“Pocket-Sized” High Frequency Trader (PSHFT)

Gabriel Blanco - gab2135
Brian Bourn - bab2177
David Naveen Dhas Arthur - da2647
Suchith Vasudevan - sv2340

Table of Contents

1. Introduction

High Frequency Trading is a primary form of algorithmic trading that uses sophisticated technological tools and computer algorithms to trade financial securities. HFT uses proprietary automated trading strategies on large samples of data to make many many small transactions in fractions of a second. With HFT, the two most important

With the fastest execution speeds will be more profitable than traders with slower execution speeds. As of 2009, it is estimated that more than 50% of exchange volume comes from high frequency trading orders. High frequency traders move in and out of short term positions at high volumes aiming to capture even fractions of a cent in profit on every trade. HFT became most popular when exchanges began to offer incentives for companies to add liquidity to the market.

Electronic trading of stocks is conducted by sending orders in electronic form to a stock exchange. Bid and ask orders are then matched by the exchange to execute a trade. Outstanding orders are made visible to the market participants through ‘feeds’, which are compressed or uncompressed real time data streams provided by an independent institution like the Options Price Reporting Authority (OPRA). A feed carries pricing information of stocks and is multicasted to the market participants using standardized protocols which are generally transmitted over UDP over the Ethernet. The standard protocol that is applied is the Financial Information Exchange (FIX) protocol Adapted for Streaming (FAST) which is used by multiple stock exchanges to distribute their market data.[1]

To enable minimal round trip latencies, a HFT engine needs to be optimized on all levels. The required low latency connection to the feed handler can be achieved through collocation which allows servers to be deployed very close to the stock exchange. In addition, the feed needs to be internally distributed with minimum latency to the servers of the HFT firm. An efficient decoding of the UDP data stream as well as of the FAST protocol is mandatory. Finally, the decision to issue an order as well as its transmission needs to be carried out with lowest possible latency. To achieve these, HFT trading accelerator engines can be implemented in Field Programmable Gate Arrays (FPGAs). By using FPGAs, we can offload UDP and FAST decoding tasks from the CPU to optimized hardware blocks. Using FPGA’s shows a significant latency reduction of more than 70% compared to the standard software solution while maintaining the flexibility to support new and modified exchange protocols with low efforts in contrast to an Application Specific Integrated Circuit (ASIC) solution[1]

This project aims to implement an FPGA engine that takes in market spread data as its input, calculates the tiny discrepancies between fair spread values and current market spread values in parallel, and makes order decisions based on these discrepancies.

Market Data Terminology

Future:

A future is a contract to sell or buy a commodity at a later date, at a price agreed upon well in advance. Futures serve as a form of insurance for many industries so that a company or individual can guarantee a price for their goods when they come to fruition. This tactic is frequently employed by the Agricultural Industry and the Oil Industry, locking in prices for crops or oil to be delivered in the distant future, months or years away.

Spread Trade (Spread):

A spread is a simultaneous purchase and sale of multiple related commodities, in this case futures, as a unit. The relation can vary from length of contract to company offering the future. The volatility of spreads is generally much lower than that of individual futures but as a result the generally provide less yield when traded and sold.

Delta:

The difference between calculated value of a financial instrument (from a financial model) and it's real world market price.

2. Design

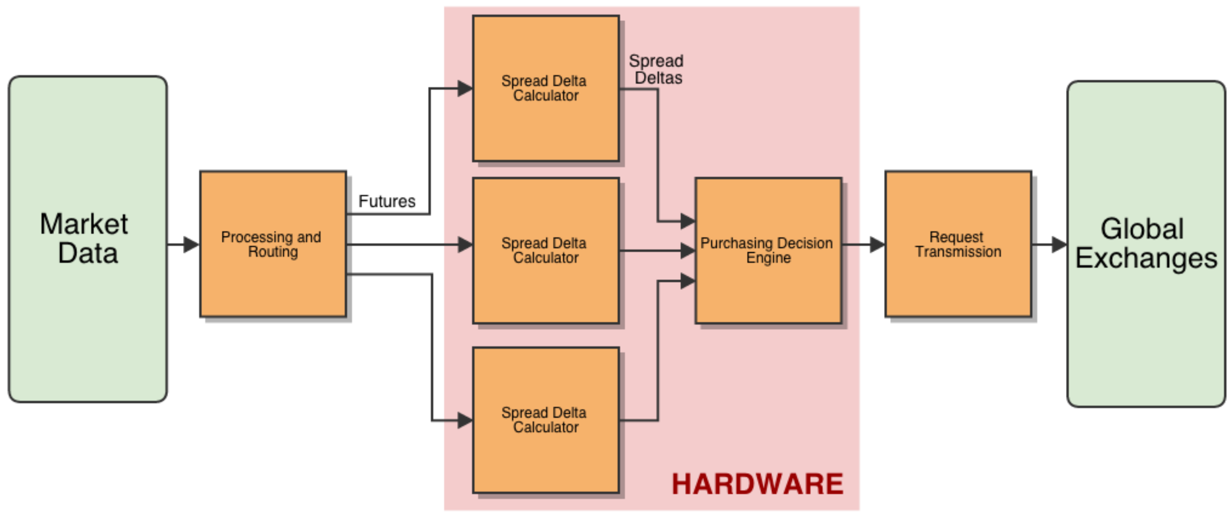

Figure 1: Block Diagram Overview

We use hardware acceleration in order to calculate the momentary discrepancies between Spreads as quickly as possible, and since this task embarrassingly parallelizable, we can also have scalable number of spread delta calculation modules.

We receive, process and route the Market Data on futures and spreads in software, and pass that information into our spread data calculators. A decision engine evaluates whether it is beneficial to make an order request based on the calculated marginal discrepancies, and writes to a set of registers in user space, which can be read and sent to the global exchange.

3. Implementation

Market Data Processing and Routing:

In a live implementation of an algorithmic trading HFT, we would have an open TCP connection where market data is continuously being updated. In order to simulate the flow of new trading information, we will instead use the Linux File IO libraries, which read from a file as a stream, similar to the way that TCP works.

The test data suite, stored as a Linux file, will simulate the rising and falling of various futures and spreads in order for the HFT algorithm to do the necessary calculations to determine whether or not to make purchase or sale of that given future.

The parsing of the file will be done in software, but then the C program will communicate with the delta calculation modules in order to route the correct information in order to calculate the various deltas.

Spread Delta Calculation:

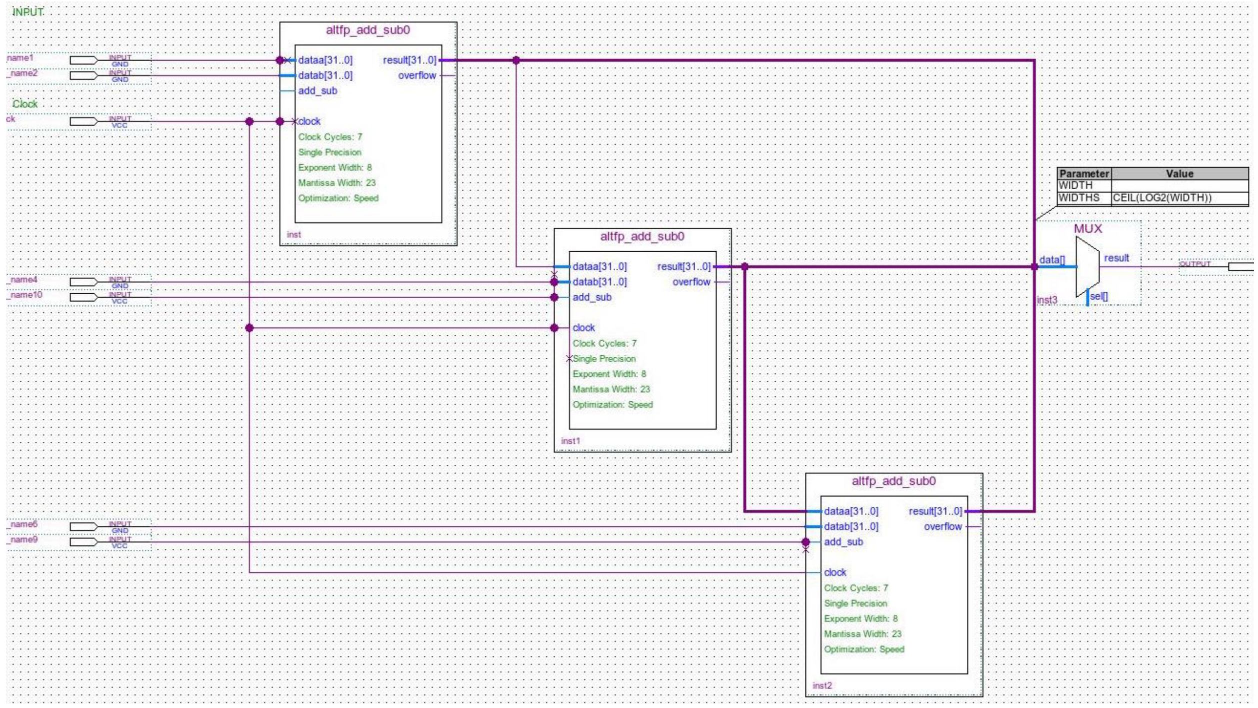

Figure 2: Design schematic built using Altera Quartus

In order to calculate the delta between determined value and market price the data is sent into a set of cascaded floating point arithmetic modules. The output from each of these modules is then fed into a multiplexer at the various stages in order to provide the correct delta based upon number of inputs used by the current calculation. The output of the Spread Delta Module is then a floating point representation of the current delta of the financial instrument. This module will be implemented on hardware in order to accelerate the calculations, receiving data from the registers used in the software level.

Module:

```
SpreadDeltaCalculator (
   input logic clk, reset,
   input logic [31:0] in1, in2, in3, in4,   -- inputs
   input logic as2, as3, as4,               -- add/subtract
   input logic [1:0] is                     -- input select
   output logic delta
);
```

Each SpreadDeltaCalculator contains 3 Floating Point Adder/Subtractor modules (taken from Quartus libraries).

Purchasing Decision Engine:

The Spread Data Calculators send the spreads in parallel to the Purchase Decision Engine, where the each spread is compared against a minimum value. The result (or delta) from each comparison is written to a pre-programmed register, and the deltas are indexed to market spreads (eg. AAPL - MSFT) through a lookup table. Each register has a 1 written to it if the spread is greater than the minimum value, in which case a purchase order is executed if the spread is positive, or a sell order is executed if the spread is negative. The register has a 0 written to it if the spread is less than or equal to the minimum value. This is how buy and sell orders are generated.

Request Transmission:

Purchasing decisions written by the Purchase Decision Engine to registers are then read into Software. From there, you look them up from a table (as mentioned above). This lookup table contains the associated index, name, account, ID, type (buy or sell), and quantity of each spread processed. Using this information, these requests are then transmitted to the CME exchange.

The Data Flow Engine (DFE) is connected to the CME exchange via a network switch and our trading strategy interfaces with the handler via the framed stream input and output in hardware and the software API[2]. The Financial Information Exchange (FIX) requests from the DFE are encoded and their corresponding decoding is implemented on Hardware. This implementation is done using the Maxeler MaxCompiler and MaxIDE design tools. Maxeler allows a ultra-low latency implementation

Hardware:

During the encoding phase, the Encoder converts the data frames to FIX messages. The Decoder decodes FIX messages to a specified output frame.

Software:

The sequence number is generated in the software for every outbound FIX message. An event monitoring API is also implemented in the software that allows monitoring of the state of the handler

Message Arbitration:

The message arbitration unit combines the hardware and software streams. This is now combined with a Sequence number that for every outbound FIX message.

This is now passed through a TCP Handler module that communicates with the CME exchange.

Maxeler Framework:

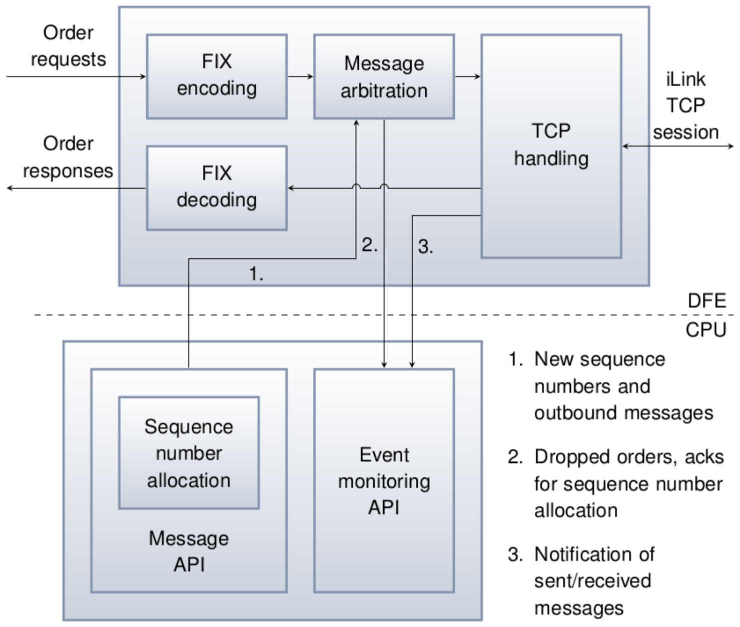
Order requests
FIX encoding
Message arbitration
TCP handling
iLink TCP session
Order responses
FIX decoding
1.
2.
3.
DFE
CPU
Sequence number allocation
Message API
Event monitoring API
1. New sequence numbers and outbound messages
2. Dropped orders, acks for sequence number allocation
3. Notification of sent/received messages

Figure 3: Block diagram of the Maxeler CME iLink Handler.

4. Deliverables

Milestone 1 (04/02/15) :

- Delta Spread Calculation Module in Quartus
- Simulated Market Data Input from Linux File IO

Milestone 2 (04/14/15) :

- Purchasing Decision Engine in Quartus
- Register Interfacing between Hardware/Software

Milestone 3 (04/28/15) :

- Request Ordering using Maxeler Framework

Final Deliverable (05/14/15) :

- Functional, Fully Tested HFT Simulation

5. References

1. *High Frequency Trading Acceleration using FPGAs*, University of Heidelberg
2. CME Globex Link Data sheet