Ticker Plant System Implemented in Max Compiler

Gabriel Blanco (gab2135@columbia.edu)
Brian Bourn (bab2177@columbia.edu)
Suchith Vasudevan (sv2340@columbia.edu)
David Naveen Dhas Arthur (da2647@columbia.edu)

Guided by

Prof. David Lariviere
Prof. Stephen Edwards
Department of Computer Science,
Columbia University,
New York

May 14th, 2015

Table of Contents

1 Abstract

Our project aims to implement a Ticker Plant on a Stratix V FPGA running on a Maxeler MAX4N board. Our program, written using the Max Compiler, receives a simulated input stream of market data (futures and calendar spread prices) that is similar in structure to real world market data, and aggregates them in the on-chip memory. The FPGA then actively sends out a completed book of known instruments, as well as the implied bidding and asking prices of those instruments, calculated by using calendar spread data.

2 Introduction

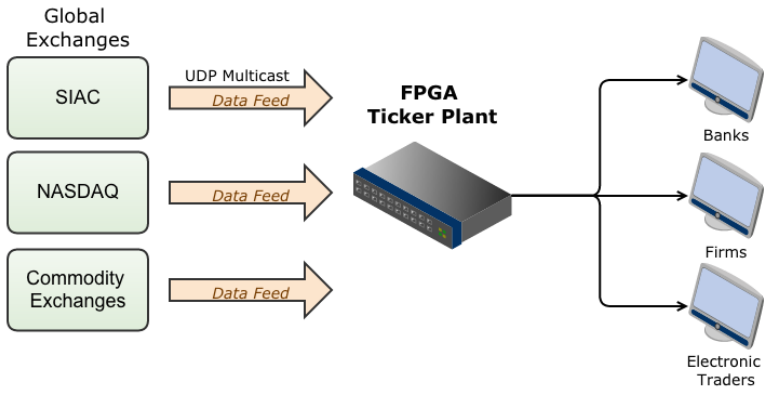


Fig. 1: Ticker Plant

If a bank or another firm wants a real-time order book for a specific set of instruments and spreads, instead of parsing live market data from a global exchange, they will use a ticker plant to reduce the data load and overhead. Ticker plants consumes large amounts of market data and output an order book that is filtered and normalized.

Modern electronic markets are high packet rate systems, so in order to mitigate the risk of losing any information, ticker plants need to be low-latency, deterministic, and most importantly: reliable. Software implementations are not as fast as hardware solutions; they are often susceptible to packet loss as well as long latency tails caused by serialization delays. The legacy x86 architecture that is found in most personal computers was never intended to be used for real time data transmission. It is for that reason that it is much more suitable to build a ticker plant using an ethernet enabled FPGA.

For example, if a ticker plant running at a full network load (40 Gb/sec) goes down for a millisecond, and we assume that the average UDP packet is about 1 kilobyte, that would mean a potential loss of almost 50,000 packets of information. With uptime being so important, hardware implementations of ticker plants are a necessity for the modern marketplace.

.

2.1 Financial Exchanges

The market data that is collected by the ticker plant comes from one of the many financial exchange, each of which has different protocols and format, dealing with a wide array of trades.

Examples of modern financial exchanges are:

- Chicago Mercantile Exchange
- Bombay Stock Exchange
- Dubai Mercantile Exchange
- Moscow Exchange

These financial exchanges are responsible for 'clearing' all financial transactions that they service. In banking and finance, clearing includes all activities from the time a commitment is made for a transaction until it is settled. Clearing of payments is required to turn the promise of payment into actual movement of money. In trading, clearing is necessary because the speed of trades is much faster than the cycle time for completing the underlying transaction. Central Counterparty Clearing is a process by which financial transactions in equities are cleared by a single ("central") counterparty. A financial exchange is a central clearing counterparty which bears all default risk on all transactions that they service.

3 System Overview

Our system begins by simulating market data. Using a Python script, we generate a large sample size of randomized packets of market data which include the Instrument ID, Side, Level, Price and Quantity of individual instruments, which is then stored in a .csv file. In software, we load and parse that csv and send each of the instrument packets as a package of data via UDP, simulating how we would otherwise be receiving the data from a global exchange.

In hardware, our 10-gigabit ethernet enabled FPGA receives the UDP messages and parses them as input frames. Using a series of data flow logic, we parse the various parameters of the input frames to find where to store the pricing and quantity data in registers of the hardware. Using the most up to date values of the instruments stored in register on the on-chip memory of the FPGA, we calculate the "implied" instruments. Knowing the bidding and asking prices of spreads (which are made up of two or more legs), we can then create these implied orders based on the actual prices of the other two instruments.

In hardware, we then aggregate all of the known and implied instruments into an output frame. We then send the data via UDP as a single package in a combined "book". That information could then be read and acted upon by an electronic traders.

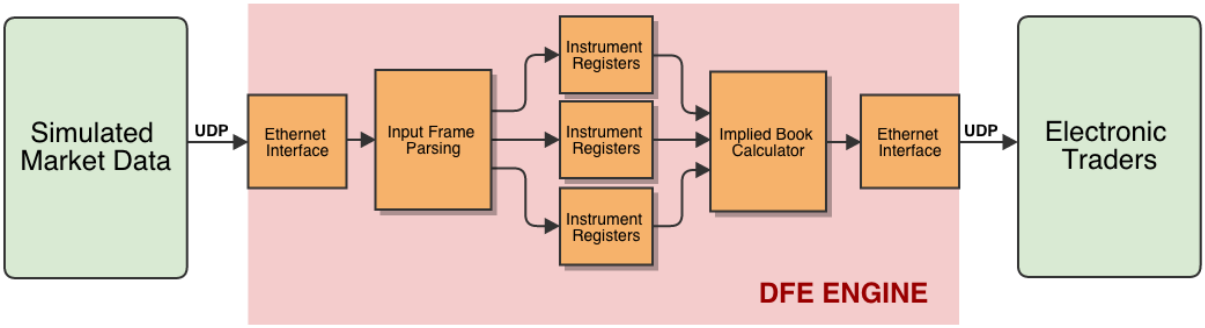


Fig. 2: Project Workflow

3.1 CPU Code

In software, we simulate the flow of input data coming from a market data source, configure the IP and UDP socket for the engine, and connect to it with a CPU side socket. Since we are able to control both the input as well as receive the output of the Maxeler engine, we can validate the results of the calculation in software by mirroring the same calculations (albeit much slower) and testing to see if the real output matches our expected output.

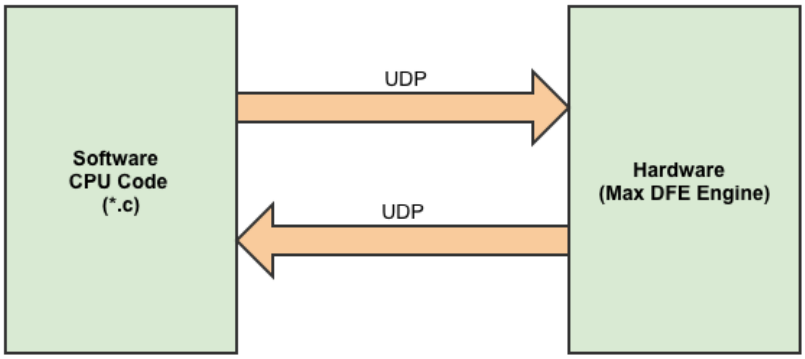


Fig. 3: Run-time Software Hardware Interaction

3.1.1 Market Data Simulation

When receiving new stock information from a market data exchange such as GlobEx, the information is sent as individual instrument packages in XML. The data is sent by *IP Multicast* in UDP because dropped packages should not inhibit the flow of new data.

The data feed implementation used by the CME Group is called MDP 3.0. An MDP 3.0 packet contains the following:

- *Packet Header:* Contains packet sequence number and sending time.

- *Message Size:* field indicating size of message
- *Message Header:* contains block length, Template ID, Schema ID, and Version
- *FIX header:* Indicates FIX message type
- *FIX message body:* event driven business data such as book updates and trade summary

Our simulated input data contains just the packet sequence number and message body.

We generate input data using a python script which generates randomized input data for the known instrument IDs. In our simulation, to simplify the parsing of the input data, we format our input data as Comma Separated Values (*.csv) instead of XML. In our Maxeler CPU code, we build a CSV parsing function that receives the CSV file, parses it line by line and converts it into a usable package. In this way, we are modeling an incremental data feed.

The data format that we are using is a simplified variant of the actual fields sent by an exchange. We have chosen to use these fields in our input data:

- *Instrument ID*
- *Side*
- *Price Level*
- *Price*
- *Quantity*

Each parameter is described in the Terminologies section. For more information on the implementation of our random input data and an output CSV File, see:

generate_input.py
source_data.csv

3.1.2 UDP Networking

We create payload packages for both the input from our incremental feed as well as for the output of the ticker plant. We are sending messages by UDP as packages formatted with the five parameters listed in the section above, and are receiving an output comprised of an order book made up of six instruments. The packet size of the input data is 20 Bytes, but the corresponding output is 120 Bytes, six times as large as the input but still relatively large.

UDP (*User Datagram Protocol*) is a networking protocol described in our Terminology section. In order to create a UDP connection between the CPU and the Engine, we need to create two separate sockets UDP sockets, one on the DFE IP and one with the CPU IP, on an unused port.

This is the relevant section from the CPU code:

```
/* Create DFE Socket, then listen */
const int port = 5008;
max_file_t *maxfile = FieldAccumulator_init();
max_engine_t *engine = max_load(maxfile, "*");
max_ip_config(engine, MAX_NET_CONNECTION_QSFP_TOP_10G_PORT1, &dfe_ip, &netmask);
max_udp_socket_t *dfe_socket = max_udp_create_socket(engine, "udp_ch2_sfp1");
max_udp_bind(dfe_socket, port);
max_udp_connect(dfe_socket, &cpu_ip, port);
```

3.1.3 Result Validation

In order to check that our calculations are correct, we also perform the same operations in parallel on the CPU. Instead of registers we use static variables and we use ternary operators similar to how they are implemented in Kernel. We output this to an output package, and compare each value in the received package and the "expected" package generated by

This was very useful in debugging and testing the engine code, making sure that the order book that is returned is correct. In practice, when tackling a high throughput we will turn off the validation code, since that would only slow down the ticker plant algorithm, but with low quantities of data it is useful to have.

3.2 Data Flow Engine

Data Flow Engine (DFE) is the name of a paradigm in the MaxCompiler synthesis language for writing optimized hardware code. This abstraction makes it easier to write hardware code without the need for considering timing closures and cycle level verification. The DFE is comprised of a Manager which orchestrates data flow and one or more Kernels which are responsible for implementing hardware computation. MaxCompiler generates dataflow implementations which can then be called from the CPU via the SLiC interface.

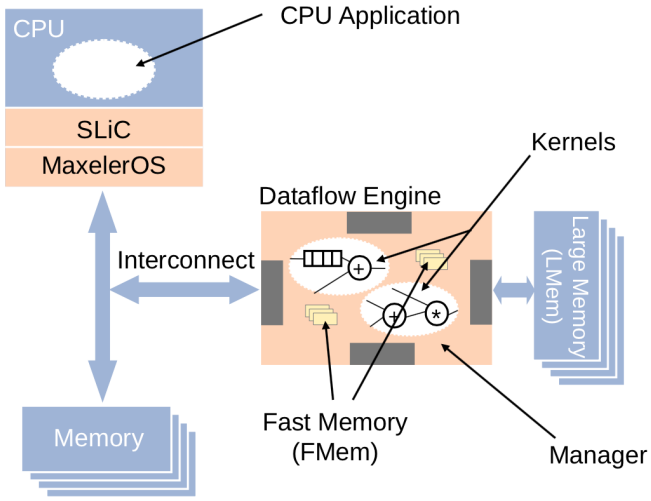


Fig. 4: Dataflow Engine Architecture

3.3 Kernel

The Kernel is responsible for performing calculations on the hardware. In our implementation, there is only one Kernel responsible for parsing frames, calculating implied orders and generating the order book as an output frame.

3.3.1 Input Frame Parsing

Frames arrive via the UDP stream, and we define the input frame format within the Kernel:

```
static class
DataIn extends FrameFormat
{
    DataIn()
    {
        super(ByteOrder.LITTLE_ENDIAN);
        addField("instrument_id", dfeInt(32));
        addField("level", dfeInt(32));
        addField("side", dfeInt(32));
        addField("quantity", dfeInt(32));
        addField("price", dfeInt(32));
    }
}
```

Based on the values of the *instrument_id*, *level* and *side* fields, we route the incoming message into one of the predeclared registers associated with that instrument. If no new data arrives, the value in the register is held so that a register only updates when new data with its specific routing information arrives.

3.3.2 Calculating Implied Orders

After a new data value arrives, we actively recalculate implied orders. Implied orders are orders that can be built out of existing instruments. For example, take a calendar spread which involves the simultaneous purchase of a future, A, and the sale of another later future, B. Even if you don't have any bidding prices for that calendar spread "A-B", you can generate an implied bid for that spread by looking at the top-level bids for A and the top-level asking price of B. The difference between those two prices is the price of the implied bid of "A-B", and the quantity is the minimum between "A" and "B".

Not only can you generate implied bids for spreads, but you can also calculate implied bids for individual instruments as well if you know the calendar spread value as well as the bidding or asking price of the other leg in the spread.

Here are the algorithms we are using in calculating the implied orders in Kernel. Given “A-B”, a spread which involves the simultaneous purchase of instrument A and sale of instrument B, the implied prices and quantities are:

Implied Bid Price of A = “A-B” Bid Price + “B” Ask Price
Implied Bid Quantity of A = min(“A-B” Bid Quantity, “B” Ask Quantity)

Implied Ask Price of B = “A” Bid Price - “A-B” Bid Price
Implied Ask Quantity of B = min(“A” Bid Quantity, “A-B” Bid Quantity)

Implied Bid Price of A-B = “A” Bid Price - “B” Asking Price
Implied Bid Quantity of A-B = min(“A” Bid Quantity, “B” Asking Quantity)

3.3.4 Output Format

The output data is six times as large as the input data, and is sent to the ethernet module. It involves the fields in DataOut() displayed below.

```
static class
DataOut extends FrameFormat
{
    DataOut()
    {
        super(ByteOrder.LITTLE_ENDIAN);
        addField("a_bid_instrument_id", dfeInt(32));
        addField("a_bid_level", dfeInt(32));
        addField("a_bid_side", dfeInt(32));
        addField("a_bid_quantity", dfeInt(32));
        addField("a_bid_price", dfeInt(32));

        addField("ai_bid_instrument_id", dfeInt(32));
        addField("ai_bid_level", dfeInt(32));
        addField("ai_bid_side", dfeInt(32));
        addField("ai_bid_quantity", dfeInt(32));
        addField("ai_bid_price", dfeInt(32));

        addField("b_ask_instrument_id", dfeInt(32));
        addField("b_ask_level", dfeInt(32));
        addField("b_ask_side", dfeInt(32));
        addField("b_ask_quantity", dfeInt(32));
        addField("b_ask_price", dfeInt(32));

        addField("bi_ask_instrument_id", dfeInt(32));
        addField("bi_ask_level", dfeInt(32));
        addField("bi_ask_side", dfeInt(32));
        addField("bi_ask_quantity", dfeInt(32));
        addField("bi_ask_price", dfeInt(32));

        addField("ab_bid_instrument_id", dfeInt(32));
        addField("ab_bid_level", dfeInt(32));
        addField("ab_bid_side", dfeInt(32));
        addField("ab_bid_quantity", dfeInt(32));
        addField("ab_bid_price", dfeInt(32));
```

```
            addField("abi_bid_instrument_id", dfeInt(32));
            addField("abi_bid_level", dfeInt(32));
            addField("abi_bid_side", dfeInt(32));
            addField("abi_bid_quantity", dfeInt(32));
            addField("abi_bid_price", dfeInt(32));
        }
    }
```

3.3.5 Framed Kernel

Network protocols enable interoperability between applications on a wide range of hosts and across a wide range of networks. This requires a rigid set of interfaces and a simple way to communicate both data and metadata to the network and remote hosts. To this end, small chunks of data known as "frames" or "packets" are combined with "headers" from the different network layers before being transmitted. These headers typically have a fixed or semi-fixed set of "fields" at well-known offsets from the start of the header. The combination of in-band control, fixed-position fields, variable length fields and unpredictable arrival time presents unique challenges to an otherwise static Dataflow model. "Framed Kernels" in Maxeler greatly simplify dealing with data frames of varying length, and dealing with fields in an intuitive way, regardless of the size of the underlying DFE link.

In a Framed Kernel, network data is considered as a stream of 'frames'. Frames are typically received on a Kernel input stream, processed in a suitable way and transmitted through a Kernel output stream. Each frame consists of a fixed set of fields. Fields have familiar types, such as floating point or integer numbers and may be fixed or variable size. The order of the fields in a frame as well as their type and name is typically application-dependent and defined by the developer as a "frame format" which can be associated with Kernel input or output streams.

A frame format may describe frames whose total size exceeds the width of the carrier stream. In this case, sufficiently small segments of the frame are streamed sequentially and MaxCompiler automatically manages the marshalling and unmarshalling of the segments.

We use framed kernels to employ an input stream of simulated futures prices, and an output stream of orders. Our input and output frames contain fields for *instrument ID*, *side*, *level*, *quantity*, and *price*.

3.3.6 Kernel Graph

A kernel is a streaming core with a data flow described by a unidirectional graph without cycles. Kernel graphs describe several different node types:

Computation Nodes
They perform arithmetic and logic operations (e.g., +, *, <, &) as well as type casts to convert between floating point, fixed point and integer variables.

Value Nodes
They provide parameters which are either constant or set by the host application at runtime.

Stream Offsets
They allow access to elements at different positions in data streams.

Multiplexer Nodes
They are nodes for decision making.

Counter Nodes
They are for catching specific stream positions such as boundary conditions.

I/O Nodes
They connect the kernel to the manager and serve for streaming data in and out.

The Kernel graph for our Kernel generated by Maxeler is displayed below:

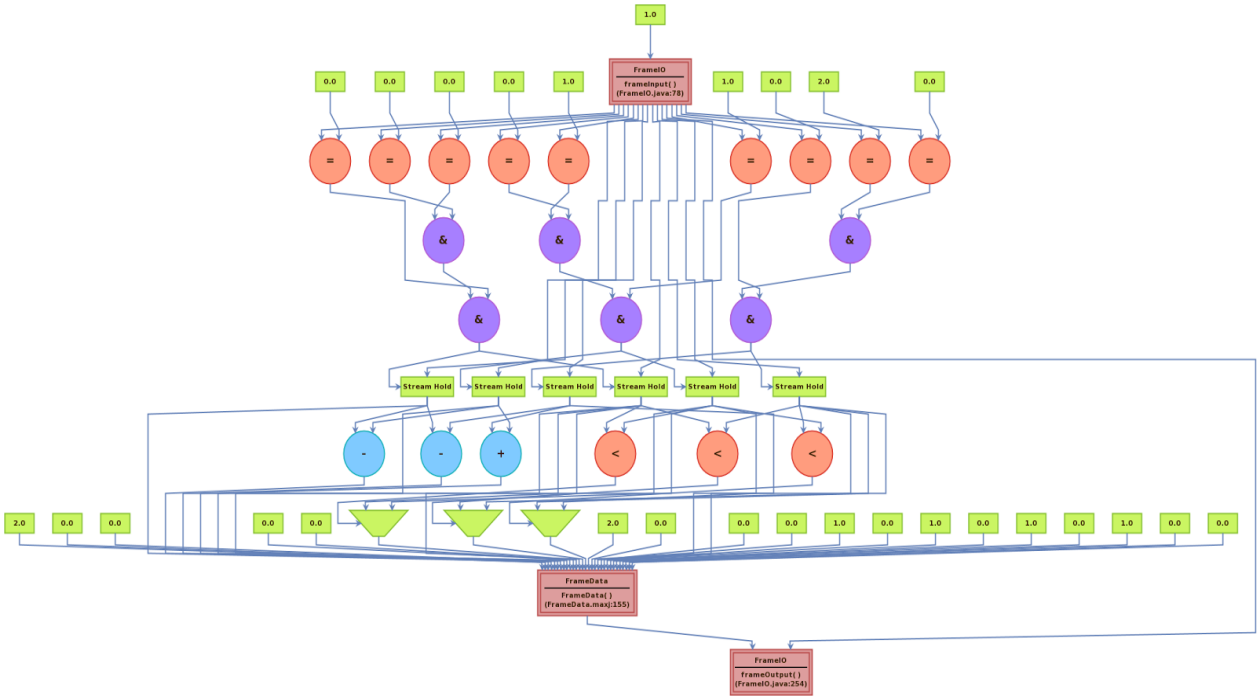


Fig 5. Kernel Graph

3.3 Manager

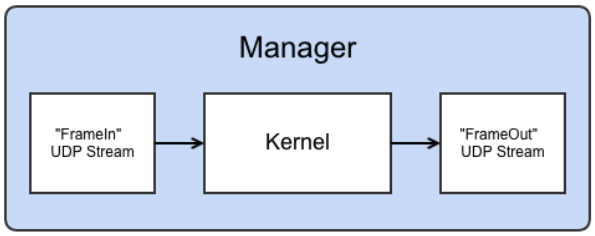


Fig. 6: Max Manager

The manager wraps kernels and orchestrates their data I/O. It also connects Kernels to the CPU, engine RAM, other Kernels and other dataflow engines. The Manager, in addition to defining build configurations, also declares:

- "FrameIn" UDP Stream
- "FrameOut" UDP Stream
- "FieldAccumulator" Kernel

4. Tool Chain

The Maxcompiler tool flow abstracts the cumbersome Quartus tool flow to simple specification of the Kernel and the kernel manager.

4.1 Maxeler Framework

The Maxeler framework targets the Maxeler FPGA development platforms (Maxeler DFE accelerator platforms) that posses the Altera Stratix V FPGAs that posses over 234,720 Adaptive Logic Modules (ALMs) and over 2,560 M20K blocks offering maximum on chip memory and tremendous flexibility.

In the Maxeler framework, an extension of Java (MaxJ) is used to design the hardware modules (Kernel) where a majority of the computation is intended to happen. The Maxeler framework also has a well defined set of Computational and Networking libraries to perform computations. The Dataflow Engine (DFE) in dataflow programming refers to the hardware implementation by Maxeler. This normally takes the input as a stream. These engines contain small amounts of local memories that are attached by an interconnect to the CPU.

This is analogous to the Altera Quartus interface where:

Hardware Interface	Data Flow Engine (DFE), Maxeler files and the Kernel Manager
Software Interface	*.c files

One of the major advantages of using Maxeler is less compile time required for simulation. The simulation run by Maxeler almost assures that the design is synthesizable and good to be implemented on the hardware. The direct integration with the Quartus fitter and Assembler tool flow eases the tool chain where the Maxeler framework first compiles the Hardware (Max files) into the equivalent VHDL files, then the Host CPU code.

4.2 Max Workflow

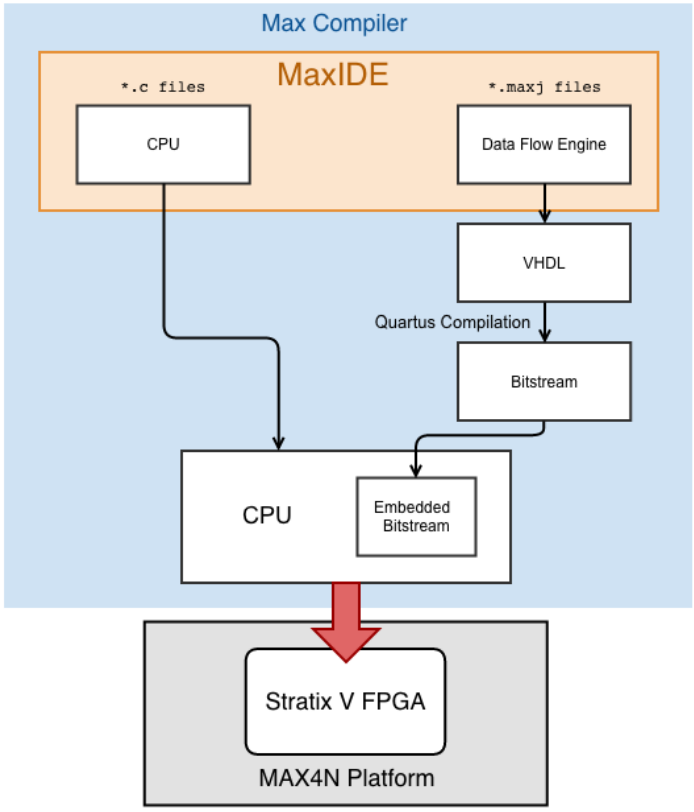


Fig. 7: Compilation Workflow

1. MaxJ Compilation

A hardware build automatically runs FPGA synthesis and backend tools and generates a file with the FPGA configuration bitstream. The hardware build process further provides reports with estimates on hardware resource usage and data for making performance projections. The host application is compiled and linked with the .max file and the software part of MaxelerOS to the application executable. This executable includes all the code necessary to deal with the acceleration hardware, such as downloading the FPGA configuration and setting up the required data flows between CPU, FPGAs and memories. MaxIDE compiles the MaxJ kernel and manager files into a massive number of VHDL files (267 *.vhdl) . A subset of files is shown below

```
-rw-rw-r--. 1 sv2340 sv2340 634K May 12 20:07 fieldSwapKernel.vhdl
-rw-rw-r--. 1 sv2340 sv2340 133K May 12 20:07 FPGAWrapperEntity_Manager_FieldAccumulator.vhdl
-rw-rw-r--. 1 sv2340 sv2340 117K May 12 20:07 MAX4FPGATop.vhdl
-rw-rw-r--. 1 sv2340 sv2340 103K May 12 20:07 Manager_FieldAccumulator.vhdl
-rw-rw-r--. 1 sv2340 sv2340  78K May 12 20:07 MAX4NPeripheryTop.vhdl
```

Quartus creates a project file named MAX4NPeripheryTop.qpf

2. Quartus Synthesis

Quartus performs complete Analysis & Synthesis on a design, including technology mapping. Analysis & Synthesis performs logic synthesis to minimize the logic usage of the design, and performs technology mapping to implement the design logic using device resources such as logic elements. Finally, Analysis & Synthesis generates a single project database integrating all the design files in a design.

3. Placing & Routing

The Quartus Fitter matches the logic and timing requirements of the project with the available resources of a Stratix V FPGA in the Max4N board. The Fitter assigns each logic function to the best logic cell location for routing and timing, and selects appropriate interconnection paths and pin assignments.

4. Quartus Assembler

The Quartus Assembler is the Compiler module that completes project processing by generating a device programming image of the Stratix V FPGA.

5. TimeQuest Timing Analysis

Maxeler runs an extensive timing analysis to check for timing closure using the TimeQuest timing analyzer. The Quartus II TimeQuest Timing Analyzer is a powerful ASIC-style timing analysis tool that validates the timing performance of all logic in your design using an industry-standard constraint, analysis, and reporting methodology. The TimeQuest Timing Analyzer returns the following Fmax operating frequencies for the circuit.

	Fmax	Restricted Fmax	Clock Name	Note
1	33333.33 MHz	800.0 MHz	maxring_refclk	limit due to minimum period restriction (tmin)
2	33333.33 MHz	800.0 MHz	qsfp_refclk	limit due to minimum period restriction (tmin)
3	293.0 MHz	293.0 MHz	pcie_coreclkout	
4	210.53 MHz	210.53 MHz	CLK_10G	
5	204.37 MHz	204.37 MHz	sv_reconfig_pma_testbus_clk_1	
6	162.23 MHz	162.23 MHz	STREAM_clkout0	
7	154.75 MHz	154.75 MHz	PHY_QSFP_TOP|PHY|PHY|phy_10gbase_r...mm_interface_inst|pmatestbussel[0]	
8	140.96 MHz	140.96 MHz	PHY_QSFP_TOP|PHY|PHY|phy_10gbase_r...mm_interface_inst|pmatestbussel[0]	
9	133.08 MHz	133.08 MHz	refclk_pci_express	
10	113.68 MHz	113.68 MHz	CLK_MGMT_100_RIGHT	
11	52.91 MHz	52.91 MHz	cclk	

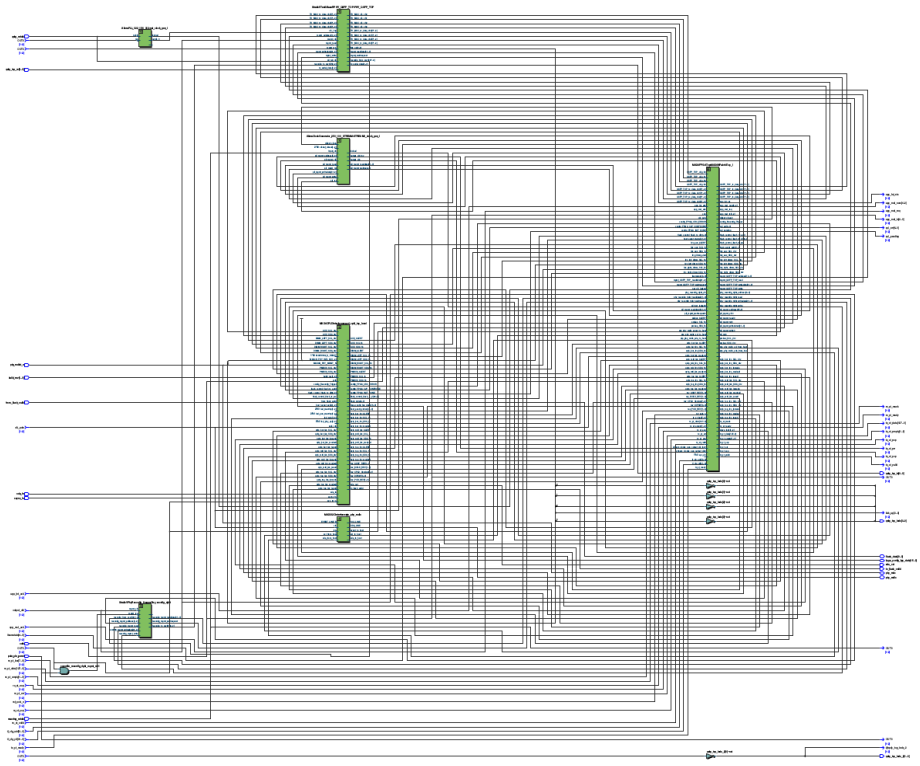
This panel reports FMAX for every clock in the design, regardless of the manually-specified clock periods. FMAX is only computed for paths where the source and destination registers or ports are driven by the same clock. Paths of different clocks, including generated clocks, are ignored.

6. Netlist Writer

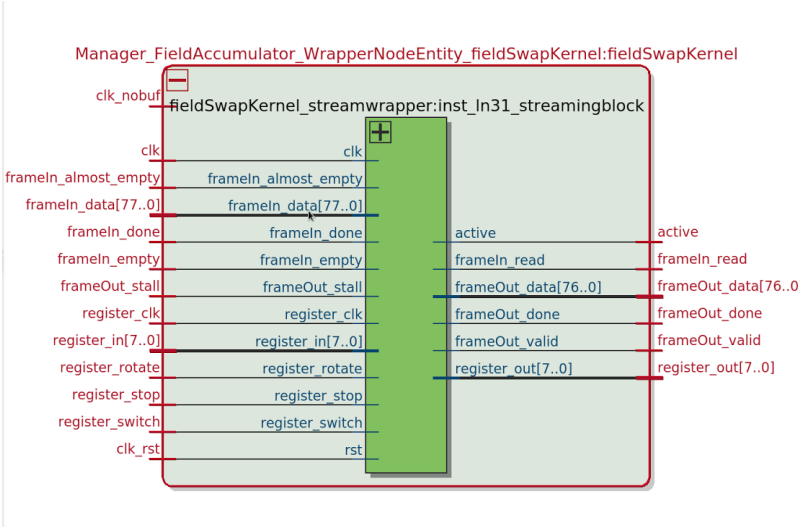
During synthesis, Quartus generates a netlist that uses the primitives of the Altera library and routes it. The Quartus RTL Viewer displays a schematic view of the design netlist after Analysis and Elaboration is performed by the Quartus II software, but before technology mapping and any synthesis or fitter

optimizations. This view is not the final design structure because optimizations have not yet occurred. This view most closely represents your original source design.

Netlist Viewer:



Delving deeper into the RTL netlist, the field swap kernel manager is implemented as



4.2 Resource Utilization

Flow Summary	
Flow Status	Successful - Wed May 13 19:13:08 2015
Quartus II 64-Bit Version	13.1.0 Build 162 10/23/2013 SJ Full Version
Revision Name	MAX4NPeripheryTop
Top-level Entity Name	MAX4NPeripheryTop
Family	Stratix V
Device	5SGXMABN2F45C2
Timing Models	Final
Logic utilization (in ALMs)	20,245 / 359,200 (6 %)
Total registers	37492
Total pins	129 / 1,064 (12 %)
Total virtual pins	69
Total block memory bits	1,948,042 / 54,067,200 (4 %)
Total DSP Blocks	0 / 352 (0 %)
Total HSSI STD RX PCSs	9 / 48 (19 %)
Total HSSI 10G RX PCSs	4 / 48 (8 %)
Total HSSI GEN3 RX PCSs	0 / 48 (0 %)
Total HSSI PMA RX Deserializers	12 / 48 (25 %)
Total HSSI STD TX PCSs	9 / 48 (19 %)
Total HSSI 10G TX PCSs	4 / 48 (8 %)
Total HSSI GEN3 TX PCSs	0 / 48 (0 %)
Total HSSI TX Channels	13 / 48 (27 %)
Total HSSI PIPE GEN1_2s	9 / 48 (19 %)
Total HSSI GEN3s	9 / 48 (19 %)
Total PLLs	4 / 98 (4 %)
Total DLLs	0 / 4 (0 %)

5. Hardware

5.1 Max4N Platform

In this project we target the MAX4N FPGA accelerator platforms that possess the Stratix V (5SGXMABN2F45C2)

Fig. 8: Max4N FPGA platform

5.1.1 QSFP Ports

The Max4N platform has two *Quad Small Form-factor Pluggable* (QSFP) ports. QSFP is a hot-pluggable transceiver that is able to to instantiate up to four 10 Gbps ethernet modules, for a data rate of 40 Gbps. In our ticker plant implementation, we are only using one of the two QSFP ports.

5.1.2 10-Gbps Ethernet

The 10-Gbps Ethernet IP core includes an Ethernet Media Access Control (MAC) with an Avalon Streaming (Avalon-ST) interface on the client side, and a XAUI or a standard XGMII interface on the network side. The XAUI interface is implemented as a hard IP in an Altera FPGA transceiver or as soft logic, which results in a soft 10GBASE-X XAUI PCS. Data arrives in 8 bytes chunks at a rate of 156.25 Mhz.

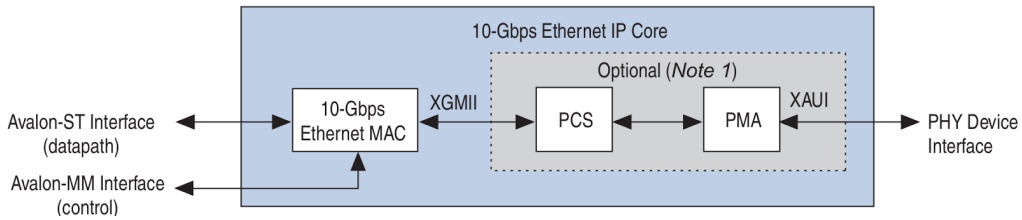


Fig. 9: Ethernet Core

5.3 Altera Stratix V FPGA

Resource intensive applications such as the Ticker plant design and other High Frequency Trading applications require state of the art FPGAs that posses a higher count of logic elements and on chip memory. State of the art FPGAs such as Altera's Stratix V FPGAs and Xilinx's Virtex 7 FPGAs suit these applications. The Stratix V FPGAs are optimized for bandwidth-centric applications and protocols, including the PCI Express bus. They are also efficient in handling data-intensive applications for 40G/100G using the 10-Gbps ethernet modules.

The following table shows a comparison of resources in SoCKit Cyclone V vs Stratix V Max4N FPGA platform.

FPGA	Altera Cyclone V 5CSXFC6D6F31C8ES	Altera Stratix V 5SGXMABN2F45C2
Platform	SoCKit Board	Max4N Platform
ALMS	41,910	359,200
Block Memory Bits	5,662,720	54,067,200
DSP Blocks	112	352

5.4 Chip Utilization

Resource Utilisation on the Chip. The Maxeler-Quartus flow provides the resource utilization results on the FPGA

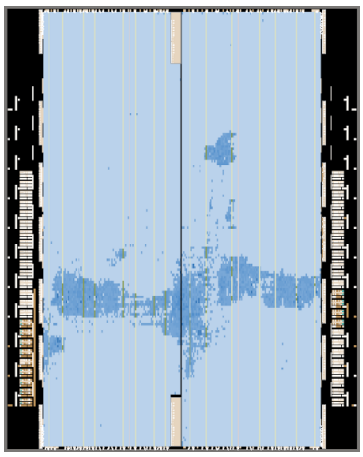


Fig. 10: Resource Utilisation

6. Conclusions

Maxeler uses a powerful simulation system that simulates the entire system in much shorter time than running the Quartus compilation flow would. This also gives us an assurance that our system is good to be implemented on the FPGA

A sample selection of our output from Maxeler compilation and runtime is shown below

```
Thu 00:02: ===== Bytes Received: 120 =====
Thu 00:02: [v] Instr A   BID:  Q = 19, Price = 127
Thu 00:02: [v] Instr Ai  BID:  Q = 26, Price = 1298
Thu 00:02: [v] Instr B   ASK:  Q = 51, Price = 4612
Thu 00:02: [v] Instr Bi  ASK:  Q = 19, Price = 3441
Thu 00:02: [v] Instr AB  BID:  Q = 26, Price = -3314
Thu 00:02: [v] Instr ABi BID:  Q = 19, Price = -4485
Thu 00:02: ===== Bytes Received: 120 =====
Thu 00:02: [v] Instr A   BID:  Q = 19, Price = 127
Thu 00:02: [v] Instr Ai  BID:  Q = 26, Price = 1298
Thu 00:02: [v] Instr B   ASK:  Q = 51, Price = 4612
Thu 00:02: [v] Instr Bi  ASK:  Q = 19, Price = 3441
Thu 00:02: [v] Instr AB  BID:  Q = 26, Price = -3314
Thu 00:02: [v] Instr ABi BID:  Q = 19, Price = -4485
Thu 00:02: ===== Bytes Received: 120 =====
Thu 00:02: [v] Instr A   BID:  Q = 19, Price = 127
Thu 00:02: [v] Instr Ai  BID:  Q = 26, Price = -3017
Thu 00:02: [v] Instr B   ASK:  Q = 81, Price = 297
Thu 00:02: [v] Instr Bi  ASK:  Q = 19, Price = 3441
Thu 00:02: [v] Instr AB  BID:  Q = 26, Price = -3314
```

```
Thu 00:02: [v] Instr ABi BID:  Q = 19, Price = -170
Thu 00:02: ===== Bytes Received: 120 =====
Thu 00:02: [v] Instr A   BID:  Q = 19, Price = 127
Thu 00:02: [v] Instr Ai  BID:  Q = 26, Price = -3017
Thu 00:02: [v] Instr B   ASK:  Q = 81, Price = 297
Thu 00:02: [v] Instr Bi  ASK:  Q = 19, Price = 3441
Thu 00:02: [v] Instr AB  BID:  Q = 26, Price = -3314
Thu 00:02: [v] Instr ABi BID:  Q = 19, Price = -170
```

7. References

[1]https://eventbooking.stfc.ac.uk/uploads/mew25/day11700maxeler.pdf
[2]https://www.maxeler.com/media/documents/MaxelerWhitePaperMaxCompiler.pdf
[3]https://www.altera.com/en_US/pdfs/literature/ug/xcvr_user_guide.pdf
[4]https://eventbooking.stfc.ac.uk/uploads/mew25/day11700maxeler.pdf
[5]http://www2.hmc.edu/~evans/e104l12.pdf
[6]http://www.boerse-frankfurt.de/en/glossary/o/order+book+930
[7]Narang, Rishi K. Inside the Black Box: A Simple Guide to Quantitative and High Frequency Trading. Second ed. Print.
[8]https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/ug/archives/10g_ethernet_user_guide.pdf
[9]http://www.doc.ic.ac.uk/~georgig/OpenSPL2014/lectures/OpenSPL_4.pdf

Appendix A: Terminology

Future

A futures contract allows a trader to undertake a contract to accept or make delivery of a commodity or some kind of financial asset (a) in the future on a known date, (b) under specified conditions, (c) for a price contracted today. The party to the contract who is agreeing to take delivery of the commodity is long in the position, whereas the party who is agreeing to deliver the commodity is short in the position. A speculator will benefit when she is long if the prices rise, short if the price falls. Through submission of bids and asks, the exchange will match long orders with short orders, either with outside traders or with their own trades

Futures can be used either to hedge or to speculate on the price movement of the underlying asset. For example, a producer of corn could use futures to lock in a certain price and reduce risk (hedge). On the other hand, anybody could speculate on the price movement of corn by going long or short using futures. Airline companies may place hedges either based on future prices of jet fuel or on future prices of jet fuel or on future prices of crude oil (which is the source of jet fuel).

Spread Trading

Spread trading is the simultaneous purchase of one security (i.e. stocks, futures contracts) combined with the sale of another related security. The individual securities are called legs. Common spreads are priced and traded as a unit on futures exchanges rather than as individual legs, thus ensuring simultaneous execution and eliminating the execution risk of one leg executing but the other failing. Spread trades are executed to attempt to profit from the widening or narrowing of the spread, rather than from movement in the prices of the legs directly. There are several types of spreads, including:

Calendar Spreads

A Calendar Spread is a spread trade involving the simultaneous purchase of futures expiring on a particular date and sale of the same instrument expiring on another date. The legs of the spread vary only in expiration date.

Intercommodity Spreads

Intercommodity spreads are formed from two distinct but related commodities, reflecting the economic relationship between them. An example would be a spread of apples and applesauce.

Ticker Plant

A ticker plant is a specialized software or hardware systems designed to handle collection and throughput of an incoming data stream and updating the system's order book's states. which can then be seen by traders or algorithms to understand the current state of

Order Book

An order book is used to pool, compare and match the volumes and prices of buy and sell orders for a particular security. When several orders contain the same price, they are referred as a price level, meaning that if, say, a bid comes at that price level, all the orders on that price level could potentially fulfill that.

There are two general approaches to order execution: aggressive and passive. Aggressive orders are are submitted to the marketplace and are generally unconditional. They can be filled in pieces or full at whatever price prevails in the market at the time the order's turn to be executed arrives (within reasonable boundaries, as long as there is a bid or offer resting in the order book to take the other side of the market order). In contrast, passive orders allow the trader to control the worst price at which he is willing to transact, but the trader must accept that his order might not get executed at all or that only a part of it might be executed.

Instrument ID

Our program gets a stream of data that includes prices and quantities for two futures, and their spread. Each of the futures and their spread is indexed using an Instrument ID.

Side

Side indicates whether a future is a bid or an ask. 0 signifies a bid or an offer to buy, and 1 signifies and ask or an offer to sell.

Price Level

The Price level of a bid or ask indicates how many other offers of the same side persist in the order book that are a better offer than that particular bid or ask. Our engine operates at level 0 indicating that it only considers the best current bid and ask of the stock.

Ethernet

The Ethernet access method is used to connect hosts in a company, home, or network; as well as to connect a single host to a modem for Internet access.

Internet Protocol (IP)

The Internet Protocol is a set of rules for exchange of information between hosts on the Internet. Each host that uses the Internet Protocol has at least one IP address that identifies it to all other devices on the planet, just like a person might have a postal address.

User Datagram Protocol (UDP)

With UDP, computer applications can send messages, sometimes known as datagrams, to other hosts on an Internet Protocol (IP) network without requiring other communications to set up special transmission channels or data paths. UDP is known as unreliable as it does not guarantee that all data with reach the intended destination in order.

Appendix B: Code

FieldAccumulatorCpuCode.c

```
/* Ticker Plant System Implemented in Max Compiler
 * Columbia University: CSEE 4840, Spring 2015
 * May 14th 2015
 *
 *   - Gabriel Blanco
 *   - Suchith Vasudevan
 *   - Brian Bourn
 *   - David Naveen Dhas Arthur
 */
#define _GNU_SOURCE

#include <string.h>
#include <unistd.h>
#include <stdio.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/tcp.h>

#include "Maxfiles.h"
#include <MaxSLiCInterface.h>

#define BUFFERSIZE 1024
#define FIELDS 5

struct input_data
{
    int32_t instrument_id;
    int32_t level;
    int32_t side;             // 0 is Bidding, 1 is Asking
    int32_t quantity;
    int32_t price;
};

typedef struct output_data
{
    struct input_data a_bid;
    struct input_data ai_bid;
    struct input_data b_ask;
    struct input_data bi_ask;
    struct input_data ab_bid;
    struct input_data abi_bid;
} __attribute__ ((__packed__)) frame_t;

static void calculateDeltas(int, struct input_data *);
static void validateData(struct input_data *, struct output_data *);
static int  create_cpu_udp_socket(struct in_addr *, struct in_addr *, int);
static void parse(char *, struct input_data *);
static int isEqual(struct input_data *, struct input_data *);

int
main(int argc, char *argv[])
{
    if(argc != 4)
    {
        printf("Usage: %s <dfe_ip> <cpu_ip> <netmask>\n", argv[0]);
```

```
        return 1;
    }

    struct in_addr dfe_ip, cpu_ip, netmask;
    const int port = 5008;
    char line[BUFFERSIZE];

    inet_aton(argv[1], &dfe_ip);
    inet_aton(argv[2], &cpu_ip);
    inet_aton(argv[3], &netmask);

    // Create DFE Socket, then listen
    max_file_t *maxfile = FieldAccumulator_init();
    max_engine_t *engine = max_load(maxfile, "*");
    max_ip_config(engine, MAX_NET_CONNECTION_QSFP_TOP_10G_PORT1, &dfe_ip, &netmask);
    max_udp_socket_t *dfe_socket = max_udp_create_socket(engine, "udp_ch2_sfp1");
    max_udp_bind(dfe_socket, port);
    max_udp_connect(dfe_socket, &cpu_ip, port);

    int cpu_socket = create_cpu_udp_socket(&cpu_ip, &dfe_ip, port);


    // CSV Input Data Parsing
    FILE *stream = fopen("./source_data2.csv", "r");

    if(stream == NULL)
    {
        printf("fopen() failed ");
        return -1 ;
    }

    // Ignore Header File
    fgets(line, BUFFERSIZE, stream);

    int linum = 0;
    while (fgets(line, sizeof(line), stream))
    {
        struct input_data data;
        parse(line, &data);
        calculateDeltas(cpu_socket, &data);
        linum++;
    }

    printf("number of lines: %d\n",linum);

    max_udp_close(dfe_socket);
    max_unload(engine);
    max_file_free(maxfile);

    return 0;
}

/*
 * Parses a CSV file into an input structs
 */
static void
parse(char *line, struct input_data *in)
{
    int i;
    int fv[FIELDS];

    char *element = strtok(line, ",");
    fv[0] = atoi(element);

    for (i=1; i<FIELDS; i++)
```

```
    {
        char *element = strtok(NULL, ",");
        fv[i] = atoi(element);
    }

    in->instrument_id = fv[0];
    in->level         = fv[1];
    in->side          = fv[2];
    in->quantity      = fv[3];
    in->price         = fv[4];
}

/*
 * Sending and Receiving input data via UDP, then validating and printing results
 */
static void
calculateDeltas(int sock, struct input_data *data)
{
    frame_t instruments, instruments_exp;
    int32_t bytesRecv, num_instr;

    num_instr = (int32_t)sizeof(struct output_data) / (int32_t)sizeof(struct input_data);

    // Send Data to Engine via TCP
    send(sock, data, sizeof(struct input_data), 0);

    // Receive Data from Engine via TCP
    validateData(data, &instruments_exp);
    bytesRecv = recv(sock, &instruments, sizeof(struct output_data), 0);
    if (bytesRecv == -1)
    {
        printf("No bytes recv\n");
        exit(0);
    }
    else if (bytesRecv < (int32_t)sizeof(struct output_data))
    {
        printf("WARNING: Received less bytes than expected\n");
    }

    printf("===== Bytes Received: %d =====\n", bytesRecv);

    char valid [num_instr];

    valid[0] = isEqual(&instruments.a_bid, &instruments_exp.a_bid) ?     'v' : 'x';
    valid[1] = isEqual(&instruments.ai_bid, &instruments_exp.ai_bid) ?   'v' : 'x';
    valid[2] = isEqual(&instruments.b_ask, &instruments_exp.b_ask) ?     'v' : 'x';
    valid[3] = isEqual(&instruments.bi_ask, &instruments_exp.bi_ask) ?   'v' : 'x';
    valid[4] = isEqual(&instruments.ab_bid, &instruments_exp.ab_bid) ?   'v' : 'x';
    valid[5] = isEqual(&instruments.abi_bid, &instruments_exp.abi_bid) ? 'v' : 'x';

    printf("[%c] Instr A   BID:  Q = %d, Price = %d\n", valid[0], instruments.a_bid.quantity,
instruments.a_bid.price);
    printf("[%c] Instr Ai  BID:  Q = %d, Price = %d\n", valid[1], instruments.ai_bid.quantity,
instruments.ai_bid.price);
    printf("[%c] Instr B   ASK:  Q = %d, Price = %d\n", valid[2], instruments.b_ask.quantity,
instruments.b_ask.price);
    printf("[%c] Instr Bi  ASK:  Q = %d, Price = %d\n", valid[3], instruments.bi_ask.quantity,
instruments.bi_ask.price);
    printf("[%c] Instr AB  BID:  Q = %d, Price = %d\n", valid[4], instruments.ab_bid.quantity,
instruments.ab_bid.price);
    printf("[%c] Instr ABi BID:  Q = %d, Price = %d\n", valid[5], instruments.abi_bid.quantity,
instruments.abi_bid.price);
}
```

```
/*
 * Runs the same calculations as in the DFE Engine to Validate the received results
 */
static void
validateData(struct input_data *in, struct output_data *out)
{
    // -------- Expected Value, Calculated in Software --------

    // Instrument A
    static int32_t a_bidprice = 0;
    static int32_t a_bidquant = 0;
    static int32_t a_askprice = 0;
    static int32_t a_askquant = 0;

    // Instrument B
    static int32_t b_bidprice = 0;
    static int32_t b_bidquant = 0;
    static int32_t b_askprice = 0;
    static int32_t b_askquant = 0;

    // Instrument A-B Spread
    static int32_t ab_bidprice = 0;
    static int32_t ab_bidquant = 0;
    static int32_t ab_askprice = 0;
    static int32_t ab_askquant = 0;

    // Update Correct Register
    if(in->instrument_id==0 && in->side==0 && in->level==0) { a_bidprice = in->price;  a_bidquant =
in->quantity; }
    if(in->instrument_id==0 && in->side==1 && in->level==0) { a_askprice = in->price;  a_askquant =
in->quantity; }
    if(in->instrument_id==1 && in->side==0 && in->level==0) { b_bidprice = in->price;  b_bidquant =
in->quantity; }
    if(in->instrument_id==1 && in->side==1 && in->level==0) { b_askprice = in->price;  b_askquant =
in->quantity; }
    if(in->instrument_id==2 && in->side==0 && in->level==0) { ab_bidprice = in->price; ab_bidquant =
in->quantity; }
    if(in->instrument_id==2 && in->side==1 && in->level==0) { ab_askprice = in->price; ab_askquant =
in->quantity; }

    // Implied Instrument
    int32_t ai_bidquant = ab_bidquant < b_askquant ? ab_bidquant : b_askquant;
    int32_t ai_bidprice = ab_bidprice + b_askprice;
    int32_t bi_askquant = a_bidquant < ab_bidquant ? a_bidquant : ab_bidquant;
    int32_t bi_askprice = a_bidprice - ab_bidprice;
    int32_t abi_bidquant = a_bidquant < b_askquant ? a_bidquant : b_askquant;
    int32_t abi_bidprice = a_bidprice - b_askprice;

    // Output Parameters
    out->a_bid.instrument_id = 0;
    out->a_bid.level = 0;
    out->a_bid.side = 0;
    out->a_bid.quantity = a_bidquant;
    out->a_bid.price = a_bidprice;

    out->ai_bid.instrument_id = 0;
    out->ai_bid.level = 0;
    out->ai_bid.side = 0;
    out->ai_bid.quantity = ai_bidquant;
    out->ai_bid.price = ai_bidprice;

    out->b_ask.instrument_id = 1;
    out->b_ask.level = 0;
    out->b_ask.side = 1;
    out->b_ask.quantity = b_askquant;
```

```
    out->b_ask.price = b_askprice;

    out->bi_ask.instrument_id = 1;
    out->bi_ask.level = 0;
    out->bi_ask.side = 1;
    out->bi_ask.quantity = bi_askquant;
    out->bi_ask.price = bi_askprice;

    out->ab_bid.instrument_id = 2;
    out->ab_bid.level = 0;
    out->ab_bid.side = 0;
    out->ab_bid.quantity = ab_bidquant;
    out->ab_bid.price = ab_bidprice;

    out->abi_bid.instrument_id = 2;
    out->abi_bid.level = 0;
    out->abi_bid.side = 0;
    out->abi_bid.quantity = abi_bidquant;
    out->abi_bid.price = abi_bidprice;
}


/*
 * Create a UDP Socket on the CPU
 */
static int
create_cpu_udp_socket(struct in_addr *local_ip, struct in_addr *remote_ip, int port)
{
    int sock = socket(AF_INET, SOCK_DGRAM, 0);

    struct sockaddr_in cpu;
    memset(&cpu, 0, sizeof(cpu));
    cpu.sin_family = AF_INET;
    cpu.sin_port = htons(port);

    cpu.sin_addr = *local_ip;
    bind(sock, (struct sockaddr *)&cpu, sizeof(cpu));

    cpu.sin_addr = *remote_ip;
    connect(sock, (const struct sockaddr*) &cpu, sizeof(cpu));

    return sock;
}

/*
 * Comparison between input data structs, used for validation
 */
static int
isEqual(struct input_data *a, struct input_data *b)
{
    if (a->instrument_id != b->instrument_id)
    {
        printf("ID MISMATCH\n");
        return 0;
    }
    else if (a->level != b->level)
    {
        printf("LEVEL MISMATCH\n");
        return 0;
    }
    else if (a->side != b->side)
    {
        printf("SIDE MISMATCH\n");
        return 0;
    }
```

```
    else if (a->quantity != b->quantity)
    {
        printf("QUANTITY MISMATCH\n");
        return 0;
    }
    else if (a->price != b->price)
    {
        printf("PRICE MISMATCH\n");
        return 0;
    }
    return 1;
}
```

FieldAccumulatorKernel.maxj

```
/* Ticker Plant System Implemented in Max Compiler
 * Columbia University: CSEE 4840, Spring 2015
 * May 14th 2015
 *
 *   - Gabriel Blanco
 *   - Suchith Vasudevan
 *   - Brian Bourn
 *   - David Naveen Dhas Arthur
 */
package fieldaccumulator;

import com.maxeler.maxcompiler.v2.kernelcompiler.KernelParameters;
import com.maxeler.maxcompiler.v2.kernelcompiler.stdlib.Reductions;
import com.maxeler.maxcompiler.v2.kernelcompiler.types.base.DFEVar;
import com.maxeler.networking.v1.framed_kernels.ByteOrder;
import com.maxeler.networking.v1.framed_kernels.FrameData;
import com.maxeler.networking.v1.framed_kernels.FrameFormat;
import com.maxeler.networking.v1.framed_kernels.FramedKernel;
import com.maxeler.networking.v1.kernel_types.UDPOneToOneRXType;
import com.maxeler.networking.v1.kernel_types.UDPOneToOneTXType;

public class
FieldAccumulatorKernel extends FramedKernel
{

    // Input Frame Format
    static class
    DataIn extends FrameFormat
    {
        DataIn()
        {
            super(ByteOrder.LITTLE_ENDIAN);
            addField("instrument_id", dfeInt(32));
            addField("level", dfeInt(32));
            addField("side", dfeInt(32));
            addField("quantity", dfeInt(32));
            addField("price", dfeInt(32));
        }
    }

    // Output Frame Format
    static class
    DataOut extends FrameFormat
    {
        DataOut()
        {
```

```
            super(ByteOrder.LITTLE_ENDIAN);
            addField("a_bid_instrument_id", dfeInt(32));
            addField("a_bid_level", dfeInt(32));
            addField("a_bid_side", dfeInt(32));
            addField("a_bid_quantity", dfeInt(32));
            addField("a_bid_price", dfeInt(32));

            addField("ai_bid_instrument_id", dfeInt(32));
            addField("ai_bid_level", dfeInt(32));
            addField("ai_bid_side", dfeInt(32));
            addField("ai_bid_quantity", dfeInt(32));
            addField("ai_bid_price", dfeInt(32));

            addField("b_ask_instrument_id", dfeInt(32));
            addField("b_ask_level", dfeInt(32));
            addField("b_ask_side", dfeInt(32));
            addField("b_ask_quantity", dfeInt(32));
            addField("b_ask_price", dfeInt(32));

            addField("bi_ask_instrument_id", dfeInt(32));
            addField("bi_ask_level", dfeInt(32));
            addField("bi_ask_side", dfeInt(32));
            addField("bi_ask_quantity", dfeInt(32));
            addField("bi_ask_price", dfeInt(32));

            addField("ab_bid_instrument_id", dfeInt(32));
            addField("ab_bid_level", dfeInt(32));
            addField("ab_bid_side", dfeInt(32));
            addField("ab_bid_quantity", dfeInt(32));
            addField("ab_bid_price", dfeInt(32));

            addField("abi_bid_instrument_id", dfeInt(32));
            addField("abi_bid_level", dfeInt(32));
            addField("abi_bid_side", dfeInt(32));
            addField("abi_bid_quantity", dfeInt(32));
            addField("abi_bid_price", dfeInt(32));
        }
    }

    // Kernel Computation
    public FieldAccumulatorKernel(KernelParameters parameters)
    {
        super(parameters);

        // Bid Registers
        DFEVar a_bidprice  = dfeInt(32).newInstance(this);
        DFEVar a_bidquant  = dfeInt(32).newInstance(this);
        DFEVar b_bidprice  = dfeInt(32).newInstance(this);
        DFEVar b_bidquant  = dfeInt(32).newInstance(this);
        DFEVar ab_bidprice = dfeInt(32).newInstance(this);
        DFEVar ab_bidquant = dfeInt(32).newInstance(this);

        //Ask Registers
        DFEVar a_askprice  = dfeInt(32).newInstance(this);
        DFEVar a_askquant  = dfeInt(32).newInstance(this);
        DFEVar b_askprice  = dfeInt(32).newInstance(this);
        DFEVar b_askquant  = dfeInt(32).newInstance(this);
        DFEVar ab_askprice = dfeInt(32).newInstance(this);
        DFEVar ab_askquant = dfeInt(32).newInstance(this);

        /* Declare Frame In */
        FrameData<DataIn> frameIn = io.frameInput("frameIn", new DataIn(), new UDPOneToOneRXType());

        pushResetBetweenFrames(false);
```

```
        // Conditional
        DFEVar a_bid  = frameIn["instrument_id"].eq(constant.var(dfeInt(32), 0)) &
                        frameIn["level"].eq(constant.var(dfeInt(32), 0)) &
                        frameIn["side"].eq(constant.var(dfeInt(32), 0));

        DFEVar b_bid  = frameIn["instrument_id"].eq(constant.var(dfeInt(32), 1)) &
                        frameIn["level"].eq(constant.var(dfeInt(32), 0)) &
                        frameIn["side"].eq(constant.var(dfeInt(32), 0));

        DFEVar ab_bid = frameIn["instrument_id"].eq(constant.var(dfeInt(32), 2)) &
                        frameIn["level"].eq(constant.var(dfeInt(32), 0)) &
                        frameIn["side"].eq(constant.var(dfeInt(32), 0));

        DFEVar a_ask  = frameIn["instrument_id"].eq(constant.var(dfeInt(32), 0)) &
                        frameIn["level"].eq(constant.var(dfeInt(32), 0)) &
                        frameIn["side"].eq(constant.var(dfeInt(32), 1));

        DFEVar b_ask  = frameIn["instrument_id"].eq(constant.var(dfeInt(32), 1)) &
                        frameIn["level"].eq(constant.var(dfeInt(32), 0)) &
                        frameIn["side"].eq(constant.var(dfeInt(32), 1));

        DFEVar ab_ask = frameIn["instrument_id"].eq(constant.var(dfeInt(32), 2)) &
                        frameIn["level"].eq(constant.var(dfeInt(32), 0)) &
                        frameIn["side"].eq(constant.var(dfeInt(32), 1));

        // Update Register Value
        a_bidprice  = Reductions.streamHold(frameIn["price"], a_bid);
        a_bidquant  = Reductions.streamHold(frameIn["quantity"], a_bid);
        b_bidprice  = Reductions.streamHold(frameIn["price"], b_bid);
        b_bidquant  = Reductions.streamHold(frameIn["quantity"], b_bid);
        ab_bidprice = Reductions.streamHold(frameIn["price"], ab_bid);
        ab_bidquant = Reductions.streamHold(frameIn["quantity"], ab_bid);
        a_askprice  = Reductions.streamHold(frameIn["price"], a_ask);
        a_askquant  = Reductions.streamHold(frameIn["quantity"], a_ask);
        b_askprice  = Reductions.streamHold(frameIn["price"], b_ask);
        b_askquant  = Reductions.streamHold(frameIn["quantity"], b_ask);
        ab_askprice = Reductions.streamHold(frameIn["price"], ab_ask);
        ab_askquant = Reductions.streamHold(frameIn["quantity"], ab_ask);

        popResetBetweenFrames();

        // Calculate Implied Prices and Quantities
        DFEVar ai_bidquant = ab_bidquant < b_askquant ? ab_bidquant : b_askquant;
        DFEVar ai_bidprice = ab_bidprice + b_askprice;
        DFEVar bi_askquant = a_bidquant < ab_bidquant ? a_bidquant : ab_bidquant;
        DFEVar bi_askprice = a_bidprice - ab_bidprice;
        DFEVar abi_bidquant = a_bidquant < b_askquant ? a_bidquant : b_askquant;
        DFEVar abi_bidprice = a_bidprice - b_askprice;

        // Declare Frame Out
        FrameData<DataOut> frameOut = new FrameData<DataOut>(this, new DataOut(), new
UDPOneToOneTXType());

        frameOut["a_bid_instrument_id"] <== constant.var(dfeInt(32), 0);
        frameOut["a_bid_level"] <== constant.var(dfeInt(32), 0);
        frameOut["a_bid_side"] <== constant.var(dfeInt(32), 0); // Bidding
        frameOut["a_bid_quantity"] <== a_bidquant;
        frameOut["a_bid_price"] <== a_bidprice;

        frameOut["ai_bid_instrument_id"] <== constant.var(dfeInt(32), 0);
        frameOut["ai_bid_level"] <== constant.var(dfeInt(32), 0);
        frameOut["ai_bid_side"] <== constant.var(dfeInt(32), 0); // Bidding
        frameOut["ai_bid_quantity"] <== ai_bidquant;
        frameOut["ai_bid_price"] <== ai_bidprice;
```

```
        frameOut["b_ask_instrument_id"] <== constant.var(dfeInt(32), 1);
        frameOut["b_ask_level"] <== constant.var(dfeInt(32), 0);
        frameOut["b_ask_side"] <== constant.var(dfeInt(32), 1); // Bidding
        frameOut["b_ask_quantity"] <== b_askquant;
        frameOut["b_ask_price"] <== b_askprice;

        frameOut["bi_ask_instrument_id"] <== constant.var(dfeInt(32), 1);
        frameOut["bi_ask_level"] <== constant.var(dfeInt(32), 0);
        frameOut["bi_ask_side"] <== constant.var(dfeInt(32), 1); // Bidding
        frameOut["bi_ask_quantity"] <== bi_askquant;
        frameOut["bi_ask_price"] <== bi_askprice;

        frameOut["ab_bid_instrument_id"] <== constant.var(dfeInt(32), 2);
        frameOut["ab_bid_level"] <== constant.var(dfeInt(32), 0);
        frameOut["ab_bid_side"] <== constant.var(dfeInt(32), 0); // Bidding
        frameOut["ab_bid_quantity"] <== ab_bidquant;
        frameOut["ab_bid_price"] <== ab_bidprice;

        frameOut["abi_bid_instrument_id"] <== constant.var(dfeInt(32), 2);
        frameOut["abi_bid_level"] <== constant.var(dfeInt(32), 0);
        frameOut["abi_bid_side"] <== constant.var(dfeInt(32), 0); // Bidding
        frameOut["abi_bid_quantity"] <== abi_bidquant;
        frameOut["abi_bid_price"] <== abi_bidprice;

        frameOut.linkfield[UDPOneToOneTXType.SOCKET] <==
frameIn.linkfield[UDPOneToOneRXType.SOCKET];

        io.frameOutput("frameOut", frameOut);
    }
}
```

FieldAccumulatorManager.maxj

```
/* Ticker Plant System Implemented in Max Compiler
 * Columbia University: CSEE 4840, Spring 2015
 * May 14th 2015
 *
 *   - Gabriel Blanco
 *   - Suchith Vasudevan
 *   - Brian Bourn
 *   - David Naveen Dhas Arthur
 */
package fieldaccumulator;

import com.maxeler.maxcompiler.v2.build.EngineParameters;
import com.maxeler.maxcompiler.v2.managers.BuildConfig;
import com.maxeler.maxcompiler.v2.managers.custom.blocks.KernelBlock;
import com.maxeler.networking.v1.managers.NetworkManager;
import com.maxeler.networking.v1.managers.netlib.Max4NetworkConnection;
import com.maxeler.networking.v1.managers.netlib.UDPChecksumMode;
import com.maxeler.networking.v1.managers.netlib.UDPConnectionMode;
import com.maxeler.networking.v1.managers.netlib.UDPStream;

public class
FieldAccumulatorManager extends NetworkManager
{

    public
    FieldAccumulatorManager(EngineParameters configuration)
    {
        super(configuration);
```

```
        UDPStream frameIn = addUDPStream("udp_ch2_sfp1", Max4NetworkConnection.QSFP_TOP_10G_PORT1,
UDPConnectionMode.OneToOne, UDPChecksumMode.DropBadFrames);
        UDPStream frameOut = addUDPStream("frameOut", Max4NetworkConnection.QSFP_TOP_10G_PORT1,
UDPConnectionMode.OneToOne, UDPChecksumMode.DropBadFrames);
        KernelBlock kernel = addKernel(new
FieldAccumulatorKernel(makeKernelParameters("fieldSwapKernel")));
        kernel.getInput("frameIn") <== frameIn.getReceiveStream();
        frameOut.getTransmitStream() <== kernel.getOutput("frameOut");
    }

    public static void
    main(String[] args)
    {
    FieldAccumulatorEngineParameters params = new FieldAccumulatorEngineParameters(args);
        FieldAccumulatorManager m = new FieldAccumulatorManager(params);

        BuildConfig buildConfig = m.getBuildConfig();
        buildConfig.setMPPRCostTableSearchRange(params.getMPPRStartCT(), params.getMPPREndCT());
        buildConfig.setMPPRParallelism(params.getMPPRThreads());
        buildConfig.setMPPRRetryNearMissesThreshold(params.getMPPRRetryThreshold());

        m.build();
    }
}
```

generate_input.py

```
# Ticker Plant System Implemented in Max Compiler
# Columbia University: CSEE 4840, Spring 2015
# May 14th 2015
#
#   - Gabriel Blanco
#   - Suchith Vasudevan
#   - Brian Bourn
#   - David Naveen Dhas Arthur

import random

NUMLINES = 500

ids = [0, 1, 2]
levels = [0]
sides = [0, 1]
spreads = [2]

print("instrument_id,level,side,quantity,price")

for i in range(NUMLINES):

    instrument_id = random.choice(ids)
    side = random.choice(sides)
    level = random.choice(levels)

    quantity = random.randint(1, 100)
    price = random.randint(1, 5000)

    if instrument_id in spreads:
        price = random.randint(-5000, 5000)

    print("{},{},{},{},{}".format(instrument_id, level, side, quantity, price))
```

source_data.csv

```
instrument_id,level,side,quantity,price
0,0,1,3,1293
1,0,0,46,1140
2,0,1,12,-231
0,0,1,21,1229
1,0,0,19,1496
2,0,1,49,228
0,0,1,25,1272
1,0,0,49,1470
2,0,1,15,-242
0,0,1,13,1114
1,0,0,40,1159
2,0,1,16,63
0,0,1,48,1229
1,0,0,34,1325
2,0,1,19,365
0,0,1,41,1153
1,0,0,28,1481
2,0,1,16,-162
0,0,1,34,1151
1,0,0,0,1246
2,0,1,18,165
0,0,1,2,1276
1,0,0,10,1371
2,0,1,48,-530
0,0,1,48,1477
1,0,0,32,1336
2,0,1,31,-142
0,0,1,34,1135
1,0,0,29,1315
2,0,1,1,-296
0,0,1,16,1025
1,0,0,37,1284
2,0,1,27,-571
0,0,1,29,1387
1,0,0,11,1270
2,0,1,40,-256
0,0,1,15,1328
1,0,0,39,1417
2,0,1,36,258
0,0,1,49,1385
1,0,0,44,1141
2,0,1,26,549
0,0,1,29,1255
1,0,0,24,1303
2,0,1,22,43
0,0,1,24,1431
1,0,0,42,1190
2,0,1,34,261
0,0,1,32,1213
1,0,0,32,1257
2,0,1,15,386
0,0,1,27,1492
1,0,0,17,1326
2,0,1,20,145
0,0,1,28,1321
1,0,0,32,1058
2,0,1,34,215
0,0,1,27,1365
```

```
1,0,0,33,1408
2,0,1,19,166
0,0,1,40,1299
1,0,0,24,1416
2,0,1,25,-219
0,0,1,3,1270
1,0,0,21,1003
2,0,1,30,640
0,0,1,34,1493
1,0,0,27,1473
2,0,1,48,411
0,0,1,14,1438
1,0,0,22,1462
2,0,1,10,-207
0,0,1,10,1166
1,0,0,7,1358
2,0,1,49,-289
0,0,1,27,1229
1,0,0,42,1488
2,0,1,47,-429
0,0,1,22,1358
1,0,0,35,1147
2,0,1,23,-154
0,0,1,43,1180
1,0,0,41,1490
2,0,1,17,-588
0,0,1,46,1175
1,0,0,29,1016
2,0,1,44,476
0,0,1,8,1175
1,0,0,19,1001
2,0,1,21,-264
0,0,1,28,1036
1,0,0,36,1226
2,0,1,7,-679
0,0,1,2,1495
1,0,0,24,1248
2,0,1,1,-94
0,0,1,38,1337
1,0,0,18,1482
2,0,1,17,-110
0,0,1,18,1316
1,0,0,35,1310
2,0,1,10,309
0,0,1,0,1197
1,0,0,8,1011
2,0,1,21,68
0,0,1,38,1184
1,0,0,5,1349
2,0,1,42,-120
0,0,1,25,1280
1,0,0,12,1243
2,0,1,36,-93
0,0,1,12,1300
1,0,0,3,1056
2,0,1,48,728
0,0,1,12,1275
1,0,0,1,1426
2,0,1,40,135
0,0,1,8,1162
1,0,0,22,1105
2,0,1,10,80
0,0,1,9,1101
1,0,0,11,1481
2,0,1,3,-857
```

```
0,0,1,25,1395
1,0,0,6,1266
2,0,1,29,75
0,0,1,31,1295
1,0,0,36,1265
2,0,1,6,-32
0,0,1,11,1380
1,0,0,6,1079
2,0,1,14,100
0,0,1,33,1172
1,0,0,2,1297
2,0,1,30,-19
0,0,1,41,1012
1,0,0,5,1261
2,0,1,36,-693
0,0,1,11,1194
1,0,0,10,1425
2,0,1,30,240
0,0,1,41,1332
1,0,0,0,1083
2,0,1,38,682
0,0,1,35,1116
1,0,0,36,1044
2,0,1,9,-420
0,0,1,21,1436
1,0,0,25,1477
2,0,1,25,-498
0,0,1,25,1473
1,0,0,5,1018
2,0,1,30,544
0,0,1,29,1129
1,0,0,7,1235
2,0,1,43,-523
0,0,1,3,1178
1,0,0,34,1013
2,0,1,33,-120
0,0,1,4,1142
1,0,0,28,1188
2,0,1,3,-524
0,0,1,37,1079
1,0,0,49,1350
2,0,1,25,-700
0,0,1,23,1493
1,0,0,33,1020
2,0,1,24,448
0,0,1,19,1207
1,0,0,31,1194
2,0,1,43,77
0,0,1,0,1337
1,0,0,40,1288
2,0,1,20,308
0,0,1,42,1497
1,0,0,49,1266
2,0,1,12,-262
0,0,1,41,1079
1,0,0,19,1189
2,0,1,47,222
0,0,1,9,1284
1,0,0,46,1331
2,0,1,31,-245
0,0,1,24,1028
1,0,0,28,1013
2,0,1,22,-235
0,0,1,26,1186
1,0,0,41,1442
```

```
2,0,1,4,-205
0,0,1,12,1379
1,0,0,18,1117
2,0,1,0,-7
0,0,1,36,1207
1,0,0,44,1450
2,0,1,44,-368
0,0,1,6,1179
1,0,0,47,1273
2,0,1,35,25
0,0,1,22,1134
1,0,0,18,1221
2,0,1,19,-313
0,0,1,29,1142
1,0,0,9,1491
2,0,1,45,-196
0,0,1,37,1398
1,0,0,27,1170
2,0,1,47,-176
0,0,1,11,1352
1,0,0,30,1413
2,0,1,23,333
0,0,1,12,1183
1,0,0,38,1192
2,0,1,22,-308
0,0,1,13,1380
1,0,0,46,1462
2,0,1,36,-458
0,0,1,38,1449
1,0,0,33,1481
2,0,1,16,29
0,0,1,37,1181
1,0,0,33,1448
2,0,1,25,-261
0,0,1,20,1132
1,0,0,21,1063
2,0,1,11,120
0,0,1,37,1449
1,0,0,37,1023
2,0,1,8,831
0,0,1,41,1037
1,0,0,35,1297
2,0,1,9,-24
0,0,1,31,1413
1,0,0,1,1090
2,0,1,11,548
0,0,1,26,1261
1,0,0,27,1050
2,0,1,8,97
0,0,1,3,1295
1,0,0,7,1397
2,0,1,9,3
0,0,1,33,1061
1,0,0,4,1166
2,0,1,25,147
0,0,1,0,1093
1,0,0,0,1243
2,0,1,8,120
0,0,1,18,1324
1,0,0,7,1208
2,0,1,24,464
0,0,1,44,1445
1,0,0,41,1300
2,0,1,12,428
0,0,1,40,1190
```

```
1,0,0,7,1335
2,0,1,17,-5
0,0,1,35,1144
1,0,0,14,1306
2,0,1,17,148
0,0,1,10,1068
1,0,0,35,1483
2,0,1,32,-538
0,0,1,1,1174
1,0,0,1,1367
2,0,1,9,241
0,0,1,7,1462
1,0,0,8,1464
2,0,1,22,-302
0,0,1,42,1131
1,0,0,0,1066
2,0,1,27,549
0,0,1,24,1115
1,0,0,1,1160
2,0,1,14,-502
0,0,1,16,1356
1,0,0,25,1250
2,0,1,6,-183
0,0,1,7,1045
1,0,0,22,1272
2,0,1,21,230
0,0,1,23,1224
1,0,0,16,1095
2,0,1,49,-113
0,0,1,41,1061
1,0,0,27,1044
2,0,1,46,74
0,0,1,36,1462
1,0,0,26,1005
2,0,1,26,596
0,0,1,40,1420
1,0,0,10,1327
2,0,1,13,123
0,0,1,24,1430
1,0,0,28,1218
2,0,1,35,650
0,0,1,1,1048
1,0,0,7,1371
2,0,1,1,-527
0,0,1,13,1012
1,0,0,3,1343
2,0,1,47,-41
```