(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
2 August 2007 (02.08.2007)

PCT

(10) International Publication Number
WO 2007/087507 A2

(51) International Patent Classification:
G06F 15/78 (2006.01) *G06F 9/38* (2006.01)
G06F 15/80 (2006.01)

(21) International Application Number:
PCT/US2007/060835

(22) International Filing Date: 22 January 2007 (22.01.2007)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
11/339,892 26 January 2006 (26.01.2006) US

(71) Applicants *(for all designated States except US)*: **EXEGY INCORPORATED** [US/US]; Suite 300, 3668 South Geyer Road, St. Louis, MO 63127 (US). **WASHINGTON UNIVERSITY** [US/US]; One Brookings Drive, St. Louis, MO 63130 (US).

(72) Inventors; and

(75) Inventors/Applicants *(for US only)*: **CHAMBERLAIN, Roger D.** [US/US]; 64 Notre Dame Drive, St. Louis, MO 63141 (US). **SHANDS, E.F. Berkley** [US/US]; 603 Meadowridge Lane, Kirkwood, MO 63122 (US). **BRODIE, Benjamin C.** [US/US]; 726 Leland Avenue, 1W, University City, MO 63130 (US). **HENRICHS, Michael** [US/US]; 5708 Southwest Avenue, St. Louis, MO 63139 (US). **WHITE, Jason R.** [US/US]; 902 Dogwood Creek Drive, Apt. 1, Manchester, MO 63021 (US).

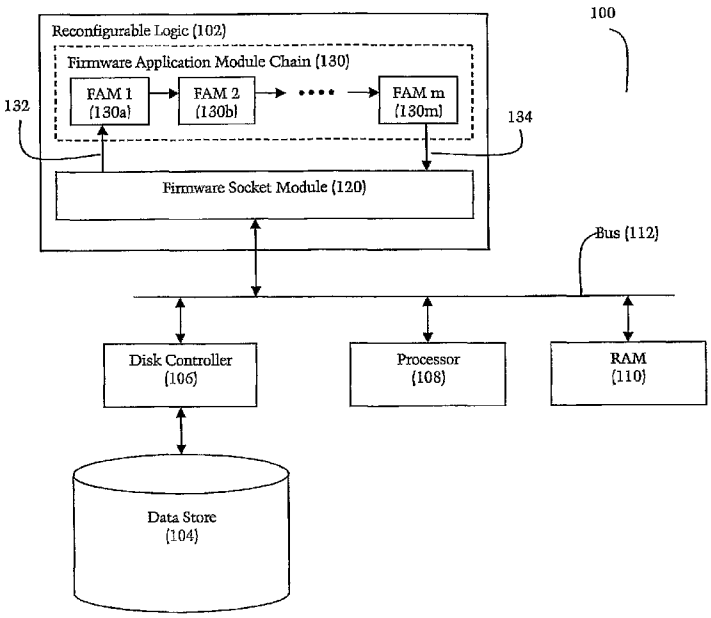
(74) Agents: **VOLK, JR., Benjamin L.** et al.; Thompson Coburn, LLP, One US Bank Plaza, St. Louis, MO 63101 (US).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT, LU, LV, LY, MA, MD, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RS, RU, SC, SD, SE, SG, SK, SL, SM, SV, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM,

[Continued on next page]

(54) Title: FIRMWARE SOCKET MODULE FOR FPGA-BASED PIPELINE PROCESSING



(57) Abstract: A firmware socket module is deployed on a reconfigurable logic device, wherein the firmware socket module has a communication path between itself and an entry point into a data processing pipeline, wherein the firmware socket module is configured to provide both commands and target data to the entry point in the data processing pipeline via the same communication path, wherein each command defines a data processing operation that is to be performed by the data processing pipeline, and wherein the target data corresponds to the data upon which the data processing pipeline performs its commanded data processing operation. Preferably, the firmware socket module is configured to provide the commands and target data in a predetermined order that is maintained throughout the data processing pipeline. Also, the firmware socket module may be configured to (1) access an external input descriptor pool buffer that defines the order in which commands and target data are to be provided to the data processing pipeline, and (2) transfer the commands and target data from an external memory to the data processing pipeline in accordance with the identified defined order. Results of the processing by the data processing pipeline are also returned to external memory by the firmware socket module, whereupon those results can be subsequently used by software executing on a computer system.

WO 2007/087507 A2

ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IS, IT, LT, LU, LV, MC, NL, PL, PT, RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:

— *without international search report and to be republished upon receipt of that report*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

Firmware Socket Module for FPGA-Based Pipeline Processing

Field of the Invention:

The present invention is directed toward the field of interfacing hardware with software to support high speed data processing operations in hardware.

Background and Summary of the Invention:

The ability to improve a computing system's data throughput capabilities when processing large volumes of data is an ever present challenge. In many instances, processing large data volumes using software executed by a general-purpose processor will be too slow to meet the needs of users. Therefore, it is desirable to either re-locate data processing functionality away from software executed by the general-purpose processor of a computer and into firmware deployed on hardware or to partition data processing functionality between such firmware and software. However, when doing so, hardware devices (such as reconfigurable logic devices) need to be interconnected with external resources such as data storage and the software executed by the computer's general-purpose processor in a manner that does not create a bottleneck as data and commands flow back and forth between the hardware and the software.

As used herein, the term "general-purpose processor" will refer 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" will refer 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 general-purpose processor 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 general-purpose processor. The term "firmware" will refer to data processing functionality that is deployed on reconfigurable logic.

In an effort to address the needs in the art, the inventors herein disclose a new design for a firmware socket module that interfaces at least one firmware application module deployed on a reconfigurable logic device with external memory and software. The firmware socket module is also preferably deployed on the reconfigurable logic device. The at least one firmware application module is configured to perform a data processing operation on any target data that it receives, wherein the data processing operation that the at least one firmware application module performs is controlled by a software-based command. Preferably, the at least one firmware application module comprises a plurality of firmware application modules that are arranged in a pipeline. Each firmware application module in the pipeline is preferably individually addressable for command information. Thus, commands can be issued to specific firmware application modules in the pipeline to control the data processing operations of those specific firmware application modules.

The firmware socket module is configured to access external memory and software to receive the commands and target data that are to be processed through the firmware application module pipeline. Preferably, the firmware socket module issues transactions to a system bus to perform direct memory access (DMA) transfers of commands and target data from the computer system's memory to itself. The firmware socket module is configured to in turn provide these commands and target data to the first one of the firmware application modules in the pipeline in a predetermined order that is defined by software.

The firmware socket module and firmware application module pipeline are preferably configured to strictly maintain

the predetermined order of inbound commands and target data when propagating commands and target data through the system. When target data reaches a firmware application module in the pipeline, the firmware application module performs its specified data processing operation on the target data and then provides the so-processed target data to the next firmware application module in the pipeline. When command data reaches a firmware application module in the pipeline, the firmware application module will check to see whether the command is directed toward it and, if it is, will interpret that command to re-arrange its data processing operation as appropriate. If the command is to be propagated further down the pipeline, then the firmware application module will pass the command to the next firmware application module in the pipeline.

The flow of commands and target data, either into the firmware socket module or into the entry point of the firmware application module pipeline, can be thought of as a single stream in which both commands and target data are interleaved in accordance with the defined order. When it is said that the commands and data are interleaved, this does not require (although it does not exclude) a stream of command/data/command/data/command/data... Instead, the interleaved stream of commands and data described herein encompasses a stream such as command/command/command/data/data/data/data/ data/command... wherein the order of commands and data in the stream is defined by software and preserved by the firmware socket module when it propagates the stream to the firmware application module pipeline.

Appropriate commands that control the firmware application module's data processing operation should precede that target data in the stream of commands and target data entering the firmware application pipeline, thereby allowing the data processing operations of the firmware application modules to be appropriately controlled prior to processing

target data. To facilitate the ease by which this strict ordering of commands and target data is maintained, the firmware socket module is configured to provide both command and target data to the first firmware application module in the pipeline over the same communication path that links the firmware socket module with the first firmware application module of the pipeline.

The natural synchronization between commands and target data provided by this firmware socket module-to-firmware application module pipeline connection avoids complexity in the system and also enhances data throughput. In prior art socket interfaces known to the inventors herein, commands are communicated to data processing modules via a different communication path than the communication path used to communicate target data to data processing modules. When such data processing modules are pipelined, such dual communication paths creates management difficulties when attempting to synchronize commands with data. In such cases, when new commands are issued to a data processing module in the pipeline, the entire pipeline will typically need to be flushed of previous commands and previous data before that command and any further target data can be processed through the pipeline, thereby greatly detracting from the pipeline's throughput capabilities. By way of distinction, however, with the present invention one firmware application module of the pipeline can take action on a command while other firmware application modules in the pipeline are simultaneously processing data in accordance with their defined data processing operations. Thus, commands can be issued to firmware application modules to adjust their data processing operations without requiring the entire pipeline to be flushed out.

High level software that is executed by the computer system's general-purpose processor preferably defines the order of commands and data that are eventually propagated through the firmware socket module and the firmware

application module pipeline. Lower level device driver software that is also executed by the computer system's general-purpose processor then preferably preserves this defined order of commands and data and makes such ordered commands and data available to the firmware socket module. The device driver software preferably preserves this order by managing an input descriptor pool buffer in which pointers to commands and target data are stored. The firmware socket module will access the input descriptor pool buffer to learn of the commands and target data that are to be delivered to the firmware application module pipeline.

On the outbound side of the firmware socket module (outbound to software), the device driver software preferably maintains separate buffers for output commands and outbound data to notify the firmware socket module of where commands and data that have been processed by the firmware application module should be stored in memory for subsequent access by computer system software.

Among the advantages that the preferred embodiment of the invention provides are the ability to reliably deliver flow-controlled data from software to a reconfigurable logic device and vice versa, and the ability to develop firmware application modules independent of the computer system in which they are deployed (so long as the firmware application modules conform to the signaling requirements of the firmware socket module). These and other features of the present invention will be in part pointed out and in part apparent to those having ordinary skill in the art upon review of the following description and figures.

Brief Description of the Drawings:

Figure 1 is a block diagram view of a preferred system that embodies the present invention;

Figure 2 illustrates an exemplary framework for the deployment of software and firmware for the preferred system;

Figure 3(a) is a block diagram view of a preferred printed circuit board for installation into a computer system to carry out data processing tasks in accordance with the present invention;

Figure 3(b) is a block diagram view of an alternate printed circuit board for installation into a computer system to carry out data processing tasks in accordance with the present invention;

Figure 4 depicts the preferred signal layer interface between the firmware socket module and the firmware application modules;

Figure 5 depicts the preferred hardware/software interface between the firmware socket module and the memory space managed by the device driver;

Figure 6(a) depicts an example of the input descriptor pool buffer and kernel memory space that is managed by the device driver;

Figures 6(b)-(o) illustrate how commands and target data are sequenced through the firmware socket module and firmware application modules in accordance with the order of commands and target data defined by the input descriptor pool buffer example of Figure 6(a);

Figure 7 illustrates an exemplary FPGA that has been configured with a firmware socket module and a plurality of pipelined firmware application modules, each of which is arranged to perform a different data processing operation; and

Figure 8 illustrates an example of how the firmware application modules of a pipeline can be deployed across multiple FPGAs.

Detailed Description of the Preferred Embodiment:

Figure 1 depicts a preferred system 100 in accordance with the present invention. In this system, a reconfigurable logic device 102 is positioned to receive data that streams off the disk subsystem defined by disk controller 106 and data store 104 (either directly or indirectly by way of system

memory such as RAM 110). Preferably, this data streams into the reconfigurable logic device by way of system bus 112, although other design architectures are possible (see Figure 3(b)). Preferably, the reconfigurable logic device is a field programmable gate array (FPGA), although this need not be the case. System bus 112 also interconnects the reconfigurable logic device 102 with the computer system's main processor 108 as well as the computer system's RAM 110. 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 112 may be a PCI-X bus, although this need not be the case.

The data store can be any data storage device/system, but is preferably some form of a mass storage medium. For example, the data store 104 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).

The computer system defined by main processor 108 and RAM 110 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.

The reconfigurable logic device 102 has firmware modules deployed thereon that define its functionality. The firmware socket module 120 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 130 that is also deployed on the reconfigurable logic device. The FAMs 130i of the FAM chain 130 are configured to perform specified data processing operations on any target data that streams through the chain 130 from the firmware socket module 120. Preferred examples of FAMs that can be deployed on reconfigurable logic are described in United States patent 6,711,558 (entitled "Associative Database Scanning and Information Retrieval"), pending United States patent application 10/153,151 (filed May 21, 2002 entitled "Associative Database Scanning and Information Retrieval using FPGA Devices" and published as 2003/0018630), 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"), United States provisional patent application 60/658,418 (filed March 3, 2005 and entitled "Biosequence Similarity Searching Using FPGA Devices"), United States provisional patent application 60/736,081 (filed November 11, 2005 and entitled "Method and Apparatus for Performing Biosequence Similarity Searching") and United States patent application 11/293,619 (filed December 2, 2005 and entitled "Method and Device for High Performance Regular Expression Pattern Matching"), the entire disclosures of each of which are incorporated herein by reference. For example, a non-exhaustive list of exemplary data processing operations that can be performed by FAMs include data search operations (of various types), data encryption operations (using any of a number of encryption techniques and/or encryption keys), data decryption operations (using any of a number of decryption techniques and/or decryption keys), data compression operations (using any of a number of data compression techniques), data decompression operations (using any of a number of data decompression techniques), and data reduction operations.

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 120. 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.

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 target data that it receives from the firmware socket module. Thus, a FAM can be arranged through an appropriate command to search a specified stream of target data for the presence of the term "Smith" therein. Once the FAM has performed the search operation on the target data stream for the term "Smith", another command can be sent to that FAM that will cause the FAM to re-arrange itself to search for the term "Jones". Not only will the FAM operate at hardware speeds (thereby providing a high throughput of target data through the FAM), but the FAMs can also be flexibly reprogrammed to change the parameters of their data processing operations.

The FAM chain 130 preferably comprises a plurality of firmware application modules (FAMs) 130a, 130b, ... 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.

A communication path 132 connects the firmware socket module 120 with the input of the first one of the pipelined FAMs 130a. The input of the first FAM 130a serves as the entry point into the FAM chain 130. A communication path 134 connects the output of the final one of the pipelined FAMs 130m with the firmware socket module 120. The output of the final FAM 130m serves as the exit point from the FAM chain 130. Both communication path 132 and communication path 134 are preferably multi-bit paths.

Figure 2 depicts an exemplary framework for the deployment of applications on the system 100 of Figure 1. The top three layers of Figure 2 represent functionality that is executed in software on the computer system's general-purpose processor 108. The bottom two layers represent functionality that is executed in firmware on the reconfigurable logic device 102.

The application software layer 200 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 target data those data processing operations are to be performed upon.

The next layer is the module application programming interface (API) layer 202 which comprises a high level module API 202a and a low level module API 202b. The high level module API 202a can provide generic services to application level software (for example, managing callbacks). The low level module API 202b manages the operation of the operating system (OS) level/device driver software 204. A software library interface 210 interfaces the high level module API 202a with the low level module API 202b. Details about this software library interface are provided below.

The interface between the device driver software 204 and the firmware socket module 120 serves as the hardware/software interface 212 for the system 100. The details of this interface 212 will be described in greater detail in connection with Figure 5.

The interface between the firmware socket module 120 and the FAM chain 130 is the firmware module interface 214. The details of this interface will be described in greater detail in connection with Figure 4.

Figure 3(a) depicts a printed circuit board or card 300 that can be connected to the PCI-X bus 112 of a commodity computer system. In the example of Figure 3(a), the printed circuit board includes an FPGA 302 (such as a Xilinx Virtex II

FPGA) that is in communication with a memory device 304 and a PCI-X bus connector 306. A preferred memory device 304 comprises SRAM and DRAM memory. A preferred PCI-X bus connector 306 is a standard card edge connector.

Figure 3(b) depicts an alternate configuration for a printed circuit board/card 300. In the example of Figure 3(b), a private bus 308 (such as a PCI-X bus), a disk controller 310, and a disk connector 312 are also installed on the printed circuit board 300. Any commodity disk technology can be supported, including but not limited to SCSI, SATA, Fibre Channel (FC), etc. In this configuration, the firmware socket 120 also serves as a PCI-X to PCI-X bridge to provide the processor 108 with normal access to the disk(s) connected via the private PCI-X bus 308.

It is worth noting that in either the configuration of Figure 3(a) or 3(b), the firmware socket 120 can make memory 304 accessible to the PCI-X bus, which thereby makes memory 304 available for use by the OS kernel 204 as the buffers for transfers from the disk controller to the FAMs. It is also worth noting that while a single FPGA 302 is shown on the printed circuit boards of Figures 3(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 300 or by installing more than one printed circuit board 300 in the computer system. Figure 8 depicts an example where numerous FAMs in a single pipeline are deployed across multiple FPGAs.

As shown in Figures 1-3, inbound data (from the kernel 204 to the card 300) is moved across the bus 112 in the computer system to the firmware socket module 120 and then delivered by the firmware socket module 120 to the FAM chain 130. Outbound data (from the card 300 to the kernel 204) are delivered from the FAM chain 130 to the firmware socket module 120 and then delivered by the firmware socket module 120 across the PCI-X bus to the software application executing on the computer system. As shown in Figure 2, the three

interacting interfaces that are used are the firmware module interface 214, the hardware/software interface 212, and the software library interface 210.

Firmware Module Interface 214 - Signal Layer

The firmware module interface 214, which is shown in Figure 4, includes the signals exchanged between the firmware socket module 120 and the signals exchanged between FAMs within the FAM chain 130. Table 1 and the description below further describe these signals.

Table 1:

Signal Name	Description	Signal Direction	Signal Size
clk	Clock	Input	Signal
reset	Reset	Input	Signal
module_id	Module ID	Input	6-bit bus
data_in	Input Data	Input	64-bit bus
data_cnt_in	Data Count	Input	3-bit bus
data_vld_in	Data Valid	Input	Signal
ctrl_vld_in	Control Valid	Input	Signal
wait_upstream	Wait Upstream	Output	Signal
data_out	Output Data	Output	64-bit bus
data_cnt_out	Data Count	Output	3-bit bus
data_vld_out	Data Valid	Output	Signal
ctrl_vld_out	Control Valid	Output	Signal
wait_dnstrm	Wait Downstream	Input	Signal

The signal direction entries in the table are listed from the perspective of a firmware module. Thus, signals (4) through (8) ("data_in", "data_cnt_in", "data_vld_in", "ctrl_vld_in", and "wait_upstrm") comprise the interface signals with an upstream firmware module (e.g., communication path 132). Signals (9) through (13) ("data_out", "data_cnt_out", "data_vld_out", "ctrl_vld_out", and "wait_dnstrm") comprise the interface signals with a

downstream firmware module (e.g., communication path 134).

The signal "clk" is a 1-bit input signal that is used to clock the firmware socket module and FAM chain.

The signal "reset" is a 1-bit input signal that is used to reset the firmware socket module and the FAM chain to its start-up state.

The signal "module_ID" is a 6-bit input bus that informs each firmware module of its position in the FAM chain.

The signal "data_in" is a 64-bit bus that is used to transfer target data and command data into the FAM chain. Of note, by using the same bus to transfer both data and commands, synchronization between commands and target data can be maintained with little complexity.

The signal "data_cnt_in" is a 3-bit bus that is used to determine how many of the 8 bytes on the input bus "data_in" represent meaningful data. Preferably, values of 1-7 are literally interpreted, and a value of 0 means that all 8 bytes are valid.

The signal "data_vld_in" is a 1-bit input signal that is used to indicate that the data on the "data_in" bus and the "data_cnt_in" bus are valid on a given clock cycle.

The signal "ctrl_vld_in" is a 1-bit input signal that is used to indicate that the data on the "data_in" bus is valid command (control) information on a given clock cycle. It is worth noting that the "ctrl_vld_in" and "data_vld_in" signals into the same FAM should not be asserted at the same time.

The signal "wait_upstrm" is a 1-bit output signal that is used to tell the upstream firmware module to stop pushing data into the current firmware module. Preferably, the firmware module is configured to absorb 2 data values after the wait is asserted.

The signal "data_out" is a 64-bit output bus that is used to transfer both data and commands out of a given FAM. Once again, because the same bus is used to transfer both data and commands, synchronization between commands and target data can

be maintained through the FAM chain as each FAM operates on the received commands and target data.

The signal "data_cnt_out" is a 3-bit bus that is used to determine how many of the 8 bytes on the output bus "data_out" represent meaningful data. Preferably, values of 1-7 are literally interpreted, and a value of 0 means that all 8 bytes are valid.

The signal "data_vld_out" is a 1-bit output signal that is used to indicate that the data on the "data_out" bus and the "data_cnt_out" bus are valid on a given clock cycle.

The signal "ctrl_vld_out" is a 1-bit output signal that is used to indicate that the data on the "data_out" bus is valid command (control) information on a given clock cycle. It is worth noting that the "ctrl_vld_out" and "data_vld_out" signals out of the same FAM should not be asserted at the same time.

The signal "wait_dnstrm" is a 1-bit input signal that is used to indicate to the FAM that either the firmware socket module or a downstream module cannot absorb data at the moment. Preferably, no more data should be shipped out from that FAM until "wait_dnstrm" is deasserted.

It is worth noting that some contiguous FAMs in the FAM chain may form a FAM subset, the FAM subset having its own entry point and exit point. In such instances, the signal interface described in the table above would be maintained with respect to the entry point and the exit point of the FAM subset, but a different signaling interface can optionally be used internally between the FAMs of the FAM subset. It is also worth noting that a FAM itself may be comprised of a chain of firmware application submodules, wherein communications between the submodules may optionally follow a different signaling interface than that described in the above table.

Firmware Module Interface 214 - Logical Layer

As noted, the information delivered across the firmware module interface on the 64-bit "data_in" and "data_out" buses can either be application data or command (control) information. Both the firmware socket module 120 and the individual FAMs 130i maintain the ordering of commands and data, so this order is preserved throughout throughout the chain of FAMs. Information flowing down the chain along the "data_in" and "data_out" buses (from the firmware socket module to the first FAM in the chain, from one FAM to the next downstream FAM, and from the final FAM in the chain to the firmware socket module) that is data is referred to as the data channel. Information flowing down the chain along the "data_in" and "data_out" buses that is command (control) information is referred to as the command channel. Commands that are delivered via the command channel are typically not consumed, but are interpreted (if appropriate) and passed on to downstream modules. As a result, many commands that enter the command channel are also delivered back to the software.

On the command channel, a preferred format for individual commands is for them to have a 64 bit length with the following fields:
command (16 bits); error (1 bit); sync (1 bit); reserved (8 bits); module ID (6 bits); and parameters (32 bits).

The command field is preferably encoded as two ASCII characters (e.g., "QY" can be a query command), but this need not be the case - any 16 bit value is valid. Upon entry into the command channel, the error bit is clear. Any module can thereafter set the error bit if appropriate, which is then communicated back to the software when the command exits the FAM chain and firmware socket module. The sync bit is preferably set to control synchronization of the outbound commands and data on their way back to the software.

The module ID field identifies the specific firmware module that is to respond to the command. Preferably, a value of 0 for the module ID means that all modules are to respond

to the command (including not only all of the FAMs but also the firmware socket module). Also, it is preferred that a value of 1 for the module ID means that all modules except the firmware socket module are to respond to the command. Preferably, the module ID value of 2 is reserved for the firmware socket module alone, and module ID values greater than or equal to 3 are directed toward specific FAMs. Each FAM will know of its own module ID value by virtue of the input signal module_ID described in connection with the firmware module interface's signal layer.

Preferred global commands for the firmware module interface include a reset command, query command, query response command, pass through command, start of data command, and end of data command.

The reset command (RS) operates to reset the entire FAM chain or individual FAMs, as indicated by the command's module ID field. Preferably, the parameter field is unused for this command.

The query command (QY) operates to query a module to assess its current status. The module should respond with one or more Query Response (QR) commands. After the module has responded with its query responses, that module passes the original QY command to the next module in the chain. Preferably, the parameter field is unused for this command.

One or more query response commands (QR) are generated by a module when it receives a QY command. Upon receipt of a query response, FAMs should simply forward them downstream. Preferably, the parameter field for a QR command is module-specific, wherein this 32-bit field is split into two subfields - the most significant 8 bits being a tag that indicates what type of information is being sent and the least significant 24 bits being the data associated with the given tag.

The pass through command (PS) informs a module if it is to pass data through unaltered or if it is to perform its normal data processing operation. The pass through mode does

not impact command processing or command forwarding - its impact is limited only to the data channel. Preferably, the least significant bit (bit 0) of the parameter field enables/disables the pass through mode - with a pass through bit value of 1 enabling the pass through mode and a pass through bit value of 0 disabling the pass through mode.

The start of data command (SD) is used to mark the start of a data stream (typically on a file boundary). The end of data command (ED) is used to mark the end of a data stream (once again, typically on a file boundary). The parameter fields for the SD and ED command are preferably unused, although in some instances, they can be used to specify a file ID or a frame ID.

Command propagation through the FAM chain preferably proceeds as follows: (1) global commands (those commands whose module ID value is 0 or 1) are propagated forward by each FAM in the FAM chain, (2) a FAM will propagate forward any command whose module ID value is set for a different FAM, and (3) a FAM will propagate forward any command whose sync bit is set.

Preferred commands that are or can be specific to the firmware socket module include a data count command, reset command, and a query response command.

A data count command (DC) can be used to ask the firmware socket module how much data has passed through the FAM chain since the last time it was reset. One DC command sent to the firmware socket module will result in the 3 DC commands being returned. The first responsive DC command will identify the number of bytes that have gone into the FAM chain. The second responsive DC command will identify the number of bytes of data that has exited the FAM chain. The third responsive DC command will be the original sent DC command that is being returned. Preferably, an SD command or an RS command will operate to reset the data counts. With respect to the parameters field, the sent DC command's parameters field is preferably ignored while the parameters field of the

responsive DC commands will preferably identify the relevant number of bytes as described above.

The reset command (RS) format is described above in connection with the global commands. When the firmware socket module receives an RS command, it preferably resets the entire FAM chain and also clears any input and output queues. Any data that may not yet have been returned from the card by the firmware socket module will be cleared by the RS command; thus, applications should make sure that they have received all expected data before sending a reset command to the firmware socket module. This can be achieved by sending a command with the sync bit set or by strobing the Flush bit in the firmware socket module's doorbell register.

The query response command (QR) format is also described above in connection with the global commands. Preferably, when the firmware socket module issues a QR command, its parameters field will be set as follows in Table 2 below:

Table 2:

Tag (8 bits)	Value (24 bits)
0	Module Type ("Firmware Socket Module")
1	Version Number
2	Build (Revision Number)

Commands that are specific to the FAMs will vary as a function of each FAM's data processing operation. For example, a set of commands applicable to a FAM that is configured to perform a search operation where a pattern is matched to a stream of target data would preferably include commands that define the pattern(s) against which the data stream is searched, commands that identify how much context from the data stream should be returned when a match to the pattern is found in the data stream (e.g., returning X number of bytes before and after the match within the data stream), commands that define the number of mismatches allowed for a given pattern (e.g., if the pattern is of length 10, a

mismatch value of K will allow a match to occur when any (10-K) of the 10 pattern characters match within the data stream), commands that define which bits in a pattern must find a match in the data stream for a match to be declared, commands that define a file within the data that is to be searched, and commands that define a subset of data within a file that should be searched. The parameters field or optionally other fields of such commands can be used to define these aspects of the pattern matching operation performed by the FAM. Also the QR command from a FAM that is configured to perform a pattern matching operation preferably responds with tags whose values correspond to an identification of the FAM type, an identification of the FAM type's version, an identification of the FAM type's build, and an identification of the number of patterns supported by the FAM.

As another example, a set of commands applicable to a FAM that is configured to perform an encryption or decryption operation would preferably include commands that identify the key that should be used in the encryption/decryption operation.

In addition to any other previously listed data processing operations that can be performed by FAMs, possible data processing operations also include a field selection operation that takes in fixed length records and passes only the bytes of the record that have been selected (wherein a command to such a FAM would indicate which bytes are to be selected), a record gate operation that adds padding to records whose size is not a multiple of the data size processed through the FAMs (8 bytes in the preferred example described herein) (wherein a command to such a FAM would indicate the desired record size), an inverse record gate operation that removes padding bytes from records (wherein a command to such a FAM would indicate which bytes of a record are to be removed), and a record select operation that searches within specified column boundaries of fixed length records for matches to one or more patterns (an operation

whose commands would be similar to the pattern matching commands described above but also including a command that identifies how records are to be masked to limit the searches to occur within specified column boundaries).

Preferably, the FAMs that are deployed on the FPGA are dedicated to performing one type of data processing operation, wherein the command signals operate to define specific aspects of that type of data processing operation. For example, as shown in Figure 7, each FAM in the FAM chain is devoted to a type of data processing operation - e.g., the first FAM is configured to performing a decompression operation, the 4th FAM is configured to perform one type of search operation, the 5th FAM is configured to perform another type of search operation, and the second to last FAM is configured to perform an encryption operation. Through appropriate commands, the specifics of each FAM's data processing operation can be controlled (e.g., to arrange the 4th FAM to search for the term "Smith", or to arrange the second to last FAM to perform an encryption operation using Key A). If a FAM is to be entirely reprogrammed to perform a different type of data processing operation, it is preferred that the FPGA be reconfigured to achieve this end, although this need not be the case.

Furthermore, by appropriately turning on or off each FAM in the FAM chain, specialized processing operations can be achieved by the FAM chain. For example, to perform a search within a data stream for a given pattern wherein that data is stored in an encrypted format, commands can be sent to the FAM chain that (1) turns on one of the search FAMs and arranges it with the given pattern, (2) turns on the decryption FAM and arranges it with the appropriate key to decrypt the data stream, and (3) turns off all other FAMs in the chain. In such an instance, an encrypted data stream will be decrypted by the second FAM and the decrypted data stream will be searched by one of the search FAMs for the given pattern.

Hardware/Software Interface 212

Preferably, the firmware socket module uses a Xilinx PCI-X core with a backend to transfer data to/from the system's memory space when used in conjunction with driver level system software 204. The nature of this backend interface is described in greater detail herein.

The Xilinx PCI-X Core Version 5.0 handles translation from the interface as described by the PCI-X specification to an interface designed by Xilinx as described by the Xilinx LogiCore PCI-X Design Guide Version 5.0 and the Xilinx LogiCore PCI-X Implementation Guide Version 5.0. Preferably, the configuration parameters are set to indicate that the PCI-X-connected printed circuit board 300 is 64-bit and 133 MHz capable. Whether the system BIOS sets the speed of the device to 100 MHz or 133 MHz depends upon the 133 MHz capability bit and the physical voltage and time delay on the PCIXCAP pin. If the modules deployed on the FPGA 302 only run at 100 MHz, then the PCIXCAP pin should be set to indicate this. Also, the PCI-X configuration can be set to use up to 3 64-bit Base Address Registers (BARs) or 6 32-bit BARs. Preferably, the firmware socket module 120 uses the first BAR, configured as a 64-bit BAR, to map its device registers for communication with the device driver software 204.

The bulk of communications between the firmware socket module 120 and the device driver software 204 preferably occurs via three sets of descriptor pools. As shown in Figure 5, the buffers for these descriptor pools comprise a kernel to card input descriptor pool buffer 500, a card to kernel output descriptor pool buffer 504 that is for data, and a card to kernel output descriptor pool buffer 506 that is for commands. These descriptor pool buffers preferably reside in memory space that is managed by the OS kernel/device driver software 204 (for example, within RAM 110). Each entry in buffers 500, 504 and 506 comprises a descriptor, wherein each descriptor may span multiple addresses within the buffer.

In addition to the byte count fields and other flags described below, the descriptors each preferably contain the physical and virtual address of memory locations for data movement either from the software to the hardware (for an input descriptor) or from the hardware to the software (for an output descriptor). Thus, the entries in the descriptor pool buffers serve as pointers to where the firmware socket module is to read data from or write data to.

The memory space 502 from which and to which the firmware socket module is to read/write data (both commands and target data) can be any memory in the physical address space of the computer system and any physically addressable data storage device connected to the computer system. As such, memory space 502 need not be limited to the motherboard of the computer system (e.g., RAM 110). For example, if memory 304 of the card is made available to the driver level software 204, then the memory space 502 for storing commands and target data could include memory 304. Also, the memory space 502 can include addresses within data store 104 or some other system-connected storage device if the address space of data store 104 or such a storage device is made available to driver level software (e.g., through disk controller 106 for data store 104).

Preferred normal operation for the hardware/software interface 212 is as follows:

1) The software device driver 204 puts commands and data that are to be delivered to the FAMs into a set of buffers in the memory space 502. Preferably, the driver level software 204 stores target data that is to be streamed through the re-configurable logic device as a single set in contiguous addresses of memory space 502, which enhances not only the rate at which target data can be loaded into the FPGA but also provides flexibility in that consecutive descriptors pointing to such contiguously stored target data can potentially be

consolidated into fewer descriptors by updating the descriptor's byte count fields appropriately.

2) The device driver 204 then builds the input descriptors that tell the firmware socket module about the content of the buffers in memory space 502. Preferably, one descriptor is used for each command or segment of M bytes of target data, wherein the size of M is variable according to a variety of factors. The size for M can be defaulted to 32 kilobytes. However, as noted this value can be variable by software giving consideration to a desire to minimize the processor overheads associated with buffer setup, descriptor management, and the like while still moving large amounts of data through the card for each descriptor. The device driver 204 also preferably builds output descriptors that tell the firmware socket module where to place data and commands that come back from the FAMs. The input descriptors are stored by the device driver 204 in the input descriptor pool buffer 500. The output descriptors that correspond to data are stored by the device driver 204 in the output descriptor pool buffer 504, and the output descriptors that correspond to commands are stored by the device driver 204 in the output descriptor pool buffer 506.

3) The device driver 204 then informs the firmware socket module that new input descriptors and output descriptors are available via a doorbell signal over communication path 510.

4) The firmware socket then reads a first input descriptor in buffer 500 to identify the location where the data (command or target data) to be delivered to the FAM chain is stored, performs a DMA transfer of the data (command or target data) from that location to the FAM chain, and moves on to the next input descriptor in the buffer 500. Thus, commands and target data are provided to the FAM chain in the same order as set within the input descriptor pool buffer 500.

5) The firmware socket module then delivers commands exiting the FAM chain to a buffer in memory space 502c as specified by the command output descriptor in buffer 506. The firmware socket module also delivers data exiting from the FAM chain to a buffer in memory space 502b as specified by the data output descriptor in buffer 504.
6) The firmware socket module then notifies the device driver (via an interrupt) that commands and/or data are available.
7) Software then invokes the appropriate data or command handler as appropriate to address the returned data and/or commands (which is controlled via the software library interface 210).

When sending information from the software to the reconfigurable logic device, all transfers (both command and data) are preferably managed by the common input descriptors stored in buffer 500. As long as individual FAMs in the FAM chain do not reorder data and commands, the order of data and commands defined by the order of input descriptors in buffer 500 will be maintained throughout processing within the reconfigurable logic device.

Strict ordering is not necessarily required for data and commands that are sent from the reconfigurable logic device back to the software. If strict synchronization is desired throughout the process (including the return of commands and data to the software), then the device driver software can set the sync bit in an individual command. When a command with the sync bit set arrives at the firmware socket module from the exit point from the FAM chain, then the firmware socket module preferably performs the following functions: (1) flush the data channel buffers to memory, (2) create a null (empty) buffer in the data channel that has a sync flag set in the data output descriptor, (3) ensure that this data output descriptor is flushed to memory, and (4) flush the command channel buffer (with the sync bit set in the command itself).

The firmware socket module 120 preferably has 16 device registers that are mapped into the memory region of the first BAR identified above (and which is configured as a 64-bit BAR). The physical addresses of each device register are the Base Address plus an offset. The registers and their offset from the BAR are listed in table 3 below:

Table 3:

Offset from BAR	Device Register
0x00	Firmware ID
0x08	FPGA Info
0x10	Device Status
0x18	Onboard RAM Size
0x20	Doorbell
0x28	Interrupt Status
0x30	Data-to-Card Address
0x38	Data-to-Card Count
0x40	Data-to-Card Next
0x48	Data-to-Kernel Address
0x50	Data-to-Kernel Count
0x58	Data-to-Kernel Next
0x60	Data-to-Kernel Address
0x68	Data-to-Kernel Count
0x70	Data-to-Kernel Next
0x78	Parameters

The firmware ID is a 64-bit read-only register composed of 2 values. The least significant 32-bits are the firmware ID number and the most significant 32-bits are the revision number of the particular firmware indicated by the ID.

The FPGA information register is a 64-bit read-only register that contains 2 pieces of information- the FPGA type and the FPGA size. The FPGA type is 4 ASCII characters (32-bits) that indicate the type of FPGA. For example, if the FPGA device is a Xilinx Virtex-II device, then the FPGA type would be "xc2v". The FPGA size is a 32-bit integer that indicates what size FPGA is being used. For example, if the

FPGA device is a Xilinx Virtex-II 4000, then the FPGA size would be 4000. The FPGA type is the least significant 32 bits of the FPGA information register, while the FPGA size is the most significant 32-bits of the register.

The device status register is a 64-bit read-only register containing 2 pieces of information about the hardware system attached to the hardware device. The least significant 32-bits indicate the type of hardware system that is attached to the hardware device. Table 4 below indicates an example of device type numbering.

Table 4:

Device Type Number	Type of Attached System
0	Invalid Device Type
1	Standalone System (no attached devices)
2	SCSI
3	IDE
4	FibreChannel
5	iSCSI
6	Reserved

The most significant 32-bits of the device status register are an integer indicating the number of devices connected to the hardware system.

The onboard RAM size is a 64-bit register that indicates the amount of memory 304 (in kilobytes) that is attached to the FPGA device.

The doorbell register is a 64-bit read/write register that contains flags used for communication between the driver level software 204 and the firmware socket module 120. The doorbell flags and their functionalities are listed in table 5 below:

Table 5:

Doorbell Bit(s)	Name of Functionality
0	Run
1	Stop
2	Firmware Socket Module Reset
3	Flush
4	Strobe Debug
5	Clock Counter Reset
6-63	Reserved

The Run bit is set by the driver level software 204 to tell the firmware socket module to start or resume running. This bit will be cleared by the firmware socket module when it is acknowledged by the hardware. The Stop bit is set by the driver level software 204 to tell the firmware socket module to stop running or pause. This bit will be cleared by the firmware socket module when it is acknowledged by the hardware. The Reset bit is set by the driver level software 204 to reset the firmware socket module and all of the FAMs in the FAM chain. This bit will be cleared by the firmware socket module when it is acknowledged by the hardware. The Flush bit is set by the driver level software 204 to flush the outbound buffer(s) on the firmware socket module that holds commands and data destined for the software. This bit will be cleared by the firmware socket module when it is acknowledged by the hardware. The Strobe Debug bit is a bit that can be used for debugging purposes. The Clock Counter Reset bit resets the counters in the firmware socket module that are used for determining the clock speed of the PCI-X bus. The clock speed of the PCI-X bus can be determined by reading the upper 9 bits of the firmware socket module parameters register.

The interrupt status register is a 64-bit read/write register that is used to notify to the driver level software 204 of the interrupt conditions listed in table 6 below. Any

time an interrupt condition is set, an interrupt is triggered on the PCI-X bus to interrupt the operating system and should be responded to by the driver level software 204. If multiple interrupt conditions occur before the driver level software 204 reads the interrupt status register, then multiple bits will be set when the software reads the interrupt status register. When the interrupt status register is read, it is cleared.

Table 6:

Interrupt Bit	Name
0	Data-to-Card Input Descriptor Buffer Empty
1	Data-to-Card Error
2	Data-to-Card Input Descriptor Buffer Done
3	Data-to-Kernel Output Descriptor Buffer Full
4	Data-to-Kernel Error
5	Data-to-Kernel Output Descriptor Buffer Ready
6	No Results Found
7	Command-to-Kernel Output Descriptor Buffer Full
8	Command-to-Kernel Error
9	Command-to-Kernel Output Descriptor Buffer Ready
10-63	Reserved

The firmware socket module sets Interrupt Bit 0 when it tries to read an input descriptor buffer of data and the next input descriptor buffer is not ready to be read (by checking the Ready bit of the next input descriptor queued in the buffer 500). If Interrupt Bit 0 has been written, then the driver level software 204 must write to the Run bit of the doorbell register before the firmware socket module will try to read the input descriptor pool buffer again.

Interrupt Bit 1 signals that an error has occurred when reading an input descriptor of data destined for input to the firmware socket module.

If the interrupt flag of the input descriptor pool buffer 500 is set, then the firmware socket module will set Interrupt

Bit 2 after it is done reading an input descriptor buffer of input data.

The firmware socket module sets Interrupt Bit 3 if it has data to output to the kernel, but the next data-to-kernel output descriptor buffer is unavailable (by checking the Ready bit of the next data output descriptor queued in the buffer 504). Once the firmware socket module has set this interrupt, the driver level software 204 must write to the Run bit of the doorbell register after it has made the data-to-kernel output descriptor buffer(s) available before the firmware socket module will try again to output data.

Interrupt Bit 4 signals that an error has occurred when the firmware socket module tried to write data to an output descriptor buffer.

The firmware socket module set Interrupt Bit 5 whenever it writes data to a data-to-kernel output descriptor buffer and has closed the descriptor (i.e., the firmware socket module is moving on to the next data-to-kernel descriptor buffer for writing data).

Interrupt Bit 6 is defined to indicate that there is no output data to be returned when a Flush bit has been set in the doorbell register.

The firmware socket module sets Interrupt Bit 7 if it has command data to output to the kernel, but the next command-to-kernel output descriptor buffer is unavailable (by checking the Ready bit of the next command output descriptor queued in the buffer 506). Once the firmware socket module has set this interrupt, the driver level software 204 must write to the Run bit of the doorbell register after it has made the command-to-kernel output descriptor buffer(s) available before the firmware socket module will try again to output command data.

Interrupt Bit 8 signals that an error has occurred when the firmware socket module tried to write command data to a command output descriptor buffer.

The firmware socket module sets Interrupt Bit 9 whenever it writes data to a command-to-kernel output descriptor buffer

and has closed the descriptor (i.e., the firmware socket module is moving on to the next command-to-kernel descriptor buffer for writing command data).

The data-to-card address register is a 64-bit read/write register that is used to indicate the physical address of the beginning of the data-to-card input descriptor pool buffer 500. The driver level software 204 should write to this register before the Run bit of the doorbell register is written to for the first time.

The data-to-card count register is a 32-bit read/write register that is used to indicate the number of data-to-card input descriptor buffers available for reading data. The driver level software 204 should write to this register before the Run bit of the Doorbell register is written to for the first time. When this address is read from, a 64-bit value is returned wherein the upper 32 bits are padded with zeros.

The data-to-card next register is a 32-bit read/write pointer to the next data-to-card input descriptor buffer that is to be read from. This is used to ensure that the firmware socket module and the driver level software are in sync. When this address is read from, a 64-bit value is returned wherein the upper 32 bits are padded with zeros.

The data-to-kernel address register is a 48-bit read/write register that is used to indicate the physical address of the beginning of the data-to-kernel output descriptor pool buffer 504. The driver level software 204 should write to this register before the Run bit of the doorbell register is written to for the first time. When this address is read from, a 64-bit value is returned wherein the upper 16 bits are padded with zeros.

The data-to-kernel count register is a 32-bit read/write register that is used to indicate the number of data-to-kernel output descriptor buffers available for writing data. The driver level software 204 should write to this register before the Run bit of the Doorbell register is written to for the

first time. When this address is read from, a 64-bit value is returned wherein the upper 32 bits are padded with zeros.

The data-to-kernel next register is a 32-bit read/write pointer to the next data-to-kernel output descriptor buffer that is to be written to. This is used to ensure that the firmware socket module and the driver level software are in sync. When this address is read from, a 64-bit value is returned wherein the upper 32 bits are padded with zeros.

The command-to-kernel address register is a 48-bit read/write register that is used to indicate the physical address of the beginning of the command-to-kernel output descriptor pool buffer 506. The driver level software 204 should write to this register before the Run bit of the doorbell register is written to for the first time. When this address is read from, a 64-bit value is returned wherein the upper 16 bits are padded with zeros.

The command-to-kernel count register is a 32-bit read/write register that is used to indicate the number of command-to-kernel output descriptor buffers available for writing command data. The driver level software 204 should write to this register before the Run bit of the Doorbell register is written to for the first time. When this address is read from, a 64-bit value is returned wherein the upper 32 bits are padded with zeros.

The command-to-kernel next register is a 32-bit read/write pointer to the next command-to-kernel output descriptor buffer that is to be written to. This is used to ensure that the firmware socket module and the driver level software are in sync. When this address is read from, a 64-bit value is returned wherein the upper 32 bits are padded with zeros.

The parameters register is a 64-bit register that is used to set programmable parameters in the firmware socket module. Some of these parameters are read/write while others are read-only as depicted in table 7 below.

Table 7:

Bits	Parameter	Read/Write
5-0	Max PCI-X Burst Size	Read/Write
54-6	Reserved	Read/Write
55	Bus Speed Valid	Read-Only
63-56	PCI-X Bus Speed	Read-Only

The Max PCI-X Burst Size parameter is used to set the maximum size transaction that the firmware socket module will try to execute on the PCI-X bus 112. This maximum transaction size will be the 6-bit value in the parameters register times 128, although setting this field to zero will set the maximum transaction size to 4096 (the maximum transaction size allowed by the PCI-X specification).

Bit 55 of the parameters register is used to indicate if the PCI-X bus speed in the upper 8 bits of this register is valid (wherein a "1" indicates validity and a "0" indicates invalidity).

The most significant 8 bits of the parameters register indicates the calculate speed of the PCI-X bus in MHz. The Bus Speed Valid bit (bit 55) should also be read to determine if this value is valid or not. If the PCI-X Bus Speed value is not valid, the driver level software should wait a short time and reread the value.

As discussed, communications between the firmware socket module and the driver level software 204 occurs using 3 sets of descriptors - 1 input descriptor pool and 2 output descriptor pools (one for commands and the other for data). Each pool has its own set of descriptors which contain flags and fields with information about the descriptor buffers (kernel buffers).

Each descriptor is preferably 32 bytes long, broken down into 4 pieces of 8 bytes each, as shown in table 8 below:

Table 8:

Number of Bytes	Field
8	Flags
8	Byte Count
8	Physical Address
8	Virtual Address

The flags field is a 64-bit field containing information about the descriptor buffer as described in table 9 below:

Table 9:

Bit(s)	Flag Name
0	Busy
1	Command/Data
2	Data-to-Kernel
3	Data-to-Card
4	Ready
5	Done
6	Interrupt
7-63	Reserved

The Busy bit is set by the driver level software 204 to indicate that the descriptor buffer is in use.

The Command/Data bit is used to tell the firmware socket module whether the buffer contains command information or data. If the buffer contains command information, this bit should be set; otherwise this bit should be cleared. The firmware socket module reads this bit on the data-to-card input descriptor buffers and marks all the data in that buffer appropriately (as either command or data).

The Data-to-Kernel bit indicates whether or not the data in the buffer is destined for the driver level software 204 (the kernel). If this bit is set, the data is output from the firmware socket module, destined for the kernel. The firmware socket module is responsible for setting this bit. Either this bit or the Data-to-Card bit should always be set, but never both.

The Data-to-Card bit indicates whether the data in the buffer is destined for the firmware socket module (deployed on the card). If this bit is set, the data is input to the firmware socket module from the driver level software 204. The driver level software 204 is responsible for setting this bit. Either this bit or the Data-to-Kernel bit should always be set, but never both.

The Ready bit is set by the driver level software 204 to indicate that a buffer is ready for use by the firmware socket module. The firmware socket module should clear this bit when it processes the buffer.

The Done bit is set by the firmware socket module to indicate to the driver level software 204 that the firmware socket module has finished processing the buffer. Driver level software 204 should clear this bit when it has finished cleaning up the buffer.

The Interrupt bit is set by the driver level software to instruct the firmware socket module to interrupt with a Data-to-Card Descriptor Buffer Done Interrupt after it has processed this buffer.

The byte count field is a 64-bit field that contains a byte count for the buffer. The meaning of the byte count differs for input and output descriptor buffers. For data-to-card input descriptor buffers, the byte count field is the number of bytes of data contained in the buffer. The firmware socket module only uses 20 bits for the incoming byte count, thus the maximum data-to-card buffer that the firmware socket module can handle is (1MB-1) bytes. This byte count value should be set by the driver level software 204. For the data-

to-kernel and command-to-kernel output descriptor buffers, the byte count is initially set by the driver level software 204 to indicate how many bytes are available in the buffer for writing. When the firmware socket module sets the done bit on the buffer, it should also update the byte count with the actual number of bytes written to the buffer. This will also be a maximum of (1MB-1) bytes.

The physical address field is a 64-bit field containing the physical address of the kernel buffer in main memory. This field is set by the driver level software 204. Preferably, the firmware socket module only handles physical addresses that can be represented in 48 bits.

The virtual address field is a 64-bit field that contains the virtual address of the kernel buffer and is used only by the driver level software 204.

In operation, when the firmware socket module receive a Run signal on its doorbell register, the firmware socket module will read and continue to read data-to-card input descriptors in the buffers 500 until it tries to read from a buffer that is not ready or until there is command information or data in an output FIFO maintained by the firmware socket module.

The firmware socket module preferably gives command information for output the highest priority. If there is any command information for output, the firmware socket module will output it to an available Command-to-Kernel output descriptor buffer 506. If such a buffer 506 is unavailable, the firmware socket module will issue a Command-to-Kernel Output Descriptor Buffer Full interrupt. The firmware socket will then not try to output command information to a buffer 506 again until the Run bit in the doorbell register is written to by the driver level software 204 to indicate that buffers 506 have been made available.

The firmware socket module preferably gives data for output the second highest priority. If the amount of data in the firmware socket module's output FIFO is greater than or

equal to the maximum PCI-X burst size, then the firmware socket module will output data to an available data-to-kernel output descriptor buffer 504. If the next data-to-kernel buffer 504 is unavailable, the firmware socket module will signal a Data-to-Kernel Output Descriptor Buffer Full interrupt. The firmware socket module will then not try to output data to a buffer 504 again until the Run bit in the doorbell register is written to by the driver level software 204 to indicate that buffers 504 have been made available.

The firmware socket module preferably gives incoming data or commands the lowest priority. If the next data-to-card input descriptor buffer 500 is unavailable, then the firmware socket module will signal a Data-to-Card Input Descriptor Buffer Empty interrupt. The firmware socket module will not try to read the next buffer 500 again until the Run bit of the doorbell register is written to by the driver level software 204, indicating that the buffers 500 have been filled.

The firmware socket module will also preferably continue processing buffers 500, 504, and 506 in order until the Next pointer reaches the Count, at which point the Next pointer will reset to the first buffer.

Figure 6(a) illustrates an example of how inbound data and commands can be processed by the system 100. The input descriptor pool buffer 500 of Figure 6(a) stores a plurality of descriptors, the descriptors including pointers to addresses in memory space 502 where commands and target data are stored. In this example, pointer 1 points to a command that arranges FAM#3 in a FAM chain 130 to perform a search for the term "Smith" in a data stream. Pointers to other commands are identified in Figure 6(a). Buffer 500 also stores descriptors that point to target data that is to be processed through the system 100 (see, for example, pointer 4). As indicated, the driver level software 204 will populate the buffer 500 with these descriptors, thereby defining the order of commands and data that are to be delivered to the firmware socket module and propagated through the FAM chain.

Figures 6(b)-(o) depict such propagation using the order defined by the example of Figure 6(a). For ease of illustration, SD and ED commands are omitted at the boundaries of the target data in the buffer 500. As can be seen, the defined order between commands and target data is maintained by the firmware socket module and FAM chain as these commands and data are processed through the FPGA. By appropriately ordering commands and target data in the input descriptor buffer, the software can achieve powerful high speed data processing operations. For example, the descriptors in buffer 500 corresponding to pointers 1-5 are effective to perform a decryption operation on encrypted target data stored in memory from Address A+1 through C followed by a search through the decrypted data for the presence of the term "Smith". Also, presuming that the data stored in memory from Address P+1 through W has been compressed after being encrypted, the descriptors in buffer 500 corresponding to pointers 6 et seq. are effective to perform a decompression operation on that target data, followed by a decryption operation on the decompressed target data, followed by a search through the decrypted and decompressed target data for the presence of the term "Jones". It is also worth noting that in practice, the length of target data processed following a set of commands is likely to be considerably greater than the example shown in Figures 6(a)-(o), for example the length of target data can be on the order of megabytes, gigabytes, and even terabytes (or greater) depending upon the scope of the desired data processing operation.

Software Library Interface 210:

The software API that supports application use of FAMs is preferably embodied in a class, such as a class that could be labeled ExegyFpga. This class preferably includes the following public methods:

```
bool ExegyFpga::SendCommand ( const char * TwoChars,
                              const it ModuleID,
                              const unsigned int Parameter,
                              const bool WaitForIt,
                              const bool Sync )
```

The SendCommand method delivers the given command to the firmware socket module and FAMs via the command channel. The argument "TwoChars" is a pair of bytes (typically ASCII characters, but this need not be the case) that goes in the command field (16 bits). The error field is cleared (set to 0). The argument "Sync" goes in the sync field (1 bit). The argument "ModuleId" goes in the Module ID field (6 bits). The argument "Parameter" goes in the parameters field (32 bits). The argument "WaitForIt" is a flag that indicates whether the method invocation should block (if WaitForIt is 1) or not block (if WaitForIt is 0) until the command returns from the firmware. The method returns false on error.

```
size_t     ExegyFpga::Write ( size_t  bytes, char * buffer )
```

The Write method delivers data from the given buffer to the data channel. The method returns the number of bytes transferred or an error in "errno".

```
size_t     ExegyFpga::ReadCmd ( size_t  bytes, char * buffer )
```

The ReadCmd method reads commands from the Command-to-Kernel output descriptor buffer(s) and delivers them to a user buffer. This method returns bytes placed in the buffer.

```
size_t     ExegyFpga::Read( size_t  bytes, char * buffer )
```

The Read method reads data from the Data-to-Kernel output descriptor buffer(s) and delivers them to a user buffer. This method returns bytes placed in the buffer or an "errno".

ENODATA is returned on a sync mark being encountered (i.e., the null buffer which is the result of a command sync bit).

An exemplary use of the software library interface is illustrated below.

```
Card = ExegyFpga::Open(...);

Card->SendCommand(...);        //send a command to FAM chain
Card->Write(...);              //send test data
Card->ReadCmd(..);             //ensure command made it
Card->Read(...);               //read test data
Card->Close();
```

While the present invention has been described above in relation to its preferred embodiment, 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. 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 of interfacing at least one firmware application module that is deployed on a reconfigurable logic device with a computer system, the computer system comprising a processor and a memory space that are in communication with each other, wherein the at least one firmware application module is configured to perform a data processing operation on any target data that it receives, the method comprising:

within a firmware module that is deployed on the reconfigurable logic device and that interfaces the at least one firmware application module to the computer system memory space, (1) reading from a buffer within the memory space to identify a plurality of commands and target data that are to be provided to the at least one firmware application module, wherein the buffer defines an order in which the commands and target data are to be provided to the at least one firmware application module, each command controlling the data processing operation that the at least one firmware application module is to perform on any target data that it receives, and (2) transferring commands and target data from the computer system memory to the at least one firmware application module in accordance with the defined order.

2. The method of claim 1 further comprising:

processing the transferred commands and target data through the at least one firmware application module as a stream of interleaved commands and target data.

3. The method of claim 2 wherein a communication path links the firmware module with the at least one firmware application module, and wherein the transferring step comprises providing the commands and target data to the at least one firmware application module over the same communication path.

4. The method of claim 3 wherein the at least one firmware application module comprises a plurality of firmware

application modules that are arranged in a pipeline, the pipeline having a first firmware application module and a final firmware application module, wherein the transferring step further comprises transferring, over the same communication path, the commands and target data from the computer system memory to the first firmware application module of the pipeline in accordance with the determined order.

5. The method of claim 4 wherein the processing step further comprises sequentially processing the transferred commands and target data through each additional firmware application module of the pipeline culminating in the final firmware application module of the pipeline.

6. The method of claim 5 wherein a plurality of the commands are specific to individual ones of the firmware application modules.

7. The method of claim 6 wherein at least one of the commands comprises a pass through command to one of the firmware application modules.

8. The method of claim 6 wherein the data processing operations defined by the commands comprise at least two members of the group consisting of a search operation, a data reduction operation, an encryption operation, a decryption operation, a compression operation, a decompression operation, and a pass through operation.

9. The method of claim 5 wherein the computer system memory space includes random access memory (RAM), and wherein the external buffer is stored in the RAM.

10. The method of claim 9 further comprising:

managing the external buffer with a device driver that operates under control of the processor.

11. The method of claim 10 wherein the buffer comprises a first buffer, wherein the final firmware application module of the pipeline is configured to produce an output as a result of the data processing operation that it performs on any target data that it receives, the method further comprising:

via an additional communication path that links the final firmware application module with the firmware module, transferring the final firmware application module's output to the firmware module; and

within the firmware module, (1) reading an additional buffer managed by the device driver to determine a location in the computer system memory space where the final firmware application module's output is to be stored, and (2) transferring the final firmware application module's output to the determined location in the computer system memory.

12. The method of claim 10 further comprising:

storing the target data in the computer system memory space in contiguous memory addresses in the same order as the target data are to be provided to the firmware application module pipeline.

13. The method of claim 12 wherein the managing step comprises populating the first buffer with pointers to the memory space addresses where the commands and target data are stored in the computer system memory space.

14. The method of claim 13 wherein the populating step comprises storing the pointers in the first buffer in the same order as the commands and target data corresponding thereto are to be provided to the firmware application module pipeline.

15. The method of claim 4 wherein the reconfigurable logic device comprises a field programmable gate array (FPGA).

16. A method for controlling the flow of data and commands to at least one firmware application module that is deployed on a re-configurable logic device, the commands comprising instructions for controlling the processing of the data by the at least one firmware application module, the method comprising:

providing a common communication path to said at least one firmware application module; and

delivering both the commands and data over the common communication path to said at least one firmware application module.

17. The method of claim 16 wherein the at least one firmware application module comprises a plurality of firmware application modules arranged in a pipeline, and wherein the delivering step comprises delivering a stream of interleaved commands and data to a first one of the firmware application modules in the pipeline.

18. The method of claim 17 wherein at least a plurality of the commands are specific to individual ones of the firmware application modules in the pipeline.

19. A data processing device comprising:

a firmware socket module in communication with a data processing pipeline, said data processing pipeline being deployed on at least one reconfigurable logic device, and wherein the firmware socket module is configured to control the propagation of both commands and target data to the data processing pipeline via a communication path common to both the commands and the target data, wherein each command controls a data processing operation that is to be performed by the data processing pipeline, and wherein the target data

corresponds to the data upon which the data processing pipeline performs its commanded data processing operation.

20. The data processing device of claim 19 wherein the firmware socket module is also deployed on the at least one reconfigurable logic device.

21. The data processing device of claim 20 wherein the data processing pipeline comprises a plurality of firmware application modules that are connected in series, wherein the firmware socket module is configured to provide the commands and target data in a predetermined order to an entry point into the data processing pipeline, and wherein the predetermined order is maintained throughout the data processing pipeline.

22. The data processing device of claim 21 wherein the firmware socket module is configured to (1) access an external input descriptor pool buffer to identify where in an external memory space the commands and target data to be delivered to the data processing pipeline are stored, wherein the input descriptor pool buffer defines the order in which commands and target data are to be delivered to the data processing pipeline, and (2) transfer the identified commands and target data from the external memory space to the data processing pipeline in accordance with the defined order.

23. The data processing device of claim 22 wherein the reconfigurable logic device comprises a field programmable gate array (FPGA), wherein the firmware socket module has another communication path between itself and an exit point from the data processing pipeline, wherein the firmware socket module is further configured to receive an output from the data processing pipeline via the additional communication path.

24. The data processing device of claim 23 wherein the firmware socket module is further configured to (1) access a first external output descriptor pool buffer to identify a location in the memory space where command output from the data processing pipeline should be stored, (2) access a second external output descriptor pool buffer to identify a location in the memory space where target data output from the data processing pipeline should be stored, and (3) write the command output and the target data output from the data processing pipeline to their respective identified locations in the memory space.

25. A data processing system comprising:

a processor;

a memory space that is managed by the processor;

a reconfigurable logic device, the reconfigurable logic device comprising at least one firmware application module that is deployed thereon and a firmware socket module that interfaces the at least one firmware application module with the processor and the memory, the firmware socket module and the at least one firmware application module being in communication with each other via a communication path; and

a bus through which the reconfigurable logic device communicates with the processor and the memory space; and

wherein the firmware application module is configured to perform a data processing operation on a data stream that it receives;

wherein the firmware socket module is configured to receive both a plurality of commands and target data from the memory space;

wherein the firmware socket module is configured provide both the commands and the target data to the at least one firmware application module via the same communication path between itself and the at least one firmware application module, wherein each command controls the data processing operation that is performed by the at least one firmware

application module, and wherein the target data comprises the data upon which the at least one firmware application module's data processing operation is performed.

26. The system of claim 25 wherein the firmware socket module is further configured to (1) receive the commands and the target data as a stream of interleaved and ordered commands and target data, and (2) maintain the order or the ordered commands and target data when it provides the stream to the at least one firmware application module via the same communication path.

27. The system of claim 26 wherein the reconfigurable logic device comprises a field programmable gate array (FPGA).

28. The system of claim 27 wherein the at least one firmware application module comprises a plurality of firmware application modules that are arranged in a pipeline, wherein each of the firmware application modules in the pipeline are configured to sequentially process the commands and the target data of the stream in the same order in which the firmware socket module provided the commands and the target data.

29. The system of claim 28 wherein a plurality of the commands are specific to individual ones of the firmware application modules in the pipeline.

30. The system of claim 28 wherein the firmware application modules and the firmware socket module are arranged in a chain, wherein the firmware socket module provides the stream of interleaved and ordered commands and target data to the first one of the firmware application modules in the pipeline.

31. The system of claim 28 wherein the processor is further configured with a device driver that maintains a buffer in which data corresponding to a plurality of commands and data

corresponding to a plurality of data streams are stored in said order, and wherein the firmware socket module is configured to access the buffer to identify the commands and the data streams to be processed through the firmware application modules.

32. The system of claim 31 wherein the commands and the target data are stored in the memory space at a plurality of memory addresses, and wherein the data corresponding to the plurality of commands and the data corresponding to the target data that are stored in the buffer comprise a plurality of pointers to the memory addresses where the corresponding commands and target data are stored.

33. The system of claim 32 wherein the target data that is to be processed as a group through the chain is stored in contiguous memory addresses in the same order as which pointers thereto are stored in the buffer.

34. The system of claim 27 wherein the data processing operations comprise at least two members of the group consisting of a search operation, an encryption operation, a decryption operation, a compression operation, a decompression operation, and a data reduction operation.

35. A data processing card for connection with a bus of a computer system, the data processing card comprising:

a reconfigurable logic device, the reconfigurable logic device having deployed thereon a firmware application module pipeline and a firmware module;

wherein the firmware module is configured to interface the firmware application module pipeline with at least one external source of target data and commands, wherein the commands control how the firmware application module pipeline processes target data;

wherein the firmware application module pipeline comprises a plurality of firmware application modules arranged in a sequence, each firmware application module being configured to perform a data processing operation on target data, the firmware application module pipeline having an entry communication path from the firmware module to a first one of the firmware application modules in the sequence;

wherein the firmware module is further configured to control how the firmware application module pipeline processes target data by providing, through the entry communication path, the first one of the firmware application modules in the sequence with a stream of interleaved commands and target data, wherein the commands and target data have a defined order within the stream;

wherein the firmware module is configured to read entries that are stored in an external buffer to determine the commands and the target data that are to be provided to the firmware application module pipeline, wherein the order of entries in the external buffer defines the order of commands and target data within the stream; and

wherein the firmware application module pipeline is configured to sequentially process the commands and target data of the stream through the sequence of firmware application modules in the same order in which those commands and target data were ordered within the stream.

36. The data processing card of claim 35 wherein the external buffer entries comprise pointers to addresses in the at least one external source where the commands and target data are stored, wherein the firmware module is further configured to access the at least one external source to retrieve, from the addresses, the commands and target data in accordance with the defined order.

37. The data processing card of claim 36 wherein a plurality of the commands are specific to individual ones of the firmware application modules in the sequence.

38. The data processing card of claim 37 wherein at least one of the commands comprises a pass through command to one of the firmware application modules in the sequence..

39. The data processing card of claim 37 wherein the data processing operations defined by the commands comprise at least two members of the group consisting of a search operation, a data reduction operation, an encryption operation, a decryption operation, a compression operation, a decompression operation, and a pass through operation.

40. The data processing card of claim 39 wherein the firmware application module pipeline further has an exit communication path from a final one of the firmware application modules in the sequence to the firmware module, wherein each data procession operation performed by the final one of the firmware application modules in the sequence produces an output, and wherein the firmware module is further configured to (1) receive the outputs from the final one of the firmware application modules in the sequence via the exit communication path, (2) read entries that are stored in an external buffer to determine a plurality of locations in the at least one external source where the outputs are to be stored, and (3) write the outputs to the at least one external source at the determined locations.

41. The data processing card of claim 35 wherein the reconfigurable logic device comprises a field programmable gate array (FPGA).

42. A system controlling the processing of data through a reconfigurable logic device via software, the system comprising:

device driver software for execution by a processor, the device driver software being configured to manage a memory space in which data is stored at a plurality of memory addresses; and

a reconfigurable logic device for communicating with the device driver software and the memory space over a bus;

wherein at least one firmware application module and a firmware socket module are deployed on the reconfigurable logic device;

wherein the at least one firmware application module is configured to perform a data processing operation on any target data that it receives;

wherein the device driver software is configured to store a plurality of commands in the memory space, each command for controlling a data processing operation that is to be performed by the at least one firmware application module;

wherein the device driver software is further configured to maintain an input descriptor pool buffer in the memory space that defines an order in which commands and data are to be processed through the at least one firmware application module; and

wherein the firmware socket module is configured to (1) access the input descriptor pool buffer to identify the ordered commands and data that are to be processed through the at least one firmware application module, and (2) access the memory space to transfer the commands and data to the at least one firmware application module in accordance with the defined order.

43. The system of claim 42 wherein the at least one firmware application module comprises a plurality of firmware application modules that are arranged in a pipelined sequence, the pipelined sequence including a first firmware application

module, wherein the firmware socket module is in communication with the first firmware application module of the pipelined sequence via a communication path, wherein the firmware socket module is further configured to transfer the commands and data to the first firmware application module in the pipelined sequence over the same communication path.

44. The system of claim 43 wherein each command includes a firmware application module identifier that identifies the firmware application module in the pipelined sequence to which that command is applicable.

45. The system of claim 44 wherein the device driver software is further configured to maintain the input descriptor pool buffer by storing therein a plurality of pointers to memory addresses in the memory space where the commands and data to be processed through the pipelined sequence of firmware application modules are stored, wherein the order of pointers in the input descriptor pool buffer defines the order in which commands and data are provided to the pipelined sequence of firmware application modules by the firmware socket module.

46. The system of claim 45 wherein the reconfigurable logic device comprises a field programmable gate array (FPGA), the system further comprising application software for execution by the processor and module application programming interface software for execution by the processor, wherein the application software is configured to receive input from a user that specifies (1) at least one data processing operation to be performed by the pipelined sequence of firmware application modules and (2) the data upon which the pipelined sequence of firmware application modules will perform the at least specified data processing operation, and wherein module application programming interface software is configured to interface the application software with the device driver software such that the device driver software appropriately

manages the input descriptor pool buffer and the memory space in accordance with the user input.

47. The system of claim 45 wherein the device driver software is further configured to store the data to be processed through the pipelined sequence in contiguous memory addresses of the memory space in same order as that data is to be provided to the pipelined sequence of firmware application modules.

48. A device for interfacing a pipeline of firmware application modules that are deployed on at least one reconfigurable logic device with a memory space that is external to the reconfigurable logic device, the device comprising:

a firmware module that is deployed the at least one reconfigurable logic device, wherein the firmware module is configured to (1) serially access the external memory space to retrieve a plurality of commands and a plurality of groups of target data in an order and (2) deliver both the retrieved commands and target data groups to an entry point in the pipeline as a single interleaved stream in the same order over a common communication path that connects the firmware module with the entry point into the pipeline, wherein each firmware application module in the pipeline is configured to perform a data processing operation on any target data that it receives, and wherein each command is configured to control the data processing operation of at least one firmware application module in the pipeline.

49. The device of claim 48 wherein the firmware module is further configured determine which commands and which target data groups are to be retrieved from the external memory space by reading an ordered set of descriptors that are stored in the external memory space by a software program executed by an external processor, wherein the ordered set of descriptors

defines the order in which commands and target data groups are to be interleaved in the stream.

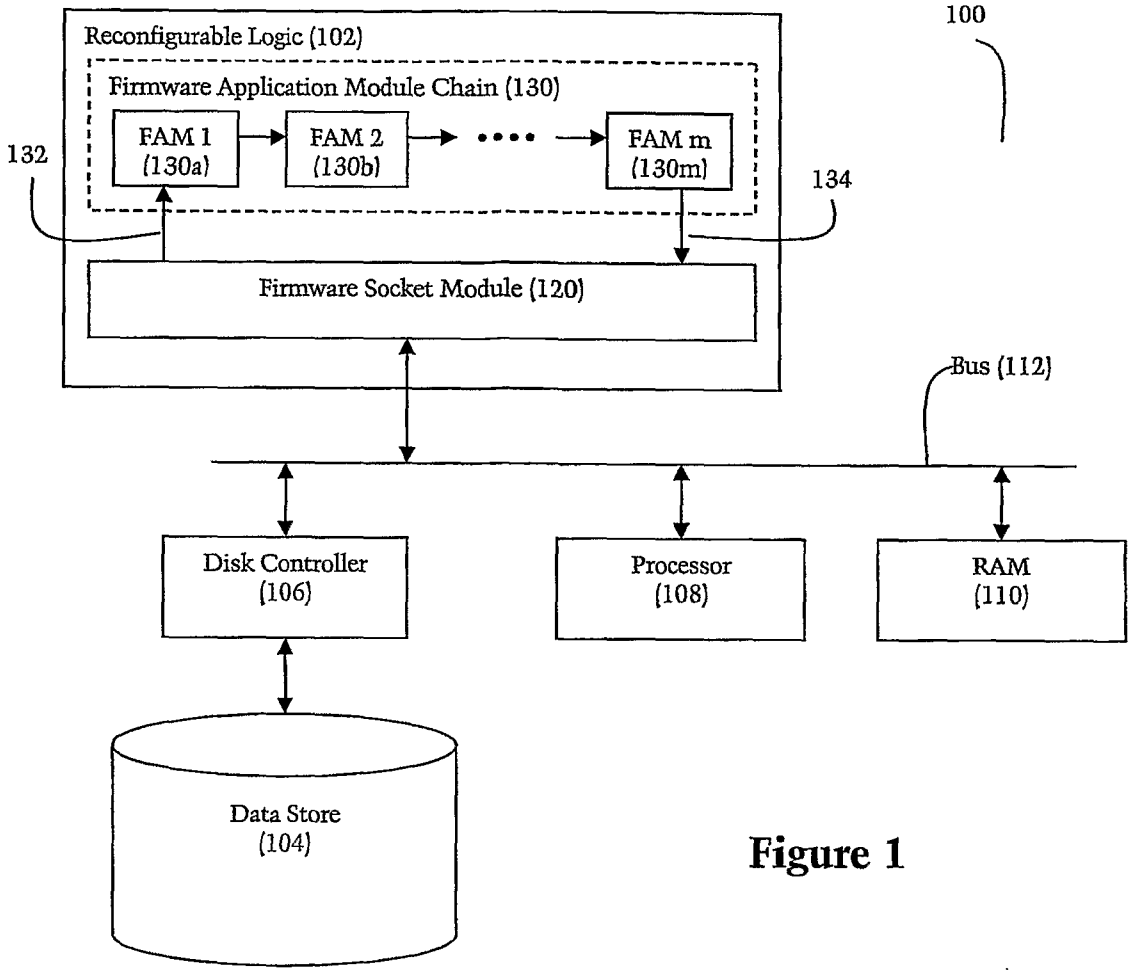


Figure 1

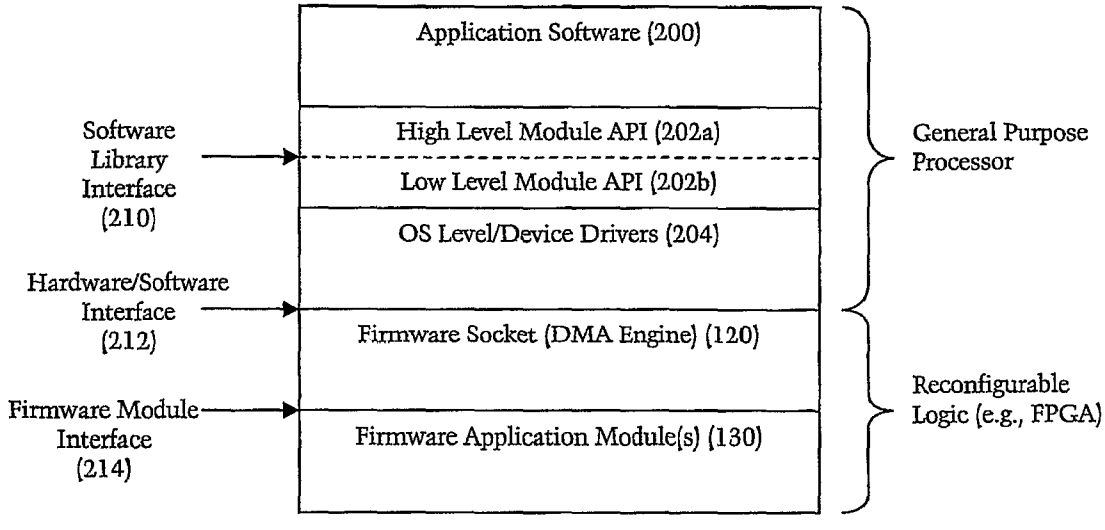


Figure 2

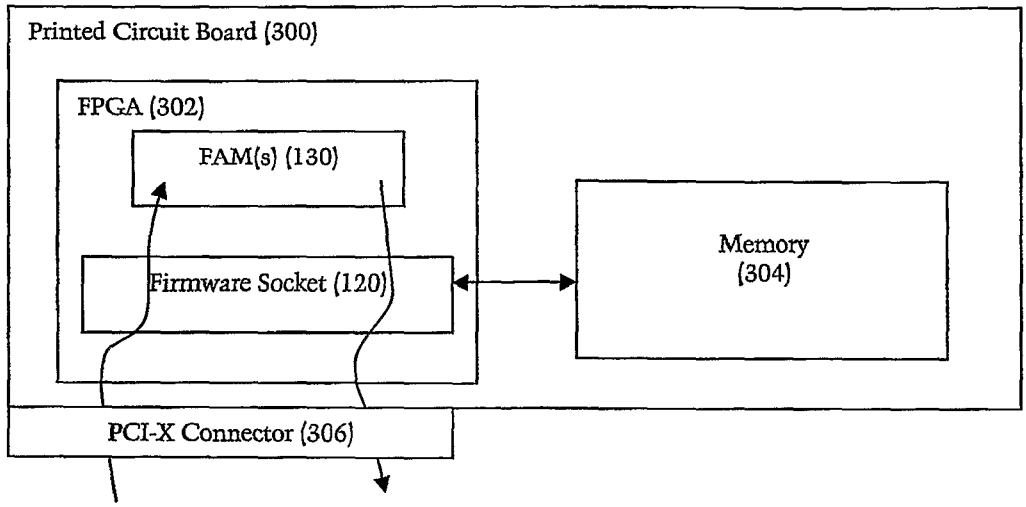


Figure 3(a)

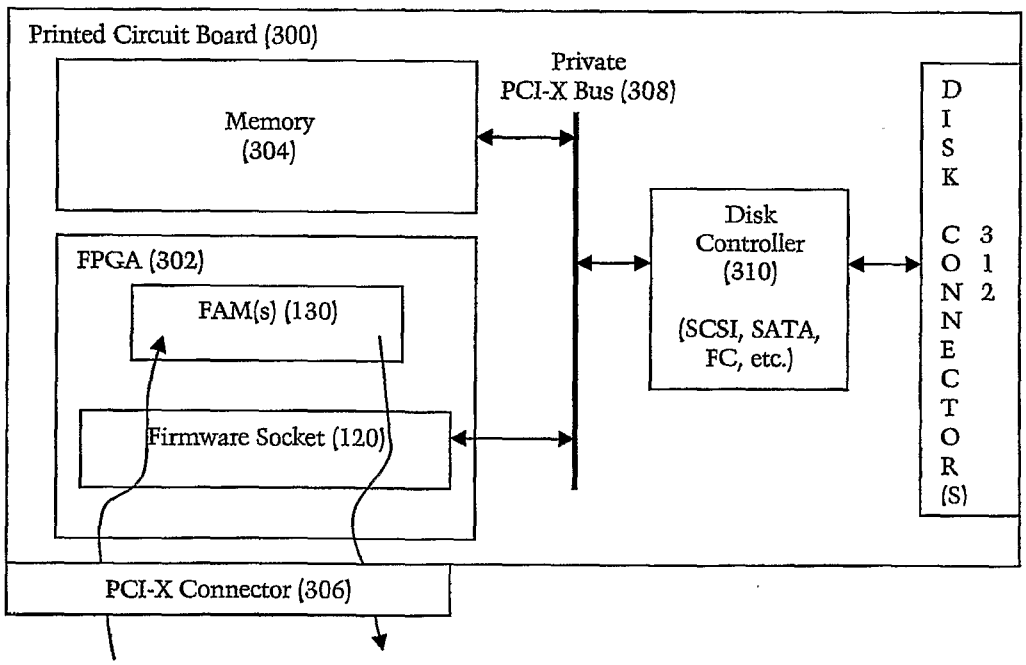


Figure 3(b)

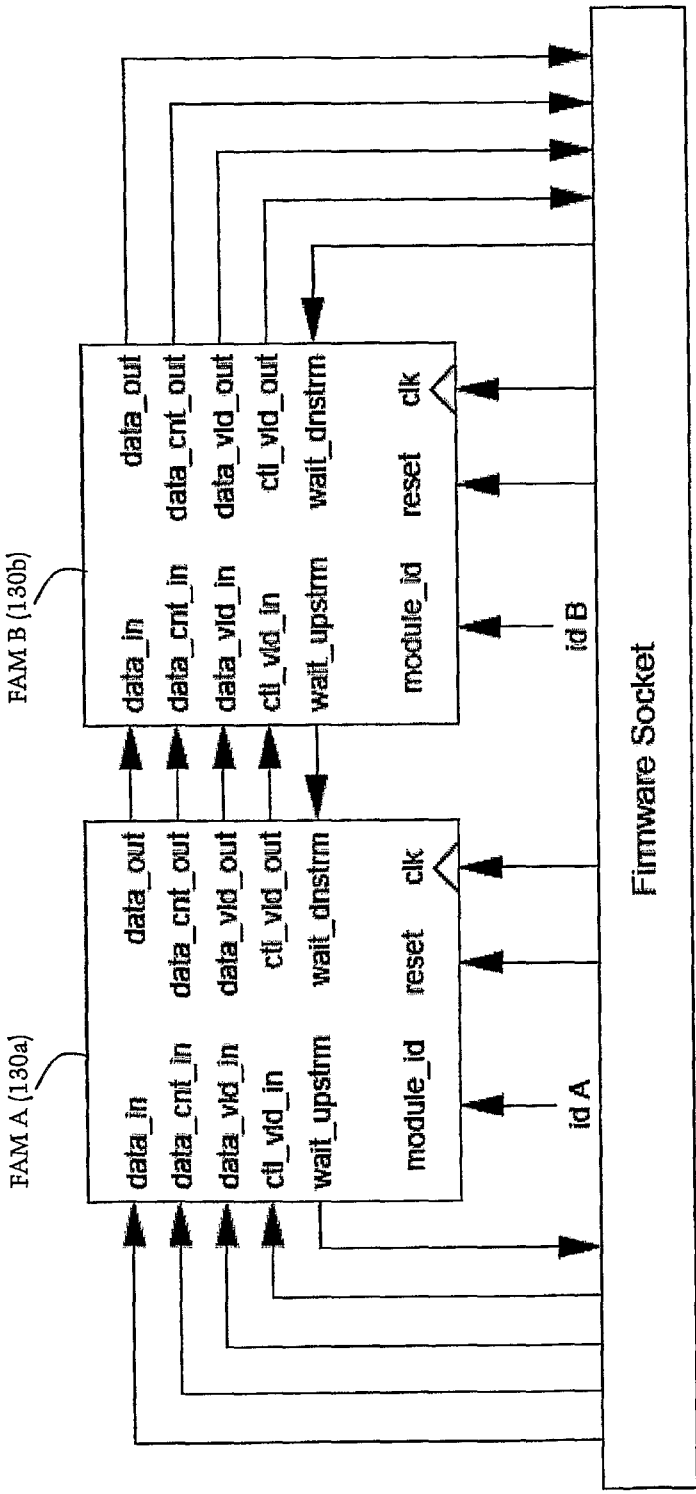

FAM A (130a)
FAM B (130b)
data_in
data_cnt_in
data_vld_in
ctl_vld_in
wait_upstrm
module_id
data_out
data_cnt_out
data_vld_out
ctl_vld_out
wait_dnstrm
reset
clk
id A
id B
Firmware Socket
120

Figure 4

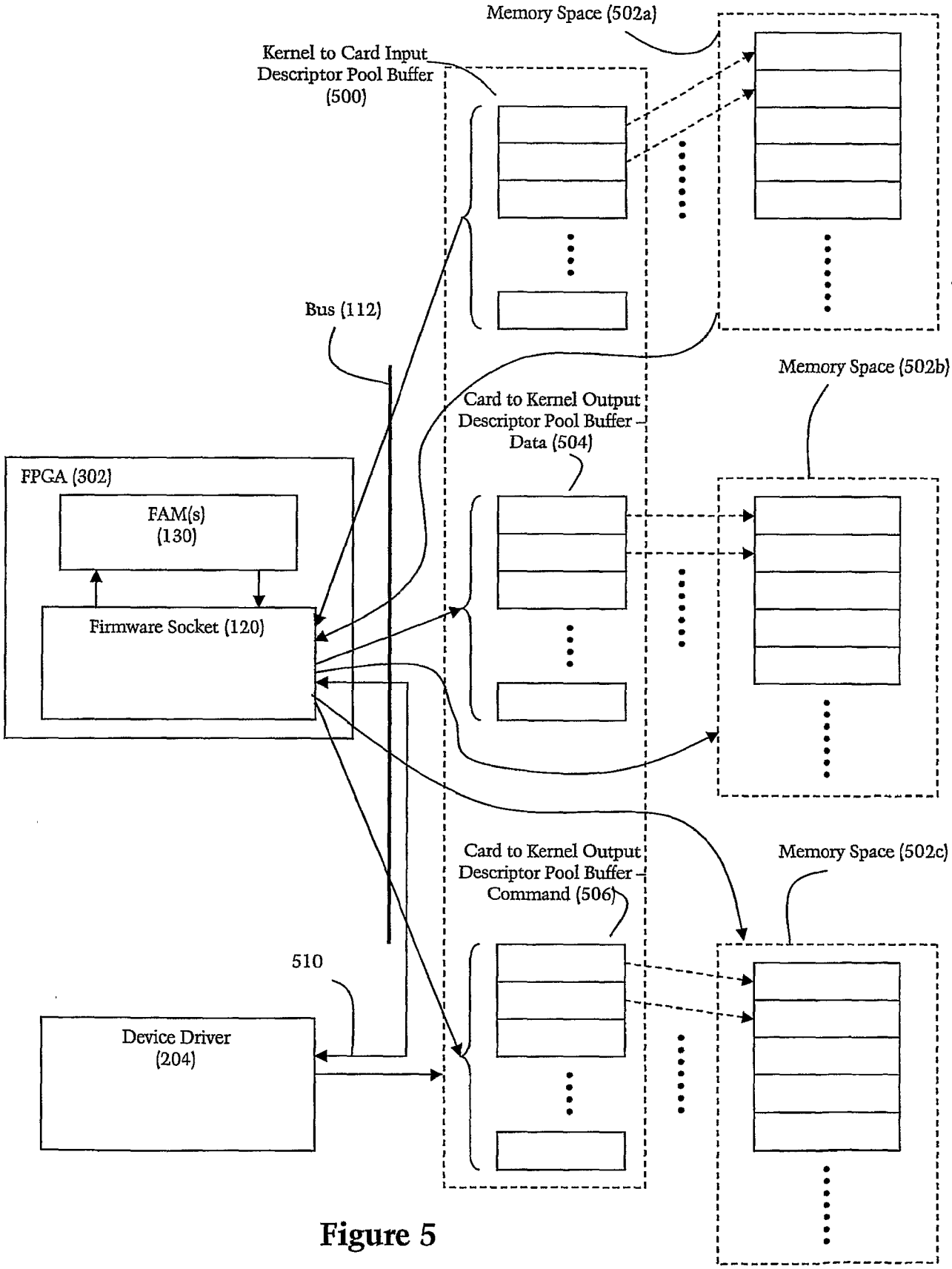

Memory Space (502a)
Kernel to Card Input Descriptor Pool Buffer (500)
Bus (112)
Memory Space (502b)
Card to Kernel Output Descriptor Pool Buffer Data (504)
FPGA (302)
FAM(s) (130)
Firmware Socket (120)
Card to Kernel Output Descriptor Pool Buffer Command (506)
Memory Space (502c)
510
Device Driver (204)

Figure 5

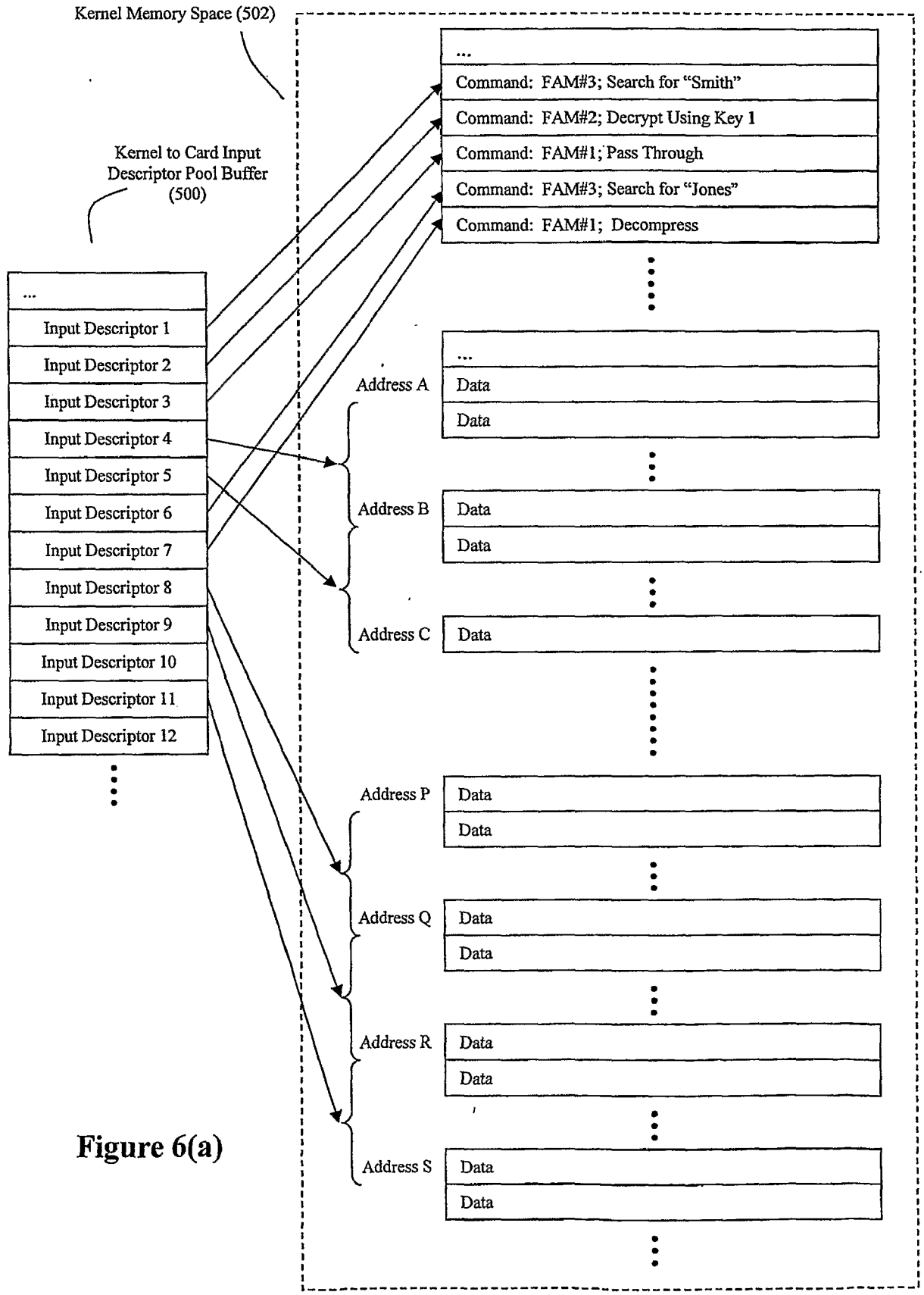

Kernel Memory Space (502)
Kernel to Card Input Descriptor Pool Buffer (500)
...
Input Descriptor 1
Input Descriptor 2
Input Descriptor 3
Input Descriptor 4
Input Descriptor 5
Input Descriptor 6
Input Descriptor 7
Input Descriptor 8
Input Descriptor 9
Input Descriptor 10
Input Descriptor 11
Input Descriptor 12
...
Command: FAM#3; Search for "Smith"
Command: FAM#2; Decrypt Using Key 1
Command: FAM#1; Pass Through
Command: FAM#3; Search for "Jones"
Command: FAM#1; Decompress
Address A
Address B
Address C
Address P
Address Q
Address R
Address S
Data

Figure 6(a)

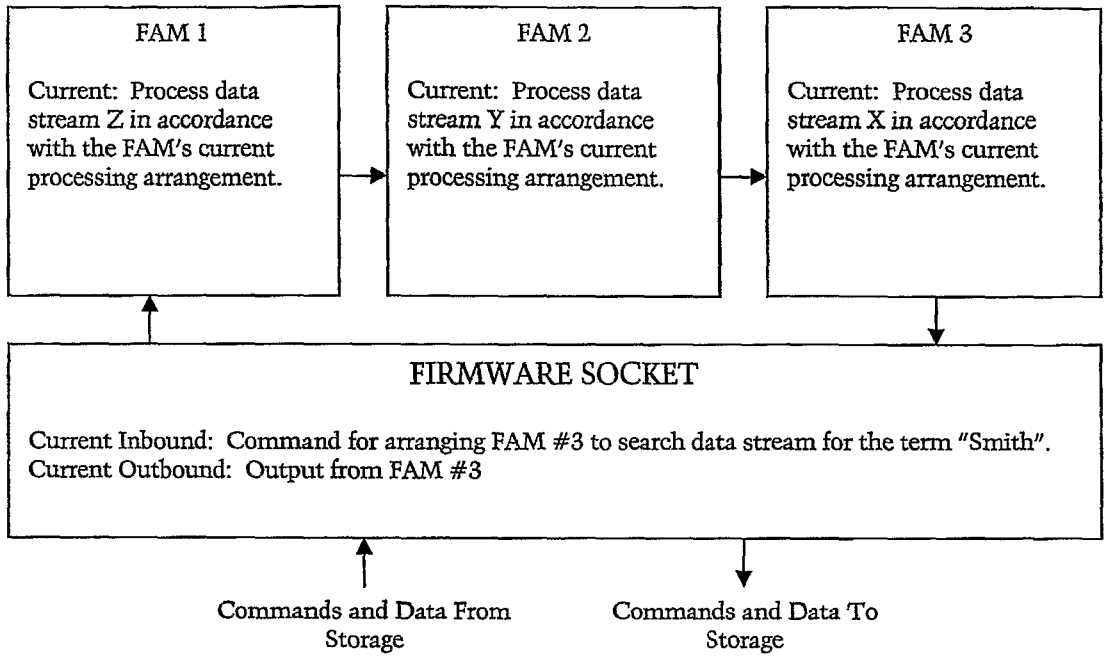


Figure 6(b)

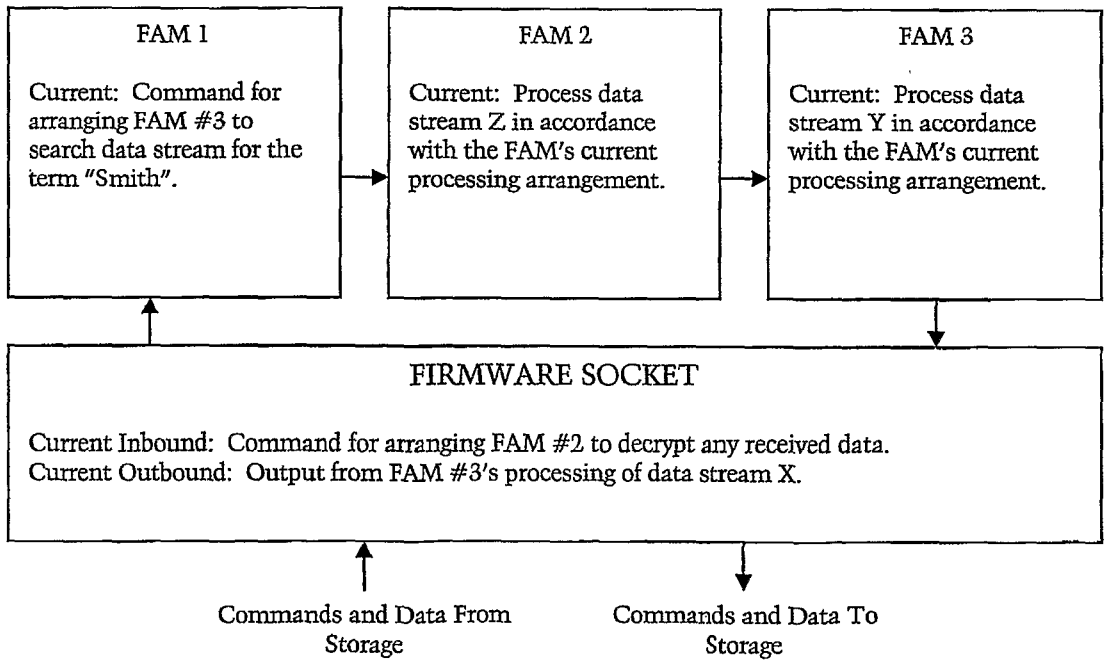


Figure 6(c)

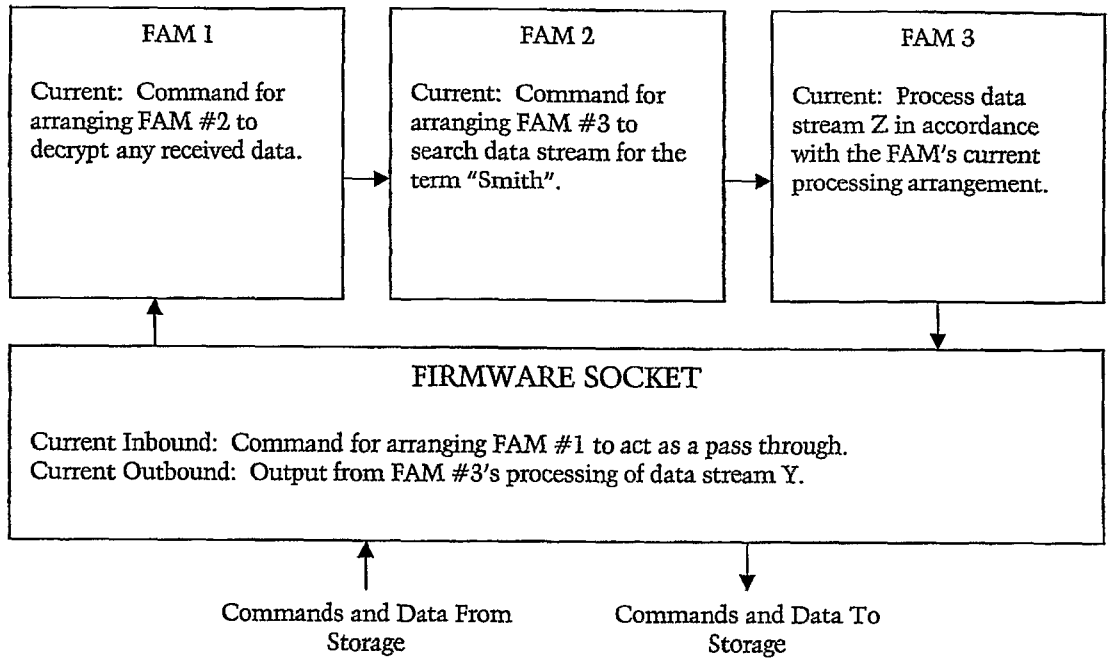


Figure 6(d)

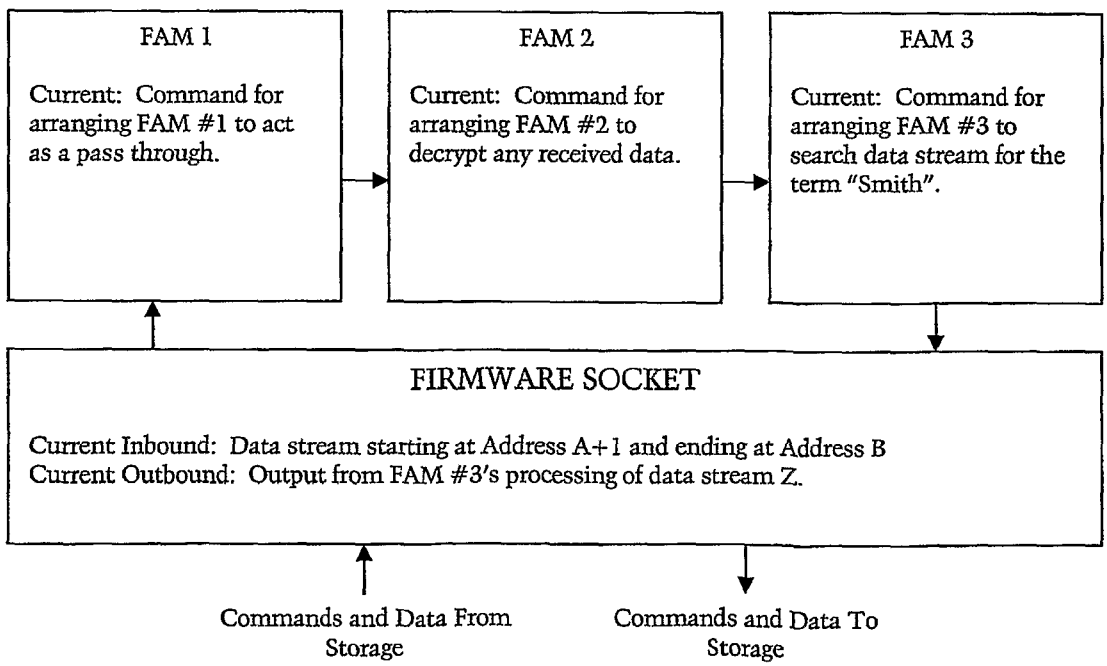


Figure 6(e)

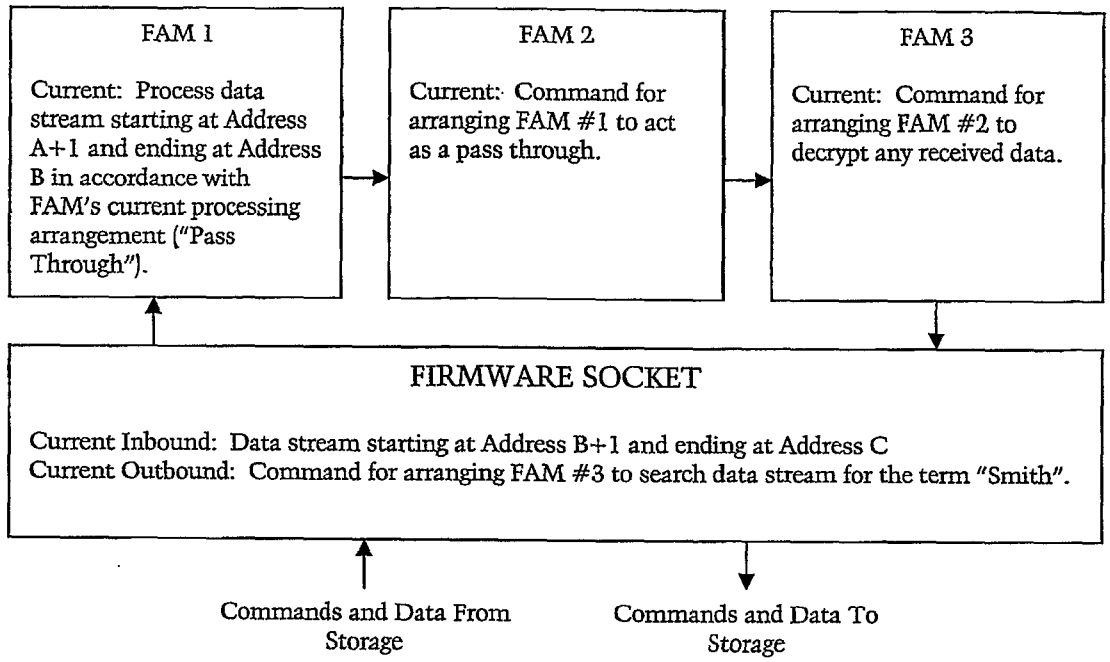


Figure 6(f)

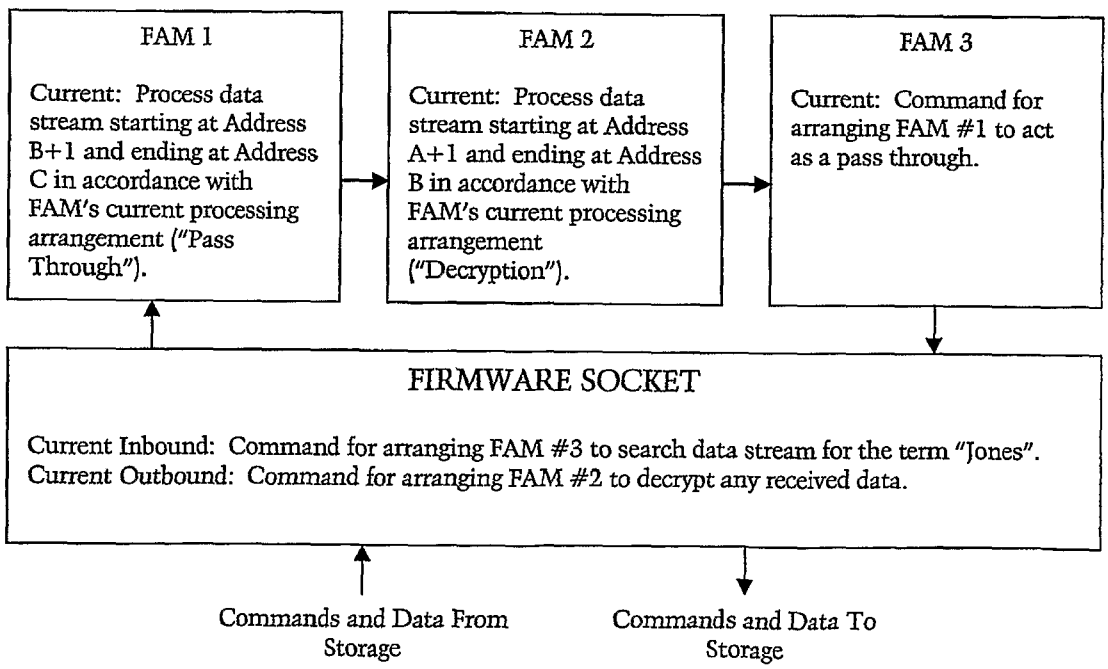


Figure 6(g)

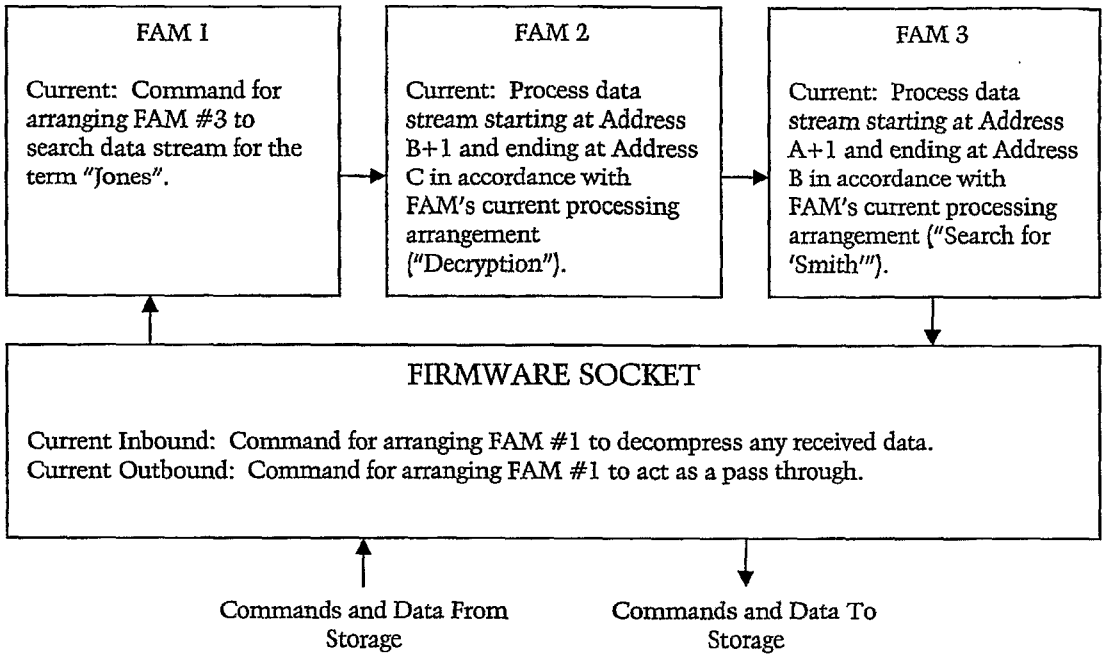


Figure 6(h)

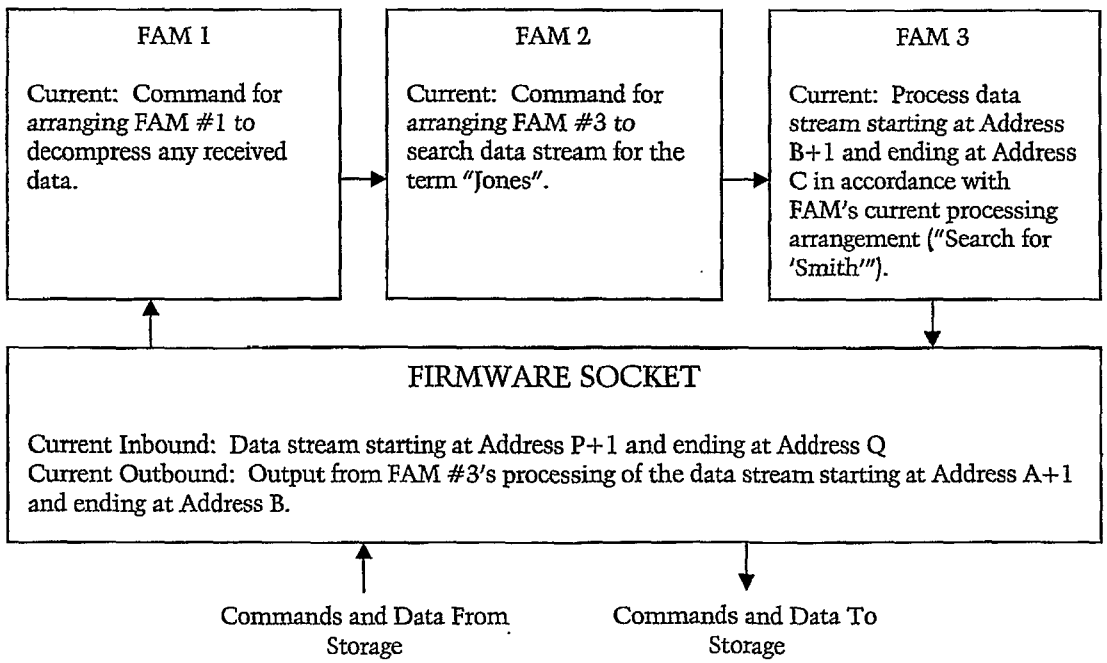


Figure 6(i)

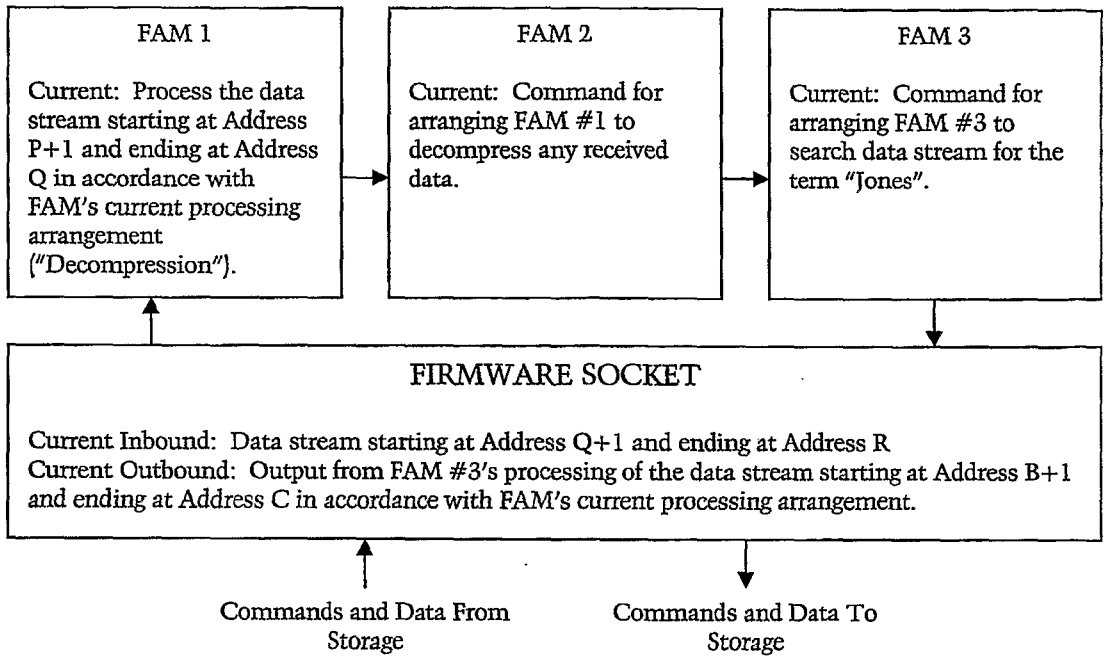


Figure 6(j)

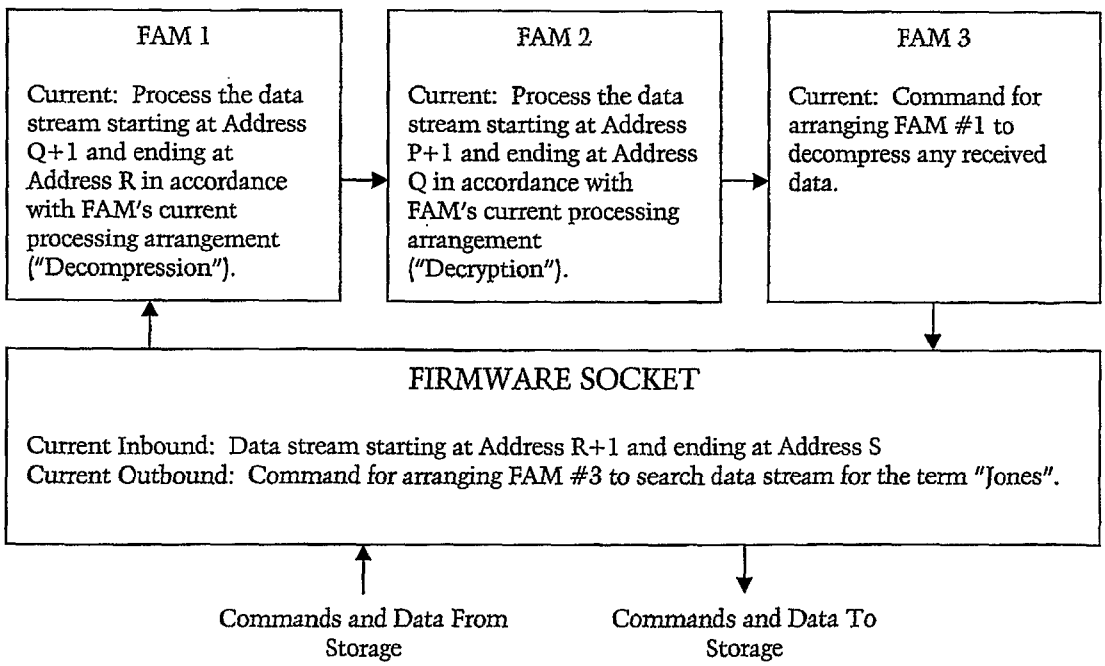


Figure 6(k)

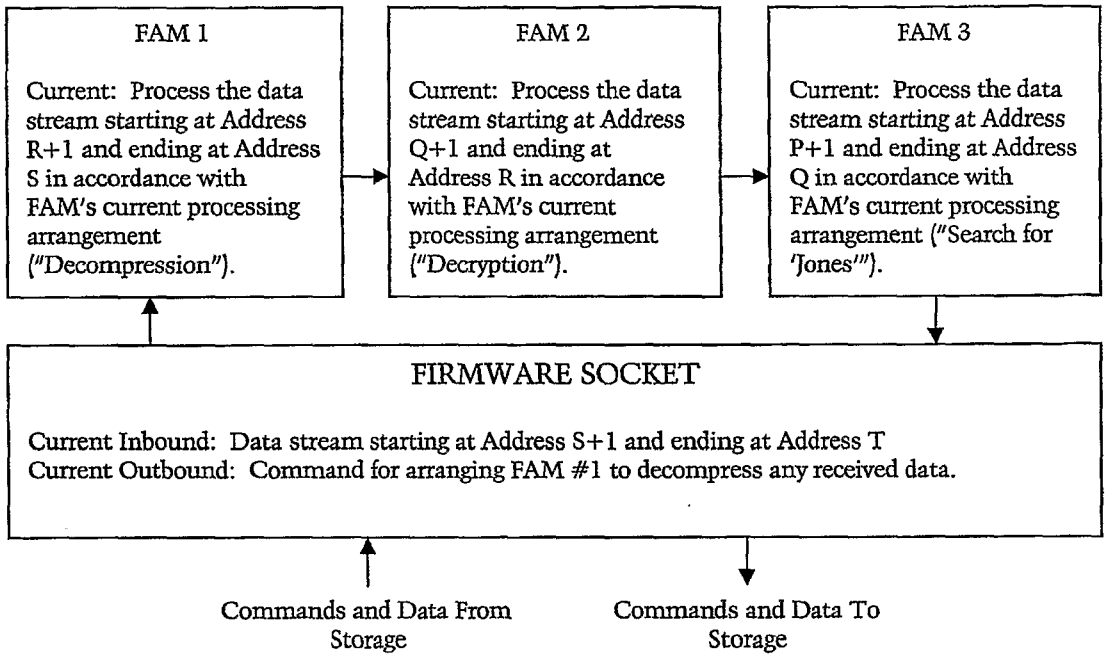


Figure 6(l)

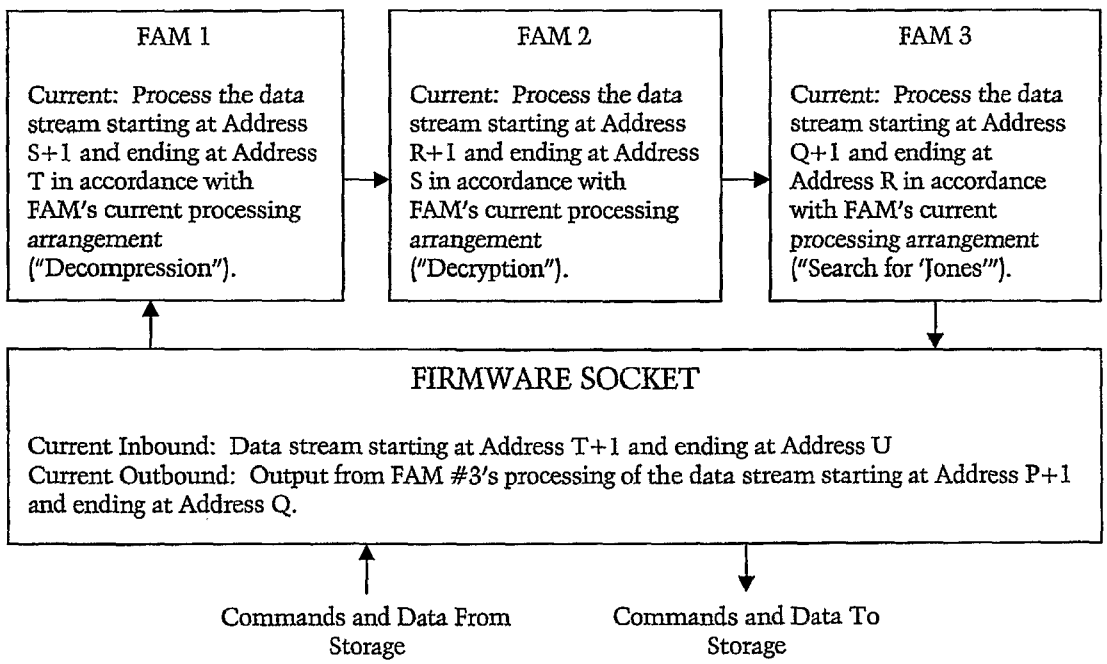


Figure 6(m)

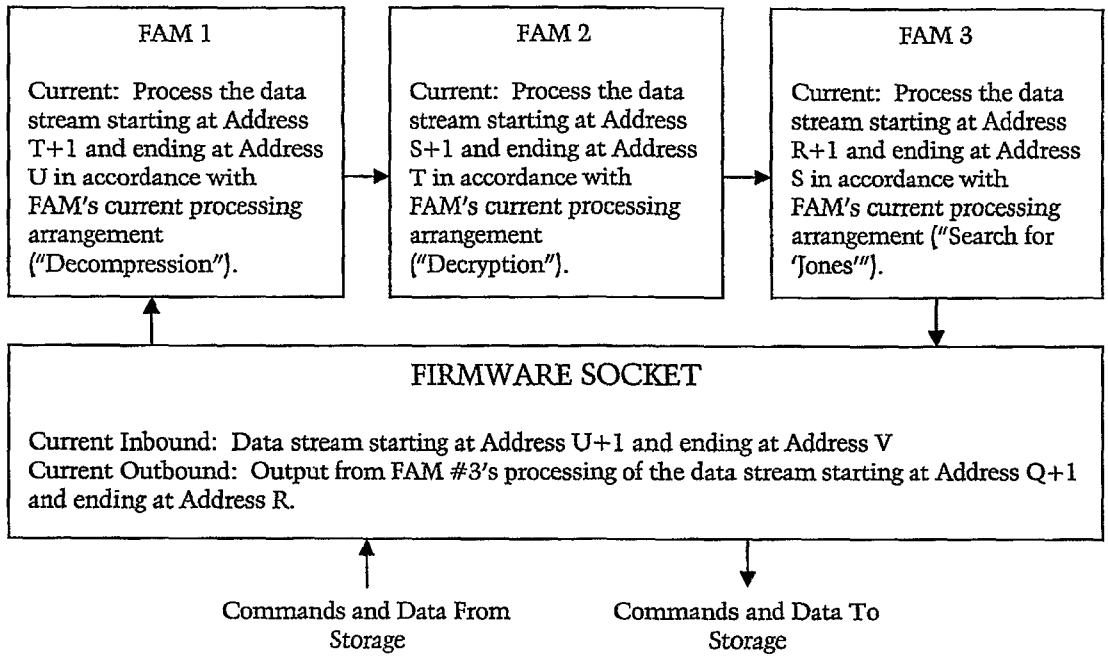


Figure 6(n)

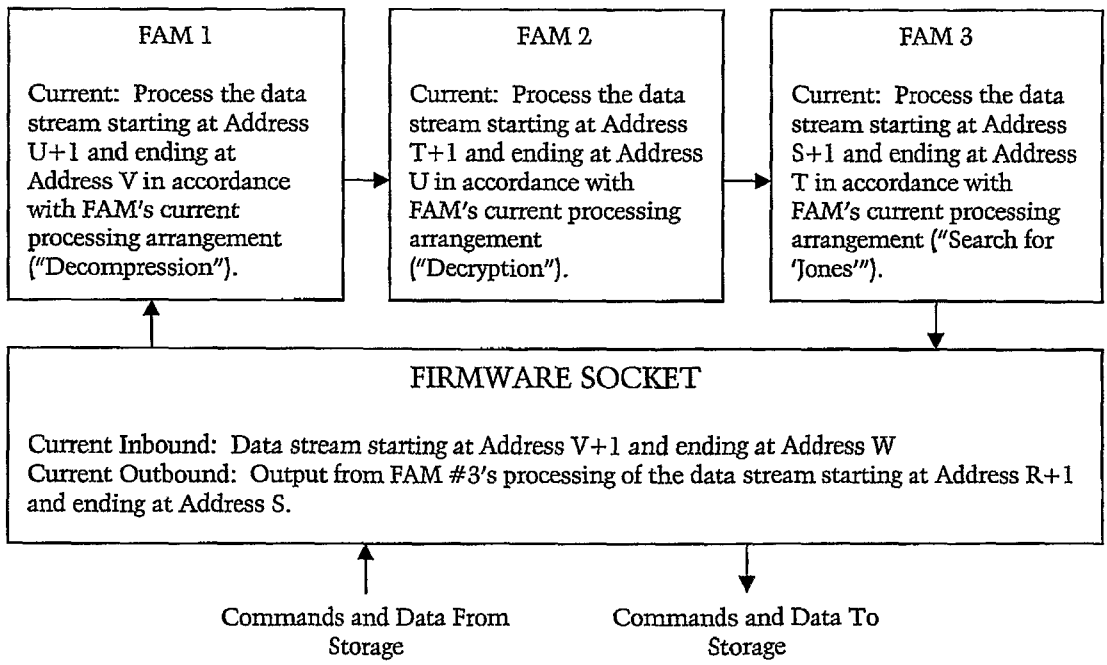


Figure 6(o)

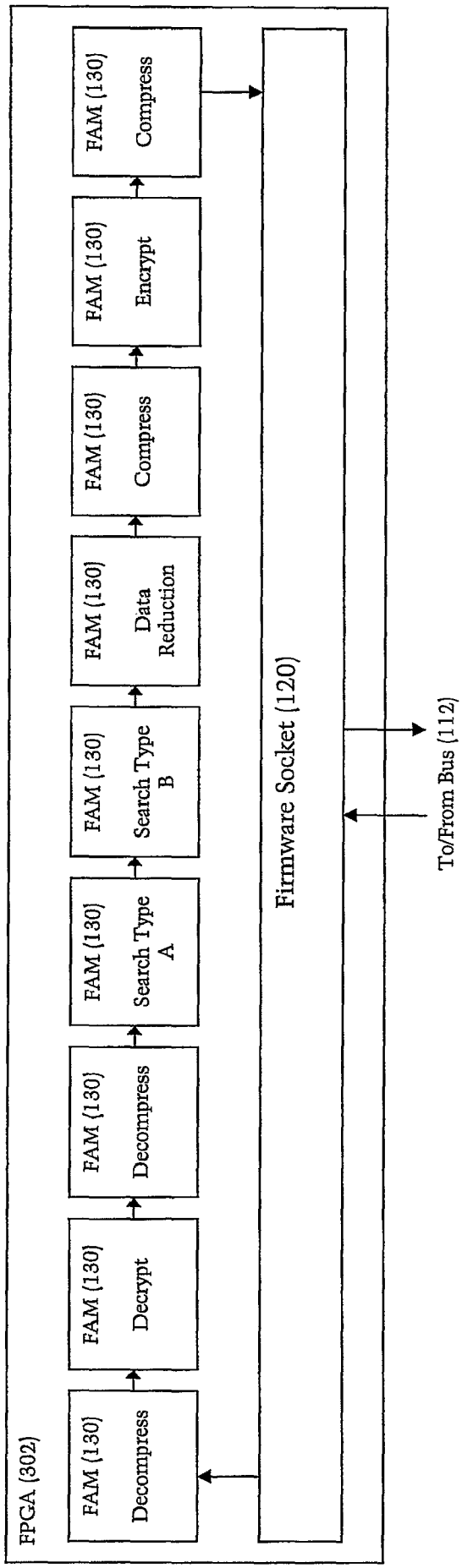


Figure 7

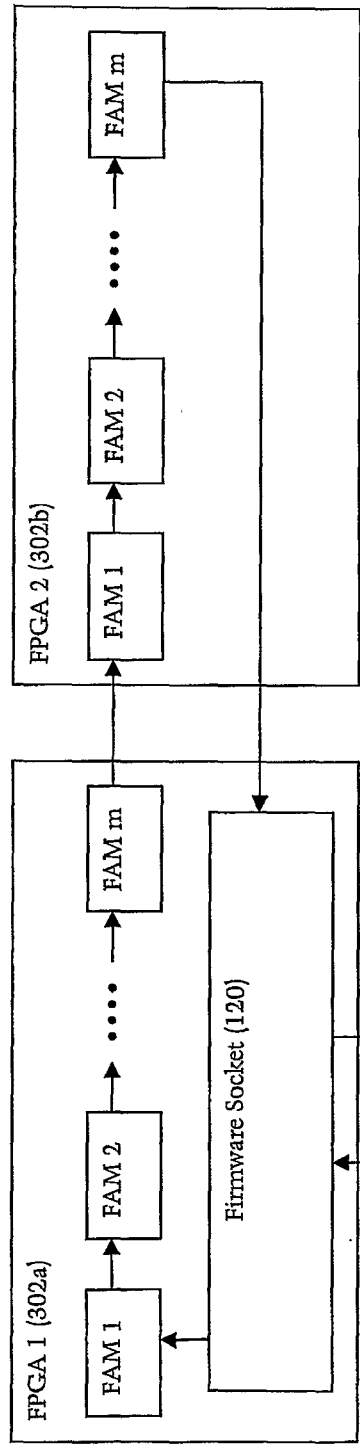


Figure 8