneration tool (Exegy Ticker Plant) was developed by Exegy Inc., [1]. The market data redistribution tool accepts the packets via an Ethernet card and the processing is done using an FPGA accelerator and is then distributed in a unicast fashion over the Ethernet and Infiniband networks. The exegy ticker plant has a mean latency of 80μs at a data rate of 2 million updates per second as benchmarked by Securities Technology Analysis Center (STAC). Activ Financial [2] developed the ActivFeed MPU ticker plant using a XtremeData XD1000 FPGA co-processor board. The co-processor board plugs directly into an AMD Opteron 64 processor socket. This enables access to the main system memory. The ticker plant has a turnaround latency of 100μs. A Options Price Reporting Authority (OPRA) FAST data feed redistribution system was built using a Celoxica AMDC accelerator card [3], which uses a Xilinx Virtex 5 LX110T FPGA for packet processing. They use Handel-C and the Hyper-Streams programming model for building the packet processing engine. The AMDC card was able to filter and process redundant pairs of market data feeds at gigabit line rates. The AMDC card was inserted in a quad core machine and one CPU was locked for polling incoming FPGA messages and another for redistribution. Their end-to-end latency was 26μs and the FPGA internal latency was 4μs.

Event processing hardware is described by Sadoghi et. al [4] for algorithmic trading, where a NetFPGA card is used to receive and process the event packet trace. Given a market event the implementation searches for a best set of financial strategies. The work describes a soft-processor-based architecture, a hardware architecture and a hybrid architecture. The hybrid architecture used block RAM to store the static strategies while the hardware architecture uses FPGA logic to build the strategies. An end-to-end latency (order of 7μs) comparison shows the hybrid architecture to be 10x faster than the PC based solution for up to 1K events. A 1.7x speed up was observed for events higher than 10K. Due to the lack of resources they were only able to demonstrate the hardware architecture for 250 events which had an end-to-end latency of 3.22 μs. Wray et. al [5] discusses a reconfigurable hardware based algorithmic trading engine designed on a Xilinx Virtex 5 XC5VLX30 chip. Six parallel instances of the “Participate” algorithm were routed on the FPGA. The hardware implementation was developed using Handel-C, Mentor Graphics DK Design Suite 5.2 and Xilinx ISE Webpack 11.1. The trading engine generates child orders from parent orders with updates from the market data. The design has high throughput which is 133 times better than the software implementation. The work does not highlight the latency between an order entering the system and leaving the system. The work also does not present the actual time taken to generate an order created on the basis of events and market updates. Most of the systems discussed above have turn around latencies ranging from 10μs to 80μs.

III. Design and Implementation

A. Experimental Setup

This section briefly describes the experimental setup used for testing the feed handlers. In order to create an optimized data feed handler and to process the ITCH feed, we designed a setup that would retransmit a day’s worth of ITCH data to the latency monitor, FPGA-based data feed handler and the CPU-based data feed handler. The block diagram of the setup is shown in Figure 1. The same lengths of CAT5 Ethernet cables are used from the generator to the Ethernet switch and from the switch to feed handler blocks. This is done to make sure the time required for the packets to travel to the feed handlers and the monitors are approximately the same. The NASDAQ ITCH feed sends out a heartbeat every second. This heartbeat is taken as the reference point for the latency measurement. A brief design overview of all the blocks and their functionality is explained in the following sections.

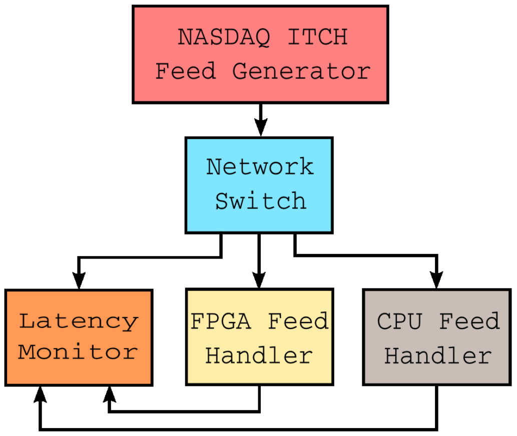

Figure 1: Block Diagram of Experimental Setup

B. NASDAQ ITCH Feed Generator

NASDAQ stands for National Association of Securities Dealers Automated Quotations. The ITCH feed falls under level 2 NASDAQ quote. The ITCH feed follows a User Datagram Protocol (UDP). An ITCH UDP packet, also called MoldUDP, contains an IP header, UDP header, payload header and payload data. The payload header contains a session ID, a packet/sequence number and the number of messages in the sequence. The packets are numbered to facilitate re-transmission of lost packets. Messages in the packet have a nanosecond time-stamp followed by a stock related message. The stock related message could be adding, updating, deleting or executing an order message. Every stock related message has a specific format and length (64 - 80 bytes). As the activity in the stock market increases (as the number of orders, updates and executes increases), the message rate also increases.

The NASDAQ ITCH feed generator transmits a day's worth of data through a gigabit Ethernet link. The ITCH data is encoded in binary format. A python script is written to replay the ITCH data and is matched with the time stamp stored in the ITCH data. The script can replay the data at rates between 1x to 8x. 1x speed is the normal time stamp matched transmission and 8x speed is eight times faster than the normal transmission. The replay speeds help to simulate market surges and test the deterministic characteristics of the FPGA and CPU feed handler implementation. The ITCH data is wrapped with a User Datagram Protocol (UDP) header and is broadcast with the help of a network switch. The network switch sends the ITCH packets to the monitor, CPU feed handler and the FPGA feed handler.

C. CPU Feed Handler

The CPU feed handler accepts data from the feed generator and parses the data. A 'C' program was written that accepts ITCH data through a UDP socket. The received data is then parsed for 'add' messages and time stamp messages. These 'add' messages contain information about a stock's buy/sell quantity and buy/sell price. A set of sixteen stocks are selected and a hash table is created for stock lookup. All the 'add' messages go through the hash table and if a match is found with the desired stock symbols (sixteen), the corresponding volume is accumulated. The average, minimum and maximum buy/sell price for the corresponding stock symbol are also computed. When a time stamp message is received the values for every stock (sixteen of them) are then transmitted as a UDP packet to the latency monitor. The CPU feed handler was designed on an Intel core 2 duo 2.8 GHz CPU with 4 GB of main memory.

D. FPGA Feed Handler

The FPGA feed handler is a PCI express based Stone Ridge RDX-11 board. The board has two Xilinx XC5VLX110T FPGAs for the user to configure and a Xilinx XC5VLX50T FPGA for the PCI bridge interface as shown in Figure 2. The ITCH feed is routed through the bridge FPGA's RJ-45 socket to its Quad Ethernet PHY. All data from the bridge (PCIe and Ethernet) goes over a bus operating at 300 MHz DDR using the ISERDES and OSERDES capability of the IOBs. 19 LVDS pairs are used for data, 3 are used for control, and one for the clock. Data from PCIe and Ethernet are buffered on the bridge, and then multiplexed over the bus, with Ethernet taking priority. The bus has sufficient bandwidth to support full 1 Gbps on all Ethernet ports simultaneously (4 Gbps total), with additional bandwidth remaining for PCIe traffic. There is one bus in

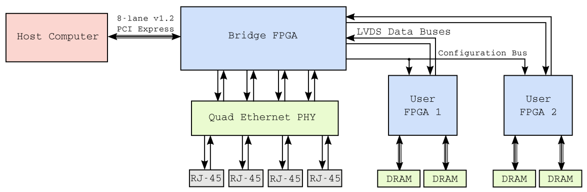

Figure 2: Block Diagram of the Board

each direction to support full duplex. A filter is designed on the user side FPGA to examine the Ethernet packets that are received from the bridge. The design on the bridge FPGA was synthesized, placed and routed using Xilinx Integrated Software Environment (ISE) 11.2 and consumed about 85% slices and 98% BRAM blocks.

Each Ethernet frame (ITCH data) that is received across the bridge is examined by the filter to determine if it is a UDP/IP packet and destined for the correct port number. Valid frames are passed on to the next stage while undesired frames are ignored. Frames that arrive at the selected UDP port contain ITCH feed information in the form of MoldUDP messages. Each packet may contain multiple messages, which must be unpacked and processed individually. Each moldUDP message consists of a message length (16 bits) and a message payload. The first message is located immediately after the packet header. The length of each message is loaded into a second counter so that the beginning of the next message can be located. The first field (1 byte) lists the message type. Only messages with type 'T' (time stamp) or type 'A' (add order) are desired, and all other messages are ignored. When a 'T' (time stamp) message arrives, the timestamp field (4 bytes) is extracted and passed to the ImpulseC application. For 'A' (add order) messages, the following fields are extracted. Stock name (6 bytes), stock prices (4 bytes), number of shares (4 bytes), transaction type buy or sell (1 byte). These fields are written to a set of FIFO buffers for processing by the next stage. Only 16 stock symbols are desired for the current implementation. In the next stage, for each message that is retrieved from the FIFO buffer the stock name is examined. Four DSP slices are used in parallel, such that 4 clock cycles are required to determine if the stock symbol is desired. This implementation is not as efficient as a hash table method, but is easily modifiable and uses otherwise idle on-chip resources. Matching stock symbols are represented with a 4-bit index field. The volume field is reduced to 3 bytes, and the buy/sell indicator is reduced to 1 bit. An additional bit is used to indicate whether the original message was an 'A' or 'T'. All fields are packed into a 64-bit word and passed from the filter core to the ImpulseC application module as a stream.

The ImpulseC application waits on the stream data. As the first data is received by the application, it looks for the first bit and determines whether the 64-bit word is an add order or a time stamp message. If the 64-bit word is an add order, the application reads the stock value (0 to 15) and then accumulates the volume, computes high, low and average buy/sell price for the respective stock. The application stores the accumulated values for sixteen different stock symbols until a time stamp message is received. As soon as a time stamp message is received the ImpulseC application packs the stored information for sixteen stocks into a single packet and sends it back to the filter block as a stream. The filter block receives the data from the ImpulseC stream and adds the UDP header, IP header and frame header and sends it to the Ethernet transmit FIFO. The transmit FIFO sends the data to the bridge FPGA and the bridge FPGA transmits the packets to the latency monitor. The design on the user FPGA along with the ImpulseC generated HDL was synthesized, placed and routed using Xilinx Platform Studio (XPS) 11.2 and consumed about 53% slices and 80% BRAM blocks.

E. Latency Monitor

The latency monitor measures the latency of the packets from the CPU feed handler and the FPGA feed handler. The NASDAQ ITCH heart beat (time stamp message) is considered as the reference message. The latency monitor also has a Stone Ridge RDX-11 board for receiving the packets from the CPU feed handler, FPGA feed handler and from the NASDAQ ITCH feed generator and transmits the data to the User 1 FPGA as shown in Figure 2. The User 1 FPGA computes the latencies between the reference (NASDAQ ITCH feed heart beat), CPU feed handler out-bound packet and FPGA feed handler out-bound packet. The latencies are then transmitted to the PCIe endpoint core. The PCIe endpoint core reads the FIFO data and transmits it to the host. A Matlab application was written to display the computed latencies.

IV. Performance Tests and Results

The NASDAQ ITCH feed generator was designed for generating the ITCH feed between normal speed (1x) and 8x speed. A peak feed rate of 120 Mbps and 420 Mbps was observed when the feed was transmitted at 1x speed and at 8x speed respectively. The peak feed rate at 8x speed should be 960 Mbps, but due to the inherent latency of the NASDAQ ITCH feed generator software, we were only able generate a peak feed rate of 420 Mbps at 8x speed from the feed generator. The peak feed rate for 4x, 5x, 6x and 7x also got saturated at 420 Mbps. The latency response of the CPU and FPGA based feed handler is shown in Figure 3. As we can see from the figure the latency of the FPGA based feed handler is deterministic and that of CPU-based feed handler varies between 38±22μs. The feed rates that are greater than the normal (1x rate) simulate volume surges in the market. The CPU-based feed handler clearly shows a non-deterministic latency, however the FPGA based feed handler shows a deterministic ultra low-latency of 2.7μs. The filtering algorithm filters the data from the feed and passes the filtered data to the ImpluseC application. The filtering algorithm was able to handle a data rate of 11 Mbps and around 36 Mbps at normal and at faster rate respectively.

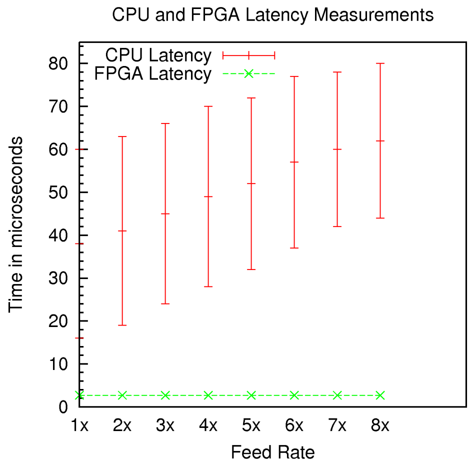

Figure 3: CPU and FPGA Latency Plots

V. Conclusion

The paper describes the design and implementation of an ultra low-latency financial feed handler. The implemented FPGA design was able to handle data rates of 120 Mbps and a peak data rate of 420 Mbps. The CPU-based design showed a non-deterministic latency of 38±22μs (1x rate), which increased with market data feed rates and the FPGA based feed handler had a deterministic latency of 2.7μs regardless of feed rates. The work demonstrates the advantages of FPGAs as part of financial feed handling system providing ultra-low deterministic latency to end users.

References

[1] "Exegy, Inc." July 23, 2007, `www.exegy.com/PDFs/PressReleases/STAC_pressrelease.pdf`.

[2] "ActivFeed MPU," June 6, 2008, `www.activfinancial.com/docs/ActivFeed%20MPU.pdf`.

[3] G. Morris, D. Thomas, and W. Luk, "FPGA accelerated low-latency market data feed processing," in *High Performance Interconnects, 2009. HOTI 2009. 17th IEEE Symposium on*, August 2009.

[4] M. Sadoghi, H.-A. Jacobsen, M. Labrecque, W. Shum, and H. Singh, "Efficient event processing through reconfigurable hardware for algorithmic trading," *PVLDB*, vol. 3, no. 2, pp. 1525–1528, 2010.

[5] S. Wray, W. Luk, and P. Pietzuch, "Exploring algorithmic trading in reconfigurable hardware," *21st IEEE International Conference on Application-specific Systems Architectures and Processors (ASAP)*, p. 325, 2010.