



(22) Date de dépôt/Filing Date: 2008/06/13

(41) Mise à la disp. pub./Open to Public Insp.: 2008/12/24

(45) Date de délivrance/Issue Date: 2019/03/12

(62) Demande originale/Original Application: 2 691 229

(30) Priorité/Priority: 2007/06/19 (US11/765,306)

(51) Cl.Int./Int.Cl. *G06Q 40/04* (2012.01)

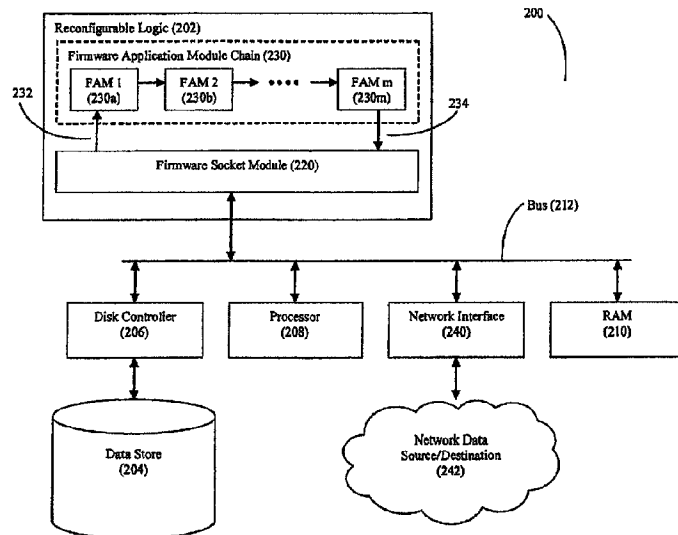
(72) Inventeurs/Inventors:
PARSONS, SCOTT, US;
TAYLOR, DAVID E., US;
SCHUEHLER, DAVID VINCENT, US;
FRANKLIN, MARK A., US;
CHAMBERLAIN, ROGER D., US

(73) Propriétaire/Owner:
IP RESERVOIR, LLC, US

(74) Agent: OSLER, HOSKIN & HARCOURT LLP

(54) Titre : METHODE ET APPAREIL DE TRAITEMENT HAUTE VITESSE DE RENSEIGNEMENTS FINANCIERS A DES FINS DE GESTION DE DOCUMENTATION FINANCIERE

(54) Title: METHOD AND APPARATUS FOR HIGH SPEED PROCESSING OF FINANCIAL INFORMATION FOR FINANCIAL RECORD MANAGEMENT



(57) **Abrégé/Abstract:**

An apparatus for processing financial market data, the apparatus comprising: a reconfigurable logic device with firmware logic deployed thereon, the firmware logic being configured to perform a specified data processing operation on the financial market data; and a processor in communication with the reconfigurable logic device, the processor being programmed with software logic, the software logic being configured to (1) receive the financial market data as an input, and (2) manage a flow of the received financial market data into and out of the reconfigurable logic device.

ABSTRACT

An apparatus for processing financial market data, the apparatus comprising: a reconfigurable logic device with firmware logic deployed thereon, the firmware logic being configured to perform a specified data processing operation on the financial market data; and a processor in communication with the reconfigurable logic device, the processor being programmed with software logic, the software logic being configured to (1) receive the financial market data as an input, and (2) manage a flow of the received financial market data into and out of the reconfigurable logic device.

Method and Apparatus for High Speed Processing of Financial
Information for Financial Record Management

This application is a division of Canadian Serial No. 2,691,229,
filed June 13, 2008.

Cross-Reference and Priority Claim to Related Patent Applications:

This application claims priority to provisional patent
application 60/814,796, filed June 19, 2006, and entitled "High
5 Speed Processing of Financial Information Using FPGA Devices".

This patent application is related to the following patent
applications: U.S. patent application 09/545,472 (filed April 7,
2000, and entitled "Associative Database Scanning and Information
Retrieval", now U.S. patent 6,711,558), U.S. patent application
10 10/153,151 (filed May 21, 2002, and entitled "Associative Database
Scanning and Information Retrieval using FPGA Devices", now U.S.
Patent 7,139,743), published PCT applications WO 05/048134 and WO
05/026925 (both filed May 21, 2004, and entitled "Intelligent Data
Storage and Processing Using FPGA Devices"), published PCT patent
15 application WO 06/096324 (filed February 22, 2006, entitled "Method
and Apparatus for Performing Biosequence Similarity Searching", and
published as 2007/0067108), U.S. patent application 11/293,619
(filed December 2, 2005, entitled "Method and Device for High
Performance Regular Expression Pattern Matching", and published as
20 2007/0130140), U.S. patent application 11/339,892 (filed January 26,
2006, and entitled "Firmware Socket Module for FPGA-Based Pipeline
Processing", and published as 2007/0174841), U.S. patent application
11/381,214 (filed May 2, 2006, and entitled "Method and Apparatus
for Approximate Pattern Matching", and published as 2007/0260602),
25 U.S. patent application 11/561,615 (filed November 20, 2006,
entitled "Method and Apparatus for Processing Financial Information
at Hardware Speeds Using FPGA Devices", and published as
2007/0078837), and U.S. patent application 11/760,211 (filed June 8,
2007, and entitled "Method and System for High Speed Options
30 Pricing", and published as 2007/0294157).

Field of the Invention:

The present invention relates to the field of data processing platforms for financial market data.

5

Background and Summary of the Invention:

Speed of information delivery is a valuable dimension to the financial instrument trading and brokerage industry. The ability of a trader to obtain pricing information on financial instruments such as stocks, bonds and particularly options as quickly as possible cannot be understated; improvements in information delivery delay on the order of fractions of a second can provide important value to traders.

For example, suppose there is an outstanding "bid" on stock X that is a firm quote to buy 100 shares of Stock X for \$21.50 per share. Also suppose there are two traders, A and B, each trying to sell 100 shares of stock X, but would prefer not to sell at a price of \$21.50. Next, suppose another party suddenly indicates a willingness to buy 100 shares of Stock X for a price of \$21.60. A new quote for that amount is then submitted, which sets the "best bid" for Stock X to \$21.60, up 10 cents from its previous value of \$21.50. The first trader, A or B, to see the new best bid price for Stock X and issue a counter-party order to sell Stock X will "hit the bid", and sell his/her Stock X for \$21.60 per share. The other trader will either have to settle for selling his/her shares of Stock X for the lower \$21.50 price or will have to decide not to sell at all at that lower price. Thus, it can be seen that speed of information delivery can often translate into actual dollars and cents for traders, which in large volume situations, can translate to significant sums of money.

In an attempt to promptly deliver financial information to interested parties such as traders, a variety of market data platforms have been developed for the purpose of ostensible "real time" delivery of streaming bid, offer, and trade information for financial instruments to traders. Figure 1 illustrates an exemplary platform that is currently known in the art. As shown in Figure 1, the market data platform 100 comprises a plurality of functional

units 102 that are configured to carry out data processing operations such as the ones depicted in units 102, whereby traders at workstations 104 have access to financial data of interest and whereby trade information can be sent to various exchanges or other outside systems via output path 110. The purpose and details of the functions performed by functional units 102 are well-known in the art. A stream 106 of financial data arrives at the system 100 from an external source such as the exchanges themselves (e.g., NYSE, NASDAQ, etc.) over private data communication lines or from extranet providers such as Savvis or BT Radians. The financial data source stream 106 comprises a series of messages that individually represent a new offer to buy or sell a financial instrument, an indication of a completed sale of a financial instrument, notifications of corrections to previously-reported sales of a financial instrument, administrative messages related to such transactions, and the like. As used herein, a "financial instrument" refers to a contract representing equity ownership, debt or credit, typically in relation to a corporate or governmental entity, wherein the contract is saleable. Examples of "financial instruments" include stocks, bonds, commodities, currency traded on currency markets, etc. but would not include cash or checks in the sense of how those items are used outside financial trading markets (i.e., the purchase of groceries at a grocery store using cash or check would not be covered by the term "financial instrument" as used herein; similarly, the withdrawal of \$100 in cash from an Automatic Teller Machine using a debit card would not be covered by the term "financial instrument" as used herein). Functional units 102 of the system then operate on stream 106 or data derived therefrom to carry out a variety of financial processing tasks. As used herein, the term "financial market data" refers to the data contained in or derived from a series of messages that individually represent a new offer to buy or sell a financial instrument, an indication of a completed sale of a financial instrument, notifications of corrections to previously-reported sales of a financial instrument, administrative messages related to such transactions, and the like. The term "financial market source data" refers to a feed of financial market data directly from a data

source such as an exchange itself or a third party provider (e.g., a Savvis or BT Radianz provider). The term "financial market secondary data" refers to financial market data that has been derived from financial market source data, such as data produced by a feed compression operation, a feed handling operation, an option pricing operation, etc.

Because of the massive computations required to support such a platform, current implementations known to the inventors herein typically deploy these functions across a number of individual computer systems that are networked together, to thereby achieve the appropriate processing scale for information delivery to traders with an acceptable degree of latency. This distribution process involves partitioning a given function into multiple logical units and implementing each logical unit in software on its own computer system/server. The particular partitioning scheme that is used is dependent on the particular function and the nature of the data with which that function works. The inventors believe that a number of different partitioning schemes for market data platforms have been developed over the years. For large market data platforms, the scale of deployment across multiple computer systems and servers can be physically massive, often filling entire rooms with computer systems and servers, thereby contributing to expensive and complex purchasing, maintenance, and service issues.

This partitioning approach is shown by Figure 1 wherein each functional unit 102 can be thought of as its own computer system or server. Buses 108 and 110 can be used to network different functional units 102 together. For many functions, redundancy and scale can be provided by parallel computer systems/servers such as those shown in connection with options pricing and others. To the inventors' knowledge, these functions are deployed in software that is executed by the conventional general purpose processors (GPPs) resident on the computer systems/servers 102. The nature of general purpose processors and software systems in the current state of the art known to the inventors herein imposes constraints that limit the performance of these functions. Performance is typically measured as some number of units of computational work that can be performed per unit time on a system (commonly called "throughput"), and the

time required to perform each individual unit of computational work from start to finish (commonly called "latency" or delay). Also, because of the many physical machines required by system 100, communication latencies are introduced into the data processing operations because of the processing overhead involved in transmitting messages to and from different machines.

Despite the improvements to the industry that these systems have provided, the inventors herein believe that significant further improvements can be made. In effect, the inventors believe that a technical problem exists in the art with respect to how one can accelerate the processing of financial market data to thereby enable the ultimate delivery of that data to downstream applications and/or parties with a very low degree of latency. As a part of a solution to such a technical problem, the inventors herein disclose the use of the underlying technology disclosed in the related patents and patent applications listed above to fundamentally change the system architecture in which market data platforms are deployed.

In above-referenced related patent application 10/153,151, it was first disclosed that reconfigurable logic, such as Field Programmable Gate Arrays (FPGAs), can be deployed to process streaming financial information at hardware speeds. As examples, the 10/153,151 application disclosed the use of FPGAs to perform data reduction operations on streaming financial information, with specific examples of such data reduction operations being a minimum price function, a maximum price function, and a latest price function. (See also the above-referenced 11/561,615 patent application).

Since that time, the inventors herein have devised a number of streamlined applications which have greatly expanded the scope of functionality for processing streams of financial information with reconfigurable logic. The inventors believe that these applications themselves have been architected in a manner that accelerates the processing of financial market data. With the embodiments described herein, vast amounts of streaming financial information can be processed with varying degrees of complexity at hardware speeds via reconfigurable logic deployed in hardware appliances that greatly

consolidate the distributed GPP architecture shown in Figure 1 such that a market data platform built in accordance with the principles of the present invention can be implemented within fewer and much smaller appliances while providing faster data processing

5 capabilities relative to the conventional market data platform as illustrated by Figure 1; for example, the inventors envision that a 5:1 or greater reduction of appliances relative to the system architecture of Figure 1 can be achieved in the practice of the present invention.

10 As used herein, the term "general-purpose processor" (or GPP) refers to a hardware device that fetches instructions and executes those instructions (for example, an Intel Xeon processor or an AMD Opteron processor). The term "reconfigurable logic" refers to any logic technology whose form and function can be significantly
15 altered (i.e., reconfigured) in the field post-manufacture. This is to be contrasted with a GPP, whose function can change post-manufacture, but whose form is fixed at manufacture. The term "software" will refer to data processing functionality that is deployed on a GPP. The term "firmware" will refer to data
20 processing functionality that is deployed on reconfigurable logic.

Thus, as embodiments of the present invention, the inventors herein disclose a variety of data processing pipelines implemented in firmware deployed on reconfigurable logic, wherein a stream of financial data can be processed through these pipelines at hardware
25 speeds. The pipelines include various firmware application modules that themselves are architected to provide acceleration. An exemplary firmware application module disclosed herein is a firmware application module that provides accelerated symbol mapping for received financial market data messages. Another exemplary firmware
30 application module disclosed herein is a firmware application module that provides accelerated global exchange identifier mapping for received financial market data messages. Another exemplary firmware application module disclosed herein is a firmware application module that provides accelerated financial instrument record updating based
35 on received financial market data messages. Such a module can be used to realize a function such as a high speed Last Value Cache (LVC). Yet another exemplary firmware application module disclosed

herein is a firmware application module that provides accelerated interest list and/or entitlement filtering for financial market data. Yet another exemplary firmware application module

disclosed herein is a firmware application module that provides accelerated message parsing for received financial market data messages. Still another exemplary firmware application module disclosed herein is a firmware application module that provides accelerated message generation for outgoing financial market data messages.

In accordance with an embodiment of the present invention, there is provided an apparatus for processing financial market data, the apparatus comprising: a reconfigurable logic device with firmware logic deployed thereon, the firmware logic being configured to perform a specified data processing operation on the financial market data; and a processor in communication with the reconfigurable logic device, the processor being programmed with software logic, the software logic being configured to (1) receive the financial market data as an input, and (2) manage a flow of the received financial market data into and out of the reconfigurable logic device.

Another embodiment of the present invention provides a method for processing financial market data, the method comprising: controlling with software logic a flow of financial market data from the software logic to and from firmware logic; and performing a specified financial data processing operation on financial market data with firmware logic on a reconfigurable logic device in response to the software logic controlling step.

A still further embodiment provides an apparatus comprising: a computer system comprising a processor and a reconfigurable logic device that are configured to cooperate with each other to process streaming financial market data relating to

a plurality of financial instruments and manage a delivery of the processed financial market data for consumption by a plurality of clients; wherein the processor is configured to execute an
5 operating system, the operating system including a user space for a user mode and a kernel space for a kernel mode; wherein the reconfigurable logic device is configured to (1) perform a processing operation on the streaming financial market data to thereby generate processed financial market data and (2) stream
10 the processed financial market data out of the reconfigurable logic device into a memory within the kernel space of the operating system via DMA operations; and wherein a driver that executes within the kernel space of the operating system while the operating system is in the kernel mode is configured to (1)
15 read the processed financial market data from the memory, and (2) selectively route the processed financial market data for delivery to the clients via a path that is wholly within the kernel space.

A further embodiment provides a method for processing
20 financial market data using a system comprising a processor and a reconfigurable logic device, the method comprising: the processor executing an operating system, the operating system including a user space for a user mode and a kernel space for a kernel mode, wherein the executing step comprises: a network protocol stack
25 executed by the operating system receiving a feed of streaming financial market data for a plurality of financial instruments; a first driver executed within the kernel space of the operating system while the operating system is in the kernel mode (1) maintaining a kernel level interface into the network protocol
30 stack, and (2) copying the streaming financial market data from

the network protocol stack into a shared memory that is mapped into the kernel space and the user space, wherein the copying is performed by the driver without the operating system

5 transitioning to the user mode; user mode code executed within the user space of the operating system (1) accessing the financial market data from the shared memory, (2) normalizing the accessed financial market data, and (3) writing the normalized financial market data to additional shared memory that is mapped
10 into the kernel space and the user space; and a second driver executed within the kernel space of the operating system while the operating system is in the kernel mode facilitating DMA transfers of the normalized financial market data from the additional shared memory into the reconfigurable logic device;
15 the reconfigurable logic device streaming the normalized financial market data into the reconfigurable logic device from the additional shared memory via DMA operations, the reconfigurable logic device having a resident firmware pipeline; and the firmware pipeline (1) receiving the streaming normalized
20 financial market data, and (2) performing a processing operation on the received streaming normalized financial market data to generate processed financial market data for delivery to a plurality of clients.

Also disclosed is an apparatus for processing a plurality of financial market data messages, each financial market data message comprising financial market data and being associated with a financial instrument, the apparatus comprising: a reconfigurable logic device comprising a plurality of pipelined data processing blocks implemented in firmware logic on the reconfigurable logic device, the pipelined data processing blocks comprising a first data processing block and a second data processing block downstream from the first data processing block; wherein the first data processing block is configured to retrieve pointer data from a memory in response to a plurality of received financial market data messages, the retrieved pointer data being configured to point to a plurality of records that are associated with the financial instruments that are associated with the received financial market data messages; and wherein the second data processing block is configured to retrieve a plurality of financial instrument records from a memory based on the retrieved pointer data, the retrieved financial instrument records being associated with the financial instruments that are associated with the received financial market data messages; and wherein the first and second data processing blocks are configured to operate simultaneously with respect to each other in a pipelined fashion.

Another embodiment of the present invention provides a method for processing a plurality of financial market data messages, each financial market data message comprising financial market data and being associated with a financial instrument, the method comprising: receiving the financial market data messages; and processing a plurality of the received financial market data messages through a pipeline deployed in firmware logic on a reconfigurable logic device, the pipeline comprising a first data processing block and a second data processing block that is downstream from the first data processing block, the processing step comprising: the first data processing block retrieving pointer data from a memory in response to the processed financial market data messages, the retrieved pointer data pointing to a plurality of records that are associated with the financial instruments that are associated with the processed financial market data messages; and the second data processing block retrieving a plurality of financial instrument records from a memory based on the retrieved pointer data, the retrieved financial instrument records being associated with the financial instruments that are associated with the received financial market data messages; and the first and second data processing blocks operating simultaneously with respect to each other in a pipelined fashion.

A further embodiment provides a method for processing a plurality of financial market data messages, each financial market data message comprising financial market data and being associated with a financial instrument, the method: comprising: receiving streaming data, the streaming data comprising a

plurality of data fields, a plurality of financial instrument identifiers, and a plurality of exchange identifiers, each data field comprising financial market data for a financial instrument and being associated with a financial instrument identifier and an exchange identifier; processing the streaming data through a pipeline deployed in firmware logic on a reconfigurable logic device, the pipeline comprising a first data processing block, a second data processing block downstream from the first data processing block, and a third data processing block downstream from the second data processing block, wherein the processing step comprises: the first data processing block resolving the streaming data to a plurality of memory addresses based on the financial instrument identifiers and exchange identifiers in the streaming data; the second data processing block retrieving a plurality of composite financial instrument records and a plurality of regional financial instrument records from a memory based on the resolved memory addresses; and the third data processing block performing at least one financial data processing operation on the data fields in the streaming data to generate a plurality of updates for the retrieved composite financial instrument records and the retrieved regional financial instrument records; and the first data processing block, the second data processing block, and the third data processing block operating simultaneously with respect to each other in a pipelined fashion as the streaming data is processed through the pipeline.

Also disclosed as an embodiment of the invention is a ticker
5 plant that is configured to process financial market data with a
combination of software logic and firmware logic. Through firmware
pipelines deployed on the ticker plant and efficient software
control and management over data flows to and from the firmware
pipelines, the inventors herein believe that the ticker plant of the
10 preferred embodiment is capable of greatly accelerating the speed
with which financial market data is processed. In a preferred
embodiment, financial market data is first processed within the
ticker plant by software logic. The software logic controls and
manages the flow of received financial market data into and out of
15 the firmware logic deployed on the reconfigurable logic device(s),
preferably in a manner such that each financial market data message
travels only once from the software logic to the firmware logic and
only once from the firmware logic back to the software logic. As
used herein, the term "ticker plant" refers to a plurality of
20 functional units, such as functional units 102 depicted in Figure 1,
that are arranged together to operate on a financial market data
stream 106 or data derived therefrom.

These and other features and advantages of the present
invention will be understood by those having ordinary skill in the
25 art upon review of the description and figures hereinafter.

Brief Description of the Drawings:

Figure 1 depicts an exemplary system architecture for a
conventional market data platform;

30 Figure 2 is a block diagram view of an exemplary system
architecture in accordance with an embodiment of the present
invention;

35

Figure 3 illustrates an exemplary framework for the deployment of software and firmware for an embodiment of the present invention;

Figure 4(a) is a block diagram view of a preferred printed circuit board for installation into a market data platform to carry out data processing tasks in accordance with the present invention;

Figure 4(b) is a block diagram view of an alternate printed circuit board for installation into a market data platform to carry out data processing tasks in accordance with the present invention;

Figure 5 illustrates an example of how the firmware application modules of a pipeline can be deployed across multiple FPGAs;

Figure 6 illustrates an exemplary architecture for a market data platform in accordance with an embodiment of the present invention;

Figure 7 illustrates an exemplary firmware application module pipeline for a message transformation from the Financial Information Exchange (FIX) format to the FIX Adapted for Streaming (FAST) format;

Figure 8 illustrates an exemplary firmware application module pipeline for a message transformation from the FAST format to the FIX format;

Figure 9 illustrates an exemplary firmware application module pipeline for message format transformation, message data processing, and message encoding; and

Figure 10 illustrates another exemplary firmware application module pipeline for message format transformation, message data processing, and message encoding.

Figure 11 depicts another exemplary firmware application module pipeline for performing the functions including symbol mapping, Last Value Cache (LVC) updates, interest and entitlement filtering;

Figure 12 depicts an exemplary embodiment of a compression function used to generate a hash key within a firmware application module configured to perform symbol mapping;

Figure 13 depicts an exemplary embodiment of a hash function for deployment within a firmware application module configured to perform symbol mapping;

Figure 14 depicts a preferred embodiment for generating a global exchange identifier (GEID) within a firmware application module configured to perform symbol mapping;

Figures 15(a) and (b) depict an exemplary embodiment for a
5 firmware application module configured to perform Last Value Cache (LVC) updating;

Figure 16 depicts an exemplary embodiment for a firmware application module configured to perform interest and entitlement filtering;

10 Figure 17 depicts an exemplary embodiment of a ticker plant where the primary data processing functional units are deployed in reconfigurable hardware and where the control and management functional units are deployed in software on general purpose processors;

15 Figure 18 depicts an exemplary data flow for inbound exchange traffic in the ticker plant of Figure 17;

Figure 19 depicts an exemplary processing of multiple thread groups within the ticker plant of Figure 17;

20 Figure 20 depicts an example of data flow between the hardware interface driver and reconfigurable logic within the ticker plant of Figure 17;

Figure 21 depicts an example of data flows within the ticker plant of Figure 17 for data exiting the reconfigurable logic; and

25 Figure 22 depicts an exemplary model for managing client connections with the ticker plant of Figure 17.

Detailed Description of the Preferred Embodiment:

30 Figure 2 depicts an exemplary system 200 in accordance with the present invention. In this system, a reconfigurable logic device 202 is positioned to receive data that streams off either or both a disk subsystem defined by disk controller 206 and data store 204 (either directly or indirectly by way of system memory such as RAM 210) and a network data source/destination 242 (via network
35 interface 240) . Preferably, data streams into the reconfigurable logic device by way of system bus 212, although other design architectures are possible (see Figure 4(b)). Preferably, the

reconfigurable logic device 202 is a FPGA, although this need not be the case. System bus 212 can also interconnect the reconfigurable logic device 202 with the computer system's main processor 208 as well as the computer system's RAM 210. The term "bus" as used
5 herein refers to a logical bus which encompasses any physical interconnect for which devices and locations are accessed by an address. Examples of buses that could be used in the practice of the present invention include, but are not limited to the PCI family of buses (e.g., PCI-X and PCI-Express) and HyperTransport buses. In
10 a preferred embodiment, system bus 212 may be a PCI-X bus, although this need not be the case.

The data store can be any data storage device/system, but is preferably some form of a mass storage medium. For example, the data store 204 can be a magnetic storage device such as an array of
15 Seagate disks. However, it should be noted that other types of storage media are suitable for use in the practice of the invention. For example, the data store could also be one or more remote data storage devices that are accessed over a network such as the Internet or some local area network (LAN). Another
20 source/destination for data streaming to or from the reconfigurable logic device 202, is network 242 by way of network interface 240, as described above. In the financial industry, a network data source (e.g., the exchanges themselves, a third party provider, etc.) can provide the financial data stream 106 described above in connection
25 with Figure 1.

The computer system defined by main processor 208 and RAM 210 is preferably any commodity computer system as would be understood by those having ordinary skill in the art. For example, the computer system may be an Intel Xeon system or an AMD Opteron
30 system.

The reconfigurable logic device 202 has firmware modules deployed thereon that define its functionality. The firmware socket module 220 handles the data movement requirements (both command data and target data) into and out of the reconfigurable logic device,
35 thereby providing a consistent application interface to the firmware application module (FAM) chain 230 that is also deployed on the reconfigurable logic device. The FAMs 230i of the FAM chain 230 are

configured to perform specified data processing operations on any data that streams through the chain 230 from the firmware socket module 220. Preferred examples of FAMs that can be deployed on reconfigurable logic in accordance with a preferred embodiment of the present invention are described below.

The specific data processing operation that is performed by a FAM is controlled/parameterized by the command data that FAM receives from the firmware socket module 220. This command data can be FAM-specific, and upon receipt of the command, the FAM will arrange itself to carry out the data processing operation controlled by the received command. For example, within a FAM that is configured to compute an index value (such as the Dow Jones Industrial Average), the FAM's index computation operation can be parameterized to define which stocks will be used for the computation and to define the appropriate weighting that will be applied to the value of each stock to compute the index value. In this way, a FAM that is configured to compute an index value can be readily re-arranged to compute a different index value by simply loading new parameters for the different index value in that FAM.

Once a FAM has been arranged to perform the data processing operation specified by a received command, that FAM is ready to carry out its specified data processing operation on the data stream that it receives from the firmware socket module. Thus, a FAM can be arranged through an appropriate command to process a specified stream of data in a specified manner. Once the FAM has completed its data processing operation, another command can be sent to that FAM that will cause the FAM to re-arrange itself to alter the nature of the data processing operation performed thereby. Not only will the FAM operate at hardware speeds (thereby providing a high throughput of target data through the FAM), but the FAMs can also be flexibly reprogrammed to change the parameters of their data processing operations.

The FAM chain 230 preferably comprises a plurality of firmware application modules (FAMs) 230a, 230b, ... that are arranged in a pipelined sequence. As used herein, "pipeline", "pipelined sequence", or "chain" refers to an arrangement of FAMs wherein the output of one FAM is connected to the input of the next FAM in the

sequence. This pipelining arrangement allows each FAM to independently operate on any data it receives during a given clock cycle and then pass its output to the next downstream FAM in the sequence during another clock cycle.

5 A communication path 232 connects the firmware socket module 220 with the input of the first one of the pipelined FAMs 230a. The input of the first FAM 230a serves as the entry point into the FAM chain 230. A communication path 234 connects the output of the final one of the pipelined FAMs 230m with the firmware socket module 220. The output of the final FAM 230m serves as the exit point from the FAM chain 230. Both communication path 232 and communication path 234 are preferably multi-bit paths.

Figure 3 depicts an exemplary framework for the deployment of applications on the system 200 of Figure 2. The top three layers of Figure 3 represent functionality that is executed in software on the computer system's general-purpose processor 208. The bottom two layers represent functionality that is executed in firmware on the reconfigurable logic device 202.

20 The application software layer 300 corresponds to high level functionality such as the type of functionality wherein one or more users interact with the application to define which data processing operations are to be performed by the FAMs and to define what data those data processing operations are to be performed upon.

25 The next layer is the module application programming interface (API) layer 302 which comprises a high level module API 302a and a low level module API 302b. The high level module API 302a can provide generic services to application level software (for example, managing callbacks). The low level module API 302b manages the operation of the operating system (OS) level/device driver software 304. A software library interface 310 interfaces the high level module API 302a with the low level module API 302b. Additional details about this software library interface can be found in the above-referenced patent application 11/339,892.

35 The interface between the device driver software 304 and the firmware socket module 220 serves as the hardware/software interface 312 for the system 200. The details of this interface 312 are

described in greater detail in the above-referenced patent application 11/339,892.

The interface between the firmware socket module 220 and the FAM chain 230 is the firmware module interface 314. The details of this interface are described in greater detail in the above-referenced patent application 11/339,892.

Figure 4(a) depicts a printed circuit board or card 400 that can be connected to the PCI-X bus 212 of a commodity computer system for use in a market data platform. In the example of Figure 4(a), the printed circuit board includes an FPGA 402 (such as a Xilinx Virtex II FPGA) that is in communication with a memory device 404 and a PCI-X bus connector 406. A preferred memory device 404 comprises SRAM and SDRAM memory. A preferred PCI-X bus connector 406 is a standard card edge connector.

Figure 4(b) depicts an alternate configuration for a printed circuit board/card 400. In the example of Figure 4(b), a private bus 408 (such as a PCI-X bus), a network interface controller 410, and a network connector 412 are also installed on the printed circuit board 400. Any commodity network interface technology can be supported, as is understood in the art. In this configuration, the firmware socket 220 also serves as a PCI-X to PCI-X bridge to provide the processor 208 with normal access to the network(s) connected via the private PCI-X bus 408.

It is worth noting that in either the configuration of Figure 4(a) or 4(b), the firmware socket 220 can make memory 404 accessible to the PCI-X bus, which thereby makes memory 404 available for use by the OS kernel 304 as the buffers for transfers from the disk controller and/or network interface controller to the FAMs. It is also worth noting that while a single FPGA 402 is shown on the printed circuit boards of Figures 4(a) and (b), it should be understood that multiple FPGAs can be supported by either including more than one FPGA on the printed circuit board 400 or by installing more than one printed circuit board 400 in the computer system. Figure 5 depicts an example where numerous FAMs in a single pipeline are deployed across multiple FPGAs.

As shown in Figures 2-4, inbound data (from the kernel 304 to the card 400) is moved across the bus 212 in the computer system to

the firmware socket module 220 and then delivered by the firmware socket module 220 to the FAM chain 230. Outbound data (from the card 400 to the kernel 304) are delivered from the FAM chain 230 to the firmware socket module 220 and then delivered by the firmware
5 socket module 220 across the PCI-X bus to the software application executing on the computer system. As shown in Figure 3, the three interacting interfaces that are used are the firmware module interface 314, the hardware/software interface 312, and the software library interface 310.

10 In an effort to improve upon conventional market data platforms, the inventors herein disclose a new market data platform architecture, an embodiment of which is shown in Figure 6. The market data platform 600 shown in Figure 6 consolidates the functional units 102 shown in Figure 1 into much fewer physical
15 devices and also offloads much of the data processing performed by the GPPs of the functional units 102 to reconfigurable logic.

For example, with the architecture of Figure 6, the feed compressor 602 can be deployed in an appliance such as system 200 shown in Figure 2. The reconfigurable logic 202 can be implemented
20 on a board 400 as described in connection with Figures 4(a) or 4(b). Feed compressor 602 is used to compress the content of the financial data stream 106 arriving from various individual sources. Examples of compression techniques that can be used include the open standard "glib" as well as any proprietary compression technique that may be
25 used by a practitioner of the present invention. Appropriate FAM modules and a corresponding FAM pipeline to implement such a feed compression operation can be carried out by a person having ordinary skill in the art using the design techniques described in connection with the above-referenced patent and patent applications and basic
30 knowledge in the art concerning feed compression. As a result, a variety of hardware templates available for loading on reconfigurable logic can be designed and stored for use by the market data platform 600 to implement a desired feed compression operation.

35 Preferably, the feed compressor device 602 is deployed in a physical location as close to the feed source 106 as possible, to thereby reduce communication costs and latency. For example, it

would be advantageous to deploy the feed compressor device 602 in a data center of an extranet provider (e.g., Savvis, BT Radianz, etc.) due to the data center's geographic proximity to the source of the financial market data 106. Because the compression reduces message sizes within the feed stream 106, it will be advantageous to perform the compression prior to the stream reaching wide area network (WAN) 620a; thereby improving communication latency through the network because of the smaller message sizes.

WAN 620 preferably comprises an extranet infrastructure or private communication lines for connection, on the inbound side, to the feed handlers deployed in device 604. On the outbound side, WAN 620 preferably connects with device 606, as explained below. It should be noted that WAN 620 can comprise a single network or multiple networks 620a and 620b segmented by their inbound/outbound role in relation to platform 600. It is also worth noting that a news feed with real-time news wire reports can also be fed into WAN 620a for delivery to device 604.

Device 604 can be deployed in an appliance such as system 200 shown in Figure 2. The reconfigurable logic 202 can be implemented on a board 400 as described in connection with Figures 4(a) or 4(b). Whereas the conventional GPP-based system architecture shown in Figure 1 deployed the functional units of feed handling/ticker plant, rule-based calculation engines, an alert generation engine, options pricing, Last Value Cache (LVC) servers supplying snapshot and/or streaming interfaces, historical time-series oriented databases with analytics, and news databases with search capabilities in software on separate GPPs, the architecture of Figure 6 can consolidate these functions, either partially or in total, in firmware resident on the reconfigurable logic (such as one or more FPGAs) of device 604.

Feed handlers, which can also be referred to as feed producers, receive the real-time data stream, either compressed from the feed compressor 602 as shown in Figure 6 or uncompressed from a feed source, and converts that compressed or uncompressed stream from a source-specific format (e.g., an NYSE format) to a format that is common throughout the market data platform 600. This conversion process can be referred to as "normalization". This

"normalization" can be implemented in a FAM chain that transforms the message structure, converts the based units of specific field values within each message, maps key field information to the common format of the platform, and fills in missing field information from
5 cached or database records. In situations where the received feed stream is a compressed feed stream, the feed handler preferably also implements a feed decompression operation.

LVCs maintain a database of financial instrument records whose functionality can be implemented in a FAM pipeline. Each record
10 represents the current state of that financial instrument in the market place. These records are updated in real-time via a stream of update messages received from the feed handlers. The LVC is configured to respond to requests from other devices for an up-to-the-instant record image for a set of financial instruments and
15 redistribute a selective stream of update messages pertaining to those requested records, thereby providing real-time snapshots of financial instrument status. From these snapshots, information such as the "latest price" for a financial instrument can be determined, as described in the above-referenced 10/153,151 application.

20 Rule-based calculation engines are engines that allow a user to create his/her own synthetic records whose field values are derived from calculations performed against information obtained from the LVC, information extracted from a stream of update messages generated from the LVC, or from alternate sources. These rule-based
25 calculation engines are amenable to implementation in a FAM pipeline. It should also be noted that the rule-based calculation engine can be configured to create new synthetic fields that are included in existing records maintained by the LVC. The new values computed by the engine are computed by following a set of rules or
30 formulas that have been specified for each synthetic field. For example, a rule-based calculation engine can be configured to compute a financial instrument's Volume Weighted Average Price (VWAP) via a FAM pipeline that computes the VWAP as the sum of P x S for every trade meeting criteria X, wherein P equals the trade price
35 and wherein S equals the trade size. Criteria X can be parameterized into a FAM filter that filters trades based on size, types, market conditions, etc. Additional examples of rule-based

calculations that can be performed by the rule-based calculation engine include, but are not limited to, a minimum price calculation for a financial instrument, a maximum price calculation for a financial instrument, a Top 10 list for a financial instrument or set of financial instruments, etc.

An alert generation engine can also be deployed in a FAM pipeline. Alert generation engines are similar to a rule-based calculation engine in that they monitor the current state of a financial instrument record (or set of financial instrument records), and the alert generation engine will trigger an alert when any of a set of specified conditions is met. An indication is then delivered via a variety of means to consuming applications or end users that wish to be notified upon the occurrence of the alert.

Option pricing is another function that is highly amenable to implementation via a FAM pipeline. An "option" is a derivative financial instrument that is related to an underlying financial instrument, and the option allows a person to buy or sell that underlying financial instrument at a specific price at some specific time in the future. An option pricing engine is configured to perform a number of computations related to these options and their underlying instruments (e.g., the theoretical fair market value of an option or the implied volatility of the underlying instrument based upon the market price of the option). A wide array of computational rules can be used for pricing options, as is known in the art. Most if not all industry-accepted techniques for options pricing are extremely computation intensive which introduces significant latency when the computations are performed in software. However, by implementing option pricing in a FAM pipeline, the market data platform 600 can significantly speed up the computation of option pricing, thereby providing an important edge to traders who use the present invention. An example of options pricing functionality that can be deployed in firmware is described in pending U.S. patent application 11/760,211, filed June 8, 2007.

A time series database is a database that maintains a record for each trade or quote event that occurs for a set of financial instruments. This information may be retrieved upon request and

returned in an event-by-event view. Alternative views are available wherein events are "rolled up" by time intervals and summarized for each interval. Common intervals include monthly, weekly, daily, and "minute bars" where the interval is specified to be some number of
5 minutes. The time series database also preferably compute a variety of functions against these historic views of data, including such statistical measures as volume weighted average price (VWAP), money flow, or correlations between disparate financial instruments.

A news database maintains a historical archive of news stories
10 that have been received from a news wire feed by way of the feed handler. The news database is preferably configured to allow end users or other applications to retrieve news stories or headlines based upon a variety of query parameters. These query parameters often include news category assignments, source identifiers, or even
15 keywords or keyword phrases. The inventors herein note that this searching functionality can also be enhanced using the search and data matching techniques described in the above-referenced patent and patent applications.

Using the techniques described herein, a variety of hardware
20 templates available for loading on reconfigurable logic can be designed and stored in memory (such as on a disk embodied by data store 204 in connection with Figure 2) for use by the market data platform 600 to implement a desired data processing function. Persistent data storage unit 630 can be accessible to device 604 as
25 device 604 processes the feed stream in accordance with the functionality described above. Storage 630 can be embodied by data store 204 or other memory devices as desired by a practitioner of the invention.

Traders at workstations 104 (or application programs 150
30 running on an entity's own trading platform) can then access the streaming financial data processed by device 604 via a connection to local area network (LAN) 622. Through this LAN connection, workstations 104 (and application program 15) also have access to the data produced by devices 606, 608, 610, 612, 614, and 616. Like
35 devices 602 and 604, devices 606, 608, 610, 612, 614, and 616 can also be deployed in an appliance such as system 200 shown in Figure 2, wherein the reconfigurable logic 202 of system 200 can be

implemented on a board 400 as described in connection with Figures 4(a) or 4(b).

Device 606 preferably consolidates the following functionality at least partially into firmware resident on reconfigurable logic:

5 an order book server; an order router; direct market access gateways to exchanges, Electronic Communication Networks (ECNs), and other liquidity pools; trading engines; an auto-quote server; and a compliance journal.

An "order book server" is similar to a LVC in that the order

10 book server maintains a database in memory (e.g., in memory device 404 on board 400) of financial instrument records, and keeps that database up to date in real-time via update messages received from the feed handlers. For each record, the order book server preferably maintains a sorted list of the bids and offers associated

15 with all outstanding orders for that instrument. This list is known as the "book" for that instrument. The order information for each instrument is received from a variety of different trading venues in stream 106 and is aggregated together to form one holistic view of the market for that particular instrument. The order book server is

20 configured to respond to requests from workstation 104 users or application programs 150 to present the book in a number of different ways. There are a variety of different "views", including but not limited to: a "top slice" of the book that returns orders whose prices are considered to be within a specified number of price

25 points of the best price available in the market (the best price being considered to be the "top" of the book); a price aggregate view where orders at the same price point are aggregated together to create entries that are indicative of the total number of orders available at each price point; and an ordinary view with specific

30 trading venues (which are the source of orders) excluded.

An order router is a function that can take a buy or sell order for a specified financial instrument, and based upon a variety of criteria associated with the order itself or the end user or application submitting the order, route the order (in whole or in

35 part) to the most appropriate trading venue, such as an exchange, an Alternate Trading System (ATS), or an ECN.

The direct market access gateway functionality operates to relay orders to a trading venue (such as an exchange, ECN, ATS, etc.) via WAN 620b. Before sending an order out however, the gateway preferably transforms the order message to a format
5 appropriate for the trading venue.

The trading engine functionality can also be deployed on reconfigurable logic. An algorithmic trading engine operates to apply a quantitative model to trade orders of a defined quantity to thereby automatically subdivide that trade order into smaller orders
10 whose timing and size are guided by the goals of the quantitative model so as to reduce the impact that the original trade order may have on the current market price. Also, a black box trading engine operates to automatically generate trades by following a mathematical model that specifies relationships or conditional
15 parameters for an instrument or set of instruments. To aid this processing, the black box trading engine is fed with real-time market data.

An auto-quote server is similar to a black box trading engine. The auto-quote server operates to automatically generate firm quotes
20 to buy or sell a particular financial instrument at the behest of a "market maker"; wherein a "market maker" is a person or entity which quotes a buy and/or sell price in a financial instrument hoping to make a profit on the "turn" or the bid/offer spread.

A feed/compliance journal can also be implemented in a FAM
25 pipeline. The feed/compliance journal functions to store information (in persistent storage 632) related to the current state of the entire market with regard to a particular financial instrument at the time a firm quote or trade order is submitted to a single particular marketplace. The feed/compliance journal can also
30 provide a means for searching storage 632 to provide detailed audit information on the state of the market when a particular firm quote or trade order was submitted. The inventors herein note that this searching functionality can also be enhanced using the search and data matching techniques described in the above-referenced patent
35 and patent applications.

As mentioned above in connection with device 604, appropriate FAM modules and corresponding FAM pipelines to implement these

various functions for device 606 can be carried out by a person having ordinary skill in the art using the design techniques described in connection with the above-referenced patent and patent applications and basic knowledge in the art concerning each
5 function. As a result, a variety of hardware templates available for loading on reconfigurable logic can be designed and stored for use by the market data platform 600 to implement a desired data processing function. Persistent data storage unit 632, which can be embodied by data store 204, can be accessible to device 606 as
10 device 606 processes the feed stream in accordance with the functionality described above.

Device 608 preferably implements an internal matching system/engine in firmware resident on reconfigurable logic. An internal matching system/engine operates to match a buyer's bid with
15 a seller's offer to sell for a particular financial instrument, to thereby execute a deal or trade. An indication of a completed trade is then submitted to the appropriate reporting and settlement systems. The internal matching system/engine may create bids or offers as would a market maker in order to provide an orderly market
20 and a minimum amount of liquidity by following a set of programmatically-defined rules.

Device 610 preferably implements an order management system (OMS) in firmware resident on reconfigurable logic. An OMS operates to facilitate the management of a group of trading accounts,
25 typically on behalf of a broker. The OMS will monitor buy and sell orders to ensure that they are appropriate for the account owner in question based upon his/her account status, credit and risk profiles. The OMS typically incorporates a database via persistent storage 638 (which may be embodied by data store 204) used to hold
30 account information as well as an archive of orders and other activity for each account.

Device 612 preferably implements entitlements and reporting functionality. A market data platform such as system 600 is a mechanism for distributing data content to a variety of end users.
35 Many content providers charge on a per user basis for access to their data content. Such content providers thus prefer a market data platform to have a mechanism to prohibit (or entitle) access to

specific content on an individual user basis. Entitlement systems may also supply a variety of reports that detail the usage of different content sets. To achieve this functionality, device 612, in conjunction with database 634, preferably operates to maintain a database of users, including authentication credentials and entitlement information which can be used by devices 604, 606, 608, 610 and 616 for entitlement filtering operations in conjunction with the data processing operations performed thereby.

Device 614 preferably implements management and monitoring for the market data platform 600. Management and monitoring functionality provides a means for users to operate the applications running within the platform 600 and monitor the operational state and health of individual components thereon. Preferably, the management and monitoring functionality also provides facilities for reconfiguring the components as well as means for performing any other appropriate manual chores associated with running the platform.

Device 616 preferably implements publishing and contribution server functionality. Contribution servers (also known as publishing servers) allow users to convert information obtained from an end-user application (or some other source within his/her enterprise) into a suitable form, and to have it distributed by the market data platform 600.

As mentioned above in connection with devices 604 and 606, appropriate FAM modules and corresponding FAM pipelines to implement these various functions for devices 608, 610, 612, 614, and 616 can be carried out by a person having ordinary skill in the art using the design techniques described in connection with the above-referenced patent and patent applications and basic knowledge in the art concerning each function. As a result, a variety of hardware templates available for loading on reconfigurable logic can be designed and stored for use by the market data platform 600 to implement a desired data processing function. Persistent data storage units 634 and 636 can be accessible to devices 612 and 614 respectively as those devices process the data in accordance with the functionality described above.

In deploying this functionality, at least in part, upon reconfigurable logic, the following modules/submodules of the functions described above are particularly amenable to implementation on an FPGA: fixed record format message parsing, fixed record format message generation, FIX message parsing, FIX message generation, FIX/FAST message parsing, FIX/FAST message generation, message compression, message decompression, interest and entitlement filtering, financial instrument symbol mapping, record ID mapping, price summary LVC update/retrieve/normalize (LVC), order book cache update/retrieve/normalize (OBC), generic LVC (GVC), minute bar generation, programmatic field generation (with LVC, OBC, etc.), historic record search and filter, book-based algorithmic order routing, trade order generation, basket calculation (including ETF, index, and portfolio valuation), and autoquote generation. It should be understood by those having ordinary skill in the art that this list is exemplary only and not exhaustive; additional modules for financial data processing can also be employed in a FAM or FAM pipeline in the practice of the present invention.

With fixed record format message parsing, a fixed format message is decomposed into its constituent fields as defined by a programmable "data dictionary". Entries within the data dictionary describe the fields within each type of message, their positions and sizes within those messages, and other metadata about the field (such as data type, field identifiers, etc.). Preferably, the data dictionary is stored in persistent storage such as data store 204 of the system 200. Upon initialization of the FAM pipeline on board 400, the data dictionary is then preferably loaded into memory 404 for usage by the FAM pipeline during data processing operations.

With fixed record format message generation, a fixed format message is generated by concatenating the appropriate data representing fields into a message record. The message structure and format is described by a programmable data dictionary as described above.

With FIX message parsing, a FIX-formatted message is decomposed into its constituent fields as defined by a programmable data dictionary as described above; FIX being a well-known industry standard for encoding financial message transactions.

With FIX message generation, a FIX-formatted message is generated by concatenating the appropriate data representing the fields into a FIX message record. Once again, the message structure and format is described by a programmable data dictionary as described above.

With FIX/FAST message parsing, a FIX and/or FAST message (FAST being a well known variation of FIX) is decomposed into its constituent fields as defined by a programmable data dictionary as described above.

With FIX/FAST message generation, a FIX-formatted and/or FAST-formatted message is generated by concatenating the appropriate data representing fields into a FIX/FAST message record. The message structure and format is defined by a programmable data dictionary as described above.

With message compression, a message record is compressed so as to require less space when contained in a memory device and to require less communication bandwidth when delivered to other systems. The compression technique employed is preferably sufficient to allow for reconstruction of the original message when the compressed message is processed by a corresponding message decompression module.

With interest and entitlement filtering, a stream of messages coming from a module such as one of the caching modules described below (e.g., price summary LVC, order book OBC, or generic GVC) is filtered based upon a set of entitlement data and interest data that is stored for each record in the cache. This entitlement and interest data defines a set of users (or applications) that are both entitled to receive the messages associated with the record and have expressed an interest in receiving them. This data can be loaded into memory from storage 634 during initialization of the board 400, or from Application Software 300 during normal operation of the board 400. An exemplary embodiment of a FAM configured to perform interest and entitlement filtering is described hereinafter with respect to Figure 16.

With financial instrument symbol mapping, a common identifying string for a financial instrument (typically referred to as the "symbol") is mapped into a direct record key number that can be used

by modules such as caching modules (LVC, OBC, GVC) to directly address the cache record associated with that financial instrument. The record key number may also be used by software to directly address a separate record corresponding to that instrument that is
5 kept in a storage, preferably separate from board 400. An exemplary embodiment of a FAM configured to perform symbol mapping is described hereinafter with respect to Figures 12-14.

With record ID mapping, a generic identifying string for a record is mapped into a direct record key number that can be used by
10 a caching module (e.g., LVC, OBC, GVC) or software to directly address the record in a storage medium.

The price summary Last Value Cache update/retrieve/normalize (LVC) operation operates to maintain a cache of financial instrument records whose fields are updated in real-time with information
15 contained in streaming messages received from a message parsing module, and to enhance or filter the messages received from a message parsing module before passing them on to subsequent processing modules. The type of update performed for an individual field in a record will be defined by a programmable data dictionary
20 as described above, and may consist of moving the data field from the message to the record, updating the record field by accumulating the data field over a series of messages defined within a time-bounded window, updating the record field only if certain conditions as defined by a set of programmable rules are true, or computing a
25 new value based upon message and/or record field values as guided by a programmable formula. The type of enhancement or filtering applied to an individual message may consist of replacing a message field with one created by accumulating the data over a series of messages defined within a time-bounded window, flagging a field whose value
30 falls outside of a programmatically defined range of values, or suppressing the message in its entirety if the value of a field or set of fields fails to change with respect to the corresponding values contained within the cache record. An exemplary embodiment of a FAM configured to perform LVC updating is described hereinafter
35 with respect to Figures 15(a) and (b).

The order book cache update, retrieve and normalize (OBC) operation operates to maintain a cache of financial instrument

records where each record consists of an array of sub-records that define individual price or order entries for that financial instrument. A sort order is maintained for the sub-records by the price associated with each sub-record. The fields of the sub-records are updated in real-time with information contained in streaming messages received from a message parsing module. Sub-records associated with a record are created and removed in real-time according to information extracted from the message stream, and the sort order of sub-records associated with a given record is continuously maintained in real-time. The type of update performed for an individual field in a sub-record will be defined by a programmable data dictionary, and may consist of moving the data field from the message to the sub-record, updating the sub-record field by accumulating the data field over a series of messages defined within a time-bounded window, updating the sub-record field only if certain conditions as defined by a set of programmable rules are true, or computing a new value based upon message and/or record or sub-record fields as guided by a programmable formula. The OBC includes the ability to generate various views of the book for a financial instrument including but not limited to a price-aggregated view and a composite view. A composite view is a sort order of the price or order entries for a financial instrument across multiple exchanges. The OBC also includes the ability to synthesize a top-of-book quote stream. When an update operation causes the best bid or offer entry in a given record to change, the OBC may be configured to generate a top-of-book quote reporting the current best bid and offer information for the financial instrument. A synthesized top-of-book quote stream has the ability to report best bid and offer information with less latency than an exchange-generated quote stream. This may be used to accelerate a variety of latency sensitive applications.

The Generic Last Value Cache (GVC) operation operates to maintain a cache of records whose fields are updated in real-time with information contained in streaming messages received from a message parsing module. The structure of a record and the fields contained within it are defined by a programmable data dictionary, as described above. The type of update performed for an individual

field in a record will be defined by a programmable data dictionary, and may consist of moving the data field from the message to the record, updating the record field by accumulating the data field over a series of messages defined within a time-bounded window, updating the record field only if certain conditions as defined by a set of programmable rules are true, or computing a new value based upon message and/or record field values as guided by a programmable formula.

10 A minute bar generation operation operates to monitor real-time messages from a message parsing module or last value cache module for trade events containing trade price information, or for quote events containing quote price information, and create "minute bar" events that summarize the range of trade and/or quote prices that have occurred over the previous time interval. The time interval is a programmable parameter, as is the list of records for which minute bars should be generated, and the fields to include in the generated events.

20 A Top 10 list generation operation operates to monitor real-time messages from a message parsing module or last value cache module for trade events containing price information and create lists of instruments that indicate overall activity in the market. Such lists may include (where 'N' is programmatically defined): top N stocks with the highest traded volume on the day; top N stocks with the greatest positive price change on the day; top N stocks with the largest percentage price change on the day; top N stocks with the greatest negative price change on the day; top N stocks with the greatest number of trade events recorded on the day; top N stocks with the greatest number of "large block" trades on the day, where the threshold that indicates whether a trade is a large block trade is defined programmatically.

30 A programmatic field generation (via LVC, OBV, GVC, etc.) operation operates to augment messages received from a message parsing module with additional fields whose values are defined by a mathematical formula that is supplied programmatically. The formula may reference any field within the stream of messages received from a message parsing module, any field contained within a scratchpad

memory associated with this module, or any field contained within any record held within any the record caches described herein.

A programmatic record generation (with LVC, OBC, GVC, etc.) operation operates to generate records that represent synthetic
5 financial instruments or other arbitrary entities, and a series of event messages that signal a change in state of each record when the record is updated. The structure of the records and the event messages are programmatically defined by a data dictionary. The field values contained with the record and the event messages are
10 defined by mathematical formulas that are supplied programmatically. The formulas may reference any field within the stream of messages received from a message parsing module, any field contained within a scratchpad memory associated with this module, or any field contained within any record held within any the record caches
15 described herein. Updates to field values may be generated upon receipt of a message received from another module, or on a time interval basis where the interval is defined programmatically. A basket calculation engine is one example of programmatic record generation. A synthetic instrument may be defined to represent a
20 given portfolio of financial instruments, constituent instruments in an Exchange Traded Fund (ETF), or market index. The record for that synthetic instrument may include fields such as the Net Asset Value (NAV) and total change.

A historic record search and filter operation operates to
25 filter messages received from a message parsing module that represent a time series of events to partition the events into various sets, where each set is defined by a collection of criteria applied to event attributes. The event message structure, criteria and attributes are all programmatically defined. Event attributes
30 include, but are not limited to: financial instrument symbol, class of symbol, time and date of event, type of event, or various indicator fields contained within the event. Multiple events within a set may be aggregated into a single event record according to a collection of aggregation rules that are programmatically defined
35 and applied to attributes of the individual events. Aggregation rules may include, but are not limited to, aggregating hourly events into a single daily event, aggregating daily events into a single

weekly event, or aggregating multiple financial instruments into a single composite instrument.

These functions (as well as other suitable financial data processing operations) as embodied in FAMs can then be combined to form FAM pipelines that are configured to produce useful data for a market data platform. For example, a feed compressor FAM pipeline can employ FAMs configured with the following functions: fixed record format message parsing, fixed record format message generation, FIX message parsing, FIX message generation, FIX/FAST message parsing, FIX/FAST message generation, message compression, and message decompression.

Figure 7 illustrates an exemplary FAM pipeline for performing a FIX-to-FAST message transformation. FAM 702 is configured to receive an incoming stream of FIX messages and perform FIX message parsing thereon, as described above. Then the parsed message is passed to FAM 704, which is configured to field decode and validate the parsed message. To aid this process, FAM 704 preferably has access to stored templates and field maps in memory 710 (embodied by memory 404 on board 400). The templates identify what fields exist in a given message type, while the field maps uniquely identify specific fields in those message (by location or otherwise). Next, FAM 704 provides its output to FAM 706, which is configured to perform a FAST field encode on the received message components. Memory 710 and any operator values stored in memory 712 aid this process (memory 712 also being embodied by memory 404 on board 400). The operator values in memory 712 contains various state values that are preserved from one message to the next, as defined by the FAST encoding standard. Then, the encoded FAST messages are serialized by FAM 708 to form a FAST message stream and thereby complete the FIX to FAST transformation.

Figure 8 illustrates an exemplary FAM pipeline for performing a FAST-to-FIX message transformation. An incoming FAST stream is received by FAM 802, which deserializes the stream of FAST messages. The deserialized FAST messages are then provided to FAM 804, which operates to decode the various fields of the FAST messages as aided by the templates and field maps in memory 812 and the operator values in memory 810. Thereafter, FAM 806 is preferably configured

to perform message-query filtering. Message query filters allow for certain messages to be excluded from the message flow. Such filters are preferably parameterized in FAM 806 such that filtering criteria based on the field values contained within each message can be flexibly defined and loaded onto the FPGA. Examples of filtering criteria that can be used to filter messages include a particular type of instrument (e.g., common stock, warrant, bond, option, commodity, future, etc.), membership within a prescribed set of financial instruments (e.g., an index or "exchange traded fund" (ETF)), message type, etc. Next, FAM 808 operates to perform FIX message generation by appropriately encoding the various message fields, as aided by the templates and field maps in memory 812. Thus, the FAM pipeline shown in Figure 8 operates to transform FAST message to FIX messages. Memory units 810 and 812 are preferably embodied by memory 404 of board 400.

Figure 9 depicts an exemplary FAM pipeline for carrying out a variety of data processing tasks. The FAM pipeline of Figure 9 takes in a FAST message stream. FAMS 902 and 904 operate in the same manner as FAMS 802 and 804 in Figure 8. Thus, the output of FAM 904 comprises the data content of the FAST message decomposed into its constituent fields. This content is then passed to a variety of parallel FAMS 906, 908, and 910. FAM 906 performs an administrative record filter on the data it receives. The administrative record filter preferably operates to pass through message types that are not processed by any of the other FAM modules of the pipeline. FAM 908 serves as a Top 10 lists engine, as described above. FAM 910 serves as a message query filter, as described above.

The output of FAM 910 is then passed to FAM 912, which is configured as a rule-based calculation engine, as described above. FAM 912 also receives data from a real time field value cache 926 to obtain LVC data, as does the top 10 list FAM 908. Cache 926 is preferably embodied by memory 404 of board 400. The output from the rule-based calculation engine FAM 912 is then passed to parallel FAMS 914, 916, and 918. FAM 914 serves as a message multiplexer, and receives messages from the outputs of FAMS 906, 908 and 912. FAM 920 receives the messages multiplexed by FAM 914, and serves to

encode those messages to a desired format. FAM 916 serves as an alert engine, whose function is explained above, and whose output exits the pipeline. FAM 918 serves as a value cache update engine to ensuring that cache 926 stays current.

5 Figure 10 depicts another exemplary FAM pipeline for carrying out multiple data processing tasks. FAM 1002 takes in a stream of fixed format messages and parses those messages into their constituent data fields. The output of FAM 1002 can be provided to FAM 1004 and FAM 1018. FAM 1018 serves as a message synchronization
10 buffer. Thus, as the fields of the original parsed message are passed directly from FAM 1002 to FAM 1018, FAM 1018 will buffer those data fields while the upper path of Figure 10 (defined by FAMs 1004, 1006, 1008, 1010, 1012, and 1014) process select fields of the parsed message. Thus, upon completion of the processing performed
15 by the FAMs of the upper path, the message formatting FAM 1016, can generate a new message for output from the pipeline using the fields as processed by the upper path for that parsed message as well as the fields buffered in FAM 1018. The message formatter 1016 can then append the fields processed by the upper path FAMs to the
20 fields buffered in FAM 1018 for that message, replace select fields buffered in FAM 1018 for that message with fields processed by the upper path FAMs, or some combination of this appending and replacing.

 FAM 1004 operates to map the known symbol for a financial
25 instrument (or set of financial instruments) as defined in the parsed message to a symbology that is internal to the platform (e.g., mapping the symbol for IBM stock to an internal symbol "12345"). FAM 1006 receives the output from FAM 1004 and serves to update the LVC cache via memory 1024. The output of FAM 1006 is
30 then provided in parallel to FAMs 1008, 1010, 1012, and 1014.

 FAM 1008 operates as a Top 10 list generator, as described above. FAM 1010 operates as a Minute Bar generator, as described above. FAM 1012 operates as an interest/entitlement filter, as described above, and FAM 1014 operates as a programmatic calculation
35 engine, as described above. The outputs from FAMs 1008, 1010, 1012 and 1014 are then provided to a message formatter FAM 1016, which operates as described above to construct a fixed format message of a

desired format from the outputs of FAMs 1008, 1010, 1012, 1014 and 1018.

In performing these tasks, FAM 1004 is aided by memory 1020 that stores templates and field maps, as well as memory 1022 that stores a symbol index. FAM 1006 is also aided by memory 1020 as well as memory 1024 which serves as an LVC cache. Memory 1020 is also accessed by FAM 1008, while memory 1024 is also accessed by FAM 1014. FAM 1012 accesses interest entitlement memory 1026, as loaded from storage 634 or provided by Application Software 300 during initialization of the board 400.

Figure 11 depicts an exemplary FAM pipeline for performing the functions of an exemplary ticker plant embodiment, including message parsing, symbol mapping, Last Value Cache (LVC) updates, and interest and entitlement filtering.

Message Parser FAM 1102 ingests a stream of messages, parses each message into its constituent fields, and propagates the fields to downstream FAMs. Message fields required for processing in FAMs 1104, 1106, and 1108 are passed to FAM 1104. Other message fields are passed to Message Synchronization Buffer FAM 1112. Message Parser FAM 1102 may be implemented to support a variety of message formats, including various types of fixed-formats and self-describing formats. A preferable message format provides sufficient flexibility to support the range of possible input events from financial exchanges. In a preferred implementation, the Message Parser FAM 1102 may be configured to support different message formats without altering the firmware. This may be achieved by loading message format templates into Template & Field Map buffer 1120. Message Parser FAM 1102 reads the message format description from buffer 1120 prior to processing input messages to learn how a given message is to be parsed.

Like FAM 1004 in Figure 10, Symbol ID Mapping FAM 1104 operates to map the known symbol for a financial instrument (or set of financial instruments) as defined in the parsed message to a symbology that is internal to the platform (e.g., mapping the symbol for IBM stock to an internal symbol "12345"). Preferably, the internal platform symbol identifier (ID) is an integer in the range 0 to N-1, where N is the number of entries in Symbol Index Memory

1122. Preferably, the symbol ID is formatted as a binary value of size $M = \log_2(N)$ bits. The format of financial instrument symbols in input exchange messages varies for different message feeds and financial instrument types. Typically, the symbol is a variable-length ASCII character string. A symbology ID is an internal control field that uniquely identifies the format of the symbol string in the message. As shown in Figure 12, a symbology ID is preferably assigned by the feed handler, as the symbol string format is typically shared by all messages on a given input feed.

10 A preferred embodiment of the Symbol ID Mapping FAM maps each unique symbol character string to a unique binary number of size M bits. In the preferred embodiment, the symbol mapping FAM performs a format-specific compression of the symbol to generate a hash key of size K bits, where K is the size of the entries in the Symbol Index Memory 1122. The symbology ID may be used to lookup a Key Code that identifies the symbol compression technique that should be used for the input symbol. Preferably, the symbol mapping FAM compresses the symbol using format-specific compression engines and selects the correct compressed symbol output using the key code.

15 20 Preferably, the key code is concatenated with the compressed symbol to form the hash key. In doing so, each compression technique is allocated a subset of the range of possible hash keys. This ensures that hash keys will be unique, regardless of the compression technique used to compress the symbol. An example is shown in Figure 12 wherein the ASCII symbol for a financial instrument is compressed in parallel by a plurality of different compression operations (e.g., alpha-numeric ticker compression, ISIN compression, and commodity compression). Compression techniques for different symbologies can be selected and/or devised on an ad hoc basis as desired by a practitioner of the invention. A practitioner of the present invention is free to select a different compression operation as may be appropriate for a given symbology. Based on the value of the key code, the symbol mapping FAM will pass one of the concatenations of the key code and compression results as the output from the multiplexer for use as the hash key.

35 Alternatively, the format-specific compression engines may be implemented in a programmable processor. The key code may then be

used to fetch a sequence of instructions that specify how the symbol should be compressed.

Once the hash key is generated, the symbol mapping FAM maps the hash key to a unique address in the Symbol Index Memory in the range 0 to N-1. The Symbol Index Memory may be implemented in a memory "on-chip" (within the reconfigurable logic device) or in "off-chip" high speed memory devices such as SRAM and SDRAM that are accessible to the reconfigurable logic device. Preferably, this mapping is performed by a hash function. A hash function attempts to minimize the number of probes, or table lookups, to find the input hash key. In many applications, additional meta-data is associated with the hash key. In the preferred embodiment, the location of the hash key in the Symbol Index Memory is used as the unique internal Symbol ID for the financial instrument.

Figure 13 shows a preferred embodiment of a hash function to perform this mapping that represents a novel combination of known hashing methods. The hash function of Figure 13 uses near-perfect hashing to compute a primary hash function, then uses open-addressing to resolve collisions. The hash function $H(x)$ is described as follows:

$$H(x) = (h1(x) + (i * h2(x))) \bmod N$$

$$h1(x) = A(x) \oplus d(x)$$

$$d(x) = T(B(x))$$

$$h2(x) = C(x)$$

The operand x is the hash key generated by the previously described compression stage. The function $h1(x)$ is the primary hash function. The value i is the iteration count. The iteration count i is initialized to zero and incremented for each hash probe that results in a collision. For the first hash probe, hash function $H(x) = h1(x)$, thus the primary hash function determines the first hash probe. The preferred hash function disclosed herein attempts to maximize the probability that the hash key is located on the first hash probe. If the hash probe results in a collision, the hash key stored in the hash slot does not match hash key x , the iteration count is incremented and combined with the secondary hash function $h2(x)$ to generate an offset from the first hash probe location. The modulo N operation ensures that the final result is within the range

0 to N-1, where N is the size of the Symbol Index Memory. The secondary hash function $h2(x)$ is designed so that its outputs are prime relative to N. The process of incrementing i and recomputing H(x) continues until the input hash key is located in the table or
5 an empty table slot is encountered. This technique of resolving collisions is known as open-addressing.

The primary hash function, $h1(x)$, is computed as follows. Compute hash function B(x) where the result is in the range 0 to Q-1. Use the result of the B(x) function to lookup a displacement
10 vector d(x) in table T containing Q displacement vectors. Preferably the size of the displacement vector d(x) in bits is equal to M. Compute hash function A(x) where the result is M bits in size. Compute the bitwise exclusive OR, $\oplus$, of A(x) and d(x). This
15 is one example of near-perfect hashing where the displacement vector is used to resolve collisions among the set of hash keys that are known prior to the beginning of the query stream. Typically, this fits well with streaming financial data where the majority of the symbols for the instruments trading in a given day is known.
20 Methods for computing displacement table entries are known in the art.

The secondary hash function, $h2(x)$, is computed by computing a single hash function C(x) where the result is always prime relative to N. Hash functions A(x), B(x), and C(x) may be selected from the
25 body of known hash functions with favorable randomization properties. Preferably, hash functions A(x), B(x), and C(x) are efficiently implemented in hardware. The set of H3 hash functions are good candidates. (See Krishnamurthy et al., "Biosequence Similarity Search on the Mercury System", Proc. of the IEEE 15th
30 Int'l Conf. on Application-Specific Systems, Architectures and Processors, September 2004, pp. 365-375).

Once the hash function H(x) produces an address whose entry is equal to the input hash key, the address is passed on as the new Symbol ID to be used internally by the ticker plant to reference the
35 financial instrument. As shown in Figure 13, the result of the hash key compare function may be used as a valid signal for the symbol ID output.

Hash keys are inserted in the table when an exchange message contains a symbol that was unknown at system initialization. Hash keys are removed from the table when a financial instrument is no longer traded. Alternatively, the symbol for the financial instrument may be removed from the set of known symbols and the hash table may be cleared, recomputed, and initialized. By doing so, the displacement table used for the near-perfect hash function of the primary hash may be optimized. Typically, financial markets have established trading hours that allow for after-hours or overnight processing. The general procedures for inserting and deleting hash keys from a hash table where open-addressing is used to resolve collisions is well-known in the art.

In a preferred embodiment, the symbol mapping FAM also computes a global exchange identifier (GEID) that maps the exchange code and country code fields in the exchange message to an integer in the range 0 to G-1, as shown in Figure 14. Similar to the symbol field for financial instruments, the exchange code and country code fields uniquely identify the source of the exchange message. The value of G should be selected such that it is larger than the total number of sources (financial exchanges) that will be generating input messages for a given instance of the system. Hashing could be used to map the country codes and exchange codes to the GEID. Alternatively, a "direct addressing" approach can be used to map country and exchange codes to GEIDs. For example, the exchange code and country codes can each be represented by two character codes, where the characters are 8-bit upper-case ASCII alpha characters. These codes can then be truncated to 5-bit characters in embodiment where only 26 unique values of these codes are needed. For each code, these truncated values are concatenated to generate a 10-bit address that is used to lookup a compressed intermediate value in a stage 1 table. Then the compressed intermediate values for the exchange and country code can be concatenated to generate an address for a stage 2 lookup. The result of the stage 2 lookup is the GEID. The size of the intermediate values and the stage 2 address will depend on the number of unique countries and the max number of exchanges in any one country, which can be adjusted as new exchanges open in different countries.

Symbol mapping FAM 1106 passes input message field values, the symbol ID, and global exchange ID to Last Value Cache (LVC) Update FAM 1106. LVC Update FAM serves to update the LVC cache via memory 1124, as well as message fields that may depend on record field values. One example is the tick direction which indicates if the price in the message is larger or smaller than the previous price captured in the record.

As shown in Figures 15(a) and (b), the LVC memory manager retrieves one or more records associated with the financial instrument. The LVC memory manager passes the record and message fields to the LVC message/record updater. The LVC message/record updater contains a set of update engines that update the record and message fields according to specified business logic. The business logic for field updates may vary according to a number of parameters including event type (trade, quote, cancel, etc.), financial instrument type (security, option, commodity, etc.), and record type. In a preferred embodiment, the update engines are directed by business logic templates contained in Templates & Field Maps 1120. Techniques for template-driven update engines are well-known in the art.

Record fields may include but are not limited to: last trade price, last trade size, last trade time, best bid price, best bid size, best bid time, best ask price, best ask size, best ask time, total trade volume, daily change, tick direction, price direction, high trade price, high price time, low trade price, low price time, and close price. In a preferred embodiment, record fields also include derived fields such as: total trade volume at bid, total trade volume at ask, traded value, traded value at bid, traded value at ask, and volume-weighted average price (VWAP).

As reflected in Figures 15(a) and (b), a preferred embodiment of the LVC Update FAM maintains a composite record and a set of regional records for every financial instrument observed on the input feeds. A composite record reflects the current state of the financial instrument across all exchanges upon which it trades. A regional record reflects the current state of the financial instrument on a given exchange. For example, if stock ABC trades on exchanges AA, BB, and CC, then four records will be maintained by

the LVC Update FAM, one composite record and three regional records,
If an input message reports a trade of stock ABC on exchange BB,
then the LVC Update FAM updates the composite record for ABC, the
regional record for ABC on exchange BB, and the message fields
5 according to the business logic for stock trade events on exchange
BB.

As shown in Figure 15(a), the LVC Memory Manger uses the
symbol ID and global exchange ID to retrieve the composite and
regional record. In a preferred embodiment, the symbol ID is used
10 to retrieve an entry in the record management memory. The entry
contains a valid flag, a composite record pointer, and a regional
list pointer. The valid flag indicates whether the symbol ID is
known, record(s) have been allocated, and the pointers in the entry
are valid.

15 If the valid flag is set, the LVC Memory Manager uses the
composite record pointer to retrieve the composite record from the
record storage memory. The composite record is passed to the LVC
message/record updater where it is stored in a composite record
buffer for processing by the update engines. The LVC Memory Manger
20 uses the regional list pointer to retrieve a regional list from the
record storage memory. Note that regional list blocks may also be
stored in the record management memory or in another independent
memory. The regional list block contains pointers to the regional
records for the financial instrument identified by the symbol ID.
25 Since each regional record reflects the state of the instrument on a
given exchange, a global exchange ID is stored with each regional
pointer. The pointer to the regional record associated with the
exchange specified in the message is located by matching the global
exchange ID computed by the Symbol ID Mapping FAM. The LVC Memory
30 Manger uses the regional pointer associated with the matching global
exchange ID to retrieve the regional record from the record storage
memory. The regional record is passed to the LVC message/record
updater where it is stored in a regional record buffer for
processing by the update engines.

35 If the valid flag in the record management memory entry is not
set, then the LVC Memory Manager creates a new composite record, a
new regional list block, and a new regional record for the financial

instrument. The initial values for record fields may be drawn from Templates and Field Maps 1120. The regional list block will be initialized with at least one entry that contains a pointer to the new regional record and the global exchange ID received from the Symbol ID Mapping FAM. The LVC Memory Manger uses a free space pointer to allocate available memory in the record storage memory. After the memory is allocated, the free space pointer is updated. Freeing unused memory space, defragmenting memory, and adjusting the free space pointer may be performed by the LVC Memory Manager or by control software during market down times. Techniques for freeing memory space and defragmenting are well-known in the art. Once the records are initialized in record storage memory, the LVC Memory Manger writes the pointers into the management memory entry and sets the valid flag.

The LVC Memory Manager may also encounter a case where the valid flag in the memory management entry is set, but a matching global exchange ID is not found in the regional list. This will occur when a known financial instrument begins trading on a new exchange. In this case, the LVC Memory Manager allocates a new regional record and creates a new entry in the regional list block.

Once the record and message fields are loaded into their respective buffers, the update engines perform the field update tasks as specified by the business logic. Upon completion of their update tasks, the update engines signal the LVC Memory Manager. When all processor engines complete, the LVC Memory Manager writes updated records back to record storage memory. Processing can be deployed across the plurality of update engines in any of a number of ways. In one embodiment, a given record and its related message fields are passed through a sequence of update engines arranged in a pipeline. In another embodiment, each record and its related message fields are passed directly to an update engine that is configured to perform processing appropriate for the type of processing that the record and message fields needs. Preferably, the LVC updater is configured to balance the distribution of records and message fields across the plurality of different update engines so that a high throughput is maintained. In an exemplary embodiment, each update engine is configured to be responsible for

updating a subset of the record fields (either regional or composite), with multiple engines operating in parallel with each other.

The LVC message/record updater passes updated message fields
5 and interest lists to the Interest Entitlement Filter FAM 1108. An
interest list contains a set of unique identifiers for
users/applications that registered interest in receiving updates for
the financial instrument. In a preferred embodiment, the set of
user identifiers is specified using a bit vector where each bit
10 position in the vector corresponds to a user identifier. For
example, a 4-bit vector with the value 1010 represents the set of
user identifiers {3,1}. The size of the interest list in bits is
equal to the total number of user subscriptions allowed by the
Ticker Plant. In a preferred embodiment, each record contains an
15 interest list that is updated in response to user subscribe and
unsubscribe events. By maintaining an interest list in the
composite record, the Ticker Plant allows a subscription to include
all transactions for a given financial instrument on every exchange
upon which it trades. Preferably, each interest list for a given
20 record is stored with that record in the record storage memory.
Control software for the ticker plant, which maintains the set of
interest lists for each record in a control memory can be configured
to advise the LVC FAM of a new interest list vector for a given
record so that the record storage memory can be updated as
25 appropriate. Other types of subscriptions, such as exchange-based
subscriptions, may also be enabled by the FAM pipeline.

In a preferred embodiment, the record storage memory and/or
the record management memory is an external memory to the
reconfigurable logic, such as a Synchronous Random Access Memory
30 (SRAM) or Synchronous Dynamic Random Access Memory (SDRAM) device.
Read and write transactions to external memory devices incur
processing delays. A common technique for improving processing
performance is to mask these delays by performing multiple
transactions in a pipelined fashion. The LVC Memory Manger is
35 designed as a pipelined circuit capable of performing the various
processing steps in parallel, therefore allowing it to mask memory
latencies and process multiple messages in parallel. Doing so

enables the LVC Memory Manager to process more messages per unit time, i.e. achieve higher message throughput. By employing a functional pipeline, the LVC Memory Manager preferably recognizes occurrences of the same symbol ID within the pipeline and ensure
5 correctness of records in update engine buffers. One method for doing so is to stall the pipeline until the updated records associated with the previous occurrence of the symbol ID are written back to record storage memory. In a preferred embodiment, the LVC Memory Manager utilizes a caching mechanism to always feed the
10 correct record field values to the update engine buffers. Techniques for a memory cache are well-known in the art. The caching mechanism can be embodied as a memory, preferably a high-speed memory, located either on-chip (within the reconfigurable logic device) or off-chip (e.g., an SRAM or SDRAM device accessible
15 to the reconfigurable logic device). However, it should also be noted that the cache can also be embodied by a full memory hierarchy with a multi-level cache, system memory, and magnetic storage. A record typically stays in the cache memory for the duration of a trading day. Such recently updated records can then be flushed out
20 of the cache during an overnight processing ("roll") and archived. However, it should be noted that the cache can be configured to maintain records so long as space is available in the cache for storing new records, in which case a FIFO scheme can be used to maintain the cache.

25 Figure 15(b) presents an exemplary functional pipeline for masking memory and processing latencies. As discussed in relation to Figure 15(a), the address resolution block determines the address of the regional and composite records using the symbol ID and global exchange ID to locate the memory management entry and regional list
30 block. Note that each functional block in Figure 15(b) may also be implemented in a pipelined fashion. For example, one sub-block in the address resolution block may issue a stream of read commands to the record management memory to retrieve record management entries. Another sub-block may process the stream of returned entries and
35 issue a stream of read commands to the record storage memory to retrieve regional list blocks. Another sub-block may process the stream of returned list blocks to resolve regional record addresses.

The last sub-block of the address resolution block passes the regional and composite record addresses to the cache determination block. By tracking the sequence of record addresses, this block determines the physical location of the "current" (most up-to-date) records: in the record cache or in the record storage memory. The cache determination block passes the physical record pointers to the record retrieval block. The record retrieval block fetches the records from the specified locations, loads them into the processing buffers for updating, then signals the processors to begin processing. Note that if the record cache and processing buffers are in the same memory space, the record retrieval block may essentially queue up several sets of records for the processors, allowing the processors to complete a processing task and immediately move on to the next set of records, even if the same set of records must be updated sequentially. Once the processors complete a set of records, they signal the record updating block to write the updated records back to record storage memory. Depending on the implementation of the record cache, copies of the updated records are written to cache or simply persist in the processing buffers for a period of time. In parallel, updated message fields are passed to the downstream FAM.

The LVC Update FAM 1106 passes interest lists, the global exchange ID, and updated message fields to the Interest Entitlement FAM 1108. The Interest Entitlement FAM computes a single interest list that is used to distribute the output message to the set of users/applications that have registered interest and are entitled to receive the message. As previously described, interest may be registered by subscribing to updates for the regional or composite interest, as well as subscribing to all updates from a given exchange. Access to real-time financial data is typically a purchased service where the price may vary depending on the scope of data access. In a preferred embodiment, a ticker plant is capable of accepting subscription requests from users/applications with varying levels of data access privileges.

As shown in Figure 16, a preferred embodiment of the Interest Entitlement FAM operates on interest lists specified as bit vectors. The global exchange ID is used to lookup the exchange interest list

in the exchange interest table. By using an exchange interest list, the embodiment of Figure 16 allows traders to request notification of everything traded on a given exchange without individually subscribing to every instrument traded on that given exchange. Note that a global interest list, specifying users interested in all events, may also be stored in a register in the FAM. As shown in Figure 16, all interest list vectors applicable to a message are combined using a bitwise OR operation. The resulting composite interest list vector contains the set of users that are interested in receiving the output message. An entitlement ID is used to lookup an entitlement mask. The entitlement ID may be specified by the feed handler that receives messages from the exchange, or alternative mechanisms such as a lookup based on message fields such as the global exchange ID, event type, and instrument type. Similar to the interest lists, the entitlement mask specifies which users/applications are entitled to receive the output message. The entitlement mask is applied to the combined interest list using a bitwise AND operation. The result is the final entitled interest vector specifying the set of entitled and interested users/applications. If the entitled interest bit vector is empty (e.g., all zeros), the message can be dropped from the pipeline. This entitled interest list and the message fields received from the LVC Update FAM are passed to the Message Formatter FAM 1110.

As previously described, the Message Formatter FAM 1110 serves to construct an output message from updated fields received from the Interest Entitlement Filter FAM and fields contained in the Message Synch Buffer FAM 1112. In a preferred embodiment, the format of the output message is specified by the Templates and Field Maps 1120. In a preferred embodiment, the output message includes the entitled interest list computed by the Interest Entitlement Filter. A subsequent functional block in the Ticker Plant processes the interest list and transmits copies of the output message to the interested and entitled users/applications.

Figure 17 depicts an exemplary embodiment of a ticker plant highlighting the interaction and data flow amongst the major subcomponents that enable this embodiment of a ticker plant to maintain high performance and low latency while processing financial

data. Figures 18, 19, 20, 21, and 22 depict detailed views of the data flow and the component interaction presented in Figure 16. These figures provide additional information for inbound exchange data processing, data normalization, interaction with reconfigurable logic, data routing, and client interaction respectively.

Financial market data generated by exchanges is increasing at an exponential rate. Individual market events (trades, quotes, etc) are typically bundled together in groups and delivered via an exchange feed. These exchange feeds are overwhelmingly delivered to subscribers using the Internet Protocol over an Ethernet network. Due to constraints on packet size dictated by the network environment, data groups transmitted by the exchange tend to be limited to sizes less than 1500 bytes.

As market data rates increase, the number of data groups that must be processed by a ticker plant increases. In typical ticker plant environments, each network packet received by the ticker plant must be processed by the network protocol stack contained within the Operating System and delivered to a user buffer. This processing includes one or more data copies and an Operating System transition from "kernel" or "supervisor" mode to user for each exchange data packet. An increase in data rates in turn increases the processing burden on the ticker plant system to deliver individual exchange data messages to the user level process.

The device depicted in Figure 17 uses a novel approach to efficiently deliver the exchange data to the user process. Figure 18 shows the exemplary data flow for inbound exchange traffic in a ticker plant. Exchange data enters the ticker plant at 1801 and is processed by the Operating System supplied network protocol stack. Typical ticker plants use a user-mode interface into the network protocol stack at 1802. This method of connecting to the protocol stack incurs processing overhead relating to buffer copies, buffer validation, memory descriptor table modifications, and kernel to user mode transitions for every network data packet received by the ticker plant. As shown in Figure 18, an Upject Driver is employed that interfaces with the operating system supplied network protocol stack at the kernel level at 1803. Individual data packets are processed at the kernel level and copied directly into a ring buffer

at 1804, thus avoiding subsequent data copies and kernel to user mode transitions incurred when accessing the protocol stack via the user mode interface.

5 The ring buffers employed by the Upject Driver are shared memory ring buffers that are mapped into both kernel and user address spaces supported by the Operating System at 1805. The boundary between kernel mode operations and user mode operations is shown at 1806. Data written to the kernel address space of one of these ring buffers is instantly accessible to the user mode code
10 because both the user mode and kernel mode virtual addresses refer to the same physical memory. Utilizing the shared ring buffer concepts, the preferred embodiment of a Ticker Plant does not have to perform user to kernel mode transitions for each network data packet received and thus achieves a performance boost.
15 Additionally, the Upject Driver can utilize the shared ring buffer library to directly transfer inbound data to other kernel processes, device drivers, or user processes at 1807. This versatile shared ring buffer interconnect enables fast-track routing of network traffic directly to Reconfigurable logic via the Hardware Interface
20 Driver.

General purpose computers as known in the art employ "multi-core" or "multi-processor" technology to increase the available compute resources in a computer system. Such multi-core systems allow the simultaneous execution of two or more instruction streams,
25 commonly referred to as "threads of execution". To fully utilize the compute power of these multiple processor systems, software must be designed to intelligently manage thread usage, resource contention and interdependencies between processing threads. The data normalization component of the preferred embodiment of a Ticker
30 Plant employs thread groups to efficiently normalize raw exchange data.

Thread groups improve processing efficiency of the preferred embodiment of a Ticker Plant by using the following techniques:

1. Limit the execution of the Operating System scheduling
35 mechanism by matching the number of worker threads to the number of available instruction processors.

2. Enable multiple exchange feeds to be processed by a single thread group.
3. Eliminate resource contention between different thread groups.
4. Remove synchronization points within the compute path of each thread group, further increasing processing efficiency.
5. Perform message normalization in a single compute path, including line arbitration, gap detection, retransmission processing, event parsing, and normalized event generation.
6. Associate exchange feeds with individual thread groups using a configuration file.

Figure 19 depicts the processing of several thread groups. Each thread group contains a single thread of execution at 1901. All operations performed by a thread group are executed on the single processing thread associated with the thread group. A separate input ring buffer is associated with each thread group at 1902. Inbound exchange data is deposited into the appropriate ring buffer. The processing thread for the thread group detects the presence of new data in the ring buffer and initiates normalization processing on the data one event at a time at 1903. Normalization processing involves parsing inbound messages, arbitrating amongst messages received on multiple lines, detecting sequence gaps, initiating retransmission requests for missed messages, eliminating duplicate events, and generating normalized exchanged events. This processing results in the creation of normalized events that are deposited into an output ring buffer at 1904.

All of the processing for any single thread group is completely independent of the processing for any other thread group. No data locking or resource management is required during the normalization process which eliminates the possibility of thread blocking due to contention for a shared resource. The preferred embodiment of a Ticker Plant supports a variable number of thread groups at 1905. The number of thread groups and the number of exchange feeds processed by each thread group are configurable, enabling the Ticker Plant to efficiently utilize additional compute resources as they become available in future generations of computer systems. The association of inbound data feeds with individual

thread groups is defined in a configuration file that is read during initialization processing.

The Hardware Interface Driver in the preferred embodiment of a Ticker Plant is optimized to facilitate the efficient movement of large amounts of data between system memory and the reconfigurable logic. Figure 20 shows the data movement between the Hardware Interface Driver and the reconfigurable logic. User mode application data destined for the reconfigurable logic is written to one of the shared memory ring buffers at 2001. These are the same buffers shown at 1904 in Figure 19. These ring buffers are mapped into both kernel address space and into user space. Data written to these buffers is immediately available for use by the Hardware Interface Driver at 2002. The boundary between user space and kernel space is noted at 2003.

The Hardware Interface Driver is responsible for updating descriptor tables which facilitates the direct memory access (DMA) data transfers to the reconfigurable logic. Normalized market data events are transferred to the reconfigurable logic at 2004. The reconfigurable logic and Firmware Application Module Chain perform the operational functions as noted above. Processed market events are transferred back to the Hardware Interface Driver at 2005 and deposited into a ring buffer at 2006.

A novel feature of the preferred embodiment of a Ticker Plant is the ability to route data to consumers through a "fast track" by bypassing the time consuming data copies and Operating System mode switches. An operating system mode switch occurs whenever software transitions between user mode processing and kernel mode processing. Mode switches are expensive operations which can include one or more of the following operations: software interrupt processing, address validation, memory locking and unlocking, page table modifications, data copies, and process scheduling. Figure 21 depicts an exemplary design of a low latency data routing module. After processing by the reconfigurable logic, market data events are delivered to the Hardware Interface Driver at 2101. Each market data event as shown at 2102 contains a routing vector at 2103. This routing vector, which is preferably embodied by the entitled interest bit vector, is populated by the reconfigurable logic (preferably the interest and

entitlement FAM) and contains the information necessary to deliver each event to the appropriate consumers. A table maintained by the software preferably translates the bit positions of the entitled interest bit vector to the actual entities entitled to the subject
5 data and who have expressed interest in being notified of that data.

The Hardware Interface Driver calls into the MDC Driver for each event received from the reconfigurable logic at 2104. The MDC Driver is responsible for the fast track data routing of individual enhanced market data events. The routing information associated
10 with each event is interrogated at 2105. This interrogation determines the set of destination points for each event. Each event can be routed to one or more of the following: kernel modules, protocol stacks, device drivers, and/or user processes. Exception events, results from maintenance commands, and events that require
15 additional processing are routed via a slow path to the user mode background and maintenance processing module at 2106. The background and maintenance processing module has the ability to inject events directly into the Hardware Interface Driver at 2107 for delivery to the reconfigurable logic or to the MDC Driver at
20 2108 for delivery to a connected consumer.

Similar to the Upject Driver, the MDC Driver also maintains a kernel level interface into the Operating System supplied network protocol stack at 2109. This kernel level interface between the MDC Driver and the protocol stack provides a fast path for delivering
25 real-time market events to clients connects via a network at 2110. The event routing logic contained within the MDC Driver interrogates the event routing information contained in each event and passes the appropriate events directly to the network protocol stack.

The MDC driver also has the ability to route market events to
30 other consumers at 2111. These other consumers of real-time market events include, but are not limited to, network drivers for clients connected via a variety of network interconnect methodologies, kernel-mode modules or device drivers, hardware devices including reconfigurable logic, and different user mode processes. The MDC
35 Driver is a flexible data routing component that enables the preferred embodiment of a Ticker Plant to deliver data to clients with the lowest possible latency.

Figure 22 depicts an exemplary model for managing client connections. Remote clients connect to the over a network that is driven by an Operating System supplied network protocol stack at 2201. A request processing module interfaces with the Operating System supplied network protocol stack in user mode at 2202. The request processing module parses and validates all client requests. Client requests are then passed to the background and maintenance processing module at 2203. Clients typically make subscription requests that include the name of one or more securities instruments. A subscription request for a valid instrument results in a successful response sent back to the client via 2205 and a refresh image representing the current prices of the requested instrument that is initiated at 2204. Additional control information is sent to the Firmware Application Module Chain to enable the routing of all subsequent events on the specified instrument to the client. Additional requests can be made for an instantaneous data snapshot, historical data, and other data or services, including but not limited to options calculation, user defined composites, and basket calculations.

Depending on the nature of the client request, the background and maintenance processing module can either issue commands to the FAMS contained in reconfigurable logic via the Hardware Interface Driver at 2204, or it can respond directly to client request by sending properly formatted responses to the MDC driver at 2205. The MDC Driver uses spinlocks to synchronize responses to client requests with real-time market events at 2206. Responses to client requests and real-time market events are processed in the same manner by the MDC Driver using common event routing logic. Events and responses destined for a remote client are passed via a fast track path to the Operating System supplied network protocol stack at 2207 for delivery to the remote client.

Thus, as shown in Figure 6, a platform 600 developed in the practice of the invention can be designed to improve data processing speeds for financial market information, all while reducing the number of appliances needed for platform 600 (relative to conventional GPP-based systems) as well as the space consumed by such a platform. With a platform 600, a user such as a trader at a

work station 104 (or even a customer-supplied application software program 150 that accesses the platform via an application programming interface (API), can obtain a variety of information on the financial markets with less latency than would be expected from a conventional system. This improvement in latency can translate into tremendous value for practitioners of the invention.

While these figures illustrate several embodiments of FAM pipelines that can be implemented to process real time financial data streams, it should be noted that numerous other FAM pipelines could be readily devised and developed by persons having ordinary skill in the art following the teachings herein.

Further still it should be noted that for redundancy purposes and/or scaling purposes, redundant appliances 604, 606, 608, 610, 612, 614 and 616 can be deployed in a given market data platform 600.

Furthermore, it should also be noted that a practitioner of the present invention may choose to deploy less than all of the functionality described herein in reconfigurable logic. For example, device 604 may be arranged to perform only options pricing in reconfigurable logic, or some other subset of the functions listed in Figure 6. If a user later wanted to add additional functionality to device 604, it can do so by simply re-configuring the reconfigurable logic of system 200 to add any desired new functionality. Also, the dashed boxes shown in Figure 6 enclose data processing functionality that can be considered to belong to the same category of data processing operations. That is, devices 612 and 614 can be categorized as management operations. Device 604 can be categorized as providing feed handling/processing for data access, value-added services, and historic services. Devices 606, 608 and 610 can be categorized as direct market access trading systems. As improvements to reconfigurable logic continues over time such that more resources become available thereon (e.g., more available memory on FPGAs), the inventors envision that further consolidation of financial data processing functionality can be achieved by combining data processing operations of like categories, as indicated by the dashed boxes, thereby further reducing the number of appliances 200 needed to implement platform 600. Further

still, in the event of such resource improvements over time for FPGAs, it can be foreseen that even further consolidation occur, including consolidation of all functionality shown in Figure 6 on a single system 200.

5 While the present invention has been described above in relation to its preferred embodiments, various modifications may be made thereto that still fall within the invention's scope as will be recognizable upon review of the teachings herein. As such, the full scope of the present invention is to be defined solely by the
10 appended claims.

The embodiments of the present invention for which an exclusive property or privilege is claimed are defined as follows:

1. An apparatus for processing financial market data, the apparatus comprising:

a reconfigurable logic device with firmware logic deployed thereon, the firmware logic being configured to perform a specified data processing operation on the financial market data; and

a processor in communication with the reconfigurable logic device, the processor being programmed with software logic, the software logic being configured to (1) receive the financial market data as an input, and (2) manage a flow of the received financial market data into and out of the reconfigurable logic device.

2. The apparatus of claim 1 wherein the financial market data comprises a plurality of financial market data messages, and wherein the software logic is further configured to manage the flow of the received financial market data into and out of the reconfigurable logic device such that each message passes only once from the software logic to the firmware logic.

3. The apparatus of claim 2 wherein the firmware logic is further configured with a firmware pipeline that is configured to perform a plurality of different financial data processing operations on the received flow to thereby generate the flow of firmware-processed financial market data.

4. The apparatus of claim 3 further comprising a memory in which a plurality of financial instrument records are stored in memory addresses associated with a financial instrument symbology that is internal to the apparatus, wherein the reconfigurable logic is in communication with the memory, wherein the financial market data

messages relate to financial instruments, wherein the financial market data messages reference their related financial instruments using a financial instrument symbology that is different than the internal financial instrument symbology, and wherein the firmware logic is further configured to (1) convert each received financial market data message to the internal financial instrument symbology, (2) retrieve financial instrument records pertaining to the received financial market data messages from the memory using the internal financial instrument symbology for the received financial market data messages, and (3) update the financial instrument records in response to the financial data processing operations performed on the received financial market data messages.

5. The apparatus of claim 1 wherein the financial market data comprises a plurality of financial market data messages, and wherein the software logic is further configured to manage the flow of the received financial market data into and out of the reconfigurable logic device such that each message passes only once from the firmware logic back to the software logic.

6. The apparatus of claim 1 wherein the financial market data comprises a plurality of financial market data messages, wherein processor is further configured to receive the financial market data messages from a plurality of different financial market data feeds, and wherein the software logic is further configured to (1) maintain a plurality of different thread groups simultaneously, (2) assign each financial market data message to a thread group based on a configuration file, wherein the configuration file is configured to associate each thread group with a different financial market data feed such that messages from a given one of the feeds will be processed by the thread group associated with that message's feed, and (3) normalize the financial market data messages with the thread groups such that each thread group normalizes the financial market data messages assigned thereto independently of the other thread groups.

7. The apparatus of claim 6 wherein the configuration file is further configured to associate at least one thread group with a plurality of different financial market data feeds.

8. The apparatus of claim 1 wherein the financial market data comprises a plurality of financial market data messages, and wherein the software logic is further configured to (1) receive the financial market data messages destined for the firmware logic via a kernel mode process supported an operating system of the processor, and (2) transfer the financial market data messages destined for the firmware logic from the kernel mode process supported by an operating system of the processor to a user mode process supported by the operating system without a transition from a kernel space mode to a user address space mode by writing each received financial market data message destined for the firmware logic to a buffer that is mapped into both a kernel space and a user address space supported by the operating system.

9. The apparatus of claim 1 wherein the financial market data comprises a plurality of financial market data messages, and wherein the software logic further comprises a hardware interface driver and a data routing driver in communication with the hardware interface driver, wherein the hardware interface driver is configured to receive financial market data messages that have been processed by the firmware logic, and wherein the data routing driver is configured to (1) receive the firmware-processed financial market data messages from the hardware interface driver, (2) provide a first path and a second path for the firmware-processed financial market data messages, the first path comprising a path for the firmware-processed financial market data messages to at least one client through a network protocol stack, the second path comprising a path for the firmware-processed financial market data messages to an exceptions handling software module.

10. The apparatus of claim 9 wherein the data routing driver is further configured to interrogate at least a portion of the firmware-processed financial market data messages to determine the path over which each firmware-processed financial market data message should travel.

11. The apparatus of claim 9 wherein the first path is wholly within a kernel space supported by the operating system of the processor.

12. The apparatus of claim 1 wherein the financial market data comprises a plurality of financial market data messages, and wherein the software logic comprises:

a protocol stack resident in a kernel space supported by an operating system of the processor, the protocol stack configured to receive the financial market data messages; and

a driver configured to connect, in the kernel space, to the protocol stack to access the financial market data messages.

13. The apparatus of claim 12 wherein the driver is further configured to write a plurality of financial market data messages destined for the firmware logic to a buffer that is mapped into both the kernel space and a user address space supported by the operating system of the processor.

14. The apparatus of claim 13 wherein the buffer comprises a plurality of input buffers that are mapped into both the kernel space and the user address space, each buffer being associated with a thread group, wherein the driver is further configured to (1) assign each of the plurality of financial market data messages destined for the firmware logic to a thread group from among a plurality of thread groups based on a configuration file, and (2) write each of the plurality of financial market data messages destined for the firmware logic to the buffer associated with its assigned thread group, and wherein each of the thread groups is configured to process, in the

user address space, the financial market data messages present in its associated buffer independently of the other thread groups.

15. The apparatus of claim 14 wherein each of the thread groups is further configured to write its processed financial market data messages to an output buffer that is mapped into both the kernel space and the user address space, wherein the software logic further comprises a hardware interface driver resident in the kernel space, the hardware interface driver configured to transfer the processed financial market data messages from the output buffers to the firmware logic deployed on the reconfigurable logic device.

16. The apparatus of claim 15 wherein the protocol stack comprises a first protocol stack, and wherein the software logic further comprises:

- a second protocol stack resident in the kernel space; and
- a data routing driver configured to, in the kernel space, (1) receive financial market data messages from the reconfigurable logic device, (2) connect to the second protocol stack, and (3) selectively route a subset of the received financial market data messages to the second protocol stack for subsequent delivery to downstream clients.

17. The apparatus of claim 1 wherein the software logic is further configured to issue a control command to the firmware logic in response to an instruction received by the processor from a client computer in communication therewith, the control command at least partially specifying the data processing operation to be performed by the firmware logic.

18. A method for processing financial market data, the method comprising:

- controlling with software logic a flow of financial market data from the software logic to and from firmware logic; and

performing a specified financial data processing operation on financial market data with firmware logic on a reconfigurable logic device in response to the software logic controlling step.

19. The method of claim 18 wherein the financial market data comprises a plurality of financial market data messages, and wherein the controlling step further comprises controlling the flow of the financial market data such that each message passes only once from the software logic to the firmware logic.

20. The method of claim 19 wherein the performing step comprises performing with the firmware logic a plurality of different financial data processing operations on the received flow to thereby generate the flow of firmware-processed financial market data.

21. The method of claim 20 wherein the financial market data messages relate to financial instruments, wherein the financial market data messages reference their related financial instruments using a first financial instrument symbology, the method further comprising:

storing a plurality of financial instrument records in memory addresses associated with a second financial instrument symbology that is different than the first financial instrument symbology; and

using the firmware logic, (1) converting each received financial market data message to the second financial instrument symbology, (2) retrieving stored financial instrument records pertaining to the received financial market data messages using the second financial instrument symbology for the received financial market data messages, and (3) updating the financial instrument records in response to at least one of the financial data processing operations performed on the received financial market data messages.

22. The method of claim 18 wherein the financial market data comprises a plurality of financial market data messages, and wherein the controlling step further comprises controlling the flow of the financial market data such that each message passes only once from the firmware logic to the software logic.

23. The method of claim 18 wherein the financial market data comprises a plurality of financial market data messages, wherein a processor executes the software logic, the processor receiving the financial market data messages from a plurality of different financial market data feeds, and wherein the controlling step further comprises the software logic is (1) maintaining a plurality of different thread groups simultaneously, (2) assigning each financial market data message to a thread group based on a configuration file, wherein the configuration file associates each thread group with a different financial market data feed such that messages from a given one of the feeds will be processed by the thread group associated with that message's feed, and (3) normalizing the financial market data messages with the thread groups such that each thread group normalizes the financial market data messages assigned thereto independently of the other thread groups.

24. The method of claim 23 wherein the configuration file associates at least one thread group with a plurality of different financial market data feeds.

25. The method of claim 18 wherein the financial market data comprises a plurality of financial market data messages, and wherein controlling step comprises the software logic:

receiving the financial market data messages destined for the firmware logic via a kernel mode process supported an operating system of the processor; and

transferring the financial market data messages destined for the firmware logic from the kernel mode process supported by an operating system of the processor to a user mode process supported by the operating system without a transition from a kernel space mode to a user address space mode by writing each received financial market data message destined for the firmware logic to a buffer that is mapped into both a kernel space and a user address space supported by the operating system.

26. The method of claim 18 wherein the financial market data comprises a plurality of financial market data messages, and wherein the software logic further comprises a hardware interface driver and a data routing driver in communication with the hardware interface driver, the hardware interface driver receiving financial market data messages that have been processed by the firmware logic, and the data routing driver (1) receiving the firmware-processed financial market data messages from the hardware interface driver, (2) providing a first path and a second path for the firmware-processed financial market data messages, the first path comprising a path for the firmware-processed financial market data messages to at least one client through a network protocol stack, the second path comprising a path for the firmware-processed financial market data messages to an exceptions handling software module.

27. The method of claim 26, the data routing driver interrogating at least a portion of the firmware-processed financial market data messages to determine the path over which each firmware-processed financial market data message should travel.

28. The method of claim 26 wherein the first path is wholly within a kernel space supported by the operating system of a processor.

29. The method of claim 18 wherein the controlling step further comprises the software logic issuing a control command to the firmware logic in response to an instruction received by the processor from a client computer in communication therewith, the control command at least partially specifying the data processing operation to be performed by the firmware logic.

30. The method of claim 18 wherein the financial market data comprises a plurality of financial market data messages, and wherein the software logic comprises a driver and a protocol stack resident in a kernel space supported by an operating system of a processor, the protocol stack receiving the financial market data messages, and the driver connecting, in the kernel space, to the protocol stack and accessing the financial market data messages.

31. The method of claim 30, the driver writing a plurality of financial market data messages destined for the firmware logic to a buffer that is mapped into both the kernel space and a user address space supported by the operating system of the processor.

32. The method of claim 31 wherein the buffer comprises a plurality of input buffers that are mapped into both the kernel space and the user address space, each buffer being associated with a thread group, the driver (1) assigning each of the plurality of financial market data messages destined for the firmware logic to a thread group from among a plurality of thread groups based on a configuration file, and (2) writing each of the plurality of financial market data messages destined for the firmware logic to the buffer associated with its assigned thread group, and each of the thread groups processing, in the user address space, the financial market data messages present in its associated buffer independently of the other thread groups.

33. The method of claim 32, each of the thread groups writing its processed financial market data messages to an output buffer that is mapped into both the kernel space and the user address space, wherein the software logic further comprises a hardware interface driver, the hardware interface driver transferring the processed financial market data messages from the output buffers to the firmware logic.

34. The method of claim 33 wherein the protocol stack comprises a first protocol stack, and wherein the software logic further comprises a second protocol stack resident in the kernel space and a data routing driver, the data routing driver performing the following steps in the kernel space: (1) receiving financial market data messages from the reconfigurable logic device, (2) connecting to the second protocol stack, and (3) selectively routing a subset of the received financial market data messages to the second protocol stack for subsequent delivery to downstream clients.

35. An apparatus comprising:

a computer system comprising a processor and a reconfigurable logic device that are configured to cooperate with each other to process streaming financial market data relating to a plurality of financial instruments and manage a delivery of the processed financial market data for consumption by a plurality of clients;

wherein the processor is configured to execute an operating system, the operating system including a user space for a user mode and a kernel space for a kernel mode;

wherein the reconfigurable logic device is configured to (1) perform a processing operation on the streaming financial market data to thereby generate processed financial market data and (2) stream the processed financial market data out of the reconfigurable logic device into a memory within the kernel space of the operating system via DMA operations; and

wherein a driver that executes within the kernel space of the operating system while the operating system is in the kernel mode is configured to (1) read the processed financial market data from the memory, and (2) selectively route the processed financial market data for delivery to the clients via a path that is wholly within the kernel space.

36. The apparatus of claim 35 wherein the reconfigurable logic device comprises a field programmable gate array (FPGA).

37. The apparatus of claim 36 wherein the processor comprises a general purpose processor (GPP).

38. The apparatus of claim 35 wherein the driver is further configured to directly route the processed financial market data to a destination within the kernel space while the operating system is in the kernel mode without a transition to the user mode, the destination being associated with a client.

39. The apparatus of claim 35 wherein the operating system comprises a network protocol stack through which the computer system receives a feed of the streaming financial market data;

wherein another driver that executes within the kernel space of the operating system while the operating system is in the kernel mode is configured to (1) maintain a kernel level interface into the network protocol stack, and (2) copy the streaming financial market data from the network protocol stack into a shared memory that is mapped into the kernel space and the user space; and

wherein user mode code that executes within the user space of the operating system is configured to access financial market data from the shared memory and perform a processing operation on the accessed financial market data.

40. The apparatus of claim 39 wherein the user mode code comprises a plurality of threads configured to (1) normalize the accessed financial market data, and (2) write the normalized financial market data to additional shared memory that is mapped into the kernel space and the user space;

wherein the driver comprises a hardware interface driver, the hardware interface driver configured facilitate DMA transfers of the normalized financial market data from the additional shared memory into the reconfigurable logic device; and

wherein the reconfigurable logic device is further configured to (1) stream in the normalized financial market data via DMA transfers from the additional shared memory, and (2) perform the processing operation on the streaming normalized financial market data to generate the processed financial market data.

41. The apparatus of claim 40 wherein the processing operation performed by the reconfigurable logic device comprises a member of the group consisting of (1) a last value cache (LVC) operation, (2) an order book maintenance operation, and (3) an interest and entitlement filtering operation.

42. The apparatus of claim 40 wherein the additional shared memory comprises a ring buffer.

43. The apparatus of claim 39 wherein the shared memory comprises a ring buffer.

44. The apparatus of claim 35 wherein the operating system comprises a network protocol stack through which a plurality of the clients connect with the computer system to access processed financial market data; and

wherein the driver is further configured to maintain a kernel level interface into the network protocol stack and deliver the processed financial market data to the network protocol stack through the kernel level interface while the operating system is in the kernel mode.

45. The apparatus of claim 35 wherein the driver comprises a first driver and a second driver, wherein the first and second drivers define the path wholly within the kernel space for delivery of the processed financial market data from the reconfigurable logic device to the clients without a transition by the operating system from the kernel mode to the user mode as part of the delivery;

wherein the first driver is configured to read the processed financial market data from the memory; and

wherein the second driver is configured to selectively route the processed financial market data within the kernel space for delivery to the clients.

46. The apparatus of claim 35 wherein the computer system comprises a ticker plant.

47. A method for processing financial market data using a system comprising a processor and a reconfigurable logic device, the method comprising:

the processor executing an operating system, the operating system including a user space for a user mode and a kernel space for a kernel mode, wherein the executing step comprises:

a network protocol stack executed by the operating system receiving a feed of streaming financial market data for a plurality of financial instruments;

a first driver executed within the kernel space of the operating system while the operating system is in the kernel mode (1) maintaining a kernel level interface into the network protocol stack, and (2) copying the streaming financial market data from the network protocol stack into a shared memory that is mapped into the kernel space and the user space, wherein the copying is performed by the driver without the operating system transitioning to the user mode;

user mode code executed within the user space of the operating system (1) accessing the financial market data from the shared memory, (2) normalizing the accessed financial market data, and (3) writing the normalized financial market data to additional shared memory that is mapped into the kernel space and the user space; and

a second driver executed within the kernel space of the operating system while the operating system is in the kernel mode facilitating DMA transfers of the normalized financial market data from the additional shared memory into the reconfigurable logic device;

the reconfigurable logic device streaming the normalized financial market data into the reconfigurable logic device from the additional shared memory via DMA operations, the reconfigurable logic device having a resident firmware pipeline; and

the firmware pipeline (1) receiving the streaming normalized financial market data, and (2) performing a processing operation on the received streaming normalized financial market data to generate processed financial market data for delivery to a plurality of clients.

48. The method of claim 28 wherein the reconfigurable logic device further comprises a firmware socket module, the method further comprising:

the firmware socket module (1) streaming the normalized financial market data into the reconfigurable logic device from the additional shared memory via DMA operations, and (2) providing the streaming normalized financial market data to the firmware pipeline.

49. The method of claim 48 wherein the streaming step further comprises the firmware socket module providing both command data and the streaming normalized financial market data to the firmware pipeline, the method further comprising:

the firmware pipeline arranging itself in accordance with the command data to control how the processing operation is performed.

50. The method of claim 49 wherein the reconfigurable logic device is in communication with the processor via a bus, the method further comprising:

the reconfigurable logic device accessing the command data and the streaming normalized financial market data via the bus; and
the processor generating the command data.

51. The method of claim 47 wherein the executing step further comprises the processor directly routing the processed financial market data for delivery to the clients without the operating system transitioning to the user mode as part of the routing.

52. The method of claim 47 wherein the user mode code comprises a plurality of threads, each thread executing to normalize a different group of the accessed financial market data independently of the other threads.

53. The method of claim 47 wherein the reconfigurable logic device comprises a field programmable gate array (FPGA).

54. The method of claim 47 wherein the processor and reconfigurable logic device are part of a ticker plant.

55. The method of claim 47 wherein the shared memory comprises a ring buffer.

56. The method of claim 47 wherein the additional shared memory comprises a ring buffer.

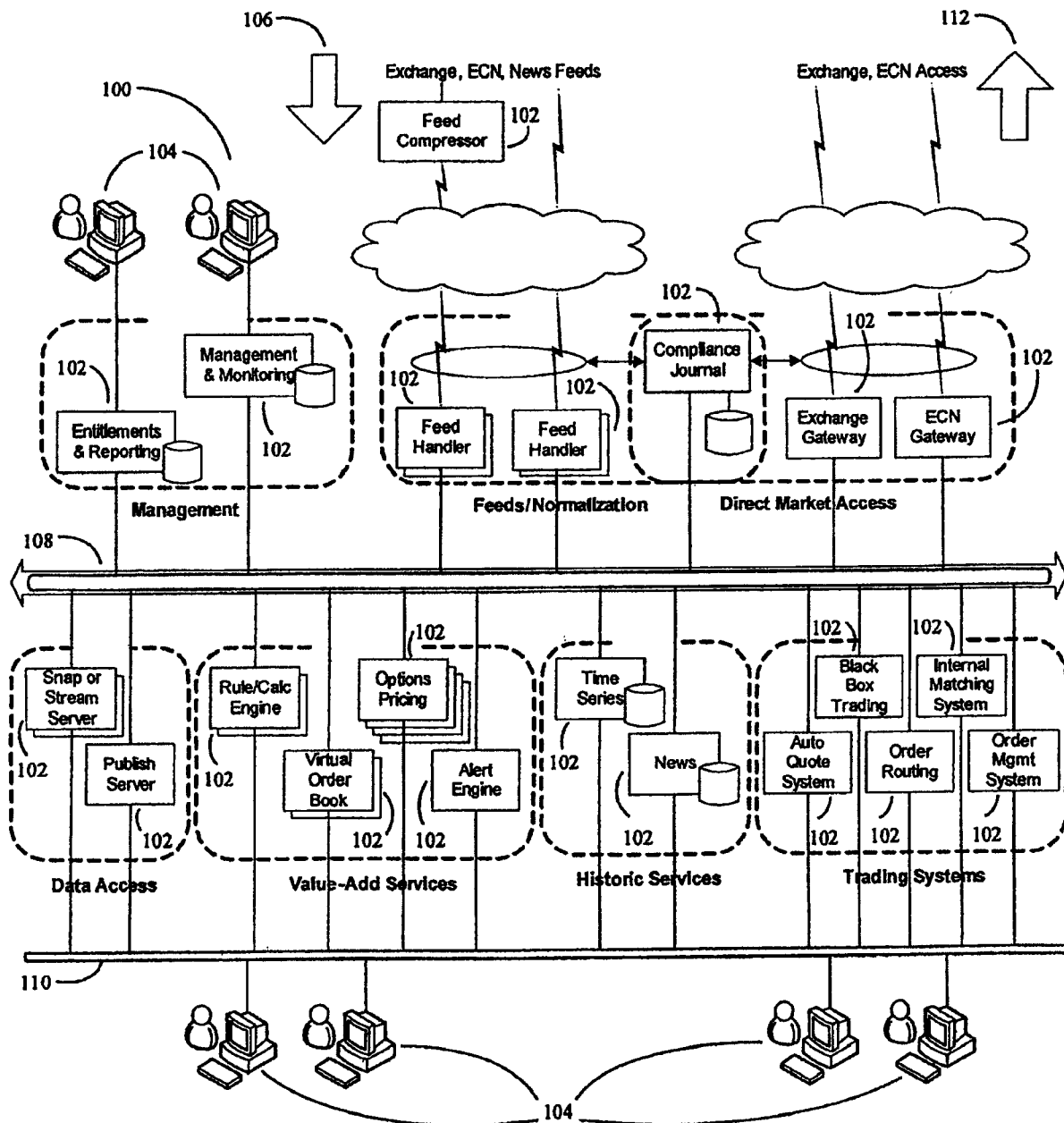


Figure 1
PRIOR ART

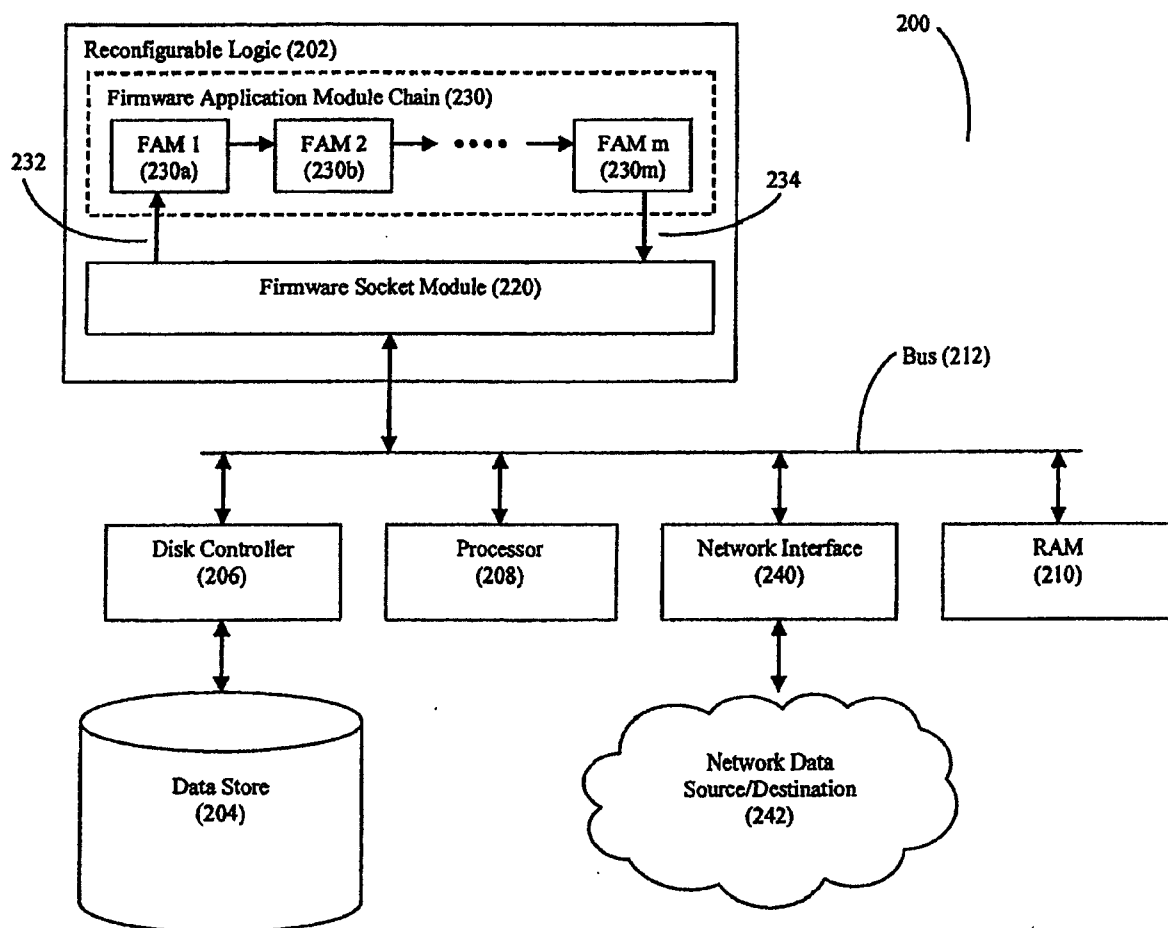


Figure 2

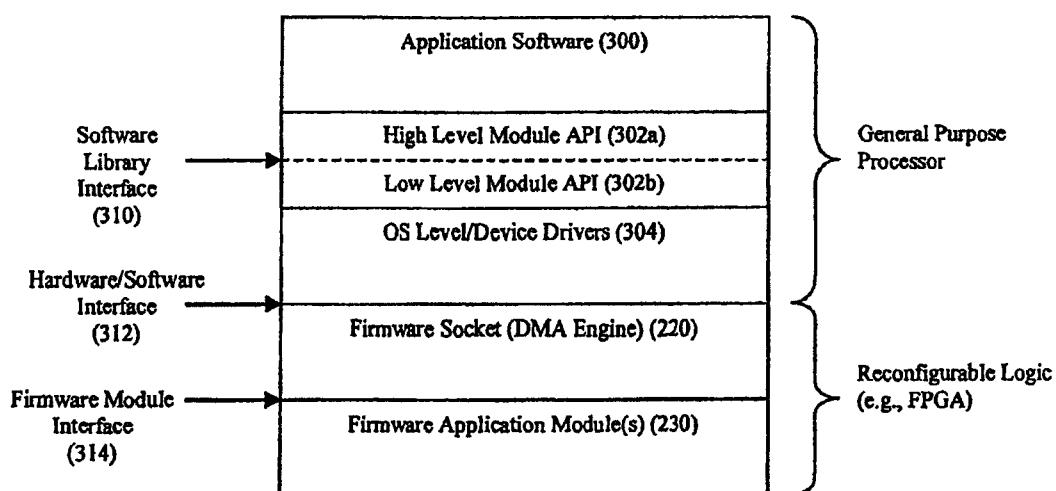


Figure 3

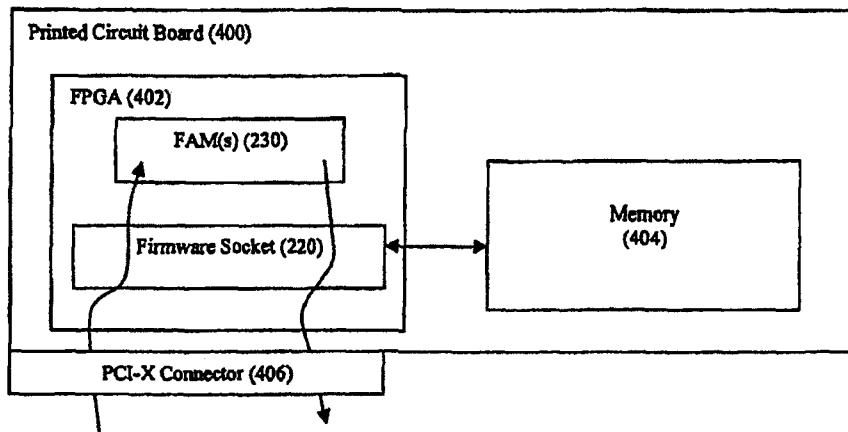


Figure 4(a)

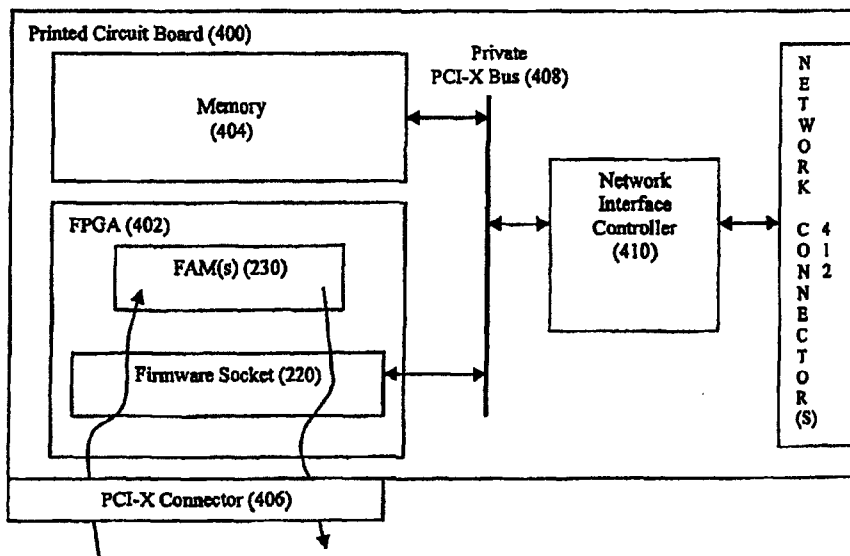


Figure 4(b)

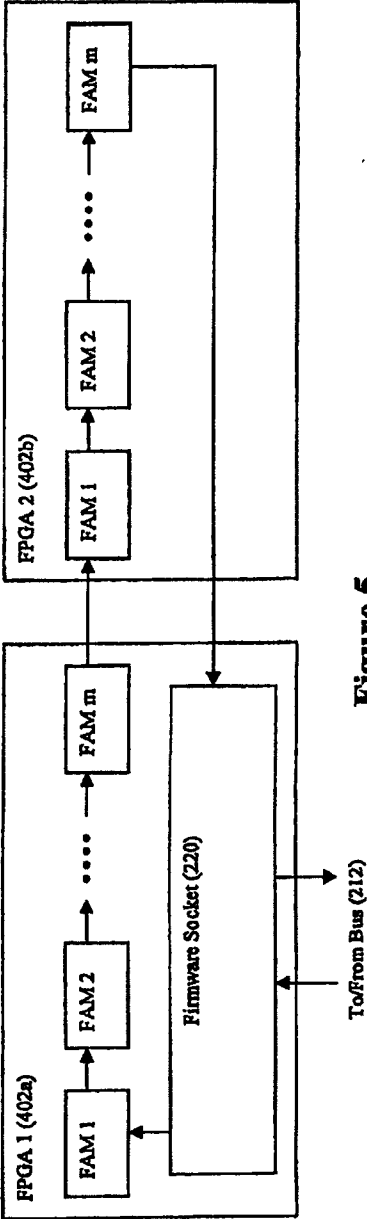


Figure 5

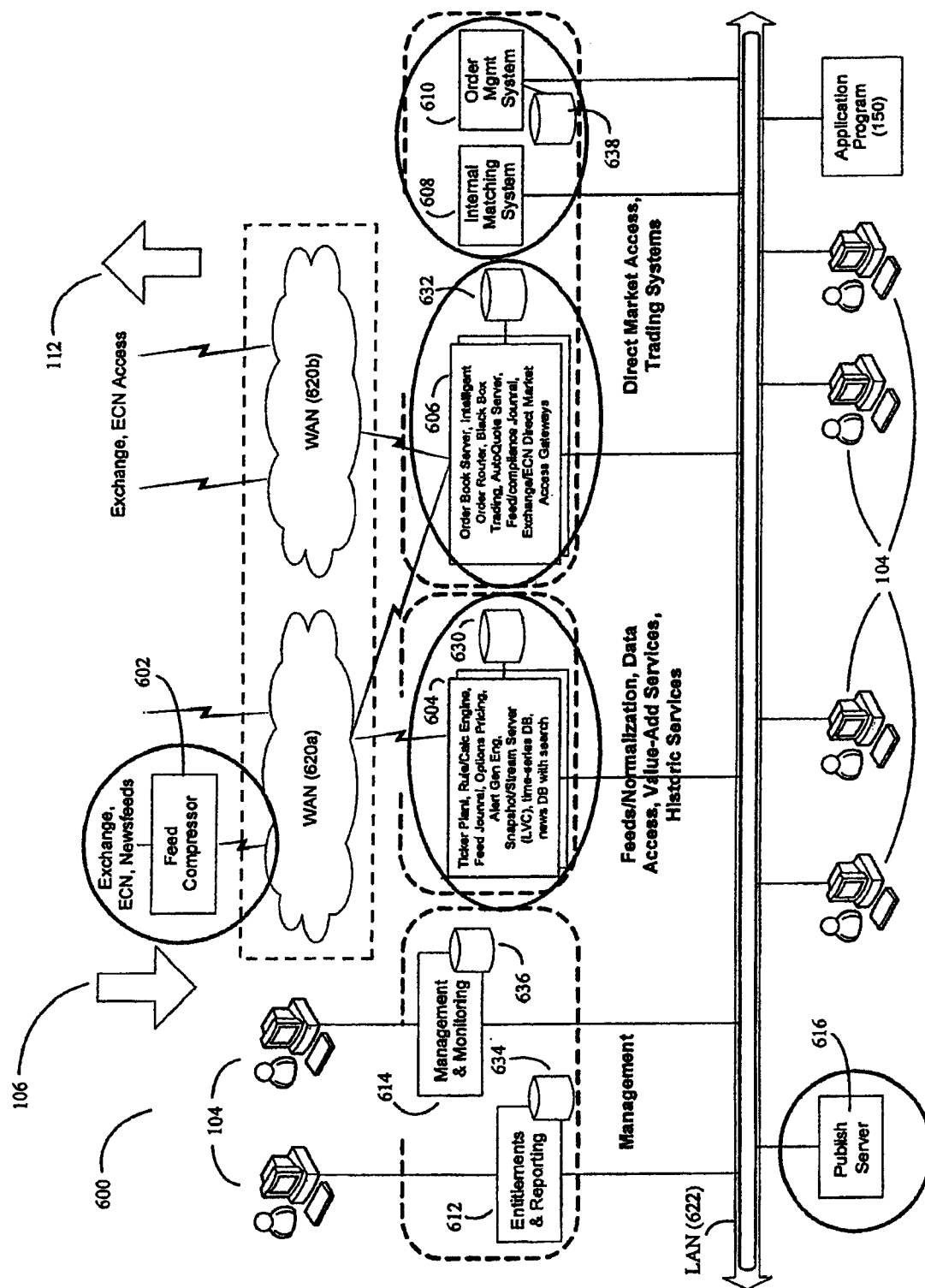


Figure 6

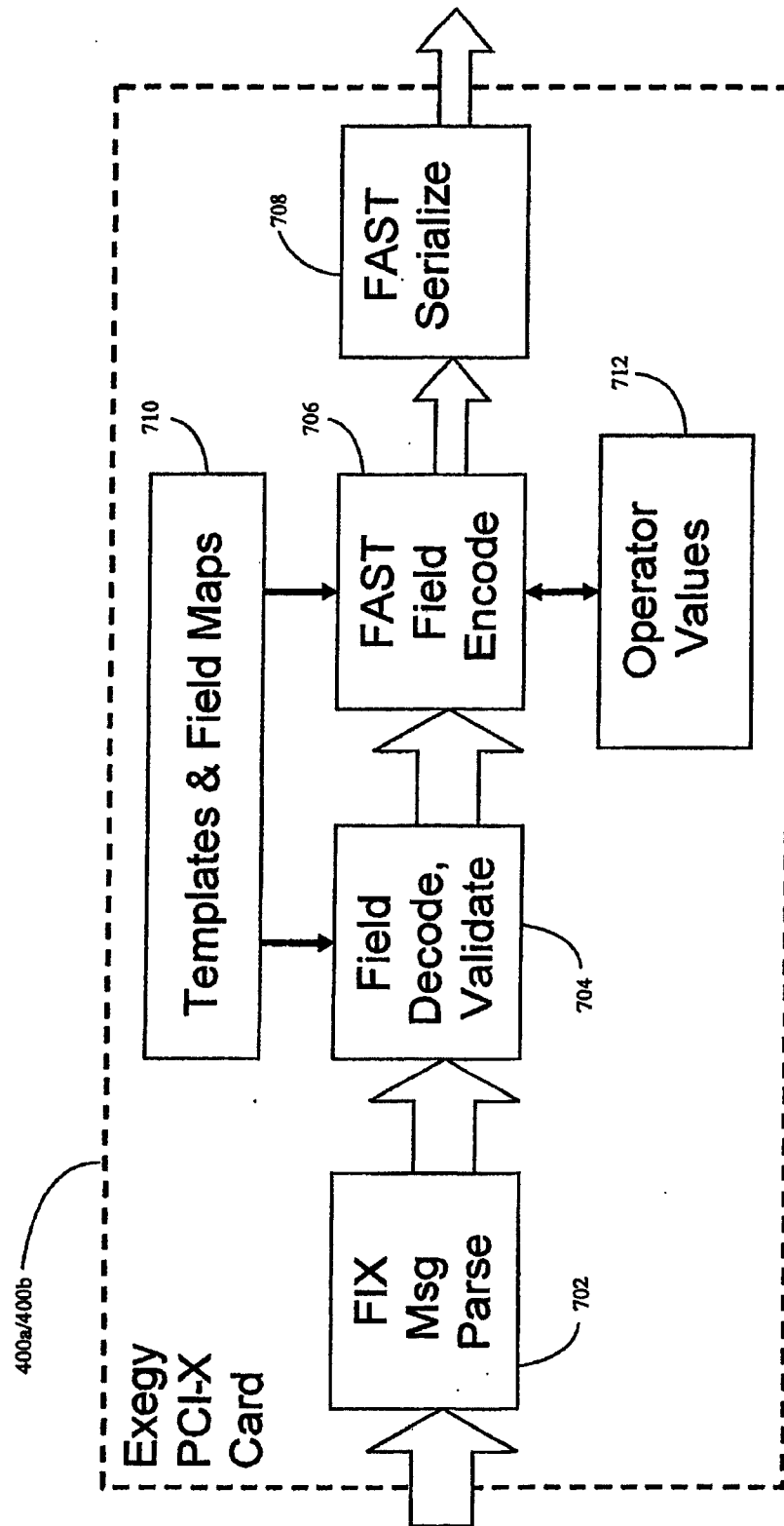


Figure 7

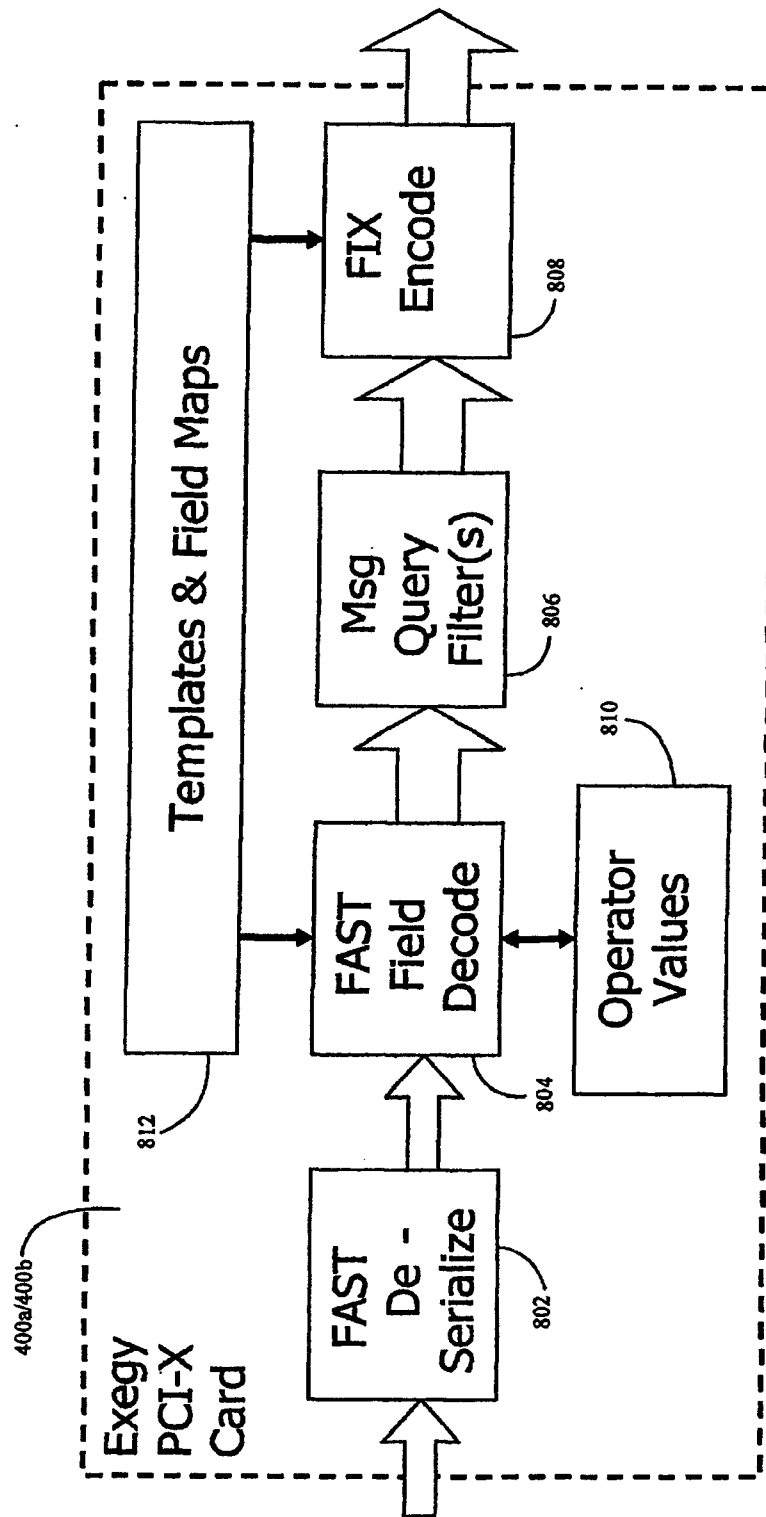


Figure 8

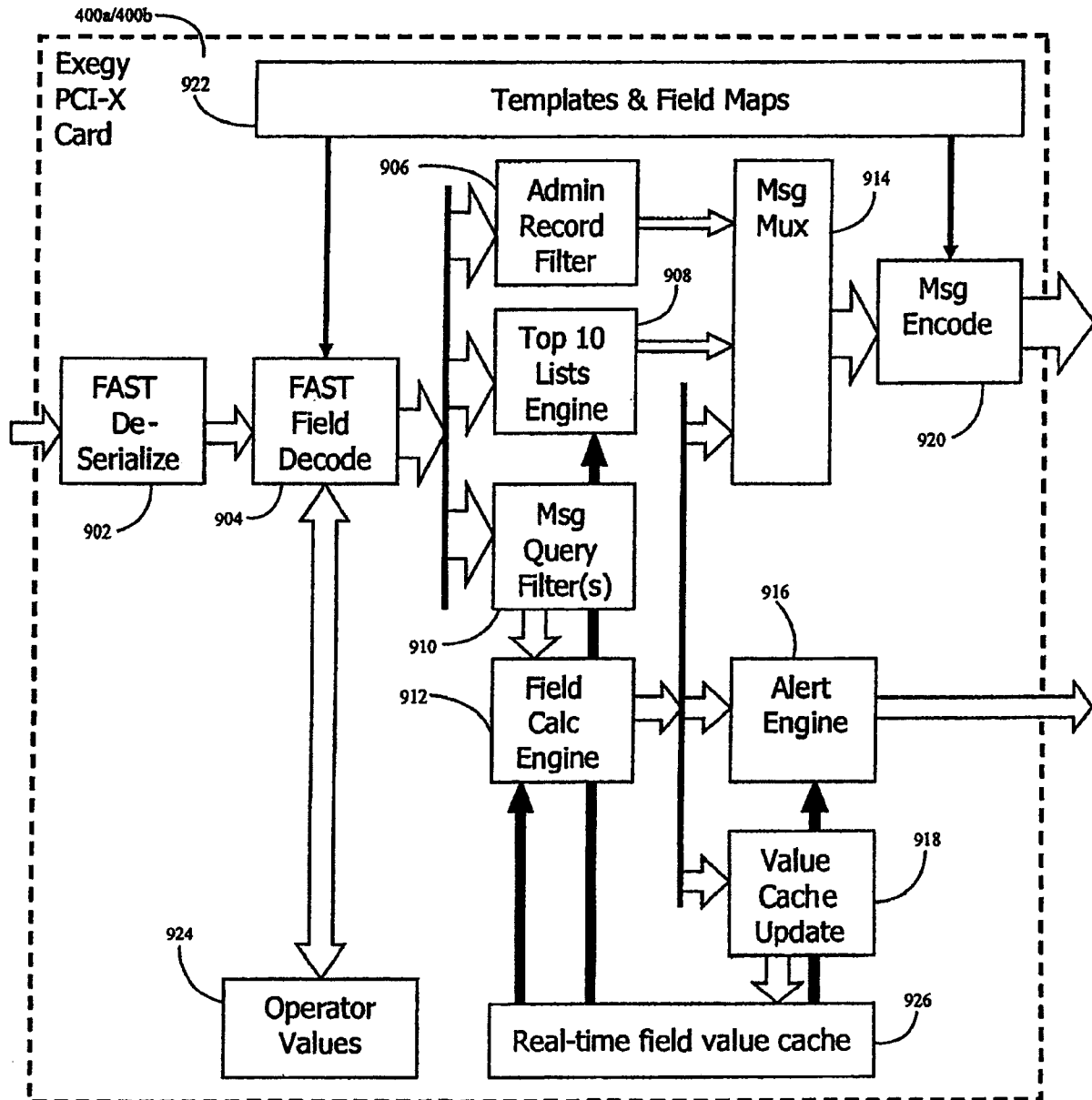


Figure 9

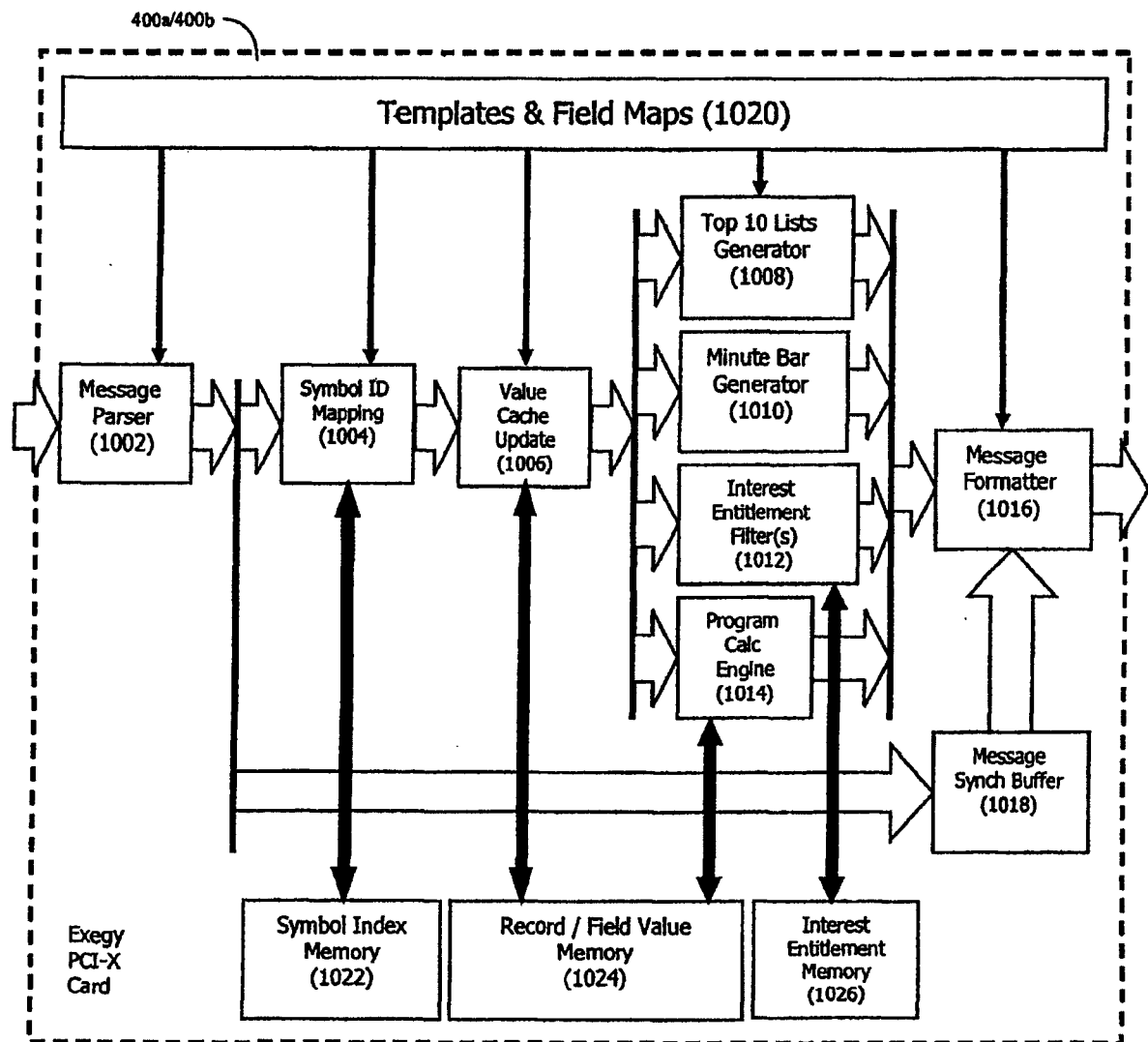
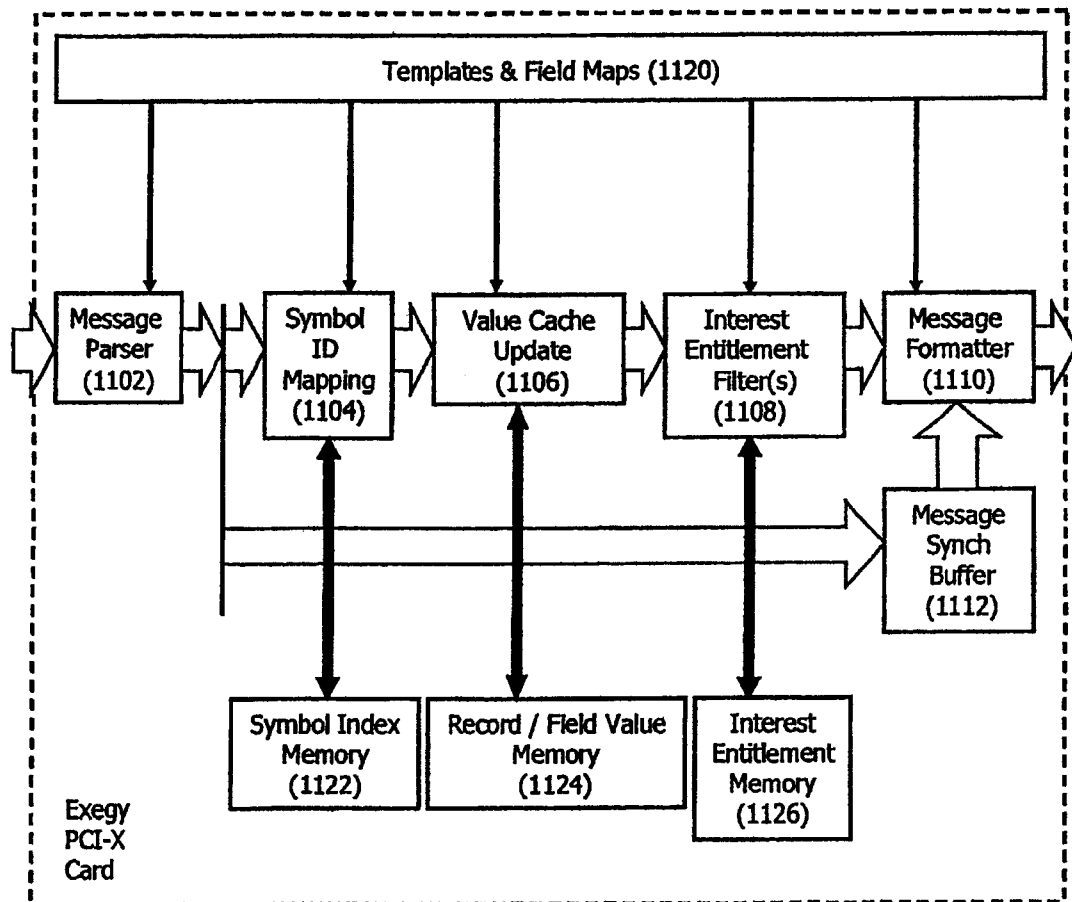


Figure 10

**Figure 11**

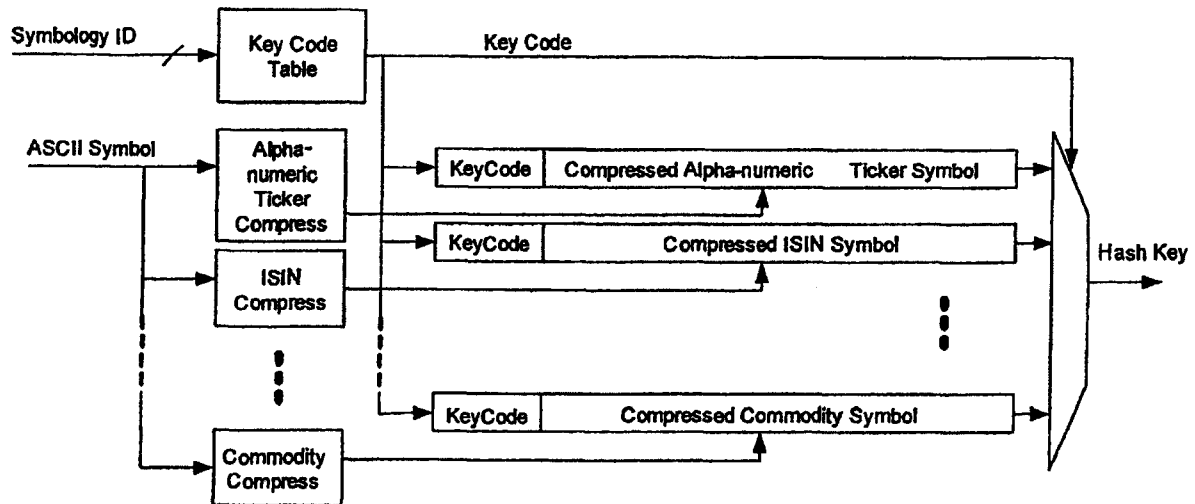


Figure 12

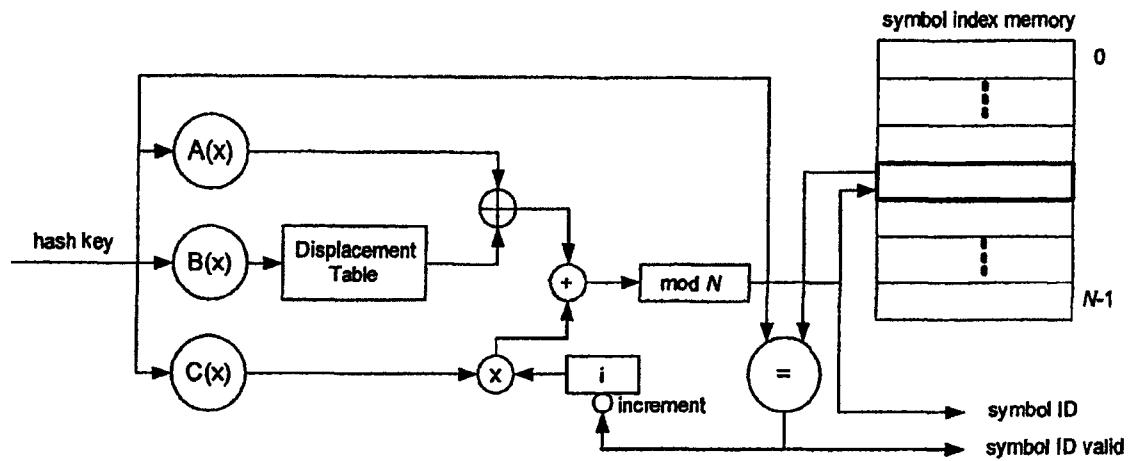


Figure 13

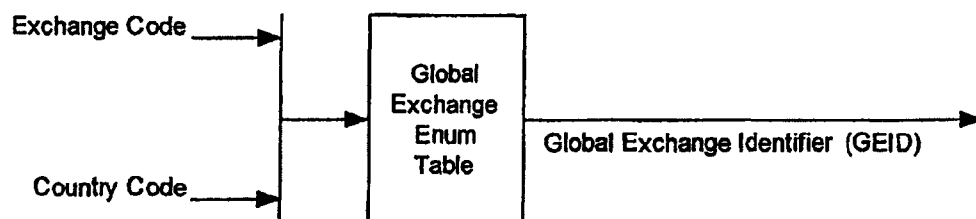


Figure 14

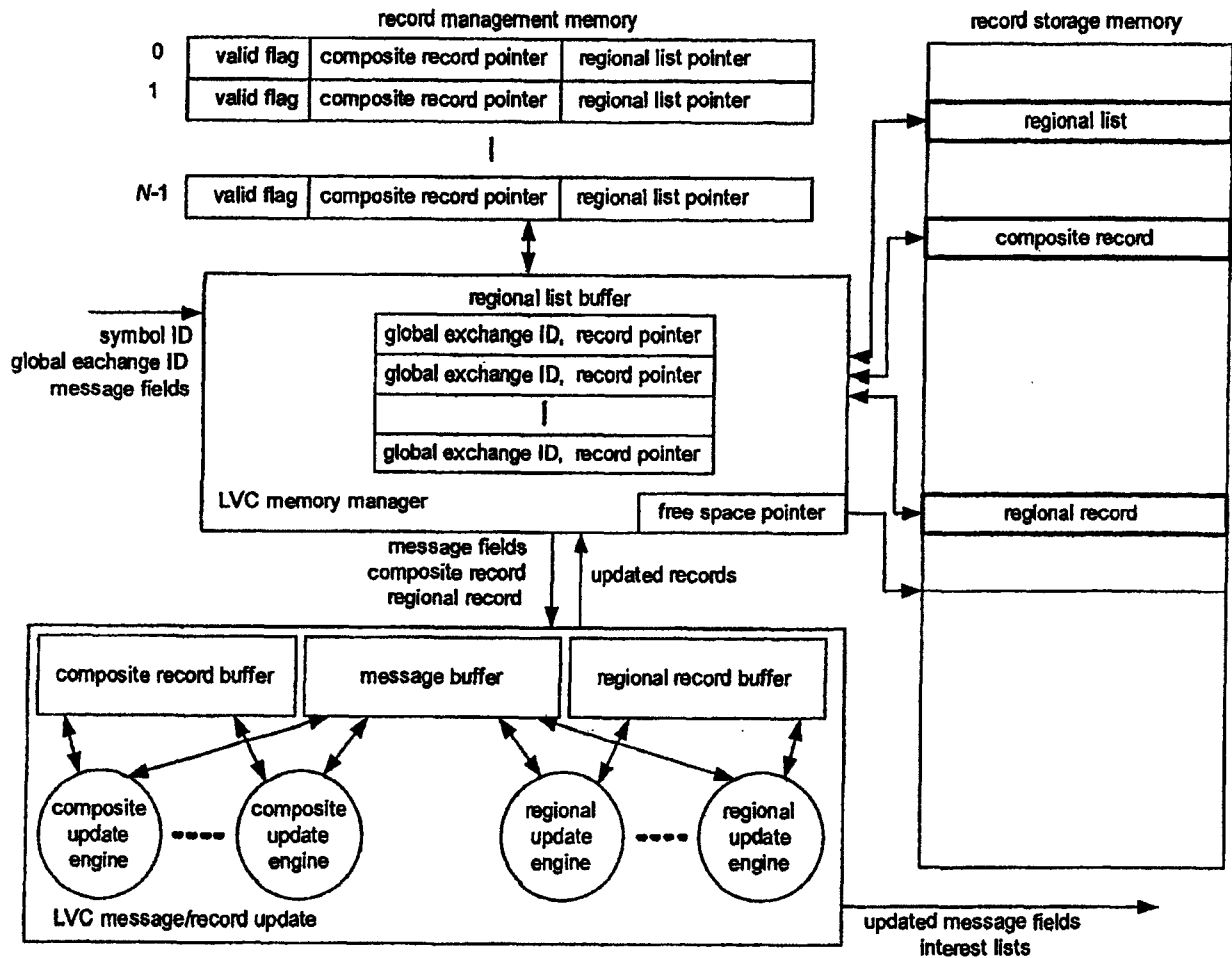


Figure 15 (a)

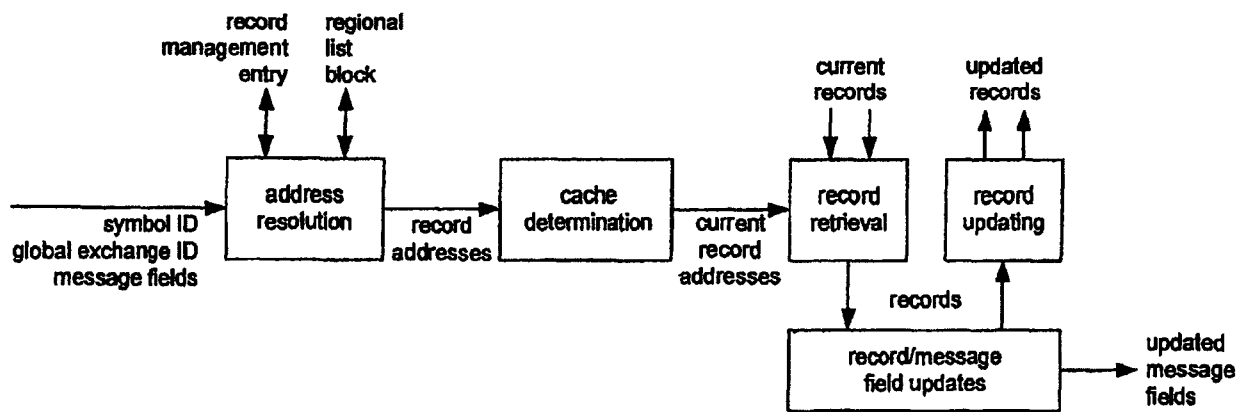


Figure 15(b)

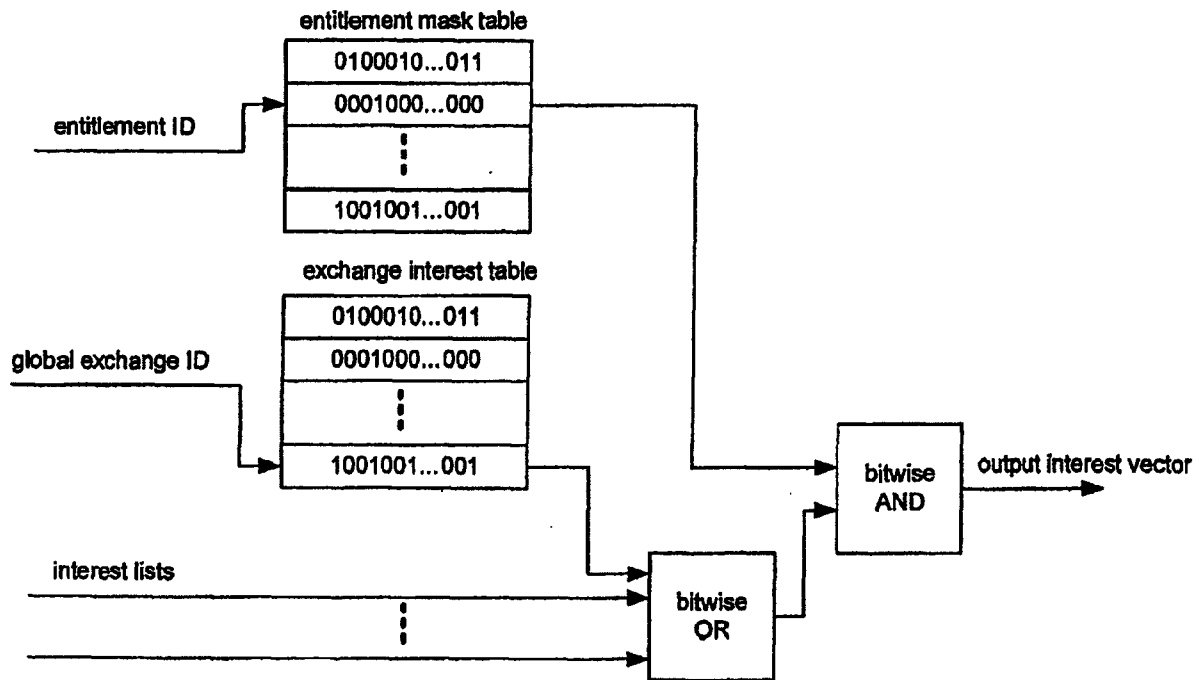


Figure 16

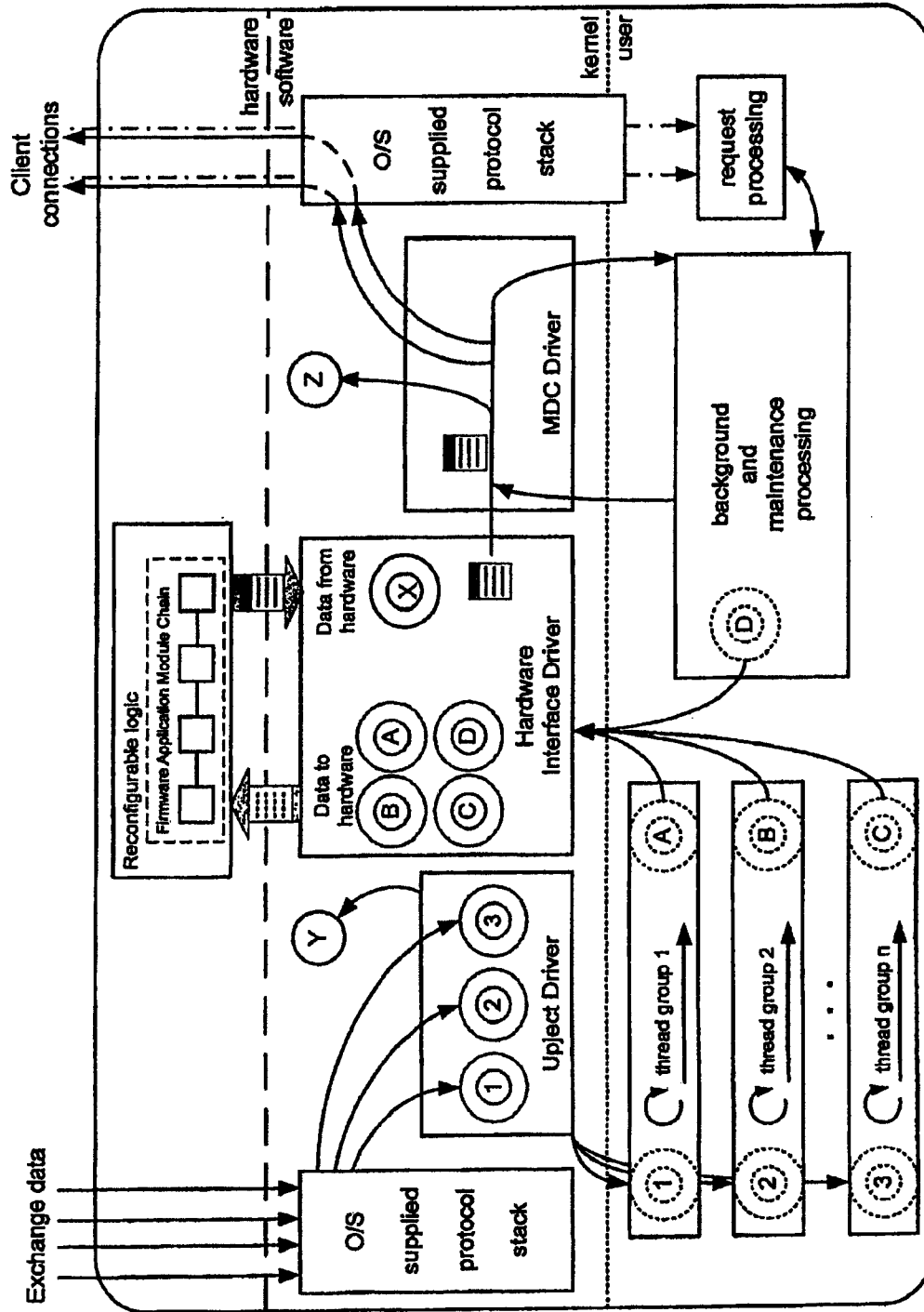


Figure 17

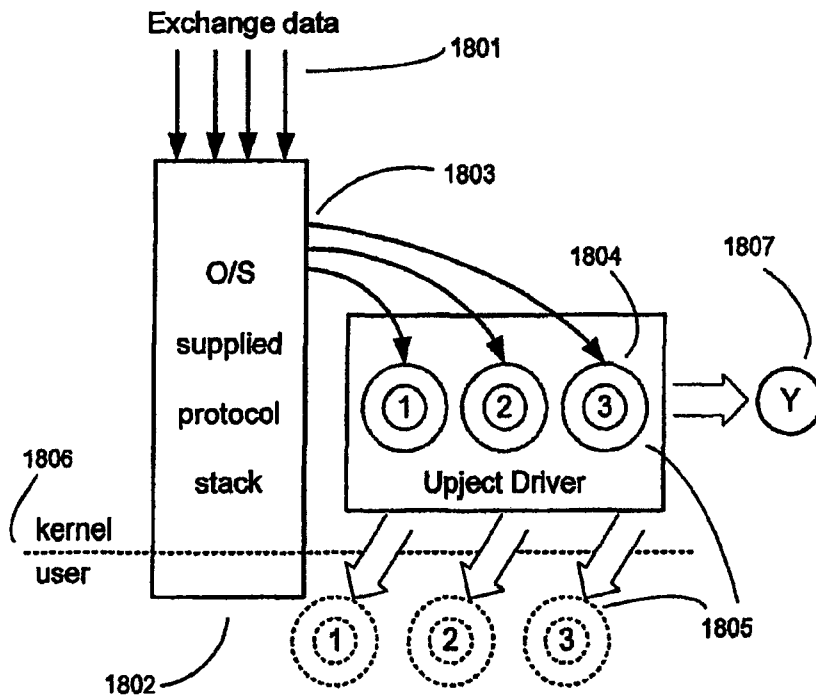


Figure 18

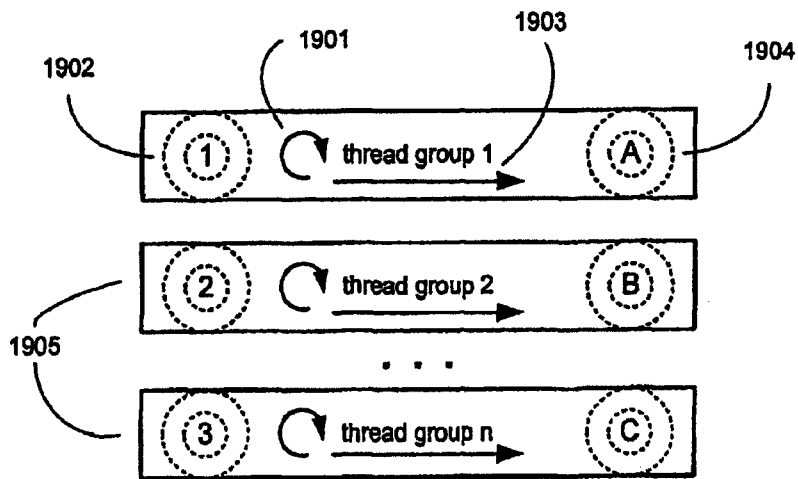
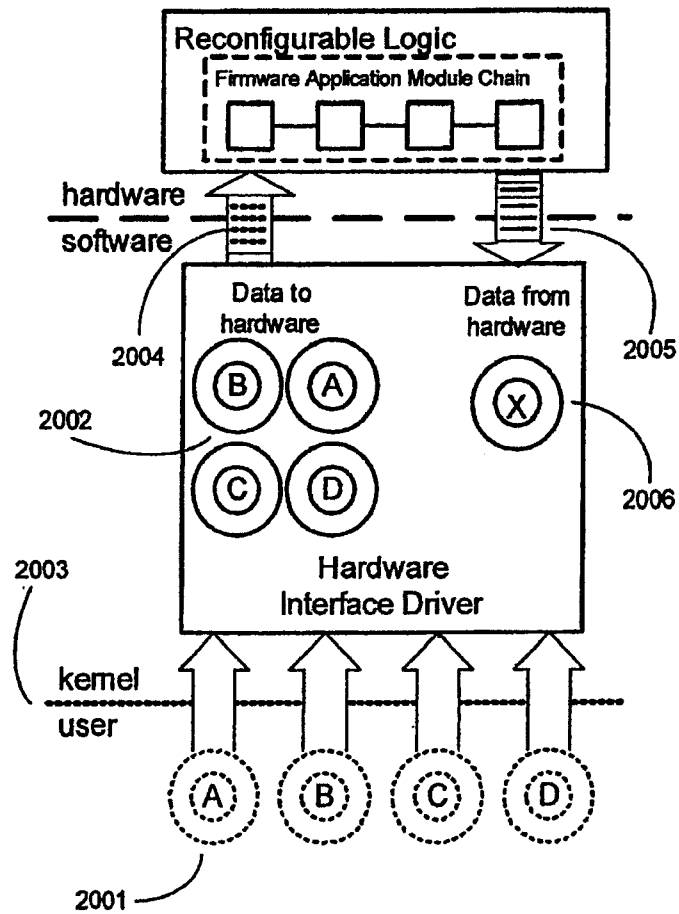
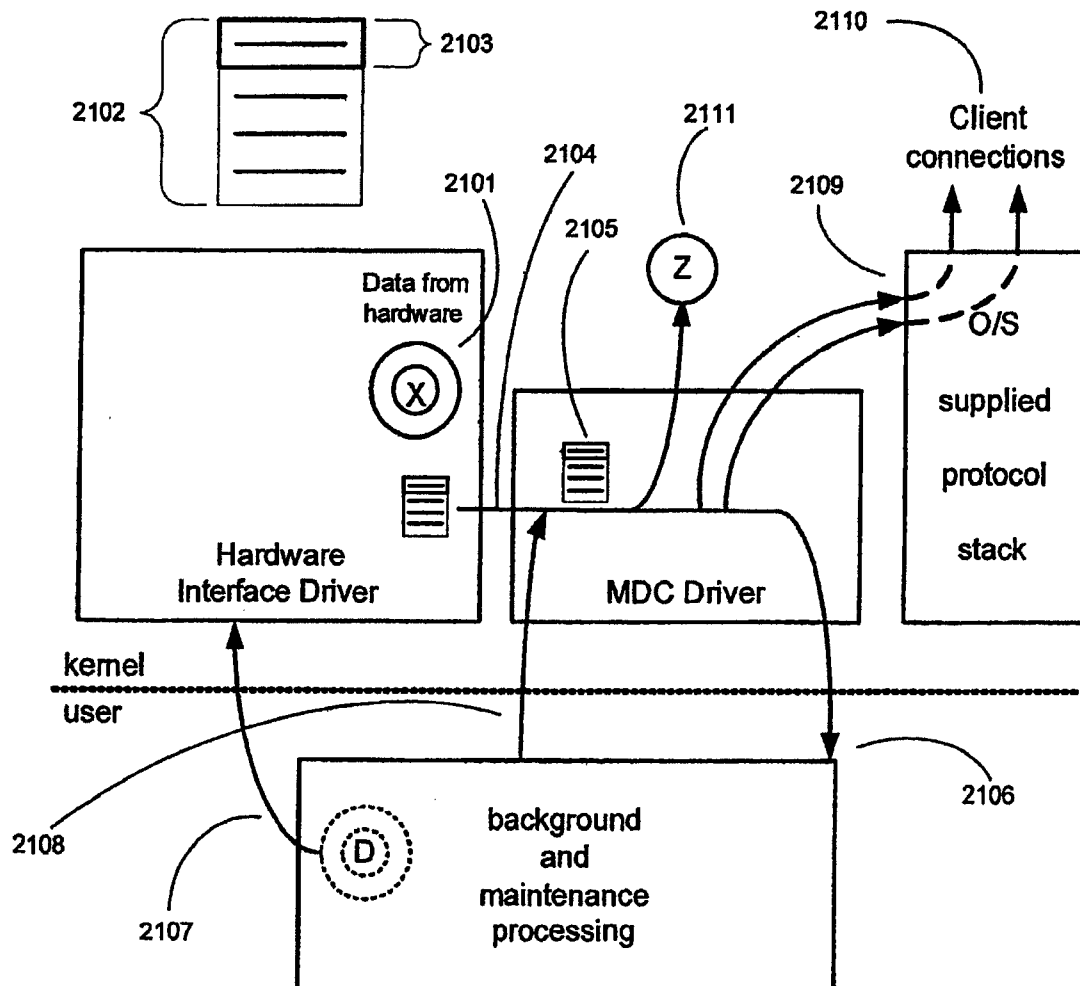


Figure 19

**Figure 20**

**Figure 21**

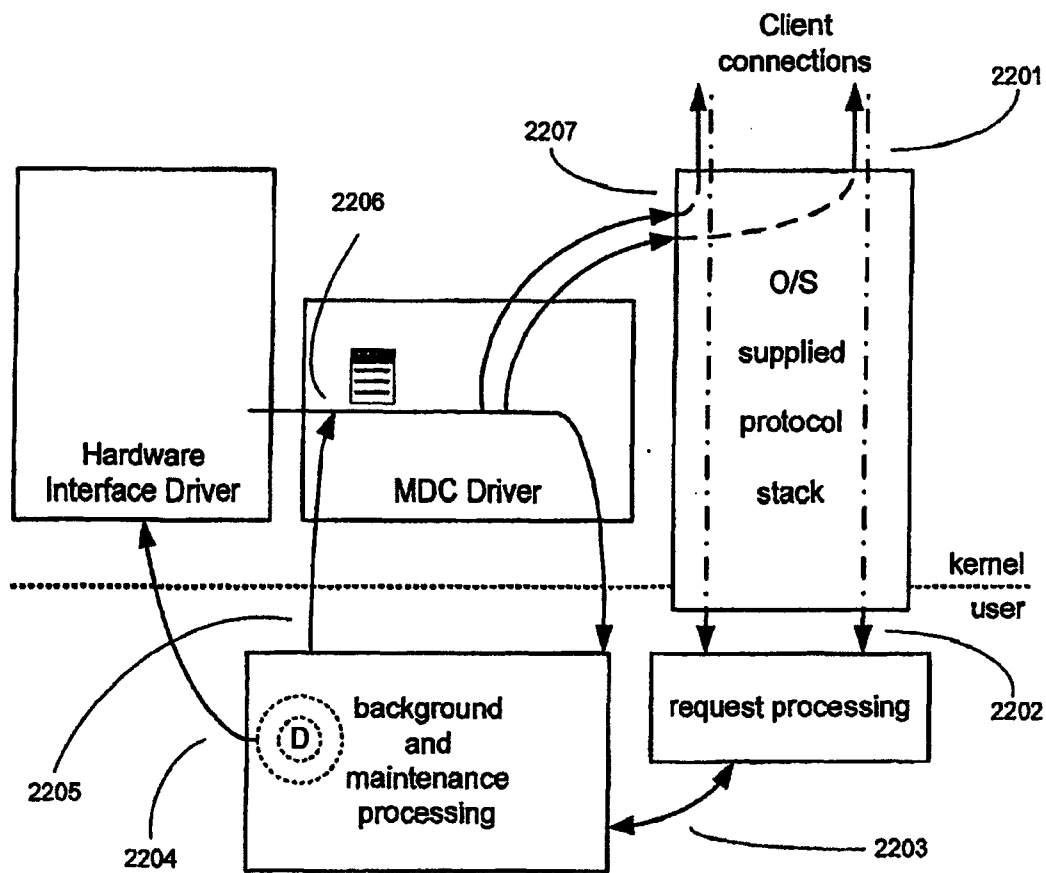


Figure 22

