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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
06 August 2020 (06.08.2020)

WIPO | PCT

(10) International Publication Number
WO 2020/157594 A1

(51) International Patent Classification:
G06F 13/10 (2006.01)

(21) International Application Number:
PCT/IB2020/050339

(22) International Filing Date:
16 January 2020 (16.01.2020)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
19154740.5 31 January 2019 (31.01.2019) EP

(71) Applicant: **INTERNATIONAL BUSINESS MACHINES CORPORATION** [US/US]; New Orchard Road, Armonk, New York 10504 (US).

(71) Applicants *(for MG only)*: **IBM DEUTSCHLAND GMBH** [DE/DE]; c/o IBM Deutschland Management & Business Support GmbH, Patentwesen und Urheberrecht, IBM Allee 1, 71139 Ehningen (DE). **IBM (CHINA) INVESTMENT COMPANY LTD.** [CN/CN]; 25/F, Pangu Plaza, No. 27, Central North 4th Ring Road, Chaoyang District, Beijing, Beijing 100101 (CN).

(72) Inventors: **RAISCH, Christoph**; c/o IBM Deutschland Research & Development GmbH, Schoenaicher Strasse 220, 71032 Boeblingen (DE). **KRAEMER, Marco**; c/o IBM Deutschland Research & Development GmbH, Schoenaicher Strasse 220, 71032 Boeblingen (DE). **LEHNERT, Frank**; c/o IBM Deutschland Research & Development GmbH, Schoenaicher Strasse 220, 71032 Boeblingen (DE). **KLEIN, Matthias**; c/o IBM Corp., 2455 South Road, Poughkeepsie, New York 12601 (US). **BRADBURY, Jonathan**; c/o IBM Corp., 2455 South Road, Poughkeepsie, New York 12601 (US). **JACOBI, Christian**; c/o IBM Corp., 2455 South Road, Poughkeepsie, New York 12601 (US). **BELMAR, Brenton**; c/o IBM

(54) Title: HANDLING AN INPUT/OUTPUT STORE INSTRUCTION

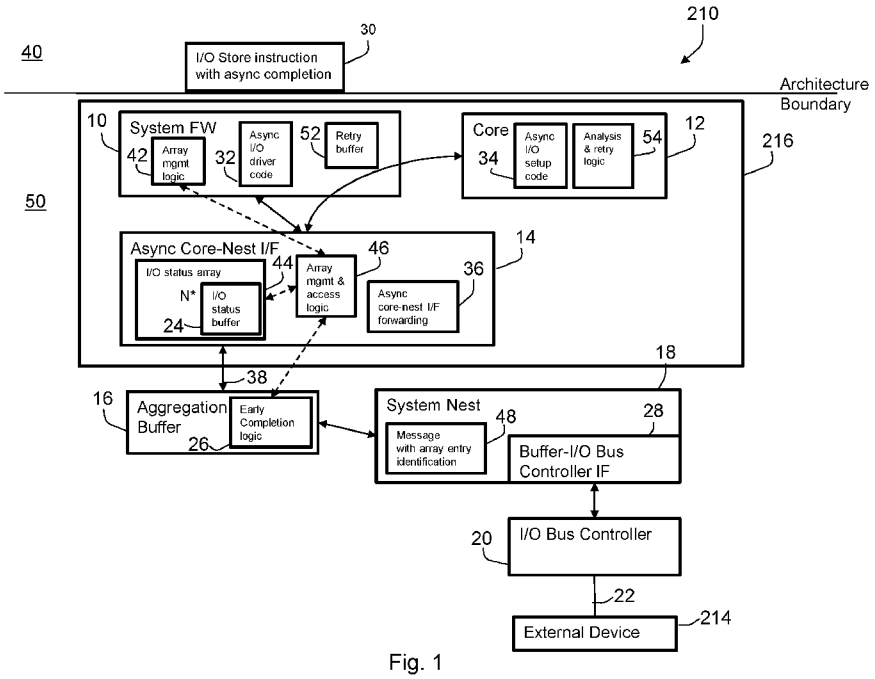

Fig. 1

(57) Abstract: A data processing system (210) and a method for handling an input/output store instruction (30), comprising a system nest (18) coupled to at least one input/output bus (22) by an input/output bus controller (20). A data processing unit (216) is coupled to the system nest (18) via an aggregation buffer (16). A system nest (18) is configured to asynchronously load from and/or store data to at least one external device (214). The data processing unit (216) is configured to complete the input/output store instruction (30) before an execution of the input/output store instruction (30) in the system nest (18) is completed. An asynchronous core-nest interface (14) comprises an input/output status array (44) with multiple input/output status buffers (24). A system firmware (10) comprises a retry buffer (52) and the core (12) comprises an analysis and retry logic (54).

[Continued on next page]

WO 2020/157594 A1

Corp., 2455 South Road, Poughkeepsie, New York 12601 (US). **DRIEVER, Peter**; c/o IBM Corp., 2455 South Road, Poughkeepsie, New York 12601 (US).

(74) **Agent: DOEHLER, Denis**; c/o IBM Deutschland Management & Business Support GmbH, Patentwesen und Urheberrecht, 71139 Ehningen (DE).

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

(84) **Designated States** *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, ST, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

Published:

— *with international search report (Art. 21(3))*

HANDLING AN INPUT/OUTPUT STORE INSTRUCTION

[0001] The present invention relates in general to data processing systems, in particular to a method for handling an input/output store instruction to multiple external devices as well as a computer program product and a data processing system.

BACKGROUND

[0002] A computing environment may include one or more types of input/output devices, including various types of adapters. One type of adapter is a Peripheral Component Interconnect (PCI) or Peripheral Component Interconnect Express (PCIe) adapter. This adapter includes one or more address spaces used in communicating data between the adapter and the system to which the adapter is attached.

[0003] In some systems, a portion of an address space of the central processing unit (CPU) coupled to the adapter is mapped to an address space of the adapter enabling CPU instructions that access storage to directly manipulate the data in the adapter's address space.

[0004] Communication with adapters, such as PCI or PCIe adapters can be facilitated by control instructions specifically designed for communicating data to and from adapters and used for communication.

[0005] In the state of the art, a store instruction for storing data in an adapter includes, for instance, obtaining a machine instruction for execution, the machine instruction being defined for computer execution according to a computer architecture, the machine instruction including, for instance, an opcode field identifying a store to adapter instruction. A first field identifies a first location that includes data to be stored in an adapter. A second field identifies a second location, the contents of which include a function handle identifying the adapter, a designation of an address space within the adapter in which data is to be stored, and an offset within the address space. The machine instruction is executed, the executing including using the function handle to obtain a function table entry associated with the adapter. A data address of the adapter is obtained using at least one of information

in the function table entry and the offset. Data are stored from the first location in a specific location in the address space identified by the designation of the address space, the specific location identified by the data address of the adapter.

[0006] An existing feature in a large multi-processor system is the ability to quiesce all processors within a target zone. Quiesce functions operate to temporarily pause or alter the state of a processor or group of processors to perfom1, e.g., system updates or backups. In some instances, a quiesce interruption is applicable to only a subset of the system resources. In such instances, the system can be divided into different zones. For a quiesce operation applicable to one zone (a target zone), processors outside of the target zone are permitted to continue running, although new translations may be blocked. Typically, at least one system controller or other mechanism broadcasts the quiesce to all physical processors in the system, handles collecting quiesce state information and indicates to a requesting processor when all processors have started, or are ignoring (filtering) the quiesce request.

[0007] A quiesce controller can be communicatively coupled to a processor in a multi-processor system, and a quiesce state machine configured to receive a quiesce request. The computer system is configured to perform a method that includes receiving a quiesce request at the quiesce controller from a requesting processor, the requesting processor being one of a plurality of processors in a multi-processor system, and determining that the quiesce request is not accepted based on a state of the quiesce state machine. The method also includes, based on the request being not accepted, generating a reject message configured to indicate that the quiesce request has been rejected, holding the reject message 1mtil a quiesce command is broadcast to the multi-processor system, the quiesce command based on a different quiesce request, and sending the reject message to the requesting processor based on the broadcast of the quiesce command being detected by the quiesce controller.

SUMMARY

[0008] A data processing system is proposed for handling an input/output store instruction, comprising a system nest communicatively coupled to at least one input/output bus by an input/output bus controller. The data processing system further comprises at least

a data processing unit comprising a core,
a system firmware and an asynchronous core-nest interface. The data processing unit is communicatively coupled to the system nest via an aggregation buffer. The system nest is configured to asynchronously load from and/or store data to at least one external device which is communicatively coupled to the input/output bus. The asynchronous core-nest interface comprises an input/output status array with multiple input/output status buffers, as well as an array management and access logic. The system firmware further comprises a retry buffer and the core comprises an analysis and retry logic.

[0009] The data processing system is configured to perform: (i) an operating system running on the data processing system issues the input/output store instruction specifying at least an input/output function with an offset through an address, data to be transferred and/or a pointer to data to be transferred, and a length of the data; (ii) the data processing unit is configured to identify the input/output function by the address specified in the input/output store instruction; (iii) the data processing unit is configured to verify if access to the input/output function is allowed on an address space and on a guest instance level, the guest running on the data processing system; (iv) the data processing unit is configured to complete the input/output store instruction before an execution of the input/output store instruction in the system nest is completed; (v) the system firmware is configured to notify the operating system through an interrupt, if during the asynchronous execution of the input/output store instruction an error is detected by the data processing unit, transmitting the data of the failed asynchronous execution; (vi) the analysis and retry logic detecting separately errors being ensured by the hardware that the store instruction has not been forwarded to an input/output bus yet; (vii) the retry buffer keeping store information for retries of executing the store instruction in system hardware/firmware; (viii) the analysis and retry logic analyzing errors and checking for retry possibility; and (ix) the analysis and retry logic triggering retries.

[0010] Favorably an asynchronous, but related to instruction execution, error detection and recovery device for a store instruction and a barrier instruction is added to the data processing system. All related technology within the data processing system may store related error detection and recovery for store instruction synchronously. Other errors, e.g. link drop, error reported by an input/output device, are detected and handled independently from store instruction execution.

[0011] According to an embodiment of the invention retryable errors are introduced, wherein a retry is performed within hardware and firmware below an architecture boundary.

[0012] The data processing system according to a first embodiment of the invention comprises instructions loading from and storing to at least one external device of the data processing system via an input/output bus. Asynchronous instructions complete before data has been stored to the external device while synchronous instructions complete after data has been stored to the external device. Within the embodiments described here, PCI will be used interchangeably for any other input/output technology, thus not restricting the embodiment of the invention to PCI.

[0013] Embodiments of the invention describe an input/output store instruction execution in a strictly ordered way as observable from above the architecture boundary while the actual execution may be out of order within the hardware of the data processing unit (CPU).

[0014] According to embodiments of the invention a PCI store instruction may be executed with an asynchronous execution of the PCIe store effect and an asynchronous status handling. Asynchronous reliable execution is based on reliable forwarding mechanisms in microarchitecture of the inventive data processing system.

[0015] An existing PCI store and store block instruction is usually synchronous up to the point where the PCI store data has been delivered to the PCIe interface and completion returned to a processing unit.

[0016] PCI standard only requires an asynchronous send command of PCI information, which is typically implemented through a store queue in the processor aggregating data with asynchronous send-out.

[0017] Advantageously, according to embodiments of the invention, an improvement concerning cycles per instruction may be achieved by replacing a synchronous PCI instruction by a reliable asynchronous send process of an input/output store instruction.

[0018] Alternatively or additionally of data to be transferred, the store instruction

according to an embodiment of the invention may also specify a pointer to a main memory which should be used to fetch data from, instead of containing the data directly.

[0019] Guest instance level may also mean that a single guest or host may be running on the data processing system.

[0020] The address of the offset of the input/output function itself can be a virtual, physical, logical address. Virtual and logical addresses typically get translated through a memory management unit (MMU) into a physical address, and the physical address then allows to identify which function and offset is meant.

[0021] Physical address in this context means "lowest address in the address translation hierarchy accessible from within a guest/operating system".

[0022] Advantageously, the input/output status buffers may collect returned states from the system nest and/or from the input/output bus controller, in particular a completion message from the system nest. These input/output status buffers may collect the returned states acting as an asynchronous system message buffer supporting the asynchronous transmit process. Advantageously the input/output status buffers may be integrated directly in the asynchronous core-nest interface for quick response.

[0023] According to a favourable embodiment of the inventive data processing system, the analysis and retry logic is counting a number of retry errors and checking thresholds of error detection and reporting a number of failed retries to the operating system. Thus a determination of the retryable errors occurred may favorably be performed.

[0024] Advantageously, the input/output status buffers may collect message states from the system nest and/or from the input/output bus controller, in particular a completion status from the system nest. By this way information about the completion status of different store instructions may be handled in an ordered and efficient manner.

[0025] Advantageously, the message states and/or the completion status may be numbered by an input/output status buffer index. The numbering enables the possibility of handling messages, and particularly completion states in an ordered and efficient way for further processing other store instructions.

[0026] According to a favourable embodiment of the inventive data processing system, the aggregation buffer may be communicatively coupled to the asynchronous core-nest interface via an asynchronous bus. Thus the aggregation buffer can handle data directly sent by the asynchronous core-nest interface consecutively until all data to be transferred to the external device are stored in the aggregation buffer. By this way the asynchronous transmit mechanism for data transfer from the asynchronous core-nest interface may be favourably supported.

[0027] According to a favourable embodiment of the inventive data processing system, the data may be transferred by the input/output store instruction through an asynchronous transmit mechanism with an early completion message in multiple data packets to the aggregation buffer, if the length of the source data exceeds eight bytes, else the data may be transferred in one data packet. The asynchronous transmit mechanism is favourable because the sending device is free for reuse at an earlier state.

[0028] According to a favourable embodiment of the inventive data processing system, the system firmware may comprise an asynchronous input/output driver code for handling the input/output store instruction. Thus an asynchronous transmit mechanism may be used for transferring data from the data processing unit to the external device.

[0029] According to a favourable embodiment of the inventive data processing system, the core may comprise an asynchronous setup code for handling memory requirements for status information of the asynchronous input/output driver code. This asynchronous setup code may further facilitate the asynchronous transmit mechanism through the aggregation buffer to the system nest and the input/output bus controller.

[0030] According to a favourable embodiment of the inventive data processing system, the asynchronous core-nest interface may comprise an asynchronous core-nest interface forwarding component for forwarding the data with local completion. This component may be implemented in hardware in the asynchronous core-nest interface. Thus a favourable asynchronous transmit mode for sending the data in data packets to the aggregation buffer may be supported.

[0031] According to a favourable embodiment of the inventive data processing system, the aggregation buffer may comprise an early completion logic for delivering a free for reuse message after sending a request. This enables an early continuation of the transmit process of the data via the aggregation buffer to the system nest and the input/output bus controller.

[0032] According to a favourable embodiment of the inventive data processing system, the system firmware may comprise an array management logic, which allocates/deallocates input/output status buffers in the input/output status array and/or initiates a start of a new store instruction. Thus idle status buffers may be attributed to further store instructions. An ordered processing of store instructions may be handled in an efficient and time saving way.

[0033] According to a favourable embodiment of the inventive data processing system, the asynchronous input/output driver code may store a copy of each input/output operation in the retry buffer and, if the asynchronous input/output driver code detects an error, control may be transferred to the analysis and retry logic. By this way a system recovery may favourably be possible, wherein only real errors may be handled.

[0034] According to a favourable embodiment of the inventive data processing system, the analysis and retry logic may determine a failing store instruction. Further it may determine an input/output function of that store instruction, initiate a retry for retryable errors, if below a threshold. If an error occurs in the asynchronous core-nest interface and the threshold is exceeded, then the analysis and retry logic may delete all outstanding requests for that input/output function; set the input/output function to an error state, and signal an asynchronous error to the operating system. If a fatal error occurs, then the analysis and retry logic may determine the source of the error. If the source is the system nest the analysis and retry logic may delete all, set all involved input/output functions to an error state, and signal an asynchronous error to the operating system. If the source is the input/output bus controller, the analysis and retry logic may delete all outstanding requests for input/output functions attached to that input/output bus controller; set all input/output functions to an error state, and signal an asynchronous error to the operating system. Thus a system recovery may favourably be possible, wherein only real errors may be handled.

[0035] According to a favourable embodiment of the inventive data processing system, a system message may comprise one of - a hierarchical physical target address, - sourcing an

SMT (simultaneous multithreading) thread or an aggregate buffer identifier, - a length of data, an input/output bus address, or - an input/output status buffer index. Thus an advantageous passing of relevant information through the data processing system can be guaranteed.

[0036] Further a method is proposed for handling an input/output store instruction to at least one external device of a data processing system, the data processing system comprising a system nest communicatively coupled to at least one input/output bus by an input/output bus controller. The data processing system further comprises at least a data processing unit comprising a core, a system firmware and an asynchronous core-nest interface. The data processing unit is communicatively coupled to the system nest via an aggregation buffer. The external device is communicatively coupled to the input/output bus. The asynchronous core-nest interface comprises an input/output status array with multiple input/output status buffers, as well as an array management and access logic. The system firmware further comprises a retry buffer and the core comprises an analysis and retry logic.

[0037] The method comprises: (i) an operating system running on the data processing system issuing the input/output store instruction specifying at least an input/output function with an offset through an address, data to be transferred and/or a pointer to data to be transferred, and a length of the data; (ii) the data processing unit being configured to identify the input/output function by the address specified in the input/output store instruction; (iii) the data processing unit being configured to verify if access to the input/output function is allowed on an address space and on a guest instance level, the guest running on the data processing system; (iv) the data processing unit being configured to complete the input/output store instruction before an execution of the input/output store instruction in the system nest is completed; (v) the system firmware being configured to notify the operating system through an interrupt, if during the asynchronous execution of the input/output store instruction an error is detected by the data processing unit, transmitting the data of the failed asynchronous execution; (vi) the analysis and retry logic detecting separately errors being ensured by the hardware that the store instruction has not been forwarded to an input/output bus yet; (vii) the retry buffer keeping store information for retries of executing the store instruction in system hardware/firmware; (viii) the analysis and retry logic analyzing errors and checking for retry possibility; and (ix) the analysis and retry logic triggering retries.

[0038] Favorably multiple outstanding asynchronous store instructions may thus be allowed at the same time to reduce cycles per instruction of repeated asynchronous store instructions. An ordering is defined between asynchronous store instructions and synchronous load/store instructions. Supporting multiple outstanding asynchronous store instructions is based on book keeping of multiple status messages and correlation of responses with status entries.

[0039] The method according to a further embodiment of the invention comprises instructions loading from and storing to an external device of the data processing system via an input/output bus. Asynchronous instructions complete before data has been stored to the external device while synchronous instructions complete after data has been stored to the external device. Within the embodiments described here, PCI will be used interchangeably for any other input/output technology, thus not restricting the embodiment of the invention to PCI.

[0040] Embodiments of the inventive method describe an input/output store instruction execution in a strictly ordered way as observable from above the architecture boundary while the actual execution may be out of order within the hardware of the data processing unit (CPU).

[0041] According to embodiments of the inventive method a PCI store instruction may be executed with an asynchronous execution of the PCIe store effect and an asynchronous status handling. Asynchronous reliable execution is based on reliable forwarding mechanisms in microarchitecture of the inventive data processing system.

[0042] An existing PCI store and store block instruction is usually synchronous up to the point where the PCI store data has been delivered to the PCIe interface and completion returned to a processing unit.

[0043] PCI standard only requires an asynchronous send command of PCI information, which is typically implemented through a store queue in the processor aggregating data with asynchronous send-out.

[0044] Advantageously, according to embodiments of the inventive method, an improvement concerning cycles per instruction may be achieved by replacing a synchronous PCI instruction by a reliable asynchronous send process of an input/output store instruction.

[0045] Alternatively or additionally of data to be transferred, the store instruction according to an embodiment of the invention may also specify a pointer to a main memory which should be used to fetch data from, instead of containing the data directly.

[0046] Guest instance level may also mean that a single guest or host may be running on the data processing system.

[0047] The address of the offset of the input/output function itself can be virtual, physical, logical address. Virtual and logical addresses typically get translated through a memory management unit (MMU) into a physical address, and the physical address then allows to identify which function and offset is meant.

[0048] Physical address in this context means "lowest address in the address translation hierarchy accessible from within a guest/operating system".

[0049] According to a favourable embodiment of the inventive method, the analysis and retry logic is counting a number of retry errors and checking thresholds of error detection and reporting a number of failed retries to the operating system. Thus a determination of the retryable errors occurred may favorably be performed.

[0050] Advantageously, the input/output status buffers may collect message states from the system nest and/or from the input/output bus controller, in particular a completion status from the system nest, wherein the message states and/or the completion status are numbered by an input/output status buffer index. By this way information about the completion status of different store instructions may be handled in an ordered and efficient manner. The numbering enables the possibility of handling messages, and particularly completion states in an ordered and efficient way for further processing other store instructions.

[0051] Advantageously, the system firmware may comprise an array management logic, allocating/deallocating input/output status buffers in the input/output status array and/or initiating a start of a new store instruction. Thus idle status buffers may be attributed to

further store instructions. An ordered processing of store instructions may be handled in an efficient and time saving way.

[0052] According to a favourable embodiment, the method may further comprise: (i) the operating system issuing the input/output store instruction; (ii) the system firmware (10) allocating a free input/output status buffer index; if there is no free input/output status buffer index available, then waiting for a free input/output status buffer index ; (iii) the system firmware injecting the store instruction into the asynchronous send engine; if blocked by another store instruction waiting until the store instruction has been completed; (iv) depending on the length of the data: if a length of the data exceeds eight bytes, the system firmware issuing repeatedly a system message to send a data packet to the aggregation buffer until all data of a store block have been forwarded to the aggregation buffer, while the system firmware waiting until the data have been sent by the system message; else the system firmware issuing a system message to send the data to the aggregation buffer; further independent of the length of the data, (v) the system firmware issuing a system message to the aggregation buffer to forward the data asynchronously as single nest message to the input/output bus controller, while waiting for the aggregation buffer to send a completion message; (vi) the aggregation buffer injecting the nest message into the system nest, wherein the aggregation buffer is free for reuse right after the send operation, signaling back to the system firmware; then the aggregation buffer sending a free for reuse message; (vii) the system nest forwarding the message to the target location; (viii) the input/output bus controller receiving the message and forwarding data in a data frame to the input/output bus; (ix) the input/output bus controller sending a completion message to the system nest; (x) the system nest forwarding the completion message to the originating aggregation buffer; (xi) the aggregation buffer forwarding completion to the asynchronous core-nest interface; (xii) the asynchronous core-nest interface storing the completion status in the input/output status buffer for the input/output status buffer index and signalling completion of operation to the system firmware; (xiii) the system firmware updating an input/output status buffer tracking by the input/output status buffer index; and (xiv) the system firmware signalling asynchronously defects to the operating system in case of an error.

[0053] Only step (ii) is dependent on the length of the data and is different for the length of the data exceeding eight bytes from for the length of the data not exceeding eight bytes.

[0054] According to the embodiment of the inventive method, the data are transmitted in slices to the aggregation buffer until all data of a store block are forwarded to the aggregation buffer, wherein the system firmware is waiting until the data has been sent by the asynchronous core-nest interface.

[0055] Thus if data is less than eight bytes the filling process of the aggregation buffer in slices with data packets may be skipped and the transmit process of the data to the external device can be completed in a single step.

[0056] According to a favourable embodiment of the inventive method, the data may be transferred by the input/output store instruction through an asynchronous transmit mechanism with an early completion message in multiple data packets to the aggregation buffer, if the length of the data exceeds eight bytes. The asynchronous transmit mechanism is favourable because the sending device is free for reuse at an earlier state.

[0057] According to a favourable embodiment of the inventive method, the system firmware may use an asynchronous input/output driver code for handling the input/output store instruction. Thus an asynchronous transmit mechanism may be used for transferring data from the data processing unit to the external device.

[0058] According to a favourable embodiment of the inventive method, the core may use an asynchronous setup code for handling memory requirements for status information of the asynchronous input/output driver code. This asynchronous setup code may further facilitate the asynchronous transmit mechanism through the aggregation buffer to the system nest and the input/output bus controller.

[0059] According to a favourable embodiment of the inventive method, the asynchronous core-nest interface may use an asynchronous core-nest interface forwarding component for forwarding the data with local completion. Thus a favourable asynchronous transmit mode for sending the data in data packets to the aggregation buffer may be supported.

[0060] According to a favourable embodiment of the inventive method, the aggregation

buffer may use an early completion logic for delivering a free for reuse message after sending a request. This enables an early continuation of the transmit process of the data via the aggregation buffer to the system nest and the input/output bus controller.

[0061] Advantageously, the input/output status buffers may collect returned states from the system nest and/or from the input/output bus controller, in particular a completion message from the system nest. These input/output status buffers may collect the returned states acting as an asynchronous system message buffer supporting the asynchronous transmit process.

[0062] According to a favourable embodiment of the inventive method, a system message may comprise one of - a hierarchical physical target address, - sourcing an SMT thread or an aggregate buffer identifier, - a length of data, - an input/output bus address, or - an input/output status buffer index. Thus an advantageous passing of relevant information through the data processing system can be guaranteed.

[0063] According to a favourable embodiment of the inventive method, the asynchronous input/output driver code may store a copy of each input/output operation in the retry buffer and, if the asynchronous input/output driver code detects an error, control may be transferred to the analysis and retry logic. By this way a system recovery may favourably be possible, wherein only real errors may be handled.

[0064] According to a favourable embodiment of the inventive method, the analysis and retry logic may determine a failing store instruction. Further it may determine an input/output function of that store instruction, initiate a retry for retryable errors, if below a threshold. If an error occurs in the asynchronous core-nest interface and the threshold is exceeded, then the analysis and retry logic may delete all outstanding requests for that input/output function; set the input/output function to an error state, and signal an asynchronous error to the operating system. If a fatal error occurs, then the analysis and retry logic may determine the source of the error. If the source is the system nest the analysis and retry logic may delete all, set all involved input/output functions to an error state, and signal an asynchronous error to the operating system. If the source is the input/output bus controller, the analysis and retry logic may delete all outstanding requests for input/output functions attached to that input/output bus controller; set all input/output

functions to an error state, and signal an asynchronous error to the operating system. Thus a system recovery may favourably be possible, wherein only real errors may be handled.

[0065] Further, a favorable computer program product is proposed for handling an input/output store instruction to at least one external device of a data processing system, the data processing system comprising a system nest communicatively coupled to at least one input/output bus by an input/output bus controller. The data processing system further comprises at least a data processing unit comprising a core, a system firmware and an asynchronous core-nest interface. The data processing unit is communicatively coupled to the system nest via an aggregation buffer. The external device is communicatively coupled to the input/output bus. The asynchronous core-nest interface comprises an input/output status array with multiple input/output status buffers, as well as an array management and access logic. The system firmware further comprises a retry buffer and the core comprises an analysis and retry logic.

[0066] The computer program product comprises a computer readable storage medium having program instructions embodied therewith, the program instructions executable by the computer system to cause the computer system to perform a method comprising: (i) an operating system running on the data processing system issuing the input/output store instruction specifying at least an input/output function with an offset through an address, data to be transferred and/or a pointer to data to be transferred, and a length of the data; (ii) the data processing unit being configured to identify the input/output function by the address specified in the input/output store instruction; (iii) the data processing unit being configured to verify if access to the input/output function is allowed on an address space and on a guest instance level, the guest running on the data processing system; (iv) the data processing unit being configured to complete the input/output store instruction before an execution of the input/output store instruction in the system nest is completed; (v) the system firmware being configured to notify the operating system through an interrupt, if during the asynchronous execution of the input/output store instruction an error is detected by the data processing unit, transmitting the data of the failed asynchronous execution; (vi) the analysis and retry logic detecting separately errors being ensured by the hardware that the store instruction has not been forwarded to an input/output bus yet; (vii) the retry buffer keeping store information for retries of executing the store instruction in system hardware/firmware; (viii) the analysis and retry logic analyzing errors and checking for retry possibility; and (ix) the

analysis and retry logic triggering retries.

[0067] Further, a data processing system for execution of a data processing program is proposed, comprising computer readable program instructions for performing the method described above.

BRIEF DESCRIPTION OF THE SEVERAL VIEWS OF THE DRAWINGS

[0068] The present invention together with the above-mentioned and other objects and advantages may best be understood from the following detailed description of the embodiments, but not restricted to the embodiments.

[0069] Figure 1 depicts a block diagram of a data processing system for handling an input/output store instruction to an external device according to an embodiment of the invention.

[0070] Figure 2 depicts a message sequence chart of a method for handling an input/output store instruction to an external device according to an embodiment of the invention.

[0071] Figure 3 depicts a first part of a flow chart for handling an input/output store instruction to an external device according to an embodiment of the invention.

[0072] Figure 4 depicts a second part of a flow chart for handling an input/output store instruction to an external device according to an embodiment of the invention.

[0073] Figure 5 depicts an example embodiment of a data processing system for executing a method according to the invention.

DETAILED DESCRIPTION

[0074] In the drawings, like elements are referred to with equal reference numerals. The

drawings are merely schematic representations, not intended to portray specific parameters of the invention. Moreover, the drawings are intended to depict only typical embodiments of the invention and therefore should not be considered as limiting the scope of the invention.

[0075] The illustrative embodiments described herein provide a data processing system and a method for handling an input/output store instruction, comprising a system nest communicatively coupled to at least one input/output bus by an input/output bus controller. The data processing system further comprises at least a data processing unit comprising a core, a system firmware and an asynchronous core-nest interface. The data processing unit is communicatively coupled to the system nest via an aggregation buffer. The system nest is configured to asynchronously load from and/or store data to an external device which is communicatively coupled to the input/output bus. The asynchronous core-nest interface comprises an input/output status array with multiple input/output status buffers, as well as an array management and access logic. The system firmware further comprises a retry buffer and the core comprises an analysis and retry logic.

[0076] The illustrative embodiments may be used for the method comprising: (i) an operating system running on the data processing system issues the input/output store instruction specifying at least an input/output function with an offset through an address, data to be transferred and/or a pointer to data to be transferred, and a length of the data; (ii) the data processing unit is configured to identify the input/output function by the address specified in the input/output store instruction; (iii) the data processing unit is configured to verify if access to the input/output function is allowed on an address space and on a guest instance level, the guest running on the data processing system; (iv) the data processing unit is configured to complete the input/output store instruction before an execution of the input/output store instruction in the system nest is completed; (v) the system firmware is configured to notify the operating system through an interrupt, if during the asynchronous execution of the input/output store instruction an error is detected by the data processing unit, transmitting the data of the failed asynchronous execution; (vi) the analysis and retry logic detecting separately errors being ensured by the hardware that the store instruction has not been forwarded to an input/output bus yet; (vii) the retry buffer keeping store information for retries of executing the store instruction in system hardware/firmware; (viii) the analysis and retry logic analyzing errors and checking for retry possibility; and (ix) the analysis and retry logic triggering retries.

[0077] Alternatively or additionally of data to be transferred, the store instruction according to an embodiment of the invention may also specify a pointer to main memory which should be used to fetch data from, instead of containing the data directly.

[0078] Guest instance level may also mean that a single guest or host may be running on the data processing system.

[0079] The address of the offset of the input/output function itself can be virtual, physical, logical address. Virtual and logical addresses typically get translated through a memory management unit (MMU) into a physical address, and the physical address then allows to identify which function and offset is meant.

[0080] Physical address in this context means "lowest address in the address translation hierarchy accessible from within a guest/operating system".

[0081] Figure 1 depicts a block diagram of a data processing system 210 for handling an input/output store instruction 30 to at least one external device 214 according to an embodiment of the invention. The data processing system 210 comprises a system nest 18 communicatively coupled to an input/output bus 22 by an input/output bus controller 20, a data processing unit 216 comprising a core 12, a system firmware 10 and an asynchronous core-nest interface 14. The input/output bus controller 20 may also be coupled via multiple input/output busses 22 to multiple external devices 214.

[0082] The data processing unit 216 is communicatively coupled to the system nest 18 via an aggregation buffer 16. The system nest 18 is configured to asynchronously load from and/or store data to the external device 214 which is communicatively coupled to the input/output bus 22 via a buffer-input/output bus controller interface 28 as part of the system nest 18 and the input/output bus controller 20. The asynchronous core-nest interface 14 comprises an input/output status array 44 with multiple input/output status buffers 24 and an array management and access logic 46. the system firmware 10 comprises a retry buffer 52 and the core 12 comprises an analysis and retry logic 54. The analysis and retry logic 54 is counting a number of retry errors and checking thresholds of error detection and reporting a number of failed retries to the operating system.

[0083] The aggregation buffer 16 is communicatively coupled to the asynchronous core-nest interface 14. The system firmware 10 comprises an asynchronous input/output driver code 32 for handling the input/output store instruction 30. The core 12 comprises an asynchronous setup code 34 for handling memory requirements for status information of the asynchronous input/output driver code 32. According to recovery semantics for multiple external devices 214, the asynchronous input/output driver code 32 stores a copy of each input/output operation in the retry buffer 52 and, if the asynchronous input/output driver code 32 detects an error, control is transferred to the analysis and retry logic 54.

[0084] The array management and access logic 46 provides an interface to the core 12 to query a status of all input/output status buffers 24. Further it provides an interface to the core 12 to reset a status of selected input/output status buffers 24.

[0085] The analysis and retry logic 54 determines a failing store instruction 30. Further it determines an input/output function of that store instruction 30, initiates a retry for retryable errors, if below a threshold. Further, if an error occurs in the asynchronous core-nest interface 14 and the threshold is exceeded, the analysis and retry logic 54 deletes all outstanding requests for that input/output function; sets the input/output function to an error state, and signals an asynchronous error to the operating system. If a fatal error occurs, the analysis and retry logic 54 determines the source of the error. If the source is the system nest 18, the analysis and retry logic 54 deletes all, sets all involved input/output functions to an error state, and signals an asynchronous error to the operating system. If the source is the input/output bus controller 20, the analysis and retry logic 54 deletwa all outstanding requests for input/output functions attached to that input/output bus controller 20; sets all input/output functions to an error state, and signals an asynchronous error to the operating system.

[0086] The system firmware 10 comprises an array management logic 42, which allocates/deallocates input/output status buffers 24 in the input/output status array 44 and/or initiates a start of a new store instruction 30.

[0087] The asynchronous core-nest interface 14 comprises an asynchronous core-nest interface forwarding component 36 for forwarding the data with local completion. The aggregation buffer 16 comprises an early completion logic 26 for delivering a free for reuse

message after sending a request. The aggregation buffer 16 is coupled to the asynchronous core-nest interface 14 via an asynchronous bus 38. The asynchronous core-nest interface 14 comprises an input/output status array 44 with multiple input/output status buffers 24, as well as an array management and access logic 46. The input/output status buffers 24 collect returned states from the system nest 18 and/or from the input/output bus controller 20, in particular a completion message from the system nest 18. The input/output status buffers 24 are integrated directly in the asynchronous core-nest interface 14. A message 48 with an identification of an array entry, e.g. a completion message to one of the input/output status buffers 24 may be received by the system nest 18.

[0088] According to an embodiment of the inventive method, an operating system running on the data processing system 210 issues the input/output store instruction 30 specifying at least an input/output function with an offset through an address, data to be transferred and/or a pointer to data to be transferred, and a length of the data. The data processing unit 216 is hereby configured to identify the input/output function by the address specified in the input/output store instruction 30. The data processing unit 216 is configured to verify if access to the input/output function is allowed on an address space and on a guest instance level, the guest running on the data processing system 210. The data processing unit 216 is configured to complete the input/output store instruction 30 before an execution of the input/output store instruction 30 in the system nest 18 is completed. The system firmware 10 is configured to notify the operating system through an interrupt, if during the asynchronous execution of the input/output store instruction 30 an error is detected by the data processing unit 216, transmitting the data of the failed asynchronous execution.

[0089] The array management and access logic 46 collects a completion of the store instruction 30 and updates the input/output status buffers 24 based on received completion messages.

[0090] The input/output status buffers 24 collect message states from the system nest 18 and/or from the input/output bus controller 20, in particular a completion status from the system nest 18. The message states and/or the completion status may be favorably numbered by an input/output status buffer index.

[0091] The input/output store instruction 30 is located in the data processing system

210 on the side of the user interface 40 across the architecture boundary which separates the system hardware/firmware 50 from the user side 40.

[0092] Thus the data are transferred by the input/output store instruction 30 through an asynchronous transmit mechanism with an early completion message in multiple data packets to the aggregation buffer 16, if the length of the source data exceeds eight bytes, else the data are transferred in one data packet.

[0093] A system message according to an embodiment of the inventive data processing system comprises one of a hierarchical physical target address, sourcing an SMT thread or an aggregate buffer identifier, a length of data, an input/output bus address, or an input/output status buffer index.

[0094] The queueing and ordering semantics for handling store instructions 30 to multiple external devices 214 may advantageously performed as described in the following. For an individual SMT thread versus input/output function relation, all legacy input/output load/store operations may be ordered in respect to a single thread of the processor unit 216. The new input/output store instructions are completely unordered amongst each other. New input/output store instructions are ordered against legacy input/output instructions. All input/output instructions for different input/output functions are not ordered against each other.

[0095] Figure 2 depicts a message sequence chart of the method for handling an input/output store instruction 30 to an external device 214 according to an embodiment of the invention.

[0096] As shown in Figure 2 the method starts with the operating system issuing the input/output store instruction 30. In step S101, the system firmware 10 allocates a free input/output status buffer index. If there is no free input/output status buffer index available, the system firmware 10 waits. In step S103, the system firmware 10 checks, if the store instruction can be injected into an asynchronous send engine. If this is possible, the process continues. If this is not possible, the store instruction is delayed until the store instructions causing the delay have been completed.

[0097] Next, as is indicated by the steps S100 and S104, the system firmware 10 issues repeatedly, if a length of the data exceeds eight bytes, a system message to send a data packet to the aggregation buffer 16 until all data of a store block have been forwarded to the aggregation buffer 16, while the system firmware 10 is waiting until the data have been sent by the system message. In steps S102 and S106 a local completion message is sent back to the system firmware 10.

[0098] Then in step S108, the system firmware 10 issues a system message to the aggregation buffer 16 to forward the data asynchronously as single nest message to the input/output bus controller 20, while waiting for the aggregation buffer 16 to send a completion message.

[0099] Next in step S110, the aggregation buffer 16 injects the nest message into the system nest 18, wherein the aggregation buffer 16 is free for reuse right after the send operation, signaling back to the system firmware 10. Then the aggregation buffer 16 sends a free for reuse message.

[00100] In step S112, the system nest 18 forwards the message to the target location, followed by step S114, the input/output bus controller 20 receiving the message and forwarding data in a data frame to the input/output bus, followed by the input/output bus controller 20 sending a completion message to the system nest 18 in step S116.

[00101] Next in step S118, the system nest 18 forwards the completion message to the originating aggregation buffer 16, followed by the aggregation buffer 16 forwarding completion to the asynchronous core-nest interface 14 in step S120. Then in step S122 the asynchronous core-nest interface 14 stores the status in the input/output buffer 24 for the respective input/output status buffer index and signals completion of operation to the system firmware 10. Finally in step S123, the system firmware 10 updates the input/output status buffer 24 tracking by the input/output status buffer index. The input/output status buffer 24 is now free again.

[00102] In case of an error occurring during transfer of data, the system firmware 10 signals asynchronously defects to the operating system.

[00103] In case, the data to be transferred are less than eight bytes, the repeatedly filling of the aggregation buffer 16 is skipped.

[00104] Figure 3 depicts a first part of a flow chart for handling an input/output store instruction 30 to an external device 214 according to an embodiment of the invention, whereas Figure 4 depict a second part of the flow chart.

[00105] If an error interrupt occurs in step s200, the interrupt is received as a message in step S202 by the system firmware 10. Next a status message is fetched in step S204 from the input/output status array comprising the input/output status buffers 24. The affected input/output function is determined in step S206, checking if it is a retryable error in step S208. If this is the case, it is checked in step S210 if a threshold is reached. If this is the case the process flow is continued at the continuation point A according to the flow chart described in Figure 4. If not, the retry counter is increased in step S212, followed by clearing the error in the status array entry in step S214. Then a retry is triggered in step S218. Next it is checked in step S220 if the error is found in the status array 44. If this is the case, the process loops back to step S206 determining the affected function. If not the process is ended.

[00106] If in step S208 no retryable error is determined, it is checked in step S222 if this is a fatal error. Then it is checked if functions are affected, which are attached to the input/output bus 22 with an outstanding store function (step 226) or if all functions are affected with an outstanding store function (step S228). The affected functions are set into an error state in step S230, followed by signaling asynchronously the error to the operating system in step S232.

[00107] If in step S222 no fatal error is found, the affected functions are set into an error state in step S230, followed by signaling asynchronously the error to the operating system in step S232 and in parallel the process flow continues at the continuation point A with the second part of the flow chart, described in Figure 4.

[00108] Information about the determined affected input/output functions is stored in the memory 228 in a store data shadow copy 62. Information about the affected functions which are set into error state is stored in a store access table 64 of the memory 228.

[00109] The second part of the flow chart, beginning with connection point A, is depicted in Figure 4. First in step S304 it is checked if the more than 8 bytes are to be transferred. If this is the case the core-nest interface fills the aggregation buffer with an up to 16 bytes message in step S306. The system firmware is waiting, step S308, until a message of local completion is sent in step S310, returning to step S304. If there are less than 8 bytes left in the check of step S304, the flow continues in step S312 with the core-nest interface sending an asynchronous input/output message, followed by waiting in step S314 for a buffer response in step S316. Then in step S318 a finish store block instruction is executed and the flow ends in step S320 with an ending in the system firmware.

[00110] In step S328 the asynchronous core-nest interface logic starts an outbound process loop, followed by receiving an aggregation buffer completion message in step S322 and a forward data message to the aggregation buffer in step S324, followed by a send completion message back to the system firmware in step S326. In step S330 an asynchronous input/output send message is received followed by a forward of the input/output send message to the aggregation buffer.

[00111] In step S338 the aggregation buffer logic starts an outbound process loop followed by a receive data in step S334 and aggregating data in the aggregation buffer in step S336. The aggregation buffer is also receiving an input/output send message in step S340, followed by forwarding data from the aggregation buffer with an input/output send message in step S242. Next in step S344 a response message from the aggregation buffer is sent via the core-nest interface to the system firmware.

[00112] Referring now to Figure 5, a schematic of an example of a data processing system 210 is shown. Data processing system 210 is only one example of a suitable data processing system and is not intended to suggest any limitation as to the scope of use or functionality of embodiments of the invention described herein. Regardless, data processing system 210 is capable of being implemented and/or performing any of the functionality set forth herein above.

[00113] In data processing system 210 there is a computer system/server 212, which is operational with numerous other general-purpose or special-purpose computing system

environments or configurations. Examples of well-known computing systems, environments, and/or configurations that may be suitable for use with computer system/server 212 include, but are not limited to, personal computer systems, server computer systems, thin clients, thick clients, handheld or laptop devices, multiprocessor systems, microprocessor-based systems, set top boxes, programmable consumer electronics, network PCs, minicomputer systems, mainframe computer systems, and distributed cloud computing environments that include any of the above systems or devices, and the like.

[00114] Computer system/server 212 may be described in the general context of computer system executable instructions, such as program modules, being executed by a computer system. Generally, program modules may include routines, programs, objects, components, logic, data structures, and so on that perform particular tasks or implement particular abstract data types. Computer system/server 212 may be practiced in distributed cloud computing environments where tasks are performed by remote processing devices that are linked through a communications network. In a distributed cloud computing environment, program modules may be located in both local and remote computer system storage media including memory storage devices.

[00115] As shown in Fig. 5, computer system/server 212 in data processing system 210 is shown in the form of a general-purpose computing device. The components of computer system/server 212 may include, but are not limited to, one or more processors or processing units 216, a system memory 228, and a bus 218 that couples various system components including system memory 228 to processor 216.

[00116] Bus 218 represents one or more of any of several types of bus structures, including a memory bus or memory controller, a peripheral bus, an accelerated graphics port, and a processor or local bus using any of a variety of bus architectures. By way of example, and not limitation, such architectures include Industry Standard Architecture (ISA) bus, Micro Channel Architecture (MCA) bus, Enhanced ISA (EISA) bus, Video Electronics Standards Association (VESA) local bus, and Peripheral Component Interconnect (PCI) bus.

[00117] Computer system/server 212 typically includes a variety of computer system readable media. Such media may be any available media that is accessible by computer system/server 212, and it includes both volatile and non-volatile media, removable and non-

removable media.

[00118] System memory 228 can include computer system readable media in the form of volatile memory, such as random access memory (RAM) 230 and/or cache memory 232. Computer system/server 212 may further include other removable/non-removable, volatile/non-volatile computer system storage media. By way of example only, storage system 234 can be provided for reading from and writing to a non-removable, non-volatile magnetic media (not shown and typically called a "hard drive"). Although not shown, a magnetic disk drive for reading from and writing to a removable, non-volatile magnetic disk (e.g., a "floppy disk"), and an optical disk drive for reading from or writing to a removable, non-volatile optical disk such as a CD-ROM, DVD-ROM or other optical media can be provided. In such instances, each can be connected to bus 218 by one or more data media interfaces. As will be further depicted and described below, memory 228 may include at least one program product having a set (e.g., at least one) of program modules that are configured to carry out the functions of embodiments of the invention.

[00119] Program/utility 240, having a set (at least one) of program modules 242, may be stored in memory 228 by way of example, and not limitation, as well as an operating system, one or more application programs, other program modules, and program data. Each of the operating system, one or more application programs, other program modules, and program data or some combination thereof, may include an implementation of a networking environment. Program modules 242 generally carry out the functions and/or methodologies of embodiments of the invention as described herein.

[00120] Computer system/server 212 may also communicate with one or more external devices 214 such as a keyboard, a pointing device, a display 224, etc.; one or more devices that enable a user to interact with computer system/server 212; and/or any devices (e.g., network card, modem, etc.) that enable computer system/server 212 to communicate with one or more other computing devices. Such communication can occur via Input/Output (I/O) interfaces 222. Still yet, computer system/server 212 can communicate with one or more networks such as a local area network (LAN), a general wide area network (WAN), and/or a public network (e.g., the Internet) via network adapter 220. As depicted, network adapter 220 communicates with the other components of computer system/server 212 via bus 218. It should be understood that although not shown, other hardware and/or software

components could be used in conjunction with computer system/server 212. Examples, include, but are not limited to: microcode, device drivers, redundant processing units, external disk drive arrays, RAID systems, tape drives, and data archival storage systems, etc.

[00121] The present invention may be a system, a method, and/or a computer program product. The computer program product may include a computer readable storage medium (or media) having computer readable program instructions thereon for causing a processor to carry out aspects of the present invention.

[00122] The computer readable storage medium can be a tangible device that can retain and store instructions for use by an instruction execution device. The computer readable storage medium may be, for example, but is not limited to, an electronic storage device, a magnetic storage device, an optical storage device, an electromagnetic storage device, a semiconductor storage device, or any suitable combination of the foregoing. A non-exhaustive list of more specific examples of the computer readable storage medium includes the following: a portable computer diskette, a hard disk, a random access memory (RAM), a read-only memory (ROM), an erasable programmable read-only memory (EPROM or Flash memory), a static random access memory (SRAM), a portable compact disc read-only memory (CD-ROM), a digital versatile disk (DVD), a memory stick, a floppy disk, a mechanically encoded device such as punch-cards or raised structures in a groove having instructions recorded thereon, and any suitable combination of the foregoing. A computer readable storage medium, as used herein, is not to be construed as being transitory signals per se, such as radio waves or other freely propagating electromagnetic waves, electromagnetic waves propagating through a waveguide or other transmission media (e.g., light pulses passing through a fiber-optic cable), or electrical signals transmitted through a wire.

[00123] Computer readable program instructions described herein can be downloaded to respective computing/processing devices from a computer readable storage medium or to an external computer or external storage device via a network, for example, the Internet, a local area network, a wide area network and/or a wireless network. The network may comprise copper transmission cables, optical transmission fibers, wireless transmission, routers, firewalls, switches, gateway computers and/or edge servers. A network adapter card or

network interface in each computing/processing device receives computer readable program instructions from the network and forwards the computer readable program instructions for storage in a computer readable storage medium within the respective computing/processing device.

[00124] Computer readable program instructions for carrying out operations of the present invention may be assembler instructions, instruction-set-architecture (ISA) instructions, machine instructions, machine dependent instructions, microcode, firmware instructions, state-setting data, or either source code or object code written in any combination of one or more programming languages, including an object oriented programming language such as Smalltalk, C++ or the like, and conventional procedural programming languages, such as the "C" programming language or similar programming languages. The computer readable program instructions may execute entirely on the user's computer, partly on the user's computer, as a stand-alone software package, partly on the user's computer and partly on a remote computer or entirely on the remote computer or server. In the latter scenario, the remote computer may be connected to the user's computer through any type of network, including a local area network (LAN) or a wide area network (WAN), or the connection may be made to an external computer (for example, through the Internet using an Internet Service Provider). In some embodiments, electronic circuitry including, for example, programmable logic circuitry, field-programmable gate arrays (FPGA), or programmable logic arrays (PLA) may execute the computer readable program instructions by utilizing state information of the computer readable program instructions to personalize the electronic circuitry, in order to perform aspects of the present invention.

[00125] Aspects of the present invention are described herein with reference to flowchart illustrations and/or block diagrams of methods, apparatus (systems), and computer program products according to embodiments of the invention. It will be understood that each block of the flowchart illustrations and/or block diagrams, and combinations of blocks in the flowchart illustrations and/or block diagrams, can be implemented by computer readable program instructions.

[00126] These computer readable program instructions may be provided to a processor of a general-purpose computer, special-purpose computer, or other programmable data processing apparatus to produce a machine, such that the instructions, which execute via the

processor of the computer or other programmable data processing apparatus, create means for implementing the functions/acts specified in the flowchart and/or block diagram block or blocks. These computer readable program instructions may also be stored in a computer readable storage medium that can direct a computer, a programmable data processing apparatus, and/or other devices to function in a particular manner, such that the computer readable storage medium having instructions stored therein comprises an article of manufacture including instructions which implement aspects of the function/act specified in the flowchart and/or block diagram block or blocks.

[00127] The computer readable program instructions may also be loaded onto a computer, other programmable data processing apparatus, or other device to cause a series of operational steps to be performed on the computer, other programmable apparatus or other device to produce a computer implemented process, such that the instructions which execute on the computer, other programmable apparatus, or other device implement the functions/acts specified in the flowchart and/or block diagram block or blocks.

[00128] The flowchart and block diagrams in the Figures illustrate the architecture, functionality, and operation of possible implementations of systems, methods, and computer program products according to various embodiments of the present invention. In this regard, each block in the flowchart or block diagrams may represent a module, segment, or portion of instructions, which comprises one or more executable instructions for implementing the specified logical function(s). In some alternative implementations, the functions noted in the block may occur out of the order noted in the figures. For example, two blocks shown in succession may, in fact, be executed substantially concurrently, or the blocks may sometimes be executed in the reverse order, depending upon the functionality involved. It will also be noted that each block of the block diagrams and/or flowchart illustration, and combinations of blocks in the block diagrams and/or flowchart illustration, can be implemented by special-purpose hardware-based systems that perform the specified functions or acts or carry out combinations of special-purpose hardware and computer instructions.

[00129] The descriptions of the various embodiments of the present invention have been presented for purposes of illustration, but are not intended to be exhaustive or limited to the embodiments disclosed. Many modifications and variations will be apparent to those of

ordinary skill in the art without departing from the scope and spirit of the described embodiments. The terminology used herein was chosen to best explain the principles of the embodiments, the practical application or technical improvement over technologies found in the marketplace, or to enable others of ordinary skill in the art to understand the embodiments disclosed herein.

REFERENCE NUMERALS

10	system FW
12	core
14	async core-nest IF
16	aggregation buffer
18	system nest
20	I/O bus controller
22	I/O bus
24	I/O status buffer
26	early completion logic
28	buffer-I/O bus controller IF
30	I/O store instruction
32	async I/O driver code
34	I/O setup code
36	async forwarding
38	async bus
40	user IF
42	array management logic
44	I/O status array
46	array management & access logic
48	message with array entry identification
50	system HW/FW
52	retry buffer
54	analysis & retry logic
60	
62	store data shadow copy
64	store access table
210	data processing system
212	computer system/server
214	external devices
216	CPU / data processing unit
218	IO Bus
220	network adapter
222	IO interfaces
224	display
228	memory
230	RAM
232	cache
234	storage system
240	program/utility
242	program modules

CLAIMS

1. A data processing system (210) for handling an input/output store instruction (30), comprising a system nest (18) communicatively coupled to at least one input/output bus (22) by an input/output bus controller (20),
further comprising at least
a data processing unit (216) comprising a core (12),
a system firmware (10) and an asynchronous core-nest interface (14),
wherein the data processing unit (216) is communicatively coupled to the system nest (18) via an aggregation buffer (16),
wherein the system nest (18) is configured to asynchronously load from and/or store data to at least one external device (214) which is communicatively coupled to the input/output bus (22),
wherein the asynchronous core-nest interface (14) comprises an input/output status array (44) with multiple input/output status buffers (24) and an array management and access logic (46),
wherein the system firmware (10) comprises a retry buffer (52) and the core (12) comprises an analysis and retry logic (54),
and wherein
(i) an operating system running on the data processing system (210) issuing the input/output store instruction (30) specifying at least an input/output function with an offset through an address, data to be transferred and/or a pointer to data to be transferred, and a length of the data;
(ii) the data processing unit (216) being configured to identify the input/output function by the address specified in the input/output store instruction (30);
(iii) the data processing unit (216) being configured to verify if access to the input/output function is allowed on an address space and on a guest instance level, the guest running on the data processing system (210);
(iv) the data processing unit (216) being configured to complete the input/output store instruction (30) before an execution of the input/output store instruction (30) in the system nest (18) is completed;
(v) the system firmware (10) being configured to notify the operating system through an interrupt, if during the asynchronous execution of the

input/output store instruction (30) an error is detected by the data processing unit (216);

(vi) the analysis and retry logic (54) detecting separately errors being ensured by the hardware that the store instruction (30) has not been forwarded to an input/output bus (22) yet;

(vii) the retry buffer (52) keeping store information for retries of executing the store instruction (30) in system hardware/firmware (50);

(viii) the analysis and retry logic (54) analyzing errors and checking for retry possibility;

(ix) the analysis and retry logic (54) triggering retries.

2. The data processing system according to claim 1, wherein the analysis and retry logic (54) is counting a number of retry errors and checking thresholds of error detection and reporting a number of failed retries to the operating system.

3. The data processing system according to claim 1 or 2, the aggregation buffer (16) being communicatively coupled to the asynchronous core-nest interface (14) via an asynchronous bus (38).

4. The data processing system according to any one of the preceding claims, wherein the data are transferred by the input/output store instruction (30) through an asynchronous transmit mechanism with an early completion message in multiple data packets to the aggregation buffer (16), if the length of the data exceeds eight bytes, else the data are transferred in one data packet.

5. The data processing system according to any one of the preceding claims, the system firmware (10) comprising an asynchronous input/output driver code (32) for handling the input/output store instruction (30).

6. The data processing system according to claim 5, the core (12) comprising an asynchronous setup code (34) for handling memory requirements for status information of the asynchronous input/output driver code (32).

7. The data processing system according to any one of the preceding claims, the

asynchronous core-nest interface (14) comprising an asynchronous core-nest interface forwarding component (36) for forwarding the data with local completion.

8. The data processing system according to any one of the preceding claims, the aggregation buffer (16) comprising an early completion logic (26) for delivering a free for reuse message after sending a request.

9. The data processing system according to any one of the preceding claims, wherein the system firmware (10) comprises an array management logic (42), which allocates/deallocates input/output status buffers (24) in the input/output status array (44) and/or initiating a start of a new store instruction (30).

10. The data processing system according to any one of the preceding claims, wherein the asynchronous input/output driver code (32) stores a copy of each input/output operation in the retry buffer (52) and, if the asynchronous input/output driver code (32) detects an error, control is transferred to the analysis and retry logic (54).

11. The data processing system according to any one of the preceding claims, wherein the analysis and retry logic (54)

- determines a failing store instruction (30),
- determines an input/output function of that store instruction (30),
- initiates a retry for retryable errors, if below a threshold,
- if an error occurs in the asynchronous core-nest interface (14) and the threshold is exceeded, deletes all outstanding requests for that input/output function; sets the input/output function to an error state, and signals an asynchronous error to the operating system,
- if a fatal error occurs, determine the source of the error; if the source is the system nest (18) delete all, sets all involved input/output functions to an error state, and signals an asynchronous error to the operating system; if the source is the input/output bus controller (20), delete all outstanding requests for input/output functions attached to that input/output bus controller (20); sets all input/output functions to an error state, and signals an asynchronous error to the operating system.

12. The data processing system according to any one of the preceding claims, a system message comprising one of

- a hierarchical physical target address,
- sourcing an SMT thread or an aggregate buffer identifier,
- a length of data,
- an input/output bus address,
- an input/output status buffer index.

13. A method for handling an input/output store instruction (30) to at least one external device (214) of a data processing system (210), the data processing system (210) comprising
a system nest (10) communicatively coupled to at least one input/output bus (22) by an input/output bus controller (14),
and further comprising at least a data processing unit (216) comprising a core (12), a system firmware (10) and an asynchronous core-nest interface (14),
wherein the data processing unit (216) is communicatively coupled to the system nest (18) via an aggregation buffer (16),
wherein the external device (214) is communicatively coupled to the input/output bus (22),
wherein the asynchronous core-nest interface (14) comprises an input/output status array (44) with multiple input/output status buffers (24) and an array management and access logic (46),
wherein the system firmware (10) comprises a retry buffer (52) and the core (12) comprises an analysis and retry logic (54),

the method comprising

(i) an operating system running on the data processing system (210) issuing the input/output store instruction (30) specifying at least an input/output function with an offset through an address, data to be transferred and/or a pointer to data to be transferred, and a length of the data;

(ii) the data processing unit (216) being configured to identify the input/output function by the address specified in the input/output store instruction (30);

(iii) the data processing unit (216) being configured to verify if access to the input/output function is allowed on an address space and on a guest instance level, the guest running on the data processing system (210);

(iv) the data processing unit (216) being configured to complete the input/output store instruction (30) before an execution of the input/output store instruction (30) in the system nest (18) is completed;

(v) the system firmware (10) being configured to notify the operating system through an interrupt, if during the asynchronous execution of the input/output store instruction (30) an error is detected by the data processing unit (216);

(vi) the analysis and retry logic (54) detecting separately errors being ensured by the hardware that the store instruction (30) has not been forwarded to an input/output bus (22) yet;

(vii) the retry buffer (52) keeping store information for retries of executing the store instruction (30) in system hardware/firmware (50);

(viii) the analysis and retry logic (54) analyzing errors and checking for retry possibility;

(ix) the analysis and retry logic (54) triggering retries.

14. The method to claim 13, wherein the analysis and retry logic (54) is counting a number of retry errors and checking thresholds of error detection and reporting a number of failed retries to the operating system.

15. The method according to 13 or 14, further comprising

(i) the operating system issuing the input/output store instruction (30);

(ii) the system firmware (10) allocating a free input/output status buffer index; if there is no free input/output status buffer index available, then waiting for a free input/output status buffer index;

(iii) the system firmware (10) injecting the store instruction (30) into the asynchronous send engine; if blocked by another store instruction waiting until the store instruction has been completed;

(iv) depending on the length of the data: if a length of the data exceeds eight bytes, the system firmware (10) issuing repeatedly a system message to send a data packet to the aggregation buffer (16) until all data of a store block have been forwarded to the aggregation buffer (16), while the system firmware (10) waiting until the data have been sent by the system message; else

the system firmware (10) issuing a system message to send the data to the aggregation buffer (16);

(v) the system firmware (10) issuing a system message to the aggregation buffer (16) to forward the data asynchronously as single nest message to the input/output bus controller (20), while waiting for the aggregation buffer (16) to send a completion message;

(vi) the aggregation buffer (16) injecting the nest message into the system nest (18), wherein the aggregation buffer (16) is free for reuse right after the send operation, signaling back to the system firmware (10); then the aggregation buffer (16) sending a free for reuse message;

(vii) the system nest (18) forwarding the message to the target location;

(viii) the input/output bus controller (20) receiving the message and forwarding data in a data frame to the input/output bus;

(ix) the input/output bus controller (20) sending a completion message to the system nest (18);

(x) the system nest (18) forwarding the completion message to the originating aggregation buffer (16);

(xi) the aggregation buffer (16) forwarding completion to the asynchronous core-nest interface (14);

(xii) the asynchronous core-nest interface (14) storing the completion status in the input/output status buffer (24) for the input/output status buffer index and signaling completion of operation to the system firmware (10);

(xiii) the system firmware (10) updating an input/output status buffer tracking by the input/output status buffer index;

(xiv) the system firmware (10) signaling asynchronously defects to the operating system in case of an error.

16. The method according to any one of claims 13 to 15, further transferring the data by the input/output store instruction (30) through an asynchronous transmit mechanism with an early completion message in multiple data packets to the aggregation buffer (16), if the length of the data exceeds eight bytes.

17. The method according to any one of claims 13 to 16, further the system firmware (10) using an asynchronous input/output driver code (32) for handling the input/output store instruction (30).

18. The method according to claim 17, further the core (12) using an asynchronous setup code (34) for handling memory requirements for status information of the asynchronous input/output driver code (32).

19. The method according to any one of claim 13 to 18, further the asynchronous core-nest interface (14) using an asynchronous core-nest interface forwarding component (36) for forwarding the data with local completion.

20. The method according to any one of claim 13 to 19, further the aggregation buffer (16) using an early completion logic (26) for delivering a free for reuse message after sending a request.

21. The method according to any one of claim 13 to 20, wherein a system message comprises one of

- a hierarchical physical target address,
- sourcing an SMT thread or an aggregate buffer identifier,
- a length of data,
- an input/output bus address,
- a input/output status buffer index.

22. The method according to any one of the claims 13 to 21, wherein the asynchronous input/output driver code (32) is storing a copy of each input/output operation in the retry buffer (52) and, if the asynchronous input/output driver code (32) detects an error, transferring control to the analysis and retry logic (54).

23. The method according to any one of the claims 13 to 22, the analysis and retry logic (54)

- determining a failing store instruction (30),
- determining an input/output function of that store instruction (30),
- initiating a retry for retryable errors, if below a threshold,
- if an error occurs in the asynchronous core-nest interface (14) and the threshold is exceeded, deleting all outstanding requests for that input/output function; setting the input/output function to an error state, and signaling an asynchronous error to the operating system,
- if a fatal error occurs, determining the source of the error; if the source is the

system nest (18) deleting all, setting all involved input/output functions to an error state, and signaling an asynchronous error to the operating system; if the source is the input/output bus controller (20), deleting all outstanding requests for input/output functions attached to that input/output bus controller (20); setting all input/output functions to an error state, and signaling an asynchronous error to the operating system.

24. A computer program product for handling an input/output store instruction (30) to at least one external device (214) of a data processing system (210), the data processing system (210) comprising
a system nest (10) communicatively coupled to at least one input/output bus (22) by an input/output bus controller (14),
and further comprising at least a data processing unit (216) comprising a core (12), a system firmware (10) and an asynchronous core-nest interface (14),
wherein the data processing unit (216) is communicatively coupled to the system nest (18) via an aggregation buffer (16),
wherein the external device (214) is communicatively coupled to the input/output bus (22),
wherein the asynchronous core-nest interface (14) comprises an input/output status array (44) with multiple input/output status buffers (24) and an array management and access logic (46),
wherein the system firmware (10) comprises a retry buffer (52) and the core (12) comprises an analysis and retry logic (54),
the computer program product comprising a computer readable storage medium having program instructions embodied therewith, the program instructions executable by the computer system (212) to cause the computer system (212) to perform a method comprising

(i) an operating system running on the data processing system (210) issuing the input/output store instruction (30) specifying at least an input/output function with an offset through an address, data to be transferred and/or a pointer to data to be transferred, and a length of the data;

(ii) the data processing unit (216) being configured to identify the input/output function by the address specified in the input/output store instruction (30);

(iii) the data processing unit (216) being configured to verify if access to the input/output function is allowed on an address space and on a guest instance level, the guest running on the data processing system (210);

(iv) the data processing unit (216) being configured to complete the input/output store instruction (30) before an execution of the input/output store instruction (30) in the system nest (18) is completed;

(v) the system firmware (10) being configured to notify the operating system through an interrupt, if during the asynchronous execution of the input/output store instruction (30) an error is detected by the data processing unit (216);

(vi) the analysis and retry logic (54) detecting separately errors being ensured by the hardware that the store instruction (30) has not been forwarded to an input/output bus (22) yet;

(vii) the retry buffer (52) keeping store information for retries of executing the store instruction (30) in system hardware/firmware (50);

(viii) the analysis and retry logic (54) analyzing errors and checking for retry possibility;

(ix) the analysis and retry logic (54) triggering retries.

25. A data processing system (210) for execution of a data processing program (240) comprising computer readable program instructions for performing a method according to any one of claims 13 to 23.

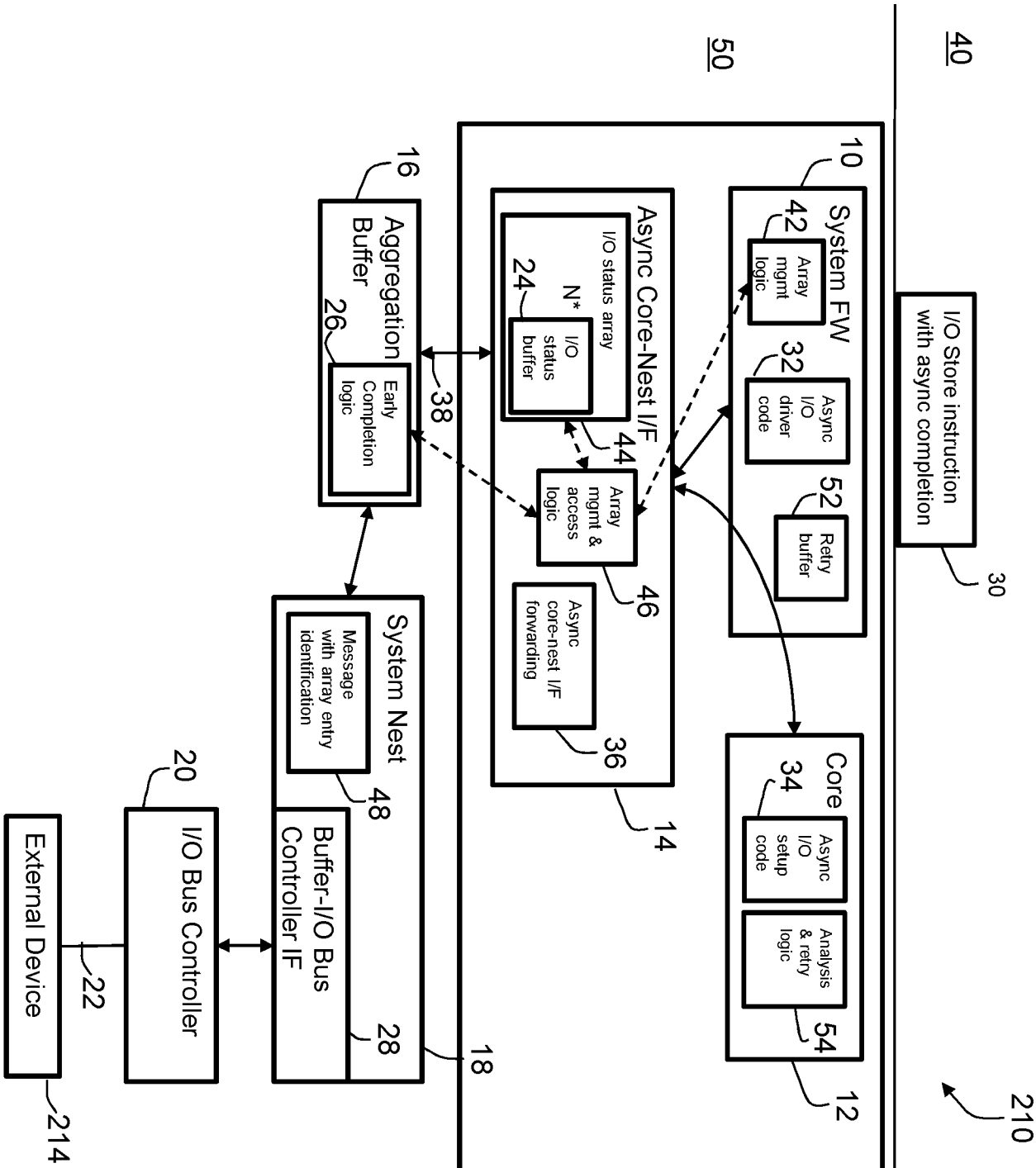
40
50
I/O Store instruction with async completion
30
10
System FW
42
Array mgmt logic
32
Async I/O driver code
52
Retry buffer
Core
34
Async I/O setup code
Analysis & retry logic
54
12
210
216
Architecture Boundary
Async Core-Nest I/F
I/O status array
N*
24
I/O status buffer
44
Array mgmt & access logic
46
Async core-nest I/F forwarding
36
14
16
Aggregation Buffer
26
Early Completion logic
38
System Nest
Message with array entry identification
48
Buffer-I/O Bus Controller IF
28
18
20
I/O Bus Controller
22
External Device
214

Fig. 1

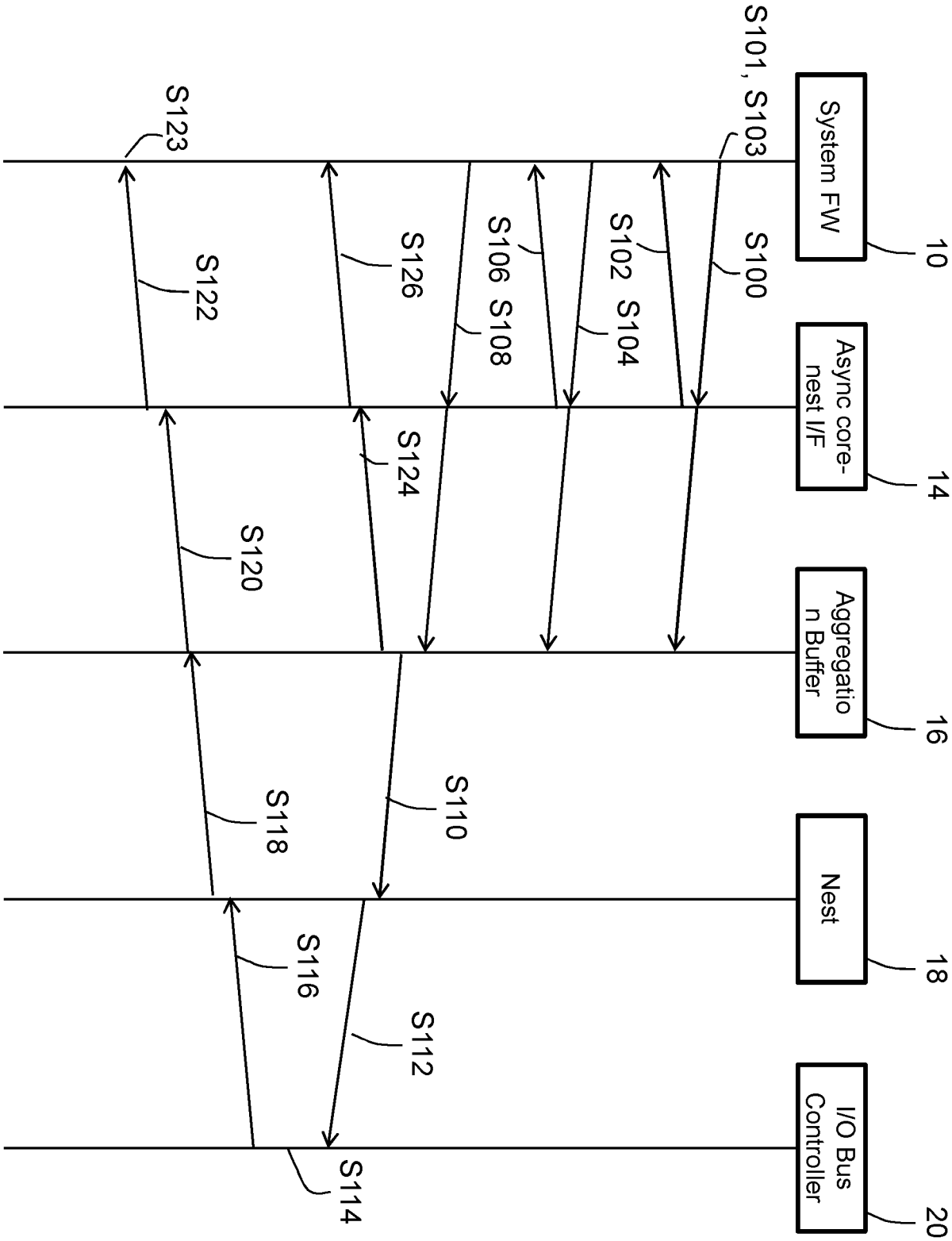
System FW
10
Async core-nest I/F
14
Aggregation Buffer
16
Nest
18
I/O Bus Controller
20
S101, S103
S100
S102
S104
S106
S108
S110
S112
S114
S116
S118
S120
S122
S123
S124
S126

Fig. 2

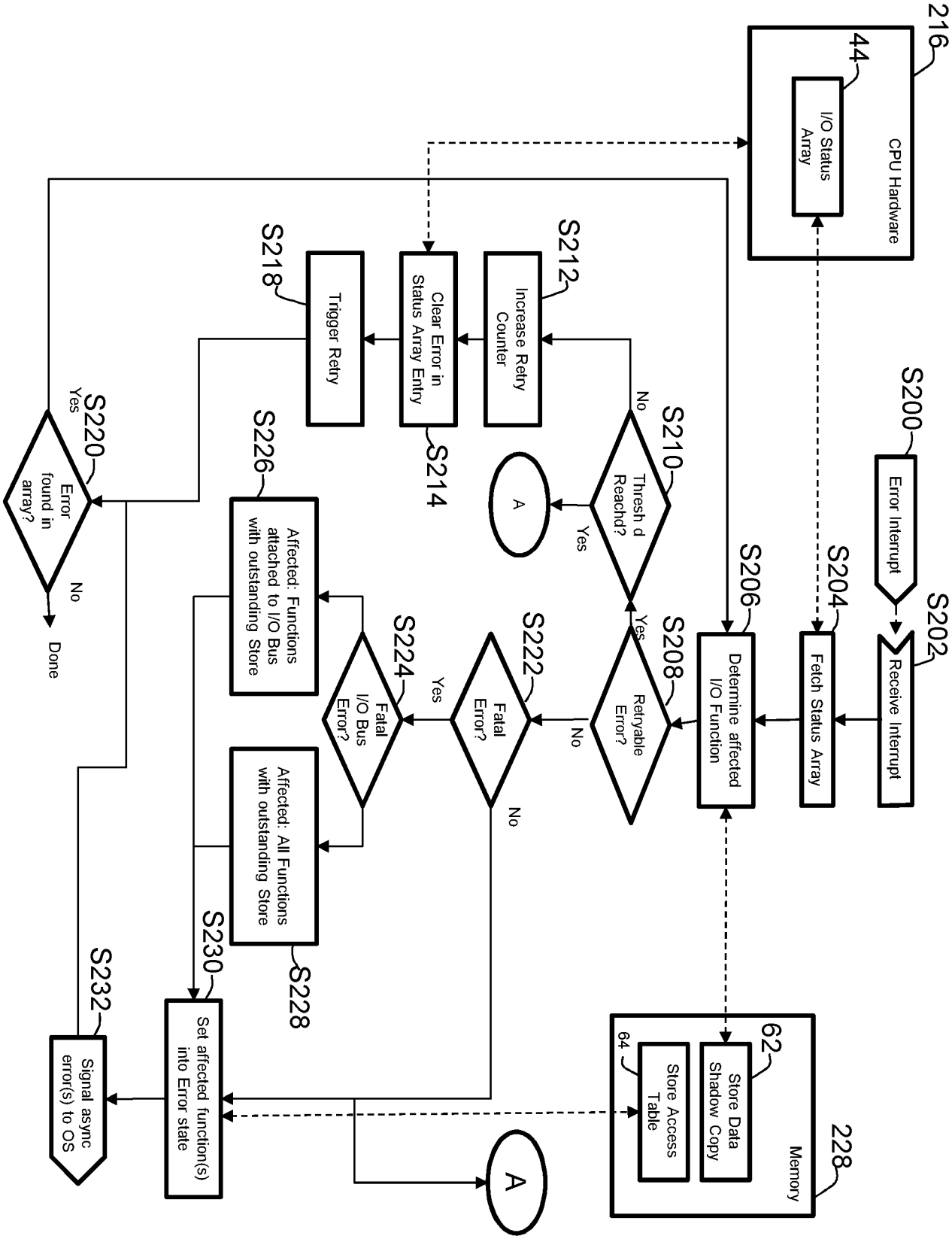
216
CPU Hardware
44
I/O Status Array
S200
Error Interrupt
S202
Receive Interrupt
S204
Fetch Status Array
S206
Determine affected I/O Function
S208
Retryable Error?
No
Yes
S210
Thresh d Reachd?
No
Yes
S212
Increase Retry Counter
S214
Clear Error in Status Array Entry
S218
Trigger Retry
A
S220
Error found in array?
Yes
No
Done
S222
Fatal Error?
Yes
No
S224
Fatal I/O Bus Error?
Yes
S226
Affected: Functions attached to I/O Bus with outstanding Store
S228
Affected: All Functions with outstanding Store
S230
Set affected function(s) into Error state
S232
Signal async error(s) to OS
A
62
Store Data Shadow Copy
64
Store Access Table
228
Memory

Fig. 3

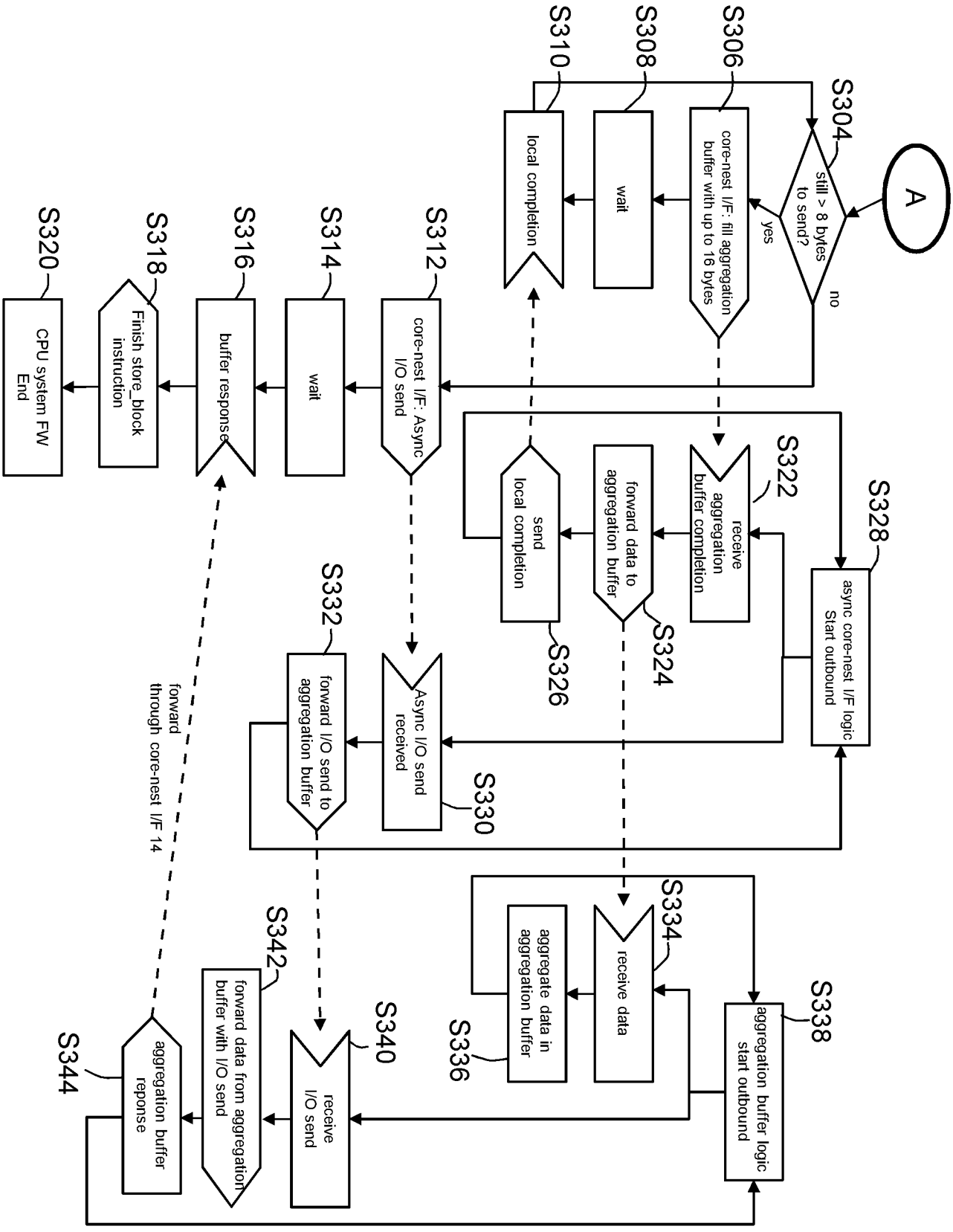
A
S304
still > 8 bytes to send?
yes
no
S306
core-nest I/F: fill aggregation buffer with up to 16 bytes
S308
wait
S310
local completion
S312
core-nest I/F: Async I/O send
S314
wait
S316
buffer response
S318
Finish store_block instruction
S320
CPU system FW End
S322
receive aggregation buffer completion
S324
forward data to aggregation buffer
S326
send local completion
S328
async core-nest I/F logic Start outbound
S330
Async I/O send received
S332
forward I/O send to aggregation buffer
forward through core-nest I/F 14
S334
receive data
S336
aggregate data in aggregation buffer
S338
aggregation buffer logic start outbound
S340
receive I/O send
S342
forward data from aggregation buffer with I/O send
S344
aggregation buffer reponse

Fig. 4

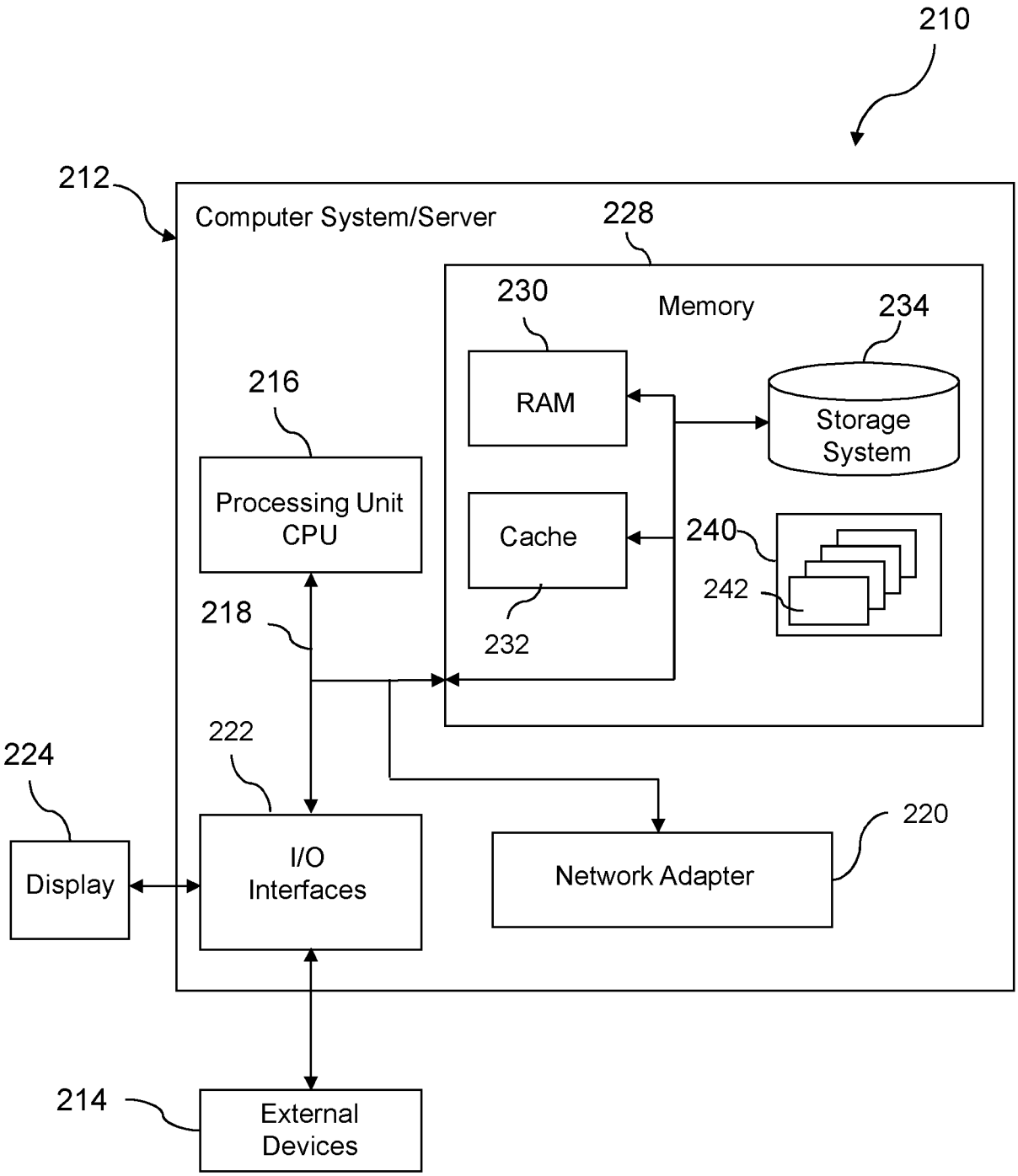
210
212
Computer System/Server
228
230
Memory
234
RAM
Storage System
216
Processing Unit CPU
Cache
240
242
232
218
222
224
220
Display
I/O Interfaces
Network Adapter
214
External Devices

Fig. 5

INTERNATIONAL SEARCH REPORT

International application No.
PCT/IB2020/050339

A. CLASSIFICATION OF SUBJECT MATTER

G06F 13/10(2006.01)i

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

CNKI, SIPOABS, DWPI, CNTXT, USTXT: data, process, input, output, store, instruction, nest, system, bus, controller, core, asynchronous, interface, load, status, array, buffer, management, access, firmware, retry, analysis, offset, pointer, address, space, guest, instance, interrupt, error, forward, check, trigger

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	CN 105912489 A (QUANTA COMPUTER INC.) 31 August 2016 (2016-08-31) the whole document	1-25
A	US 7178019 B2 (HEWLETT PACKARD DEVELOPMENT COMPANY) 13 February 2007 (2007-02-13) the whole document	1-25
A	US 7631097 B2 (NATIONAL INSTRUMENTS CORPORATION) 08 December 2009 (2009-12-08) the whole document	1-25

☐ Further documents are listed in the continuation of Box C. ☑ See patent family annex.

* Special categories of cited documents:
"A" document defining the general state of the art which is not considered to be of particular relevance
"E" earlier application or patent but published on or after the international filing date
"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
"O" document referring to an oral disclosure, use, exhibition or other means
"P" document published prior to the international filing date but later than the priority date claimed
"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
21 April 2020	**07 May 2020**
Name and mailing address of the ISA/CN **National Intellectual Property Administration, PRC** **6, Xitucheng Rd., Jimen Bridge, Haidian District, Beijing 100088** **China** Facsimile No. **(86-10)62019451**	Authorized officer **ZHONG,Yangxue** Telephone No. **86-010-62411634**

INTERNATIONAL SEARCH REPORT
Information on patent family members

International application No.
PCT/IB2020/050339

Patent document cited in search report			Publication date (day/month/year)	Patent family member(s)			Publication date (day/month/year)
CN	105912489	A	31 August 2016	JP	2016157417	A	01 September 2016
				TW	I537748	B	11 June 2016
				US	9542201	B2	10 January 2017
				US	2016246612	A1	25 August 2016
				JP	6199940	B2	20 September 2017
				EP	3062216	A1	31 August 2016
				TW	201631498	A	01 September 2016
US	7178019	B2	13 February 2007	US	2005108513	A1	19 May 2005
US	7631097	B2	08 December 2009	US	7849210	B2	07 December 2010
				US	2007022204	A1	25 January 2007
				US	2009100201	A1	16 April 2009