Document made available under the Patent Cooperation Treaty (PCT)

International application number: PCT/US2009/067935

International filing date: 14 December 2009 (14.12.2009)

Document type: Certified copy of priority document

Document details: Country/Office: US
Number: 61/122,673
Filing date: 15 December 2008 (15.12.2008)

Date of receipt at the International Bureau: 07 January 2010 (07.01.2010)

Remark: Priority document submitted or transmitted to the International Bureau in compliance with Rule 17.1(a) or (b)

World Intellectual Property Organization (WIPO) - Geneva, Switzerland
Organisation Mondiale de la Propriété Intellectuelle (OMPI) - Genève, Suisse

225339

THE UNITED STATES OF AMERICA

TO ALL TO WHOM THESE PRESENTS SHALL COME:

UNITED STATES DEPARTMENT OF COMMERCE

United States Patent and Trademark Office

January 06, 2010

THIS IS TO CERTIFY THAT ANNEXED HERETO IS A TRUE COPY FROM THE RECORDS OF THE UNITED STATES PATENT AND TRADEMARK OFFICE OF THOSE PAPERS OF THE BELOW IDENTIFIED PATENT APPLICATION THAT MET THE REQUIREMENTS TO BE GRANTED A FILING DATE.

APPLICATION NUMBER: *61/122,673*
FILING DATE: *December 15, 2008*
RELATED PCT APPLICATION NUMBER: *PCT/US09/67935*

THE COUNTRY CODE AND NUMBER OF YOUR PRIORITY APPLICATION, TO BE USED FOR FILING ABROAD UNDER THE PARIS CONVENTION, IS *US61/122,673*

Certified by

Under Secretary of Commerce
for Intellectual Property
and Director of the United States
Patent and Trademark Office

PTO/SB/14 (07-07)
Approved for use through 06/30/2010. OMB 0651-0032
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE

Application Data Sheet 37 CFR 1.76	Attorney Docket Number	44826-80492
	Application Number	
Title of Invention	Method and Apparatus for High-Speed Processing of Financial Market Depth Data	

The application data sheet is part of the provisional or nonprovisional application for which it is being submitted. The following form contains the bibliographic data arranged in a format specified by the United States Patent and Trademark Office as outlined in 37 CFR 1.76.
This document may be completed electronically and submitted to the Office in electronic format using the Electronic Filing System (EFS) or the document may be printed and included in a paper filed application.

Secrecy Order 37 CFR 5.2

☐ Portions or all of the application associated with this Application Data Sheet may fall under a Secrecy Order pursuant to 37 CFR 5.2 (Paper filers only. Applications that fall under Secrecy Order may not be filed electronically.)

Applicant Information:

Applicant 1 Remove

Applicant Authority	◉ Inventor	○ Legal Representative under 35 U.S.C. 117	○ Party of Interest under 35 U.S.C. 118

Prefix	Given Name	Middle Name	Family Name	Suffix
	David	E.	Taylor	

Residence Information (Select One) ◉ US Residency ○ Non US Residency ○ Active US Military Service

City	St. Louis	State/Province	MO	Country of Residence [i]	US

Citizenship under 37 CFR 1.41(b) [i]	US

Mailing Address of Applicant:

Address 1	3448 Missouri Avenue
Address 2	

City	St. Louis	State/Province	MO
Postal Code	63118	Country [i]	US

Applicant 2 Remove

Applicant Authority	◉ Inventor	○ Legal Representative under 35 U.S.C. 117	○ Party of Interest under 35 U.S.C. 118

Prefix	Given Name	Middle Name	Family Name	Suffix
	Scott		Parsons	

Residence Information (Select One) ◉ US Residency ○ Non US Residency ○ Active US Military Service

City	St. Charles	State/Province	MO	Country of Residence [i]	US

Citizenship under 37 CFR 1.41(b) [i]	US

Mailing Address of Applicant:

Address 1	10 Beaufort Court
Address 2	

City	St. Charles	State/Province	MO
Postal Code	63301	Country [i]	US

Applicant 3 Remove

Applicant Authority	◉ Inventor	○ Legal Representative under 35 U.S.C. 117	○ Party of Interest under 35 U.S.C. 118

Prefix	Given Name	Middle Name	Family Name	Suffix
	Jeremy	Walter	Whatley	

Residence Information (Select One) ◉ US Residency ○ Non US Residency ○ Active US Military Service

City	Ballwin	State/Province	MO	Country of Residence [i]	US

PTO/SB/14 (07-07)
Approved for use through 06/30/2010. OMB 0651-0032
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE

Application Data Sheet 37 CFR 1.76	Attorney Docket Number	44826-80492
	Application Number	
Title of Invention	Method and Apparatus for High-Speed Processing of Financial Market Depth Data	

Citizenship under 37 CFR 1.41(b) i	US				
Mailing Address of Applicant:					
Address 1	1756 Stoney Terrace Drive				
Address 2					
City	Ballwin	**State/Province**	MO		
Postal Code	63021	**Country**i	US		

Applicant 4 Remove

Applicant Authority (•) Inventor	( ) Legal Representative under 35 U.S.C. 117	( ) Party of Interest under 35 U.S.C. 118			
Prefix	**Given Name**	**Middle Name**	**Family Name**	**Suffix**	
	Richard		Bradley		
Residence Information (Select One) (•) US Residency	( ) Non US Residency	( ) Active US Military Service			
City	St. Louis	**State/Province**	MO	**Country of Residence** i	US
Citizenship under 37 CFR 1.41(b) i					
Mailing Address of Applicant:					
Address 1	349 Marshall Avenue, Suite 100				
Address 2					
City	St. Louis	**State/Province**	MO		
Postal Code	63119	**Country**i	US		

Applicant 5 Remove

Applicant Authority (•) Inventor	( ) Legal Representative under 35 U.S.C. 117	( ) Party of Interest under 35 U.S.C. 118			
Prefix	**Given Name**	**Middle Name**	**Family Name**	**Suffix**	
	Kwame		Gyang		
Residence Information (Select One) (•) US Residency	( ) Non US Residency	( ) Active US Military Service			
City	St. Louis	**State/Province**	MO	**Country of Residence** i	US
Citizenship under 37 CFR 1.41(b) i	US				
Mailing Address of Applicant:					
Address 1	4355 Maryland Avenue, #405				
Address 2					
City	St. Louis	**State/Province**	MO		
Postal Code	63108	**Country**i	US		

Applicant 6 Remove

Applicant Authority (•) Inventor	( ) Legal Representative under 35 U.S.C. 117	( ) Party of Interest under 35 U.S.C. 118			
Prefix	**Given Name**	**Middle Name**	**Family Name**	**Suffix**	
	Michael		DeWulf		
Residence Information (Select One) (•) US Residency	( ) Non US Residency	( ) Active US Military Service			
City	St. Louis	**State/Province**	MO	**Country of Residence** i	US
Citizenship under 37 CFR 1.41(b) i					

PTO/SB/14 (07-07)
Approved for use through 06/30/2010. OMB 0651-0032
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE

Application Data Sheet 37 CFR 1.76	Attorney Docket Number	44826-80492
	Application Number	
Title of Invention	Method and Apparatus for High-Speed Processing of Financial Market Depth Data	

Mailing Address of Applicant:					
Address 1	349 Marshall Avenue, Suite 100				
Address 2					
City	St. Louis	State/Province	MO		
Postal Code	63119	Country[i]	US		

Applicant 7 — Remove

Applicant Authority	◉ Inventor	○ Legal Representative under 35 U.S.C. 117	○ Party of Interest under 35 U.S.C. 118

Prefix	Given Name	Middle Name	Family Name	Suffix
	Todd	Alan	Strader	

Residence Information (Select One)	◉ US Residency	○ Non US Residency	○ Active US Military Service		
City	St. Louis	State/Province	MO	Country of Residence [i]	US
Citizenship under 37 CFR 1.41(b) [i]	US				

Mailing Address of Applicant:					
Address 1	610 Lee Avenue				
Address 2					
City	St. Louis	State/Province	MO		
Postal Code	63119	Country[i]	US		

Applicant 8 — Remove

Applicant Authority	◉ Inventor	○ Legal Representative under 35 U.S.C. 117	○ Party of Interest under 35 U.S.C. 118

Prefix	Given Name	Middle Name	Family Name	Suffix
	Brandon	Parks	Thurmon	

Residence Information (Select One)	◉ US Residency	○ Non US Residency	○ Active US Military Service		
City	Webster Groves	State/Province	MO	Country of Residence [i]	US
Citizenship under 37 CFR 1.41(b) [i]	US				

Mailing Address of Applicant:					
Address 1	301 Newport Avenue				
Address 2					
City	Webster Groves	State/Province	MO		
Postal Code	63119	Country[i]	US		

Applicant 9 — Remove

Applicant Authority	◉ Inventor	○ Legal Representative under 35 U.S.C. 117	○ Party of Interest under 35 U.S.C. 118

Prefix	Given Name	Middle Name	Family Name	Suffix
	Nathaniel		McVicar	

Residence Information (Select One)	◉ US Residency	○ Non US Residency	○ Active US Military Service		
City	St. Louis	State/Province	MO	Country of Residence [i]	US
Citizenship under 37 CFR 1.41(b) [i]	US				

PTO/SB/14 (07-07)
Approved for use through 06/30/2010. OMB 0651-0032
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE

Application Data Sheet 37 CFR 1.76	Attorney Docket Number	44826-80492
	Application Number	
Title of Invention	Method and Apparatus for High-Speed Processing of Financial Market Depth Data	

Mailing Address of Applicant:			
Address 1	4605 Lindell Boulevard, Apt. 1403		
Address 2			
City	St. Louis	**State/Province**	MO
Postal Code	63108	**Country**[i]	US

Applicant 10 Remove

Applicant Authority (•) Inventor ( ) Legal Representative under 35 U.S.C. 117 ( ) Party of Interest under 35 U.S.C. 118

Prefix	**Given Name**	**Middle Name**	**Family Name**	**Suffix**
	Justin	Ryan	Thiel	

Residence Information (Select One) (•) US Residency ( ) Non US Residency ( ) Active US Military Service

City	Glen Carbon	**State/Province**	IL	**Country of Residence** [i]	US
Citizenship under 37 CFR 1.41(b) [i]	US				

Mailing Address of Applicant:			
Address 1	14 Cottonwood		
Address 2			
City	Glen Carbon	**State/Province**	IL
Postal Code	62034	**Country**[i]	US

All Inventors Must Be Listed - Additional Inventor Information blocks may be generated within this form by selecting the **Add** button. Add

Correspondence Information:

Enter either Customer Number or complete the Correspondence Information section below.
For further information see 37 CFR 1.33(a).

☐ **An Address is being provided for the correspondence Information of this application.**

Customer Number	21888		
Email Address	ipdocket@thompsoncoburn.com	Add Email	Remove Email
Email Address	bvolk@thompsoncoburn.com	Add Email	Remove Email

Application Information:

Title of the Invention	Method and Apparatus for High-Speed Processing of Financial Market Depth Data		
Attorney Docket Number	44826-80492	**Small Entity Status Claimed**	☒
Application Type	Provisional		
Subject Matter	Utility		
Suggested Class (if any)		**Sub Class (if any)**	
Suggested Technology Center (if any)			
Total Number of Drawing Sheets (if any)		**Suggested Figure for Publication (if any)**	

PTO/SB/14 (07-07)
Approved for use through 06/30/2010. OMB 0651-0032
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE

Application Data Sheet 37 CFR 1.76	Attorney Docket Number	44826-80492
	Application Number	
Title of Invention	Method and Apparatus for High-Speed Processing of Financial Market Depth Data	

Publication Information:

☐	Request Early Publication (Fee required at time of Request 37 CFR 1.219)
☐	**Request Not to Publish.** I hereby request that the attached application not be published under 35 U.S. C. 122(b) and certify that the invention disclosed in the attached application **has not and will not** be the subject of an application filed in another country, or under a multilateral international agreement, that requires publication at eighteen months after filing.

Representative Information:

Representative information should be provided for all practitioners having a power of attorney in the application. Providing this information in the Application Data Sheet does not constitute a power of attorney in the application (see 37 CFR 1.32).
Enter either Customer Number or complete the Representative Name section below. If both sections are completed the Customer Number will be used for the Representative Information during processing.

Please Select One:	◉ Customer Number	○ US Patent Practitioner	○ Limited Recognition (37 CFR 11.9)
Customer Number	21888		

Domestic Benefit/National Stage Information:

This section allows for the applicant to either claim benefit under 35 U.S.C. 119(e), 120, 121, or 365(c) or indicate National Stage entry from a PCT application. Providing this information in the application data sheet constitutes the specific reference required by 35 U.S.C. 119(e) or 120, and 37 CFR 1.78(a)(2) or CFR 1.78(a)(4), and need not otherwise be made part of the specification.

Prior Application Status			Remove
Application Number	Continuity Type	Prior Application Number	Filing Date (YYYY-MM-DD)

Additional Domestic Benefit/National Stage Data may be generated within this form by selecting the **Add** button. Add

Foreign Priority Information:

This section allows for the applicant to claim benefit of foreign priority and to identify any prior foreign application for which priority is not claimed. Providing this information in the application data sheet constitutes the claim for priority as required by 35 U.S.C. 119(b) and 37 CFR 1.55(a).

Remove

Application Number	Country ⁱ	Parent Filing Date (YYYY-MM-DD)	Priority Claimed
			○ Yes ◉ No

Additional Foreign Priority Data may be generated within this form by selecting the **Add** button. Add

Assignee Information:

Providing this information in the application data sheet does not substitute for compliance with any requirement of part 3 of Title 37 of the CFR to have an assignment recorded in the Office.

Assignee 1 Remove

PTO/SB/14 (07-07)
Approved for use through 06/30/2010. OMB 0651-0032
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE

Application Data Sheet 37 CFR 1.76	Attorney Docket Number	44826-80492
	Application Number	
Title of Invention	Method and Apparatus for High-Speed Processing of Financial Market Depth Data	

If the Assignee is an Organization check here. ☐				
Prefix	**Given Name**	Middle Name	**Family Name**	Suffix
Mailing Address Information:				
Address 1				
Address 2				
City		**State/Province**		
Country [i]		Postal Code		
Phone Number		Fax Number		
Email Address				
Additional Assignee Data may be generated within this form by selecting the **Add** button.			Add	

Signature:

A signature of the applicant or representative is required in accordance with 37 CFR 1.33 and 10.18. Please see 37 CFR 1.4(d) for the form of the signature.					
Signature	/Benjamin L. Volk, Jr./			Date (YYYY-MM-DD)	2008-12-15
First Name	Benjamin	Last Name	Volk	Registration Number	48017

This collection of information is required by 37 CFR 1.76. The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.14. This collection is estimated to take 23 minutes to complete, including gathering, preparing, and submitting the completed application data sheet form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, P.O. Box 1450, Alexandria, VA 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. **SEND TO: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.**

Privacy Act Statement

The Privacy Act of 1974 (P.L. 93-579) requires that you be given certain information in connection with your submission of the attached form related to a patent application or patent. Accordingly, pursuant to the requirements of the Act, please be advised that: (1) the general authority for the collection of this information is 35 U.S.C. 2(b)(2); (2) furnishing of the information solicited is voluntary; and (3) the principal purpose for which the information is used by the U.S. Patent and Trademark Office is to process and/or examine your submission related to a patent application or patent. If you do not furnish the requested information, the U.S. Patent and Trademark Office may not be able to process and/or examine your submission, which may result in termination of proceedings or abandonment of the application or expiration of the patent.

The information provided by you in this form will be subject to the following routine uses:

1. The information on this form will be treated confidentially to the extent allowed under the Freedom of Information Act (5 U.S.C. 552) and the Privacy Act (5 U.S.C. 552a). Records from this system of records may be disclosed to the Department of Justice to determine whether the Freedom of Information Act requires disclosure of these records.

2. A record from this system of records may be disclosed, as a routine use, in the course of presenting evidence to a court, magistrate, or administrative tribunal, including disclosures to opposing counsel in the course of settlement negotiations.

3. A record in this system of records may be disclosed, as a routine use, to a Member of Congress submitting a request involving an individual, to whom the record pertains, when the individual has requested assistance from the Member with respect to the subject matter of the record.

4. A record in this system of records may be disclosed, as a routine use, to a contractor of the Agency having need for the information in order to perform a contract. Recipients of information shall be required to comply with the requirements of the Privacy Act of 1974, as amended, pursuant to 5 U.S.C. 552a(m).

5. A record related to an International Application filed under the Patent Cooperation Treaty in this system of records may be disclosed, as a routine use, to the International Bureau of the World Intellectual Property Organization, pursuant to the Patent Cooperation Treaty.

6. A record in this system of records may be disclosed, as a routine use, to another federal agency for purposes of National Security review (35 U.S.C. 181) and for review pursuant to the Atomic Energy Act (42 U.S.C. 218(c)).

7. A record from this system of records may be disclosed, as a routine use, to the Administrator, General Services, or his/her designee, during an inspection of records conducted by GSA as part of that agency's responsibility to recommend improvements in records management practices and programs, under authority of 44 U.S.C. 2904 and 2906. Such disclosure shall be made in accordance with the GSA regulations governing inspection of records for this purpose, and any other relevant (i.e., GSA or Commerce) directive. Such disclosure shall not be used to make determinations about individuals.

8. A record from this system of records may be disclosed, as a routine use, to the public after either publication of the application pursuant to 35 U.S.C. 122(b) or issuance of a patent pursuant to 35 U.S.C. 151. Further, a record may be disclosed, subject to the limitations of 37 CFR 1.14, as a routine use, to the public if the record was filed in an application which became abandoned or in which the proceedings were terminated and which application is referenced by either a published application, an application open to public inspections or an issued patent.

9. A record from this system of records may be disclosed, as a routine use, to a Federal, State, or local law enforcement agency, if the USPTO becomes aware of a violation or potential violation of law or regulation.

WHAT IS CLAIMED IS:

1. A method of processing data, the method comprising:

streaming data through a processor, the data comprising a message, the message corresponding to a value for an element;

retrieving stored data based on the message, the retrieved data corresponding to a basket, the basket having the element as a member; and

computing a net value for the basket based on the retrieved data and the message;

wherein the retrieving step and the computing step are performed by the processor as the data streams through the processor.

2. The method of claim 1 wherein the processor comprises a coprocessor.

3. The method of claim 2 wherein the data comprises financial market data, and wherein the element comprises a financial instrument.

4. The method of claim 3 further comprising performing the retrieving step and the computing step in a pipelined manner.

5. The method of claim 4 wherein the computed net value comprises a net asset value (NAV) for the basket.

6. The method of claim 5 wherein the message comprises price information about the financial instrument, and wherein the computing step comprises computing the NAV using a delta calculation approach that is based on a contribution of the financial instrument price information to the NAV.

7. The method of claim 6 wherein the retrieving step further comprises retrieving a previous value relating to the basket and wherein the NAV computing step comprises adding the contribution to the retrieved previous value to thereby compute the NAV.

8. The method of claim 7 wherein the previous value comprises a previous NAV for the basket.

9. The method of claim 7 wherein the previous value comprises S, wherein S represents a value corresponding to a previous NAV for the basket multiplied by a divisor value for the basket, and wherein the NAV computing step further comprises dividing the sum of the contribution and S by the divisor value.

10. The method of claim 6 wherein the NAV computing step comprises simultaneously computing a plurality of different NAVs for the basket, each of the NAVs corresponding to a different NAV type.

11. The method of claim 10 wherein the plurality of different NAV types comprise a bid NAV, an ask NAV, and a last NAV.

12. The method of claim 11 wherein the plurality of different NAV types further comprise a bid-tick NAV and an ask+tick NAV.

13. The method of claim 6 wherein the financial instrument is a member of a plurality of baskets, wherein the retrieved data corresponds to the baskets, and wherein the computing step comprises computing the NAV for each of the baskets.

14. The method of claim 13 wherein the computing step comprises computing the contribution to the NAV for each basket based on the financial instrument price information and a weight value assigned to the financial instrument for that basket, and wherein the retrieving step further comprises retrieving the weight values for the baskets.

15. The method of claim 14 wherein the retrieving step further comprises retrieving a basket identifier for each of the baskets, each

basket identifier corresponding to a different one of the baskets and being associated with a different one of the weight values.

16. The method of claim 15 wherein the NAV computing step comprises calculating a price delta for the financial instrument, the method further comprising generating a plurality of delta events based on the retrieved data, each delta event comprising the calculated price delta, a retrieved basket identifier, and a retrieved weight value, and wherein the NAV computing step comprises computing the NAVs based on each delta event, and wherein the retrieving step, the generating step, and the NAV computing step are performed by the coprocessor in a pipelined manner as the financial market data streams through the coprocessor.

17. The method of claim 16 further comprising:

storing the basket identifiers and their associated weight values in a table as plurality of pairs.

18. The method of claim 17 wherein the storing step further comprises storing previous price information for the financial instrument in the table.

19. The method of claim 18 further comprising updating the previous price information in the table based on the message.

20. The method of claim 17 wherein the message comprises a financial instrument identifier for the financial instrument corresponding thereto, wherein the table comprises a first table, wherein the storing step further comprises storing a plurality of pointers in a second table, the pointers being indexed by a financial instrument identifier, and wherein the pairs in the first table are indexed by the pointers, and wherein the retrieving step further comprises (i) retrieving a pointer from the second table based on the message's

financial instrument identifier, and (ii) retrieving the pairs from the first table based on the retrieved pointer.

21. The method of claim 20 further comprising adding and deleting basket definitions within the tables in response to input from a client.

22. The method of claim 20 further comprising performing at least one of the group consisting of (i) adding a financial instrument to, (ii) modifying a weight for a financial instrument with respect to, and (iii) deleting a financial instrument from a basket by modifying the tables in response to input from a client.

23. The method of claim 13 wherein the message comprises a global exchange identifier, and wherein the retrieving step comprises retrieving data corresponding to a composite financial instrument and data corresponding to at least one regional financial instrument based on the global exchange identifier.

24. The method of claim 5 further comprising:

processing the NAV against a triggering condition to determine whether the NAV is to be reported to a client; and

generating a message for delivery to the client in response to a determination by the processing step that the NAV is to be reported to the client, the generated message comprising the NAV.

25. The method of claim 24 wherein the retrieving step, the computing step, and the processing step are performed by the coprocessor as the financial market data streams through the coprocessor.

26. The method of claim 25 further comprising performing the retrieving step, the computing step, and the processing step in a pipelined manner.

27. The method of claim 26 wherein the retrieving step, the computing step, the processing step, and the generating step are performed in a pipelined manner by the coprocessor as the financial market data streams through the coprocessor.

28. The method of claim 24 wherein the processing step comprises:

retrieving a reference value for the basket from a first table based on a basket identifier associated with the computed NAV;

determining a dissimilarity between the retrieved reference value and the computed NAV;

retrieving a trigger threshold value for the basket from a second table based on the basket identifier; and

comparing the dissimilarity with the retrieved trigger threshold value to thereby determine whether the NAV is to be reported to the client.

29. The method of claim 28 wherein the dissimilarity determining step comprises computing a difference between the retrieved reference value and the computed NAV.

30. The method of claim 24 further comprising defining a plurality of triggering conditions for a plurality of baskets in response to input from the client.

31. The method of claim 5 wherein the financial market data comprises a plurality of messages, the method further comprising:

filtering the messages prior to the retrieving step based on at least one filtering criterion such that the retrieving step is performed on the messages which pass the filtering step.

32. The method of claim 31 wherein the filtering step, the retrieving step, and the computing step are performed by the coprocessor as the financial market data streams through the coprocessor.

33. The method of claim 31 further comprising performing the filtering step, the retrieving step, and the computing step in a pipelined manner.

34. The method of claim 4 wherein the coprocessor comprises a reconfigurable logic device, and wherein the performing step comprises performing the retrieving step and the computing step with the reconfigurable logic device.

35. A system for processing data, the system comprising:

a processor configured to (i) receive a stream of data, the data comprising at least one message, the message corresponding to a value for an element, (ii) retrieve stored data based on the message, the retrieved data corresponding to a basket, the basket having the element as a member, and (iii) compute a net value for the basket based on the retrieved data and the message.

36. The system of claim 35 wherein the processor comprises a coprocessor.

37. The system of claim 36 wherein the processor further comprises a main processor in communication with the coprocessor, the processor being configured to deliver the stream of data to the coprocessor.

38. The system of claim 37 wherein the data comprises financial market data, and wherein the element comprises a financial instrument.

39. The system of claim 38 wherein the coprocessor comprises a basket association lookup module and an basket value updating module arranged in a pipelined manner, wherein the basket association lookup module is configured to retrieve the stored data based on the message, and wherein the basket value updating module is configured to compute the net value for the basket based on the retrieved data and the message.

40. The system of claim 39 wherein the computed net value comprises a net asset value (NAV) for the basket, wherein the message comprises price information about the financial instrument, and wherein the basket value updating module is further configured to compute the NAV using a delta calculation approach that is based on a contribution of the financial instrument price information to the NAV.

41. The system of claim 40 wherein the basket value updating module is further configured to simultaneously compute a plurality of different NAVs for the basket, each of the NAVs corresponding to a different NAV type.

42. The system of claim 40 wherein the financial instrument is a member of a plurality of baskets, wherein the retrieved data corresponds to the baskets, and wherein the basket value updating module is further configured to compute the NAV for each of the baskets.

43. The system of claim 42 wherein the basket value updating module is further configured to compute the contribution to the NAV for each basket based on the financial instrument price information and a weight value assigned to the financial instrument for that basket, and wherein the basket association lookup module is further configured to retrieve the weight values for the baskets.

44. The system of claim 43 wherein the basket association lookup module is further configured to retrieve a basket identifier for each of the baskets, each basket identifier corresponding to a different one of the baskets and being associated with a different one of the weight values.

45. The system of claim 44 wherein the message comprises a global exchange identifier, and wherein the basket association lookup module is further configured to retrieve data corresponding to a composite

financial instrument and data corresponding to at least one regional financial instrument based on the global exchange identifier.

46. The system of claim 40 wherein the coprocessor further comprises a price event trigger module in communication with the basket value updating module, the price event trigger module being configured to process the NAV against a triggering condition to determine whether the NAV is to be reported to a client.

47. The system of claim 46 wherein the basket association lookup module, the basket value updating module, and the price event trigger module are arranged in a pipelined manner.

48. The system of claim 47 wherein the coprocessor further comprises an event generator module in communication with the price event trigger module, wherein the event generator module is configured to generate a message for delivery to the client in response to a determination by the price event trigger module that the NAV is to be reported to the client, the generated message comprising the NAV.

49. The system of claim 48 wherein the basket association lookup module, the basket value updating module, and the price event trigger module are arranged in a pipelined manner.

50. The system of claim 46 wherein the price event trigger module is further configured to:

retrieve a reference value for the basket from a first table based on a basket identifier associated with the computed NAV;

determine a dissimilarity between the retrieved reference value and the computed NAV;

retrieve a trigger threshold value for the basket from a second table based on the basket identifier; and

compare the dissimilarity with the retrieved trigger threshold value to thereby determine whether the NAV is to be reported to the client.

51. The system of claim 40 wherein the financial market data comprises a plurality of messages, and wherein the coprocessor further comprises a message qualifier filter in communication with the basket association lookup module, the message qualifier filter being configured to filter the messages based on at least one filtering criterion such that the basket association lookup module receives the messages which pass the message qualifier filter.

52. The system of claim 51 wherein the message qualifier filter, the basket association lookup module, and the basket value updating module are arranged in pipelined manner.

53. The system of claim 39 wherein the coprocessor comprises a reconfigurable logic device, the reconfigurable logic device comprising the basket association lookup module and the basket value updating module.

54. The system of claim 38 further comprising another processor in communication with the processor, wherein the another processor is configured to execute a client application, and wherein the main processor is further configured to deliver data corresponding to the computed basket value to the client application.

55. A method of processing data, the method comprising:

performing a plurality of basket calculation operations on a bit stream using a processor as the bit stream streams through the processor to thereby compute a plurality of net basket values.

56. The method of claim 55 wherein the bit stream is representative of financial market data, and wherein the net basket values comprise a plurality of net asset values (NAVs) for financial instrument baskets.

57. The method of claim 56 wherein the processor comprises a coprocessor and a main processor, the main processor being configured to deliver the bit stream to the coprocessor, and the coprocessor being configured to perform the basket calculation operations.

58. The method of claim 57 wherein the performing step further comprises performing the basket calculation operations via a plurality of pipelined modules which are deployed on the coprocessor such that each pipeline module operates on different financial market data within the bit stream at the same time.

59. The method of claim 58 wherein the coprocessor comprises a reconfigurable logic device.

60. The method of claim 57 wherein the performing step further comprises:

processing the bit stream to determine which data should be retrieved from a plurality of tables;

retrieving the determined data from the tables; and

computing the basket NAVs based on the retrieved data and the bit stream.

61. An apparatus for computing a value pertaining to a net asset value (NAV) for a basket, the apparatus comprising:

a firmware pipeline configured to (i) receive a message comprising price information for a financial instrument and an identifier for the financial instrument, (ii) determine a plurality of baskets in which the financial instrument is a member based at least in part on the financial instrument identifier, and (iii) compute a delta contribution to a new NAV for each of the plurality of baskets.

62. The apparatus of claim 61 wherein the firmware pipeline is further configured to compute the new NAV for each of the plurality of baskets based on the computed delta contribution.

63. The apparatus of claim 61 wherein the firmware pipeline is further configured to (i) compute the delta contribution to the new NAV for each of the plurality of baskets according to the formula $\Delta j = \frac{w_j}{d}\left(T_j^{new} - T_j^{old}\right)$, wherein w_j represents a weight for the financial instrument in the basket, wherein d represents a divisor value for the basket, wherein T_j^{new} represents the price information, and wherein T_j^{old} represents old price information for the financial instrument, and (ii) compute the new NAV for each of the plurality of baskets according to the formula $NAV^{new} = NAV^{old} + \Delta j$, wherein NAV^{new} is the new NAV for a basket, wherein NAV^{old} is an old NAV for a basket.

64. The apparatus of claim 63 further comprising a plurality of tables in which the w_j values, the d values, the T_j^{old} values, and the NAV^{old} values are stored, and wherein the firmware pipeline is further configured to retrieve the w_j values, the d values, the T_j^{old} values, and the NAV^{old} values from the tables as the message streams through the pipeline.

65. The apparatus of claim 63 wherein the firmware pipeline is implemented on a coprocessor.

66. The apparatus of claim 63 wherein the coprocessor comprises a reconfigurable logic device.

67. The apparatus of claim 66 wherein the tables are stored within the reconfigurable logic device.

68. The apparatus of claim 61 wherein the firmware pipeline is further configured to (i) compute the delta contribution to the new NAV for each of the plurality of baskets according to the formula $w_j\left(T_j^{new} - T_j^{oid}\right)$, wherein w_j represents a weight for the financial instrument in the basket, wherein T_j^{new} represents the price information, and wherein T_j^{old} represents old price information for the financial instrument, (ii) retrieve a value S^{old} for each of the baskets, (iii) compute a sum of S^{old} and $w_j\left(T_j^{new} - T_j^{old}\right)$ for each of the baskets, (iv) retrieve a divisor value for the baskets, and (v) compute the new NAV for each of the plurality of baskets by dividing each sum by the retrieved divisor value.

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

processing a stream of financial market data using a coprocessor to compute a net asset value for a financial instrument basket as the financial market data streams through the coprocessor; and

detecting an arbitrage condition in connection with the basket based on the computed net asset value.

70. The method of claim 69 further comprising:

placing at least one trade order with at least one financial market in response to the detecting step to thereby take advantage of the detected arbitrage condition.

71. The method of claim 70 wherein the detecting step comprises detecting the arbitrage condition using the coprocessor as the financial market data streams through the coprocessor.

72. The method of claim 71 wherein the detecting step further comprises:

computing a difference between the computed net asset value and a reference value; and

comparing the computed difference with a trigger threshold to detect whether the arbitrage condition exists.

73. The method of claim 70 wherein the coprocessor comprises a reconfigurable logic device.

74. The method of claim 70 wherein the basket comprises an exchange traded fund (ETF).

75. A method for processing data, the method comprising:

receiving a stream of data, the data comprising a value for an element;

determining a basket which pertains to the element; and

calculating a value applicable to the determined basket based at least in part on the element value; and

wherein the determining step is performed by a first module in a pipeline; and

wherein the calculating step is performed by a second module in the pipeline.

76. The method of claim 75 wherein the data comprises financial market data, wherein the element comprises a financial instrument, wherein the element value comprises price information for the financial instrument, and wherein the basket comprises a financial instrument basket.

77. The method of claim 76 wherein the calculated value comprises a net asset value for the financial instrument basket, the method further comprising detecting an arbitrage condition in connection with the basket based on the calculated net asset value.

78. The method of claim 77 further comprising:

placing at least one trade order with at least one financial market in response to the detecting step to thereby take advantage of the detected arbitrage condition.

79. The method of claim 78 wherein the detecting step is performed by a third module in the pipeline.

80. The method of claim 79 wherein the detecting step further comprises:

computing a difference between the calculated net asset value and a reference value; and

comparing the computed difference with a trigger threshold to detect whether the arbitrage condition exists.

81. The method of claim 80 further comprising defining the reference value and the trigger threshold in response to client input.

82. The method of claim 78 wherein the calculating step comprises computing a delta contribution of the financial instrument to the determined basket's net asset value.

83. The method of claim 82 wherein the calculating step further comprises computing the determined basket's net asset value based on the computed delta contribution.

84. The method of claim 77 wherein the pipeline comprises a firmware pipeline.

85. The method of claim 84 wherein the firmware pipeline is deployed in reconfigurable logic.

86. The method of claim 77 wherein the basket comprises an exchange traded fund (ETF).

87. The method of claim 76 wherein the calculating step comprises computing a delta contribution of the financial instrument to a net asset value for the determined basket.

88. A method of administering a plurality of tables corresponding to a plurality of baskets, the method comprising:

storing a plurality of pointers in a first table, each pointer being indexed by an identifier for a financial instrument;

storing a plurality of header blocks in a second table, the header blocks being indexed by the pointers, each header block being associated with a financial instrument, each header block comprising a plurality of basket identifiers for the baskets and a plurality of weight values corresponding to the baskets, the basket identifiers and weight values being stored in the table as a plurality of pairs, each weight value defining a weight for a financial instrument in the basket corresponding to the basket identifier that is paired with that weight value.

89. The method of claim 88 wherein the header blocks further comprise price information for the financial instruments.

90. The method of claim 89 wherein each header block further comprises a count field which includes a count value indicative of how many pairs are present in that header block.

91. The method of claim 89 further comprising storing a plurality of extension blocks in the second table, wherein at least a plurality of header blocks further comprise an extension pointer field which includes a pointer to one of the extension blocks, the extension blocks comprising overflow pairs of basket identifiers and weight values.

92. The method of claim 89 wherein the first table further comprises a plurality of global exchange identifier indexes, each global

exchange identifier index corresponding to a different header block in the second table.

93. A method for computing a basket value, the method comprising:

receiving a delta event, the delta event comprising a basket identifier for a basket, a price delta with respect to a financial instrument, the financial instrument being a member of the basket, and a weight value that represents a weight of the financial instrument within the basket; and

retrieving a divisor value for the basket based on the basket identifier;

computing a net asset value (NAV) for the basket according to a delta calculation approach based on the delta event and the divisor value.

94. The method of claim 93 further comprising performing the method steps using a coprocessor.

95. An apparatus for computing a basket value, the apparatus comprising:

a processor configured to (i) receive a delta event, the delta event comprising a basket identifier for a basket, a price delta with respect to a financial instrument, the financial instrument being a member of the basket, and a weight value that represents a weight of the financial instrument within the basket, (ii) retrieve a divisor value for the basket based on the basket identifier, and (iii) compute a net asset value (NAV) for the basket according to a delta calculation approach based on the delta event and the divisor value.

96. A basket association method, the method comprising:

receiving a bit stream, the bit stream being representative of a message pertaining to a financial instrument, the message comprising a symbol identifier for the financial instrument and a global exchange identifier (GEID) for an exchange which pertains to the message;

performing a lookup in a first table based on the symbol ID to retrieve a record from the first table;

determining a pointer from the retrieved record;

determining an index value from the retrieved record based on the GEID;

performing a lookup in the second table based on the pointer and the index value to retrieve basket association data pertaining to each basket of which the financial instrument is a member; and

outputting a bit stream that is representative of the retrieved basket association data.

97. A basket association apparatus, the apparatus comprising:

a processor configured to (i) receive a message pertaining to a financial instrument, the message comprising a symbol identifier for the financial instrument and a global exchange identifier (GEID) for an exchange which pertains to the message, (ii) perform a lookup in a first table based on the symbol ID to retrieve a record from the first table, (iii) determine a pointer from the retrieved record, (iv) determine an index value from the retrieved record based on the GEID, (v) perform a lookup in the second table based on the pointer and the index value to retrieve basket association data pertaining to each basket of which the financial instrument is a member, and (vi) output the retrieved basket association data.

98. A method of reporting updated basket values to a client, the method comprising:

receiving a bit stream, the bit stream being representative of a basket event, the basket event comprising a basket identifier for a basket and an updated net asset value (NAV) for the basket;

retrieving a reference value for the basket based on the basket identifier;

determining a dissimilarity between the retrieved reference value and the updated NAV;

retrieving a trigger threshold value for the basket based on the basket identifier;

comparing the dissimilarity with the retrieved trigger threshold value to thereby determine whether the updated NAV is to be reported to the client; and

delivering a bit stream that is representative of the updated NAV to the client in response to a determination by the comparing step that the updated NAV should be reported to the client.

99. An apparatus for reporting updated basket values to a client, the apparatus comprising:

a processor configured to (i) receive a basket event, the basket event comprising a basket identifier for a basket and an updated net asset value (NAV) for the basket, (ii) retrieve a reference value for the basket based on the basket identifier, (iii) determine a dissimilarity between the retrieved reference value and the updated NAV, (iv) retrieve a trigger threshold value for the basket based on the basket identifier, (v) compare the dissimilarity with the retrieved trigger threshold value to thereby determine whether the updated NAV is to be reported to the client, and (vi) report the updated NAV to the client in response to a determination that the updated NAV should be reported to the client.

Method and Apparatus for High-Speed Processing of Financial Market Depth Data

The process of trading financial instruments may be viewed broadly as proceeding through a cycle as shown in Figure 1. At the top of the cycle is the exchange which is responsible for matching up offers to buy and sell financial instruments. Exchanges disseminate market information, such as the appearance of new buy/sell offers and trade transactions, as streams of events known as market data feeds. Trading firms receive market data from the various exchanges upon which they trade. Note that many traders manage diverse portfolios of instruments requiring them to monitor the state of multiple exchanges. Utilizing the data received from the exchange feeds, trading systems make trading decisions and issue buy/sell orders to the financial exchanges. Orders flow into the exchange where they are inserted into a sorted ``book'' of orders, triggering the publication of one or more events on the market data feeds.

Exchanges keep a sorted listing of limit orders for each financial instrument, known as an order book. A limit order is an offer to buy or sell a specified number of shares of a given financial instrument at a specified price. Orders are sorted based on price, size, and time according to exchange-specific rules. Many exchanges publish market data feeds that disseminate order book updates as order add, modify, and delete events. These feeds typically utilize one of two standard data models: full order depth or price aggregated depth. As shown in Figure 2, full order depth feeds contain events that allow recipients to construct order books that mirror the order books used by the exchange matching engines. This is useful for trading strategies that require knowledge of the state of specific orders in the market.

As shown in Figure 3, price aggregated depth feeds contain events that allow recipients to construct order books that report the distribution of liquidity (available shares) over prices for a given financial instrument.

Order book feeds are critical to electronic trading as they provide the fastest and deepest insight into market dynamics. In North America, the current set of order book feeds is primarily limited to equities, but several exchanges have announced plans to provide order book feeds for derivative instruments such as equity options. Given its explosive growth over the past several years, derivative instrument trading is responsible for the lion's share of market data traffic. The Options Price Reporting Authority (OPRA) feed is the most significant source of derivatives market data and it belongs to the class of feeds known as ``level 1''. This class of feeds reports quotes, trades, trade cancels and corrections, and a variety of summary events. For a given financial instrument, the highest buy price and lowest sell price comprise the ``best bid and offer''

(BBO) that are advertised as a quote. As the sorted listing changes due to order executions, modifications, or cancellations, the exchange publishes new quotes. When the best bid and offer prices match in the exchange's order book, the exchange executes a trade and advertises the transaction on its level 1 market data feed. Note that some amount of processing is required prior to publishing a quote or trade event, thus level 1 data feeds possess inherent latency relative to viewing ``raw'' order events on order book feeds.

In order to minimize total system latency, most electronic trading firms ingest market data feeds directly from the financial exchanges. While some loose standards are in place, most exchanges define unique protocols for disseminating their market data. This allows the exchanges to modify the protocols as needed to adjust to changes in market dynamics, regulatory controls, and the introduction of new asset classes. The ticker plant resides at the head of the platform and is responsible for the normalization, caching, filtering, and publishing of market data messages. A ticker plant typically provides a subscribe interface to a set of downstream trading applications. By normalizing data from disparate exchanges and asset classes, the ticker plant provides a consistent data model for trading applications. The subscribe interface allows each trading application to construct a custom normalized data feed containing only the information it requires. This is accomplished by performing subscription-based filtering at the ticker plant.

In traditional market data platforms, the ticker plant may perform some normalization tasks on order book feeds, but the task of constructing sorted and/or price-aggregated views of order books is typically pushed to downstream components in the market data platform. Doing so increases processing latency and the number of discrete systems required to process order book feeds. As an improvement over such an arrangement, an embodiment of the invention disclosed herein enables a ticker plant to perform order feed processing (normalization, price-aggregation, sorting) in an accelerated and integrated fashion, thereby increasing system throughput and decreasing processing latency.

We define two classes of book views: stream views (unsorted, non-cached) and summary views (sorted, cached). Stream views provide client applications with a normalized stream of updates for limit orders or aggregated price-points for the specified regional symbol, composite symbol, or feed source (exchange). Summary views provide multiple sorted views of the book, including composite views (a.k.a. “virtual order books”) that span multiple markets.

Stream views provide client applications with a normalized stream of updates for limit orders or aggregated price-points for the specified regional symbol, composite symbol, or feed source (exchange). Following the creation of a stream subscription, the ticker plant provides the

client application with a stream of normalized events containing limit order or price point updates. Stream subscriptions do not provide sorting and are intended to be employed by applications that construct their own book views or journals from the normalized event stream from one or more specified exchanges.

The order stream view provides the client application with a stream of normalized limit order update events for one or more specified regional symbols. The normalized events contain essential fields such as the type of update (add, modify, delete), the order price, order size, exchange timestamp, and order identifier (if provided by the exchange). The order exchange stream view provides the client application with a stream of normalized limit order update events for one or more specified exchanges or clusters of instruments within an exchange. The normalized events contain essential fields such as the type of update (add, modify, delete), the order price, order size, exchange timestamp, and order identifier (if provided by the exchange).

The price stream view provides the client application with a stream of normalized price level update events for one or more specified regional symbols. The normalized events contain essential fields such as the type of update (add, modify, delete), the aggregated price, order volume at the aggregated price, and order count at the aggregated price. The price exchange stream view provides the client application with a stream of normalized price level update events for one or more specified exchanges or clusters of instruments within an exchange. The normalized events contain essential fields such as the type of update (add, modify, delete), the aggregated price, order volume at the aggregated price, and order count at the aggregated price.

The aggregate stream view provides the client application with a stream of normalized price level update events for one or more specified composite symbols. The normalized events contain essential fields such as the type of update (add, modify, delete), the (virtual) aggregated price, (virtual) order volume at the aggregated price, and (virtual) order count at the aggregated price.

Summary views provide real-time liquidity insight. By offloading a significant amount of data processing from client applications, the ticker plant frees up client processing resources providing for more sophisticated trading applications that retain first mover advantage. The order summary view represents a first-order liquidity view of the raw limit order data disseminated by a single feed source. We define an order summary view to be a sorted listing of individual limit orders for a given financial instrument on a given exchange. The sort order is by price and then by time (or then by size for some exchanges). An example of an order summary view is shown in Figure 4.

The price summary view represents a second-order liquidity view of the raw limit order data disseminated by a single feed source. We define a price view to be a sorted listing of price levels for a given financial instrument on a given exchange where each price level represents an aggregation of same-priced orders from that exchange. The price level timestamp reports the timestamp of the most recent event at that price level from that exchange. An example of a price summary view is shown in Figure 5. Note that a price-aggregated view may be limited to a user-specified number of price points starting from the top of the book.

The spliced price summary view represents a second-order, pan-market liquidity view of the raw limit order data disseminated by multiple feed sources. We define a spliced price summary view to be a sorted listing of price levels for a given financial instrument across all contributing exchanges where each price level represents an aggregation of same-priced orders from a unique contributing exchange. The price level timestamp reports the timestamp of the most recent event at that price level for the specified exchange. An example of a spliced price summary view is shown in Figure 6. Note that a spliced price summary view may be limited to a user-specified number of price points starting from the top of the book.

The aggregate price summary view represents a third-order, pan-market liquidity view of the raw limit order data disseminated by multiple feed sources. We define an aggregate price summary view to be a sorted listing of price levels for a given financial instrument where each price level represents an aggregation of same-priced orders from all contributing exchanges. The price level timestamp reports the timestamp of the most recent event at that price level from any contributing exchange. An example of an aggregate price summary view is shown in Figure 7. Note that an aggregate price summary view may be limited to a user-specified number of price points starting from the top of the book.

As an embodiment of the invention, we describe a multi-stage pipeline for processing financial market depth (level 2) data that provides multiple book views to trading applications. A logical diagram of such a pipeline is shown in Figure 8.

The message parser, symbol mapping and message formatting stages are described in U.S. Patent Application Publication 2008/0243675, the entire disclosure of which is incorporated herein by reference. The 2008/0243675 publication is also included herewith as Appendix A. The symbol mapping stage resolves a unique symbol identifier for the base financial instrument and the associated market center for the event. Input events may contain a symbol field that uniquely identifies the base financial instrument. In this case, the symbol mapping stage performs a one-to-one translation from the input symbol field to the symbol identifier, which is preferably a minimally-sized binary tag that provides for efficient lookup of associated state

information for the financial instrument. An efficient hashing strategy for this symbol mapping technique is disclosed in the 2008/0243675 publication.

In the case that a symbol field is not included in the input event, the symbol mapping task is deferred to the next stage in the process: order normalization and price aggregation (OPA). In order to conserve bandwidth on data transmission links, several market centers minimize the size of event messages by sending "static" information regarding outstanding limit orders only once. Typically, the first message advertising the addition of a new limit order includes all associated information (such as the symbol, source identifier, price and size of the order, etc.) as well as a reference number for the order. Subsequent messages reporting modification or deletion of the order may only include a reference number that uniquely identifies the order. One of the roles of the OPA stage is to pass complete order information to downstream consumers of the messages. For limit order event messages, the OPA stage normalizes the messages by ensuring that all fields are present in a regular format and the message consistently communicates the market event. For example, a market center may choose to advertise all events using order modification events. It is the responsibility of the OPA stage to determine if the event results in the addition of a new order, modification of an existing order, or deletion of an existing order. In practice, market centers have disparate data models for communicating limit order events; the OPA stage ensures that a consistent data model is presented to downstream processing blocks and trading applications. All output events contain a consistent set of fields where each field has a well-defined type. For example, some exchanges may only send the order reference number in a delete event. The OPA fills in missing fields such as the price and size of the deleted order.

In order to resolve the symbol identifier for input events lacking a symbol field, the OPA stage must use another identifying field (such as an order reference number). In this case, the OPA stage performs a many-to-one translation to resolve the symbol identifier, as there may be many outstanding orders to buy or sell a given financial instrument. It is important to note that this many-to-one mapping requires maintaining a dynamic set of items that map to a given symbol identifier, items may be added, modified, or removed from the set at any time as orders enter and execute at market centers.

While there are several viable approaches to solve the order normalization problem, the preferred method is to maintain a record for every outstanding limit order advertised by the set of input market data feeds. An event reporting the creation of a new limit order must contain a symbol field, thus when the event arrives at the OPA stage it will contain the symbol identifier resolved by the symbol mapping stage. If the event is from a market center that uses order reference numbers to minimize subsequent message sizes, the event will also contain an order

reference number. The OPA maintains a map to the order record, where the record may contain the symbol identifier, order reference number, price, size, and other fields provided by the market center.

Preferably, the mapping is performed using hashing, where the hash key may be constructed from the order reference number, symbol identifier, or other uniquely identifying fields. The type of hash key is determined by the type of market center data feed. Upstream feed handlers that perform pre-normalization of the events set flags in the event that notify the OPA stage as to what type of protocol the exchange uses and what fields are available for constructing unique hash keys. There are a variety of hash functions that could be used by the OPA. In the preferred embodiment, the OPA employs H3 hash functions as discussed in the 2008/0243675 publication due to their efficiency and amenability to parallel hardware implementation. Hash collisions may be resolved in a number of ways. In the preferred embodiment, collisions are resolved via chaining, creating a linked list of entries that map to the same hash slot. A linked list is a simple data structure that allows memory to be dynamically allocated as the number of entries in the set changes.

Once the record is located, the OPA updates fields in the record and copies fields from the record to the message, filling in missing fields as necessary to normalize the output message. It is during this step that the OPA may modify the type of the message to be consistent with the result of the market event. For example, if the input message is a modify event that specifies that 100 shares should be subtracted from the order (due to a partial execution at the market center) and the outstanding order is for 100 shares, then the OPA will change the type of the message to a delete event, subject to market center rules. Note that market centers may dictate rules such as whether or not zero size orders may remain on an order book. In general, the OPA attempts to present the most descriptive and consistent view of the market data events.

In addition to normalizing order messages, the OPA may additionally perform price aggregation in order to support price aggregated views of the order book. Preferably, the OPA maintains an independent set of price point records. In this data structure, a record is maintained for each unique price point in an order book. At minimum, the set of price point records contain the price, volume (sum of order sizes at that price), and order count (total number of orders at that price). Order add events increase the volume and order count fields, order delete events decrease the volume and order count fields, etc. Price point records are created when an order event adds a new price point to the book. Likewise, price point records are deleted when on order event removes the only record with a given price point from the book.

Note that mapping an order event to a price point record is also a many-to-one mapping problem. Preferably, the set of price point records is maintained using a hash mapping, similar to the order records. In order to locate the price point record associated with an order event, the hash key is constructed from fields such as the symbol identifier, global exchange identifier, and price. Preferably, hash collisions are resolved using chaining as with the order record data structure. Other data structures may be suitable for maintaining the sets of order and price point records, but hash maps have the favorable property of constant time accesses (on average).

In the preferred embodiment, parallel engines update and maintain the order and price aggregation data structures in parallel. In one embodiment, the data structures are maintained in the same physical memory. In this case, the one or more order engines and one or more price engines interleave their accesses to memory, masking the memory access latency of the memory technology and maximizing throughput of the system. There are a variety of well-known techniques for memory interleaving. In one embodiment, an engine controller block utilizes a time-slot approach where each engine is granted access to memory at regular intervals. In another embodiment, a memory arbitration block schedules outstanding memory requests on the shared interface and notifies engines when their requests are fulfilled. Preferably, the memory technology is high-speed dynamic memory such as DDR3 SDRAM. In another embodiment, the order and price data structures are maintained in separate physical memories. As in the single memory architecture, multiple engines may interleave their accesses to memory in order to mask memory access latency and maximize throughput.

Figure 9 shows an example of how a single shared memory may be partitioned to support order normalization, regional price aggregation, and composite price aggregation. In this example, each hash table is allocated a portion of the memory space. Hash collisions are resolved by chaining, creating a linked list of entries that map to the same hash slot. Linked list entries for all three hash tables are dynamically allocated from a common memory space.

Figure 10 shows how the OPA data structures may be partitioned across multiple physical memories. In this particular example, the normalization data structure is stored in one memory, while the regional and composite price aggregation data structures are stored in a second memory. This architecture allows memory accesses to be performed in parallel.

As shown in the logical diagram of the pipeline in Figure 8, the output of the OPA may be transmitted directly to clients with appropriate stream view subscriptions. The Interest Entitlement Filter stage utilizes the mapped symbol index to resolve the set of interested and entitled clients for the given event, as described in the 2008/0243675 publication.

In order to support summary view subscriptions, the output of the OPA may be used to create one or more sorted data structures. In one embodiment of the invention, the output of the OPA is transmitted to a consuming system (client machine) where it is processed by an Application Programming Interface (API) associated with the ticker plant, as shown in connection with Figure 11. The API performs the sorting and presents the client application with a summary view of the data. The sorting task may be performed using a variety of techniques known in the art. In addition to these techniques, we disclose additional techniques below that may be used in the API or in the ticker plant pipeline. Performing the sorting process in the API distributes the sorting work to client machines with applications that require summary views.

In another embodiment of the invention, the ticker plant performs the sorting work in order to generate summary view events that are transmitted to consumers. In addition to passing stream events to appropriate subscribers, the stream events are also transmitted to the Sorted View Update (SVU) stage in the pipeline. There are a number of methods for performing sorting on order and price aggregated entries.

In one embodiment of the Sorted View Update stage, optimized insertion sorting engines independently maintain each side (bid and ask) of each order book independently. Since input order and price events only affect one side of the order book, accessing each side of the book independently reduces the potential number entries that must be accessed and provides for parallel access to the sides of the book. Each side of the book is maintained in sorted order in physical memory. The book is "anchored" at the bottom of the memory allocation for the book, i.e. the last entry is always stored at the last address in the memory allocation. As a consequence the location of the "top" of the book (the first entry) varies as the composition of the order book changes. In order to locate the top of the book, the SVU maintains a record that contains pointers to the top of the bid and ask side of the book, as well as other meta-data that may describe the book. The record may be located directly by using the symbol map index. Note that inserting an entry into the book moves entries above the insertion location up one memory location. Deleting an entry in the book moves entries above the insertion location down one memory location. While these operations may result in large numbers of memory copies, performance is typically good as the vast majority of order book transactions affect the top positions in the order book.

For regional order summary views, the SVU stage maintains a pair of bid and ask books for each symbol on each exchange upon which it trades. The entries in each book are from one exchange. For price summary views, the SVU stage maintains a pair of spliced bid and ask books for each symbol. The entries in each book are from every exchange upon which the symbol trades. Composite, spliced, and regional views of the price book may be synthesized from this

single spliced book. An order book engine in the SVU computes the desired views by aggregating multiple regional entries to produce composite entries and positions, and filters unwanted regional price entries to produce regional entries and positions. These computations are performed as the content of each book is streamed from memory through the engine. In order to minimize the memory bandwidth consumed for each update event, the engine requests chunks of memory that are typically smaller in size than the entire order book. Typically, a default memory chunk size is specified at system configuration time. Engines request the default chunk size in order to fetch the top of the book. If additional book entries must be accessed, the engines request the next memory chunk, picking up at the next address relative to the end of the previous chunk. In order to mask the latency of reading chunks of memory, processing, and requesting the next chunk of memory, multiple engines interleave their accesses to memory. As with the OPA engines, interleaving is accomplished by using a memory arbitration block that schedules memory transactions for multiple engines. Note that a time-slot memory controller may also be used. Engines may operate on unique symbols in parallel without affecting the correctness of the data.

In another embodiment of the Sorted View Update stage, each side of the book is organized as a hierarchical multi-way tree. The depth of a multi-way tree is dictated by the number of child branches leaving each node. A B+ tree is an example of a multi-way tree where all entries in the tree are stored at the same level, i.e. the leaf level. Typically, the height of a multi-way tree is minimized in order to minimize the number of nodes that must be visited in order to reach a leaf node in the tree, i.e. the desired entry. An example of a B+ tree is shown in Figure 12. Note that the leaf nodes may be organized as a linked list such that the entire contents of the tree may be accessed by navigating to the leftmost child and following the pointers to the next child node. This feature is exploited in the SVU stage to quickly produce a snapshot of the entire contents of the order book in order to initialize newly subscribed clients.

Figure 12 shows a simple example of a set of positive integers sorted using a B+ tree. Note that all of the stored integers are stored in the leaf nodes. As is well-known in the art, B+ tree nodes vary in size between a minimum and maximum size threshold that is dictated by the branching factor of the tree. Note that "next pointers" are stored in each leaf node to create a linked list of leaf nodes that may be quickly traversed to retrieve the entire sorted order. Internal tree nodes contain integers that dictate ranges stored the child nodes. For example, all integers less than or equal to 7 are stored in the left sub-tree of the root node.

The SVU stage utilizes hierarchical B+ trees to minimize the number of memory accesses required to update the various book views. As shown in Figure 13, the SVU stage maintains a

header record that contains pointers to the root of the composite and regional book trees. The root tree is the price aggregated tree. Each leaf of the price aggregated tree contains a pointer to a B+ tree that maintains a sort of the orders at the given price point. The SVU stage maintains a hierarchical B+ tree for the composite views and each regional view of the book. Note that each side of the book (bid and ask) corresponds to a hierarchical B+ tree. Note that hierarchical B+ trees may also be used in the embodiment where sorting is performed by the API on the client machine. Furthermore, note that a portion of the sorting may be offloaded to the API on the client machine. For example, construction of the spliced view of the book may be offloaded to the API by subscribing to the summary view of all regional books.

Similar to the insertion sorting engines in the previous embodiment, a parallel set of tree traversal engines operate in parallel and interleave their accesses to memory. Furthermore, the SVU stage may optionally cache recently accessed tree nodes in on-chip memory in order to further reduce memory read latency.

Another embodiment of the invention provides accelerated quote messages relative to a Level 1 feed published by an exchange. As shown in Figure 14, the pipeline is augmented to pass synthetic quote events from the SVU stage to the Value Cache Update stage that handles Level 1 traffic. When an order or price book update event modifies the top of the book (best bid or offer) (e.g., new price, size, or timestamp), the SVU stage generates a synthetic quote event that contains the new bid or ask price and aggregate order size at that price for the given regional and/or composite view of the book for the associated symbol. The Value Cache Update stage updates the associated Level 1 record for the symbol and transmits a Level 1 quote message to interested and entitled subscribers.

The aforementioned embodiments of the pipeline may be implemented in a variety of parallel processing technologies including: Field Programmable Gate Arrays (FPGAs), Chip Multi-Processors (CMPs), Application Specific Integrated Circuits (ASICs), Graphics Processing Units (GPUs), and multi-core superscalar processors. An exemplary platform in which the disclosed pipelines may be deployed is the coprocessor architecture disclosed in the 2008/0243675 publication and in pending U.S. patent application 12/013,302, entitled "Method and System for Low Latency Basket Calculation", filed January 11, 2008, the entire disclosure of which is incorporated herein by reference. A copy of this patent application is also attached as Appendix B. It should also be noted that the pipelines disclosed herein can be implemented not only in the ticker plants resident in trading firms but also in the ticker plants resident within the exchanges themselves to accelerate the exchanges' abilities to create and maintain their own order books.

Sample claims of various inventive aspects of the disclosed invention, not to be considered as exhaustive or limiting, all of which are fully described so as to satisfy the written description, enablement, and best mode requirement of the Patent Laws, are as follows:

1. A method for generating an order book view from financial market depth data, the method comprising:

maintaining a data structure representative of a plurality of order books for a plurality of financial instruments; and

hardware-accelerating a processing of a plurality of financial market depth data messages to update the order books within the data structure.

2. A system for generating an order book view from financial market depth data, the system comprising:

a memory for storing a data structure representative of a plurality of order books for a plurality of financial instruments; and

a coprocessor configured to process of a plurality of financial market depth data messages to update the order books within the data structure.

3. A computer-readable medium comprising:

a data structure for order books as disclosed herein, the data structure being resident on a computer-readable storage medium.

4. A method comprising:

processing financial market data as disclosed herein.

5. An apparatus comprising:

a ticker plant configured to process financial market data as disclosed herein.

US 20080243675A1

(19) **United States**

(12) **Patent Application Publication**

Parsons et al.

(10) **Pub. No.: US 2008/0243675 A1**

(43) **Pub. Date: Oct. 2, 2008**

(54) **HIGH SPEED PROCESSING OF FINANCIAL INFORMATION USING FPGA DEVICES**

(75) Inventors: **Scott Parsons**, St. Charles, MO (US); **David E. Taylor**, St. Louis, MO (US); **David Vincent Schuehler**, St. Louis, MO (US); **Mark A. Franklin**, St. Louis, MO (US); **Roger D. Chamberlain**, St. Louis, MO (US)

Correspondence Address:
THOMPSON COBURN, LLP
ONE US BANK PLAZA, SUITE 3500
ST LOUIS, MO 63101 (US)

(73) Assignee: **EXEGY INCORPORATED**, St. Louis, MO (US)

(21) Appl. No.: **11/765,306**

(22) Filed: **Jun. 19, 2007**

Related U.S. Application Data

(60) Provisional application No. 60/814,796, filed on Jun. 19, 2006.

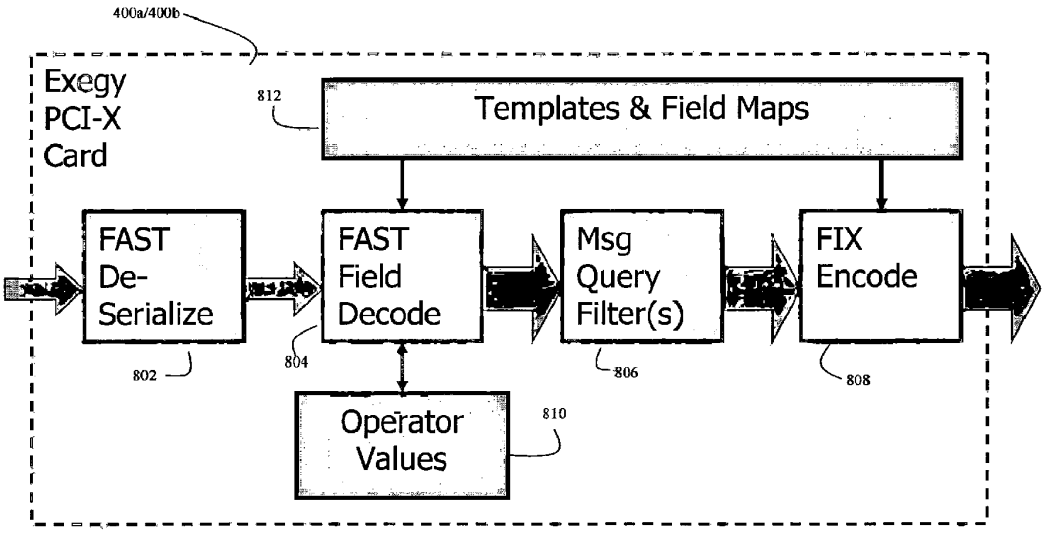
Publication Classification

(51) **Int. Cl.**
G06Q 40/00 (2006.01)

(52) **U.S. Cl.** **705/37**; 705/35

(57) **ABSTRACT**

Methods and systems for processing financial market data using reconfigurable logic are disclosed. Various functional operations to be performed on the financial market data can be implemented in firmware pipelines to accelerate the speed of processing. Also, a combination of software logic and firmware logic can be used to efficiently control and manage the high speed flow of financial market data to and from the reconfigurable logic.



APPENDIX A

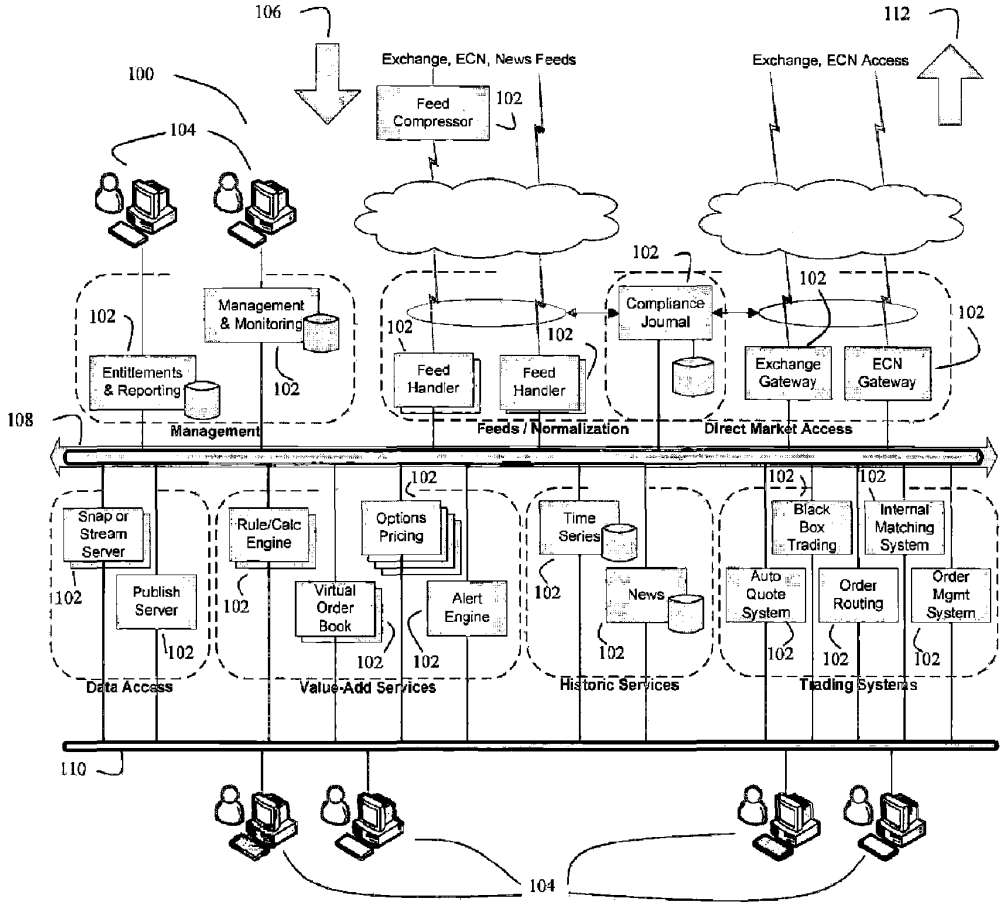


Figure 1
PRIOR ART

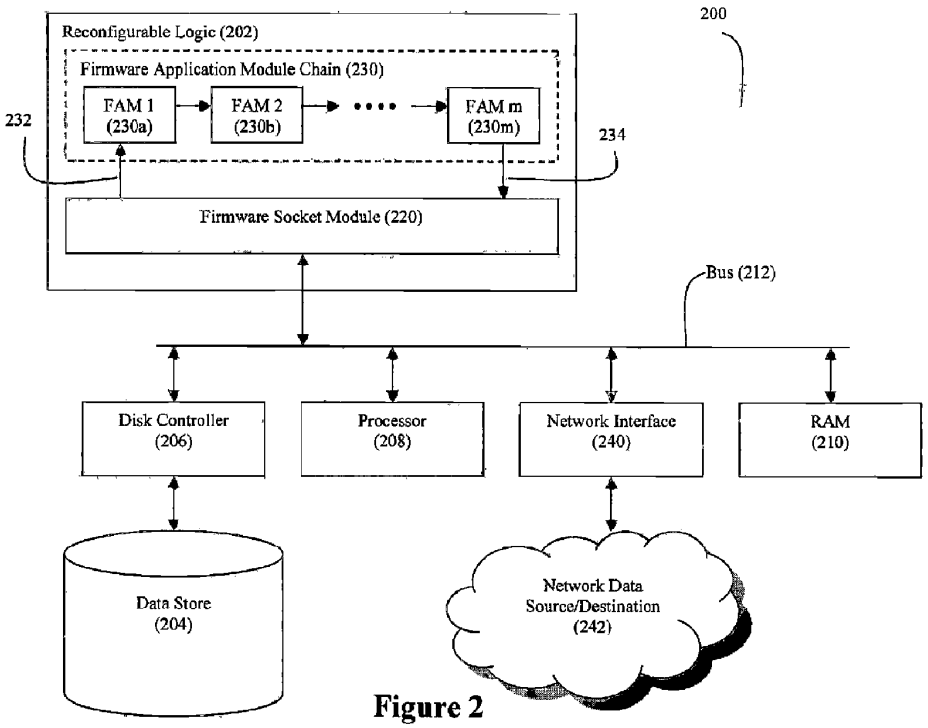


Figure 2

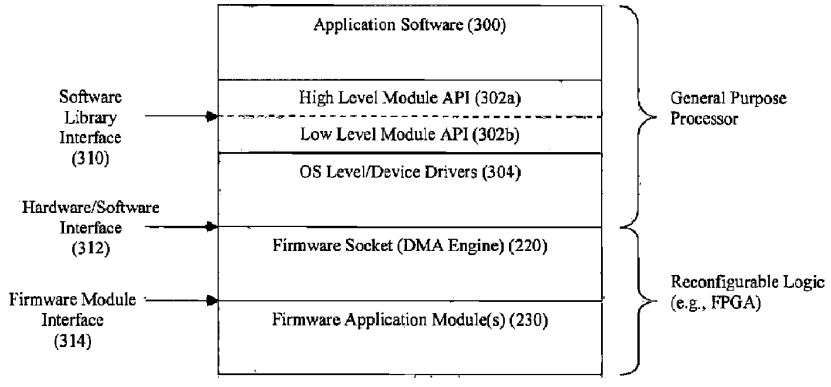


Figure 3

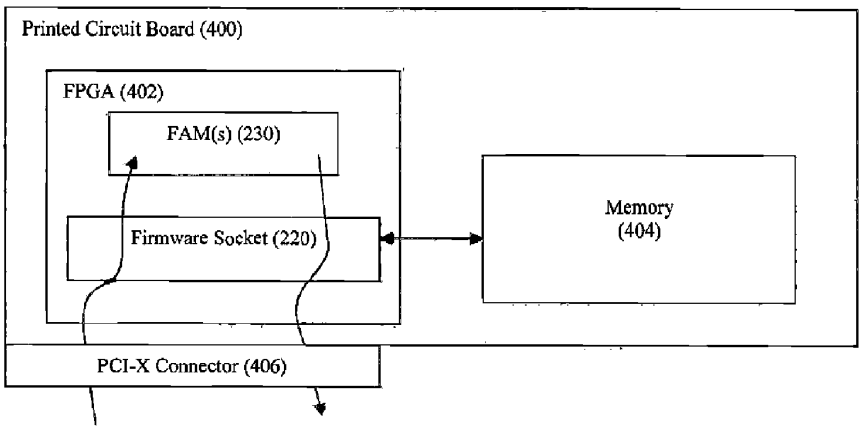


Figure 4(a)

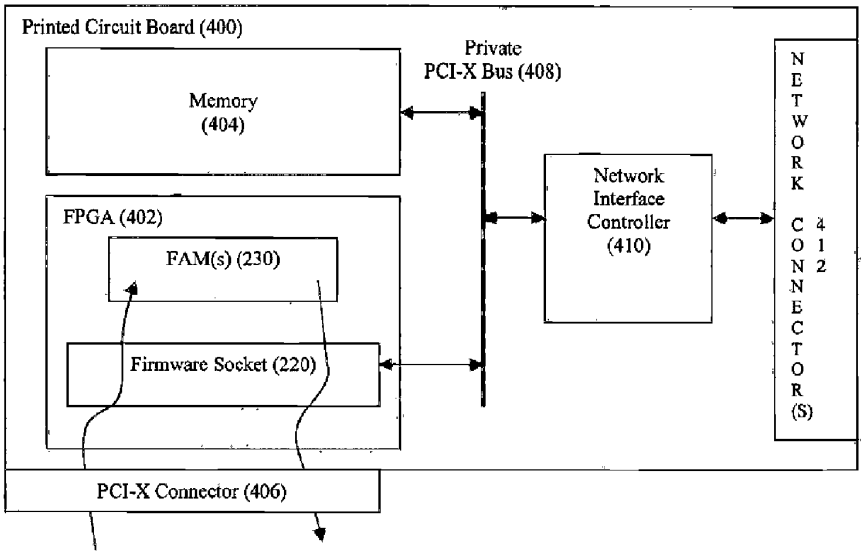


Figure 4(b)

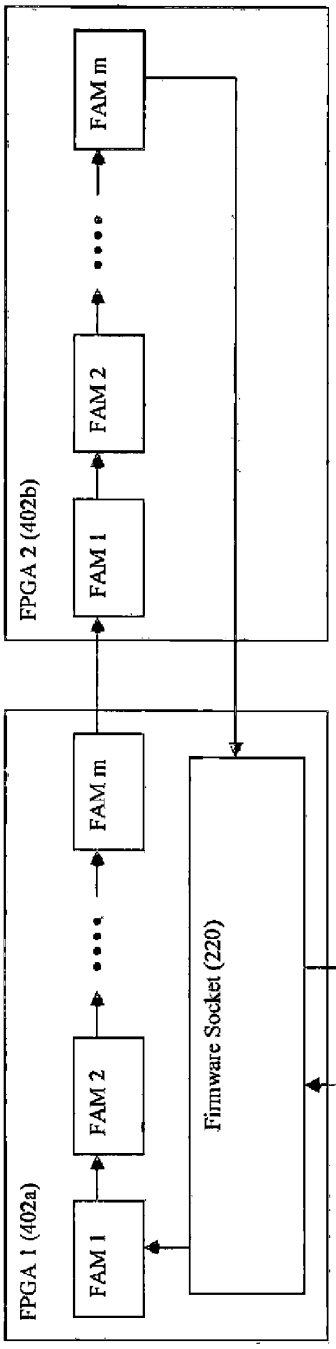

FPGA 1 (402a)
FAM 1
FAM 2
FAM m
Firmware Socket (220)
To/From Bus (212)
FPGA 2 (402b)
FAM 1
FAM 2
FAM m

Figure 5

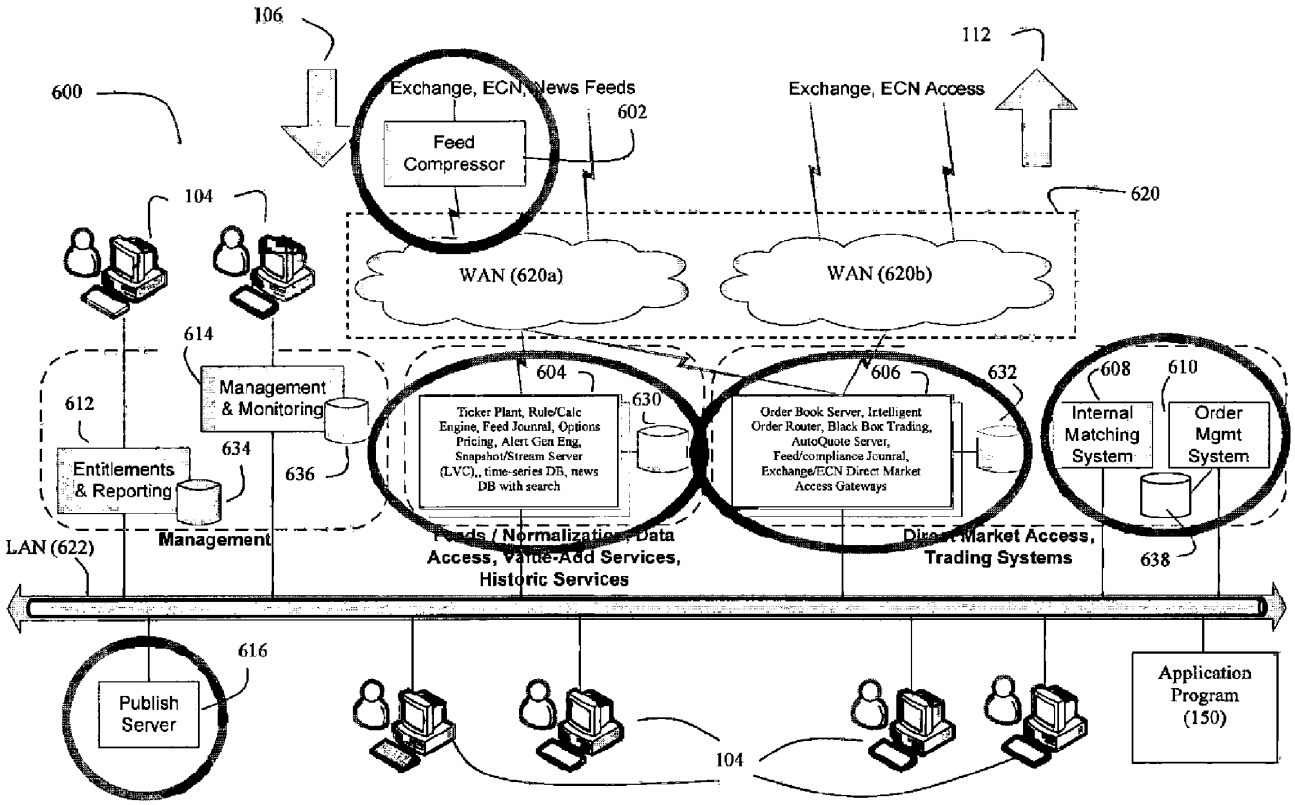

106
600
Exchange, ECN, News Feeds
602
Feed Compressor
112
Exchange, ECN Access
620
104
WAN (620a)
WAN (620b)
614
Management & Monitoring
612
Entitlements & Reporting
634
636
Management
604
Ticker Plant, Rule/Calc Engine, Feed Journal, Options Pricing, Alert Gen Eng, Snapshot/Stream Server (LVC),, time-series DB, news DB with search
630
Feeds / Normalization / Data Access, Value-Add Services, Historic Services
606
Order Book Server, Intelligent Order Router, Black Box Trading, AutoQuote Server, Feed/compliance Journal, Exchange/ECN Direct Market Access Gateways
632
Direct Market Access, Trading Systems
608
Internal Matching System
610
Order Mgmt System
638
LAN (622)
616
Publish Server
104
Application Program (150)

Figure 6

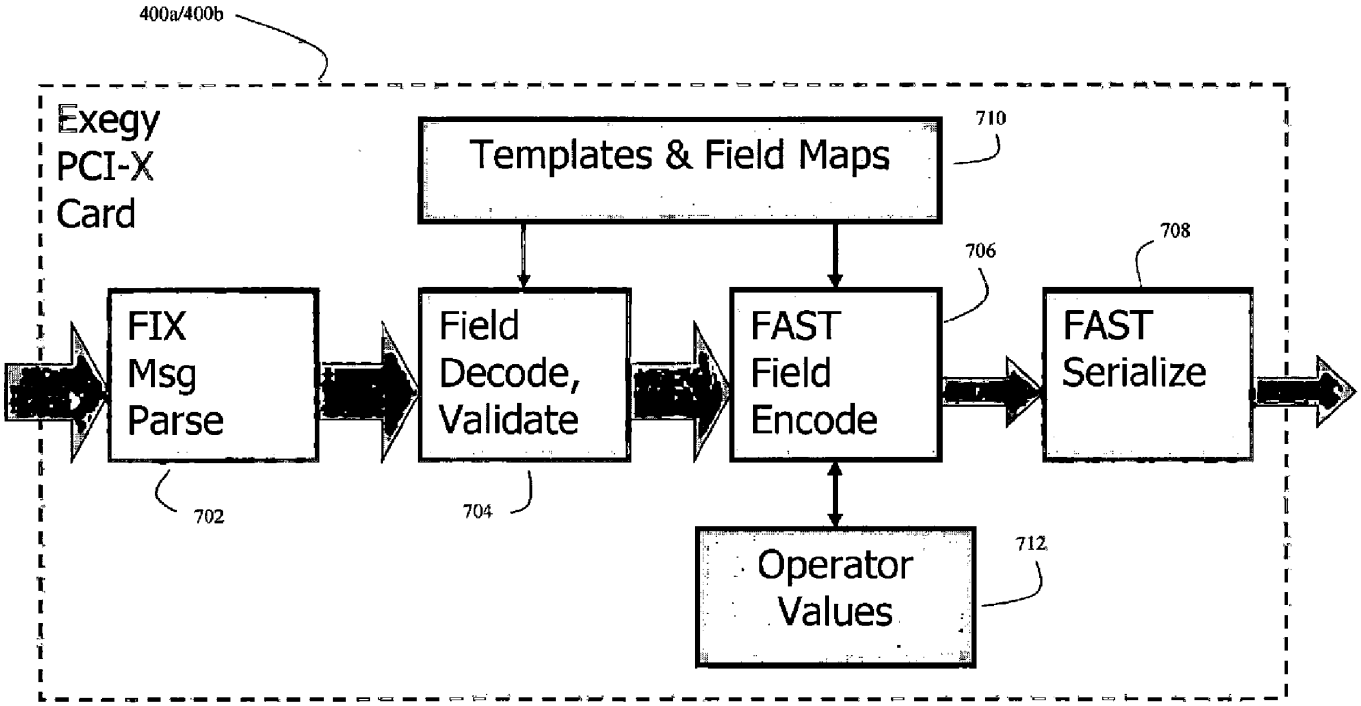

400a/400b
Exegy
PCI-X
Card
Templates & Field Maps
710
708
706
FIX
Msg
Parse
Field
Decode,
Validate
FAST
Field
Encode
FAST
Serialize
702
704
Operator
Values
712

Figure 7

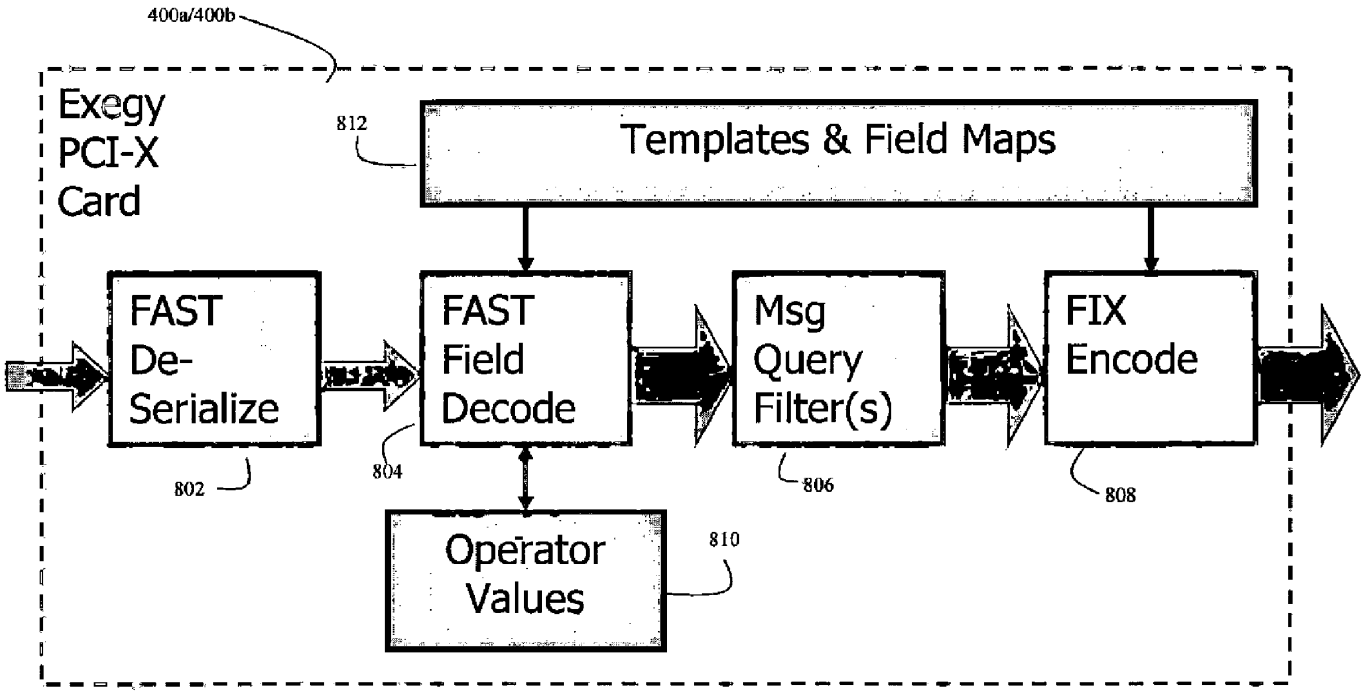

400a/400b
Exegy
PCI-X
Card
812
Templates & Field Maps
FAST
De-
Serialize
FAST
Field
Decode
Msg
Query
Filter(s)
FIX
Encode
802
804
806
808
Operator
Values
810

Figure 8

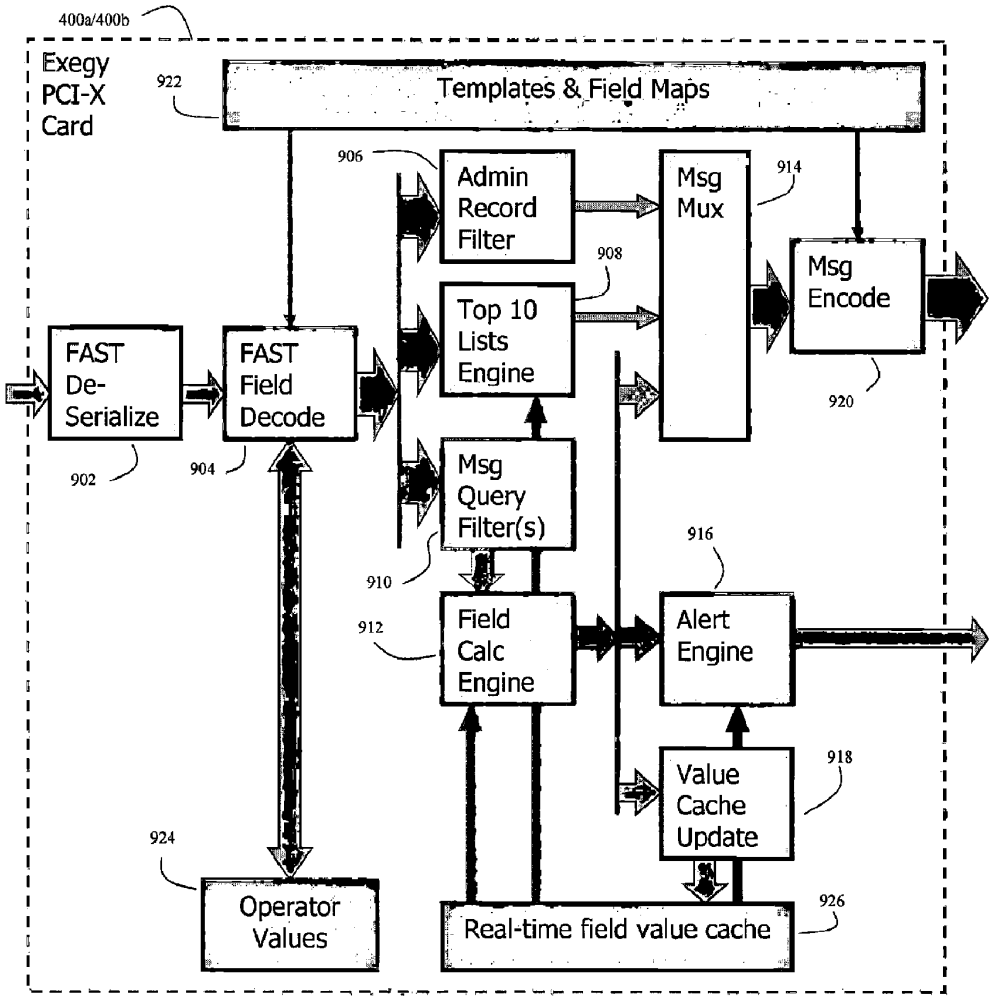

400a/400b
Exegy PCI-X Card
922
Templates & Field Maps
906
Admin Record Filter
914
Msg Mux
908
Top 10 Lists Engine
Msg Encode
920
FAST De-Serialize
902
FAST Field Decode
904
Msg Query Filter(s)
910
916
912
Field Calc Engine
Alert Engine
Value Cache Update
918
924
Operator Values
926
Real-time field value cache

Figure 9

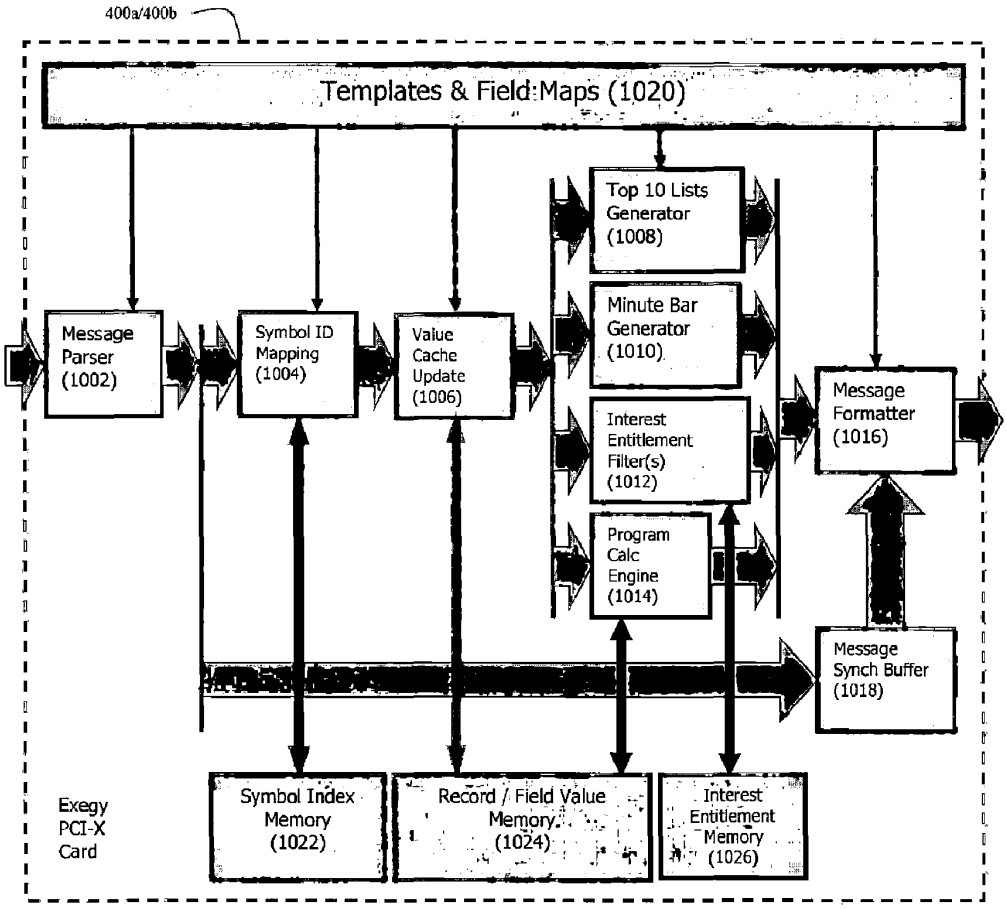

400a/400b
Templates & Field:Maps (1020)
Top 10 Lists Generator (1008)
Minute Bar Generator (1010)
Interest Entitlement Filter(s) (1012)
Program Calc Engine (1014)
Message Formatter (1016)
Message Synch Buffer (1018)
Message Parser (1002)
Symbol ID Mapping (1004)
Value Cache Update (1006)
Exegy PCI-X Card
Symbol Index Memory (1022)
Record / Field Value Memory (1024)
Interest Entitlement Memory (1026)

Figure 10

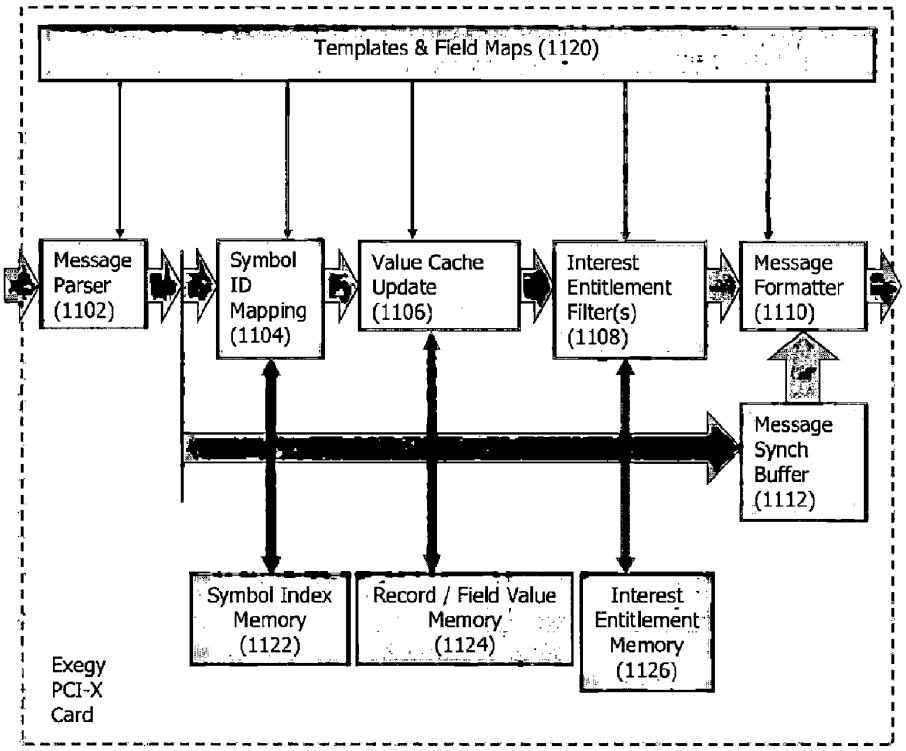

Templates & Field Maps (1120)
Message Parser (1102)
Symbol ID Mapping (1104)
Value Cache Update (1106)
Interest Entitlement Filter(s) (1108)
Message Formatter (1110)
Message Synch Buffer (1112)
Symbol Index Memory (1122)
Record / Field Value Memory (1124)
Interest Entitlement Memory (1126)
Exegy PCI-X Card

Figure 11

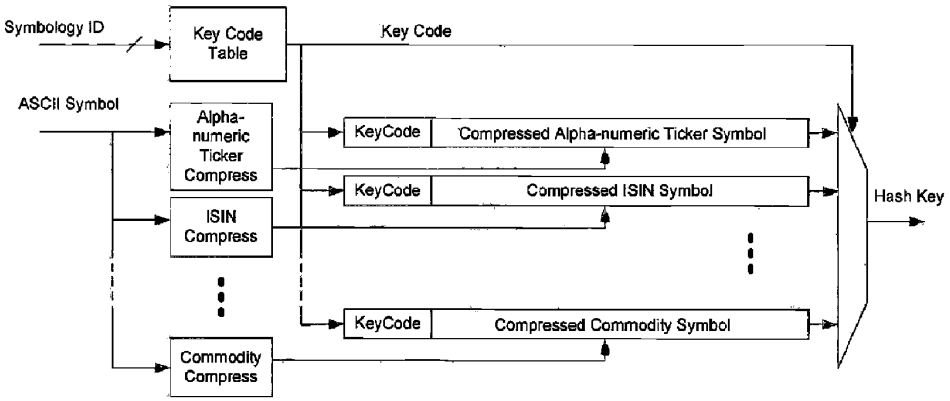


Figure 12

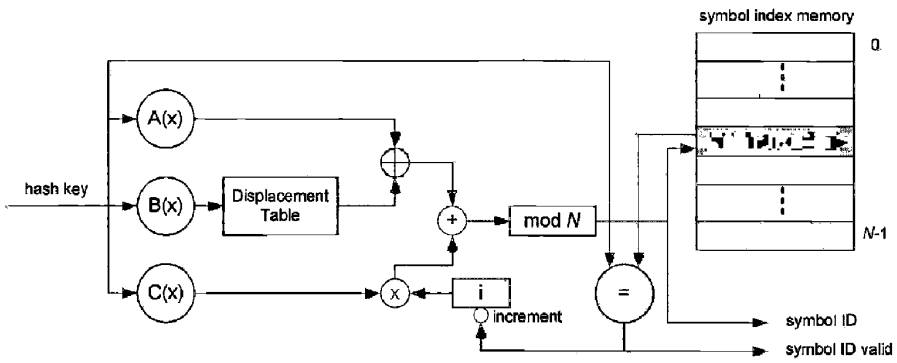


Figure 13

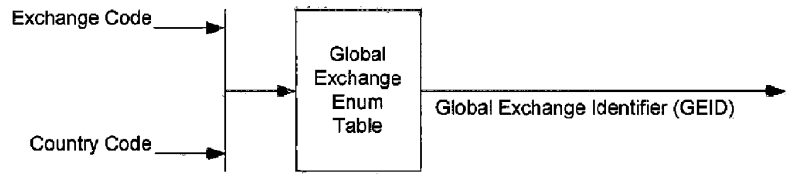


Figure 14

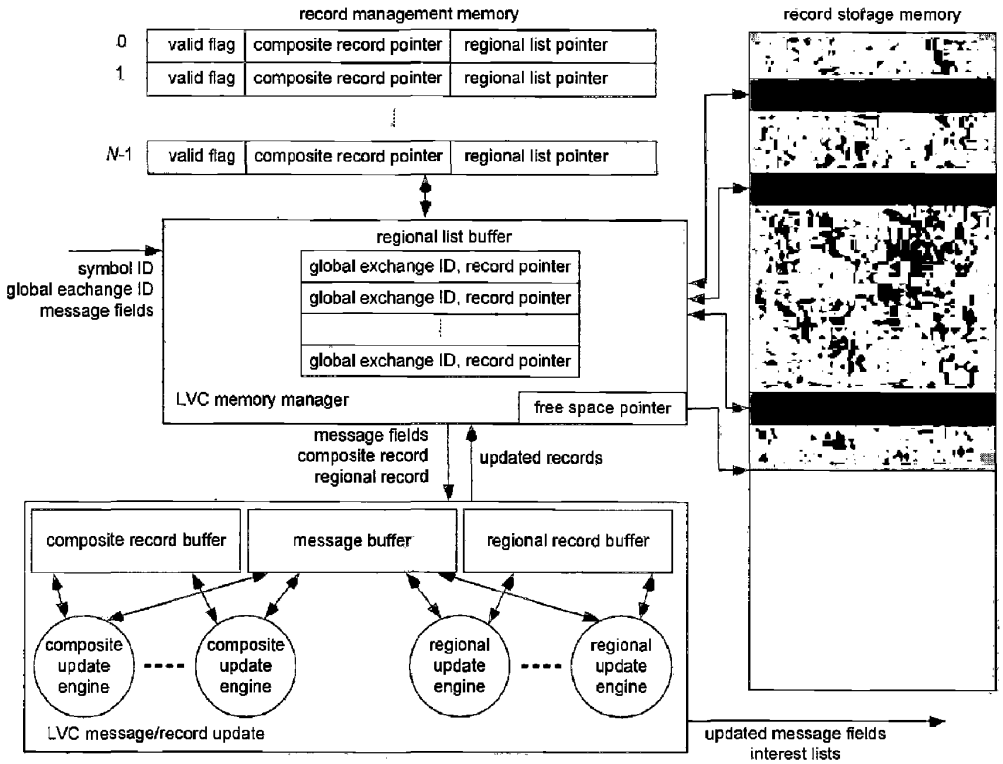

record management memory
record storage memory
0
valid flag
composite record pointer
regional list pointer
1
valid flag
composite record pointer
regional list pointer
N-1
valid flag
composite record pointer
regional list pointer
regional list buffer
global exchange ID, record pointer
global exchange ID, record pointer
global exchange ID, record pointer
symbol ID
global eachange ID
message fields
LVC memory manager
free space pointer
message fields
composite record
regional record
updated records
composite record buffer
message buffer
regional record buffer
composite update engine
composite update engine
regional update engine
regional update engine
LVC message/record update
updated message fields
interest lists

Figure 15(a)

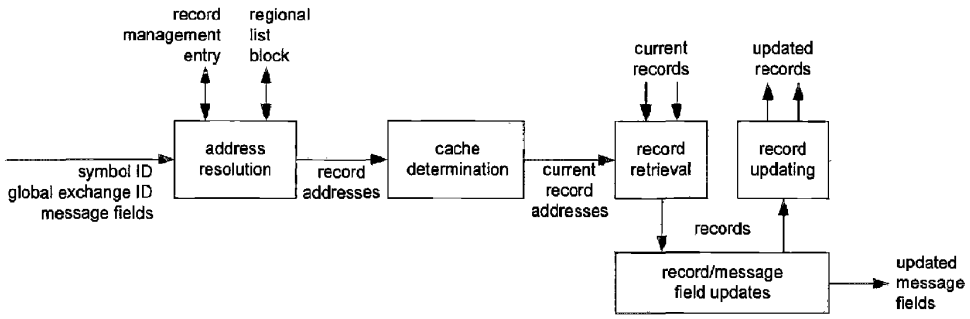

record management entry
regional list block
address resolution
symbol ID
global exchange ID
message fields
record addresses
cache determination
current record addresses
current records
record retrieval
updated records
record updating
records
record/message field updates
updated message fields

Figure 15(b)

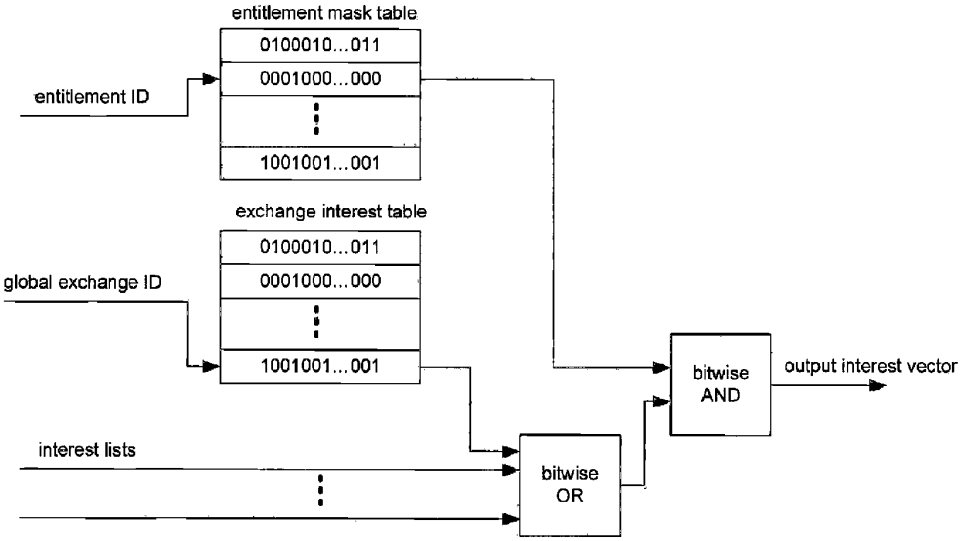

entitlement mask table
0100010...011
0001000...000
1001001...001
entitlement ID
exchange interest table
0100010...011
0001000...000
1001001...001
global exchange ID
bitwise AND
output interest vector
interest lists
bitwise OR

Figure 16

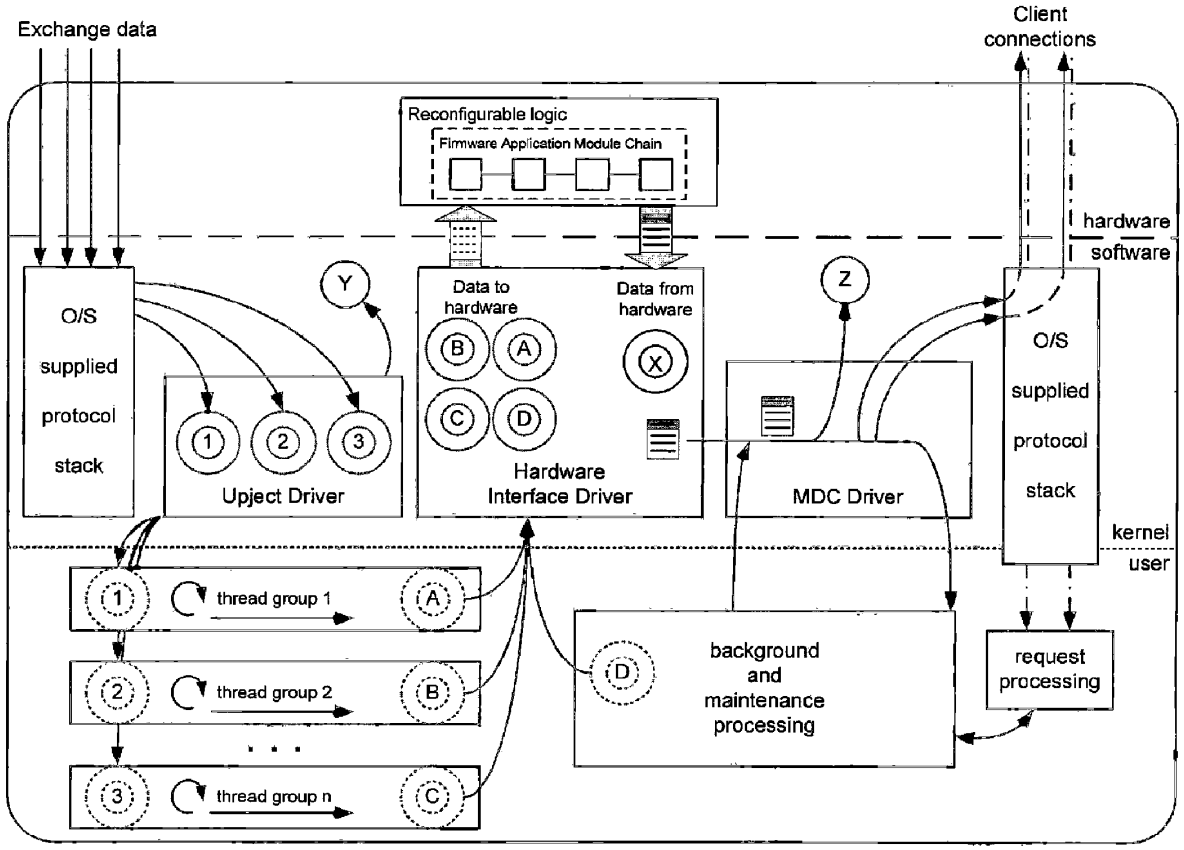

Exchange data
Client connections
Reconfigurable logic
Firmware Application Module Chain
hardware
software
O/S supplied protocol stack
Y
Data to hardware
Data from hardware
B
A
C
D
X
Z
1
2
3
Upject Driver
Hardware Interface Driver
MDC Driver
O/S supplied protocol stack
kernel
user
thread group 1
thread group 2
thread group n
background and maintenance processing
request processing

Figure 17

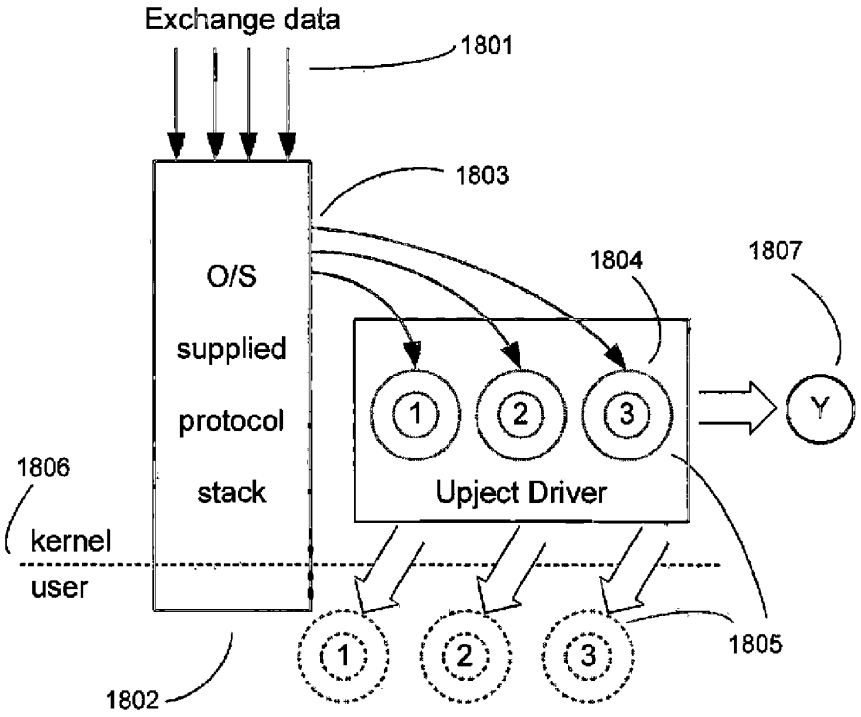

Exchange data
1801
1803
O/S
supplied
protocol
stack
1804
1807
1
2
3
Y
Upject Driver
1806
kernel
user
1
2
3
1805
1802

Figure 18

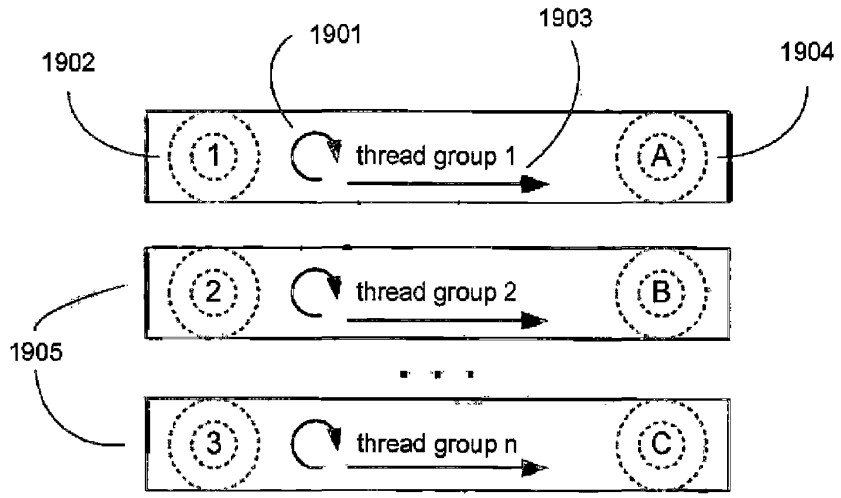

1902
1901
1903
1904
1
thread group 1
A
2
thread group 2
B
1905
3
thread group n
C

Figure 19

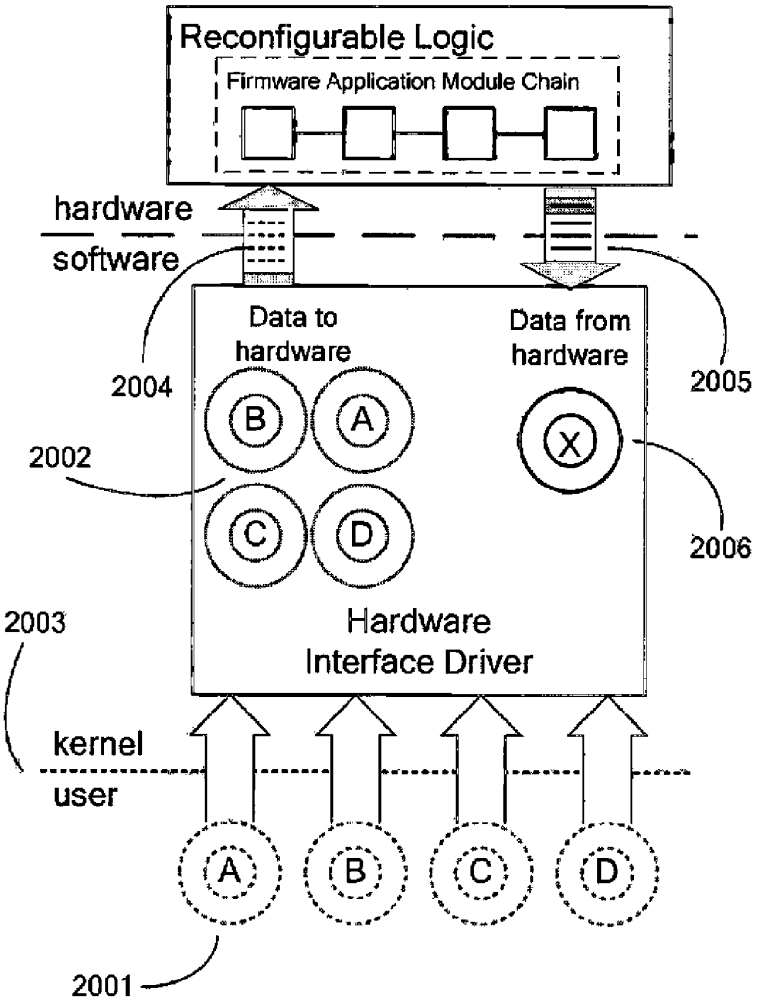

Reconfigurable Logic
Firmware Application Module Chain
hardware
software
Data to hardware
Data from hardware
2004
2005
2002
B
A
X
C
D
2006
2003
Hardware Interface Driver
kernel
user
A
B
C
D
2001

Figure 20

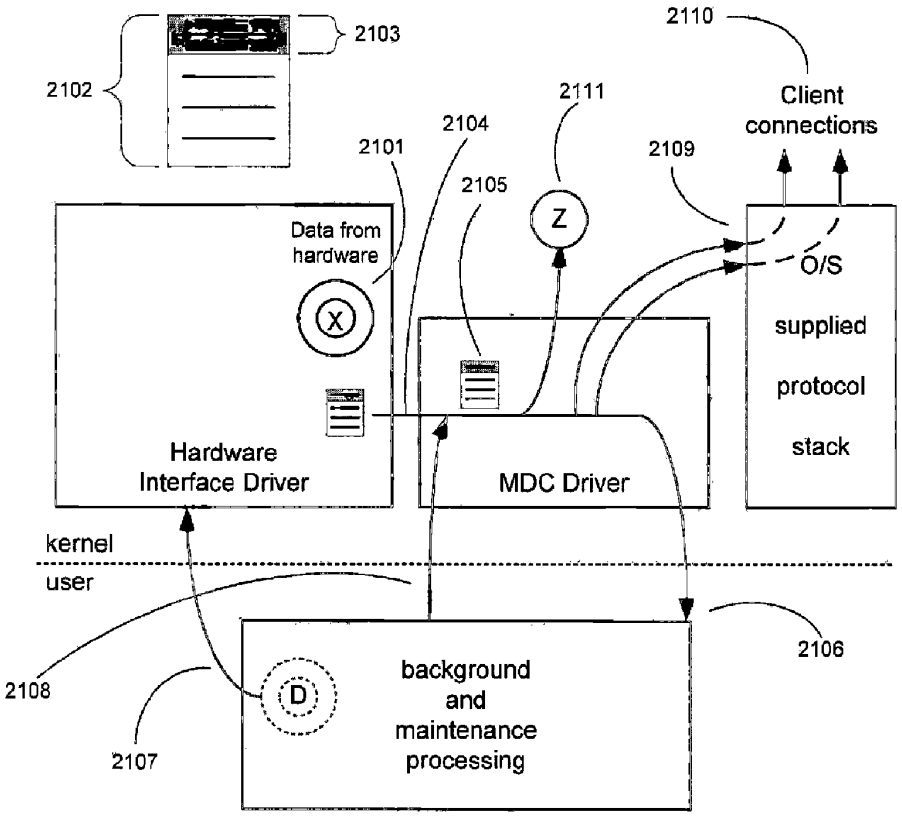

2103
2102
2110
Client connections
2111
2104
2101
2109
2105
Z
Data from hardware
X
O/S
supplied protocol stack
Hardware Interface Driver
MDC Driver
kernel
user
2106
2108
D
background and maintenance processing
2107

Figure 21

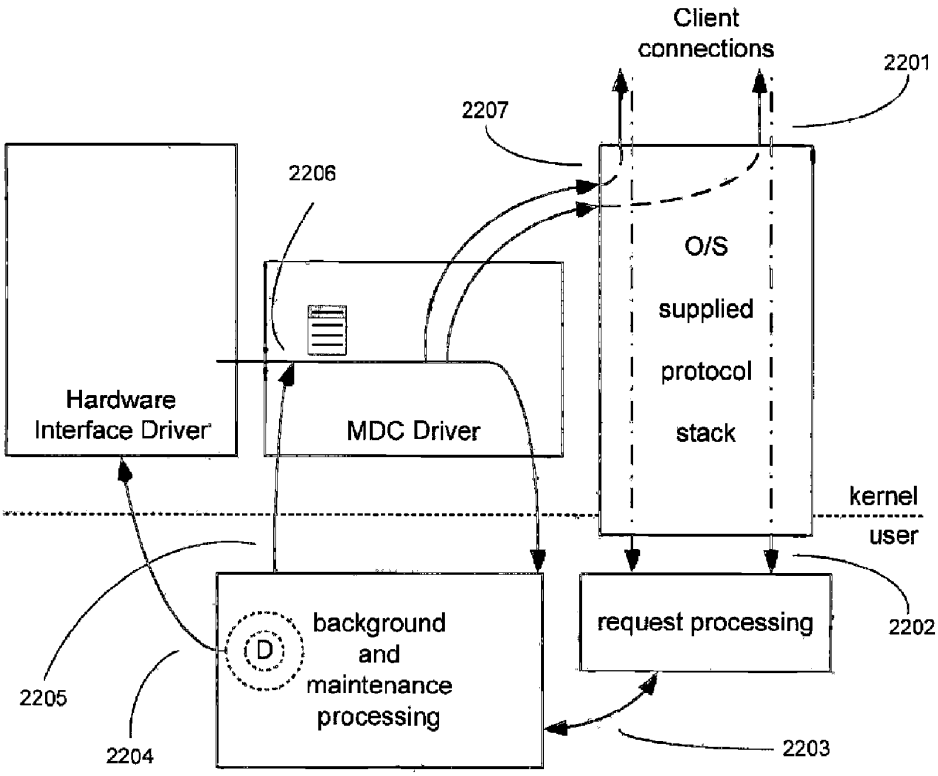

Client
connections
2201
2207
2206
O/S
supplied
protocol
stack
Hardware
Interface Driver
MDC Driver
kernel
user
request processing
2202
D
background
and
maintenance
processing
2205
2204
2203

Figure 22

HIGH SPEED PROCESSING OF FINANCIAL INFORMATION USING FPGA DEVICES

CROSS-REFERENCE AND PRIORITY CLAIM TO RELATED PATENT APPLICATIONS

[0001] This application claims priority to provisional patent application 60/814,796, filed Jun. 19, 2006, and entitled "High Speed Processing of Financial Information Using FPGA Devices", the entire disclosure of which is incorporated herein by reference.

[0002] This patent application is related to the following patent applications: U.S. patent application Ser. No. 09/545,472 (filed Apr. 7, 2000, and entitled "Associative Database Scanning and Information Retrieval", now U.S. Pat. No. 6,711,558), U.S. patent application Ser. No. 10/153,151 (filed May 21, 2002, and entitled "Associative Database Scanning and Information Retrieval using FPGA Devices", now U.S. Pat. No. 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 application WO 06/096324 (filed Feb. 22, 2006, entitled "Method and Apparatus for Performing Biosequence Similarity Searching"), U.S. patent application Ser. No. 11/293,619 (filed Dec. 2, 2005, entitled "Method and Device for High Performance Regular Expression Pattern Matching", and published as 2007/0130140), U.S. patent application Ser. No. 11/339,892 (filed Jan. 26, 2006, and entitled "Firmware Socket Module for FPGA-Based Pipeline Processing"), U.S. patent application Ser. No. 11/381,214 (filed May 2, 2006, and entitled "Method and Apparatus for Approximate Pattern Matching"), U.S. patent application Ser. No. 11/561,615 (filed Nov. 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 Ser. No. 11/760,211 (filed Jun. 8, 2007, and entitled "Method and System for High Speed Options Pricing"), the entire disclosures of each of which are incorporated herein by reference.

FIELD OF THE INVENTION

[0003] The present invention relates to the field of data processing platforms for financial market data.

BACKGROUND AND SUMMARY OF THE INVENTION

[0004] 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.

[0005] 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.

[0006] 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. FIG. **1** illustrates an exemplary platform that is currently known in the art. As shown in FIG. **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 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.

[0007] 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.

[0008] This partitioning approach is shown by FIG. **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.

[0009] Despite the improvements to the industry that these systems have provided, the inventors herein believe that significant further improvements can be made. In doing so, the inventors herein disclose that the underlying technology disclosed in the related patents and patent applications listed and incorporated herein above to fundamentally change the system architecture in which market data platforms are deployed.

[0010] In above-referenced related patent application Ser. No. 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 Ser. No. 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 and incorporated Ser. No. 11/561,615 patent application).

[0011] Since that time, the inventors herein have greatly expanded the scope of functionality for processing streams of financial information with reconfigurable logic. With the invention 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 FIG. **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 capabilities relative to the conventional market data platform as illustrated by FIG. **1**; for example, the inventors envision that a 5:1 or greater reduction of appliances relative to the system architecture of FIG. **1** can be achieved in the practice of the present invention.

[0012] 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 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 processing functionality that is deployed on reconfigurable logic.

[0013] 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 speeds.

[0014] Also disclosed as an embodiment of the invention is a ticker 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 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 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 functional units, such as functional units **102** depicted in FIG. **1**, that are arranged together to operate on a financial market data stream **106** or data derived therefrom.

[0015] These and other features and advantages of the present invention will be understood by those having ordinary skill in the art upon review of the description and figures hereinafter.

BRIEF DESCRIPTION OF THE DRAWINGS

[0016] FIG. **1** depicts an exemplary system architecture for a conventional market data platform;

[0017] FIG. **2** is a block diagram view of an exemplary system architecture in accordance with an embodiment of the present invention;

[0018] FIG. **3** illustrates an exemplary framework for the deployment of software and firmware for an embodiment of the present invention;

[0019] FIG. **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;

[0020] FIG. **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;

[0021] FIG. **5** illustrates an example of how the firmware application modules of a pipeline can be deployed across multiple FPGAs;

[0022] FIG. **6** illustrates an exemplary architecture for a market data platform in accordance with an embodiment of the present invention;

[0023] FIG. **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;

[0024] FIG. **8** illustrates an exemplary firmware application module pipeline for a message transformation from the FAST format to the FIX format;

[0025] FIG. **9** illustrates an exemplary firmware application module pipeline for message format transformation, message data processing, and message encoding; and

[0026] FIG. **10** illustrates another exemplary firmware application module pipeline for message format transformation, message data processing, and message encoding.

[0027] FIG. **11** depicts another exemplary firmware application module pipeline for performing the functions including symbol mapping, Last Value Cache (LVC) updates, interest and entitlement filtering;

[0028] FIG. **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;

[0029] FIG. **13** depicts an exemplary embodiment of a hash function for deployment within a firmware application module configured to perform symbol mapping;

[0030] FIG. **14** depicts a preferred embodiment for generating a global exchange identifier (GEID) within a firmware application module configured to perform symbol mapping;

[0031] FIGS. **15**(*a*) and (*b*) depict an exemplary embodiment for a firmware application module configured to perform Last Value Cache (LVC) updating;

[0032] FIG. **16** depicts an exemplary embodiment for a firmware application module configured to perform interest and entitlement filtering;

[0033] FIG. **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;

[0034] FIG. **18** depicts an exemplary data flow for inbound exchange traffic in the ticker plant of FIG. **17**;

[0035] FIG. **19** depicts an exemplary processing of multiple thread groups within the ticker plant of FIG. **17**;

[0036] FIG. **20** depicts an example of data flow between the hardware interface driver and reconfigurable logic within the ticker plant of FIG. **17**;

[0037] FIG. **21** depicts an example of data flows within the ticker plant of FIG. **17** for data exiting the reconfigurable logic; and

[0038] FIG. **22** depicts an exemplary model for managing client connections with the ticker plant of FIG. **17**.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

[0039] FIG. **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 interface **240**). Preferably, data streams into the reconfigurable logic device by way of system bus **212**, although other design architectures are possible (see FIG. **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 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 a preferred embodiment, system bus **212** may be a PCI-X bus, although this need not be the case.

[0040] 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 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 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 with FIG. **1**.

[0041] 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 system.

[0042] 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, 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 **230***i* 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.

[0043] 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 mod-

ule **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.

[0044] 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.

[0045] The FAM chain **230** preferably comprises a plurality of firmware application modules (FAMs) **230***a*, **230***b*, . . . 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.

[0046] A communication path **232** connects the firmware socket module **220** with the input of the first one of the pipelined FAMs **230***a*. The input of the first FAM **230***a* 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 **230***m* with the firmware socket module **220**. The output of the final FAM **230***m* serves as the exit point from the FAM chain **230**. Both communication path **232** and communication path **234** are preferably multi-bit paths.

[0047] FIG. **3** depicts an exemplary framework for the deployment of applications on the system **200** of FIG. **2**. The top three layers of FIG. **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**.

[0048] 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.

[0049] The next layer is the module application programming interface (API) layer **302** which comprises a high level module API **302***a* and a low level module API **302***b*. The high level module API **302***a* can provide generic services to application level software (for example, managing callbacks). The low level module API **302***b* manages the operation of the operating system (OS) level/device driver software **304**. A software library interface **310** interfaces the high level module API **302***a* with the low level module API **302***b*. Additional details about this software library interface can be found in the above-referenced patent application Ser. No. 11/339,892.

[0050] 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 Ser. No. 11/339,892.

[0051] 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 Ser. No. 11/339,892.

[0052] FIG. **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 FIG. **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.

[0053] FIG. **4**(*b*) depicts an alternate configuration for a printed circuit board/card **400**. In the example of FIG. **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**.

[0054] It is worth noting that in either the configuration of FIG. **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 FIGS. **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. FIG. **5** depicts an example where numerous FAMs in a single pipeline are deployed across multiple FPGAs.

[0055] As shown in FIGS. **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 socket module **220** across the PCI-X bus to the software application executing on the computer system. As shown in FIG. **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**.

[0056] 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 FIG. **6**. The market data platform **600** shown in FIG. **6**

consolidates the functional units **102** shown in FIG. **1** into much fewer physical devices and also offloads much of the data processing performed by the GPPs of the functional units **102** to reconfigurable logic.

[0057] For example, with the architecture of FIG. **6**, the feed compressor **602** can be deployed in an appliance such as system **200** shown in FIG. **2**. The reconfigurable logic **202** can be implemented on a board **400** as described in connection with FIG. **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 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 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.

[0058] 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) **620***a*; thereby improving communication latency through the network because of the smaller message sizes.

[0059] 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 **620***a* and **620***b* 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 **620***a* for delivery to device **604**.

[0060] Device **604** can be deployed in an appliance such as system **200** shown in FIG. **2**. The reconfigurable logic **202** can be implemented on a board **400** as described in connection with FIG. **4**(*a*) or **4**(*b*). Whereas the conventional GPP-based system architecture shown in FIG. **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 FIG. **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**.

[0061] 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 FIG. **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 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.

[0062] LVCs maintain a database of financial instrument records whose functionality can be implemented in a FAM pipeline. Each record 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 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 Ser. No. 10/153,151 application.

[0063] 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 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 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 PxS for every trade meeting criteria X, wherein P equals the trade price 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.

[0064] 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.

[0065] 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 under-

lying 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 in 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 Ser. No. 11/760,211, filed Jun. 8, 2007, the entire disclosure of which is incorporated herein by reference.

[0066] 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 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.

[0067] A news database maintains a historical archive of news stories 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 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.

[0068] Appropriate FAM modules and corresponding FAM pipelines to implement these various functions for device **604** 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 in memory (such as on a disk embodied by data store **204** in connection with FIG. **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 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.

[0069] Traders at workstations **104** (or application programs **150** 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 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 FIG. **2**, wherein the reconfigurable logic **202** of system **200** can be implemented on a board **400** as described in connection with FIG. **4**(*a*) or **4**(*b*).

[0070] Device **606** preferably consolidates the following functionality at least partially into firmware resident on reconfigurable logic: 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.

[0071] An "order book server" is similar to a LVC in that the order 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 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 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 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 trading venues (which are the source of orders) excluded.

[0072] 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 part) to the most appropriate trading venue, such as an exchange, an Alternate Trading System (ATS), or an ECN.

[0073] The direct market access gateway functionality operates to relay orders to a trading venue (such as an exchange, ECN, ATS, etc.) via WAN **620***b*. Before sending an order out however, the gateway preferably transforms the order message to a format appropriate for the trading venue.

[0074] 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 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 parameters for an instrument or set of instruments. To aid this processing, the black box trading engine is fed with real-time market data.

[0075] An auto-quote server is similar to a black box trading engine. The auto-quote server operates to automatically generate firm quotes 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.

[0076] A feed/compliance journal can also be implemented in a FAM 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 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 and patent applications.

[0077] 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 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 device **606** processes the feed stream in accordance with the functionality described above.

[0078] 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 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 and a minimum amount of liquidity by following a set of programmatically-defined rules.

[0079] 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, 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 account information as well as an archive of orders and other activity for each account.

[0080] 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. 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.

[0081] 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.

[0082] 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**.

[0083] 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.

[0084] 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.

[0085] 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 per-

sistent 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.

[0086] 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.

[0087] 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.

[0088] 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.

[0089] 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.

[0090] 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.

[0091] 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.

[0092] 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 FIG. **16**.

[0093] 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 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 FIGS. **12-14**.

[0094] With record ID mapping, a generic identifying string for a record is mapped into a direct record key number that can be used by a caching module (e.g., LVC, OBC, GVC) or software to directly address the record in a storage medium.

[0095] 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 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 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 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 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 with respect to FIGS. **15**(*a*) and (*b*).

[0096] 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.

[0097] 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.

[0098] 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.

[0099] 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.

[0100] 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.

[0101] A programmatic record generation (with LVC, OBC, GVC, etc.) operation operates to generate records that represent synthetic 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 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 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 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.

[0102] A historic record search and filter operation operates to 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 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 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.

[0103] 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.

[0104] FIG. **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 form 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.

[0105] FIG. **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 FIG. **8** operates to transform FAST message to FIX messages. Memory units **810** and **812** are preferably embodied by memory **404** of board **400**.

[0106] FIG. **9** depicts an exemplary FAM pipeline for carrying out a variety of data processing tasks. The FAM pipeline of FIG. **9** takes in a FAST message stream. FAMs **902** and **904** operate in the same manner as FAMs **802** and **804** in FIG. **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.

[0107] 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.

[0108] FIG. **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 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 FIG. **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 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 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.

[0109] FAM **1004** 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"). FAM **1006** receives the output from FAM **1004** and serves to update the LVC cache via memory **1024**. The output of FAM **1006** is then provided in parallel to FAMs **1008**, **1010**, **1012**, and **1014**.

[0110] 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 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**.

[0111] 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**.

[0112] FIG. **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.

[0113] 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.

[0114] Like FAM **1004** in FIG. **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 FIG. **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.

[0115] 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. 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 FIG. **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.

[0116] 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.

[0117] 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.

[0118] FIG. **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 FIG. **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 an empty table slot is encountered. This technique of resolving collisions is known as open-addressing.

[0119] 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 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, ⊕, of A(x) and d(x). This 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. Methods for computing displacement table entries are known in the art.

[0120] 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 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 Int'l Conf. on Application-Specific Systems, Architectures and Processors, September 2004, pp. 365-375, the entire disclosure of which is incorporated herein by reference).

[0121] 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 financial instrument. As shown in FIG. **13**, the result of the hash key compare function may be used as a valid signal for the symbol ID output.

[0122] 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.

[0123] 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 FIG. **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.

[0124] 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.

[0125] As shown in FIGS. **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.

[0126] 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).

[0127] As reflected in FIGS. **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 according to the business logic for stock trade events on exchange BB.

[0128] As shown in FIG. **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 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.

[0129] 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 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. 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 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.

[0130] 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.

[0131] 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.

[0132] 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.

[0133] The LVC message/record updater passes updated message fields 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 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 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 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 appropriate. Other types of subscriptions, such as exchange-based subscriptions, may also be enabled by the FAM pipeline.

[0134] 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 (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 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 Manger 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 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 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 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 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.

[0135] FIG. **15**(*b*) presents an exemplary functional pipeline for masking memory and processing latencies. As discussed in relation to FIG. **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 block. Note that each functional block in FIG. **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 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.

[0136] 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.

[0137] As shown in FIG. **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 FIG. **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 FIG. **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**.

[0138] 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.

[0139] FIG. **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. FIGS. **18**, **19**, **20**, **21**, and **22** depict detailed views of the data flow and the component interaction presented in FIG. **16**. These figures provide additional information for inbound exchange data processing, data normalization, interaction with reconfigurable logic, data routing, and client interaction respectively.

[0140] 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.

[0141] 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.

[0142] The device depicted in FIG. **17** uses a novel approach to efficiently deliver the exchange data to the user process. FIG. **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 FIG. **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.

[0143] 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 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. 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 Driver.

[0144] 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, 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 Plant employs thread groups to efficiently normalize raw exchange data.

[0145] Thread groups improve processing efficiency of the preferred embodiment of a Ticker Plant by using the following techniques:

[0146] 1. Limit the execution of the Operating System scheduling mechanism by matching the number of worker threads to the number of available instruction processors.

[0147] 2. Enable multiple exchange feeds to be processed by a single thread group.

[0148] 3. Eliminate resource contention between different thread groups.

[0149] 4. Remove synchronization points within the compute path of each thread group, further increasing processing efficiency.

[0150] 5. Perform message normalization in a single compute path, including line arbitration, gap detection, retransmission processing, event parsing, and normalized event generation.

[0151] 6. Associate exchange feeds with individual thread groups using a configuration file.

[0152] FIG. **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**.

[0153] 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.

[0154] 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. FIG. **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 FIG. **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**.

[0155] 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**.

[0156] 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. FIG. **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 data and who have expressed interest in being notified of that data.

[0157] 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 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 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 do inject events directly into the Hardware Interface Driver at **2107** for delivery to the reconfigurable logic or to the MDC Driver at **2108** for delivery to a connected consumer.

[0158] 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 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.

[0159] The MDC driver also has the ability to route market events to 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 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.

[0160] FIG. **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.

[0161] 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.

[0162] Thus, as shown in FIG. **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.

[0163] 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.

[0164] 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**.

[0165] 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 FIG. **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 FIG. **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 FIG. **6** on a single system **200**.

[0166] 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 appended claims and their legal equivalents.

What is claimed is:

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

streaming a financial market data feed through a reconfigurable logic device, the reconfigurable logic device having a resident firmware application module pipeline; and

performing with the firmware application module pipeline a plurality of different financial data processing operations on the streaming financial market data.

2. The method of claim **1** wherein the financial market data feed comprises a financial market source data feed, the method further comprising:

receiving the financial market source data feed from a data source.

3. The method of claim **1** wherein the financial market data feed comprises a financial market secondary data feed.

4. The method of claim **1** wherein the financial market data feed comprises a plurality of messages, and wherein the performing step further comprises:

normalizing the messages to a common format.

5. The method of claim **4** wherein the performing step further comprises:

converting the financial market data to a plurality of messages having a predetermined format.

6. The method of claim **1** wherein the financial market data feed comprises a plurality of messages, and wherein the performing step further comprises:

parsing the messages into a plurality of data fields.

7. The method of claim **6** wherein the financial market data feed comprises a plurality of messages, and wherein the performing step further comprises:

performing a calculation operation on the data fields of the parsed messages.

8. The method of claim **1** wherein the financial market data feed comprises a plurality of messages, and wherein the performing step further comprises:

transforming the messages from a first format to a second format.

9. The method of claim **8** wherein the first format comprises a format as received from an external feed source.

10. The method of claim **8** wherein the first format comprises FIX and wherein the second format comprises FAST.

11. The method of claim **8** wherein the first format comprises FAST and wherein the second format comprises FIX.

12. The method of claim **8** wherein the first format comprises a format as received from an external feed source, and wherein the second format comprises FAST.

13. The method of claim **1** wherein the firmware application module pipeline is configured to perform at least one financial data processing operation selected from the group consisting of: a feed compression operation, a feed decompression operation, a feed handler operation, a rule-based calculation operation, a feed journal operation, an alert generation engine, a Last Value Cache (LVC) server operation, a historical time-series oriented databases with analytics operation, a news database operation with a search capability, an order book server operation, an order router operation, a direct market access gateway operation, a trading engine, an auto-quote server operation, a compliance journal operation, an internal matching system operation, an order management system operation, an entitlements and reporting operation, a management and monitoring operation, and a publishing and contribution server operation.

14. The method of claim **1** further comprising:

providing data processed through the reconfigurable logic device to a trader workstation user interface.

15. A device for processing a streaming financial market data feed, the device comprising:

a reconfigurable logic device having a firmware application module pipeline, the reconfigurable logic device being configured to receive the data feed and process the data feed through the firmware application module pipeline to perform a plurality of different financial data processing operations thereon.

16. The device of claim **15** wherein the financial market data feed comprises a plurality of messages, and wherein the reconfigurable logic device is further configured to normalize the messages to a common format.

17. The device of claim **15** wherein the reconfigurable logic device is further configured to convert the financial market data feed to a plurality of messages having a predetermined format.

18. The device of claim **15** wherein the financial market data feed comprises a plurality of messages, and wherein the reconfigurable logic device is further configured to parse the messages into a plurality of data fields.

19. The device of claim **18** wherein the financial market data feed comprises a plurality of messages, and wherein the reconfigurable logic device is further configured to perform a calculation operation on the data fields of the parsed messages.

20. The device of claim **15** wherein the financial market data feed comprises a plurality of messages, and wherein the reconfigurable logic device is further configured to transform the messages from a first format to a second format.

21. The method of claim **20** wherein the first format comprises a format as received from an external feed source.

22. The device of claim **20** wherein the first format comprises FIX and wherein the second format comprises FAST.

23. The device of claim **20** wherein the first format comprises FAST and wherein the second format comprises FIX.

24. The device of claim **15** wherein the firmware application module pipeline is configured to perform at least one financial data processing operation selected from the group consisting of: a feed compression operation, a feed decompression operation, a feed handler operation, a rule-based calculation operation, a feed journal operation, an alert generation engine, a Last Value Cache (LVC) server operation, a historical time-series oriented databases with analytics operation, a news database operation with a search capability, an order book server operation, an order router operation, a direct market access gateway operation, a trading engine, an auto-quote server operation, a compliance journal operation, an internal matching system operation, an order management system operation, an entitlements and reporting operation, a management and monitoring operation, and a publishing and contribution server operation.

25. The device of claim **15** further comprising:

a trader workstation in communication with the reconfigurable logic device, and wherein a user interface on the workstation is configured to display data processed by the reconfigurable logic device.

26. The device of claim **15** further comprising:

an application programming interface (API) and customer-supplied application software in communication with the reconfigurable logic device, and wherein the customer-supplied application software is configured to receive data from the reconfigurable logic device via the API and further process the received data.

27. 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.

28. The apparatus of claim **27** 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.

29. The apparatus of claim **27** 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.

30. The apparatus of claim **27** wherein the financial market data comprises a plurality of financial market data messages, and wherein the software logic is further configured to write each received financial market data message to a buffer that is mapped into both a kernel space and a user address space supported by an operating system of the processor.

31. The apparatus of claim **27** wherein the financial market data comprises a plurality of financial market data messages, and wherein the software logic is further configured to (1) maintain a plurality of different thread groups simultaneously, and (2) assign each financial market data message to a thread group based on a configuration file.

32. The apparatus of claim **31** wherein processor is further configured to receive the financial market data messages from a plurality of different financial market data feeds, and 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.

33. The apparatus of claim **27** wherein the financial market data comprises a plurality of financial market data messages, and wherein the software logic is further configured to write 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 an operating system of the processor.

34. The apparatus of claim **27** 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 an MDC 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, and wherein the MDC 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.

35. The apparatus of claim **34** wherein the MDC 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.

36. The apparatus of claim **27** 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.

37. 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 in response to the software logic controlling step.

38. The method of claim **37** 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.

39. The method of claim **37** 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.

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

a ticker plant configured to receive a stream of financial market data messages, the ticker plant comprising a processor and a reconfigurable logic device in communication with the processor, wherein the processor is configured to execute software logic, the software logic configured to manage a flow of the financial market data messages to the reconfigurable logic device, wherein the reconfigurable logic device is configured with firmware logic, the firmware logic configured to (1) receive the flow of financial market data messages directed thereto by the software logic, and (2) perform at least one financial data processing operation on the received flow to thereby generate a flow of firmware-processed financial market data.

41. The apparatus of claim **40** wherein the ticker plant is further configured with a network interface for communicating with a plurality of client computers over a network, and wherein software logic is further configured to manage the flow of the firmware-processed financial market data from the firmware logic to the network interface for subsequent delivery to the client computers over the network.

42. The apparatus of claim **41** 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.

43. The apparatus of claim **42** wherein the ticker plant further comprises 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 ticker plant, wherein the reconfigurable logic is 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.

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

receiving a stream of financial market data messages from an external feed;

executing software logic to manage a flow of the financial market data messages to a reconfigurable logic device, the reconfigurable logic device having firmware logic deployed thereon;

performing with the firmware logic at least one financial data processing operation on the flow of financial market data messages to thereby generate a flow of firmware-processed financial market data.

45. The method of claim **44** further comprising:

executing software logic to manage the flow of the firmware-processed financial market data to thereby deliver the firmware-processed financial market data from the firmware logic to a plurality of remote client computers over a network.

46. The method of claim **45** 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.

47. The method of claim **46** 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.

48. An apparatus for processing a stream of financial market data messages, the apparatus comprising:

a reconfigurable logic device configured with a plurality of firmware application modules (FAMs) arranged in a pipeline, the pipeline comprising: (1) a message parsing FAM configured to receive and parse the financial market data messages, (2) a symbol mapping FAM in communication with the message parsing FAM, (3) a last value cache (LVC) updating FAM in communication with the symbol mapping FAM, (4) an interest and entitlement filtering FAM in communication with the LVC updating FAM, and (5) a message formatting FAM in communication with the interest and entitlement filtering FAM.

49. The apparatus of claim **48**, wherein each financial market data message comprises a plurality of date fields, each data field having a data value, and wherein the message parsing FAM is further configured to receive and parse the plurality of financial market data messages into a plurality of its constituent data fields, at least one of the data fields for each message comprising a financial instrument symbol identifier;

wherein the symbol mapping FAM is configured to map each financial instrument symbol identifier to a symbol value understood within the pipeline;

wherein the LVC updating FAM is configured to perform at least one field value update operation for a plurality of financial instrument records in response to the parsed data fields and the symbol values;

wherein the interest and entitlement filtering FAM is configured to receive data from the LVC updating FAM and perform interest and entitlement filtering on that received data; and

wherein the message formatting FAM is configured to generate a plurality of financial market data messages for output from the reconfigurable logic device in response to the data passed by the interest and entitlement filtering FAM and at least a portion of the parsed data fields.

50. The apparatus of claim **49** wherein the symbol mapping FAM is configured to map each financial instrument symbol identifier to a symbol value using a hash function in combination with open addressing.

51. The apparatus of claim **49** wherein the updating FAM is further configured to maintain a cache memory of recently updated records.

52. The apparatus of claim **49** wherein the interest and entitlement filtering FAM is configured to operate on a plurality of bit vectors associated each financial market data message, at least one bit vector for each message defining whether any entities are interested in data relating to that message, at least another of the bit vectors for each message defining whether any entities are entitled to access data relating to that message.

53. An apparatus for processing financial market data, the financial market data comprising a plurality of financial market symbols, each financial market symbol identifying a financial instrument, the apparatus comprising:

a reconfigurable logic device configured to (1) receive the financial market data, (2) process the received financial market data to determine the financial instrument symbols therein, (3) map each determined financial instrument symbol to a second financial instrument symbol that also identifies the same financial instrument.

54. The apparatus of claim **53** wherein the financial market data comprises a plurality of financial market data messages, each message identifying a financial market symbol, and wherein the reconfigurable logic device is further configured to associate each message with the second financial instrument symbol that was mapped from the determined financial instrument symbol for that message.

55. The apparatus of claim **54** further comprising a memory configured to store the second financial instrument symbols, and wherein the reconfigurable logic device is further configured to map each determined financial instrument symbol to a second financial instrument symbol by performing a lookup in the memory in response to a hash operation.

56. The apparatus of claim **55** wherein each financial market data message further comprises a symbology identifier, and wherein the reconfigurable logic device is further configured to, for each message, (1) determine a key code therefor based on that message's symbology identifier, (2) perform a compression operation on that message's financial instrument symbol to generate a compressed financial instrument symbol, and (3) generate a hash key for the hash operation based at least in part upon the determined key code and the compressed financial instrument symbol.

57. The apparatus of claim **56** wherein the hash key comprises a concatenation of the determined key code and the compressed financial instrument symbol.

58. The apparatus of claim **57** wherein reconfigurable logic device is further configured to perform the hash operation using a near-perfect hashing function in combination with open addressing into the memory to resolve any hash collisions.

59. The apparatus of claim **58** wherein the memory comprises a plurality of memory addresses, and wherein the second financial instrument symbols comprise the memory addresses.

60. The apparatus of claim **56** wherein the reconfigurable logic device is further configured to (1) perform, in parallel, a plurality of different compression operations on each message's financial instrument symbol to thereby generate a plurality of different compressed financial instrument symbols, (2) select which of the plurality of different compressed financial instrument symbols are to be used for generating the hash key in response to the determined key code.

61. The apparatus of claim **55** wherein the memory is located on the reconfigurable logic device.

62. An apparatus for processing a plurality of financial market data messages, each financial market data message comprising a country identifier and an exchange identifier, the apparatus comprising:

a reconfigurable logic device configured to (1) receive the financial market data messages, (2) process the received financial market data messages to determine the country identifiers and exchange identifiers therein, (3) map each message's determined country identifier and exchange identifier to a global exchange identifier.

63. The apparatus of claim **62** further comprising a memory configured to store the global exchange identifiers, and wherein the reconfigurable logic device is configured to perform the mapping by using the country identifier and exchange identifier for direct addressing lookups in the memory.

64. The apparatus of claim **62** wherein the reconfigurable logic device is further configured perform the mapping using a hash table.

65. A method for processing financial market data, the financial market data comprising a plurality of financial market symbols, each financial market symbol identifying a financial instrument, the method comprising:

receiving the financial market data in firmware logic deployed on a reconfigurable logic device;

processing the received financial market data with the firmware logic to determine the financial instrument symbols therein, using the firmware logic, mapping each determined financial instrument symbol to a second financial instrument symbol that also identifies the same financial instrument.

66. The method of claim **65** wherein the mapping comprises performing a near-perfect hashing operation in combination with open addressing on data derived from the determined financial instrument symbols.

67. 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 configured to (1) receive the financial market data messages, and (2) process each received financial market data message to update a stored record for the financial instrument associated with that message.

68. The apparatus of claim **67** further comprising a record memory configured to store a plurality of records for a plurality of financial instruments, and wherein the reconfigurable logic device is further configured to (1) receive a financial instrument symbol identifier in association with each received message, (2) retrieve from the record memory a record for each message based at least in part upon each message's associated financial instrument symbol identifier, and (3) update each retrieved record in response to the processing of each financial market data message.

69. The apparatus of claim **68** further comprising a record management memory, the record management memory comprising a plurality of record management memory addresses, the record management memory being configured to store a plurality of pointers to records in the record memory, each pointer being stored in a record management memory address, and wherein the reconfigurable logic device is further configured to perform a lookup in the record management memory based at least in part upon each message's associated financial instrument symbol identifier to thereby retrieve a pointer to a location in the record memory where a record for that message's associated financial instrument is stored.

70. The apparatus of claim **69** wherein the record management memory is deployed on a memory device external to the reconfigurable logic device.

71. The apparatus of claim **70** wherein the memory device comprises at least one selected from the group consisting of an SRAM device and an SDRAM device.

72. The apparatus of claim **68** wherein the record memory is further configured to store, for each of a plurality of financial instruments, a composite record and a regional record for each exchange on which that financial instrument trades, and wherein the reconfigurable logic device is further configured to (1) receive an exchange identifier in association with each received message, and (2) retrieve from the record memory both the composite record and each regional record for the financial instrument associated with each message based at least in part upon each message's associated financial instrument symbol identifier and exchange identifier.

73. The apparatus of claim **72** further comprising a record management memory, the record management memory comprising a plurality of record management memory addresses, each record management memory address being associated with a financial instrument symbol identifier and storing a composite pointer to the composite record in the record memory for the financial instrument corresponding to that record management memory address's associated financial instrument symbol identifier and a regional pointer to a location in the record memory that is associated with each regional record for the financial instrument corresponding to that record management memory address's associated financial instrument symbol identifier, and wherein the reconfigurable logic device is further configured to, for each message, (1) retrieve the pointers stored in the record management address associated with that message's associated financial instrument symbol identifier, (2) retrieve the composite record from the record memory based at least in part on the retrieved composite pointer, and (3) retrieve each regional record from the record memory based at least in part on the retrieved regional pointer and that message's associated exchange identifier.

74. The apparatus of claim **72** wherein the reconfigurable logic device comprises a plurality of composite record update

engines and a plurality of regional record update engines for performing the processing of each received financial market data message, and wherein the reconfigurable logic device is further configured to (1) provide the retrieved composite records and the financial market data to the composite record update engines for processing thereby, and (2) provide the retrieved regional records and the financial market data to the regional record update engines for processing thereby.

75. The apparatus of claim **68** wherein the reconfigurable logic device comprises a plurality of update engines for performing the processing of each received financial market data message, and wherein the reconfigurable logic device is further configured to deliver the retrieved records and the financial market data to the update engines for processing thereby.

76. The apparatus of claim **75** further comprising a template memory, the template memory being configured to store business logic for use by the update engines.

77. The apparatus of claim **76** wherein the reconfigurable logic device is further configured to select the business logic to be used by the update engines for processing each message in response to at least one of the group consisting of a field in that message and a field in a retrieved record for that message.

78. The apparatus of claim **75** wherein the reconfigurable logic device further comprises a pipelined circuit configured to perform the retrieval and delivery operations such that the pipelined circuit can operate on a plurality of the messages simultaneously.

79. The apparatus of claim **78** wherein the pipelined circuit is configured to stall processing of a second message associated with a financial instrument in response to detecting that the pipelined circuit is already processing a first message associated with the same financial instrument as the second message, and wherein the pipelined circuit is configured to maintain the stall until processing of the first message is complete.

80. The apparatus of claim **78** wherein the reconfigurable logic device is further configured to (1) maintain a cache memory for storing recently updated records, (2) for each message associated with a financial instrument for which a recently-updated record is stored in the cache memory, deliver the cached record to the update engines rather than a record retrieved from the record memory.

81. The apparatus of claim **68** wherein the reconfigurable logic device is further configured to (1) detect whether a received message pertains to a financial instrument for which no record is stored in the record memory, and (2) in response to a detecting that a received message pertains to a financial instrument for which no record is stored in the record memory, writing a new record for that financial instrument into the record memory based at least in part on that received message.

82. The apparatus of claim **68** wherein the reconfigurable logic device is further configured to add a new field to at least a plurality of the records in response to the processing.

83. The apparatus of claim **68** wherein the record memory is deployed on a memory device external to the reconfigurable logic device.

84. The apparatus of claim **83** wherein the memory device comprises at least one selected from the group consisting of an SRAM device and an SDRAM device.

85. 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:

retrieving a record for a financial instrument associated with a financial market data message;

performing at least one financial data processing operation on at least a portion of the financial market data message with reconfigurable logic; and

updating the retrieved record in response to the performing step.

86. The method of claim **85** wherein the retrieving step comprises retrieving a composite record for the financial instrument and at least one regional record for the financial instrument, and wherein the performing step comprises performing the a plurality of financial data processing operations on at least a portion of the message simultaneously for both the retrieved composite record and the at least one retrieved regional record, and wherein the updating step comprises updating the retrieved composite record and the at least one retrieved regional record in response to the performing step.

87. 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 record memory configured to store a plurality of records for a plurality of financial instruments;

a reconfigurable logic device in communication with the record memory, wherein the reconfigurable logic device is configured to (1) receive the financial market data messages, (2) retrieve from the record memory the records for the messages' associated financial instruments, (3) process each received financial market data message to update the record for the financial instrument associated with that message, and wherein each record comprises an interest list that identifies whether any of a plurality of entities have expressed an interest in being notified of data relating to the updated record.

88. The apparatus of claim **87** wherein the interest list comprises a bit vector having a plurality of bit positions, wherein each bit position corresponds to a different entity, and wherein a bit value at each bit position determines whether the entity corresponding to that bit position is an interested entity.

89. An apparatus for processing financial market data messages, each financial market data message comprising financial market data and being associated with an interest list and entitlement list, each interest list identifying whether any of a plurality of entities have expressed an interest in being notified of data relating to its associated message, each entitlement list identifying whether any of a plurality of entities are entitled to be notified of data relating to its associated message, the apparatus comprising:

a reconfigurable logic device configured to (1) receive a plurality of the messages, and (2) perform interest and entitlement filtering on the received messages in response to each message's associated interest list and entitlement list.

90. The apparatus of claim **89** wherein the interest list comprises a bit vector having a plurality of bit positions, wherein each bit position corresponds to a different entity, and wherein a bit value at each bit position determines whether the entity corresponding to that bit position is an interested entity.

91. The apparatus of claim **90** wherein the entitlement list comprises a bit vector having a plurality of bit positions, wherein each bit position corresponds to a different entity,

and wherein a bit value at each bit position determines whether the entity corresponding to that bit position is an entitled entity.

92. The apparatus of claim **91** wherein the same bit positions in the interest lists and the entitlement lists for each message correspond to the same entities.

93. The apparatus of claim **92** wherein the reconfigurable logic device is further configured to perform the interest and entitlement filtering by performing, for each message, a bitwise AND operation on the entitlement list bit vector and the interest list bit vector associated with that message to thereby generate an entitled interest list bit vector, wherein the bit values at the different bit positions of the entitled interest list bit vector identify which interested entities are entitled to be notified of data relating to that message.

94. The apparatus of claim **92** wherein the reconfigurable logic device is further configured to (1) operate on a plurality of different interest list bit vectors that are associated with each message, and (2) for each message, perform a bitwise OR operation on the plurality of interest list bit vectors for the same message to thereby generate a composite interest list bit vector for each message.

95. The apparatus of claim **94** wherein the reconfigurable logic device is further configured to perform the interest and entitlement filtering by performing, for each message, a bitwise AND operation on the entitlement list bit vector and the composite interest list bit vector associated with that message to thereby generate an entitled interest list bit vector, wherein the bit values at the different bit positions of the entitled interest list bit vector identify which interested entities are entitled to be notified of data relating to that message.

96. The apparatus of claim **95** wherein at least one of the plurality of interest list bit vectors comprises an exchange interest list bit vector.

97. The apparatus of claim **96** wherein the reconfigurable logic device is further configured to, for each message, (1) receive an exchange identifier associated therewith, and (2) retrieve an exchange interest list bit vector from a table based on the received exchange identifier for that message.

98. The apparatus of claim **89** wherein the reconfigurable logic device is further configured to drop a message if the interest and entitlement filtering results in a determination that no entitled and interested party is to be notified of data relating to that message.

99. A method for processing financial market data messages, each financial market data message comprising financial market data and being associated with an interest list and entitlement list, each interest list identifying whether any of a plurality of entities have expressed an interest in being notified of data relating to its associated message, each entitlement list identifying whether any of a plurality of entities are entitled to be notified of data relating to its associated message, the method comprising:

receiving a plurality of the messages; and

performing interest and entitlement filtering with reconfigurable logic on the received messages in response to each message's associated interest list and entitlement list.

100. An apparatus for processing financial market data messages, each financial market data message comprising a plurality of data fields, each data field having a data value, the apparatus comprising:

a reconfigurable logic device configured to (1) receive the financial market data messages, and (2) parse each received financial market data message into its constituent data fields.

101. The apparatus of claim **100** wherein each message has any of a plurality of different formats, the apparatus further comprising a memory configured to store a data dictionary that defines how financial market data messages in a plurality of formats are to be parsed, and wherein the reconfigurable logic device is further configured to (1) determine the format exhibited by each received message, and (2) parse each received message into its constituent data fields as defined for the determined format in the data dictionary.

102. The apparatus of claim **101** wherein the memory comprises a memory device that is external to the reconfigurable logic device.

103. The apparatus of claim **102** wherein the memory device comprises at least one selected from the group consisting of an SRAM device and an SDRAM device.

104. An apparatus for processing financial market data, the financial market data comprising a plurality of data fields, each data field having a data value, the apparatus comprising:

a reconfigurable logic device configured to generate a plurality of financial market data messages from a plurality of the data fields, each generated message having a specified format.

105. The apparatus of claim **104** wherein each generated message exhibits any of a plurality of different formats, the apparatus further comprising a memory configured to store a data dictionary that defines a plurality of formats for financial market data messages, and wherein the reconfigurable logic device is further configured to access the data dictionary to determine how each generated message should be formatted.

106. The apparatus of claim **105** wherein the memory comprises a memory device that is external to the reconfigurable logic device.

107. The apparatus of claim **106** wherein the memory device comprises at least one selected from the group consisting of an SRAM device and an SDRAM device.

* * * * *

Attorney Docket No. 44826-72554

Method and System for Low Latency Basket Calculation

Field of the Invention:

The present invention relates to the field of performing basket calculation operations based on streaming data, particularly streaming financial market data.

Terminology:

The following paragraphs provide several definitions for various terms used herein. These paragraphs also provide background information relating to these terms.

Financial Instrument: As used herein, a "financial instrument" refers to a contract representing an 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 the 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).

Financial Market Data: As used herein, the term "financial market data" refers to 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.

APPENDIX B

Basket: As used herein, the term "basket" refers to a collection comprising a plurality of elements, each element having one or more values. The collection may be assigned one or more Net Values (NVs), wherein a NV is derived from the values of the plurality of elements in the collection. For example, a basket may be a collection of data points from various scientific experiments. Each data point may have associated values such as size, mass, etc. One may derive a size NV by computing a weighted sum of the sizes, a mass NV by computing a weighted sum of the masses, etc. Another example of a basket would be a collection of financial instruments, as explained below.

Financial Instrument Basket: As used herein, the term "financial instrument basket" refers to a basket whose elements comprise financial instruments. The financial instrument basket may be assigned one or more Net Asset Values (NAVs), wherein a NAV is derived from the values of the elements in the basket. Examples of financial instruments that may be included in baskets are securities (stocks), bonds, options, mutual funds, exchange-traded funds, etc. Financial instrument baskets may represent standard indexes, exchange-traded funds (ETFs), mutual funds, personal portfolios, etc. One may derive a last sale NAV by computing a weighted sum of the last sale prices for each of the financial instruments in the basket, a bid NAV by computing a weighted sum of the current best bid prices for each of the financial instruments in the basket, etc.

GPP: As used herein, the term "general-purpose processor" (or GPP) refers to a hardware device having a fixed form and whose functionality is variable, wherein this variable functionality is defined by fetching instructions and executing those instructions, of which a conventional central processing unit (CPU) is a common example. Exemplary embodiments of GPPs include an Intel Xeon processor and an AMD Opteron processor.

Reconfigurable Logic: As used herein, the term "reconfigurable logic" refers to any logic technology whose form and function can be significantly 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.

Software: As used herein, the term "software" refers to data processing functionality that is deployed on a GPP or other processing devices, wherein software cannot be used to change or define the form of the device on which it is loaded.

Firmware: As used herein, the term "firmware" refers to data processing functionality that is deployed on reconfigurable logic or other processing devices, wherein firmware may be used to change or define the form of the device on which it is loaded.

Coprocessor: As used herein, the term "coprocessor" refers to a computational engine designed to operate in conjunction with other components in a computational system having a main processor (wherein the main processor itself may comprise multiple processors such as in a multi-core processor architecture). Typically, a coprocessor is optimized to perform a specific set of tasks and is used to offload tasks from a main processor (which is typically a GPP) in order to optimize system performance. The scope of tasks performed by a coprocessor may be fixed or variable, depending on the architecture of the coprocessor. Examples of fixed coprocessor architectures include Graphics Processor Units which perform a broad spectrum of tasks and floating point numeric coprocessors which perform a relatively narrow set of tasks. Examples of reconfigurable coprocessor architectures include reconfigurable logic devices such as Field Programmable Gate Arrays (FPGAs) which may be reconfigured to implement a wide variety of fixed or programmable computational engines. The functionality of a coprocessor may be defined via software and/or firmware.

Hardware Acceleration: As used herein, the term "hardware acceleration" refers to the use of software and/or firmware implemented on a coprocessor for offloading one or more processing tasks from a main processor to decrease processing latency for those tasks relative to the main processor.

Bus: As used herein, the term "bus" 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.

Pipelining: As used herein, the terms "pipeline", "pipelined sequence", or "chain" refer to an arrangement of application modules wherein the output of one application module is connected to the input of the next application module in the sequence. This pipelining arrangement allows each application module to independently operate on any data it receives during a given clock cycle and then pass its output to the next downstream application module in the sequence during another clock cycle.

Background and Summary of the Invention:

The ability to closely track the value of financial instrument baskets throughout a trading day as large volumes of events, such as trades and quotes, are constantly occurring is a daunting task. Every trade made on a financial instrument which is part of the portfolio of financial instruments underlying a financial instrument basket will potentially cause a change in that basket's value. The inventors herein believe that conventional techniques used to compute basket values have been unable to keep up with the high volume of events affecting basket values, thereby leading to inaccuracy for financial instrument basket values relative to the current state of the market because, with conventional techniques, the currently computed basket

value will not be reflective of current market conditions, but rather market conditions as they existed anywhere from milliseconds to several seconds in the past.

As noted above, financial instrument baskets may represent standard indexes, ETFs, mutual funds, personal portfolios, etc.

An index represents the performance of a group of companies, wherein the group can be decided by various criteria such as market capitalization (e.g., the S&P 500), industry (e.g., the Dow Jones Industrial Average (DJIA)), etc.

A mutual fund is a professionally-managed form of collective investment wherein the buyer pays the fund owner a certain amount of money and in exchange receives shares of the fund.

ETFs are very similar to mutual funds with an important difference being that ETFs can be traded continuously throughout the trading day on an exchange, just like stocks. Therefore, the prices of ETFs fluctuate not only according to the changes in their underlying portfolios, but also due to changes in market supply and demand for the ETFs' shares themselves. Thus, ETFs provide the trading dynamics of stocks as well as the diversity of mutual funds. Additionally, ETFs typically track a well-established market index, trying to replicate the returns of the index.

Personal portfolios are groupings of financial instruments that are defined by an individual for his/her personal investment goals.

A financial instrument basket is defined by its set of underlying financial instruments. The basket definition also specifies a weight for each of the financial instruments within the basket. For example, consider a hypothetical basket denoted by B. Basket B contains M financial instruments with symbols C_i, wherein $i=1, 2, \ldots, M$. Further, the weights of each financial instrument i in the basket is denoted by w_i, wherein $i=1, 2, \ldots, M$. Associated with a basket is a measure of its monetary value known as its net asset value (NAV). If one assumes that the value at the current instant of each financial instrument i within basket B is T_i, wherein $i=1, 2, \ldots, M$, then the NAV of basket B can be computed as:

$$NAV = \frac{1}{d}\sum_{i=1}^{M} w_i T_i \qquad (1)$$

wherein d is a divisor that is associated with the basket and discussed in greater detail hereinafter. Thus, equation (1) specifies that a basket's NAV is the weighted sum of the worth of the financial instruments within the baskets, wherein the divisor acts as a normalizing factor.

The value of T represents a price for the financial instrument's shares. The price can be expressed as the last sale/trade price (last), the best offer to buy price (bid), the best offer to sell price (ask), or specific offer prices to buy or sell (limit orders).

The value of w can be defined differently for each basket. For example, with capitalization-weighted indexes (such as the S&P 500), the weight of the financial instrument can be defined as the number of outstanding shares of the financial instrument multiplied by a "free-float" factor, wherein the free-float factor is a number between 0 and 1 that adjusts for the fact that certain shares of the financial instrument are not publicly available for trading. With an equal-weighted index (such as the DJIA), the weight of a financial instrument is set to 1. With ETFs and personal portfolios, the weight of a financial instrument can be set equal to a specified number which reflects the fact that owning many shares for each of the basket's financial instruments amounts to owning one share of that ETF or personal portfolio. Thus, in order to own one share of basket B, one would have to purchase w_1 shares of financial instrument C_1, w_2 shares of financial instrument C_2, and so on up to w_M shares of financial instrument C_M.

The value of d can also be defined differently for each basket. For an index, the divisor can be set to the current index divisor, which can be either the number of financial instruments in the index (e.g., 500 for the S&P 500) or some other specified value. For ETFs, the divisor can change during the trading day (as the divisor can also do with respect to indexes).

As indicated above, whenever there is a change in worth for one of a basket's underlying financial instruments, then the NAV for that basket should be calculated in accordance with formula (1). However, financial market data messages from exchanges stream into a platform such as a ticker plant at very high rates. For example, the options price reporting authority (OPRA) expects that its message rate in January 2008 will be approximately 800,000 messages/second. Each of these messages may potentially trigger multiple NAV recalculations. Moreover, because the same financial instrument may be a member of several baskets, the inventors herein estimate that, in order to track basket NAV values for all financial market data messages, the number of NAV calculations that will need to be performed every second will be at least on the order of 10 million NAV calculations/second. The inventors herein believe that conventional solutions are unable to keep up with such high update rates due to the serial nature of the conventional solutions, which constrain it to perform only one update at a time (including other tasks that a system may need to perform unrelated to basket calculations).

This shortcoming is problematic because it is highly desirable for financial traders to be able to calculate the basket NAVs extremely fast. An example will illustrate this desirability. As explained above, ETFs are baskets which can be traded like regular stocks on an exchange. The portfolio for an ETF is defined at the beginning of the trading day and remains fixed throughout the trading day barring rare events. Shares of ETFs can be created throughout the trading day by a creation process which works as follows. A buyer gives the owner of a fund the specified portfolio of financial instruments that make up the ETF and a cash payment in an amount equal to the dividend payments on those financial instruments (and possibly a transaction fee). The fund then issues shares of the ETF to the buyer. Shares of ETFs can also be redeemed for the underlying financial instruments in a similar manner: the investor gives the fund owner shares of the ETF (and possibly some cash as a transaction fee), and in return the investor receives the portfolio of financial

instruments that make up the ETF and cash equal to the dividend payments on those financial instruments.

During the trading day, the price of the shares of the ETF fluctuates according to supply and demand, and the ETF's NAV changes according to the supply and demand of the financial instruments in the ETF's portfolio. It may well happen that, during the trading day, the trading price of the ETF deviates from its NAV. If the price of the ETF is higher than the NAV, then the ETF is said to be trading "at a premium". If the price of the ETF is lower than the NAV, then the ETF is said to be trading "at a discount". If a financial trader detects a discrepancy between the ETF share price and the ETF NAV, then he/she can exploit that discrepancy to make a profit by way of arbitrage.

In one scenario, assume that the ETF share price, denoted by T_{ETF}, is higher than the ETF's NAV. In this case, a trader can execute the following steps:

- Assemble a basket by purchasing the shares of the financial instruments in the ETF's portfolio at a total cost approximately equal to the NAV;
- Create an ETF share by exchanging the assembled basket with the fund owner; and
- Sell the ETF on the exchange at a price approximately equal to T_{ETF}.

If the difference between T_{ETF} and the NAV is large enough to offset the transaction costs and other cash payments, then the trader can realize a profit by these actions. Moreover, this profit is realized by riskless arbitrage (assuming these steps could be performed instantaneously).

In another scenario, assume that the ETF share price, denoted by T_{ETF}, is lower than the ETF's NAV. In this case, a trader can purchase an ETF share, redeem it for the shares of the financial instruments in the portfolio, and sell those financial instrument shares on the exchange. Once again, if the difference between T_{ETF} and the NAV is

large enough to offset the transaction costs and other cash payments, then the trader can realize a profit at virtually no risk.

The crucial element in profiting from such riskless arbitrages for ETFs is detecting the difference between the ETF's share price and the ETF's NAV. The first trader to detect such discrepancies will stand to benefit the most, and the latency in NAV computation becomes important - the computation of the NAV should be performed as quickly as possible to beat competitors in the race toward detecting profit opportunities.

Independently of the ETF scenarios discussed above, another utility of basket NAV calculation is that the NAV provides a measure of how the basket's portfolio is faring. A steadily increasing NAV indicates that investing in more such baskets may be profitable. On the other hand, a decreasing NAV indicates that it may be desirable to sell the basket portfolio in order to avoid loss. This strategy is similar to ones that would be employed when trading in the shares of an individual financial instrument with the basket providing a benefit by mitigating risk through diversification across a portfolio of financial instruments (e.g., if the basket as a whole has an increasing NAV, then investing in that basket will be desirable independently of how any individual financial instrument in the basket is performing).

In an effort to satisfy a need in the art for high volume low latency basket value computations, the inventors herein disclose a technique for computing basket values for a set of baskets based on the data within a high speed data stream as that data streams through a processor.

According to one aspect of a preferred embodiment of the present invention, the inventors disclose a technique for computing basket values wherein a coprocessor is used to perform the basket calculations. By offloading the computational burden of the basket calculations to a coprocessor, the main processor for a system can be freed to perform different tasks. The coprocessor preferably hardware accelerates the basket calculations using reconfigurable logic, such

as Field Programmable Gate Arrays (FPGAs). In doing so, a preferred embodiment preferably harnesses the underlying hardware-accelerated technology disclosed in the following patents and patent applications: U.S. patent 6,711,558 entitled "Associated Database Scanning and Information Retrieval", U.S. Patent 7,139,743 entitled "Associative Database Scanning and Information Retrieval using FPGA Devices", U.S. Patent Application Publication 2006/0294059 entitled "Intelligent Data Storage and Processing Using FPGA Devices", U.S. Patent Application Publication 2007/0067108 entitled "Method and Apparatus for Performing Biosequence Similarity Searching", U.S. Patent Application Publication _____ entitled "Method and Apparatus for Protein Sequence Alignment Using FPGA Devices" (published from U.S. Application Serial No. 11/836,947, filed August 10, 2007), U.S. Patent Application Publication 2007/0130140 entitled "Method and Device for High Performance Regular Expression Pattern Matching", U.S. Patent Application Publication 2007/0260602 entitled "Method and Apparatus for Approximate Pattern Matching", U.S. Patent Application Publication 2007/0174841 entitled "Firmware Socket Module for FPGA-Based Pipeline Processing", U.S. Patent Application Publication 2007/0237327 entitled "Method and System for High Throughput Blockwise Independent Encryption/Decryption", U.S. Patent Application Publication 2007/0294157 entitled "Method and System for High Speed Options Pricing", and pending U.S. Application Serial No. 11/765,306 entitled "High Speed Processing of Financial Information Using FPGA Devices", filed June 19, 2007, the entire disclosures of each of which are incorporated herein by reference.

According to another aspect of a preferred embodiment of the present invention, the inventors disclose that the different tasks involved in such basket calculations can be relegated to different modules within a pipeline. For example, one pipeline module can be configured to determine which baskets are impacted by a current data event within the stream. Another pipeline module can be configured to receive information about the impacted baskets and compute data indicative of a new basket value for each of those impacted baskets.

By pipelining these modules together, the two modules can work on different messages and/or baskets at the same time in parallel with each other.

In a preferred embodiment, this pipeline comprises a basket association lookup module in communication with a basket value updating module. The basket association lookup module is configured to determine which baskets are impacted by each data message, while the basket value updating module is configured to compute at least one new value for each impacted basket based on the information within the data messages (for example, in an embodiment operating on financial market data, this information can be the financial instrument price information within the financial market data messages).

The basket value updating module preferably employs a delta calculation approach to the computation of the updated basket values, as explained in greater detail below. With such an approach, the number of arithmetic operations needed to compute each basket value will remain the same regardless of how many elements such as financial instruments are members of a given basket. Furthermore, the basket value updating module may optionally be configured to compute a plurality of different types of basket values for the same basket simultaneously using parallel computation logic.

The basket association lookup module preferably accesses a basket set table which stores at least a portion of the data needed by the basket value updating module for the basket value calculations. In a preferred embodiment pertaining to financial information, this data can be indirectly indexed in the table by financial instrument such that, as a new message pertaining to a financial instrument is received, the appropriate data needed for the basket calculations can be retrieved. To provide the indirection, a second table is employed which indexes pointers into the basket set table by financial instrument.

Furthermore, in an embodiment wherein the pipeline comprises a firmware pipeline deployed on a coprocessor such as a reconfigurable logic device, the design of these modules can be tailored to the

hardware capabilities of the coprocessor, thereby providing a level of synergy that streamlines the basket calculations so that the data streams can be processed at line speeds while a main processor is still free to perform other tasks.

The inventors also note that the pipeline can employ a price event trigger module downstream from the basket value updating module. The price event trigger module can be configured to determine whether any of the updated basket values meet a client-specified trigger condition. Any updated basket values which meet the client-specified trigger condition can then be forwarded to the attention of the pertinent client (e.g., a particular trader, a particular application, etc.). By doing so, clients can take advantage of the low latency nature of the basket calculation pipeline to learn of trading opportunities which may be available.

Furthermore, the pipeline can employ an event generator module downstream from the price event trigger module, wherein the role of the event generator module is to generate a message event for delivery to the client in response to a trigger being set off by the price event trigger module. This message event preferably contains the updated basket value..

Further still, to limit the volume of financial market data messages processed by the basket association lookup module (and its downstream modules in the pipeline), the pipeline can employ a message qualifier filter module upstream from the basket association lookup module. Preferably, the message qualifier filter module is configured to drop messages from the financial market data message stream which are not pertinent to the basket calculations.

A pipeline such as the one described herein can be used in an embodiment of the invention to identify arbitrage conditions (such as the ETF arbitrage scenario discussed above) as they may arise in connection with financial instrument baskets and the current state of the markets. By detecting these arbitrage conditions quickly via the low latency basket calculation pipeline, a trader is enabled to make

trades on one or more exchanges which take advantage of the detected arbitrage condition.

These and other features and advantages of the present invention will be apparent to those having ordinary skill in the art upon review of the following description and drawings.

Brief Description of the Drawings:

Figures 1(a) and (b) depict exemplary embodiments for a system on which hardware-accelerated basket calculations can be performed;

Figures 2(a) and (b) illustrate exemplary printed circuit boards for use as coprocessor 140;

Figure 3 illustrates an example of how a firmware pipeline can be deployed across multiple reconfigurable logic devices;

Figure 4(a) is a high level block diagram view of how a coprocessor can be used to perform basket calculations;

Figure 4(b) illustrates an exemplary process flow for computing an updated basket value using a delta calculation approach;

Figure 5 depicts an exemplary basket calculation engine pipeline in accordance with an embodiment of the invention;

Figure 6 depicts an exemplary basket association lookup module;

Figure 7 depicts an exemplary basket set pointer table;

Figure 8 depicts an exemplary basket set table;

Figure 9 depicts an exemplary basket value updating module;

Figures 10(a)-(c) depict exemplary embodiments for the NAV compute logic within the basket value updating module;

Figure 11 depicts an exemplary embodiment of the basket value updating module employing a plurality of NAV update engines;

Figure 12 depicts an exemplary basket calculation engine pipeline in accordance with another embodiment of the invention;

Figure 13 depicts an exemplary price event trigger module;

Figure 14 depicts an exemplary embodiment for the NAV-to-trigger comparison logic within the price event trigger module;

Figure 15 depicts an exemplary embodiment for the price event trigger module employing a plurality of client-specific price event trigger modules;

Figure 16 depicts an exemplary basket calculation engine pipeline in accordance with another embodiment of the invention;

Figure 17 depicts an exemplary event generator module;

Figure 18 depicts an exemplary basket calculation engine pipeline in accordance with yet another embodiment of the invention;

Figure 19 depicts an exemplary message qualifier filter module;

Figure 20 depicts an exemplary basket calculation engine pipeline in accordance with another embodiment of the invention; and

Figure 21 depicts an exemplary ticker plant architecture in which a basket calculation engine pipeline can be employed.

Detailed Description of the Preferred Embodiment:

Figure 1(a) depicts an exemplary embodiment for system 100 configured to perform high speed basket calculations. Preferably, system 100 employs a hardware-accelerated data processing capability through coprocessor 140 to perform basket calculations. Within system 100, a coprocessor 140 is positioned to receive data that streams into the system 100 from a network 120 (via network interface 110). In a preferred embodiment, system 100 is employed to receive financial market data and perform basket calculations for financial instrument baskets. Network 120 thus preferably comprises a network through which system 100 can access a source for financial data such as the exchanges themselves (e.g., NYSE, NASDAQ, etc.) or a third party provider (e.g., extranet providers such as Savvis or BT Radians). Such incoming data preferably comprises a series of financial market data messages, the messages representing events such as trades and quotes relating to financial instruments. These messages can exist in any of a number of formats, as is known in the art.

The computer system defined by processor 112 and RAM 108 can be 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 system. Thus, processor 112, which serves as the central or main processor for system 100, preferably comprises a GPP.

In a preferred embodiment, the coprocessor 140 comprises a reconfigurable logic device 102. Preferably, data streams into the reconfigurable logic device 102 by way of system bus 106, although other design architectures are possible (see Figure 2(b)). Preferably, the reconfigurable logic device 12 is a field programmable gate array (FPGA), although this need not be the case. System bus 106 can also interconnect the reconfigurable logic device 102 with the processor 112 as well as RAM 108. In a preferred embodiment, system bus 106 may be a PCI-X bus or a PCI-Express bus, although this need not be the case.

The reconfigurable logic device 102 has firmware modules deployed thereon that define its functionality. The firmware socket module 104 handles the data movement requirements (both command data and target data) into and out of the reconfigurable logic device, thereby providing a consistent application interface to the firmware application module (FAM) chain 150 that is also deployed on the reconfigurable logic device. The FAMs 150i of the FAM chain 150 are configured to perform specified data processing operations on any data that streams through the chain 150 from the firmware socket module 404. 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 104. 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 perform an exact match operation between data and a key, the FAM's exact match operation can be parameterized to define the key(s) that the exact match operation will be run against. In this way, a FAM that is

configured to perform an exact match operation can be readily re-arranged to perform a different exact match operation by simply loading new parameters for one or more different keys in that FAM. As another example pertaining to baskets, a command can be issued to the one or more FAMs that make up a basket calculation engine to add/delete one or more financial instruments to/from the basket.

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 data through the FAM), but the FAMs can also be flexibly reprogrammed to change the parameters of their data processing operations.

The FAM chain 150 preferably comprises a plurality of firmware application modules (FAMs) 150a, 150b, … that are arranged in a pipelined sequence. However, it should be noted that within the firmware pipeline, one or more parallel paths of FAMs 150i can be employed. For example, the firmware chain may comprise three FAMs arranged in a first pipelined path (e.g., FAMs 150a, 150b, 150c) and four FAMs arranged in a second pipelined path (e.g., FAMs 150d, 150e, 150f, and 150g), wherein the first and second pipelined paths are parallel with each other. Furthermore, the firmware pipeline can have one or more paths branch off from an existing pipeline path. A practitioner of the present invention can design an appropriate arrangement of FAMs for FAM chain 150 based on the processing needs of a given application.

A communication path 130 connects the firmware socket module 104 with the input of the first one of the pipelined FAMs 150a. The input of the first FAM 150a serves as the entry point into the FAM chain

150. A communication path 132 connects the output of the final one of the pipelined FAMs 150m with the firmware socket module 104. The output of the final FAM 150m serves as the exit point from the FAM chain 150. Both communication path 130 and communication path 132 are preferably multi-bit paths.

The nature of the software and hardware/software interfaces used by system 100, particularly in connection with data flow into and out of the firmware socket module are described in greater detail in the above-referenced and incorporated U.S. Patent Application Publication 2007/0174841.

Figure 1(b) depicts another exemplary embodiment for system 100. In the example of Figure 1(b), system 100 includes a data store 142 that is in communication with bus 106 via disk controller 114. Thus, the data that is streamed through the coprocessor 140 may also emanate from data store 142. Data store 142 can be any data storage device/system, but it is preferably some form of mass storage medium. For example, data store 142 can be a magnetic storage device such as an array of Seagate disks.

Figure 2(a) depicts a printed circuit board or card 200 that can be connected to the PCI-X or PCI-e bus 106 of a commodity computer system for use as a coprocessor 140 in system 100 for any of the embodiments of Figures 1(a)-(b). In the example of Figure 2(a), the printed circuit board includes an FPGA 102 (such as a Xilinx Virtex II FPGA) that is in communication with a memory device 202 and a PCI-X bus connector 204. A preferred memory device 202 comprises SRAM and DRAM memory. A preferred PCI-X or PCI-e bus connector 204 is a standard card edge connector.

Figure 2(b) depicts an alternate configuration for a printed circuit board/card 200. In the example of Figure 2(b), a bus 206 (such as a PCI-X or PCI-e bus), one or more disk controllers 208, and a disk connector 210 are also installed on the printed circuit board 200. Any commodity disk interface technology can be supported, as is understood in the art. In this configuration, the firmware socket 104 also serves as a PCI-X to PCI-X bridge to provide the processor 112

with normal access to any disk(s) connected via the private PCI-X bus 206. It should be noted that a network interface can be used in addition to or in place of the disk controller and disk connector shown in Figure 2(b).

It is worth noting that in either the configuration of Figure 2(a) or 2(b), the firmware socket 104 can make memory 202 accessible to the bus 106, which thereby makes memory 202 available for use by an OS kernel as the buffers for transfers to the FAMs from a data source with access to bus. It is also worth noting that while a single FPGA 102 is shown on the printed circuit boards of Figures 2(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 200 or by installing more than one printed circuit board 200 in the system 100. Figure 3 depicts an example where numerous FAMs in a single pipeline are deployed across multiple FPGAs.

Figure 4(a) depicts at a high level a coprocessor 140 that receives an incoming stream of new financial instrument prices, T_j^{new}, and computes new basket values, NAV^{new}, in response to the received stream using a basket calculation engine 400. By offloading the basket calculations to coprocessor 140, the system 100 can greatly decrease the latency of calculating new basket values in response to new financial instrument prices.

The basket calculation engine (BCE) 400 preferably derives its computation of NAV^{new} from formula (1). A direct implementation of formula (1) by BCE 400 would comprise M multiplications, M-1 additions, and 1 division. While a practitioner of the invention may choose to implement such a direct use of formula (1) within BCE 400, a preferred embodiment for BCE 400 reduces the number of arithmetic operations that need to be performed to realize the function of formula (1). Given that the basis upon which the NAV for a basket is to be updated is a new price for one of the basket's underlying financial instruments, it suffices to calculate the contribution to the basket's NAV of the difference between the new financial instrument price and the old financial instrument price. This

calculated contribution can then be added to the old NAV value for the basket to find the updated NAV value. This process is referred to herein as a "delta calculation" for the NAV. This basis for this delta calculation approach is shown below, starting with formula (1).

$$NAV^{new} = \frac{1}{d}\sum_{i=1}^{M} w_i T_i$$

$$NAV^{new} = \left(\frac{1}{d}\sum_{\substack{i=1 \\ i\neq j}}^{M} w_i T_i\right) + \left(\frac{1}{d} w_j T_j^{new}\right)$$

$$NAV^{new} = \left(\frac{1}{d}\sum_{\substack{i=1 \\ i\neq j}}^{M} w_i T_i\right) + \left(\frac{1}{d} w_j T_j^{new}\right) + \frac{1}{d}\left(w_j T_j^{old} - w_j T_j^{old}\right)$$

$$NAV^{new} = \frac{1}{d}\left(\left(\sum_{\substack{i=1 \\ i\neq j}}^{M} w_i T_i\right) + w_j T_j^{old}\right) + \frac{1}{d}\left(w_j T_j^{new} - w_j T_j^{old}\right)$$

$$NAV^{new} = \frac{1}{d}\left(\left(\sum_{\substack{i=1 \\ i\neq j}}^{M} w_i T_i\right) + w_j T_j^{old}\right) + \frac{w_j}{d}\left(T_j^{new} - T_j^{old}\right)$$

$$NAV^{new} = NAV^{old} + \Delta j \qquad (2)$$

Thus, it can be seen that NAV^{new} can be computed as the sum of the old NAV price, NAV^{old}, and Δj, wherein Δj represents the delta contribution to the NAV of the new financial instrument price, and wherein Δj is computed as:

$$\Delta j = \frac{w_j}{d}\left(T_j^{new} - T_j^{old}\right) \qquad (3)$$

Accordingly, it can be seen that the delta calculation approach can reduce the number of computations needed to calculate the new basket value from M multiplications, M-1 additions, and 1 division to only 1 subtraction, 1 multiplication, 1 division, and 1 addition. This reduction can lead to powerful improvements in computational efficiency because a single basket may contain an arbitrary number of

financial instruments. It should be noted that, for some baskets, the value of M may be quite large. For example, a basket may be used to compute the Wilshire 5000 stock index, which is an index that includes approximately 6,300 securities. However, the presence of such a large number of securities within the Wilshire 5000 stock index would not add to the computational latency of a BCE 400 which employs the delta calculation approach because the NAV computation for such a basket will be based on formula (2) above.

Figure 4(b) depicts a high level process flow for the computation of the updated basket value by the BCE 400. At step 402, the BCE computes the delta contribution Δj of the new financial instrument price T_j^{new} to the basket value (see formula (3)). At step 404, the BCE computes the updated basket value NAV^{new} from the computed delta contribution (see formula (2)).

Figure 5 depicts a preferred pipeline 500 for realizing the BCE 400. Preferably, pipeline 500 is deployed as a firmware pipeline 150 on a reconfigurable logic device 102. The pipeline 500 preferably comprises a basket association lookup module 502 and a downstream basket value updating module 504. The basket association lookup module 502 receives a stream of financial market data messages, wherein each message represents a change in the bid, ask, and/or last price of a single financial instrument. Each message preferably comprises at least a symbol identifier, a global exchange identifier, and one or more of a last price, a bid price, and an ask price for the financial instrument corresponding to the symbol identifier. The symbol identifier (or symbol ID) preferably comprises a binary tag that uniquely identifies a particular financial instrument. The global exchange identifier (GEID) preferably comprises a binary tag that uniquely identifies a particular exchange for which the message is relevant. Data tags within the messages preferably identify whether the price information within the message pertains to a bid/ask/last price for the financial instrument. Also, the message fields are preferably normalized as between the different message

sources such prior to the time the messages reach the basket association lookup module 502.

The basket association lookup (BAL) module 502 is configured to determine, for each incoming message, a set of baskets which include the financial instrument that is the subject of that message. This basket set may comprise a plurality Q of baskets (although it should be noted that only one basket may potentially be present within the basket set). Thus, the volume heavy nature of tracking basket values for each financial market data message can be understood as each message causing a need for at least Q basket value calculations (a 1:Q expansion). In an exemplary embodiment, the maximum value for Q can be 1,024 baskets. However, other values could readily be used.

Figure 6 depicts an exemplary BAL module 502. System 100 preferably maintains a record for every known financial instrument, wherein each record preferably contains the list of baskets which contain that financial instrument, the relative weight of the financial instrument within each basket on the list, and the most recent bid, ask, and last sale price for the financial instrument. These records are preferably stored in a basket set table 608. System 100 also preferably maintains a basket set pointer table 604. Table 604 preferably comprises a set of pointers and other information that point to appropriate records in the basket set table 608. The use of the basket set pointer table 604 in combination with the basket set table 608 allows for more flexibility in managing the content of the basket set table 608. In an embodiment wherein the coprocessor employs a memory 202 such as that shown in Figures 2(a) and (b), this memory 202 may comprise an SRAM memory device and an SDRAM memory device. Preferably, the basket set pointer table 604 can be stored in the SRAM memory device while the basket set table 608 can be stored in the SDRAM memory device. However, it should also be noted that, for an embodiment of the coprocessor wherein one or more FPGA chips are used, either or both of tables 604 and 608 can be stored in available on-chip memory should sufficient memory be available. Furthermore,

tables 604 and 608 could be stored in other memory devices within or in communication with system 100.

In operation, the BAL module 502 performs a lookup in the basket set pointer table 604 using the symbol ID and GEID 600 for each message. Based on the symbol ID and GEID 600, a basket set pointer 606 is retrieved from table 604, and the BAL module 502 uses this basket set pointer 606 to perform a lookup in the basket set table 608. The lookup in the basket set table 608 preferably operates to identify the basket set 610 for the financial instrument identified by the symbol ID and identify the stored last/bid/ask prices 612 for that financial instrument. Each basket set comprises identifiers for one or more baskets of which that financial instrument is a member.

A subtractor 616 preferably operates to subtract the retrieved last/bid/ask prices 612 from the new last/bid/ask prices 602 contained in the current message to thereby compute the changes in the last/bid/ask prices 614, as shown in Figure 6. It should be noted that each message may comprise one or more price fields for the subject financial instrument. For any price fields not contained within the message, the resultant price change is preferably treated as zero. Thus, if a message does not contain the last sale price for the financial instrument, then the ΔLast price will be zero.

Figure 7 depicts an exemplary embodiment for the basket set pointer table 604 in greater detail. Table 604 comprises a plurality of records 700, wherein each record comprises a bit string corresponding to a set of flags 702, a bit string corresponding to a header block pointer 704, and a plurality of bit strings corresponding to different GEIDs 706. Each record 700 is keyed by a symbol ID such that the BAL module 502 is able to retrieve the record 700 from table 604 which corresponds to the symbol ID of the current message. The flags 702 denote whether the record 700 is valid (i.e., whether the subject financial instrument is contained within at least one basket). The header block pointer 704 serves as a pointer to a basket association record in table 608 for the financial instrument corresponding to the symbol ID.

Also, a given financial instrument may be fungible on multiple financial exchanges. The state of a financial instrument on a given exchange (e.g., the NYSE) is referred to as the regional view for that financial instrument. The aggregate state of that financial instrument across all exchanges is referred to as the composite view for that financial instrument. The composite value for the last price is preferably the most recent sales price for the financial instrument on any exchange. The composite value for the bid price is preferably the best of the bid prices for the financial instrument across all of exchanges on which that financial instrument is traded. The composite view for the ask price is preferably the best of the ask prices for the financial instrument across all of exchanges on which that financial instrument is traded. Thus, the bid price and ask price in the composite view are commonly referred to as the Best Bid and Offer (BBO) prices. Tables 604 and 608 allow for baskets to be defined with regional and/or composite views of financial instruments. For example, basket A may contain the composite view of the 10 technology stocks with the largest market capitalization. Basket B may contain the NYSE regional view of the 10 technology stocks with the largest market capitalization. Basket C may contain the composite view for 10 technology stocks and the NYSE regional view for 10 industrial stocks.

To accommodate such flexible basket definitions, table 604 preferably employs a plurality of GEID fields 706 within each record 700. Each GEID field 706 serves as a further index into table 608 to retrieve the appropriate composite basket association record and/or regional basket association record for the pertinent financial instrument. Thus, BAL module 502 uses the GEID of the current message to identify a matching GEID field 706 in record 700. Field 706_0, which corresponds to $GEID_0$, is preferably used to identify any basket association records in table 608 which correspond to the composite view for the financial instrument. The other fields 706 are used to identify any basket association records in table 608 which correspond to the region-specific view for the financial instrument. For example, $GEID_1$ may correspond to the NYSE view for the financial

instrument while $GEID_n$ may correspond to the NASDAQ view for the financial instrument. The matching GEID field 706 is then used as an additional index into table 608 to retrieve a set of basket association records for the financial instrument that are applicable to both composite views and regional views of the financial instrument.

Preferably, BAL module 502 will always retrieve the composite $GEID_0$ field applicable to a given financial instrument. Furthermore, for a message with regional price information about a financial instrument, the BAL module 502 also operates to match the GEID field in the message with a GEID field 706 in record 700 for that financial instrument. Thus, the basket set pointer 606 used by the BAL module 502 preferably comprises the header block pointer 704, the composite GEID index 706_0, and the regional GEID index 706_i for the subject message.

Figure 8 depicts an exemplary embodiment for the basket set table 608 in greater detail. Table 608 comprises a plurality of basket association records, wherein each record contains basket information for a different financial instrument. Each record preferably comprises a plurality of header blocks 800 and possibly one or more extension blocks 822. Each header block 800, which preferably has a fixed size, corresponds to the subject financial instrument and a particular GEID. Furthermore, each header block 800 preferably comprises a bit string corresponding to the GEID field 802 for that block, a bit string corresponding to an extension pointer 804, a bit string corresponding to a count field 806, a set of reserved bits 808, a bit string that corresponds to the most recently stored bid price 810 for the subject financial instrument, a bit string that corresponds to the most recently stored ask price 812 for the subject financial instrument, a bit string that corresponds to the most recently stored last price for the subject financial instrument 814, and a plurality of bit strings corresponding to a plurality of sets of basket information 816.

Preferably, each block 800 preferably comprises either composite basket association information or regional basket association information for the subject financial instrument. Also, the blocks 800 are preferably located in memory such that the composite and regional blocks for a given financial instrument are stored in contiguous memory addresses. In this way, the header block pointer 704 can be used to locate the financial instrument's composite block 800, while the GEID index 706 can be used to locate the applicable regional block 800 for the financial instrument by serving as an increment to the address found in the header block pointer 704.

Each basket information field 816 in table 608 preferably comprises a basket identifier and weight pair (Basket ID, Weight). The Basket ID serves to identify a particular basket while the weight serves to identify the weight to be used when computing the delta contribution for subject financial instrument to the identified basket.

Given that a single financial instrument may be present in a number of different baskets, it is possible that the number of (Basket ID, Weight) pairs needed for a given financial instrument will not fit within a single block 800. To accommodate such overflow situations, table 608 also preferably comprises a plurality of extension block records 822. Each extension block record 822 preferably has a fixed size and comprises a bit string corresponding to an extension pointer 818, a bit string corresponding to a count field 820, any one or more (Basket ID, Weight) pairs 816 as needed for the financial instrument. If the number of (Basket ID, Weight) pairs 816 for the financial instrument are unable to fit within a single extension block 822, then the extension pointer 818 will serve to identify the address of an additional extension block 822 in which the additional (Basket ID, Weight) pairs 816 are found. It should thus be noted that a plurality of extension blocks 822 may be needed to encompass all of a financial instrument's relevant (Basket ID, Weight) pairs 816. The count field 820 identifies how many (Basket ID, Weight) pairs 816 are present within the subject extension block 822.

The count field 806 within a block 800 serves to identify how many (Basket ID, Weight) pairs 816 are present within that block 800.

Fields 810, 812, and 814 contain the most recently stored bid/ask/last prices for the subject financial instrument on the pertinent exchange. As shown by Figure 6, this price information is used for computing the price change values 614 (or delta prices) that are relevant to the delta contribution calculation.

Thus, using the pointer 704 and GEID index reference 706 retrieved from table 604, the BAL module 502 is configured to access a block 800 in table 608 corresponding to the composite information relevant to the subject message (e.g., the block 800 shown in Figure 8 labeled with "$GEID_0$") and a block 800 in table 608 corresponding to the regional information relevant to the subject message (e.g., the record 800 shown in Figure 8 labeled with "$GEID_1$"). To find the appropriate regional block, the BAL module uses the matching GEID field 706 to count down from the pertinent composite block to the appropriate regional block. Thus, if the pertinent regional block for a particular financial instrument is the NYSE block, wherein the NYSE has a GEID value of that matches the second GEID field 706, $GEID_2$, in record 700, then the BAL module will find the appropiate NYSE block 800 in table 608 two blocks down from the composite block.

In the example of Figure 8, the regional information has an overflow condition, and the extension pointer 804 points to an extension block 822, as shown. Based on these lookups, the BAL module 502 retrieves the bid/ask/last prices and (Basket ID, Weight) pairs for the composite view of the financial instrument as well as the bid/ask/last prices and (Basket ID, Weight) pairs for the appropriate regional view of the financial instrument. It should be noted that the retrieved bid/ask/last prices 612 for the composite view will need to be associated with the retrieved list of (Basket ID, Weight) pairs 610 for the composite view and the retrieved bid/ask/last prices 612 for the regional view will need to be associated with the retrieved list of (Basket ID, Weight) pairs 610 for the regional view to ensure that downstream computations are based on the appropriate values. To

maintain this association, delta events as described below are preferably employed.

BAL module 502 is also preferably configured to update the records in table 608 with the new prices 602 in each message. This is shown by arrow 618 in Figure 6, which represents a write operation into the appropriate fields of table 608. However, it should be noted that if the message does not include a particular price field such as the last price field, then the BAL module 502 preferably maintains the last price field 814 in the pertinent block 800 in its existing state.

It should be noted that the array of GEID fields 706 for each record 700 in table 604 may be of fixed size for simplicity. Depending upon how much size a practitioner allocates to the GEID fields 706 in each record 700, it may be the case that a financial instrument trades on a sufficiently large number of exchanges that there are not a sufficient number of fields 706 for storing the financial instrument's relevant GEIDs. Thus, if this situation arises and the GEID 600 of the message does not find a match to any GEID fields 706 in the pertinent record 700 of table 604, then the BAL module 502, to retrieve an appropriate regional block from table 608, preferably indexes to the last header block 800 in the record for that financial instrument and conducts a linear search through the header blocks' GEID fields 802 to find a match to the GEID 600 associated with the message.

It should also be noted that if the GEID 600 is not found in any of the GEID fields 802 for the financial instrument record in table 608, this means that the regional view of that financial instrument corresponding to GEID 600 is not included in any baskets.

It should also be noted that baskets may be dynamically changed throughout the day by adding and/or removing (Basket ID, Weight) pairs from the basket association records in table 608 for financial instruments.

To add a financial instrument to a basket, the system adds the appropriate (Basket ID, Weight) pair to that financial instrument's record in table 608. A control process, preferably performed in

software executing on a GPP such as processor 112, preferably maintains a copy of tables 604 and 608 and computes where entries need to be added to the table 608 (and possibly table 604 if the GEID for that financial instrument is not present in the GEID fields 706 of record 700 for that financial instrument) to reflect the addition of the financial instrument to the basket. These determinations can be made using the symbol ID and GEID for the financial instrument to be added. The control process then preferably sends memory write commands to the pipeline for consumption by the BAL module 502 that are effective to write update tables 604 and 608 to reflect the addition of the subject financial instrument to a basket. The control process may also be configured to immediately send a synthetic event to the pipeline for consumption by the BAL module 502 wherein this event contains the current prices for the relevant financial instrument. This action will cause delta updates to the NAV values for the subject basket wherein these delta updates are relative to a zero (so the entirety of the price contribution is determined). However, it should also be noted that the control process may be configured to simply wait for subsequent market events to arrive at the pipeline and allow the delta updates to be applied at that time.

Removing a financial instrument from a basket generally involves removing the (basket ID, weight) pair for the subject basket from the record in table 608 for the subject financial instrument. Furthermore, to appropriately update the NAV values to reflect the deletion of the financial instrument from the basket, the control process preferably generates a synthetic event for delivery to the pipeline wherein the price information for the subject financial instrument is zero. This causes the delta price values 614 for the financial instrument to be the negative value of the price information for that financial instrument stored in table 608, thus removing the entirety of the price contribution for that financial instrument to the subject basket.

To add an entire basket to the system, it follows that the control process can issue appropriate memory write commands to the

pipeline that adds (Basket ID, Weight) pairs to table 608 for all of the new basket's constituent financial instruments. The control process can also generate synthetic events for delivery to the pipeline which reflects the current price information for these constituent financial instruments, as explained above. Moreover, as can be understood in connection with Figures 9 and 13 below, the control process can initiate the addition of appropriate entries to the divisor table, NAV tables, client NAV tables, and client trigger threshold tables to reflect the addition of the new basket to the system.

To delete an entire basket from the system, it also follows that the control process can be configured to initiate a removal of the (Basket ID, Weight) pairs from table 608 for all of the subject basket's constituent financial instruments. Similarly, the records in the tables shown in Figures 9 and 13 can also be updated to remove those records associated with the Basket ID of the removed basket.

The output from the BAL module 502 will be a stream of delta events. Each delta event preferably comprises a retrieved (Basket ID, Weight) pair 610 and at least one computed price change 614 for a financial instrument corresponding to that retrieved (Basket ID, Weight) pair 610. In an embodiment of the invention, each delta event includes all of the price deltas, even if one or more of the price deltas (e.g., the ΔBid price) is zero. However, this need not be the case, as noted below.

It should also be understood that, for each message event that is received at the input to the BAL module 502, a plurality of delta events may be produced at the output of the BAL module 502 due to the fact that the financial instrument which is the subject of the received message may be a member of a plurality of baskets. Partly because of this expansion, the low latency nature of pipeline 500 is particularly advantageous in allowing the task of updating basket values to keep up with the inflow of events from the different exchanges.

Returning to Figure 5, the basket value updating module 504 is configured to compute the new basket value for each delta event produced by the BAL module 502. Figure 8 depicts an exemplary basket value updating (BVU) module 504.

A delta event buffer 900 buffers the delta events produced by the BAL module 502. For each delta event read out of buffer 900, the BVU module 504 performs a lookup in a divisor table 902 to determine the appropriate value for the divisor d from formula (3) for the basket corresponding to the Basket ID within the current delta event. Thus, it can be seen that table 902 preferably indexes each divisor value by the Basket ID for the divisor's corresponding basket. The delta event buffer 900 and the divisor table 902 are preferably located within available on-chip memory for the reconfigurable logic device 102. However, it should be noted that buffer 900 and table 902 could alternatively be stored on any memory resource within or accessible to coprocessor 140.

A NAV update engine 950 then receives as an input the delta event information (the (Basket ID, Weight) pair 610 and the delta bid/ask/last prices 614) plus the divisor value d for that delta event to compute at least one updated basket value NAV^{new}. Preferably, the NAV update engine 950 is configured to compute a plurality of different types of basket NAV's for each basket. For example, the NAV update engine 950 can be configured to compute, in parallel, new NAVs based on bid prices, ask prices, last prices, bid-tick prices, and ask+tick prices (*Bid* NAV^{new}, *Ask* NAV^{new}, *Last* NAV^{new}, *Bid-Tick* NAV^{new}, and *Ask+Tick* NAV^{new} respectively), as shown in Figure 9. In these examples, the tick value is defined as the minimum increment used by the pertinent exchange to measure a change in financial instrument price.

A demultipexer 904 operates to route portions of the delta event information and the divisor value to the appropriate NAV compute logic 916. The NAV update engine preferably employs a plurality of parallel computing paths to compute each type of basket NAV.

The computing path for computing *Bid NAV*^{new} preferably operates on the Basket ID, Weight, Divisor, and ΔBid price. The BVU module 504 performs a lookup in a Bid NAV table 906 based on the Basket ID to retrieve a stored Bid NAV value (*Bid NAV*^{old}) for the subject basket. This *Bid NAV*^{old} value serves as the *NAV*^{old} value in formula (2) during computation of *Bid NAV*^{new}. NAV compute logic 916 then (i) computes the delta contribution Δj for the financial instrument according to formula (3) using the divisor value as the d term in formula (3), the Weight value as the w_j term in formula (3), and the ΔBid value as the $\left(T_j^{new} - T_j^{old}\right)$ term in formula (3), and (ii) thereafter computes *Bid NAV*^{new} according to formula (2) based on the computed Δj value and the *Bid NAV*^{old} value.

The computing path for computing *Ask NAV*^{new} preferably operates in the same manner as the computing path for *Bid NAV*^{new} albeit performing a lookup in an Ask NAV table 908 based on the Basket ID and using the ΔAsk price rather than the ΔBid price.

The computing path for computing *Last NAV*^{new} also preferably operates in the same manner as the computing path for *Bid NAV*^{new} albeit performing a lookup in an Last NAV table 910 based on the Basket ID and using the ΔLast price rather than the ΔBid price.

Likewise, the computing paths for computing the *Bid-Tick NAV*^{new} and *Ask+Tick NAV*^{new} value preferably operate in the same manner as the computing path for *Bid NAV*^{new} albeit performing lookups in, respectively, a Bid-Tick NAV table 912 and an Ask+Tick NAV table 914 based on the Basket ID and using, respectively, the ΔBid price and the ΔAsk price.

Table 912 can store previously computed NAVs for each basket that are based on a price for the financial instrument that is the previous bid price for the financial instrument minus a tick value. Table 914 can store previously computed NAVs for each basket that are based on a price for the financial instrument that is the previous ask price for the financial instrument plus a tick value.

Preferably, tables 906, 908, 910, 912, and 914 are stored in available on-chip memory for the reconfigurable logic device 102. However, it should be noted that these tables could alternatively be stored on any memory resource within or accessible to coprocessor 140.

Thus, it can be seen that the NAV update engine 950 is preferably configured to output, each clock cycle, an updated basket event, wherein each updated basket event comprises a plurality of different updated NAVs for a particular basket together with a Basket ID for that basket.

The BVU module is also preferably configured to update tables 906, 908, 910, 912, and 914 with the newly computed NAV values. Thus, the newly computed $Bid\ NAV^{new}$ for a given Basket ID can be fed back into an entry for that Basket ID in table 906 so that the next delta event pertaining to the same Basket ID will retrieve the most current NAV^{old} value. The memory for tables 906, 908, 910, 912, and 914 is preferably dual-ported, thus allowing for two read/write operations to occur on each clock cycle. If updating logic for the BVU module is pipelined, it can be expected that contention for the same Basket ID entry in a given table during a read and write operation on the same clock cycle will be rare and can be identified and avoided using techniques known in the art.

Also, a control process can issue a memory write command to the pipeline for consumption by the BVU module to update an entry in the divisor table 902 in the event of a change in value for a basket's divisor. Such a control process is preferably performed in software executing on a GPP such as processor 112.

Figure 10(a) depicts an exemplary embodiment for the NAV compute logic 916. A multiplier 1000 operates to compute the $w_j\left(T_j^{new} - T_j^{old}\right)$ portion of formula (3) using the Weight value and delta price information from each delta event. A divider 1002 operates to divide the $w_j\left(T_j^{new} - T_j^{old}\right)$ value by d to thereby compute Δj according to formula (3). Thereafter, an adder 1004 is used to add Δj to the NAV^{old} value, thereby computing NAV^{new} according to formula (2).

Figure 10(b) depicts another exemplary embodiment for the NAV compute logic 916. In the embodiment of Figure 10(b), the division operation is performed last, thereby improving the precision of the NAV computation relative to the embodiment of Figure 10(a). To re-order the arithmetic operations such that the division operation inherent in formula (2) by way of formula (3) is performed last, the embodiment of Figure 10(b) preferably adds a multiplication operation using multiplier 1006. This multiplication operation can be performed in parallel with the multiplication operation performed by multiplier 1000. Multiplier 1006 operates to multiply the NAV^{old} value by d, thereby resulting in the value S^{old}. In this embodiment, the delta contribution of the financial instrument can be expressed as $w_j\left(T_j^{new} - T_j^{old}\right)$, as shown at 1012 in Figure 10(b). Adder 1004 then computes the sum of S^{old} and $w_j\left(T_j^{new} - T_j^{old}\right)$. Given that S^{old} equals $dNAV^{old}$, it can readily be seen that the act of dividing $S^{old} + w_j\left(T_j^{new} - T_j^{old}\right)$ by d will result in NAV^{new} in accordance with formulas (2) and (3).

Figure 10(c) depicts yet another exemplary embodiment for the NAV compute logic 916. In the embodiment of Figure 10(c), the precision improvement of the Figure 10(b) embodiment is preserved while also eliminating the need for multiplier 1006. To do so, in a BVU module which employs the NAV compute logic 916 of Figure 10(c), the NAV tables 906, 908, 910, 912, and 914 preferably store S^{old} values rather than NAV^{old} values, thereby eliminating the need for multiplier 1006. Furthermore, to update tables 906, 908, 910, 912, and 914, the value of S^{new} (which is the same as $S^{old} + w_j\left(T_j^{new} - T_j^{old}\right)$) is fed back into the relevant tables via communication link 1010 such that S^{new} serves as S^{old} when the next delta event is processed.

It should be noted that the BVU module 504 can employ a plurality of NAV update engines 950 in parallel to thereby simultaneously compute a plurality of basket NAV types for a plurality of baskets. An example of this is shown in Figure 11, wherein the BVU module 504

comprises two parallel NAV update engines 950. In such an embodiment, the BVU module 504 reads two delta events out of buffer 900 each clock cycle. After lookups are performed in the divisor table 902 to identify the appropriate divisor value for each delta event, routing logic 1100 routes the delta event information and its associated divisor value to an appropriate NAV updated engine 950. Such routing logic 1100 can be configured to assign different segments of the Basket ID range to each set of parallel engines 950.

If desired, it should be noted that a single set of NAV tables 906, 908, 910, 912, and 914 can be shared by a plurality of NAV update engines 950. If the number of NAV update engines exceeds the number of available ports for reading from the NAV tables, then the BVU module 504 can be configured to interleave the accesses to these tables from the different NAV update engines. These tables could also be replicated if desired to avoid such interleaving. The same can be said with respect to divisor table 902 (while Figure 11 depicts an embodiment wherein two divisor tables are utilized, it should be noted that a single divisor table 902 may be shared by the different paths in Figure 11).

Figure 12 depicts an embodiment wherein pipeline 500 includes a price event trigger module 1200 in communication with the output from the BVU module 504. The price event trigger (PET) module 1200 is configured to determine which updated basket values produced by the BVU module 504 are to be reported to an interested client. These determinations can be made on the basis of any of a number of criteria. The client may be a particular trader who has requested that he/she be notified when the value of a basket changes by a certain amount, or the client may be another application within a trading system. The client may also be another module deployed on the coprocessor 140 (such as another FAM 150 if pipeline 500 is a FAM pipeline).

Figure 13 depicts an exemplary embodiment for a PET module 1200. Preferably, PET module 1200 is configured to operate in parallel on each basket value type for a basket generated by the BVU module 504.

Also, to determine which of the updated basket values should be reported to interested clients, the PET module 1200 preferably compares each updated basket value with a client-defined trigger threshold.

PET module 1200 preferably accesses a plurality of tables corresponding to different NAV types, wherein these tables store client reference values for each basket. These reference values are used by the PET module 1200 to judge whether new basket values should be reported to interested clients. In an exemplary embodiment, the reference values are NAV values for the subject baskets which were previously reported to the client. Thus, a client NAV table preferably exists for each basket value type generated by the BVU module 504. Thus, reference value table 1302 stores a plurality of client Bid NAVs for the different baskets, wherein each client Bid NAV is indexed by a basket identifier. Reference value table 1304 stores a plurality of client Ask NAVs for the different baskets (each client Ask NAV being indexed by a basket identifier). Reference value table 1306 stores a plurality of client Last NAVs for the different baskets (each client Last NAV being indexed by a basket identifier). Reference value table 1308 stores a plurality of client Bid-Tick NAVs for the different baskets (each client Bid-Tick NAV being indexed by a basket identifier). Reference value table 1310 stores a plurality of client Ask+Tick NAVs for the different baskets (each client Ask+Tick NAV being indexed by a basket identifier). Each of these tables preferably stores the most recent of their respective NAV types to be reported to the client. Thus, in the event of one of the triggers being set off by a new basket value, the PET module is preferably configured to update the NAV values in the client tables with the new basket values. Such updating logic can be configured to update the tables only for those NAV values which set off the trigger or can be configured to update all of the tables when any NAV value sets off a trigger.

Also, a client NAV trigger threshold table preferably exists for each basket value type generated by the BVU module 504. Thus, table

1314 stores a plurality of client Bid NAV trigger threshold values for the different baskets, wherein each client Bid NAV trigger threshold value is indexed by a basket identifier. Table 1316 stores a plurality of client Ask NAV trigger threshold values for the different baskets (each client Ask NAV trigger threshold value being indexed by a basket identifier). Table 1318 stores a plurality of client Last NAV trigger threshold values for the different baskets (each client Last NAV trigger threshold value being indexed by a basket identifier). Table 1320 stores a plurality of client Bid-Tick NAV trigger threshold values for the different baskets (each client Bid-Tick NAV trigger threshold value being indexed by a basket identifier). Table 1322 stores a plurality of client Ask+Tick NAV trigger threshold values for the different baskets (each client Ask+Tick NAV trigger threshold value being indexed by a basket identifier).

Preferably, tables 1302, 1304, 1306, 1308, 1310, 1314, 1316, 1318, 1320, and 1322 are stored in available on-chip memory for the reconfigurable logic device 102. However, it should be noted that these tables could alternatively be stored on any memory resource within or accessible to coprocessor 140.

Also, it should be noted that the client NAV tables and the client NAV trigger threshold tables can be optionally consolidated if desired by a practitioner of this embodiment of the invention. In doing so, the records in the client Bid NAV table 1302 would be merged with the records in the client Bid NAV trigger threshold table 1314 such that each record comprises the client Bid NAV and the client Bid NAV trigger threshold value. Similarly, table 1304 could be merged with table 1316, table 1306 with table 1318, table 1308 with table 1320, and table 1310 with table 1322.

PET module 1200 preferably employs a plurality of parallel computing paths to determine whether any of the NAV types within each basket event set off a client trigger.

A first computing path is configured to perform a lookup in the client Bid NAV table 1302 and a lookup in the client Bid NAV trigger

threshold table 1314 based on the current updated basket event's Basket ID to thereby retrieve a client Bid NAV and a client Bid NAV trigger threshold value. The client Bid NAV serves as a trigger against which the Bid NAV within the updated basket event is compared. The client Bid NAV trigger threshold value is then used as the criteria to judge whether the trigger has been set off. NAV-to-trigger comparison logic 1312 then operates perform this comparison and determine whether the trigger has been set off, wherein a Bid NAV trigger flag indicates the result of this determination.

Additional computing paths perform these operations for the updated basket event's Ask NAV, Last NAV, Bid-Tick NAV, and Ask+Tick NAV to produce Ask NAV, Last NAV, Bid-Tick NAV, and Ask+Tick NAV trigger flags, respectively, as shown in Figure 13.

Trigger detection logic 1324 receives the different NAV trigger flags produced by the computing paths. This logic 1324 is configured to determine whether any of the trigger flags for a given updated basket event are high. Preferably, the trigger detection logic 1324 is configured pass an updated basket event as an output if any of the flags for that updated basket event are high. As such, it can be seen that the PET module 1200 serves as a filter which passes updated basket events that are deemed noteworthy according to client-defined criteria.

Figure 14 illustrates an exemplary embodiment for the comparison logic 1312. A subtractor 1400 operates to subtract the retrieved client NAV from the updated basket event's NAV (NAV^{new}). A comparator 1404 then compares the difference between NAV^{new} and the client NAV with the retrieved client trigger threshold to determine whether the NAV trigger flag should be set. If the difference exceeds the threshold value, then comparator preferably sets the NAV trigger flag to high.

Thus, a client may want to be notified whenever the Bid NAV of Basket 145 changes by $0.05, whenever the Ask NAV for Basket 145 changes by $0.03, or whenever the Last NAV for Basket 145 changes by $0.02. In such an instance, the client Bid NAV trigger threshold

value in table 1314 for Basket 145 can be set equal to $0.05, the client Ask NAV trigger threshold value in table 1316 for Basket 145 can be set equal to $0.03, and the client Last NAV trigger threshold value in table 1318 for Basket 145 can be set equal to $0.02. If the new bid NAV, Ask NAV, and Last NAV values within the updated basket event for Basket 145 are $20.50, $20.40, and $20.20 respectively, while the stored values for the client NAVs of Basket 145 are $20, $20.42, and $20.19 respectively, then the comparison logic will operate to set the Bid NAV trigger flag high, the Ask NAV trigger flag low, and the Last NAV trigger flag high.

However, it should be noted that the comparison logic 1312 could be configured to determine whether a client trigger has been set off in any of a number of ways. For example, the client threshold values can be expressed in terms of percentages if desired (with corresponding adjustments in the comparison logic to compute the percentage difference between the new NAV and the client NAV). Also, the trigger condition could be based on a tracking instrument that is different than the subject NAV. For example, the trigger condition for a given basket can be defined in relation to a stored price for the DJIA. The client NAV tables would then store a DJIA price for that basket, and that basket's NAV prices would be compared against the retrieved DJIA price to determine whether the trigger threshold has been reached.

It should be noted that the PET module 1200 can be configured to detect arbitrage conditions by appropriately populating the client NAV tables and the client trigger threshold tables. For example, with ETFs, a client NAV table can be populated with the current share price of the ETF itself (T_{ETF}). Thus, the client bid NAV table could be populated with the current bid price for a share of the ETF, the client ask NAV table could be populated with the current ask price for a share of the ETF, the client last NAV table could be populated with the current last price for a share of the ETF, etc. By populating the client NAV tables in this manner, the PET module can detect when there is a discrepancy between the share price of the ETF and the net asset

value of the ETF's constituent financial instruments. Such a discrepancy can be taken advantage of by a trader, as explained above.

Thus, a client application that is informed by the pipeline of a discrepancy between T_{ETF} and the ETF's NAV can place one or more trades on one or more financial markets to realize a profit from the detected arbitrage condition. For example, if the ETF's NAV is less than the ETF's share price, a client application can execute the following steps:

- Assemble a basket by purchasing the shares of the financial instruments in the ETF's portfolio at a total cost approximately equal to the NAV;
- Create an ETF share by exchanging the assembled basket with the fund owner; and
- Sell the ETF on the exchange at a price approximately equal to T_{ETF}.

Furthermore, if the ETF's NAV is greater than the ETF's share price, a client application can execute the above steps in reverse. That is, a trader can buy one or more shares of the subject ETF from the market, exchange the ETF for shares of its constituent financial instruments, and then sell those shares of the constituent financial instruments on the market.

It should also be noted that the trigger thresholds in the client trigger threshold tables can be populated with values that take into account any expected transactional costs involved in the process of making such trades to take advantage of the price discrepancy, as explained below. Thus, if the sales price for the assembled ETF exceeds the aggregate prices of the financial instrument shares purchased to assemble the basket and the related transactional costs in doing so (or the sales prices for the constituent financial instruments exceeds the purchase price of the ETF and the related transactional costs), a trader who uses the client application can yield a profit by way of arbitrage. For such actions to work, the inventors note that time is of the essence, and it is believed that the low latency nature of the disclosed pipeline greatly increases a

trader's ability to recognize and profit from such arbitrage conditions as they arise in a constantly fluctuating market.

Moreover, it should be noted that the PET module 1200 can filter updated basket events for a plurality of different clients. In this manner, each subscribing client can define its own set of reference values and trigger thresholds. To accomplish such a feature, each updated basket event can be compared in parallel with a plurality of criteria for different clients via a plurality of client-specific PET modules 1500, as shown in Figure 15. It should also be noted that the client-specific PET modules 1500 (each of which would be a client-specific replicant of PET module 1200 of Figure 13) can be arranged in series (wherein each client-specific PET module would optionally add a client-specific flag to each updated basket event to thereby indicate whether that client's triggering criteria were met). Furthermore, the client-specific PET modules 1500 could also be arranged in a hybrid parallel/series arrangement if desired.

Figure 16 depicts an embodiment wherein pipeline 500 includes an event generator module 1600 in communication with the output from the PET module 1200. The event generator module 1600 is configured to process each incoming triggered basket event to construct a normalized output message event which contains the computed new NAVs within the triggered basket event.

Figure 17 depicts an exemplary embodiment for the event generator module 1600. Preferably, the event generator module 1600 is configured to insert a bit string into each output event 1704 which defines an interest list for that event, wherein the interest list identifies each client that is to be notified of the event. Preferably, this interest list takes the form of a bit vector, wherein each bit position corresponds to a different client. For any bit positions that are high, this means that the client corresponding to that bit position should be notified.

An interest list table 1700 preferably stores such a bit vector for each basket. As the event generator module 1600 receives a triggered basket event from the PET module 1200, the event generator

module 1600 performs a lookup in the interest list table 1700 based on the Basket ID within the triggered basket event. As a result of this lookup, the interest list vector for that Basket ID is retrieved. A formatter 1702 thereafter builds an output message event 1704 from the retrieved interest list vector 1706 and the information within the received triggered basket event, as shown in Figure 17. Formatter 1702 is preferably configured to add a header 1708 and trailer 1710 to the output event 1704 such that the output event 1704 can be properly interpreted by a client recipient (or recipients) of the output event 1704. The format for such output events can comply with any of a number of messaging protocols, both well-known (e.g., FAST and FIX) and proprietary. As indicated, the value of the bit positions in the interest list vector 1706 of the output event 1704 will define which clients are to be recipients of the event 1704.

The interest list table 1700 is preferably located within available on-chip memory for the reconfigurable logic device 102. However, it should be noted that table 1700 could alternatively be stored on any memory resource within or accessible to coprocessor 140. Also, it should be noted that formatter 1702 may optionally be configured to access one or more data tables (not shown; either on-chip or off-chip) to define the appropriate fields for the output event 1704.

It should also be noted that the event generator module 1600 may be configured to store statistics such as the number of triggered basket events it receives, the number of output events it produces, and the average inter-event time for each triggered basket event.

Figure 18 depicts an embodiment wherein pipeline 500 includes a message qualifier filter module 1800 in communication with the input to the BAL module 502. The message qualifier filter module 1800 is configured to process incoming message events and filter out any message events that are not of interest to the BCE. Messages of interest will generally include messages which may affect NAV prices. Examples of messages that a practitioner of this embodiment of the invention may want to filter are messages that do not contain a bid,

ask, or last price for a financial instrument. For example, a practitioner may desire to filter out messages which merely correct the previous day's closing prices and messages relating to trades that do not contribute to the daily volume.

Figure 19 depicts an exemplary message qualifier filter module 1800 that can be employed in the pipeline 500 of Figure 18. A table 1900 stores a plurality of qualifiers. Module 1800 is preferably configured to compare one or more fields of each incoming message (such as the quote and trade qualifier fields of the incoming messages) against the different qualifiers stored by table 1900. Each message qualifier is typically a small binary code which signals a condition. Each qualifier in table 1900 preferably maps to an entry in table 1902 which contains a pass/drop flag in each address. Thus, if a match is found any of the message fields with respect to qualifier [1] from table 1900, then this match would map to the entry in position n-2 of table 1902, and the bit value in position n-2 of table 1902 would be retrieved and provided as an input to AND logic 1904. Message qualifiers from table 1900 can be directly addressed to entries in table 1902. However, if the size of table 1900 were to greatly increase, it should be noted that hashing could be used to map qualifiers to pass/drop flags.

If any of the inputs to AND logic 1904 are high, then this will cause the pass/drop signal 1906 to follow as high. Logic 1908 is configured to be controlled by signal 1906 when deciding whether to pass incoming messages to the output. Thus, if any of the matching qualifiers from table 1900 maps to a high bit in table 1902, this will result in the subject message being passed on to the output of the message qualifier filter module 1800.

In this manner, each qualifier in the qualifier table may define a message criteria which, if found in an incoming message, will result in that incoming message being passed along or blocked from propagating downstream in pipeline 500. Examples of message qualifiers which can be stored by table 1900 to indicate that a message should be blocked would be indicators in messages that

identify the message as any of a non-firm quote, penalty bid, withdrawn quote, held trade, and out of sequence trade.

Tables 1900 and 1902 are preferably located within available on-chip memory for the reconfigurable logic device 102. However, it should be noted that these tables could alternatively be stored on any memory resource within or accessible to coprocessor 140.

It should also be noted that the incoming messages to pipeline 500 are preferably normalized messages in that the message fields are formatted in a manner expected by the pipeline 500. Such normalization may optionally have been already been performed prior to the time that the messages reach pipeline 500. However, optionally, pipeline 500 may include a message normalization module 2000 as shown in Figure 20 to provide the normalization functions. Such normalization preferably includes functions such as symbol ID mapping (wherein each message is assigned with a symbol ID (preferably a fixed-size binary tag) that uniquely identifies the financial instrument which is the subject of the message) and GEID mapping (wherein each message is assigned with a GEID (preferably also a fixed-size binary tag) that uniquely identifies the exchange corresponding to the message). FAM modules to perform such normalization functions are described in the above-referenced and incorporated U.S. patent application 11/765,306.

With respect to administering the BCE 400, it should be noted that baskets and trigger conditions can be created, modified, and/or deleted in any of a number of ways. For example, a system administrator and/or client application can be given the ability to define baskets and/or trigger conditions

To create and/or modify a basket, appropriate entries and allocations will be needed for the new/modified basket in the various tables described herein in connection with pipeline 500. A system administrator may use a control interface to add/modify basket configurations. Client applications may submit basket definitions via an application programming interface (API) that runs on their local machine(s).

The BCE pipeline may be deployed on coprocessor 140 of a ticker plant platform 2100 as shown in Figure 21. Additional details regarding such a ticker plant platform 2100 can be found in the above-referenced and incorporated 11/765,306 patent application. In summary, the financial market data from the exchanges is received at the O/S supplied protocol stack 2102 executing in the kernel space on processor 112 (see Figures 1(a)-(b)). An upject driver 2104 delivers this exchange data to multi-threaded feed pre-processing software 2112 executing in the user-space on processor 112. These threads may then communicate data destined for the coprocessor 140 to the hardware interface driver software 2106 running in the kernel space.

Instructions from client applications may also be communicated to the hardware interface driver 2106 for ultimate delivery to coprocessor 140 to appropriately configure a BCE pipeline that is instantiated on coprocessor 140. Such instructions arrive at an O/S supplied protocol stack 2110 from which they are delivered to a request processing software module 2116. A background and maintenance processing software module 2114 thereafter determines whether the client application has the appropriate entitlement to add/change/modify a basket. If so entitled, the background and maintenance processing block 2114 communicates a command instruction to the hardware interface driver 2106 for delivery to the coprocessor to appropriately update the table entries in the BCE pipeline to reflect the added/changed/modified basket, as previously explained above. Deletions of baskets as well as additions/modifications/deletions of trigger conditions from the BCE pipeline can be performed via a like process, as indicated above.

The hardware interface driver 2106 then can deliver an interleaved stream of financial market data and commands to the coprocessor 140 for consumption thereby. Outgoing data from the coprocessor 140 returns to the hardware interface driver 2106, from which it can be supplied to MDC driver 2108 for delivery to the client connections (via protocol stack 2110) and/or delivery to the background and maintenance processing block 2114.

As discussed above, some financial instrument baskets such as ETFs may have some form of cash component to them. These cash components can be taken into account by the system in any of a number of ways.

For example, to insert a cash component into a basket's NAV, a control process can be configured to generate a synthetic event for delivery to the pipeline. A symbol ID would be assigned to a given cash account, and the price for that synthetic symbol ID would be $1 and the weight value *w* for that cash account would be set equal to the cash value of the cash account. The tables used by the BAL module would be updated to reflect this synthetic symbol ID and cause the cash account to impact the NAV for the basket. In a more complex scenario, events could be delivered to the pipeline so that the value of the cash account can fluctuate with the value of the pertinent currency on the foreign exchange market.

Another way of injecting a cash component of a basket into the system is to define the trigger threshold values in tables 1314, 1316, 1318, 1320, and 1322 such that the cash component amounts are built into those trigger threshold values.

Also, it should be noted that the BAL module (or the BVU module) can be configured to drop events where all of the delta prices for that event are zero.

Similarly, it should be noted that in the embodiment of Figure 9, the NAV update engine 950 is configured to compute updated NAVs even for NAV types whose corresponding delta prices are zero. That is, a current delta event being processed by the BVU module may contain a $0.03 value for the ΔBid price but zero values for the ΔAsk and ΔLast prices. With the embodiment of Figure 9, some of the different NAV compute logic engines 916 of the parallel paths would thus be used to compute a new NAV where there will not be a change relative to the old NAV (e.g., for the example given, the Ask NAV^{new}, Last NAV^{new}, and Ask+Tick NAV^{new} values would be unchanged relative to their old values). It is believed that this configuration will still yield effective efficiency given that most of the events processed by the

pipeline are expected to comprise quote activity. However, a practitioner is free to choose to increase the efficiency of the BVU module by not including any zero delta prices in the delta events and including a dynamic scheduler and memory controller which directs delta events to the appropriate computing path(s) within the NAV update engine 950 to more fully utilize all available computing paths within the NAV update engine 950.

It should also be noted that logic could be added to the BVU modules such that the NAV computations are also based on derived statistics such as the Volume Weighted Average Price (VWAP) and use other fields which may be available in a message, such as the difference in traded volume.

It should also be noted that the coprocessor 140 can be configured to perform computations in addition to those of the BCE if desired by a practitioner of an embodiment of the invention. For example, the coprocessor 140 can also employ a last value caching (LVC) module, as described in the above-referenced and incorporated 11/765,306 patent application.

It should be noted that a practitioner of an embodiment of the invention may choose to compute the value for Δj but not the value for NAV^{new}, in which case the BVU module need not produce any NAV^{new} values. In such instances, the trigger conditions used by the PET module 1200 can be built around variation in Δj rather than NAV^{new}. In such a situation, the client could optionally determine the NAV values itself from the Δj value, although a practitioner may find value in the Δj values themselves apart from their actual combination with an old NAV value to find a new NAV value.

Also, while the preferred embodiment for the BAL module 502 is configured to compute the delta values for the last/bid/ask prices in the incoming messages, it should be noted that the computation of such delta values can optionally be performed by one or more modules upstream from the BAL module 502 or the BCE pipeline 500 altogether. As such, the input messages to the BCE pipeline 500 may already

contain the ΔBid, ΔAsk, and/or ΔLast prices for the subject financial instruments. In such instances, table 608 would not need to store the previous bid/ask/last prices for each financial instrument, and BAL module 502 need not employ the logic necessary to compute those delta prices.

Furthermore, while in the preferred embodiment disclosed herein the coprocessor 140 comprises a reconfigurable logic device 102 such as an FPGA, it should be noted that the coprocessor 140 can be realized using other processing devices. For example, the coprocessor 140 may comprise graphics processor units (GPUs), general purpose graphics processors, chip multi-processors (CMPs), dedicated memory devices, complex programmable logic devices, application specific integrated circuits (ASICs), and other I/O processing components. Moreover, it should be noted that system 100 may employ a plurality of coprocessors 140 in either or both of a sequential and a parallel multi-coprocessor architecture.

Further still, while the inventors believe that special advantages exist in connection with using the pipeline disclosed herein to process financial information and compute updated values for financial instrument baskets, the inventors further note that the disclosed pipeline also provides latency advantages when processing non-financial data which pertains to baskets. For example, the basket calculation engine may be configured to compute the net asset value of the inventory held by a multi-national retail organization. Incoming events to the pipeline that correspond to product sales and/or product deliveries could thus be used to update stored values for each product, wherein these stored values correspond to inventory count values. A weight assigned to each product would correspond to a price value assigned to the product. Thus, a pipeline such as the one disclosed herein could be used to closely track the net inventory values of the organization's products. Triggers may then be used to notify management when inventory values swell or shrink by a given threshold.

Another example would be for a basket which is a collection of data points from some type of scientific experiment. Each data point may have associated values such as size, mass, etc., and these data points may arrive over time as streaming data from one or more monitoring stations. The pipeline can readily be configured to derive a size NV by computing a weighted sum of the sizes from the incoming data stream and/or a mass NV by computing a weighted sum of the masses from the incoming data stream.

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. Such modifications to the invention will be recognizable upon review of the teachings herein. Accordingly, the full scope of the present invention is to be defined solely by the appended claims and their legal equivalents.

Method and System for Low Latency Basket Calculation

Abstract of the Disclosure:

A basket calculation engine is deployed to receive a stream of data and accelerate the computation of basket values based on that data. In a preferred embodiment, the basket calculation engine is used to process financial market data to compute the net asset values (NAVs) of financial instrument baskets. The basket calculation engine can be deployed on a coprocessor and can also be realized via a pipeline, the pipeline preferably comprising a basket association lookup module and a basket value updating module. The coprocessor is preferably a reconfigurable logic device such as a field programmable gate array (FPGA).

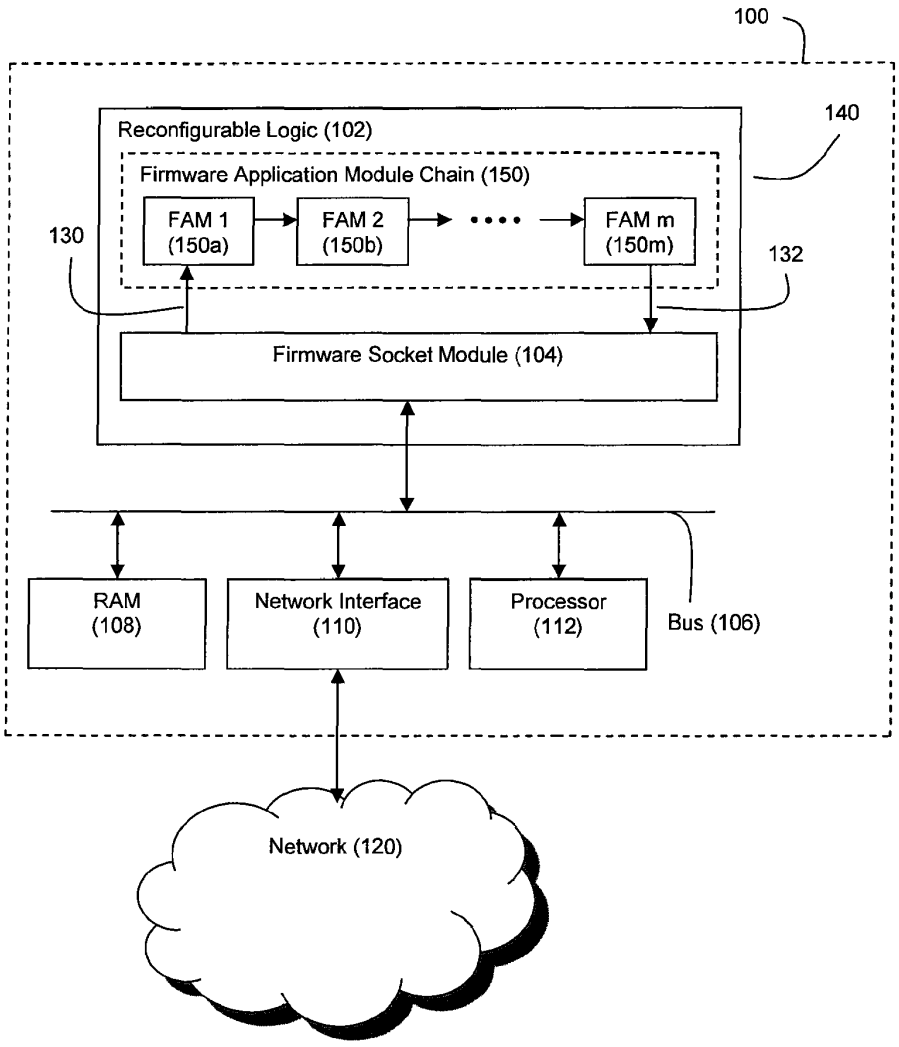

100
140
Reconfigurable Logic (102)
Firmware Application Module Chain (150)
FAM 1
(150a)
FAM 2
(150b)
FAM m
(150m)
130
132
Firmware Socket Module (104)
RAM
(108)
Network Interface
(110)
Processor
(112)
Bus (106)
Network (120)

Figure 1(a)

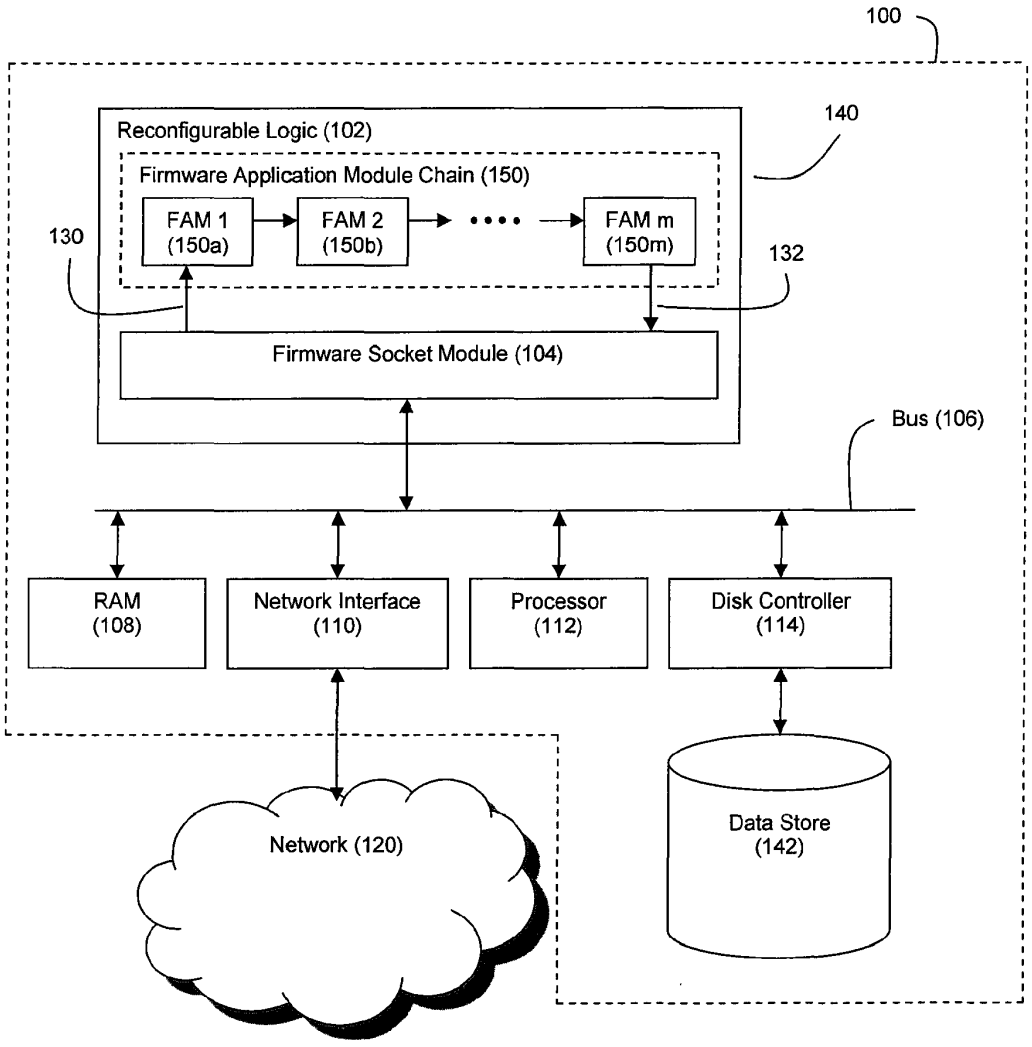

100
Reconfigurable Logic (102)
140
Firmware Application Module Chain (150)
FAM 1
(150a)
FAM 2
(150b)
FAM m
(150m)
130
132
Firmware Socket Module (104)
Bus (106)
RAM
(108)
Network Interface
(110)
Processor
(112)
Disk Controller
(114)
Network (120)
Data Store
(142)

Figure 1(b)

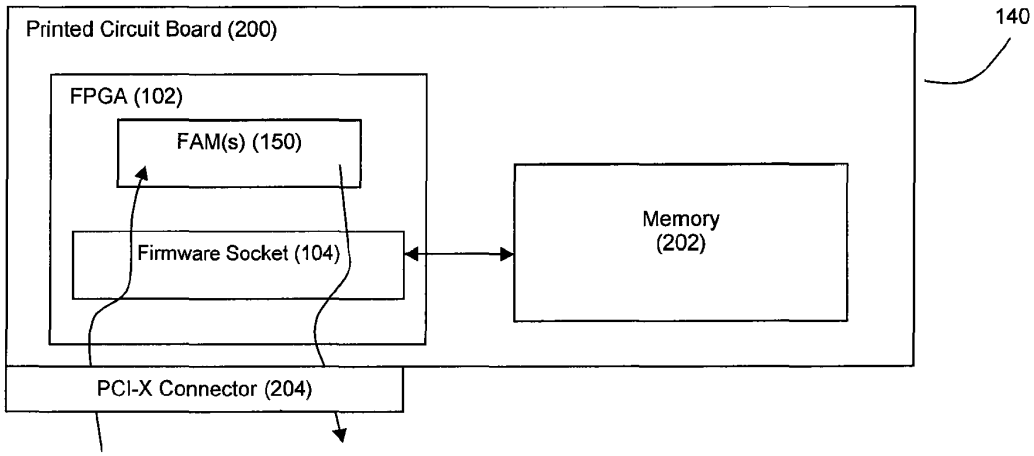


Figure 2(a)

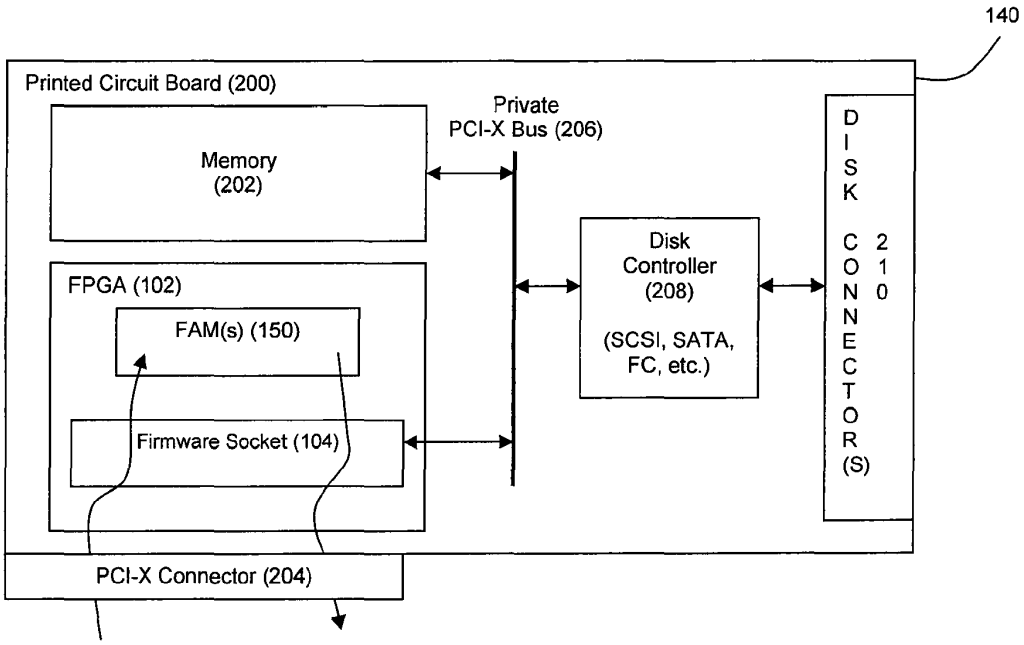


Figure 2(b)

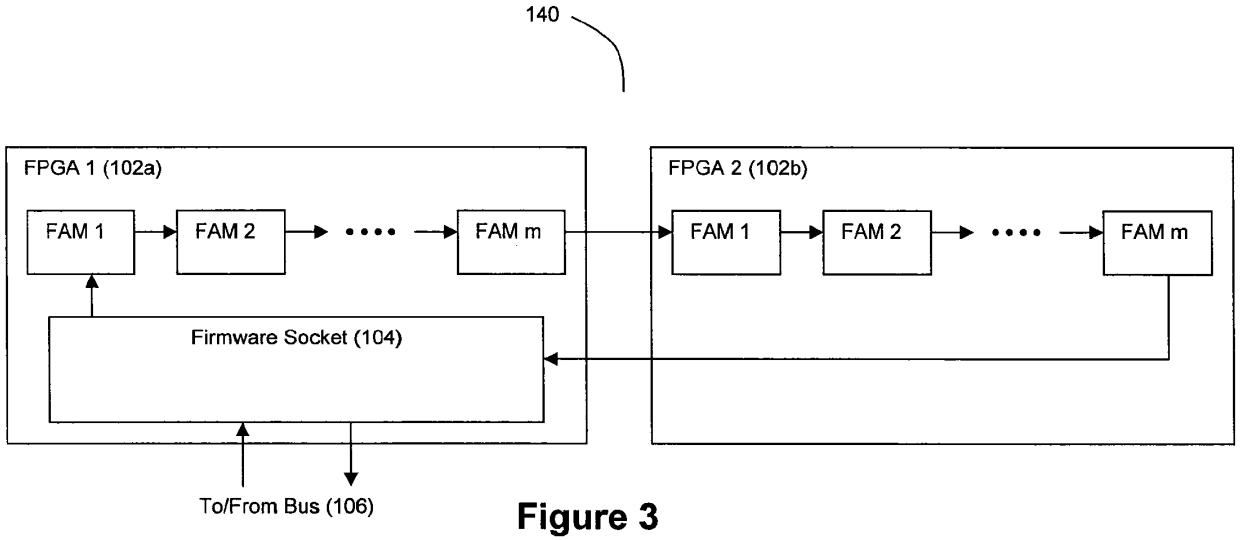


Figure 3

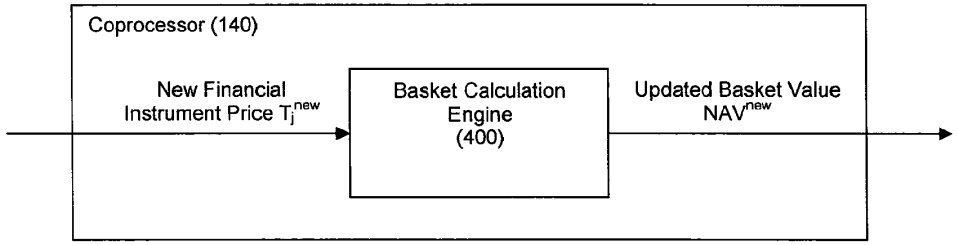


Figure 4(a)

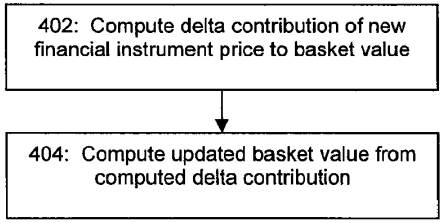


Figure 4(b)

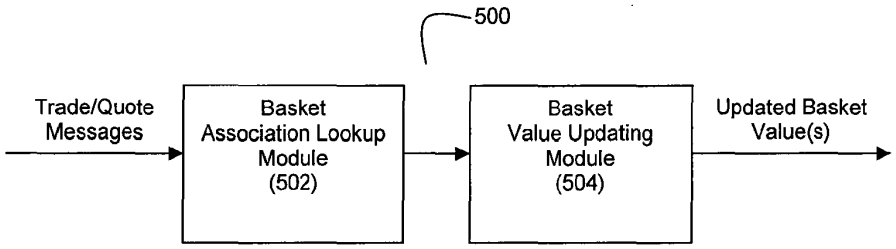


Figure 5

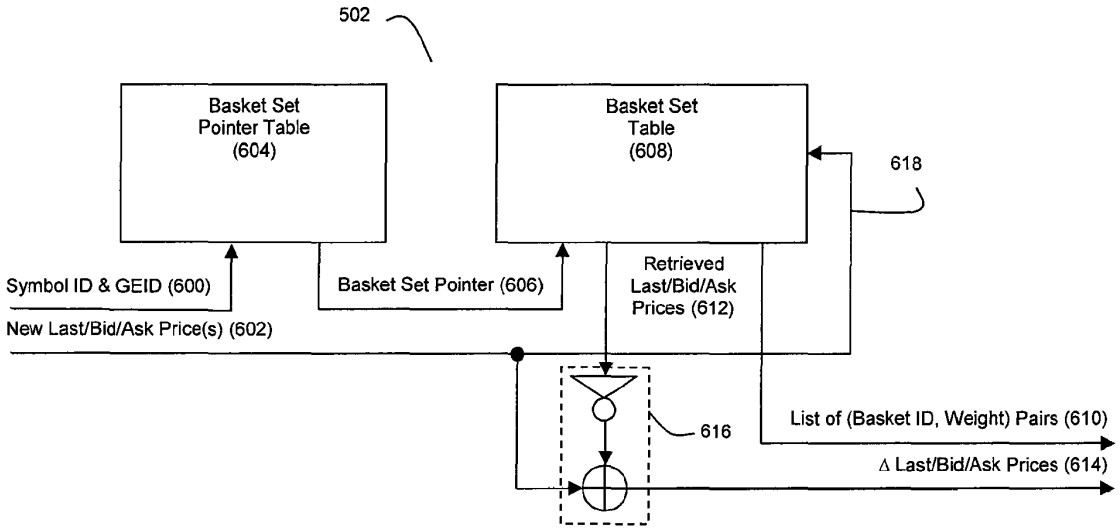


Figure 6

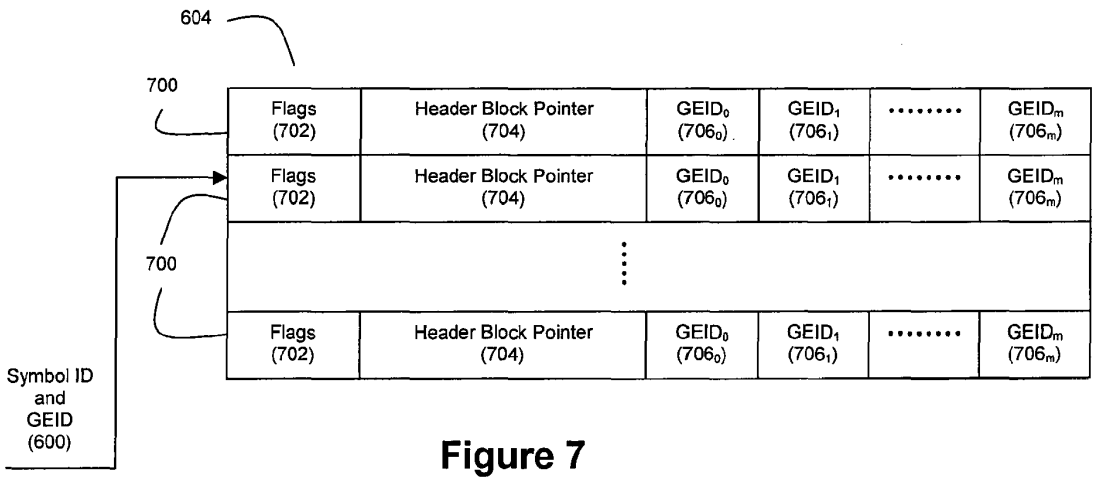


Figure 7

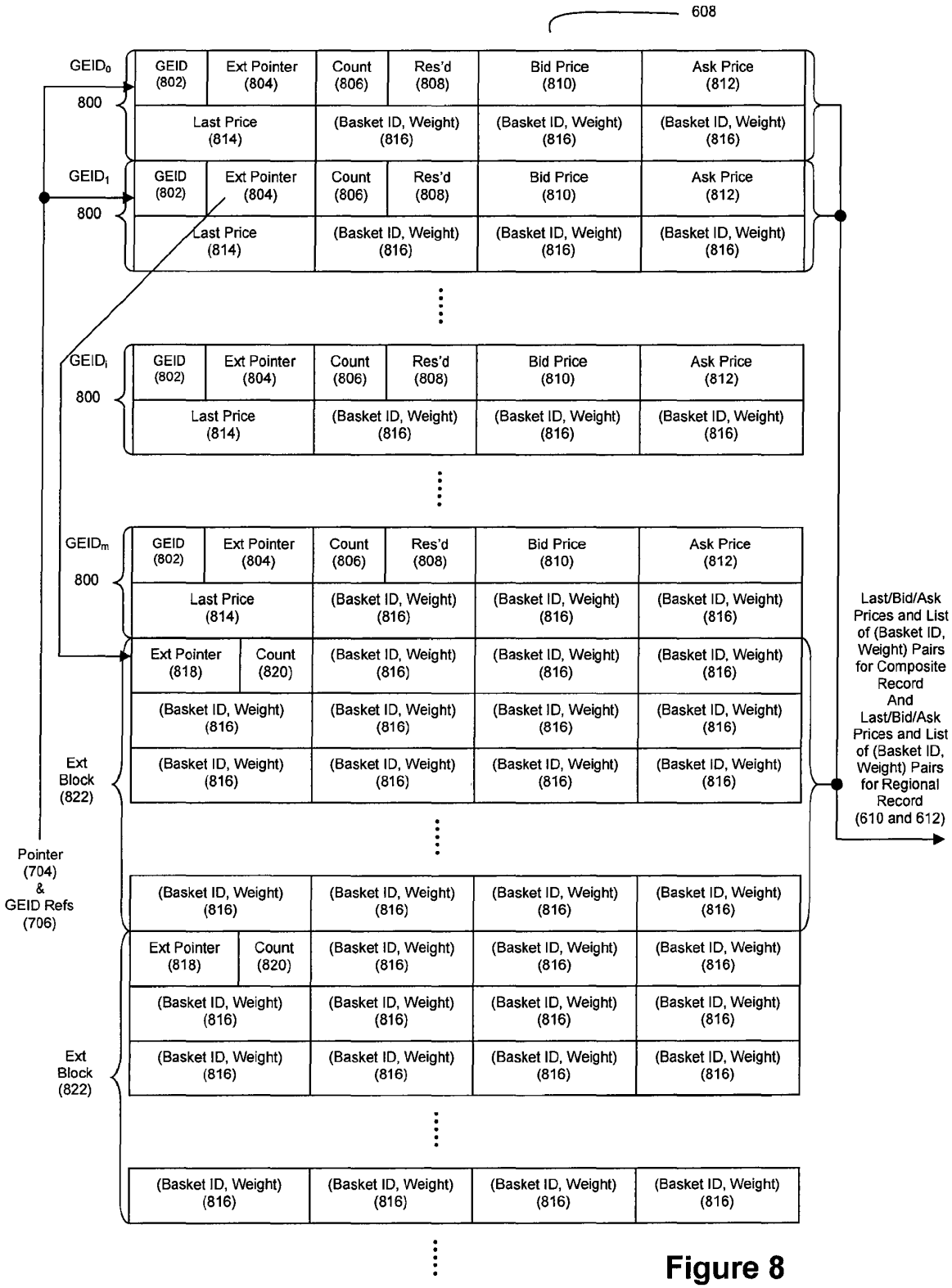

608
GEID0
800
GEID
(802)
Ext Pointer
(804)
Count
(806)
Res'd
(808)
Bid Price
(810)
Ask Price
(812)
Last Price
(814)
(Basket ID, Weight)
(816)
GEID1
GEIDi
GEIDm
Ext Pointer
(818)
Count
(820)
Ext Block
(822)
Pointer
(704)
&
GEID Refs
(706)
Last/Bid/Ask Prices and List of (Basket ID, Weight) Pairs for Composite Record
And
Last/Bid/Ask Prices and List of (Basket ID, Weight) Pairs for Regional Record
(610 and 612)

Figure 8

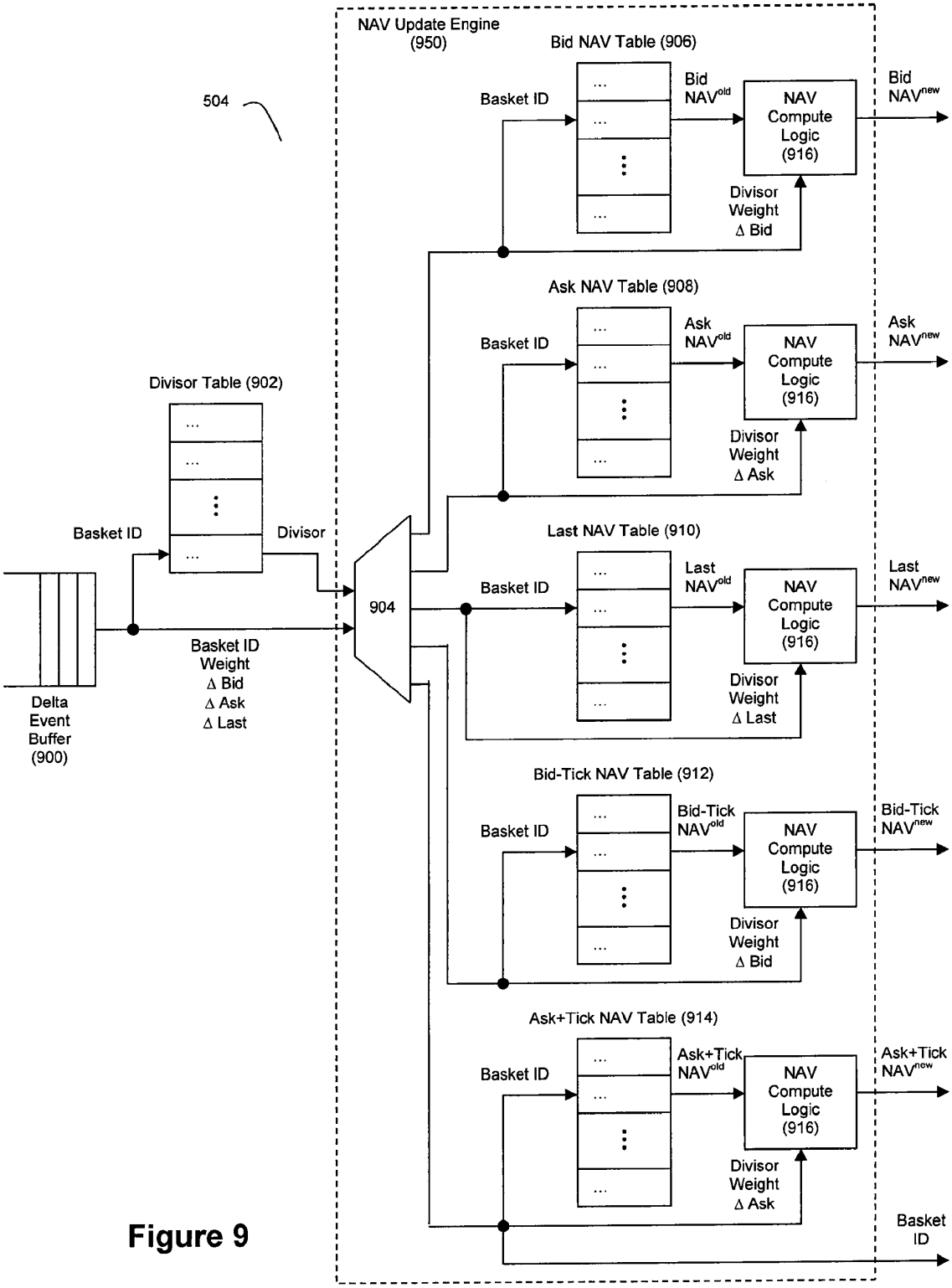

NAV Update Engine
(950)
504
Bid NAV Table (906)
Basket ID
Bid NAV^old
NAV Compute Logic (916)
Bid NAV^new
Divisor Weight Δ Bid
Ask NAV Table (908)
Basket ID
Ask NAV^old
NAV Compute Logic (916)
Ask NAV^new
Divisor Weight Δ Ask
Divisor Table (902)
Basket ID
Divisor
Last NAV Table (910)
Basket ID
Last NAV^old
NAV Compute Logic (916)
Last NAV^new
Divisor Weight Δ Last
904
Delta Event Buffer (900)
Basket ID Weight Δ Bid Δ Ask Δ Last
Bid-Tick NAV Table (912)
Basket ID
Bid-Tick NAV^old
NAV Compute Logic (916)
Bid-Tick NAV^new
Divisor Weight Δ Bid
Ask+Tick NAV Table (914)
Basket ID
Ask+Tick NAV^old
NAV Compute Logic (916)
Ask+Tick NAV^new
Divisor Weight Δ Ask
Basket ID

Figure 9

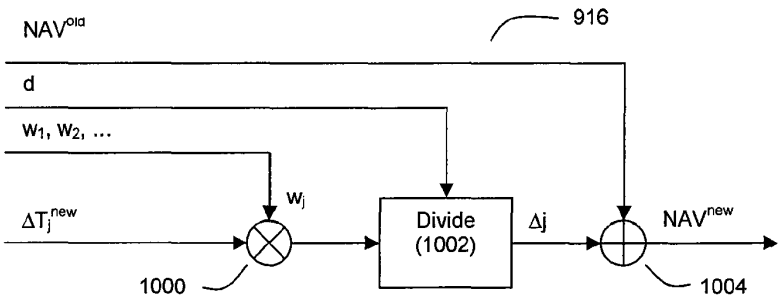


Figure 10(a)

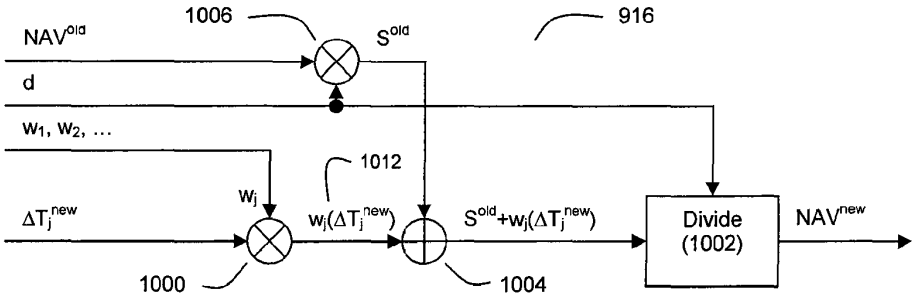


Figure 10(b)

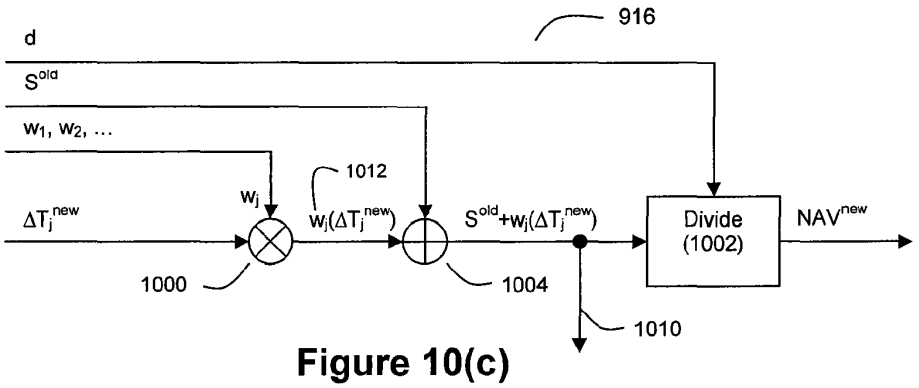


Figure 10(c)

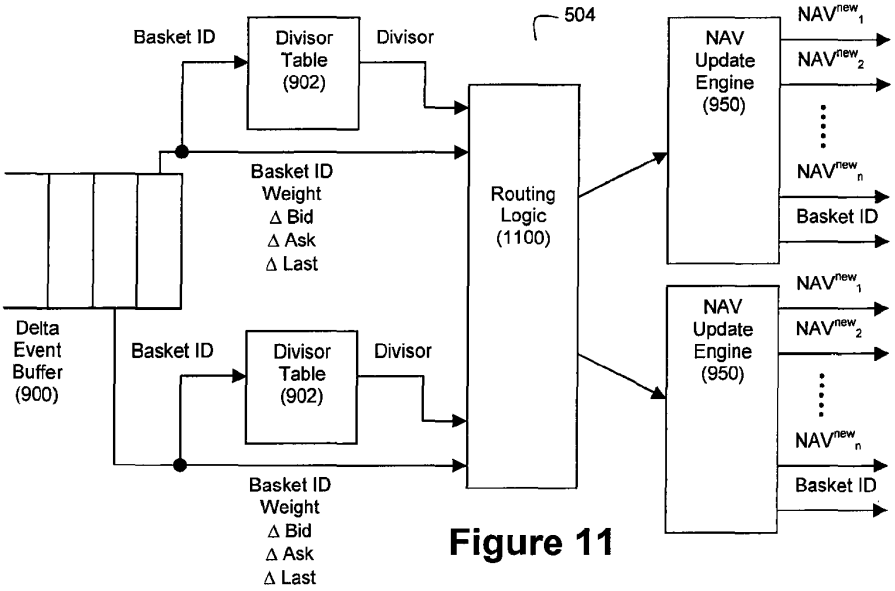


Figure 11

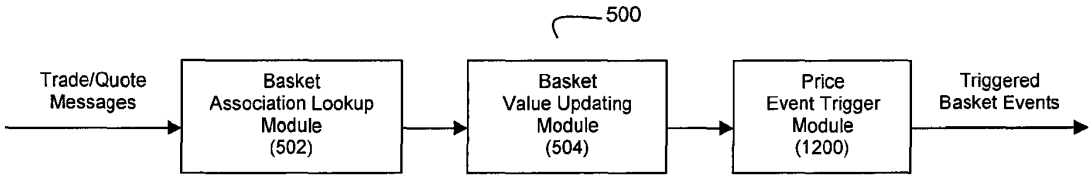


Figure 12

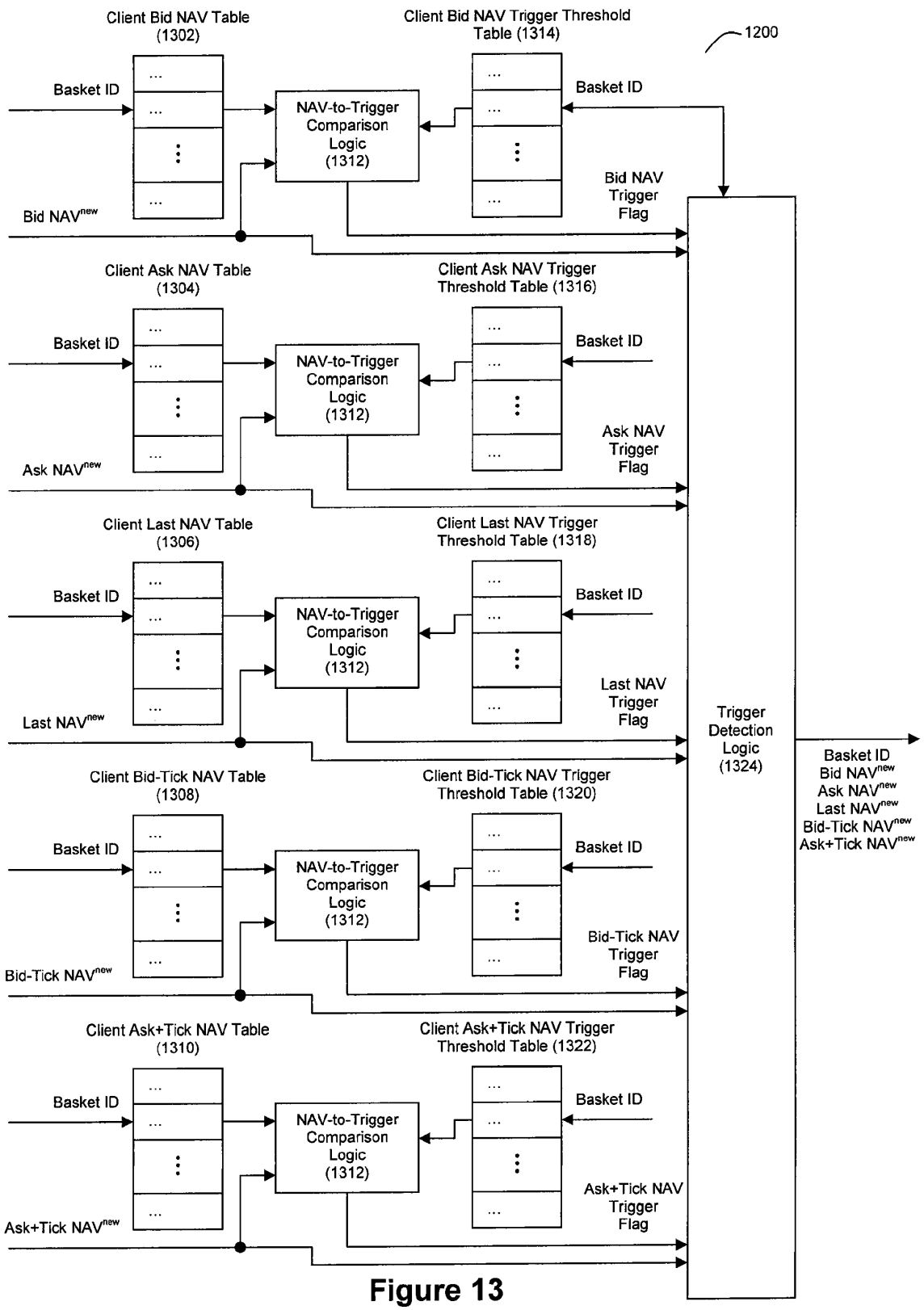

Client Bid NAV Table (1302)
Client Bid NAV Trigger Threshold Table (1314)
1200
Basket ID
NAV-to-Trigger Comparison Logic (1312)
Basket ID
Bid NAV Trigger Flag
Bid NAV^new
Client Ask NAV Table (1304)
Client Ask NAV Trigger Threshold Table (1316)
Basket ID
NAV-to-Trigger Comparison Logic (1312)
Basket ID
Ask NAV Trigger Flag
Ask NAV^new
Client Last NAV Table (1306)
Client Last NAV Trigger Threshold Table (1318)
Basket ID
NAV-to-Trigger Comparison Logic (1312)
Basket ID
Last NAV Trigger Flag
Last NAV^new
Trigger Detection Logic (1324)
Basket ID
Bid NAV^new
Ask NAV^new
Last NAV^new
Bid-Tick NAV^new
Ask+Tick NAV^new
Client Bid-Tick NAV Table (1308)
Client Bid-Tick NAV Trigger Threshold Table (1320)
Basket ID
NAV-to-Trigger Comparison Logic (1312)
Basket ID
Bid-Tick NAV Trigger Flag
Bid-Tick NAV^new
Client Ask+Tick NAV Table (1310)
Client Ask+Tick NAV Trigger Threshold Table (1322)
Basket ID
NAV-to-Trigger Comparison Logic (1312)
Basket ID
Ask+Tick NAV Trigger Flag
Ask+Tick NAV^new

Figure 13

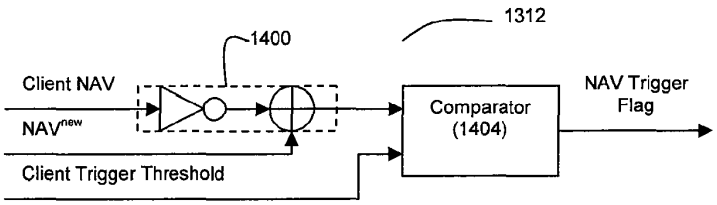


Figure 14

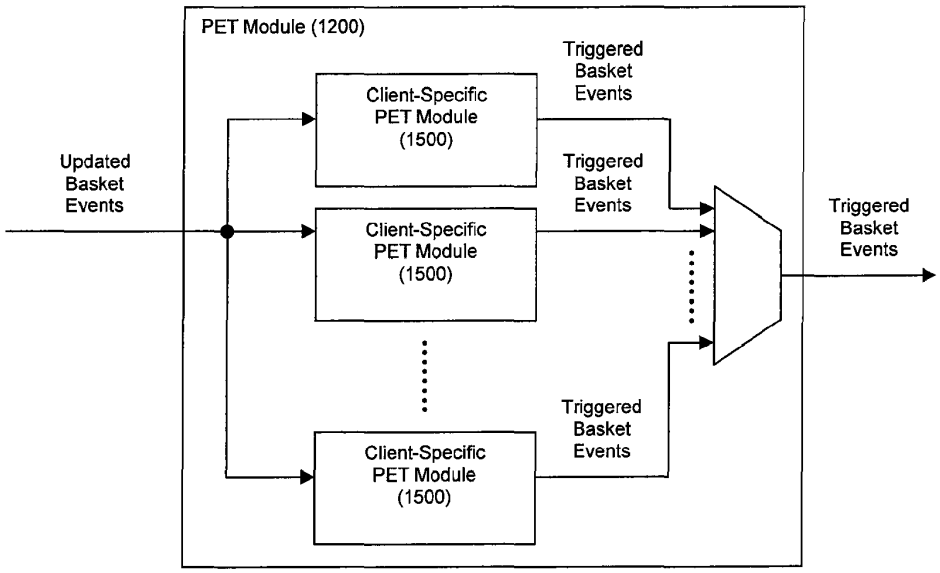


Figure 15

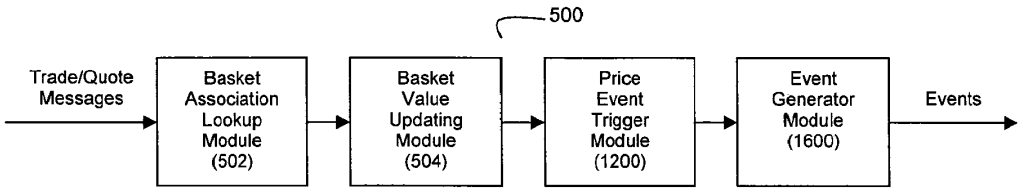


Figure 16

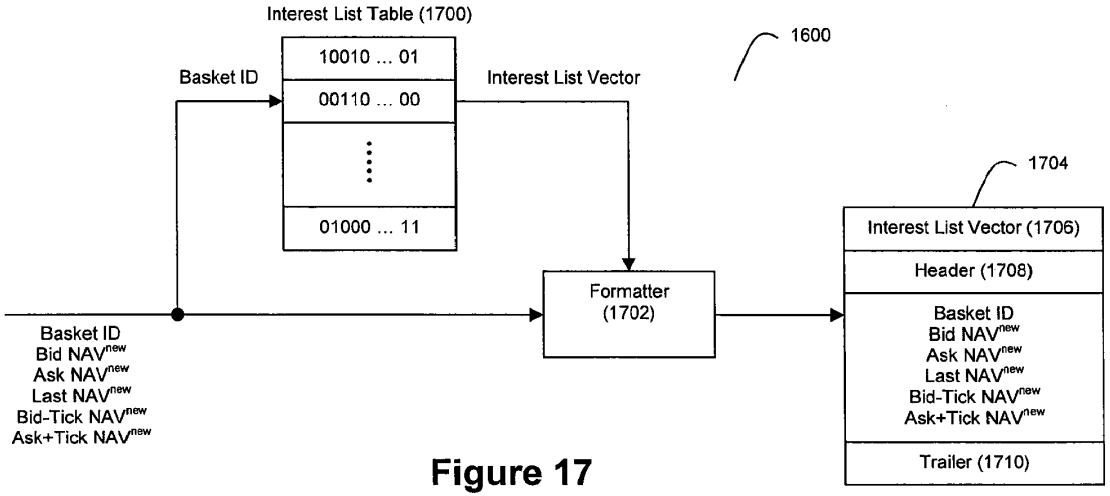


Figure 17

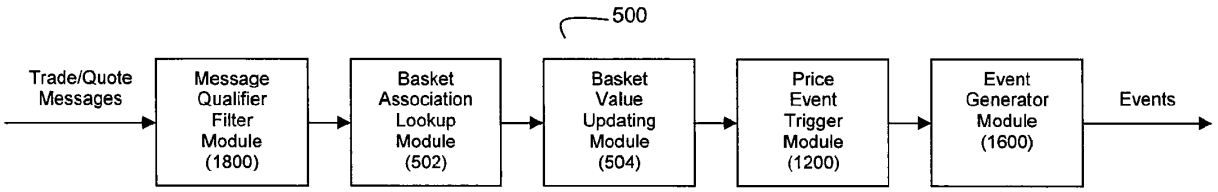


Figure 18

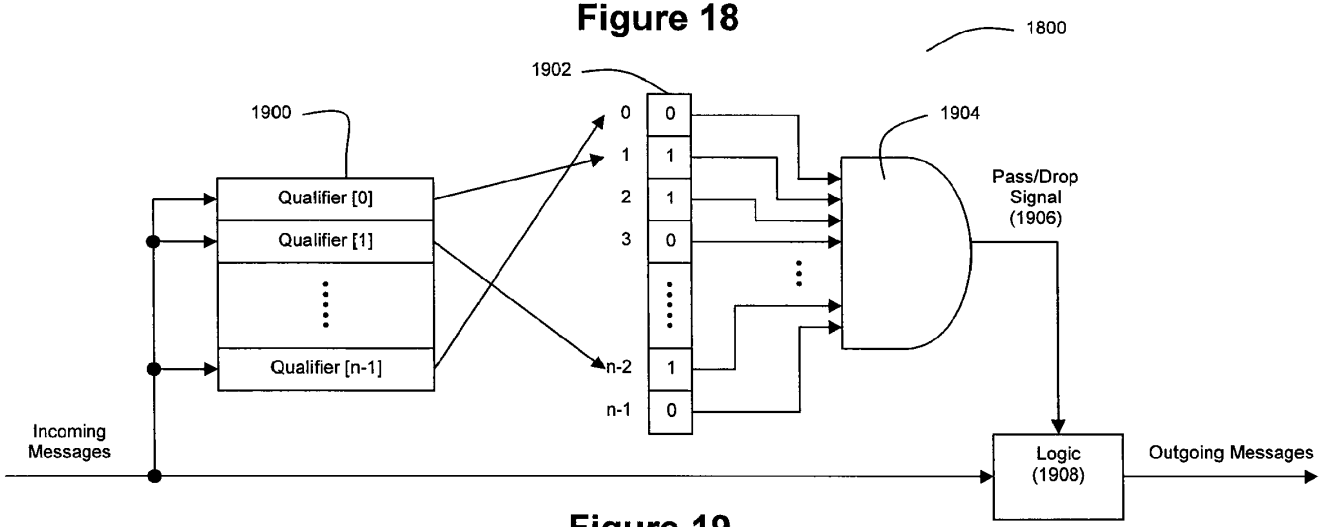


Figure 19

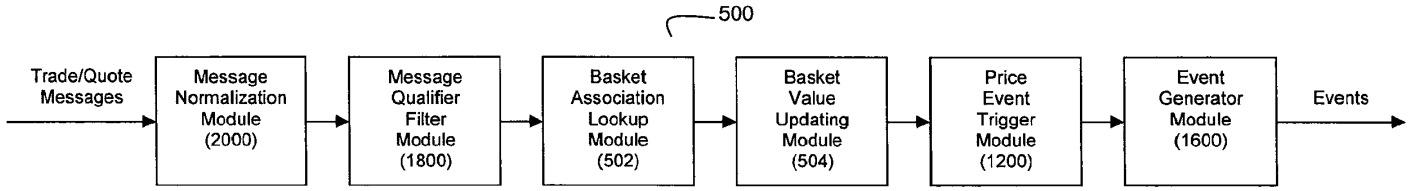


Figure 20

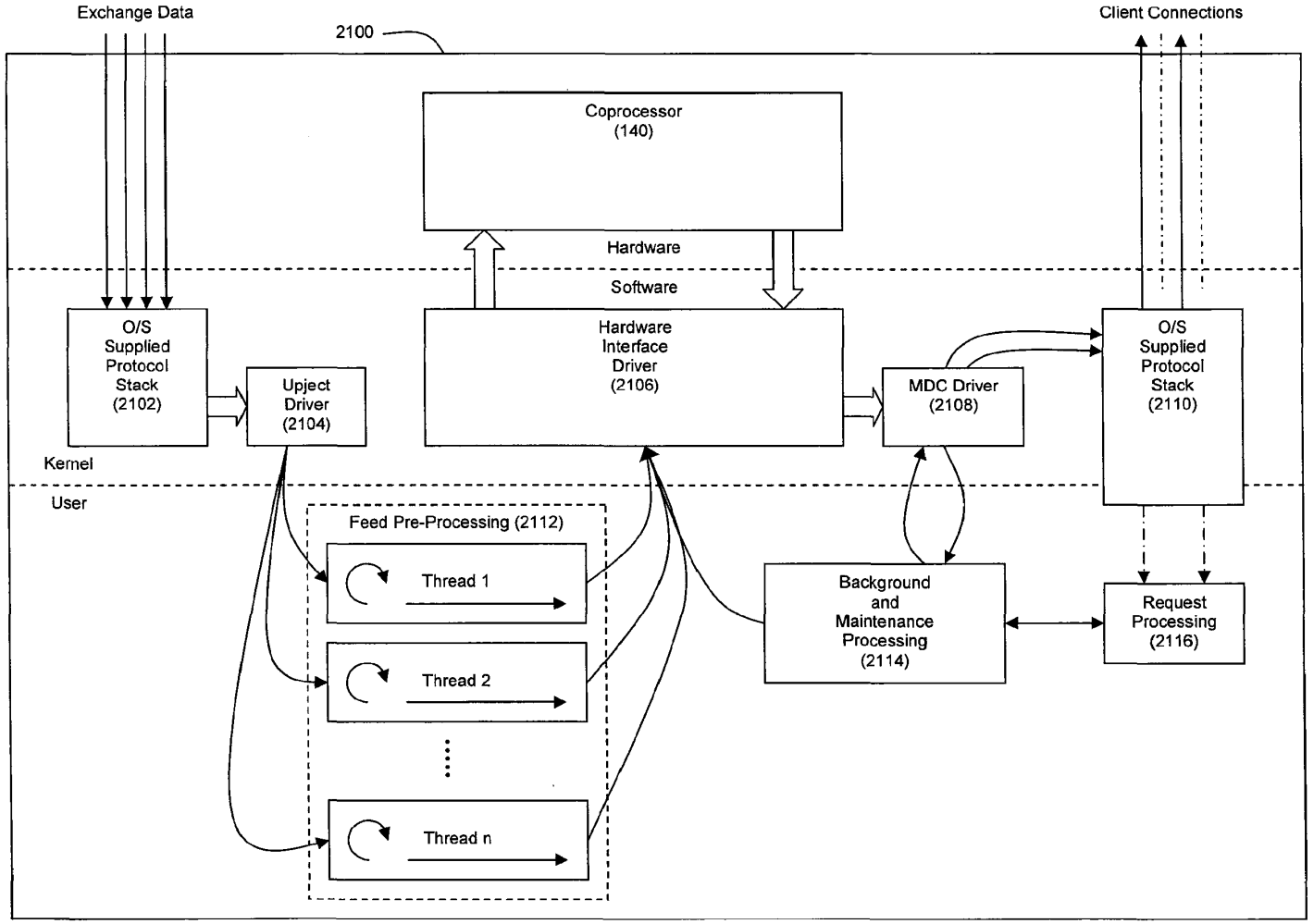

Exchange Data
2100
Client Connections
Coprocessor
(140)
Hardware
Software
O/S
Supplied
Protocol
Stack
(2102)
Upject
Driver
(2104)
Hardware
Interface
Driver
(2106)
MDC Driver
(2108)
O/S
Supplied
Protocol
Stack
(2110)
Kernel
User
Feed Pre-Processing (2112)
Thread 1
Thread 2
Thread n
Background
and
Maintenance
Processing
(2114)
Request
Processing
(2116)

Figure 21

Electronic Acknowledgement Receipt	
EFS ID:	4459804
Application Number:	61122673
International Application Number:	
Confirmation Number:	1019
Title of Invention:	Method and Apparatus for High-Speed Processing of Financial Market Depth Data
First Named Inventor/Applicant Name:	David E. Taylor
Customer Number:	21888
Filer:	Benjamin Lowell Volk/Laurie Poss
Filer Authorized By:	Benjamin Lowell Volk
Attorney Docket Number:	44826-80492
Receipt Date:	15-DEC-2008
Filing Date:	
Time Stamp:	19:37:58
Application Type:	Provisional

Payment information:

Submitted with Payment	yes
Payment Type	Deposit Account
Payment was successfully received in RAM	$245
RAM confirmation Number	4874
Deposit Account	200823
Authorized User	
The Director of the USPTO is hereby authorized to charge indicated fees and credit any overpayment as follows: Charge any Additional Fees required under 37 C.F.R. Section 1.17 (Patent application and reexamination processing fees) Charge any Additional Fees required under 37 C.F.R. Section 1.21 (Miscellaneous fees and charges)	

File Listing:

Document Number	Document Description	File Name	File Size(Bytes)/ Message Digest	Multi Part /.zip	Pages (if appl.)
1	Application Data Sheet	ADS.pdf	1546563 0f432fd3c0aec15a159aaac1d6599363d/a8c937	no	7
Warnings:					
Information:					
2	Specification	Combine.pdf	6208074 1f70897a21c9a447f7783ceb66424eb0e9bd7f6b	no	135
Warnings:					
Information:					
3	Drawings-only black and white line drawings	Drawings.PDF	143465 5285ff93be3468d389837ee2d273f9d1ee04c48f	no	5
Warnings:					
Information:					
4	Fee Worksheet (PTO-06)	fee-info.pdf	31451 0f4d648b265a443f24d771a40195da50e859749e	no	2
Warnings:					
Information:					
		Total Files Size (in bytes):	7929553		

This Acknowledgement Receipt evidences receipt on the noted date by the USPTO of the indicated documents, characterized by the applicant, and including page counts, where applicable. It serves as evidence of receipt similar to a Post Card, as described in MPEP 503.

New Applications Under 35 U.S.C. 111
If a new application is being filed and the application includes the necessary components for a filing date (see 37 CFR 1.53(b)-(d) and MPEP 506), a Filing Receipt (37 CFR 1.54) will be issued in due course and the date shown on this Acknowledgement Receipt will establish the filing date of the application.

National Stage of an International Application under 35 U.S.C. 371
If a timely submission to enter the national stage of an international application is compliant with the conditions of 35 U.S.C. 371 and other applicable requirements a Form PCT/DO/EO/903 indicating acceptance of the application as a national stage submission under 35 U.S.C. 371 will be issued in addition to the Filing Receipt, in due course.

New International Application Filed with the USPTO as a Receiving Office
If a new international application is being filed and the international application includes the necessary components for an international filing date (see PCT Article 11 and MPEP 1810), a Notification of the International Application Number and of the International Filing Date (Form PCT/RO/105) will be issued in due course, subject to prescriptions concerning national security, and the date shown on this Acknowledgement Receipt will establish the international filing date of the application.

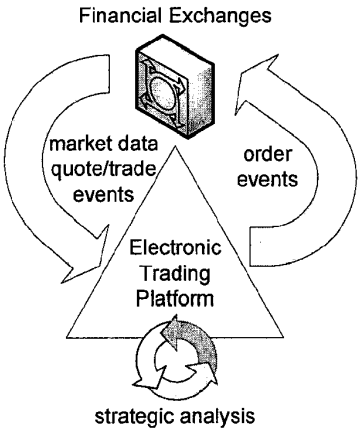


Figure 1

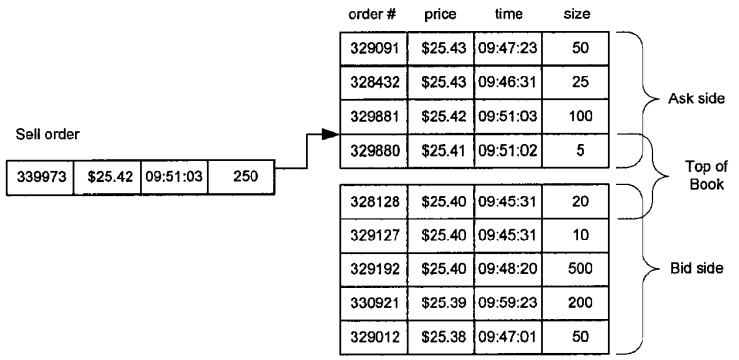


Figure 2

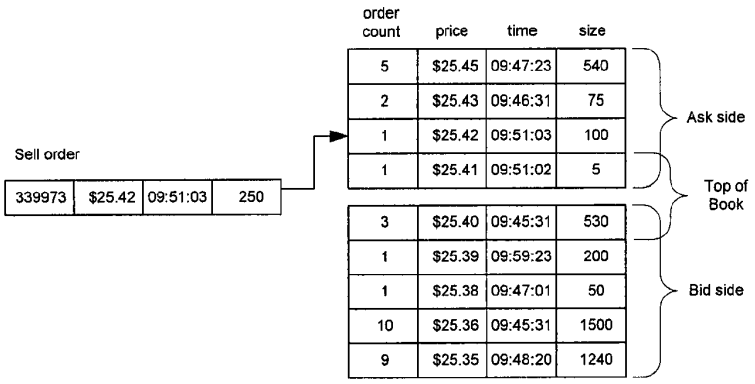


Figure 3

BID					ASK				
Price	Time	Size	Exch	Order ID	Price	Time	Size	Exch	Order ID
$25.43	9:47:23	25	NASD	N-B	$25.44	9:50:57	100	NASD	N-F
$25.42	9:51:03	100	NASD	N-C	$25.45	9:59:23	200	NASD	N-D
$25.42	9:59:49	75	NASD	N-E	$25.46	9:45:31	10	NASD	N-A
					$25.46	9:51:17	30	NASD	N-G

Figure 4

BID					ASK				
Price	Time	Size	Exch	Count	Price	Time	Size	Exch	Count
$ 25.43	9:47:23	25	NASD	1	$ 25.44	9:50:57	100	NASD	1
$ 25.42	9:59:49	175	NASD	2	$ 25.45	9:59:23	200	NASD	1
					$ 25.46	9:51:17	40	NASD	2

Figure 5

BID					ASK				
Price	Time	Size	Exch	Count	Price	Time	Size	Exch	Count
$ 25.43	9:46:31	50	ARCA	1	$ 25.44	9:47:01	50	BATS	1
$ 25.43	9:47:23	25	NASD	1	$ 25.44	9:50:57	100	NASD	1
$ 25.42	9:59:49	175	NASD	2	$ 25.45	9:49:43	125	ARCA	2
$ 25.41	9:41:48	5	ARCA	1	$ 25.45	9:59:23	200	NASD	1
$ 25.41	9:42:12	40	BATS	1	$ 25.46	9:48:20	500	ARCA	1
					$ 25.46	9:45:31	20	BATS	1
					$ 25.46	9:51:17	40	NASD	2

Figure 6

BID					ASK				
Price	Time	Size	Exch	Count	Price	Time	Size	Exch	Count
$ 25.43	9:47:23	75	-	2	$ 25.44	9:50:57	150	-	2
$ 25.42	9:59:49	175	-	2	$ 25.45	9:59:23	325	-	3
$ 25.41	9:42:12	45	-	2	$ 25.46	9:51:17	560	-	4

Figure 7

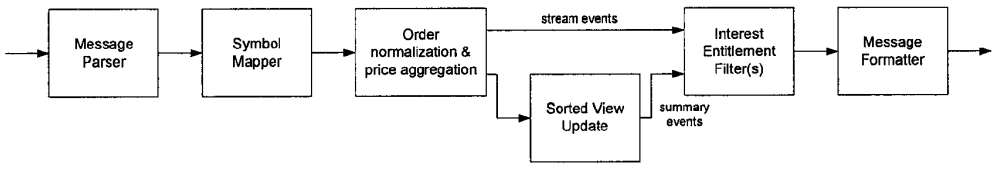


Figure 8

HYBRID APPROACH

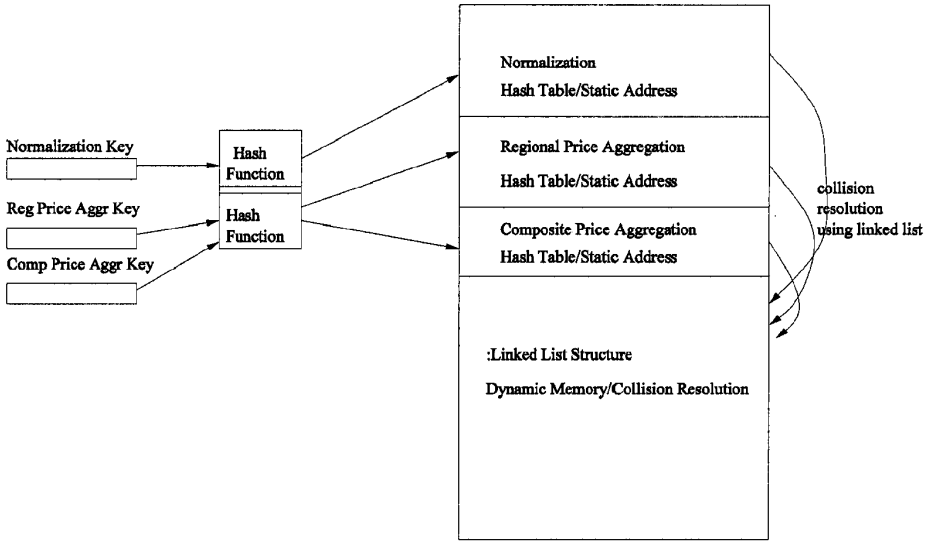


Figure 9

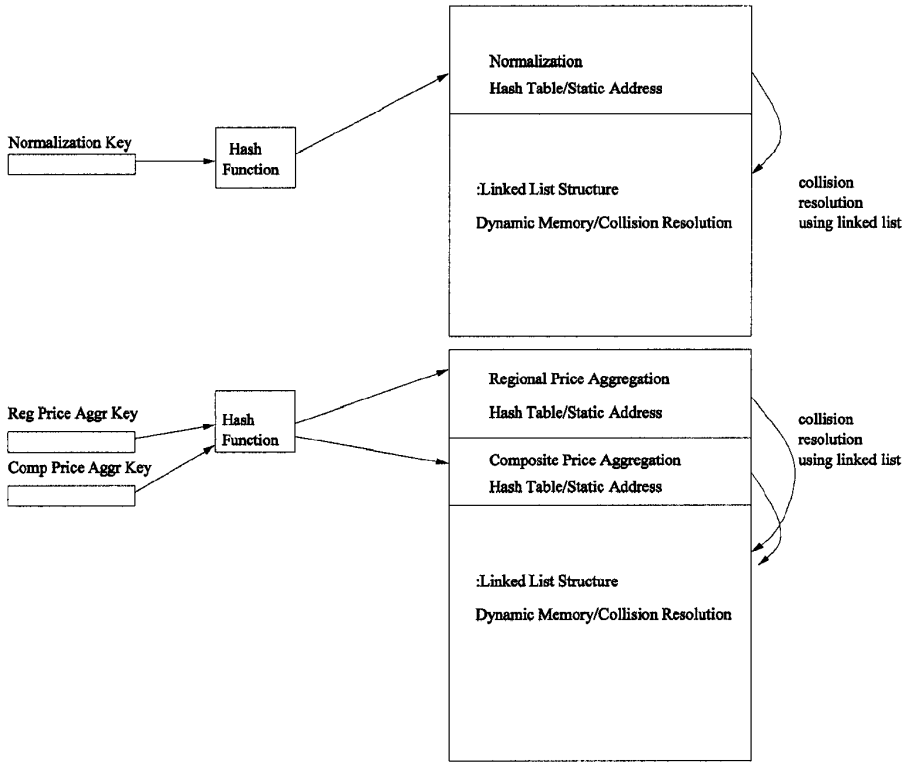


Figure 10

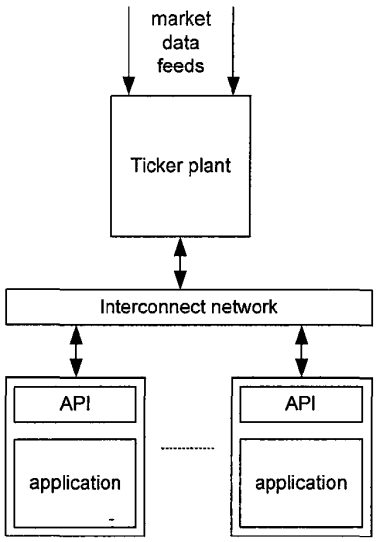


Figure 11

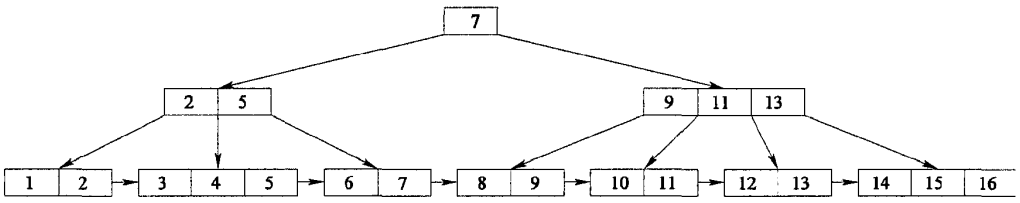


Figure 12

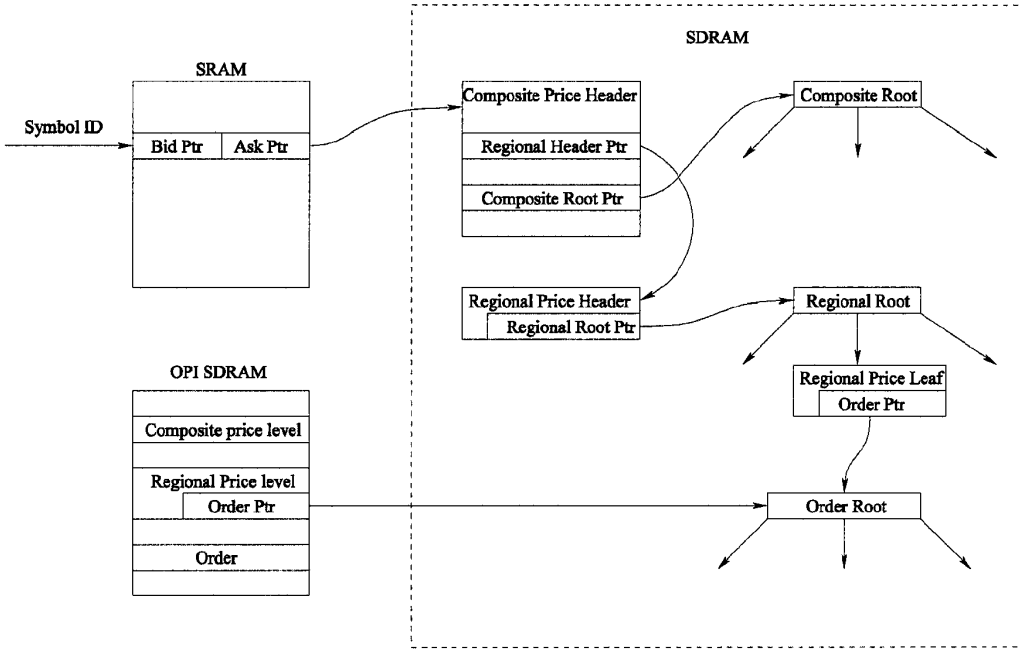


Figure 13

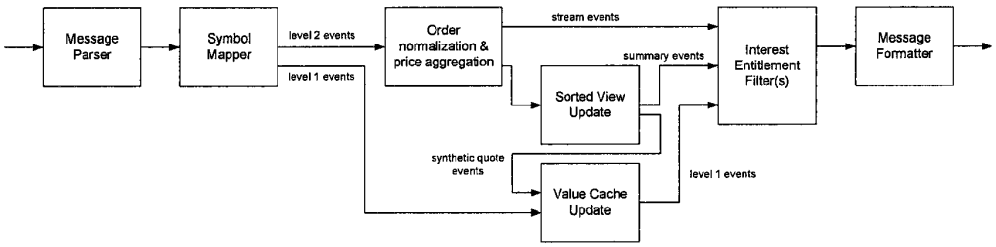


Figure 14