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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
30 March 2017 (30.03.2017)

WIPO | PCT

(10) International Publication Number
WO 2017/052909 A1

(51) International Patent Classification:
G06F 13/42 (2006.01) *G06F 13/40* (2006.01)

(21) International Application Number:
PCT/US2016/048256

(22) International Filing Date:
24 August 2016 (24.08.2016)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
14/866,862 26 September 2015 (26.09.2015) US

(71) Applicant: **INTEL CORPORATION** [US/US]; 2200 Mission College Boulevard, Santa Clara, California 95054-1549 (US).

(72) Inventors: **ARELLANO, Philip C.**; 1079 E. Minton Dr., Tempe, Arizona 85282 (US). **COLEMAN, James A.**; 2620 North Mountview, Mesa, Arizona 85203 (US).

(74) Agent: **GUPTA, Rishi**; Patent Capital Group, c/o CPA Global, 900 Second Avenue South, Suite 600, Minneapolis, Minnesota 55402 (US).

(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, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IR, IS, JP, KE, KG, KN, KP, KR, 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, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, 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).

[Continued on next page]

(54) Title: A METHOD, APPARATUS, AND SYSTEM FOR ALLOCATING CACHE USING TRAFFIC CLASS

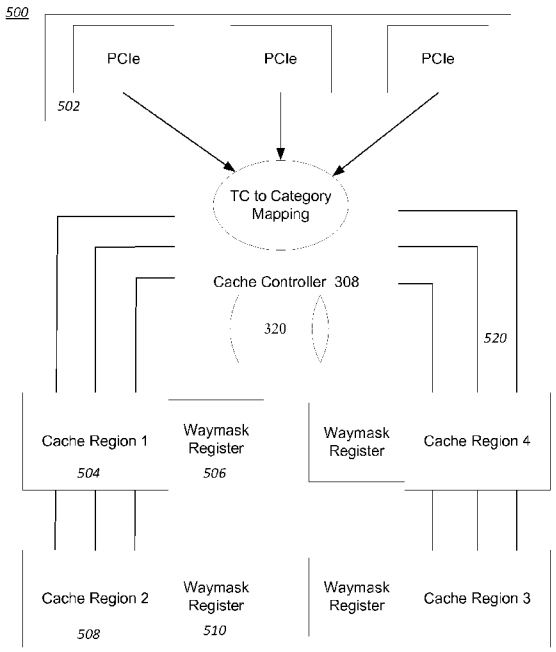


FIG. 5

(57) Abstract: This disclosure pertains to using traffic classes to selectively store data into cache memory or into system memory. A cache controller can map the traffic class of incoming data to portions of the cache memory allocated for corresponding traffic classes of data.

WO 2017/052909 A1

Declarations under Rule 4.17:

— *as to applicant's entitlement to apply for and be granted a patent (Rule 4.17(ii))*

— *as to the applicant's entitlement to claim the priority of the earlier application (Rule 4.17(iii))*

Published:

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

A METHOD, APPARATUS, AND SYSTEM FOR ALLOCATING CACHE USING TRAFFIC CLASS

CROSS-REFERENCE TO RELATED APPLICATION

[0001] This application claims the benefit of priority to U.S. Nonprovisional (Utility) Patent Application No. 14/866,862 filed 26 September 2015 entitled "METHOD, APPARATUS, AND SYSTEM FOR ALLOCATING CACHE USING TRAFFIC CLASS", which is incorporated herein by reference in its entirety.

FIELD

[0002] This disclosure pertains to computing system, and in particular (but not exclusively) to allocating cache memory using PCIe traffic classes.

BACKGROUND

[0003] Computer-based applications, including real-time embedded applications, may include input/output (IO) devices that carry high-priority (real-time) data, as well as low priority best effort data. The high priority traffic is often to be processed quickly by the CPU, however even with technologies like Data Direct I/O (DDIO) which allow IO transactions to target Cache directly; the high priority transactions can be pushed out of cache by the competing best effort IO traffic. The inability to prioritize which transactions target cache, and which are evicted when additional IO transactions come in, renders DDIO ineffective for applications that have a demand for fast cache transactions.

BRIEF DESCRIPTION OF THE DRAWINGS

[0004] FIG. 1 illustrates a block diagram for an embodiment of a computing system including a multicore processor.

[0005] FIG. 2 illustrates an embodiment of a computing system including an interconnect architecture.

[0006] FIG. 3 illustrates a block diagram of an embodiment of a computing system that includes a cache controller in accordance with embodiments of the present disclosure.

[0007] FIG. 4 illustrates a block diagram of an embodiment of a cache controller in accordance with embodiments of the present disclosure.

[0008] FIG. 5 illustrates a schematic diagram of an example implementation of a cache controller and a cache memory in accordance with embodiments of the present disclosure.

[0009] FIG. 6 is a process flow diagram of selectively storing data directly into cache memory based on traffic class.

[0010] FIG. 7 illustrates an embodiment of a interconnect architecture including a layered stack.

[0011] FIG. 8 illustrates an embodiment of a request or packet to be generated or received within an interconnect architecture.

[0012] FIG. 9 illustrates an embodiment of a transmitter and receiver pair for an interconnect architecture.

[0013] FIG. 10 an embodiment of multiple potential multi-socket configurations.

[0014] FIG. 11 illustrates an embodiment of a layered stack for a high performance interconnect architecture.

[0015] FIG. 12 illustrates another embodiment of a block diagram for a computing system including a processor.

[0016] FIG. 13 illustrates an embodiment of a block diagram for a processor.

[0017] FIG. 14 illustrates another embodiment of a block diagram for a computing system including a processor.

[0018] FIG. 15 illustrates an embodiment of a block for a computing system including multiple processor sockets.

[0019] FIG. 16 illustrates another embodiment of a block diagram for a computing system.

[0020] FIG. 17 illustrates another embodiment of a block diagram for a computing system.

[0021] FIG. 18 is a graph showing performance differences cache accesses in during scenarios.

DETAILED DESCRIPTION

[0022] This disclosure describes using traffic class bits present in transactions to selectively store data in either cache memory or in system memory. For example, the disclosure describes using the PCIe traffic class bits present in PCIe transactions. The traffic class map those transactions to a category of data, a class of service (CLOS), or other cache memory register classification. The cache register bits on the ring are used to map the transactions to way masks for cache allocation. Certain aspects of this disclosure extend the current DDIO functionality to have Traffic Class level resolution for cache allocation.

[0023] In the following description, numerous specific details are set forth, such as examples of specific types of processors and system configurations, specific hardware structures, specific architectural and micro architectural details, specific register configurations,

specific instruction types, specific system components, specific measurements/heights, specific processor pipeline stages and operation etc. in order to provide a thorough understanding of the present invention. It will be apparent, however, to one skilled in the art that these specific details need not be employed to practice the present invention. In other instances, well known components or methods, such as specific and alternative processor architectures, specific logic circuits/code for described algorithms, specific firmware code, specific interconnect operation, specific logic configurations, specific manufacturing techniques and materials, specific compiler implementations, specific expression of algorithms in code, specific power down and gating techniques/logic and other specific operational details of computer system haven't been described in detail in order to avoid unnecessarily obscuring the present invention.

[0024] Although the following embodiments may be described with reference to energy conservation and energy efficiency in specific integrated circuits, such as in computing platforms or microprocessors, other embodiments are applicable to other types of integrated circuits and logic devices. Similar techniques and teachings of embodiments described herein may be applied to other types of circuits or semiconductor devices that may also benefit from better energy efficiency and energy conservation. For example, the disclosed embodiments are not limited to desktop computer systems or Ultrabooks™. And may be also used in other devices, such as handheld devices, tablets, other thin notebooks, systems on a chip (SOC) devices, and embedded applications. Some examples of handheld devices include cellular phones, Internet protocol devices, digital cameras, personal digital assistants (PDAs), and handheld PCs. Embedded applications typically include a microcontroller, a digital signal processor (DSP), a system on a chip, network computers (NetPC), set-top boxes, network hubs, wide area network (WAN) switches, or any other system that can perform the functions and operations taught below. Moreover, the apparatus', methods, and systems described herein are not limited to physical computing devices, but may also relate to software optimizations for energy conservation and efficiency. As will become readily apparent in the description below, the embodiments of methods, apparatuses, and systems described herein (whether in reference to hardware, firmware, software, or a combination thereof) are vital to a 'green technology' future balanced with performance considerations.

[0025] As computing systems are advancing, the components therein are becoming more complex. As a result, the interconnect architecture to couple and communicate between the components is also increasing in complexity to ensure bandwidth requirements are met for optimal component operation. Furthermore, different market segments demand different aspects of interconnect architectures to suit the market's needs. For example, servers require higher performance, while the mobile ecosystem is sometimes able to sacrifice overall performance for

power savings. Yet, it is a singular purpose of most fabrics to provide highest possible performance with maximum power saving. Below, a number of interconnects are discussed, which would potentially benefit from aspects of the invention described herein.

[0026] Referring to FIG. 1, an embodiment of a block diagram for a computing system including a multicore processor is depicted. Processor 100 includes any processor or processing device, such as a microprocessor, an embedded processor, a digital signal processor (DSP), a network processor, a handheld processor, an application processor, a co-processor, a system on a chip (SOC), or other device to execute code. Processor 100, in one embodiment, includes at least two cores—core 101 and 102, which may include asymmetric cores or symmetric cores (the illustrated embodiment). However, processor 100 may include any number of processing elements that may be symmetric or asymmetric.

[0027] In one embodiment, a processing element refers to hardware or logic to support a software thread. Examples of hardware processing elements include: a thread unit, a thread slot, a thread, a process unit, a context, a context unit, a logical processor, a hardware thread, a core, and/or any other element, which is capable of holding a state for a processor, such as an execution state or architectural state. In other words, a processing element, in one embodiment, refers to any hardware capable of being independently associated with code, such as a software thread, operating system, application, or other code. A physical processor (or processor socket) typically refers to an integrated circuit, which potentially includes any number of other processing elements, such as cores or hardware threads.

[0028] A core often refers to logic located on an integrated circuit capable of maintaining an independent architectural state, wherein each independently maintained architectural state is associated with at least some dedicated execution resources. In contrast to cores, a hardware thread typically refers to any logic located on an integrated circuit capable of maintaining an independent architectural state, wherein the independently maintained architectural states share access to execution resources. As can be seen, when certain resources are shared and others are dedicated to an architectural state, the line between the nomenclature of a hardware thread and core overlaps. Yet often, a core and a hardware thread are viewed by an operating system as individual logical processors, where the operating system is able to individually schedule operations on each logical processor.

[0029] Physical processor 100, as illustrated in FIG. 1, includes two cores—core 101 and 102. Here, core 101 and 102 are considered symmetric cores, i.e. cores with the same configurations, functional units, and/or logic. In another embodiment, core 101 includes an out-of-order processor core, while core 102 includes an in-order processor core. However, cores 101 and 102 may be individually selected from any type of core, such as a native core, a

software managed core, a core adapted to execute a native Instruction Set Architecture (ISA), a core adapted to execute a translated Instruction Set Architecture (ISA), a co-designed core, or other known core. In a heterogeneous core environment (i.e. asymmetric cores), some form of translation, such a binary translation, may be utilized to schedule or execute code on one or both cores. Yet to further the discussion, the functional units illustrated in core 101 are described in further detail below, as the units in core 102 operate in a similar manner in the depicted embodiment.

[0030] As depicted, core 101 includes two hardware threads 101a and 101b, which may also be referred to as hardware thread slots 101a and 101b. Therefore, software entities, such as an operating system, in one embodiment potentially view processor 100 as four separate processors, i.e., four logical processors or processing elements capable of executing four software threads concurrently. As alluded to above, a first thread is associated with architecture state registers 101a, a second thread is associated with architecture state registers 101b, a third thread may be associated with architecture state registers 102a, and a fourth thread may be associated with architecture state registers 102b. Here, each of the architecture state registers (101a, 101b, 102a, and 102b) may be referred to as processing elements, thread slots, or thread units, as described above. As illustrated, architecture state registers 101a are replicated in architecture state registers 101b, so individual architecture states/contexts are capable of being stored for logical processor 101a and logical processor 101b. In core 101, other smaller resources, such as instruction pointers and renaming logic in allocator and renamer block 130 may also be replicated for threads 101a and 101b. Some resources, such as re-order buffers in reorder/retirement unit 135, ILTB 120, load/store buffers, and queues may be shared through partitioning. Other resources, such as general purpose internal registers, page-table base register(s), low-level data-cache and data-TLB 115, execution unit(s) 140, and portions of out-of-order unit 135 are potentially fully shared.

[0031] Processor 100 often includes other resources, which may be fully shared, shared through partitioning, or dedicated by/to processing elements. In FIG. 1, an embodiment of a purely exemplary processor with illustrative logical units/resources of a processor is illustrated. Note that a processor may include, or omit, any of these functional units, as well as include any other known functional units, logic, or firmware not depicted. As illustrated, core 101 includes a simplified, representative out-of-order (OOO) processor core. But an in-order processor may be utilized in different embodiments. The OOO core includes a branch target buffer 120 to predict branches to be executed/taken and an instruction-translation buffer (I-TLB) 120 to store address translation entries for instructions.

[0032] Core 101 further includes decode module 125 coupled to fetch unit 120 to decode fetched elements. Fetch logic, in one embodiment, includes individual sequencers associated with thread slots 101a, 101b, respectively. Usually core 101 is associated with a first ISA, which defines/specifies instructions executable on processor 100. Often machine code instructions that are part of the first ISA include a portion of the instruction (referred to as an opcode), which references/specifies an instruction or operation to be performed. Decode logic 125 includes circuitry that recognizes these instructions from their opcodes and passes the decoded instructions on in the pipeline for processing as defined by the first ISA. For example, as discussed in more detail below decoders 125, in one embodiment, include logic designed or adapted to recognize specific instructions, such as transactional instruction. As a result of the recognition by decoders 125, the architecture or core 101 takes specific, predefined actions to perform tasks associated with the appropriate instruction. It is important to note that any of the tasks, blocks, operations, and methods described herein may be performed in response to a single or multiple instructions; some of which may be new or old instructions. Note decoders 126, in one embodiment, recognize the same ISA (or a subset thereof). Alternatively, in a heterogeneous core environment, decoders 126 recognize a second ISA (either a subset of the first ISA or a distinct ISA).

[0033] In one example, allocator and renamer block 130 includes an allocator to reserve resources, such as register files to store instruction processing results. However, threads 101a and 101b are potentially capable of out-of-order execution, where allocator and renamer block 130 also reserves other resources, such as reorder buffers to track instruction results. Unit 130 may also include a register renamer to rename program/instruction reference registers to other registers internal to processor 100. Reorder/retirement unit 135 includes components, such as the reorder buffers mentioned above, load buffers, and store buffers, to support out-of-order execution and later in-order retirement of instructions executed out-of-order.

[0034] Scheduler and execution unit(s) block 140, in one embodiment, includes a scheduler unit to schedule instructions/operation on execution units. For example, a floating point instruction is scheduled on a port of an execution unit that has an available floating point execution unit. Register files associated with the execution units are also included to store information instruction processing results. Exemplary execution units include a floating point execution unit, an integer execution unit, a jump execution unit, a load execution unit, a store execution unit, and other known execution units.

[0035] Lower level data cache and data translation buffer (D-TLB) 150 are coupled to execution unit(s) 140. The data cache is to store recently used/operated on elements, such as data operands, which are potentially held in memory coherency states. The D-TLB is to store

recent virtual/linear to physical address translations. As a specific example, a processor may include a page table structure to break physical memory into a plurality of virtual pages.

[0036] Here, cores 101 and 102 share access to higher-level or further-out cache, such as a second level cache associated with on-chip interface 110. Note that higher-level or further-out refers to cache levels increasing or getting further way from the execution unit(s). In one embodiment, higher-level cache is a last-level data cache—last cache in the memory hierarchy on processor 100—such as a second or third level data cache. However, higher level cache is not so limited, as it may be associated with or include an instruction cache. A trace cache—a type of instruction cache—instead may be coupled after decoder 125 to store recently decoded traces. Here, an instruction potentially refers to a macro-instruction (i.e. a general instruction recognized by the decoders), which may decode into a number of micro-instructions (micro-operations).

[0037] In the depicted configuration, processor 100 also includes on-chip interface module 110. Historically, a memory controller, which is described in more detail below, has been included in a computing system external to processor 100. In this scenario, on-chip interface 11 is to communicate with devices external to processor 100, such as system memory 175, a chipset (often including a memory controller hub to connect to memory 175 and an I/O controller hub to connect peripheral devices), a memory controller hub, a northbridge, or other integrated circuit. And in this scenario, bus 105 may include any known interconnect, such as multi-drop bus, a point-to-point interconnect, a serial interconnect, a parallel bus, a coherent (e.g. cache coherent) bus, a layered protocol architecture, a differential bus, and a GTL bus.

[0038] Memory 175 may be dedicated to processor 100 or shared with other devices in a system. Common examples of types of memory 175 include DRAM, SRAM, non-volatile memory (NV memory), and other known storage devices. Note that device 180 may include a graphic accelerator, processor or card coupled to a memory controller hub, data storage coupled to an I/O controller hub, a wireless transceiver, a flash device, an audio controller, a network controller, or other known device.

[0039] Recently however, as more logic and devices are being integrated on a single die, such as SOC, each of these devices may be incorporated on processor 100. For example in one embodiment, a memory controller hub is on the same package and/or die with processor 100. Here, a portion of the core (an on-core portion) 110 includes one or more controller(s) for interfacing with other devices such as memory 175 or a graphics device 180. The configuration including an interconnect and controllers for interfacing with such devices is often referred to as an on-core (or un-core configuration). As an example, on-chip interface 110 includes a ring interconnect for on-chip communication and a high-speed serial point-to-point link 105 for off-

chip communication. Yet, in the SOC environment, even more devices, such as the network interface, co-processors, memory 175, graphics processor 180, and any other known computer devices/interface may be integrated on a single die or integrated circuit to provide small form factor with high functionality and low power consumption.

[0040] In one embodiment, processor 100 is capable of executing a compiler, optimization, and/or translator code 177 to compile, translate, and/or optimize application code 176 to support the apparatus and methods described herein or to interface therewith. A compiler often includes a program or set of programs to translate source text/code into target text/code. Usually, compilation of program/application code with a compiler is done in multiple phases and passes to transform hi-level programming language code into low-level machine or assembly language code. Yet, single pass compilers may still be utilized for simple compilation. A compiler may utilize any known compilation techniques and perform any known compiler operations, such as lexical analysis, preprocessing, parsing, semantic analysis, code generation, code transformation, and code optimization.

[0041] Larger compilers often include multiple phases, but most often these phases are included within two general phases: (1) a front-end, i.e. generally where syntactic processing, semantic processing, and some transformation/optimization may take place, and (2) a back-end, i.e. generally where analysis, transformations, optimizations, and code generation takes place. Some compilers refer to a middle, which illustrates the blurring of delineation between a front-end and back end of a compiler. As a result, reference to insertion, association, generation, or other operation of a compiler may take place in any of the aforementioned phases or passes, as well as any other known phases or passes of a compiler. As an illustrative example, a compiler potentially inserts operations, calls, functions, etc. in one or more phases of compilation, such as insertion of calls/operations in a front-end phase of compilation and then transformation of the calls/operations into lower-level code during a transformation phase. Note that during dynamic compilation, compiler code or dynamic optimization code may insert such operations/calls, as well as optimize the code for execution during runtime. As a specific illustrative example, binary code (already compiled code) may be dynamically optimized during runtime. Here, the program code may include the dynamic optimization code, the binary code, or a combination thereof.

[0042] Similar to a compiler, a translator, such as a binary translator, translates code either statically or dynamically to optimize and/or translate code. Therefore, reference to execution of code, application code, program code, or other software environment may refer to: (1) execution of a compiler program(s), optimization code optimizer, or translator either dynamically or statically, to compile program code, to maintain software structures, to perform

other operations, to optimize code, or to translate code; (2) execution of main program code including operations/calls, such as application code that has been optimized/compiled; (3) execution of other program code, such as libraries, associated with the main program code to maintain software structures, to perform other software related operations, or to optimize code; or (4) a combination thereof.

[0043] One interconnect fabric architecture includes the Peripheral Component Interconnect (PCI) Express (PCIe) architecture. A primary goal of PCIe is to enable components and devices from different vendors to inter-operate in an open architecture, spanning multiple market segments; Clients (Desktops and Mobile), Servers (Standard and Enterprise), and Embedded and Communication devices. PCI Express is a high performance, general purpose I/O interconnect defined for a wide variety of future computing and communication platforms. Some PCI attributes, such as its usage model, load-store architecture, and software interfaces, have been maintained through its revisions, whereas previous parallel bus implementations have been replaced by a highly scalable, fully serial interface. The more recent versions of PCI Express take advantage of advances in point-to-point interconnects, Switch-based technology, and packetized protocol to deliver new levels of performance and features. Power Management, Quality Of Service (QoS), Hot-Plug/Hot- Swap support, Data Integrity, and Error Handling are among some of the advanced features supported by PCI Express.

[0044] Referring to FIG. 2, an embodiment of a fabric composed of point-to-point Links that interconnect a set of components is illustrated. System 200 includes processor 205 and system memory 210 coupled to controller hub 215. Processor 205 includes any processing element, such as a microprocessor, a host processor, an embedded processor, a co-processor, or other processor. Processor 205 is coupled to controller hub 215 through front-side bus (FSB) 206. In one embodiment, FSB 206 is a serial point-to-point interconnect as described below. In another embodiment, link 206 includes a serial, differential interconnect architecture that is compliant with different interconnect standard.

[0045] System memory 210 includes any memory device, such as random access memory (RAM), non-volatile (NV) memory, or other memory accessible by devices in system 200. System memory 210 is coupled to controller hub 215 through memory interface 216. Examples of a memory interface include a double-data rate (DDR) memory interface, a dual-channel DDR memory interface, and a dynamic RAM (DRAM) memory interface.

[0046] In one embodiment, controller hub 215 is a root hub, root complex, or root controller in a Peripheral Component Interconnect Express (PCIe or PCIE) interconnection hierarchy. Examples of controller hub 215 include a chipset, a memory controller hub (MCH), a

northbridge, an interconnect controller hub (ICH) a southbridge, and a root controller/hub. Often the term chipset refers to two physically separate controller hubs, i.e. a memory controller hub (MCH) coupled to an interconnect controller hub (ICH). Note that current systems often include the MCH integrated with processor 205, while controller 215 is to communicate with I/O devices, in a similar manner as described below. In some embodiments, peer-to-peer routing is optionally supported through root complex 215.

[0047] Here, controller hub 215 is coupled to switch/bridge 220 through serial link 219. Input/output modules 217 and 221, which may also be referred to as interfaces/ports 217 and 221, include/implement a layered protocol stack to provide communication between controller hub 215 and switch 220. In one embodiment, multiple devices are capable of being coupled to switch 220.

[0048] Switch/bridge 220 routes packets/messages from device 225 upstream, i.e. up a hierarchy towards a root complex, to controller hub 215 and downstream, i.e. down a hierarchy away from a root controller, from processor 205 or system memory 210 to device 225. Switch 220, in one embodiment, is referred to as a logical assembly of multiple virtual PCI-to-PCI bridge devices. Device 225 includes any internal or external device or component to be coupled to an electronic system, such as an I/O device, a Network Interface Controller (NIC), an add-in card, an audio processor, a network processor, a hard-drive, a storage device, a CD/DVD ROM, a monitor, a printer, a mouse, a keyboard, a router, a portable storage device, a Firewire device, a Universal Serial Bus (USB) device, a scanner, and other input/output devices. Often in the PCIe vernacular, such as device, is referred to as an endpoint. Although not specifically shown, device 225 may include a PCIe to PCI/PCI-X bridge to support legacy or other version PCI devices. Endpoint devices in PCIe are often classified as legacy, PCIe, or root complex integrated endpoints.

[0049] Graphics accelerator 230 is also coupled to controller hub 215 through serial link 232. In one embodiment, graphics accelerator 230 is coupled to an MCH, which is coupled to an ICH. Switch 220, and accordingly I/O device 225, is then coupled to the ICH. I/O modules 231 and 218 are also to implement a layered protocol stack to communicate between graphics accelerator 230 and controller hub 215. Similar to the MCH discussion above, a graphics controller or the graphics accelerator 230 itself may be integrated in processor 205.

[0050] FIG. 3 illustrates a block diagram of an embodiment of a computing system that includes a cache controller in accordance with embodiments of the present disclosure. Computing system 300 includes a central processing unit (CPU) socket 302, one or more CPUs 304(1), 304(2), ... 304(n), a cache memory 306, a cache controller 308, a system memory 314, an I/O interconnect 310, and a network interface controller (NIC) 312. In certain

implementations, the cache memory 306 is the CPU cache or Last Level Cache (LLC). The NIC 312 is configured to interface with a fabric interconnect and serve as an interface between the computing system 300 and networked or peripheral devices (or other devices), e.g., via PCIe busses or across a network or by other sources.

[0051] For DDIO operation, one of the CPUs 304(1)–304(n) accesses the cache memory 306 to create packets in response to memory access requests (i.e., read or write requests). In read accesses, data can be read directly from cache memory 306. In write accesses, the data can be delivered directly to cache memory 306, without first writing to system memory 314. The goal of DDIO is a reduction in the number of accesses to system memory 314. For DDIO to reduce the number of system memory access operations for either read or write, particularly in high-priority data transactions (e.g., real-time applications), the cache memory 306 should maintain high-priority data (or, put differently, eviction of high-priority data in favor of best-effort or low priority data should be minimized) so as to avoid cache misses that require system memory accesses.

[0052] For high-priority applications, DDIO provides a way to achieve low latency operations, but benefits from minimized cache misses of the high-priority data. A cache controller 308 can use traffic classes that are part of data packet header information for transactions to differentiate the cache allocation. For example, the traffic classes are PCIe traffic classes. PCIe traffic classes are described in the PCI Express Base Specification, Revision 3.0 (Nov. 10, 2010) [hereinafter, PCIe Spec], which is incorporated by reference in its entirety herein. As described in the PCIe Spec, a traffic class is an end-to-end label of a transaction layer packet and is used to apply servicing policies to the data packet at different points along the data path. Additionally, the PCIe Spec describes that traffic classes can be used for "traffic class differentiation" at an endpoint or root complex. Among the traffic classes are TC0 Best effort service class and TC1–TC7, which are differentiated service classes.

[0053] The cache controller 308 can use the traffic class obtained from the data packet to determine which regions of cache, if any, that data having a certain traffic class can be allocated into. The cache controller can identify a traffic class of a data packet.

[0054] FIG. 4 illustrates a block diagram of an embodiment of a cache controller 308 in accordance with the present disclosure. For example, the cache controller 308 includes a cache allocation controller 310 and a mapping element 312. The mapping element 312 includes circuitry to identify the PCIe traffic class label that is part of the data packet to identify a traffic class of the data packet (e.g., TC0 or TC1–TC7). The mapping element 312 can access a registry 320 to map the identified traffic class of the data packet to a register bit value corresponding to a region of the cache memory associated with that register bit value. The

registry 320 may include a cross referencing of PCIe traffic classes to registers of the cache memory available or assigned for storing data associated with said traffic classes. The registry 320 may include a cross referencing of traffic classes to categories of data. The categories of data associated with registers of regions of the cache or ways of the cache.

[0055] The cache allocation controller 310 can then determine whether the data should be stored in cache or not depending on the traffic class of the data. If the data can be stored in cache based on its traffic class, then the cache controller 310 can also determine whether the data can be allocated into a corresponding region of the cache associated with the traffic class of the data. The cache allocation controller 310 can also evict low-priority data stored in cache to make room for data having high-priority traffic class.

[0056] FIG. 5 illustrates a schematic diagram of an example implementation of a cache controller and a cache memory in accordance with embodiments of the present disclosure. FIG. 5 shows PCIe root complex 502 in communication with cache controller 308. Cache controller 308 includes the features described in FIG. 4 to identify a traffic class (TC) and to map the TC to a category of data (or, in some implementations, to a class of service (CLOS)). For example, TC1 can be mapped to one or more categories of data, such as categories 1–N. TC2 can be mapped to category N+1, and so on. In some cases, a TC will not be mapped to any categories or classes of service. In certain implementations, the registry 320 of FIG. 4 includes additional registers per region of the cache. Each region of the cache may be able to store more than 1 piece of data at a time. In the example below, the cache is said to have N cache ways (e.g., an N-Way Set Associative Cache).

[0057] In certain implementations, the registers store a CLOS waymask for IO transactions. A waymask is a type of bitmask that identifies which cache way the piece of data can be stored in and how much of the cache region that data is allowed to use. Table 1 shows an example of how the CLOS can map to a CLOS waymask for the cache.

Table 1. Class of Service Waymask Registry

	Class of Service (bitmask)							
Cache Ways	1	2	3	4	5	6	7	8
1	1 bit	1 bit	1 bit	1 bit	1 bit	1 bit	1 bit	1 bit
2	2 bits	2 bits	2 bits	2 bits	2 bits	2 bits	2 bits	2 bits
3	3 bits	3 bits	3 bits	3 bits	3 bits	3 bits	3 bits	3 bits
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
N	N bits	N bits	N bits	N bits	N bits	N bits	N bits	N bits

[0058] Table 1 illustrates an example bitmask (waymask) for mapping the class of service (CLOS) to each of the available cache ways for N cache ways. As an example, if a traffic class T1 is mapped to a first CLOS (e.g., CLOS 1), then for a region of cache with 1 cache way, 1 bit of the cache is allocated for the data.

[0059] A register 320 is accessed by the cache controller 308 to select the CLOS for each traffic class. For example the cache controller 308 maps the TC to the CLOS as the cache controller 308 communicates with both a PCIe root complex 502 that tracks the traffic class of the PCIe transactions and the cache memory. The cache controller 308 also pushes data onto the cache ring 520, which means the cache controller 308 can set the CLOS. The cache controller 308 can check the traffic class of each IO transaction, and can set the CLOS for each transaction based on its traffic class. That allows the system agent or cache allocation manager 310 to allocate the IO traffic into its designated cache ways based on the class of service waymask selected based on its traffic class.

[0060] FIG. 6 is a process flow diagram 600 of selectively storing data directly into cache memory based on traffic class. A data packet is received and forwarded to the cache controller (602). The cache controller identifies the traffic class of the data packet (604). For example, the cache controller can parse the information in the packet header to extract the PCIe traffic class, which can then be used to identify whether the data should be stored in cache, and if so, where to store it and whether there is available cache for the amount of data at that traffic class. Such a packet header can be that as described in the PCIe Spec at § 2.2.6.6, the contents of which are incorporated herein in their entirety. In embodiments, the cache controller maps the traffic class to a category of data (606). For example, the cache controller maps the traffic class to a class of service (CLOS) or other categorization of cache allocation. The cache controller determines a region of cache associated with the category of data (608). In one example, the cache controller can use a CLOS waymask register to identify cache ways to allocate the data into the cache. In another example, the cache controller can use a bitmask to determine whether there is a cache way for the data and how much cache is available for the data. The cache controller can determine whether there are available cache ways for the traffic class (610). If the cache controller determines that there are available cache ways into the cache, then the cache controller can store the data into cache (612). If the cache controller determines that there are no available cache ways, then the cache controller can forward the data for storage in system memory (614).

[0061] Expanding on (610)–(614), the cache controller can determine whether there are available cache ways for a traffic class based on the mapping of the traffic class to the category of data or CLOS. The cache controller can use the bitmask or a waymask to identify where in

the cache the data should be stored. A higher or more important traffic class may be mapped to a category of data or CLOS with more than one cache way and/or to cache ways that support larger amounts of data. A lower or less important traffic class may be mapped to a category of data or CLOS with fewer cache ways and/or to cache ways that support less data. In some cases, data with a lower or less important traffic class may not get stored in cache and may get directed to system memory.

[0062] The cache ways for a particular category of data or CLOS may be occupied. In certain embodiments, data in cache can be evicted to make room for incoming data. For example, low priority data can be evicted to make room in cache for higher priority data.

[0063] The following examples pertain to embodiments in accordance with this Specification. One or more embodiments may include an apparatus, a system, a machine readable storage, a machine readable medium, a method, and hardware- and/or software-based logic (e.g., implemented in connection with a shared memory controller) to determine a traffic class of a data packet received over a link compatible with a PCIe link interconnect and to selectively store the data packet into cache memory or into system memory based on the traffic class.

[0064] One or more embodiments may include a system that includes a cache memory, a system memory, a network interface controller circuit to receive a data packet from a peripheral device across a peripheral component interface express (PCIe) protocol, and cache controller circuit to receive the data packet from the network interface controller, to determine a traffic class of the data packet, and to selectively store the data packet directly into the cache memory or into the system memory based on the traffic class of the data packet.

[0065] One or more embodiments includes a method performed at a cache controller, the method including receiving a data packet, determining a traffic class of the data packet, and selectively storing the data packet directly into cache memory or into system memory based on the traffic class of the data packet.

[0066] One or more embodiments includes a computer program product tangibly embodied on a non-transitory computer-readable medium, the computer program product comprising instructions operable when executed to to receive a data packet, to determine a traffic class of the data packet, and to selectively store the data packet directly into cache memory or into main memory based on the traffic class of the data packet.

[0067] In one example, an apparatus includes circuitry to receive a data packet, to determine a traffic class of the data packet, and to selectively store the data packet directly into cache memory or into main memory based on the traffic class of the data packet.

[0068] In one example, the circuitry is to receive the data packet across a Peripheral Component Interface Express (PCIe) link.

[0069] In one example, the data packet includes a packet header, the packet header including a transaction descriptor field, the transaction descriptor field including the traffic class of the data packet.

[0070] In one example, the traffic class is a Transaction Layer Packet (TLP) label of the data packet.

[0071] In one example, the traffic class is used to differentiate data packets having a high priority traffic class from data packets having a low priority traffic class.

[0072] In one example, the circuitry maps the traffic class of the data packet to a corresponding category of data and selectively stores data of the data packet in the cache memory or in the main memory based on the category of data.

[0073] In one example, the circuitry maps the high priority traffic class to a high priority category of data and maps the low priority traffic class to a low priority category of data, and the circuitry selectively stores data of data packets with a traffic class mapped to high priority category of data into the cache memory and selectively stores data of data packets with a traffic class mapped to a low priority category of data to the main memory.

[0074] In one example, the circuitry maps the high priority traffic class to a high priority category of data and maps the low priority traffic class to a low priority category of data, and wherein the circuitry determines whether the traffic mapped to the high priority category of data can be allocated to a region of the cache memory and determines whether the traffic mapped to the low priority category of data can be allocated to a region of the cache memory.

[0075] In one example, the circuitry determines that the low priority category of data traffic cannot be allocated into cache and stores the low priority category of data traffic into the main memory.

[0076] In one example, the cache memory includes a plurality of cache ways, each cache way reserved for a category of data, and wherein the circuitry maps the traffic class of the data packet to a corresponding category of data and selectively stores the data packet in the cache memory or in the main memory based on the category of data by a cache way reserved for the category of data.

[0077] In one example, the circuitry comprises a cache controller.

[0078] In one example, the traffic class is used to differentiate data packets having a high priority traffic class from data packets having a low priority traffic class.

[0079] In one example, the circuitry maps the traffic class of the data packet to a class of service (CLOS) and selectively stores the data packet in the cache memory or in the system memory based on the class of service.

[0080] In one example, the circuitry maps the high priority traffic class to a high priority CLOS and maps the low priority traffic class to a low priority CLOS, and the circuitry selectively stores data of data packets with a traffic class mapped to high priority CLOS into the cache memory and selectively stores data of data packets with a traffic class mapped to a low priority CLOS to the system memory.

[0081] In one example, the circuitry maps the high priority traffic class to a high priority CLOS and maps the low priority traffic class to a low priority CLOS, and wherein the circuitry determines whether the traffic mapped to the high priority CLOS can be allocated to a region of the cache memory and determines whether the traffic mapped to the low priority CLOS can be allocated to a region of the cache memory.

[0082] In one example, the circuitry determines that the low priority CLOS traffic cannot be allocated into cache and stores the low priority CLOS traffic into the system memory.

[0083] In one example, the cache memory comprises a CLOS waymask register, the CLOS waymask register comprising bits corresponding to a traffic class mapping and used to identify ways of the cache memory for storing data packets having a traffic class mapped to a corresponding CLOS, and wherein each class of service corresponds to a waymask of the cache memory, and circuitry selectively stores the data packets into cache memory based on the waymask corresponding to the CLOS.

[0084] In one example, a computer program product tangibly embodied on a non-transitory computer-readable medium includes instructions operable when executed to receive a data packet, determine a traffic class of the data packet, and selectively store the data packet directly into cache memory or into system memory based on the traffic class of the data packet.

[0085] In one example, the data packet is received across a link compliant with a Peripheral Component Interface Express (PCIe)-based interconnect.

[0086] In one example, the data packet includes a packet header, the packet header including a transaction descriptor field, the transaction descriptor field including the traffic class of the data packet.

[0087] In one example, the traffic class is a Transaction Layer Packet (TLP) label of the data packet.

[0088] In one example, the traffic class is used to differentiate data packets having a high priority traffic class from data packets having a low priority traffic class.

[0089] In one example, the instructions when executed configured to map the traffic class of the data packet to a corresponding category of data and selectively stores the data packet in the cache memory or in the system memory based on the category of data.

[0090] In one example, the instructions when executed configured to map the high priority traffic class to a high priority category of data and map the low priority traffic class to a low priority category of data, and selectively store data packets with a traffic class mapped to high priority category of data into the cache memory and selectively store data packets with a traffic class mapped to a low priority category of data to the system memory.

[0091] In one example, the instructions when executed configured to map the high priority traffic class to a high priority category of data and map the low priority traffic class to a low priority category of data, and determine whether the traffic mapped to the high priority category of data can be allocated to a region of the cache memory and determine whether the traffic mapped to the low priority category of data can be allocated to a region of the cache memory.

[0092] In one example, the instructions when executed configured to determine that the low priority category of data traffic cannot be allocated into cache and store the low priority category of data traffic into the system memory.

[0093] In one example, the instructions when executed configured to map the traffic class of the data packet to a corresponding category of data, identify a cache region of the cache memory associated with the category of data, and selectively store the data packet in the cache memory or in the system memory based on the category of data by a cache way reserved for the category of data.

[0094] In one example, a non-transient computer readable medium containing program instructions for causing a computer to receive the data packet from the network interface controller, determine a traffic class of the data packet, selectively store the data packet directly into the cache memory or into the system memory based on the traffic class of the data packet.

[0095] In one example, an apparatus includes a means to perform a method as described in the preceding paragraphs.

[0096] In one example, a machine-readable storage includes machine-readable instructions that, when executed, implement a method or realize an apparatus as described above.

[0097] Turning to FIG. 7 an embodiment of a layered protocol stack is illustrated. Layered protocol stack 700 includes any form of a layered communication stack, such as a Quick Path Interconnect (QPI) stack, a PCie stack, a next generation high performance computing interconnect stack, or other layered stack. Although the discussion immediately

below is in relation to a PCIe stack, the same concepts may be applied to other interconnect stacks. In one embodiment, protocol stack 700 is a PCIe protocol stack including transaction layer 705, link layer 710, and physical layer 720. An interface, such as interfaces 117, 118, 121, 122, 126, and 131 in FIG. 1, may be represented as communication protocol stack 700. Representation as a communication protocol stack may also be referred to as a module or interface implementing/including a protocol stack.

[0098] PCI Express uses packets to communicate information between components. Packets are formed in the Transaction Layer 705 and Data Link Layer 710 to carry the information from the transmitting component to the receiving component. As the transmitted packets flow through the other layers, they are extended with additional information necessary to handle packets at those layers. At the receiving side the reverse process occurs and packets get transformed from their Physical Layer 720 representation to the Data Link Layer 710 representation and finally (for Transaction Layer Packets) to the form that can be processed by the Transaction Layer 705 of the receiving device.

Transaction Layer

[0099] In one embodiment, transaction layer 705 is to provide an interface between a device's processing core and the interconnect architecture, such as data link layer 710 and physical layer 720. In this regard, a primary responsibility of the transaction layer 705 is the assembly and disassembly of packets (i.e., transaction layer packets, or TLPs). The translation layer 705 typically manages credit-base flow control for TLPs. PCIe implements split transactions, i.e. transactions with request and response separated by time, allowing a link to carry other traffic while the target device gathers data for the response.

[00100] In addition PCIe utilizes credit-based flow control. In this scheme, a device advertises an initial amount of credit for each of the receive buffers in Transaction Layer 705. An external device at the opposite end of the link, such as controller hub 115 in FIG. 1, counts the number of credits consumed by each TLP. A transaction may be transmitted if the transaction does not exceed a credit limit. Upon receiving a response an amount of credit is restored. An advantage of a credit scheme is that the latency of credit return does not affect performance, provided that the credit limit is not encountered.

[00101] In one embodiment, four transaction address spaces include a configuration address space, a memory address space, an input/output address space, and a message address space. Memory space transactions include one or more of read requests and write requests to transfer data to/from a memory-mapped location. In one embodiment, memory space transactions are capable of using two different address formats, e.g., a short address format, such as a 32-bit address, or a long address format, such as 64-bit address. Configuration

space transactions are used to access configuration space of the PCIe devices. Transactions to the configuration space include read requests and write requests. Message space transactions (or, simply messages) are defined to support in-band communication between PCIe agents.

[00102] Therefore, in one embodiment, transaction layer 705 assembles packet header/payload 706. Format for current packet headers/payloads may be found in the PCIe Spec.

[00103] Quickly referring to FIG. 8, an embodiment of a PCIe transaction descriptor is illustrated. In one embodiment, transaction descriptor 800 is a mechanism for carrying transaction information. In this regard, transaction descriptor 800 supports identification of transactions in a system. Other potential uses include tracking modifications of default transaction ordering and association of transaction with channels.

[00104] Transaction descriptor 800 includes global identifier field 802, attributes field 804 and channel identifier field 806. In the illustrated example, global identifier field 802 is depicted comprising local transaction identifier field 808 and source identifier field 810. In one embodiment, global transaction identifier 802 is unique for all outstanding requests.

[00105] According to one implementation, local transaction identifier field 808 is a field generated by a requesting agent, and it is unique for all outstanding requests that require a completion for that requesting agent. Furthermore, in this example, source identifier 810 uniquely identifies the requestor agent within a PCIe hierarchy. Accordingly, together with source ID 810, local transaction identifier 808 field provides global identification of a transaction within a hierarchy domain.

[00106] Attributes field 804 specifies characteristics and relationships of the transaction. In this regard, attributes field 804 is potentially used to provide additional information that allows modification of the default handling of transactions. In one embodiment, attributes field 804 includes priority field 812, reserved field 814, ordering field 816, and no-snoop field 818. Here, priority sub-field 812 may be modified by an initiator to assign a priority to the transaction. Reserved attribute field 814 is left reserved for future, or vendor-defined usage. Possible usage models using priority or security attributes may be implemented using the reserved attribute field.

[00107] In this example, ordering attribute field 816 is used to supply optional information conveying the type of ordering that may modify default ordering rules. According to one example implementation, an ordering attribute of "0" denotes default ordering rules are to apply, wherein an ordering attribute of "1" denotes relaxed ordering, wherein writes can pass writes in the same direction, and read completions can pass writes in the same direction. Snoop

attribute field 818 is utilized to determine if transactions are snooped. As shown, channel ID Field 806 identifies a channel that a transaction is associated with.

Link Layer

[00108] Link layer 710, also referred to as data link layer 710, acts as an intermediate stage between transaction layer 705 and the physical layer 720. In one embodiment, a responsibility of the data link layer 710 is providing a reliable mechanism for exchanging Transaction Layer Packets (TLPs) between two components a link. One side of the Data Link Layer 710 accepts TLPs assembled by the Transaction Layer 705, applies packet sequence identifier 711, i.e. an identification number or packet number, calculates and applies an error detection code, i.e. CRC 712, and submits the modified TLPs to the Physical Layer 720 for transmission across a physical to an external device.

Physical Layer

[00109] In one embodiment, physical layer 720 includes logical sub block 721 and electrical sub-block 722 to physically transmit a packet to an external device. Here, logical sub-block 721 is responsible for the "digital" functions of Physical Layer 721. In this regard, the logical sub-block includes a transmit section to prepare outgoing information for transmission by physical sub-block 722, and a receiver section to identify and prepare received information before passing it to the Link Layer 710.

[00110] Physical block 722 includes a transmitter and a receiver. The transmitter is supplied by logical sub-block 721 with symbols, which the transmitter serializes and transmits onto to an external device. The receiver is supplied with serialized symbols from an external device and transforms the received signals into a bit-stream. The bit-stream is de-serialized and supplied to logical sub-block 721. In one embodiment, an 8b/10b transmission code is employed, where ten-bit symbols are transmitted/received. Here, special symbols are used to frame a packet with frames 723. In addition, in one example, the receiver also provides a symbol clock recovered from the incoming serial stream.

[00111] As stated above, although transaction layer 705, link layer 710, and physical layer 720 are discussed in reference to a specific embodiment of a PCIe protocol stack, a layered protocol stack is not so limited. In fact, any layered protocol may be included/implemented. As an example, an port/interface that is represented as a layered protocol includes: (1) a first layer to assemble packets, i.e. a transaction layer; a second layer to sequence packets, i.e. a link layer; and a third layer to transmit the packets, i.e. a physical layer. As a specific example, a common standard interface (CSI) layered protocol is utilized.

[00112] Referring next to FIG. 9, an embodiment of a PCIe serial point to point fabric is illustrated. Although an embodiment of a PCIe serial point-to-point link is illustrated, a

serial point-to-point link is not so limited, as it includes any transmission path for transmitting serial data. In the embodiment shown, a basic PCIe link includes two, low-voltage, differentially driven signal pairs: a transmit pair 906/911 and a receive pair 912/907. Accordingly, device 905 includes transmission logic 906 to transmit data to device 910 and receiving logic 907 to receive data from device 910. In other words, two transmitting paths, i.e. paths 916 and 917, and two receiving paths, i.e. paths 918 and 919, are included in a PCIe link.

[00113] A transmission path refers to any path for transmitting data, such as a transmission line, a copper line, an optical line, a wireless communication channel, an infrared communication link, or other communication path. A connection between two devices, such as device 905 and device 910, is referred to as a link, such as link 415. A link may support one lane – each lane representing a set of differential signal pairs (one pair for transmission, one pair for reception). To scale bandwidth, a link may aggregate multiple lanes denoted by xN, where N is any supported Link width, such as 1, 2, 4, 8, 12, 16, 32, 64, or wider.

[00114] A differential pair refers to two transmission paths, such as lines 416 and 417, to transmit differential signals. As an example, when line 416 toggles from a low voltage level to a high voltage level, i.e. a rising edge, line 417 drives from a high logic level to a low logic level, i.e. a falling edge. Differential signals potentially demonstrate better electrical characteristics, such as better signal integrity, i.e. cross-coupling, voltage overshoot/undershoot, ringing, etc. This allows for better timing window, which enables faster transmission frequencies.

[00115] In one embodiment, a new High Performance Interconnect (HPI) is provided. HPI is a next-generation cache-coherent, link-based interconnect. As one example, HPI may be utilized in high performance computing platforms, such as workstations or servers, where PCIe is typically used to connect accelerators or I/O devices. However, HPI is not so limited. Instead, HPI may be utilized in any of the systems or platforms described herein. Furthermore, the individual ideas developed may be applied to other interconnects, such as PCIe. Moreover, HPI may be extended to compete in the same market as other interconnect, such as PCIe. To support multiple devices, in one implementation, HPI includes a Instruction Set Architecture (ISA) agnostic (i.e. HPI is able to be implemented in multiple different devices). In another scenario, HPI may also be utilized to connect high performance I/O devices, not just processors or accelerators. For example, a high performance PCIe device may be coupled to HPI through an appropriate translation bridge (i.e. HPI to PCIe). Moreover, the HPI links may be utilized many HPI based devices, such as processors, in various ways (e.g. stars, rings, meshes, etc.). FIG. 10 illustrates an embodiment of multiple potential multi-socket configurations. A two-socket configuration 1005, as depicted, includes two HPI links; however, in other

implementations, one HPI link may be utilized. For larger topologies, any configuration may be utilized as long as an ID is assignable and there is some form of virtual path. As shown 4 socket configuration 1010 has an HPI link from each processor to another. But in the 8 socket implementation shown in configuration 1015, not every socket is directly connected to each other through an HPI link. However, if a virtual path exists between the processors, the configuration is supported. A range of supported processors includes 2-32 in a native domain. Higher number of processors may be reached through use of multiple domains or other interconnects between node controllers.

[00116] The HPI architecture includes a definition of a layered protocol architecture, which is similar to PCIe in that it also includes a layered protocol architecture. In one embodiment, HPI defines protocol layers (coherent, non-coherent, and optionally other memory based protocols), a routing layer, a link layer, and a physical layer. Furthermore, as many other interconnect architecture's HPI includes enhancements related to power managers, design for test and debug (DFT), fault handling, registers, security, etc.

[00117] FIG. 11 illustrates an embodiment of potential layers in the HPI layered protocol stack; however, these layers are not required and may be optional in some implementations. Each layer deals with its own level of granularity or quantum of information (the protocol layer 1105a,b with packets 1130, link layer 1110a,b with flits 1135, and physical layer 1105a,b with phits 1140). Note that a packet, in some embodiments, may include partial flits, a single flit, or multiple flits based on the implementation.

[00118] As a first example, a width of a phit 1140 includes a 1 to 1 mapping of link width to bits (e.g. 20 bit link width includes a phit of 20 bits, etc.). Flits may have a greater size, such as 184, 192, or 200 bits. Note that if phit 1140 is 20 bits wide and the size of flit 1135 is 184 bits then it takes a fractional number of phits 1140 to transmit one flit 1135 (e.g. 9.2 phits at 20 bits to transmit an 184 bit flit 1135 or 9.6 at 20 bits to transmit a 192 bit flit). Note that widths of the fundamental link at the physical layer may vary. For example, the number of lanes per direction may include 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, etc. In one embodiment, link layer 1110a,b is capable of embedding multiple pieces of different transactions in a single flit, and within the flit multiple headers (e.g. 1, 2, 3, 4) may be embedded within the flit. Here, HPI splits the headers into corresponding slots to enable multiple messages in the flit destined for different nodes.

[00119] Physical layer 1105a,b, in one embodiment, is responsible for the fast transfer of information on the physical medium (electrical or optical etc.). The physical link is point to point between two Link layer entities, such as layer 1105a and 1105b. The Link layer 1110a,b abstracts the Physical layer 1105a,b from the upper layers and provides the capability to reliably

transfer data (as well as requests) and manage flow control between two directly connected entities. It also is responsible for virtualizing the physical channel into multiple virtual channels and message classes. The Protocol layer 1120a,b relies on the Link layer 1110a,b to map protocol messages into the appropriate message classes and virtual channels before handing them to the Physical layer 1105a,b for transfer across the physical links. Link layer 1110a,b may support multiple messages, such as a request, snoop, response, writeback, non-coherent data, etc.

[00120] In one embodiment, to provide reliable transmission cyclic redundancy check (CRC) error checking and recovery procedures are provided by the Link layer 1110a,b in order to isolate the effects of routine bit errors that occur on the physical interconnect. The Link layer 1110a generates the CRC at the transmitter and checks at the receiver Link layer 1110b.

[00121] In some implementations, link layer 1110a,b utilized a credit scheme for flow control. During initialization, a sender is given a set number of credits to send packets or flits to a receiver. Whenever a packet or flit is sent to the receiver, the sender decrements its credit counters by one credit which represents either a packet or a flit, depending on the type of virtual network being used. Whenever a buffer is freed at the receiver, a credit is returned back to the sender for that buffer type. When the sender's credits for a given channel have been exhausted, in one embodiment, it stops sending any flits in that channel. Essentially, credits are returned after the receiver has consumed the information and freed the appropriate buffers.

[00122] In one embodiment, routing layer 1115a,b provides a flexible and distributed way to route packets from a source to a destination. In some platform types (for example, uniprocessor and dual processor systems), this layer may not be explicit but could be part of the Link layer 1110a,b; in such a case, this layer is optional. It relies on the virtual network and message class abstraction provided by the Link Layer 1110a,b as part of the function to determine how to route the packets. The routing function, in one implementation, is defined through implementation specific routing tables. Such a definition allows a variety of usage models.

[00123] In one embodiment, protocol layer 1120a,b implement the communication protocols, ordering rule, and coherency maintenance, I/O, interrupts, and other higher-level communication. Note that protocol layer 1120a,b, in one implementation provides messages to negotiate power states for components and the system. As a potential addition, physical layer 1105a,b may also independently or in conjunction set power states of the individual links.

[00124] Multiple agents may be connect to an HPI architecture, such as a home agent (orders requests to memory), caching (issues requests to coherent memory and responds to snoops), configuration (deals with configuration transactions), interrupt (processes interrupts),

legacy (deals with legacy transactions), non-coherent (deals with non-coherent transactions), and others. More specific discussion of the layers for HPI are discussed below.

[00125] An overview of a few potential features of HPI includes: not utilizing preallocation at home nodes; no ordering requirements for a number of message classes; packing multiple messages in a single flit (protocol header) (i.e. a packed flit that can hold multiple messages in defined slots); a wide link that may scale from 4, 8, 111, 20, or more lanes; a large error checking scheme that may utilize 8, 16, 32, or as much as 64 bits for error protection; and utilizing an embedded clocking scheme.

HPI Physical Layer

[00126] The Physical layer 1105a,b (or PHY) of HPI rests above the electrical layer (i.e. electrical conductors connecting two components) and below the link layer 1110a,b, as illustrated in FIG. 11. The physical layer resides on each agent and connects the link layers on two agents (A and B) separated from each other. The local and remote electrical layers are connected by physical media (e.g. wires, conductors, optical, etc.). The physical layer 1105a,b, in one embodiment, has two major phases, initialization and operation. During initialization, the connection is opaque to the link layer and signaling may involve a combination of timed states and handshake events. During operation, the connection is transparent to the link layer and signaling is at a speed, with all lanes operating together as a single link. During the operation phase, the physical layer transports flits from agent A to agent B and from agent B to agent A. The connection is also referred to as a link and abstracts some physical aspects including media, width and speed from the link layers while exchanging flits and control/status of current configuration (e.g. width) with the link layer. The initialization phase includes minor phases e.g. Polling, Configuration. The operation phase also includes minor phases (e.g. link power management states).

[00127] In one embodiment, physical layer 1105a,b is also: to meet a reliability/error standard, tolerate a failure of a lane on a link and go to a fraction of nominal width, tolerate single failures in opposite direction of a link, support hot add/remove, enabling/disabling PHY ports, timeout initialization attempts when the number of attempts has exceeded a specified threshold etc.

[00128] In one embodiment, HPI utilizes a rotating bit pattern. For example when a flit size is not aligned with a multiple of the lanes in an HPI link, the flit may not be able to be sent in an integer multiple of transmissions over the lanes (e.g. a 192-bit flit is not a clean multiple of an exemplary 20 lane link. So at x20 flits may be interleaved to avoid wasting bandwidth (i.e. sending a partial flit at some point without utilizing the rest of the lanes). The interleaving, in one embodiment, is determined to optimize latency of key fields and multiplexers in the

transmitter (Tx) and receiver (Rx). The determined patterning also potentially provides for clean and quick transitioning to/from a smaller width (e.g x8) and seamless operation at the new width.

[00129] In one embodiment, HPI utilizes an embedded clock, such as a 20 bit embedded clock or other number of bit embedded clock. Other high performance interfaces may use a forwarded clock or other clock for inband reset. By embedding clock in HPI, it potentially reduces pinout. However, using an embedded clock, in some implementations, may result in different apparatus and methods to handle inband reset. As a first, example a blocking link state to hold off link flit transmission and allow PHY usage (described in more detail in the Appendix A) is utilized after initialization. As a second example, electrical ordered sets, such as an electrically idle ordered set (EIOS) may be utilized during initialization.

[00130] In one embodiment, HPI is capable of utilizing a first bit width direction without a forwarded clock and a second, smaller bit width link for power management. As an example, HPI includes a partial link width transmitting state, where a partial width is utilized (e.g. a x20 full width and a x8 partial width); however, the widths are purely illustrative and may differ. Here, the PHY may handle partial width power management without link layer assist or intervention. In one embodiment, a blocking link state (BLS) protocol is utilized to enter the partial width transmitting state (PWTS). PWTS exit, in one or more implementations, may use the BLS protocol or squelch break detection. Due to absence of a forwarded clock, PWTLS exit may include a re-deskew, which maintains determinism of the link.

[00131] In one embodiment, HPI utilizes Tx adaptation. As an example, loopback state and hardware is used for Tx Adaptation. As on example, HPI is capable of counting actual bit errors; this may be able to be performed by injecting specialized patterns. As a result, HPI should be able to get better electrical margins at lower power. When using the loopback state, one direction may be used as a hardware backchannel with metrics sent as part of a training sequence (TS) payload.

[00132] In one embodiment, HPI is able to provide latency fixing without exchanging sync counter values in a TS. Other interconnect may perform latency fixing based on such exchanging of a sync counter value in each TS. Here, HPI may utilize periodically recurring Electrically Idle Exit Ordered Sets (EIEOS) as a proxy for the sync counter value by aligning the EIEOS to the sync counter. This potentially saves TS payload space, removes aliasing, and DC balance concerns, as well as simplify the calculation of latency to be added.

[00133] In one embodiment, HPI provides for software and timer control of a link state machine transitions. Other interconnect may support a semaphore (hold bit) that is set by hardware on entering an initialization state. Exit from the state occurs when the hold bit is

cleared by software. HPI, in one implementation, allows software to control this type of mechanism for entering a transmitting link state or a loop back pattern state. In one embodiment, HPI allows for exit from handshake sates to be based on a software programmable timeout after handshake, which potentially makes test software easier.

[00134] In one embodiment, HPI utilizes Pseudo Random Bit Sequence (PRBS) scrambling of TS. As an example a 23-bit PRBS is utilized (PRBS23). In one embodiment, the PRBS is generated by a similar bit size, self-seeded storage element, such as a linear feedback shift register. As one example, a fixed UI pattern may be utilized to scramble with a bypass to an adaptation state. But by scrambling TS with PRBS23, Rx adaptation may be performed without the bypass. In addition, offset and other errors may be reduced during clock recovery and sampling. The HPI approach relies on using Fibonacci LFSRs which can be self seeded during specific portions of the TS.

[00135] In one embodiment, HPI supports an emulated slow mode without changing PLL clock frequency. Some designs may use separate PLLs for slow and fast speed. Yet, in on implementation, HPI use emulated slow mode (i.e. PLL clock runs at fast speed; TX repeats bits multiple times; RX oversamples to locate edges and identify the bit.). This means that ports sharing a PLL may coexist at slow and fast speeds. In one example where the multiple is an integer ratio of fast speed to slow speed, different fast speeds may work with the same slow speed, which may be used during the discovery phase of hot attach.

[00136] In one embodiment, HPI supports a common slow mode frequency for hot attach. Emulated slow mode, as described above, allows HPI ports sharing a PLL to coexist at slow and fast speeds. When a designer sets the emulation multiple as an integer ratio of fast speed to slow speed, then different fast speeds may work with the same slow speed. So, two agents which support at least one common frequency may be hot attached irrespective of the speed at which the host port is running. Software discovery may then use the slow mode link to identify and setup the most optimal link speeds.

[00137] In one embodiment, HPI supports re-initialization of link without termination changes. One could provide re-initialization on an inband reset having clock lane terminations changed for the discovery process used in reliability, availability, and serviceability (RAS). In one embodiment, re-initialization for HPI may be done without changing the termination values when HPI includes a RX screening of incoming signaling to identify good lanes.

[00138] In one embodiment, HPI supports robust low power link state (LPLS) entry. As an example, HPI may include a minimum stay in LPLS (i.e. a minimum amount of time, UI, counter value, etc that a link stays in LPLS before an exit). Alternatively, LPLS entry may be negotiated and then use an inband reset to enter LPLS. But this may mask an actual inband reset

originating from the second agent in some cases. HPI, in some implementations, allows a first agent to enter LPLS and a second agent to enter Reset. The first agent is unresponsive for a time period (i.e. the minimum stay), which allows the second agent to complete reset and then wake up the first agent, enabling a much more efficient, robust entry into LPLS.

[00139] In one embodiment, HPI supports features such as debouncing detect, wake and continuous screening for lane failures. HPI may look for a specified signaling pattern for an extended period of time to detect a valid wake from a LPLS thus reducing the chances of a spurious wake. The same hardware may also be used in the background for continuously screening for bad lanes during the initialization process making for a more robust RAS feature.

[00140] In one embodiment, HPI supports a deterministic exit for lock step and restart-replay. In HPI, some TS boundaries may coincide with flit boundaries when operating at full width. So HPI may identify and specify the exit boundaries such that lock-step behavior may be maintained with another link. In addition, HPI may specify timers which may be used to maintain lock step with a link pair. After initialization, HPI may also support operation with inband resets disabled to support some flavors of lock-step operation.

[00141] In one embodiment, HPI supports use of TS header instead of payload for key initialization parameters. Alternatively, TS payload may be used to exchange init parameters like ACKs and lane numbers. And DC levels for communicating lane polarity may also be used. Yet, HPI may use DC balanced codes in the TS header for key parameters. This potentially reduces the number of bytes needed for a payload and potentially allows for an entire PRBS23 pattern to be used for scrambling TS, which reduces the need for DC balancing the TS.

[00142] In one embodiment, HPI supports measures to increase noise immunity of active lanes during partial width transmitting link state (PWTLS) entry/exit of idle lanes. In one embodiment, null (or other non-retryable flits) flits may be used around the width change point to increase noise immunity of active lanes. Additionally, HPI may utilize null flits around the start of PWTLS exit (i.e. the null flits may be broken up with data flits). HPI may also use specialized signaling, whose format may be varied to reduce chances of false wake detects.

[00143] In one embodiment, HPI supports use of specialized patterns during PWTLS exit to allow non-blocking deskew. Alternatively, idle lanes may not be deskewed on PWTLS exit since they may maintain skew with help of a forwarded clock. Yet, with use of an embedded clock, HPI may use specialized signaling, whose format may be varied to reduce chances of false wake detects and also allow for deskew without blocking flit flow. This also allows for more robust RAS by seamlessly powering down failing lanes, re-adapting them, and bringing them back online without blocking the flow of flits.

[00144] In one embodiment, HPI supports low power link state (LPLS) entry without link layer support and more robust LPLS exit. Alternatively, link layer negotiation may be depended on between pre-designated master and slave to enter LPLS from transmitting link sate (TLS). In HPI, the PHY may handle negotiation using blocking link state (BLS) codes and may support both agents being masters or initiators, as well as entry into LPLS directly from PWTLS. Exit from LPLS may be based on debouncing a squelch break using a specific pattern followed by handshake between the two sides and a timeout induced inband reset if any of this fails.

[00145] In one embodiment, HPI supports controlling unproductive looping during initialization. Alternatively, a failure to init (e.g. lack of good lanes) may result in retrying the init too many times, which potentially wastes power and is difficult to debug. In HPI, the link-pair may try to init a set number of times before calling it quits and powering down in a reset state, where software may make adjustments before retrying the init. This potentially improves the RAS of the system.

[00146] In one embodiment, HPI supports advanced IBIST (interconnect built in self test) options. In one embodiment, a pattern generator may be utilized, which allows for two non-correlated PRBS23 patterns of maximum length for any pin. In one embodiment, HPI may be able to support four such patterns, as well as provide the ability to control the length of these patterns (i.e. dynamically vary test pattern, PRBS23 length).

[00147] In one embodiment, HPI provides advanced logic to deskew lanes. As an example, the TS boundary after TS lock may be used to deskew the lanes. In addition, HPI may deskew by comparing lane PRBS patterns in the LFSR during specific points in the payload. Such deskew might be useful in testchips, which may lack ability to detect TS or state machines to manage the deskew.

[00148] In one embodiment, exit from init to link transmitting occurs on a TS boundary with planetary alignment. In addition, HPI may support a negotiated delay from that point. In addition, the order of exit between the two directions may be controlled by using master-slave determinism allowing for one instead of two planetary alignment controls for the link pair.

[00149] Some implementations use a fixed 128UI pattern to scramble TS. Others use a fixed 4k PRBS23 to scramble TS. HPI, in one embodiment, allows for using any length PRBS including an entire (8M-1) PRBS23 sequence.

[00150] In some architectures adpatation is of fixed duration. In one embodiment, the exit from Adapt is handshaked rather than timed. This means that Adapt times may be asymmetric between the two directions and as long as needed by either side.

[00151] In one embodiment, a state machine may bypass states if those state actions don't need to be redone. However, this may lead to more complex designs and validation escapes.

HPI doesn't use bypasses - instead it distributes actions such that short timers in each state may be used to perform the actions and bypasses avoided. This potentially makes for more uniform and synchronized state machine transitions.

[00152] In some architectures, forwarded clock is utilized for Inband reset and link layer for staging partial width transmitting and for low power link entry. HPI uses block linking state codes similar functions. These codes potentially could have bit errors leading to 'mismatches' at Rx. HPI includes a protocol for dealing with mismatches as well as means to handle asynchronous reset, low power link state and partial width link state requests.

[00153] In one embodiment, a 128 UI scrambler is utilized for loopback TS. However, this can lead to aliasing for TS lock when loopback begins; so some architecture's changes the payload to all 0s during this. In another embodiment, HPI utilizes a uniform payload and uses the periodically occurring unscrambled EIEOS for TS lock.

[00154] Some architecture utilize scrambled TS during init. In one embodiment, HPI defines supersequences that are combinations of scrambled TS of various lengths and unscrambled EIEOS. This allows more randomized transitions during init and also simplifies TS lock, latency fixing, and other actions.

HPI Link Layer

[00155] Returning to FIG. 11, an embodiment of a logical block for link layer 1110a,b is illustrated. In one embodiment, Link Layer 1110a,b guarantees reliable data transfer between two protocol or routing entities. It abstracts Physical layer 1105a,b from the Protocol layer 1120a,b, is responsible for the flow control between two protocol agents (A, B), and provides virtual channel services to the Protocol layer (Message Classes) and Routing layer (Virtual Networks). The interface between the Protocol layer 1120a,b and the Link Layer 1110a,b is typically at the packet level. In one embodiment, the smallest transfer unit at the Link Layer is referred to as a flit which a specified number of bits, such as 192. The Link Layer 1110a,b relies on the Physical layer 1105a,b to frame the Physical layer 1105a,b's unit of transfer (phit) into the Link Layer 1110a,b' unit of transfer (flit). In addition, the Link Layer 1110a,b may be logically broken into two parts, a sender and a receiver. A sender/receiver pair on one entity may be connected to a receiver/sender pair on another entity. Flow Control is often performed on both a flit and a packet basis. Error detection and correction is also potentially performed on a flit level basis.

[00156] In one embodiment, flits are expanded 192 bits. However, any range of bits, such as 81-256 (or more) may be utilized in different variations. Here, the CRC field is also increased (e.g. 16 bits) to handle a larger payload.

[00157] In one embodiment, TIDs (Transaction IDs) are 11 bits in length. As a result, pre-allocation and the enabling of distributed home agents may be removed. Furthermore, use of 11 bits, in some implementations, allows for the TID to be used without having use for an extended TID mode.

[00158] In one embodiment, header flits are divided into 3 slots, 2 with equal size (Slots 0 and 1) and another smaller slot (Slot 2). A floating field may be available for one of Slot 0 or 1 to use. The messages that can use slot 1 and 2 are optimized, reducing the number of bits needed to encode these slots' opcodes. When a header needing more bits that Slot 0 provides enters the link layer, slotting algorithms are in place to allow it to take over Slot 1 payload bits for additional space. Special control (e.g. LLCTRL) flits may consume all 3 slots worth of bits for their needs. Slotting algorithms may also exist to allow individual slots to be utilized while other slots carry no information, for cases where the link is partially busy. Other interconnect may allow a single message per flit, instead of multiple. The sizing of the slots within the flit, and the types of messages that can be placed in each slot, potentially provide the increased bandwidth of HPI even with a reduced flit rate. For a more detailed description of flits and the multi-slot header, refer to the flit definition section of Appendix B.

[00159] In HPI, a large CRC baseline may improve error detection. For example, a 16 bit CRC is utilized. As a result of the larger CRC, a larger payload may also be utilized. The 16 bits of CRC in combination with a polynomial used with those bits improves error detection. As an example, the are a minimum number of gates to provide 1) 1-4 bit errors detected 2) errors of burst length 16 or less are detected.

[00160] In one embodiment, a rolling CRC based on two CRC-16 equations is utilized. Two 16 bit polynomials may be used, the polynomial from HPI CRC-16 and a second polynomial. The second polynomial has the smallest number of gates to implement while retaining the properties of 1) all 1-7 bit errors detected 2) per lane burst protection in x8 link widths 3) all errors of burst length 16 or less are detected.

[00161] In one embodiment, a reduced max flit rate (9.6 versus 4 UI) is utilized, but increased throughput of the link is obtained. As a result of the increased flit size, introduction of multiple slots per flit, optimized utilization of payload bits (changed algorithms to remove or relocate infrequently used fields), more interconnect efficiency is achieved.

[00162] In one embodiment, part of the support for 3 slots includes 192 bit flit. The floating field enables 11 extra bits of payload for either slot 0 or slot 1. Note if a larger flit is

used more floating bits may be used. And as a corollary, if a smaller flit is used, then less floating bits are provided. By allowing a field to float between the two slots, we can provide the extra bits needed for certain messages, while still staying within 192 bits and maximizing the utilization of the bandwidth. Alternatively, providing an 11 bit HTID field to each slot may use an extra 11 bits in the flit which would not be as efficiently utilized.

[00163] Some interconnects may transmit Viral status in protocol level messages and Poison status in data flits. In one embodiment, HPI protocol level messages and Poison status are moved to control flits. Since these bits are infrequently used (only in the case of errors), removing them from the protocol level messages potentially increases flit utilization. Injecting them using control flits still allows containment of the errors.

[00164] In one embodiment, CRD and ACK bits in a flit allow return of a number of credits, such as eight, or the number of acks, such as 8. As part of the fully encoded credit fields, these bits are utilized as Credit[n] and Acknowledge[n] when Slot 2 is encoded as LLCRD. This potentially improves efficiency by allowing any flit to return the number of VNA Credits and the number of Acknowledges using a total of only 2 bits, but also allowing their definitions to remain consistent when a fully encoded LLCRD return is used.

[00165] In one embodiment, VNA vs. VN0/1 encoding (saves bits by aligning slots to same encoding). The slots in a multi-slot header flit may be aligned to just VNA, just VN0, or just VN1. By enforcing this, per slot bits indicating VN are removed. This increases the efficiency of flit bit utilization and potentially enables expanding from 10 bit TIDs to 11 bit TIDs.

[00166] Some fields only allow return in increments of 1 (for VN0/1), 2/8/16 (for VNA), and 8 (for Acknowledge). This means that returning a large number of pending Credits or Acknowledges may use multiple return messages. It also means that odd numbered return values for VNA and Acknowledge may be left stranded pending accumulation of an evenly divisible value. HPI may have fully encoded Credit and Ack return fields, allowing an agent to return all accumulated Credits or Acks for a pool with a single message. This potentially improves link efficiency and also potentially simplifies logic implementation (return logic can implement a "clear" signal rather than a full decrementer).

Routing Layer

[00167] In one embodiment, Routing layer 1115a,b provides a flexible and distributed method to route HPI transactions from a source to a destination. The scheme is flexible since routing algorithms for multiple topologies may be specified through programmable routing

tables at each router (the programming in one embodiment is performed by firmware, software, or a combination thereof). The routing functionality may be distributed; the routing may be done through a series of routing steps, with each routing step being defined through a lookup of a table at either the source, intermediate, or destination routers. The lookup at a source may be used to inject a HPI packet into the HPI fabric. The lookup at an intermediate router may be used to route an HPI packet from an input port to an output port. The lookup at a destination port may be used to target the destination HPI protocol agent. Note that the Routing layer, on some implementations, is thin since the routing tables, and, hence the routing algorithms, are not specifically defined by specification. This allows a variety of usage models, including flexible platform architectural topologies to be defined by the system implementation. The Routing layer 1115a,b relies on the Link layer 1110a,b for providing the use of up to three (or more) virtual networks (VNs) – in one example, two deadlock-free VNs, VN0 and VN1 with several message classes defined in each virtual network. A shared adaptive virtual network (VNA) may be defined in the link layer, but this adaptive network may not be exposed directly in Routing Concepts, since each Message class and VN may have dedicated resources and guaranteed forward progress.

[00168] A non-exhaustive, exemplary list of routing rules includes: (1) (Message class invariance): An incoming packet belonging to a particular message class may berouted on an outgoing HPI port/virtual network in the *same* message class; (2) (Switching) HPI platforms may support the "store-and-forward" and "virtual cut through" types of switching. In another embodiment, HPI may not support "wormhole" or "circuit" switching. (3) (Interconnect deadlock freedom) HPI platforms may not rely on adaptive flows for deadlock-free routing. With platforms, which use both VN0 and VN1, the 2 VNs together may be used for deadlock-free routing; and (4) (VN0 for "leaf" routers). In HPI platforms, which may use both VN0 and VN1, it is permissible to use *VN0* for those components, whose routers are not used route-through; that is, incoming ports have HPI destinations that terminate at this component. In such a case, packets from different VNs can be routed to VN0. Other rules (for example, movement of packets between VN0 and VN1) may be governed by a platform dependent routing algorithm.

[00169] Routing step: A routing step, in one embodiment, is referred to by a routing function (RF) and a selection function (SF). The routing function may take, as inputs, a HPI port at which a packet arrives and a destination NodeID; it then yields as output a 2-tuple – the HPI port number and the virtual network – which the packet should follow on its path to the destination. It is permitted for the routing function to be additionally dependent on the incoming virtual network. Further, it is permitted with the routing step to yield multiple <port#, virtual network> pairs. The resulting routing algorithms are called adaptive**.** In such a case, a selection

function SF may choose a single 2-tuple based on additional state information which the router has (for example, with adaptive routing algorithms, the choice of a particular port of virtual network may depend on the local congestion conditions). A routing step, in one embodiment, consists of applying the routing function and then the selection function to yield the 2-tuple(s).

[00170] Router Table Simplifications: HPI platforms may implement legal subsets of the virtual networks. Such subsets simplify the size of the routing table (reduce the number of columns) associated virtual channel buffering and arbitration at the router switch. These simplifications ay come at the cost of platform flexibility and features. VN0 and VN1 may be deadlock-free networks which provide deadlock freedom either together or singly, depending on the usage model, usually with minimal virtual channel resources assigned to them. Flat organization of the routing table may include a size corresponding to the maximum number of *NodeIDs*. With such an organization, the routing table may be indexed by the *destination NodeID* field and possibly by the *virtual network id* field. The table organization can also be made hierarchical with the destination NodeID field being sub-divided into multiple sub-fields, which is implementation dependent. For example, with a division into "local" and "non-local" parts, the "non-local" part of the routing is completed before the routing of the "local" part. The potential advantage of reducing the table size at every input port comes at the potential cost of being forced to assign NodeIDs to HPI components in a hierarchical manner

[00171] Routing Algorithm: A routing algorithm, in one embodiment, defines the set of permissible paths from a source module to a destination module. A particular path from the source to the destination is a subset of the permissible paths and is obtained as a series of routing steps defined above starting with the router at the source, passing through zero or more intermediate routers, and ending with the router at the destination. Note that even though an HPI fabric may have multiple physical paths from a source to a destination, the paths permitted are those defined by the routing algorithm.

HPI Coherence Protocol

[00172] In one embodiment, the HPI Coherence Protocol is included in layer 1120a,b is to support agents caching lines of data from memory. An agent wishing to cache memory data may use the coherence protocol to read the line of data to load into its cache. An agent wishing to modify a line of data in its cache may use the coherence protocol to acquire *ownership* of the line before modifying the data. After modifying a line, an agent may follow protocol requirements of keeping it in its cache until it either writes the line back to memory or includes the line in a response to an external request. Lastly, an agent may fulfill external requests to

invalidate a line in its cache. The protocol ensures coherency of the data by dictating the rules all caching agents may follow. It also provides the means for agents without caches to coherently read and write memory data.

[00173] Two conditions may be enforced to support transactions utilizing the HPI Coherence Protocol. First, the protocol maintains *data consistency*, as an example on a per-address basis, among data in agents' caches and between those data and the data in memory. Informally, data consistency may refer to each valid line of data in an agent's cache representing a most up-to-date value of the data and data transmitted in a coherence protocol packet represents the most up-to-date value of the data at the time it was sent. When no valid copy of the data exists in caches or in transmission, the protocol may ensure the most up-to-date value of the data resides in memory. Second, the protocol provides well-defined commitment points for requests. Commitment points for reads may indicate when the data is usable; and for writes they may indicate when the written data is *globally observable* and will be loaded by subsequent reads. The protocol may support these commitment points for both cacheable and uncacheable (UC) requests in the coherent memory space.

[00174] The HPI Coherence Protocol also may ensure the *forward progress* of coherence requests made by an agent to an address in the coherent memory space. Certainly, transactions may eventually be satisfied and retired for proper system operation. The HPI Coherence Protocol, in some embodiments, may have no notion of retry for resolving resource allocation conflicts. Thus, the protocol itself may be defined to contain no circular resource dependencies, and implementations may take care in their designs not to introduce dependencies that can result in deadlocks. Additionally, the protocol may indicate where designs are able to provide fair access to protocol resources.

[00175] Logically, the HPI Coherence Protocol, in one embodiment, consists of three items: coherence (or caching) agents, home agents, and the HPI interconnect fabric connecting the agents. Coherence agents and home agents work together to achieve data consistency by exchanging messages over the interconnect. The link layer 1110a,b and its related description provides the details of the interconnect fabric including how it adheres to the coherence protocol's requirements, discussed herein. (It may be noted that the division into coherence agents and home agents is for clarity. A design may contain multiple agents of both types within a socket or even combine agents behaviors into a single design unit.).

[00176] In one embodiment, HPI does not pre-allocate resources of a Home Agent. Here, a Receiving Agent receiving a request allocates resource to process it. An Agent sending a request allocates resources for responses. In this scenario, HPI may follow two general rules regarding resource allocation. First, an agent receiving a request may be responsible for

allocating the resource to process it. Second, an agent generating a request may be responsible for allocating resources to process responses to the request.

[00177] For allocation of resources may also extend to HTID (along with RNID/RTID) in snoop requests The potential reduction of using a home agent and forward responses to support responses to Home Agent (and data forwarding to requesting agent).

[00178] In one embodiment, home agent resources are also not pre-allocated in snoop requests and forward responses to support responses to the home agent (and data forwarding to the requesting agent.

[00179] In one embodiment, there is no pre-allocation of Home resources Ability to send CmpO "early", before Home Agent is finished processing request, when it is safe for requesting agent to reuse its RTID resource. General handling of snoops with similar RNID/RTID in system also part of protocol.

[00180] In one embodiment, conflict resolution is performed using an ordered response channel. A Coherence Agent uses RspCnflt as request for a Home Agent to send a FwdCnfltO, which will be ordered with the CmpO (if any already scheduled) for the Coherence Agent's conflicting request.

[00181] In one embodiment, HPI supports conflict resolution via an ordered response channel. A Coherence Agent using information from snoop to aid in processing FwdCnfltO, which has no "type" information and no RTID for forwarding data to requesting agent.

[00182] In one embodiment, a Coherence Agent blocks forwards for writeback requests to maintain data consistency. But it also allows Coherence Agent to use a writeback request to commit uncacheable (UC) data before processing forward and allows the Coherence Agent to writeback partial cache lines instead of protocol supporting a partial implicit writeback for forwards.

[00183] In one embodiment, a read invalidate (RdInv) request accepting Exclusive-state data is supported. Semantics of uncacheable (UC) reads include flushing modified data to memory. Some architecture, however, allowed forwarding M data to invalidating reads, which forced the requesting agent to clean the line if it received M data. The RdInv simplifies the flow but it does not allow E data to be forwarded.

[00184] In one embodiment, HPI support an InvItoM to IODC functionality. An InvItoM requests exclusive ownership of a cache line without receiving data and with the intent of performing a writeback soon afterward. A required cache state may be an M state, and E state, or either.

[00185] In one embodiment, HPI supports a WbFlush for persistent memory flush. An embodiment of a WbFlush is illustrated below. It may be sent as a result of a persistent commit. May flush write to persistent memory.

[00186] In one embodiment, HPI supports additional operations, such as SnpF for "fanout" snoops generated by the Routing Layer. Some architectures don't have explicit support for fanout snoops. Here, a HPI Home agent generates single "fanout" snoop request and, in response, the Routing Layer generates snoops to all peer agents in the "fanout cone". Home agent may expect snoop responses from each of the agent Sections.

[00187] In one embodiment, HPI supports additional operations, such as SnpF for "fanout" snoops generated by the Routing Layer. Some architectures don't have explicit support for fanout snoops. Here, a HPI Home agent generates single "fanout" snoop request and, in response, the Routing Layer generates snoops to all peer agents in the "fanout cone". Home agent may expect snoop responses from each of the agent Sections.

[00188] In one embodiment, HPI supports an explicit writeback with cache-push hint (WbPushMtoI). In one embodiment, a Coherence Agent writes back modified data with a hint to Home Agent that it may push the modified data to a "local" cache, storing in M state, without writing the data to memory.

[00189] In one embodiment, a Coherence Agent may keep F state when forwarding shared data. In on example, a Coherence Agent with F state that receives a "sharing" snoop or forward after such a snoop may keep the F state while sending S state to the requesting agent.

[00190] In one embodiment, protocol tables may be nested by having one table refer to another sub-table in the "next state" columns, and the nested table can have additional or finer-grained guards to specify which rows (behaviors) are permitted.

[00191] In one embodiment, Protocol tables use row spanning to indicate equally permissible behaviors (rows) instead of adding "Bias" bits to select among behaviors.

[00192] In one embodiment, action tables are organized for use as functionality engine for BFM (validation environment tool) rather than having BFM team create their own BFM engine based upon their interpretation.

HPI Non-Coherent Protocol

[00193] In one embodiment, HPI supports non-coherent transactions. As an example, a non-coherent transaction is referred to as one that does not participate in the HPI coherency protocol. Non-coherent transactions comprise requests and their corresponding completions. For some special transactions, a broadcast mechanism.

[00194] Note that the apparatus', methods', and systems described above may be implemented in any electronic device or system as aforementioned. As specific illustrations, the figures below provide exemplary systems for utilizing the invention as described herein. As the systems below are described in more detail, a number of different interconnects are disclosed, described, and revisited from the discussion above. And as is readily apparent, the advances described above may be applied to any of those interconnects, fabrics, or architectures.

[00195] Referring now to FIG. 12, shown is a block diagram of an embodiment of a multicore processor. As shown in the embodiment of FIG. 12, processor 1200 includes multiple domains. Specifically, a core domain 1230 includes a plurality of cores 1230A–1230N, a graphics domain 1260 includes one or more graphics engines having a media engine 1265, and a system agent domain 1210.

[00196] In various embodiments, system agent domain 1210 handles power control events and power management, such that individual units of domains 1230 and 1260 (e.g. cores and/or graphics engines) are independently controllable to dynamically operate at an appropriate power mode/level (e.g. active, turbo, sleep, hibernate, deep sleep, or other Advanced Configuration Power Interface like state) in light of the activity (or inactivity) occurring in the given unit. Each of domains 1230 and 1260 may operate at different voltage and/or power, and furthermore the individual units within the domains each potentially operate at an independent frequency and voltage. Note that while only shown with three domains, understand the scope of the present invention is not limited in this regard and additional domains may be present in other embodiments.

[00197] As shown, each core 1230 further includes low level caches in addition to various execution units and additional processing elements. Here, the various cores are coupled to each other and to a shared cache memory that is formed of a plurality of units or slices of a last level cache (LLC) 1240A-1240N; these LLCs often include storage and cache controller functionality and are shared amongst the cores, as well as potentially among the graphics engine too.

[00198] As seen, a ring interconnect 1250 couples the cores together, and provides interconnection between the core domain 1230, graphics domain 1260 and system agent circuitry 1210, via a plurality of ring stops 1252A-1252N, each at a coupling between a core and LLC slice. As seen in FIG. 12, interconnect 1250 is used to carry various information, including address information, data information, acknowledgement information, and snoop/invalid information. Although a ring interconnect is illustrated, any known on-die interconnect or fabric may be utilized. As an illustrative example, some of the fabrics discussed above (e.g. another on-die interconnect, Intel On-chip System Fabric (IOSF), an Advanced Microcontroller Bus

Architecture (AMBA) interconnect, a multi-dimensional mesh fabric, or other known interconnect architecture) may be utilized in a similar fashion.

[00199] As further depicted, system agent domain 1210 includes display engine 1212 which is to provide control of and an interface to an associated display. System agent domain 1210 may include other units, such as: an integrated memory controller 1220 that provides for an interface to a system memory (e.g., a DRAM implemented with multiple DIMMs; coherence logic 1222 to perform memory coherence operations. Multiple interfaces may be present to enable interconnection between the processor and other circuitry. For example, in one embodiment at least one direct media interface (DMI) 1216 interface is provided as well as one or more PCIe™ interfaces 1214. The display engine and these interfaces typcally couple to memory via a PCIe™ bridge 1218. Still further, to provide for communications between other agents, such as additional processors or other circuitry, one or more other interfaces (e.g. an Intel® Quick Path Interconnect (QPI) fabric) may be provided.

[00200] Referring now to FIG. 13, shown is a block diagram of a representative core; specifically, logical blocks of a back-end of a core, such as core 1230 from FIG. 12. In general, the structure shown in FIG. 13 includes an out-of-order processor that has a front end unit 1370 used to fetch incoming instructions, perform various processing (e.g. caching, decoding, branch predicting, etc.) and passing instructions/operations along to an out-of-order (OOO) engine 1380. OOO engine 1380 performs further processing on decoded instructions.

[00201] Specifically in the embodiment of FIG. 13, out-of-order engine 1380 includes an allocate unit 1382 to receive decoded instructions, which may be in the form of one or more micro-instructions or uops, from front end unit 1370, and allocate them to appropriate resources such as registers and so forth. Next, the instructions are provided to a reservation station 1384, which reserves resources and schedules them for execution on one of a plurality of execution units 1386A-1386N. Various types of execution units may be present, including, for example, arithmetic logic units (ALUs), load and store units, vector processing units (VPUs), floating point execution units, among others. Results from these different execution units are provided to a reorder buffer (ROB) 1388, which take unordered results and return them to correct program order.

[00202] Still referring to FIG. 13, note that both front end unit 1370 and out-of-order engine 1380 are coupled to different levels of a memory hierarchy. Specifically shown is an instruction level cache 1372, that in turn couples to a mid-level cache 1376, that in turn couples to a last level cache 1395. In one embodiment, last level cache 1395 is implemented in an on-chip (sometimes referred to as uncore) unit 1390. As an example, unit 1390 is similar to system agent 1210 of FIG. 12. As discussed above, UnCore 1390 communicates with system memory

1399, which, in the illustrated embodiment, is implemented via ED RAM. Note also that the various execution units 1386 within out-of-order engine 1380 are in communication with a first level cache 1374 that also is in communication with mid-level cache 1376. Note also that additional cores 1330N-2 – 1330N can couple to LLC 1395. Although shown at this high level in the embodiment of FIG. 13, understand that various alterations and additional components may be present.

[00203] Turning to FIG. 14, a block diagram of an exemplary computer system formed with a processor that includes execution units to execute an instruction, where one or more of the interconnects implement one or more features in accordance with one embodiment of the present invention is illustrated. System 1400 includes a component, such as a processor 1402 to employ execution units including logic to perform algorithms for process data, in accordance with the present invention, such as in the embodiment described herein. System 1400 is representative of processing systems based on the PENTIUM III™, PENTIUM 4™, Xeon™, Itanium, XScale™ and/or StrongARM™ microprocessors available from Intel Corporation of Santa Clara, California, although other systems (including PCs having other microprocessors, engineering workstations, set-top boxes and the like) may also be used. In one embodiment, sample system 1400 executes a version of the WINDOWS™ operating system available from Microsoft Corporation of Redmond, Washington, although other operating systems (UNIX and Linux for example), embedded software, and/or graphical user interfaces, may also be used. Thus, embodiments of the present invention are not limited to any specific combination of hardware circuitry and software.

[00204] Embodiments are not limited to computer systems. Alternative embodiments of the present invention can be used in other devices such as handheld devices and embedded applications. Some examples of handheld devices include cellular phones, Internet Protocol devices, digital cameras, personal digital assistants (PDAs), and handheld PCs. Embedded applications can include a micro controller, a digital signal processor (DSP), system on a chip, network computers (NetPC), set-top boxes, network hubs, wide area network (WAN) switches, or any other system that can perform one or more instructions in accordance with at least one embodiment.

[00205] In this illustrated embodiment, processor 1402 includes one or more execution units 1408 to implement an algorithm that is to perform at least one instruction. One embodiment may be described in the context of a single processor desktop or server system, but alternative embodiments may be included in a multiprocessor system. System 1400 is an example of a 'hub' system architecture. The computer system 1400 includes a processor 1402 to process data signals. The processor 1402, as one illustrative example, includes a complex

instruction set computer (CISC) microprocessor, a reduced instruction set computing (RISC) microprocessor, a very long instruction word (VLIW) microprocessor, a processor implementing a combination of instruction sets, or any other processor device, such as a digital signal processor, for example. The processor 1402 is coupled to a processor bus 1410 that transmits data signals between the processor 1402 and other components in the system 1400. The elements of system 1400 (e.g. graphics accelerator 1412, memory controller hub 1416, memory 1420, I/O controller hub 1424, wireless transceiver 1426, Flash BIOS 1428, Network controller 1434, Audio controller 1436, Serial expansion port 1438, I/O controller 1440, etc.) perform their conventional functions that are well known to those familiar with the art.

[00206] In one embodiment, the processor 1402 includes a Level 1 (L1) internal cache memory 1404. Depending on the architecture, the processor 1402 may have a single internal cache or multiple levels of internal caches. Other embodiments include a combination of both internal and external caches depending on the particular implementation and needs. Register file 1406 is to store different types of data in various registers including integer registers, floating point registers, vector registers, banked registers, shadow registers, checkpoint registers, status registers, and instruction pointer register.

[00207] Execution unit 1408, including logic to perform integer and floating point operations, also resides in the processor 1402. The processor 1402, in one embodiment, includes a microcode (ucode) ROM to store microcode, which when executed, is to perform algorithms for certain macroinstructions or handle complex scenarios. Here, microcode is potentially updateable to handle logic bugs/fixes for processor 1402. For one embodiment, execution unit 1408 includes logic to handle a packed instruction set 1409. By including the packed instruction set 1409 in the instruction set of a general-purpose processor 1402, along with associated circuitry to execute the instructions, the operations used by many multimedia applications may be performed using packed data in a general-purpose processor 1402. Thus, many multimedia applications are accelerated and executed more efficiently by using the full width of a processor's data bus for performing operations on packed data. This potentially eliminates the need to transfer smaller units of data across the processor's data bus to perform one or more operations, one data element at a time.

[00208] Alternate embodiments of an execution unit 1408 may also be used in micro controllers, embedded processors, graphics devices, DSPs, and other types of logic circuits. System 1400 includes a memory 1420. Memory 1420 includes a dynamic random access memory (DRAM) device, a static random access memory (SRAM) device, flash memory device, or other memory device. Memory 1420 stores instructions and/or data represented by data signals that are to be executed by the processor 1402.

[00209] Note that any of the aforementioned features or aspects of the invention may be utilized on one or more interconnect illustrated in FIG. 14. For example, an on-die interconnect (ODI), which is not shown, for coupling internal units of processor 1402 implements one or more aspects of the invention described above. Or the invention is associated with a processor bus 1410 (e.g. Intel Quick Path Interconnect (QPI) or other known high performance computing interconnect), a high bandwidth memory path 1418 to memory 1420, a point-to-point link to graphics accelerator 1412 (e.g. a Peripheral Component Interconnect express (PCIe) compliant fabric), a controller hub interconnect 1422, an I/O or other interconnect (e.g. USB, PCI, PCIe) for coupling the other illustrated components. Some examples of such components include the audio controller 1436, firmware hub (flash BIOS) 1428, wireless transceiver 1426, data storage 1424, legacy I/O controller 1410 containing user input and keyboard interfaces 1442, a serial expansion port 1438 such as Universal Serial Bus (USB), and a network controller 1434. The data storage device 1424 can comprise a hard disk drive, a floppy disk drive, a CD-ROM device, a flash memory device, or other mass storage device.

[00210] Referring now to FIG. 15, shown is a block diagram of a second system 1500 in accordance with an embodiment of the present invention. As shown in FIG. 15, multiprocessor system 1500 is a point-to-point interconnect system, and includes a first processor 1570 and a second processor 1580 coupled via a point-to-point interconnect 1550. Each of processors 1570 and 1580 may be some version of a processor. In one embodiment, 1552 and 1554 are part of a serial, point-to-point coherent interconnect fabric, such as Intel's Quick Path Interconnect (QPI) architecture. As a result, the invention may be implemented within the QPI architecture.

[00211] While shown with only two processors 1570, 1580, it is to be understood that the scope of the present invention is not so limited. In other embodiments, one or more additional processors may be present in a given processor.

[00212] Processors 1570 and 1580 are shown including integrated memory controller units 1572 and 1582, respectively. Processor 1570 also includes as part of its bus controller units point-to-point (P-P) interfaces 1576 and 1578; similarly, second processor 1580 includes P-P interfaces 1586 and 1588. Processors 1570, 1580 may exchange information via a point-to-point (P-P) interface 1550 using P-P interface circuits 1578, 1588. As shown in FIG. 15, IMCs 1572 and 1582 couple the processors to respective memories, namely a memory 1532 and a memory 1534, which may be portions of main memory locally attached to the respective processors.

[00213] Processors 1570, 1580 each exchange information with a chipset 1590 via individual P-P interfaces 1552, 1554 using point to point interface circuits 1576, 1594, 1586,

1598. Chipset 1590 also exchanges information with a high-performance graphics circuit 1538 via an interface circuit 1592 along a high-performance graphics interconnect 1539.

[00214] A shared cache (not shown) may be included in either processor or outside of both processors; yet connected with the processors via P-P interconnect, such that either or both processors' local cache information may be stored in the shared cache if a processor is placed into a low power mode.

[00215] Chipset 1590 may be coupled to a first bus 1516 via an interface 1596. In one embodiment, first bus 1516 may be a Peripheral Component Interconnect (PCI) bus, or a bus such as a PCI Express bus or another third generation I/O interconnect bus, although the scope of the present invention is not so limited.

[00216] As shown in FIG. 15, various I/O devices 1514 are coupled to first bus 1516, along with a bus bridge 1518 which couples first bus 1516 to a second bus 1520. In one embodiment, second bus 1520 includes a low pin count (LPC) bus. Various devices are coupled to second bus 1520 including, for example, a keyboard and/or mouse 1522, communication devices 1527 and a storage unit 1528 such as a disk drive or other mass storage device which often includes instructions/code and data 1530, in one embodiment. Further, an audio I/O 1524 is shown coupled to second bus 1520. Note that other architectures are possible, where the included components and interconnect architectures vary. For example, instead of the point-to-point architecture of FIG. 15, a system may implement a multi-drop bus or other such architecture.

FIG. 16

[00217] Referring now to FIG. 16, a block diagram of components present in a computer system in accordance with an embodiment of the present invention is illustrated. As shown in FIG. 16, system 1600 includes any combination of components. These components may be implemented as ICs, portions thereof, discrete electronic devices, or other modules, logic, hardware, software, firmware, or a combination thereof adapted in a computer system, or as components otherwise incorporated within a chassis of the computer system. Note also that the block diagram of FIG. 16 is intended to show a high level view of many components of the computer system. However, it is to be understood that some of the components shown may be omitted, additional components may be present, and different arrangement of the components shown may occur in other implementations. As a result, the invention described above may be implemented in any portion of one or more of the interconnects illustrated or described below.

[00218] As seen in FIG. 16, a processor 1610, in one embodiment, includes a microprocessor, multi-core processor, multithreaded processor, an ultra low voltage processor, an embedded processor, or other known processing element. In the illustrated implementation,

processor 1610 acts as a main processing unit and central hub for communication with many of the various components of the system 1600. As one example, processor 1600 is implemented as a system on a chip (SoC). As a specific illustrative example, processor 1610 includes an Intel® Architecture Core™-based processor such as an i3, i5, i7 or another such processor available from Intel Corporation, Santa Clara, CA. However, understand that other low power processors such as available from Advanced Micro Devices, Inc. (AMD) of Sunnyvale, CA, a MIPS-based design from MIPS Technologies, Inc. of Sunnyvale, CA, an ARM-based design licensed from ARM Holdings, Ltd. or customer thereof, or their licensees or adopters may instead be present in other embodiments such as an Apple A5/A6 processor, a Qualcomm Snapdragon processor, or TI OMAP processor. Note that many of the customer versions of such processors are modified and varied; however, they may support or recognize a specific instructions set that performs defined algorithms as set forth by the processor licensor. Here, the microarchitectural implementation may vary, but the architectural function of the processor is usually consistent. Certain details regarding the architecture and operation of processor 1610 in one implementation will be discussed further below to provide an illustrative example.

[00219] Processor 1610, in one embodiment, communicates with a system memory 1615. As an illustrative example, which in an embodiment can be implemented via multiple memory devices to provide for a given amount of system memory. As examples, the memory can be in accordance with a Joint Electron Devices Engineering Council (JEDEC) low power double data rate (LPDDR)-based design such as the current LPDDR2 standard according to JEDEC JESD 209-2E (published April 2009), or a next generation LPDDR standard to be referred to as LPDDR3 or LPDDR4 that will offer extensions to LPDDR2 to increase bandwidth. In various implementations the individual memory devices may be of different package types such as single die package (SDP), dual die package (DDP) or quad die package (14P). These devices, in some embodiments, are directly soldered onto a motherboard to provide a lower profile solution, while in other embodiments the devices are configured as one or more memory modules that in turn couple to the motherboard by a given connector. And of course, other memory implementations are possible such as other types of memory modules, e.g., dual inline memory modules (DIMMs) of different varieties including but not limited to microDIMMs, MiniDIMMs. In a particular illustrative embodiment, memory is sized between 2GB and 16GB, and may be configured as a DDR3LM package or an LPDDR2 or LPDDR3 memory that is soldered onto a motherboard via a ball grid array (BGA).

[00220] To provide for persistent storage of information such as data, applications, one or more operating systems and so forth, a mass storage 1620 may also couple to processor 1610. In various embodiments, to enable a thinner and lighter system design as well as to improve system

responsiveness, this mass storage may be implemented via a SSD. However in other embodiments, the mass storage may primarily be implemented using a hard disk drive (HDD) with a smaller amount of SSD storage to act as a SSD cache to enable non-volatile storage of context state and other such information during power down events so that a fast power up can occur on re-initiation of system activities. Also shown in FIG. 16, a flash device 1622 may be coupled to processor 1610, e.g., via a serial peripheral interface (SPI). This flash device may provide for non-volatile storage of system software, including a basic input/output software (BIOS) as well as other firmware of the system.

[00221] In various embodiments, mass storage of the system is implemented by a SSD alone or as a disk, optical or other drive with an SSD cache. In some embodiments, the mass storage is implemented as a SSD or as a HDD along with a restore (RST) cache module. In various implementations, the HDD provides for storage of between 320GB-4 terabytes (TB) and upward while the RST cache is implemented with a SSD having a capacity of 24GB-256GB. Note that such SSD cache may be configured as a single level cache (SLC) or multi-level cache (MLC) option to provide an appropriate level of responsiveness. In a SSD-only option, the module may be accommodated in various locations such as in a mSATA or NGFF slot. As an example, an SSD has a capacity ranging from 120GB-1TB.

[00222] Various input/output (IO) devices may be present within system 1600. Specifically shown in the embodiment of FIG. 16 is a display 1624 which may be a high definition LCD or LED panel configured within a lid portion of the chassis. This display panel may also provide for a touch screen 1625, e.g., adapted externally over the display panel such that via a user's interaction with this touch screen, user inputs can be provided to the system to enable desired operations, e.g., with regard to the display of information, accessing of information and so forth. In one embodiment, display 1624 may be coupled to processor 1610 via a display interconnect that can be implemented as a high performance graphics interconnect. Touch screen 1625 may be coupled to processor 1610 via another interconnect, which in an embodiment can be an I^2C interconnect. As further shown in FIG. 16, in addition to touch screen 1625, user input by way of touch can also occur via a touch pad 1630 which may be configured within the chassis and may also be coupled to the same I^2C interconnect as touch screen 1625.

[00223] The display panel may operate in multiple modes. In a first mode, the display panel can be arranged in a transparent state in which the display panel is transparent to visible light. In various embodiments, the majority of the display panel may be a display except for a bezel around the periphery. When the system is operated in a notebook mode and the display panel is operated in a transparent state, a user may view information that is presented on the

display panel while also being able to view objects behind the display. In addition, information displayed on the display panel may be viewed by a user positioned behind the display. Or the operating state of the display panel can be an opaque state in which visible light does not transmit through the display panel.

[00224] In a tablet mode the system is folded shut such that the back display surface of the display panel comes to rest in a position such that it faces outwardly towards a user, when the bottom surface of the base panel is rested on a surface or held by the user. In the tablet mode of operation, the back display surface performs the role of a display and user interface, as this surface may have touch screen functionality and may perform other known functions of a conventional touch screen device, such as a tablet device. To this end, the display panel may include a transparency-adjusting layer that is disposed between a touch screen layer and a front display surface. In some embodiments the transparency-adjusting layer may be an electrochromic layer (EC), a LCD layer, or a combination of EC and LCD layers.

[00225] In various embodiments, the display can be of different sizes, e.g., an 11.6" or a 13.3" screen, and may have a 16:9 aspect ratio, and at least 300 nits brightness. Also the display may be of full high definition (HD) resolution (at least 1920 x 1080p), be compatible with an embedded display port (eDP), and be a low power panel with panel self refresh.

[00226] As to touch screen capabilities, the system may provide for a display multi-touch panel that is multi-touch capacitive and being at least 5 finger capable. And in some embodiments, the display may be 10 finger capable. In one embodiment, the touch screen is accommodated within a damage and scratch-resistant glass and coating (e.g., Gorilla Glass™ or Gorilla Glass 2™) for low friction to reduce "finger burn" and avoid "finger skipping". To provide for an enhanced touch experience and responsiveness, the touch panel, in some implementations, has multi-touch functionality, such as less than 2 frames (30Hz) per static view during pinch zoom, and single-touch functionality of less than 1 cm per frame (30Hz) with 200ms (lag on finger to pointer). The display, in some implementations, supports edge-to-edge glass with a minimal screen bezel that is also flush with the panel surface, and limited IO interference when using multi-touch.

[00227] For perceptual computing and other purposes, various sensors may be present within the system and may be coupled to processor 1610 in different manners. Certain inertial and environmental sensors may couple to processor 1610 through a sensor hub 1640, e.g., via an I^2C interconnect. In the embodiment shown in FIG. 16, these sensors may include an accelerometer 1641, an ambient light sensor (ALS) 1642, a compass 1643 and a gyroscope 1644. Other environmental sensors may include one or more thermal sensors 1646 which in some embodiments couple to processor 1610 via a system management bus (SMBus) bus.

[00228] Using the various inertial and environmental sensors present in a platform, many different use cases may be realized. These use cases enable advanced computing operations including perceptual computing and also allow for enhancements with regard to power management/battery life, security, and system responsiveness.

[00229] For example with regard to power management/battery life issues, based at least on part on information from an ambient light sensor, the ambient light conditions in a location of the platform are determined and intensity of the display controlled accordingly. Thus, power consumed in operating the display is reduced in certain light conditions.

[00230] As to security operations, based on context information obtained from the sensors such as location information, it may be determined whether a user is allowed to access certain secure documents. For example, a user may be permitted to access such documents at a work place or a home location. However, the user is prevented from accessing such documents when the platform is present at a public location. This determination, in one embodiment, is based on location information, e.g., determined via a GPS sensor or camera recognition of landmarks. Other security operations may include providing for pairing of devices within a close range of each other, e.g., a portable platform as described herein and a user's desktop computer, mobile telephone or so forth. Certain sharing, in some implementations, are realized via near field communication when these devices are so paired. However, when the devices exceed a certain range, such sharing may be disabled. Furthermore, when pairing a platform as described herein and a smartphone, an alarm may be configured to be triggered when the devices move more than a predetermined distance from each other, when in a public location. In contrast, when these paired devices are in a safe location, e.g., a work place or home location, the devices may exceed this predetermined limit without triggering such alarm.

[00231] Responsiveness may also be enhanced using the sensor information. For example, even when a platform is in a low power state, the sensors may still be enabled to run at a relatively low frequency. Accordingly, any changes in a location of the platform, e.g., as determined by inertial sensors, GPS sensor, or so forth is determined. If no such changes have been registered, a faster connection to a previous wireless hub such as a Wi-Fi™ access point or similar wireless enabler occurs, as there is no need to scan for available wireless network resources in this case. Thus, a greater level of responsiveness when waking from a low power state is achieved.

[00232] It is to be understood that many other use cases may be enabled using sensor information obtained via the integrated sensors within a platform as described herein, and the above examples are only for purposes of illustration. Using a system as described herein, a

perceptual computing system may allow for the addition of alternative input modalities, including gesture recognition, and enable the system to sense user operations and intent.

[00233] In some embodiments one or more infrared or other heat sensing elements, or any other element for sensing the presence or movement of a user may be present. Such sensing elements may include multiple different elements working together, working in sequence, or both. For example, sensing elements include elements that provide initial sensing, such as light or sound projection, followed by sensing for gesture detection by, for example, an ultrasonic time of flight camera or a patterned light camera.

[00234] Also in some embodiments, the system includes a light generator to produce an illuminated line. In some embodiments, this line provides a visual cue regarding a virtual boundary, namely an imaginary or virtual location in space, where action of the user to pass or break through the virtual boundary or plane is interpreted as an intent to engage with the computing system. In some embodiments, the illuminated line may change colors as the computing system transitions into different states with regard to the user. The illuminated line may be used to provide a visual cue for the user of a virtual boundary in space, and may be used by the system to determine transitions in state of the computer with regard to the user, including determining when the user wishes to engage with the computer.

[00235] In some embodiments, the computer senses user position and operates to interpret the movement of a hand of the user through the virtual boundary as a gesture indicating an intention of the user to engage with the computer. In some embodiments, upon the user passing through the virtual line or plane the light generated by the light generator may change, thereby providing visual feedback to the user that the user has entered an area for providing gestures to provide input to the computer.

[00236] Display screens may provide visual indications of transitions of state of the computing system with regard to a user. In some embodiments, a first screen is provided in a first state in which the presence of a user is sensed by the system, such as through use of one or more of the sensing elements.

[00237] In some implementations, the system acts to sense user identity, such as by facial recognition. Here, transition to a second screen may be provided in a second state, in which the computing system has recognized the user identity, where this second the screen provides visual feedback to the user that the user has transitioned into a new state. Transition to a third screen may occur in a third state in which the user has confirmed recognition of the user.

[00238] In some embodiments, the computing system may use a transition mechanism to determine a location of a virtual boundary for a user, where the location of the virtual boundary may vary with user and context. The computing system may generate a light, such as an

illuminated line, to indicate the virtual boundary for engaging with the system. In some embodiments, the computing system may be in a waiting state, and the light may be produced in a first color. The computing system may detect whether the user has reached past the virtual boundary, such as by sensing the presence and movement of the user using sensing elements.

[00239] In some embodiments, if the user has been detected as having crossed the virtual boundary (such as the hands of the user being closer to the computing system than the virtual boundary line), the computing system may transition to a state for receiving gesture inputs from the user, where a mechanism to indicate the transition may include the light indicating the virtual boundary changing to a second color.

[00240] In some embodiments, the computing system may then determine whether gesture movement is detected. If gesture movement is detected, the computing system may proceed with a gesture recognition process, which may include the use of data from a gesture data library, which may reside in memory in the computing device or may be otherwise accessed by the computing device.

[00241] If a gesture of the user is recognized, the computing system may perform a function in response to the input, and return to receive additional gestures if the user is within the virtual boundary. In some embodiments, if the gesture is not recognized, the computing system may transition into an error state, where a mechanism to indicate the error state may include the light indicating the virtual boundary changing to a third color, with the system returning to receive additional gestures if the user is within the virtual boundary for engaging with the computing system.

[00242] As mentioned above, in other embodiments the system can be configured as a convertible tablet system that can be used in at least two different modes, a tablet mode and a notebook mode. The convertible system may have two panels, namely a display panel and a base panel such that in the tablet mode the two panels are disposed in a stack on top of one another. In the tablet mode, the display panel faces outwardly and may provide touch screen functionality as found in conventional tablets. In the notebook mode, the two panels may be arranged in an open clamshell configuration.

[00243] In various embodiments, the accelerometer may be a 3-axis accelerometer having data rates of at least 50Hz. A gyroscope may also be included, which can be a 3-axis gyroscope. In addition, an e-compass/magnetometer may be present. Also, one or more proximity sensors may be provided (e.g., for lid open to sense when a person is in proximity (or not) to the system and adjust power/performance to extend battery life). For some OS's Sensor Fusion capability including the accelerometer, gyroscope, and compass may provide enhanced features. In

addition, via a sensor hub having a real-time clock (RTC), a wake from sensors mechanism may be realized to receive sensor input when a remainder of the system is in a low power state.

[00244] In some embodiments, an internal lid/display open switch or sensor to indicate when the lid is closed/open, and can be used to place the system into Connected Standby or automatically wake from Connected Standby state. Other system sensors can include ACPI sensors for internal processor, memory, and skin temperature monitoring to enable changes to processor and system operating states based on sensed parameters.

[00245] In an embodiment, the OS may be a Microsoft® Windows® 8 OS that implements Connected Standby (also referred to herein as Win8 CS). Windows 8 Connected Standby or another OS having a similar state can provide, via a platform as described herein, very low ultra idle power to enable applications to remain connected, e.g., to a cloud-based location, at very low power consumption. The platform can supports 3 power states, namely screen on (normal); Connected Standby (as a default "off" state); and shutdown (zero watts of power consumption). Thus in the Connected Standby state, the platform is logically on (at minimal power levels) even though the screen is off. In such a platform, power management can be made to be transparent to applications and maintain constant connectivity, in part due to offload technology to enable the lowest powered component to perform an operation.

[00246] Also seen in FIG. 16, various peripheral devices may couple to processor 1610 via a low pin count (LPC) interconnect. In the embodiment shown, various components can be coupled through an embedded controller 1635. Such components can include a keyboard 1636 (e.g., coupled via a PS2 interface), a fan 1637, and a thermal sensor 1639. In some embodiments, touch pad 1630 may also couple to EC 1635 via a PS2 interface. In addition, a security processor such as a trusted platform module (TPM) 1638 in accordance with the Trusted Computing Group (TCG) TPM Specification Version 1.2, dated Oct. 2, 2003, may also couple to processor 1610 via this LPC interconnect. However, understand the scope of the present invention is not limited in this regard and secure processing and storage of secure information may be in another protected location such as a static random access memory (SRAM) in a security coprocessor, or as encrypted data blobs that are only decrypted when protected by a secure enclave (SE) processor mode.

[00247] In a particular implementation, peripheral ports may include a high definition media interface (HDMI) connector (which can be of different form factors such as full size, mini or micro); one or more USB ports, such as full-size external ports in accordance with the Universal Serial Bus Revision 3.0 Specification (November 2008), with at least one powered for charging of USB devices (such as smartphones) when the system is in Connected Standby state and is plugged into AC wall power. In addition, one or more Thunderbolt™ ports can be

provided. Other ports may include an externally accessible card reader such as a full size SD-XC card reader and/or a SIM card reader for WWAN (e.g., an 8 pin card reader). For audio, a 3.5mm jack with stereo sound and microphone capability (e.g., combination functionality) can be present, with support for jack detection (e.g., headphone only support using microphone in the lid or headphone with microphone in cable). In some embodiments, this jack can be re-taskable between stereo headphone and stereo microphone input. Also, a power jack can be provided for coupling to an AC brick.

[00248] System 1600 can communicate with external devices in a variety of manners, including wirelessly. In the embodiment shown in FIG. 16, various wireless modules, each of which can correspond to a radio configured for a particular wireless communication protocol, are present. One manner for wireless communication in a short range such as a near field may be via a near field communication (NFC) unit 1645 which may communicate, in one embodiment with processor 1610 via an SMBus. Note that via this NFC unit 1645, devices in close proximity to each other can communicate. For example, a user can enable system 1600 to communicate with another (e.g.,) portable device such as a smartphone of the user via adapting the two devices together in close relation and enabling transfer of information such as identification information payment information, data such as image data or so forth. Wireless power transfer may also be performed using a NFC system.

[00249] Using the NFC unit described herein, users can bump devices side-to-side and place devices side-by-side for near field coupling functions (such as near field communication and wireless power transfer (WPT)) by leveraging the coupling between coils of one or more of such devices. More specifically, embodiments provide devices with strategically shaped, and placed, ferrite materials, to provide for better coupling of the coils. Each coil has an inductance associated with it, which can be chosen in conjunction with the resistive, capacitive, and other features of the system to enable a common resonant frequency for the system.

[00250] As further seen in FIG. 16, additional wireless units can include other short range wireless engines including a WLAN unit 1650 and a Bluetooth unit 1652. Using WLAN unit 1650, Wi-Fi™ communications in accordance with a given Institute of Electrical and Electronics Engineers (IEEE) 802.11 standard can be realized, while via Bluetooth unit 1652, short range communications via a Bluetooth protocol can occur. These units may communicate with processor 1610 via, e.g., a USB link or a universal asynchronous receiver transmitter (UART) link. Or these units may couple to processor 1610 via an interconnect according to a Peripheral Component Interconnect Express™ (PCIe™) protocol, e.g., in accordance with the PCI Express™ Specification Base Specification version 3.0 (published January 17, 2007), or another such protocol such as a serial data input/output (SDIO) standard. Of course, the actual

physical connection between these peripheral devices, which may be configured on one or more add-in cards, can be by way of the NGFF connectors adapted to a motherboard.

[00251] In addition, wireless wide area communications, e.g., according to a cellular or other wireless wide area protocol, can occur via a WWAN unit 1656 which in turn may couple to a subscriber identity module (SIM) 1657. In addition, to enable receipt and use of location information, a GPS module 1655 may also be present. Note that in the embodiment shown in FIG. 16, WWAN unit 1656 and an integrated capture device such as a camera module 1654 may communicate via a given USB protocol such as a USB 2.0 or 3.0 link, or a UART or I^2C protocol. Again the actual physical connection of these units can be via adaptation of a NGFF add-in card to an NGFF connector configured on the motherboard.

[00252] In a particular embodiment, wireless functionality can be provided modularly, e.g., with a WiFi™ 802.11ac solution (e.g., add-in card that is backward compatible with IEEE 802.11abgn) with support for Windows 8 CS. This card can be configured in an internal slot (e.g., via an NGFF adapter). An additional module may provide for Bluetooth capability (e.g., Bluetooth 4.0 with backwards compatibility) as well as Intel® Wireless Display functionality. In addition NFC support may be provided via a separate device or multi-function device, and can be positioned as an example, in a front right portion of the chassis for easy access. A still additional module may be a WWAN device that can provide support for 3G/4G/LTE and GPS. This module can be implemented in an internal (e.g., NGFF) slot. Integrated antenna support can be provided for WiFi™, Bluetooth, WWAN, NFC and GPS, enabling seamless transition from WiFi™ to WWAN radios, wireless gigabit (WiGig) in accordance with the Wireless Gigabit Specification (July 2010), and vice versa.

[00253] As described above, an integrated camera can be incorporated in the lid. As one example, this camera can be a high resolution camera, e.g., having a resolution of at least 2.0 megapixels (MP) and extending to 6.0 MP and beyond.

[00254] To provide for audio inputs and outputs, an audio processor can be implemented via a digital signal processor (DSP) 1660, which may couple to processor 1610 via a high definition audio (HDA) link. Similarly, DSP 1660 may communicate with an integrated coder/decoder (CODEC) and amplifier 1662 that in turn may couple to output speakers 1663 which may be implemented within the chassis. Similarly, amplifier and CODEC 1662 can be coupled to receive audio inputs from a microphone 1665 which in an embodiment can be implemented via dual array microphones (such as a digital microphone array) to provide for high quality audio inputs to enable voice-activated control of various operations within the system. Note also that audio outputs can be provided from amplifier/CODEC 1662 to a

headphone jack 1664. Although shown with these particular components in the embodiment of FIG. 16, understand the scope of the present invention is not limited in this regard.

[00255] In a particular embodiment, the digital audio codec and amplifier are capable of driving the stereo headphone jack, stereo microphone jack, an internal microphone array and stereo speakers. In different implementations, the codec can be integrated into an audio DSP or coupled via an HD audio path to a peripheral controller hub (PCH). In some implementations, in addition to integrated stereo speakers, one or more bass speakers can be provided, and the speaker solution can support DTS audio.

[00256] In some embodiments, processor 1610 may be powered by an external voltage regulator (VR) and multiple internal voltage regulators that are integrated inside the processor die, referred to as fully integrated voltage regulators (FIVRs). The use of multiple FIVRs in the processor enables the grouping of components into separate power planes, such that power is regulated and supplied by the FIVR to only those components in the group. During power management, a given power plane of one FIVR may be powered down or off when the processor is placed into a certain low power state, while another power plane of another FIVR remains active, or fully powered.

[00257] In one embodiment, a sustain power plane can be used during some deep sleep states to power on the I/O pins for several I/O signals, such as the interface between the processor and a PCH, the interface with the external VR and the interface with EC 1635. This sustain power plane also powers an on-die voltage regulator that supports the on-board SRAM or other cache memory in which the processor context is stored during the sleep state. The sustain power plane is also used to power on the processor's wakeup logic that monitors and processes the various wakeup source signals.

[00258] During power management, while other power planes are powered down or off when the processor enters certain deep sleep states, the sustain power plane remains powered on to support the above-referenced components. However, this can lead to unnecessary power consumption or dissipation when those components are not needed. To this end, embodiments may provide a connected standby sleep state to maintain processor context using a dedicated power plane. In one embodiment, the connected standby sleep state facilitates processor wakeup using resources of a PCH which itself may be present in a package with the processor. In one embodiment, the connected standby sleep state facilitates sustaining processor architectural functions in the PCH until processor wakeup, this enabling turning off all of the unnecessary processor components that were previously left powered on during deep sleep states, including turning off all of the clocks. In one embodiment, the PCH contains a time stamp counter (TSC)

and connected standby logic for controlling the system during the connected standby state. The integrated voltage regulator for the sustain power plane may reside on the PCH as well.

[00259] In an embodiment, during the connected standby state, an integrated voltage regulator may function as a dedicated power plane that remains powered on to support the dedicated cache memory in which the processor context is stored such as critical state variables when the processor enters the deep sleep states and connected standby state. This critical state may include state variables associated with the architectural, micro-architectural, debug state, and/or similar state variables associated with the processor.

[00260] The wakeup source signals from EC 1635 may be sent to the PCH instead of the processor during the connected standby state so that the PCH can manage the wakeup processing instead of the processor. In addition, the TSC is maintained in the PCH to facilitate sustaining processor architectural functions. Although shown with these particular components in the embodiment of FIG. 16, understand the scope of the present invention is not limited in this regard.

[00261] Power control in the processor can lead to enhanced power savings. For example, power can be dynamically allocate between cores, individual cores can change frequency/voltage, and multiple deep low power states can be provided to enable very low power consumption. In addition, dynamic control of the cores or independent core portions can provide for reduced power consumption by powering off components when they are not being used.

[00262] Some implementations may provide a specific power management IC (PMIC) to control platform power. Using this solution, a system may see very low (e.g., less than 5%) battery degradation over an extended duration (e.g., 16 hours) when in a given standby state, such as when in a Win8 Connected Standby state. In a Win8 idle state a battery life exceeding, e.g., 9 hours may be realized (e.g., at 150 nits). As to video playback, a long battery life can be realized, e.g., full HD video playback can occur for a minimum of 6 hours. A platform in one implementation may have an energy capacity of, e.g., 35 watt hours (Whr) for a Win8 CS using an SSD and (e.g.,) 40-44Whr for Win8 CS using an HDD with a RST cache configuration.

[00263] A particular implementation may provide support for 15W nominal CPU thermal design power (TDP), with a configurable CPU TDP of up to approximately 25W TDP design point. The platform may include minimal vents owing to the thermal features described above. In addition, the platform is pillow-friendly (in that no hot air is blowing at the user). Different maximum temperature points can be realized depending on the chassis material. In one implementation of a plastic chassis (at least having to lid or base portion of plastic), the

maximum operating temperature can be 52 degrees Celsius (C). And for an implementation of a metal chassis, the maximum operating temperature can be 46° C.

[00264] In different implementations, a security module such as a TPM can be integrated into a processor or can be a discrete device such as a TPM 2.0 device. With an integrated security module, also referred to as Platform Trust Technology (PTT), BIOS/firmware can be enabled to expose certain hardware features for certain security features, including secure instructions, secure boot, Intel® Anti-Theft Technology, Intel® Identity Protection Technology, Intel® Trusted Execution Technology (TXT), and Intel® Manageability Engine Technology along with secure user interfaces such as a secure keyboard and display.

[00265] Turning next to FIG. 17, an embodiment of a system on-chip (SOC) design in accordance with the inventions is depicted. As a specific illustrative example, SOC 1700 is included in user equipment (UE). In one embodiment, UE refers to any device to be used by an end-user to communicate, such as a hand-held phone, smartphone, tablet, ultra-thin notebook, notebook with broadband adapter, or any other similar communication device. Often a UE connects to a base station or node, which potentially corresponds in nature to a mobile station (MS) in a GSM network.

[00266] Here, SOC 1700 includes 2 cores—1706 and 1707. Similar to the discussion above, cores 1706 and 1707 may conform to an Instruction Set Architecture, such as an Intel® Architecture Core™-based processor, an Advanced Micro Devices, Inc. (AMD) processor, a MIPS-based processor, an ARM-based processor design, or a customer thereof, as well as their licensees or adopters. Cores 1706 and 1707 are coupled to cache control 1708 that is associated with bus interface unit 1709 and L2 cache 1710 to communicate with other parts of system 1700. Interconnect 1710 includes an on-chip interconnect, such as an IOSF, AMBA, or other interconnect discussed above, which potentially implements one or more aspects of the described invention.

[00267] Interface 1710 provides communication channels to the other components, such as a Subscriber Identity Module (SIM) 1730 to interface with a SIM card, a boot rom 1735 to hold boot code for execution by cores 1706 and 1707 to initialize and boot SOC 1700, a SDRAM controller 1740 to interface with external memory (e.g. DRAM 1760), a flash controller 1745 to interface with non-volatile memory (e.g. Flash 1765), a peripheral control 1350 (e.g. Serial Peripheral Interface) to interface with peripherals, video codecs 1720 and Video interface 1725 to display and receive input (e.g. touch enabled input), GPU 1715 to perform graphics related computations, etc. Any of these interfaces may incorporate aspects of the invention described herein.

[00268] In addition, the system illustrates peripherals for communication, such as a Bluetooth module 1770, 3G modem 1775, GPS 1785, and WiFi 1785. Note as stated above, a UE includes a radio for communication. As a result, these peripheral communication modules are not all required. However, in a UE some form a radio for external communication is to be included.

[00269] While the present invention has been described with respect to a limited number of embodiments, those skilled in the art will appreciate numerous modifications and variations therefrom. It is intended that the appended claims cover all such modifications and variations as fall within the true spirit and scope of this present invention.

[00270] A design may go through various stages, from creation to simulation to fabrication. Data representing a design may represent the design in a number of manners. First, as is useful in simulations, the hardware may be represented using a hardware description language or another functional description language. Additionally, a circuit level model with logic and/or transistor gates may be produced at some stages of the design process. Furthermore, most designs, at some stage, reach a level of data representing the physical placement of various devices in the hardware model. In the case where conventional semiconductor fabrication techniques are used, the data representing the hardware model may be the data specifying the presence or absence of various features on different mask layers for masks used to produce the integrated circuit. In any representation of the design, the data may be stored in any form of a machine readable medium. A memory or a magnetic or optical storage such as a disc may be the machine readable medium to store information transmitted via optical or electrical wave modulated or otherwise generated to transmit such information. When an electrical carrier wave indicating or carrying the code or design is transmitted, to the extent that copying, buffering, or re-transmission of the electrical signal is performed, a new copy is made. Thus, a communication provider or a network provider may store on a tangible, machine-readable medium, at least temporarily, an article, such as information encoded into a carrier wave, embodying techniques of embodiments of the present invention.

[00271] A module as used herein refers to any combination of hardware, software, and/or firmware. As an example, a module includes hardware, such as a micro-controller, associated with a non-transitory medium to store code adapted to be executed by the micro-controller. Therefore, reference to a module, in one embodiment, refers to the hardware, which is specifically configured to recognize and/or execute the code to be held on a non-transitory medium. Furthermore, in another embodiment, use of a module refers to the non-transitory medium including the code, which is specifically adapted to be executed by the microcontroller to perform predetermined operations. And as can be inferred, in yet another embodiment, the

term module (in this example) may refer to the combination of the microcontroller and the non-transitory medium. Often module boundaries that are illustrated as separate commonly vary and potentially overlap. For example, a first and a second module may share hardware, software, firmware, or a combination thereof, while potentially retaining some independent hardware, software, or firmware. In one embodiment, use of the term logic includes hardware, such as transistors, registers, or other hardware, such as programmable logic devices.

[00272] Use of the phrase 'to' or 'configured to,' in one embodiment, refers to arranging, putting together, manufacturing, offering to sell, importing and/or designing an apparatus, hardware, logic, or element to perform a designated or determined task. In this example, an apparatus or element thereof that is not operating is still 'configured to' perform a designated task if it is designed, coupled, and/or interconnected to perform said designated task. As a purely illustrative example, a logic gate may provide a 0 or a 1 during operation. But a logic gate 'configured to' provide an enable signal to a clock does not include every potential logic gate that may provide a 1 or 0. Instead, the logic gate is one coupled in some manner that during operation the 1 or 0 output is to enable the clock. Note once again that use of the term 'configured to' does not require operation, but instead focus on the latent state of an apparatus, hardware, and/or element, where in the latent state the apparatus, hardware, and/or element is designed to perform a particular task when the apparatus, hardware, and/or element is operating.

[00273] Furthermore, use of the phrases 'capable of/to,' and or 'operable to,' in one embodiment, refers to some apparatus, logic, hardware, and/or element designed in such a way to enable use of the apparatus, logic, hardware, and/or element in a specified manner. Note as above that use of to, capable to, or operable to, in one embodiment, refers to the latent state of an apparatus, logic, hardware, and/or element, where the apparatus, logic, hardware, and/or element is not operating but is designed in such a manner to enable use of an apparatus in a specified manner.

[00274] A value, as used herein, includes any known representation of a number, a state, a logical state, or a binary logical state. Often, the use of logic levels, logic values, or logical values is also referred to as 1's and 0's, which simply represents binary logic states. For example, a 1 refers to a high logic level and 0 refers to a low logic level. In one embodiment, a storage cell, such as a transistor or flash cell, may be capable of holding a single logical value or multiple logical values. However, other representations of values in computer systems have been used. For example the decimal number ten may also be represented as a binary value of 1010 and a hexadecimal letter A. Therefore, a value includes any representation of information capable of being held in a computer system.

[00275] Moreover, states may be represented by values or portions of values. As an example, a first value, such as a logical one, may represent a default or initial state, while a second value, such as a logical zero, may represent a non-default state. In addition, the terms reset and set, in one embodiment, refer to a default and an updated value or state, respectively. For example, a default value potentially includes a high logical value, i.e. reset, while an updated value potentially includes a low logical value, i.e. set. Note that any combination of values may be utilized to represent any number of states.

[00276] The embodiments of methods, hardware, software, firmware or code set forth above may be implemented via instructions or code stored on a machine-accessible, machine readable, computer accessible, or computer readable medium which are executable by a processing element. A non-transitory machine-accessible/readable medium includes any mechanism that provides (i.e., stores and/or transmits) information in a form readable by a machine, such as a computer or electronic system. For example, a non-transitory machine-accessible medium includes random-access memory (RAM), such as static RAM (SRAM) or dynamic RAM (DRAM); ROM; magnetic or optical storage medium; flash memory devices; electrical storage devices; optical storage devices; acoustical storage devices; other form of storage devices for holding information received from transitory (propagated) signals (e.g., carrier waves, infrared signals, digital signals); etc, which are to be distinguished from the non-transitory mediums that may receive information there from.

[00277] Instructions used to program logic to perform embodiments of the invention may be stored within a memory in the system, such as DRAM, cache, flash memory, or other storage. Furthermore, the instructions can be distributed via a network or by way of other computer readable media. Thus a machine-readable medium may include any mechanism for storing or transmitting information in a form readable by a machine (e.g., a computer), but is not limited to, floppy diskettes, optical disks, Compact Disc, Read-Only Memory (CD-ROMs), and magneto-optical disks, Read-Only Memory (ROMs), Random Access Memory (RAM), Erasable Programmable Read-Only Memory (EPROM), Electrically Erasable Programmable Read-Only Memory (EEPROM), magnetic or optical cards, flash memory, or a tangible, machine-readable storage used in the transmission of information over the Internet via electrical, optical, acoustical or other forms of propagated signals (e.g., carrier waves, infrared signals, digital signals, etc.). Accordingly, the computer-readable medium includes any type of tangible machine-readable medium suitable for storing or transmitting electronic instructions or information in a form readable by a machine (e.g., a computer)

[00278] Reference throughout this specification to “one embodiment” or “an embodiment” means that a particular feature, structure, or characteristic described in connection

with the embodiment is included in at least one embodiment of the present invention. Thus, the appearances of the phrases "in one embodiment" or "in an embodiment" in various places throughout this specification are not necessarily all referring to the same embodiment. Furthermore, the particular features, structures, or characteristics may be combined in any suitable manner in one or more embodiments.

[00279] In the foregoing specification, a detailed description has been given with reference to specific exemplary embodiments. It will, however, be evident that various modifications and changes may be made thereto without departing from the broader spirit and scope of the invention as set forth in the appended claims. The specification and drawings are, accordingly, to be regarded in an illustrative sense rather than a restrictive sense. Furthermore, the foregoing use of embodiment and other exemplarily language does not necessarily refer to the same embodiment or the same example, but may refer to different and distinct embodiments, as well as potentially the same embodiment.

[00280] This disclosure describes how multiple IO devices can differentiate the cache allocation, and as result the latency and bandwidth seen by the transactions. One use case that could benefit from such a solution is Industrial Control Applications Real-Time control network and the standard communications network controllers. The transactions from the Real-Time network need to be delivered to the CPU as quickly as possible, likewise transactions to be transmitted over the Real-Time network need to be accessed with as little latency and jitter as possible.

[00281] Technologies such as DDIO have provided significant benefit to latency sensitive transactions; however, implementations of DDIO do not provide differentiation between low priority IO traffic, and high priority IO traffic, and DDIO transactions are evicted from the LLC based on a pseudo "least recently used" mechanism that has no way of differentiating between the two. FIG. 18 is a graph showing the incremental benefits of DDIO, Cache QoS, and ultimately the concepts described herein.

[00282] The first set of data on the graph shows the average latency of a PCIe device issuing a read over TC1 in four scenarios, those scenarios being, first when the system is idle, next when another PCIe device is issuing reads, next when the CPU cores are generating CPU to memory traffic, and finally when both the cores and the other PCIe device are active. Form this set of data we can see that the idle latency and the concurrent PCIe latency are very similar, however when the CPU cores start generating traffic we see a spike in latency, and this spike is further exasperated by the addiction of PCIe traffic from another device.

[00283] The second set of data on the graph shows the same four scenarios but now with DDIO turned on. It is interesting to note that DDIO reduces the idle latency significantly as we

are now going to LLC rather than memory, but the two worst case scenarios still show significant increases in latency over idle.

[00284] The third set of data uses Cache QoS to make sure there is no overlap between the LLC ways used by the cores and those used by the IO device, this brings all but the last test case to near idle latencies.

[00285] The last graph shows the average PCIe read latency with Traffic Class Cache Allocation simulation enabled. As in the case with DDIO and CAT, the latency with concurrent CPU traffic is near idle. However, with Traffic Class Cache Allocation, the concurrent PCIe traffic uses a different waymask and is therefore unable to evict the cache lines used for the testing. This results in near idle latency for the case with concurrent PCIe traffic, and concurrent CPU and PCIe traffic.

[00286] The final set of data was collected using Cache QoS to simulate the proposed invention. The transactions from one device with a traffic class of TC1 target physical addresses were pre-allocated into cache ways not targeted by either DDIO or the CPUs.

CLAIMS:

1. An apparatus comprising:

means for receiving a data packet, the data packet comprising data;

means for identifying a traffic class of the data packet; and

means for selectively storing the data into cache memory or system memory based on the traffic class of the data packet.

2. The apparatus of claim 1, wherein the apparatus comprises means for receiving the data packet across a link compliant with a Peripheral Component Interface Express (PCIe)-based interconnect.

3. The apparatus of claim 1, wherein the data packet comprises a packet header, the packet header including a transaction descriptor field, the transaction descriptor field including the traffic class of the data packet, and

wherein the traffic class is a Transaction Layer Packet (TLP) label of the data packet.

4. The apparatus of claim 1, wherein the traffic class is used to differentiate data packets having a high priority traffic class from data packets having a low priority traffic class.

5. The apparatus of claim 4, comprising means for mapping the high priority traffic class to a high priority category of data and means for mapping the low priority traffic class to a low priority category of data, and

means for storing data packets with a traffic class mapped to high priority category of data into the cache memory and means for selectively storing data of data packets with a traffic class mapped to a low priority category of data to the system memory.

6. The apparatus of claim 1, comprising means for mapping the traffic class of the data packet to a class of service and selectively storing the data packet in the cache memory or in the system memory based on the class of service.

7. The apparatus of claim 6, comprising means for mapping the high priority traffic class to a high priority class of service and mapping the low priority traffic class to a low priority class of service, and comprising means for selectively storing data packets with a traffic class

mapped to high priority class of service into the cache memory and selectively storing data packets with a traffic class mapped to a low priority class of service to the system memory.

8. The apparatus of claim 6, comprising means for mapping the high priority traffic class to a high priority class of service and mapping the low priority traffic class to a low priority class of service, and comprising means for determining whether the traffic mapped to the high priority class of service can be allocated to a region of the cache memory and determining whether the traffic mapped to the low priority class of service can be allocated to a region of the cache memory, and comprising means for determining that the low priority class of service traffic cannot be allocated into the cache memory and for storing the low priority class of service traffic into the system memory.

9. The apparatus of claim 1, comprising a class of service waymask register, the class of service waymask register comprising bits corresponding to a traffic class mapping and used to identify ways of the cache memory for storing data packets having a traffic class mapped to a corresponding class of service, and wherein each class of service corresponds to a waymask of the cache memory, and the apparatus comprising means for selectively storing the data packets into the cache memory based on the waymask corresponding to the class of service.

10. A computer program product tangibly embodied on a non-transitory computer-readable medium, the computer program product comprising instructions operable when executed to:

receive a data packet,

determine a traffic class of the data packet, and

selectively store the data packet directly into cache memory or into system memory based on the traffic class of the data packet.

11. The computer program product of claim 10, wherein the data packet is received across a link compliant with a Peripheral Component Interface Express (PCIe)-based interconnect.

12. The computer program product of claim 10, wherein the data packet comprises a packet header, the packet header including a transaction descriptor field, the transaction descriptor field including the traffic class of the data packet.

13. The computer program product of claim 10, wherein the traffic class is a Transaction Layer Packet (TLP) label of the data packet.

14. The computer program product of claim 10, wherein the traffic class is used to differentiate data packets having a high priority traffic class from data packets having a low priority traffic class.

15. The computer program product of claim 14, the instructions when executed configured to map the traffic class of the data packet to a corresponding category of data and selectively store the data packet in the cache memory or in the system memory based on the category of data.

16. The computer program product of claim 15, the instructions when executed configured to map the high priority traffic class to a high priority category of data and map the low priority traffic class to a low priority category of data, and selectively store data packets with a traffic class mapped to high priority category of data into the cache memory and selectively store data packets with a traffic class mapped to a low priority category of data to the system memory.

17. The computer program product of claim 15, the instructions when executed configured to map the high priority traffic class to a high priority category of data and map the low priority traffic class to a low priority category of data, and determine whether the traffic mapped to the high priority category of data can be allocated to a region of the cache memory and determine whether the traffic mapped to the low priority category of data can be allocated to a region of the cache memory.

18. The computer program product of claim 17, the instructions when executed configured to determine that the low priority category of data traffic cannot be allocated into the cache memory and store the low priority category of data traffic into the system memory.

19. The computer program product of claim 10, the instructions when executed configured to map the traffic class of the data packet to a corresponding category of data, identify a cache region of the cache memory associated with the category of data, and selectively store the data packet in the cache memory or in the system memory based on the category of data by a cache way reserved for the category of data.

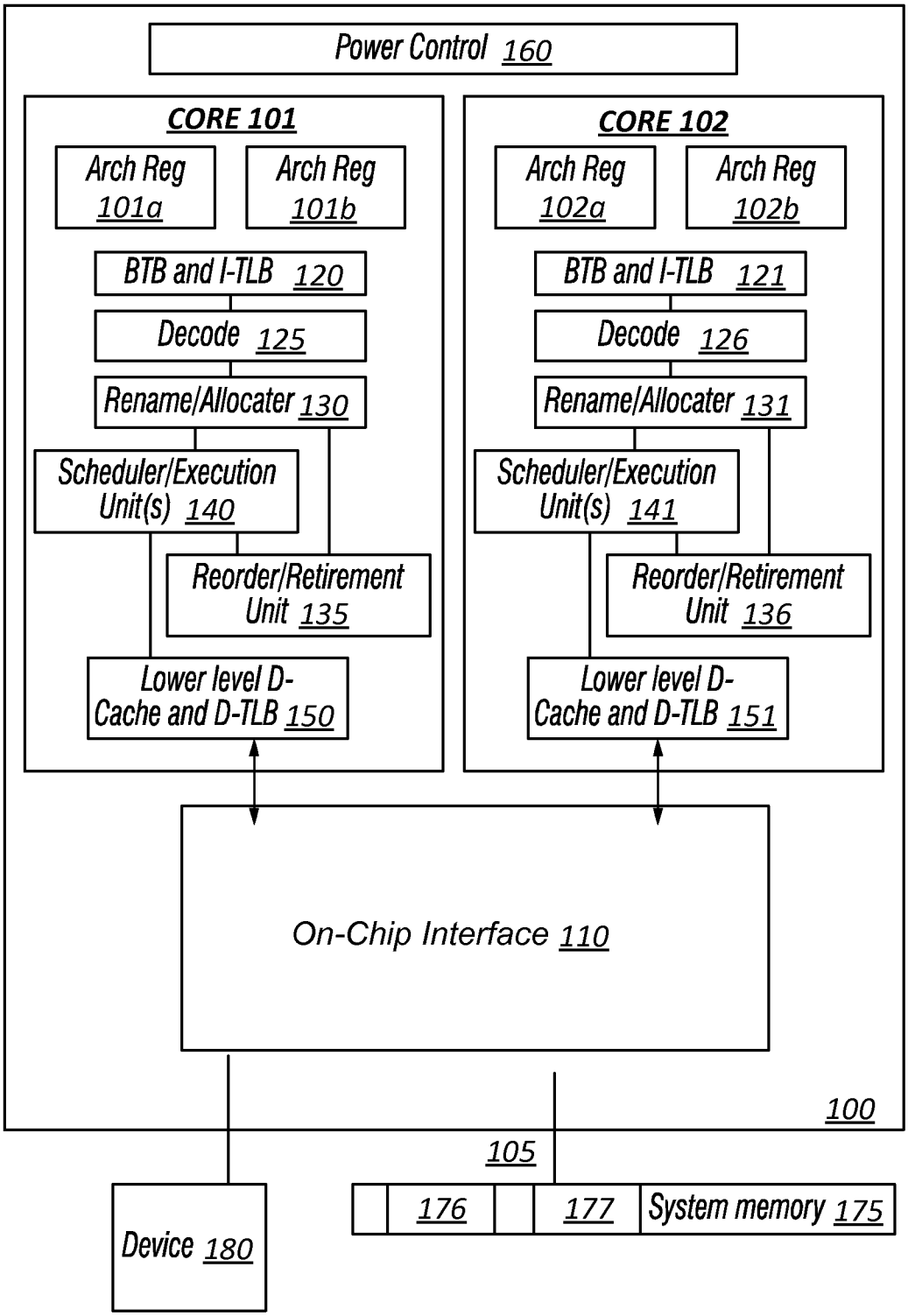

Power Control 160
CORE 101
Arch Reg 101a
Arch Reg 101b
BTB and I-TLB 120
Decode 125
Rename/Allocater 130
Scheduler/Execution Unit(s) 140
Reorder/Retirement Unit 135
Lower level D-Cache and D-TLB 150
CORE 102
Arch Reg 102a
Arch Reg 102b
BTB and I-TLB 121
Decode 126
Rename/Allocater 131
Scheduler/Execution Unit(s) 141
Reorder/Retirement Unit 136
Lower level D-Cache and D-TLB 151
On-Chip Interface 110
100
105
176
177
System memory 175
Device 180

FIG. 1

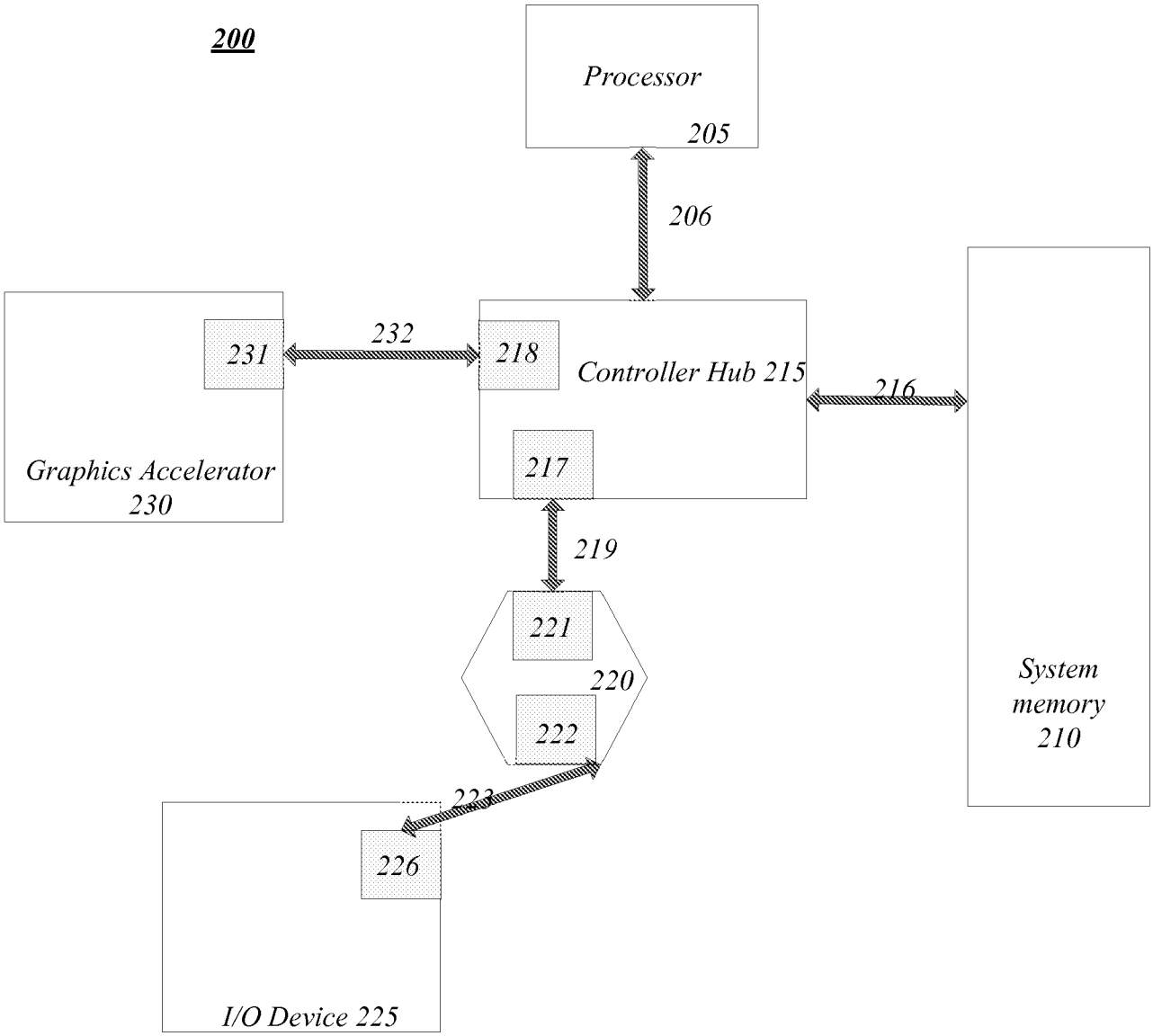

200
Processor
205
206
231
232
218
Controller Hub 215
216
Graphics Accelerator
230
217
219
221
220
222
System
memory
210
223
226
I/O Device 225

FIG. 2

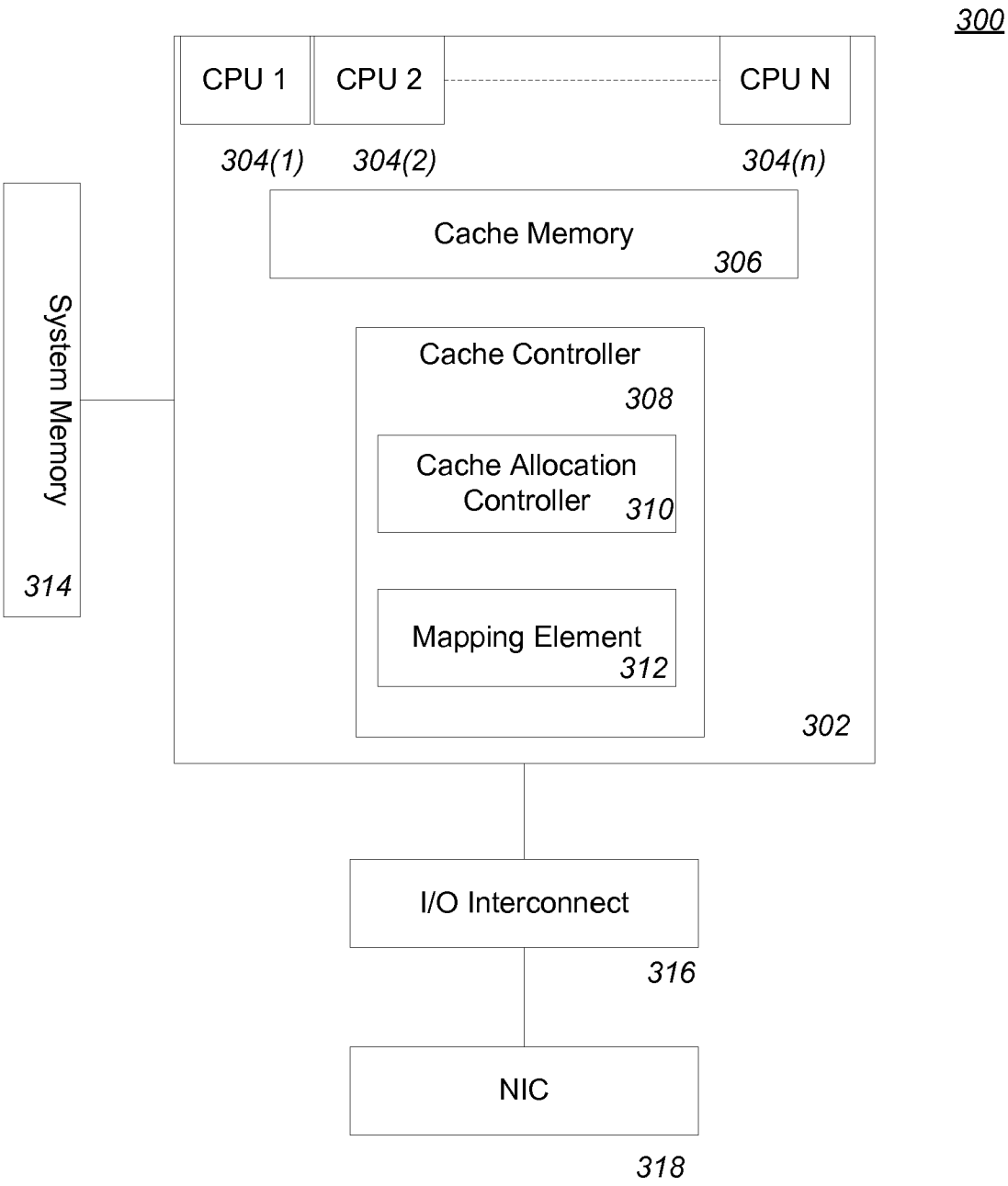

300
CPU 1
CPU 2
CPU N
304(1)
304(2)
304(n)
Cache Memory
306
System Memory
314
Cache Controller
308
Cache Allocation Controller
310
Mapping Element
312
302
I/O Interconnect
316
NIC
318

Fig. 3

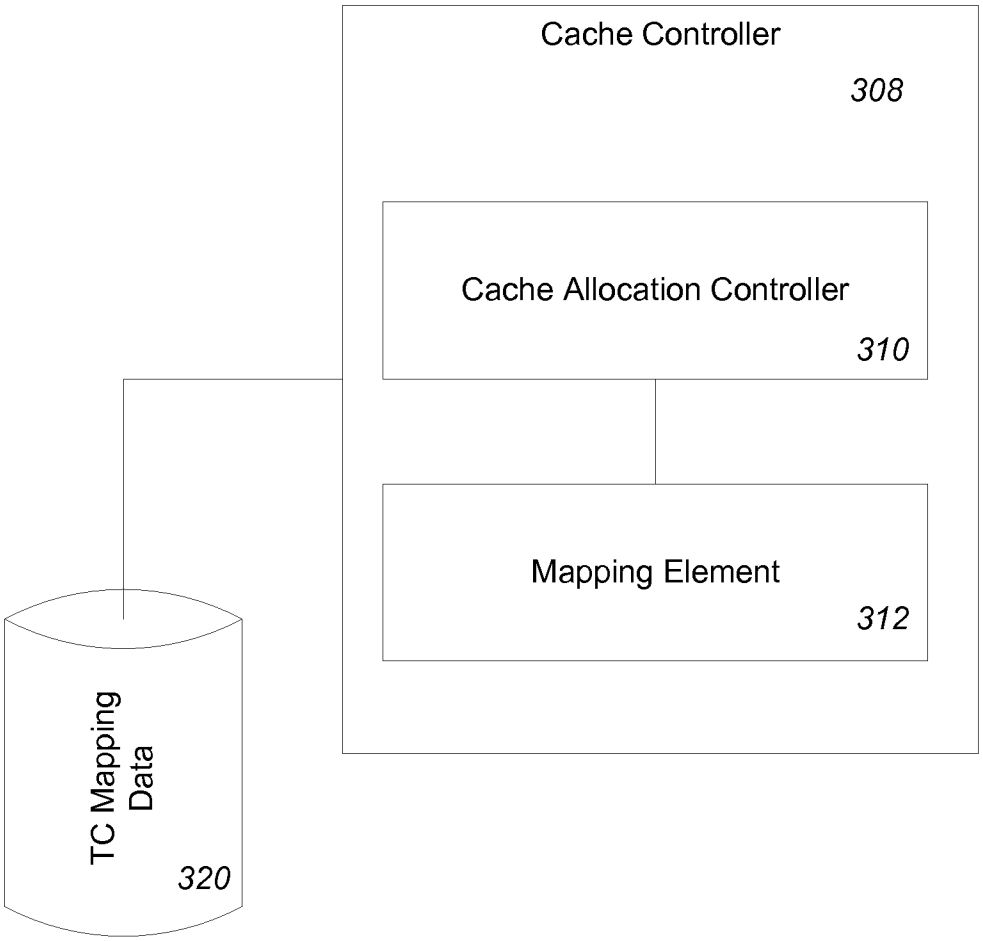

Cache Controller
308
Cache Allocation Controller
310
Mapping Element
312
TC Mapping Data
320

Fig. 4

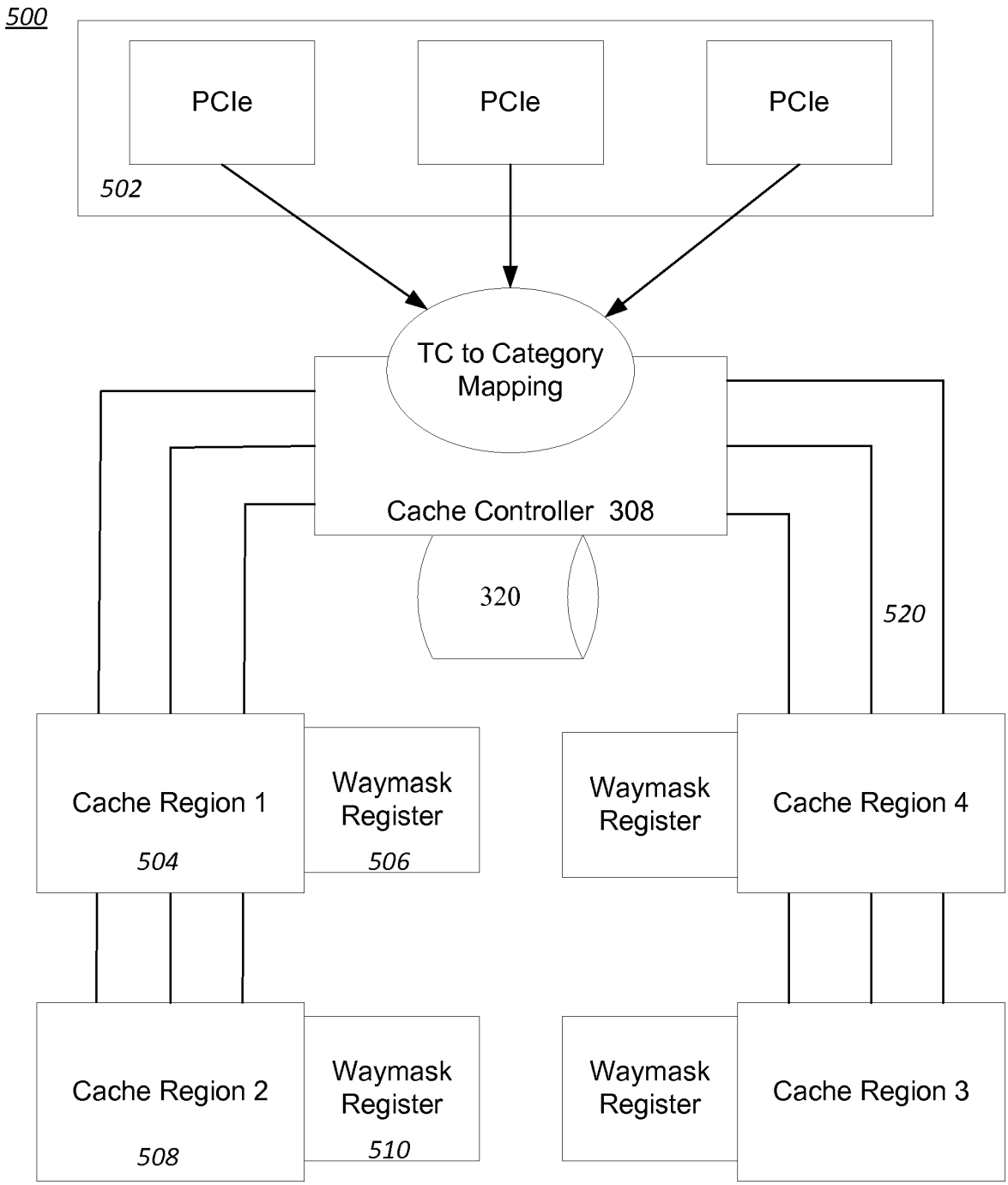

500
PCIe
PCIe
PCIe
502
TC to Category Mapping
Cache Controller 308
320
520
Cache Region 1
504
Waymask Register
506
Waymask Register
Cache Region 4
Cache Region 2
508
Waymask Register
510
Waymask Register
Cache Region 3

FIG. 5

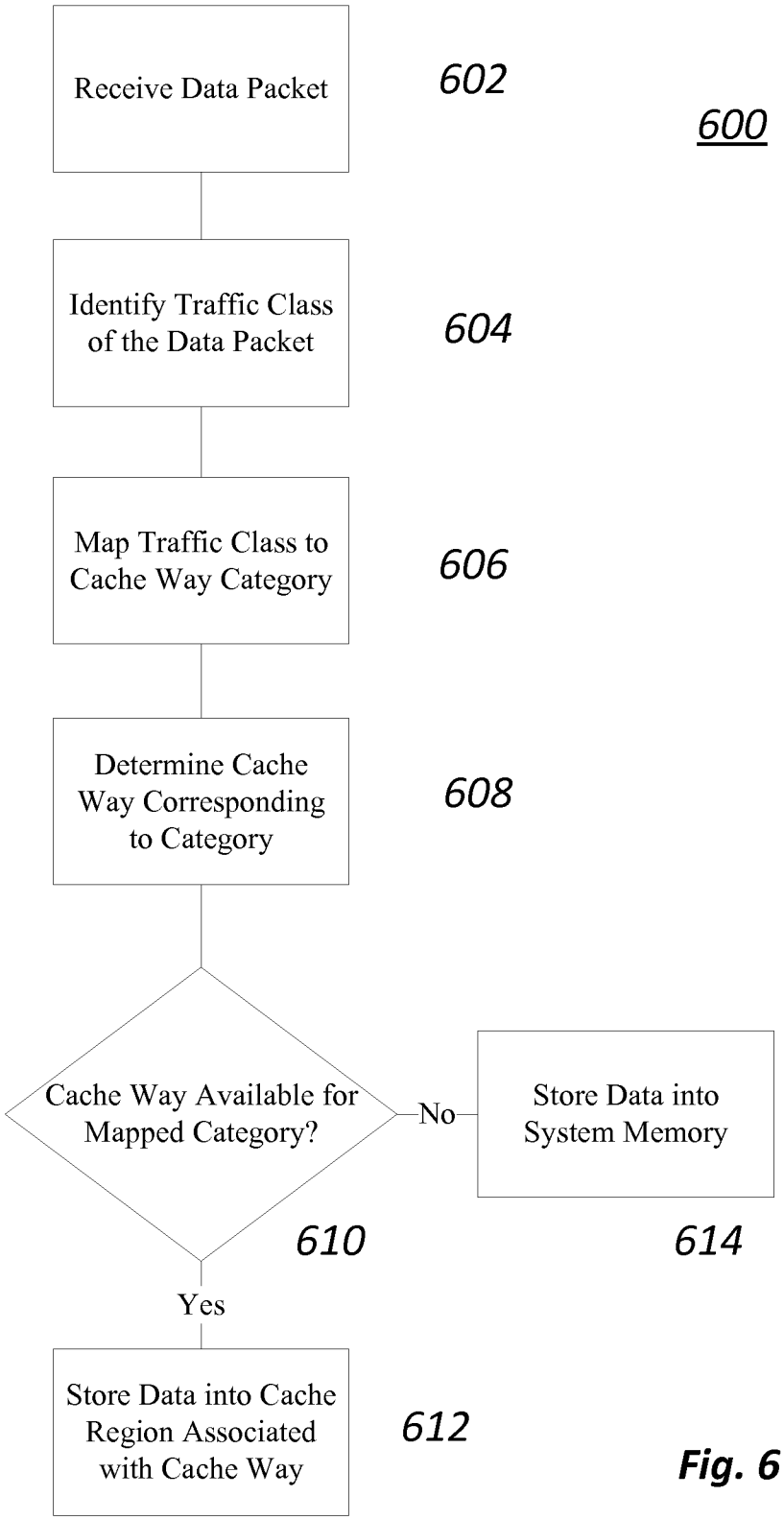

Receive Data Packet
602
600
Identify Traffic Class of the Data Packet
604
Map Traffic Class to Cache Way Category
606
Determine Cache Way Corresponding to Category
608
Cache Way Available for Mapped Category?
No
Store Data into System Memory
610
614
Yes
Store Data into Cache Region Associated with Cache Way
612

Fig. 6

Layered Protocol Stack 700

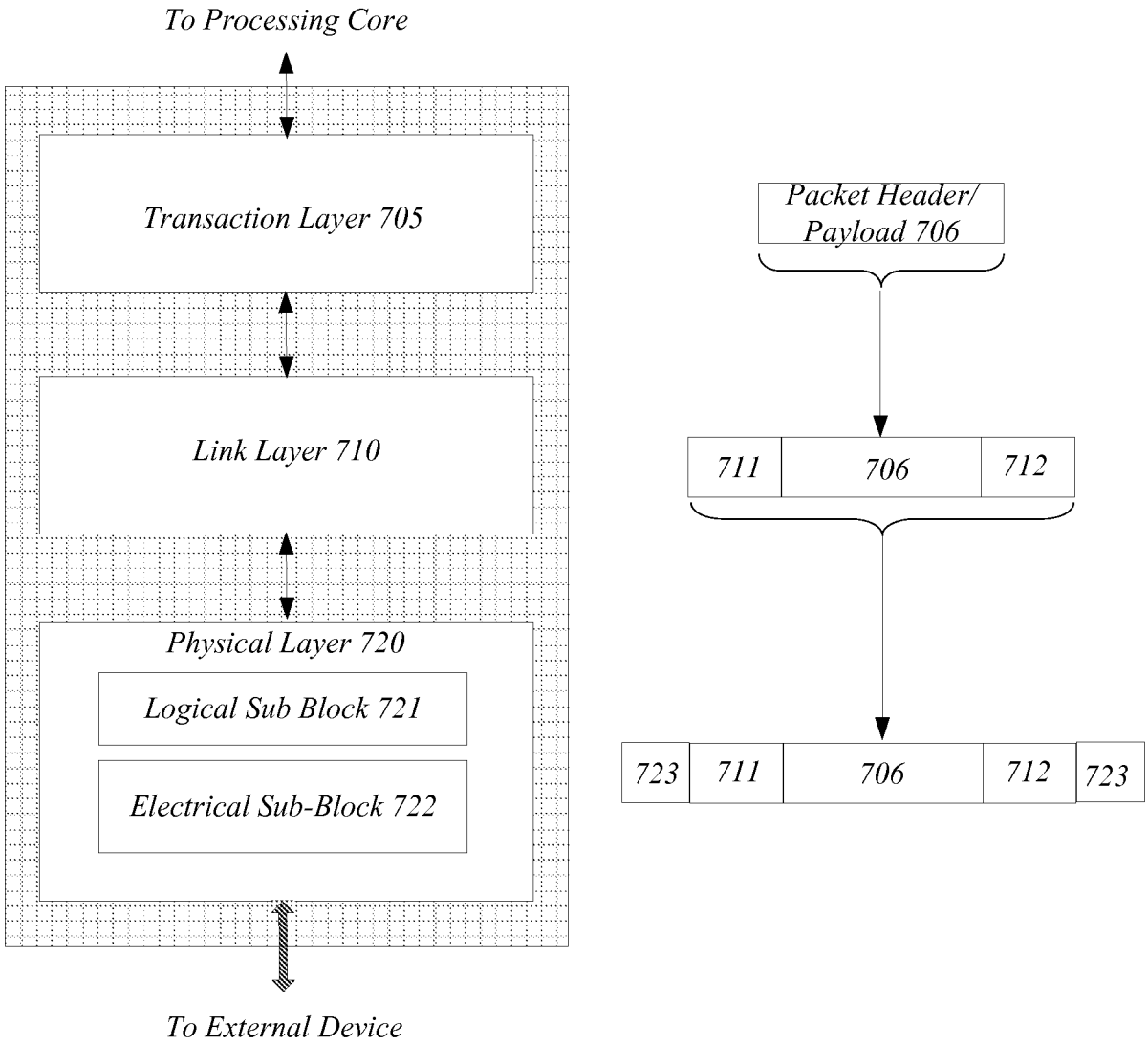


FIG. 7

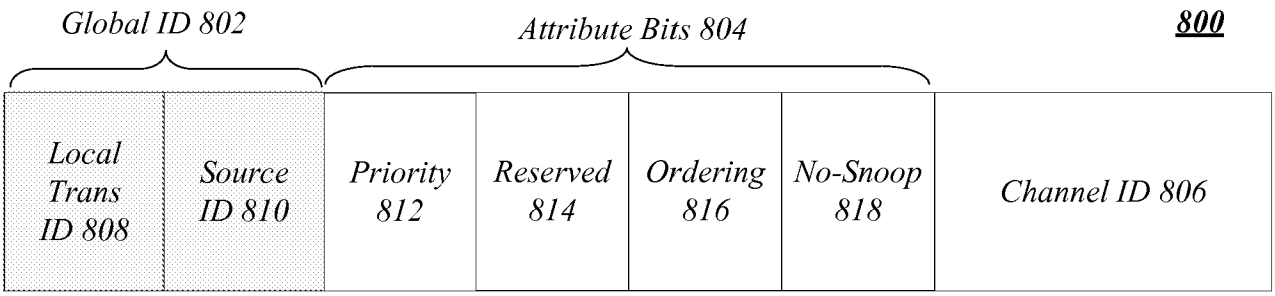

Global ID 802
Attribute Bits 804
800
Local Trans ID 808
Source ID 810
Priority 812
Reserved 814
Ordering 816
No-Snoop 818
Channel ID 806

FIG. 8

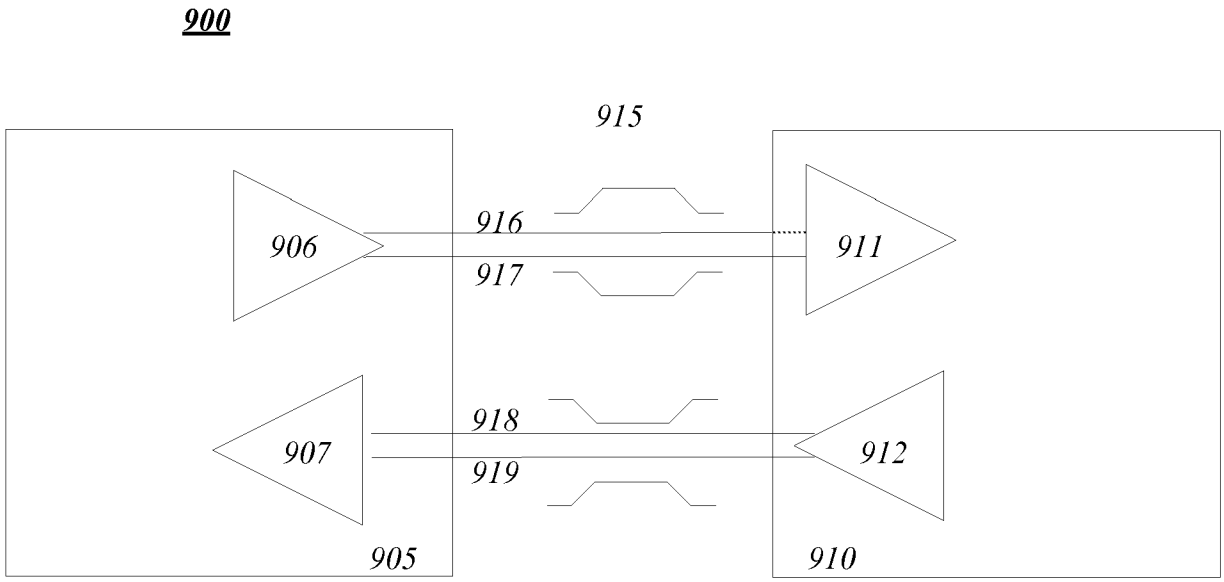

900
915
906
916
917
911
907
918
919
912
905
910

FIG. 9

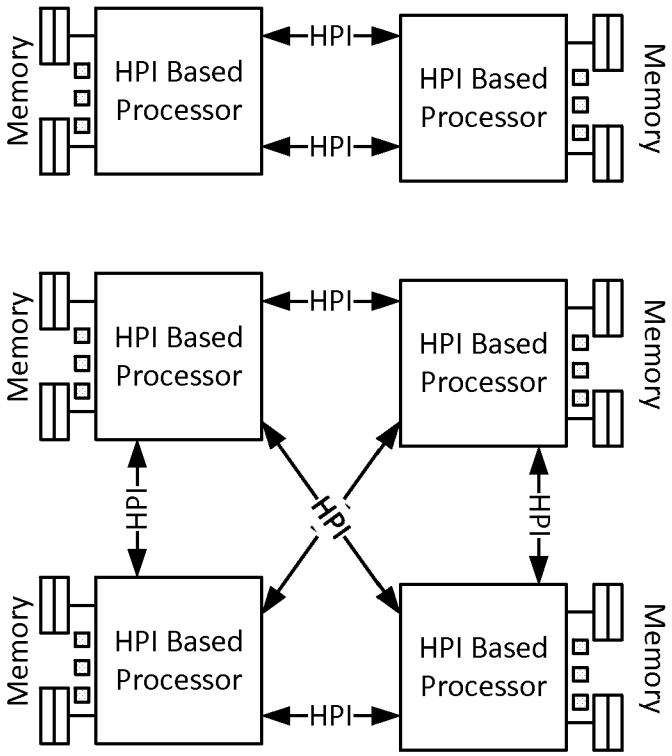
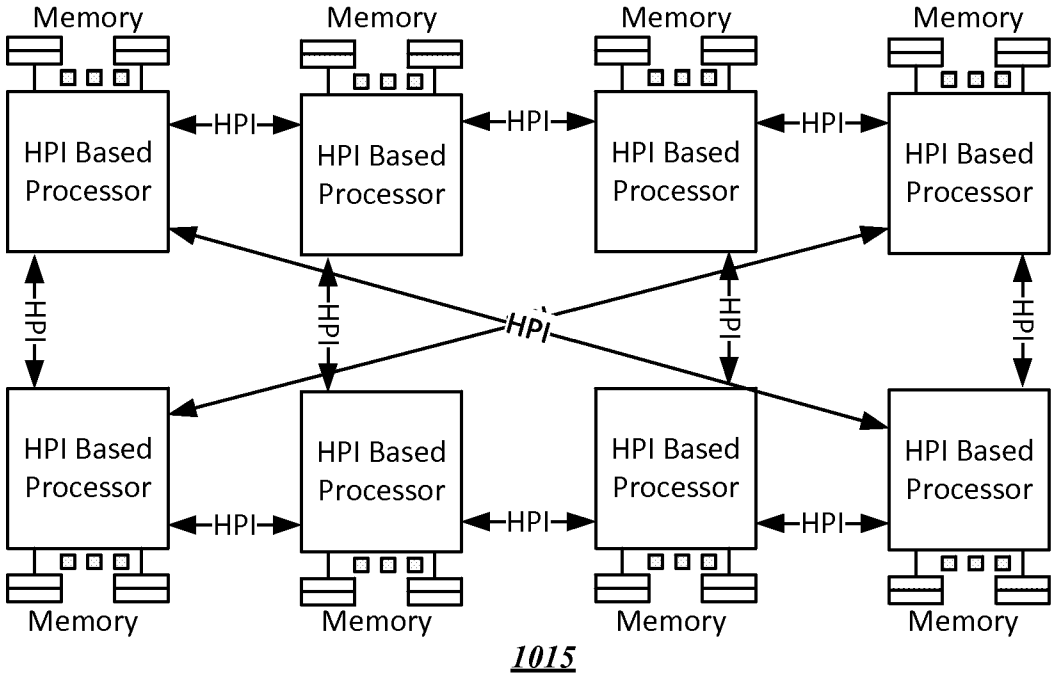

Memory
HPI Based Processor
HPI
HPI Based Processor
Memory
1005
Memory
HPI Based Processor
HPI
HPI Based Processor
Memory
HPI
HPI
HPI
Memory
HPI Based Processor
HPI Based Processor
Memory
1010

Fig. 10


Memory
Memory
Memory
Memory
HPI Based Processor
HPI
HPI Based Processor
HPI
HPI Based Processor
HPI
HPI Based Processor
HPI
HPI
HPI
HPI
HPI Based Processor
HPI Based Processor
HPI Based Processor
HPI Based Processor
HPI
HPI
HPI
Memory
Memory
Memory
Memory
1015

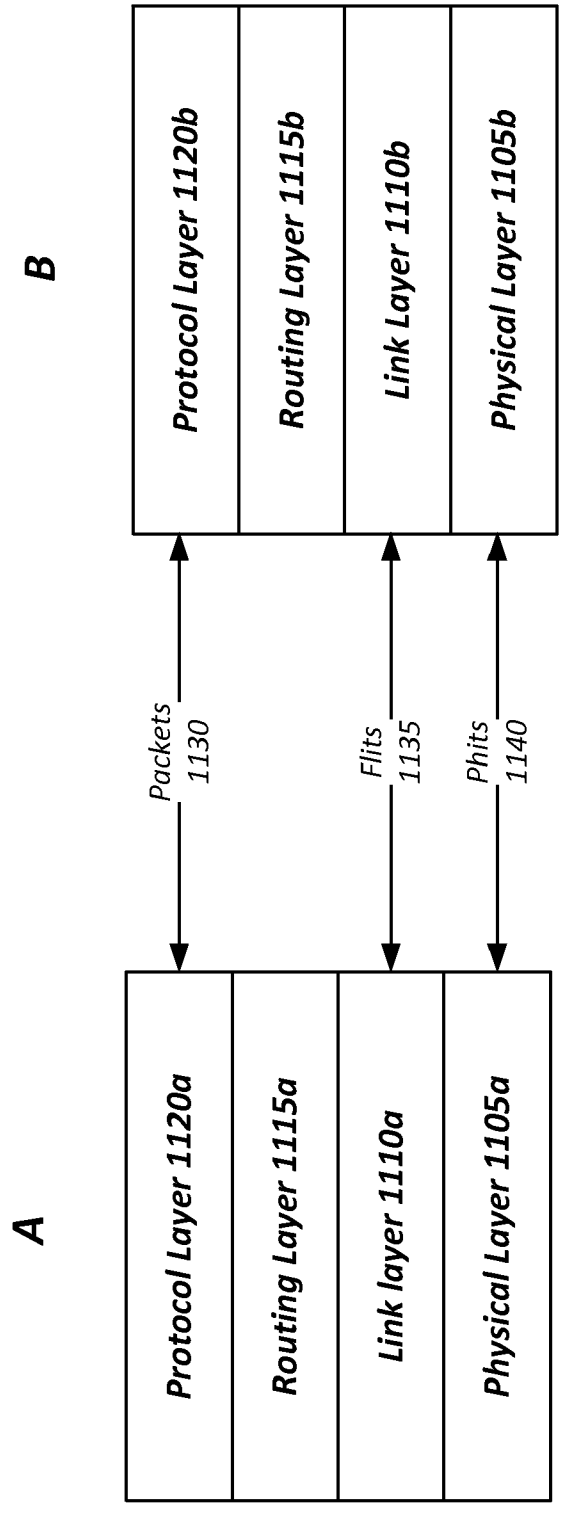

B
Protocol Layer 1120b
Routing Layer 1115b
Link Layer 1110b
Physical Layer 1105b
Packets 1130
Flits 1135
Phits 1140
A
Protocol Layer 1120a
Routing Layer 1115a
Link layer 1110a
Physical Layer 1105a

Fig. 11

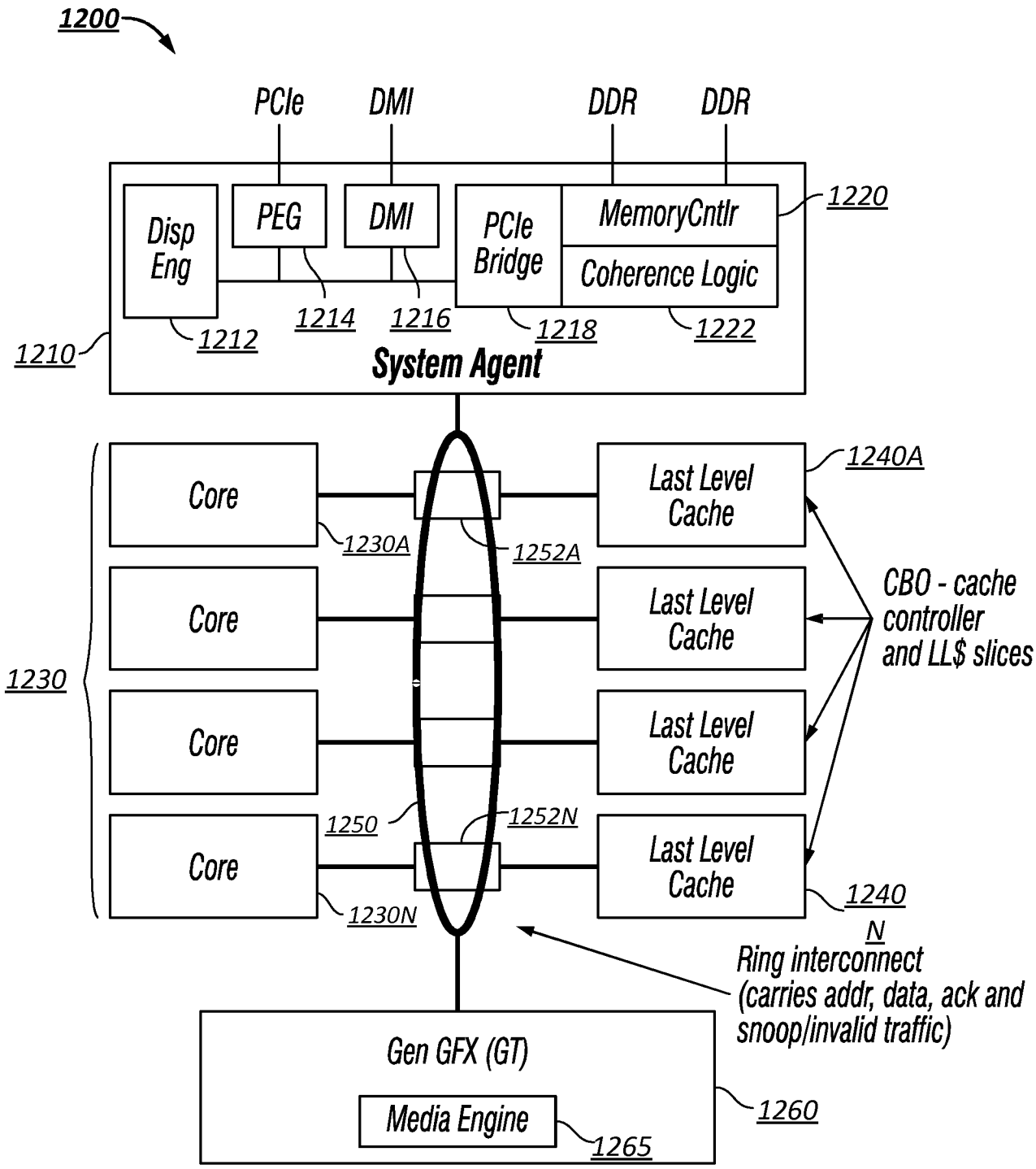

1200
PCIe
DMI
DDR
DDR
Disp Eng
PEG
DMI
PCIe Bridge
MemoryCntlr
Coherence Logic
1220
1212
1214
1216
1218
1222
1210
System Agent
Core
Core
Core
Core
1230
1230A
1230N
1250
1252A
1252N
Last Level Cache
Last Level Cache
Last Level Cache
Last Level Cache
1240A
1240 N
CBO - cache controller and LL$ slices
Ring interconnect (carries addr, data, ack and snoop/invalid traffic)
Gen GFX (GT)
Media Engine
1260
1265

FIG. 12

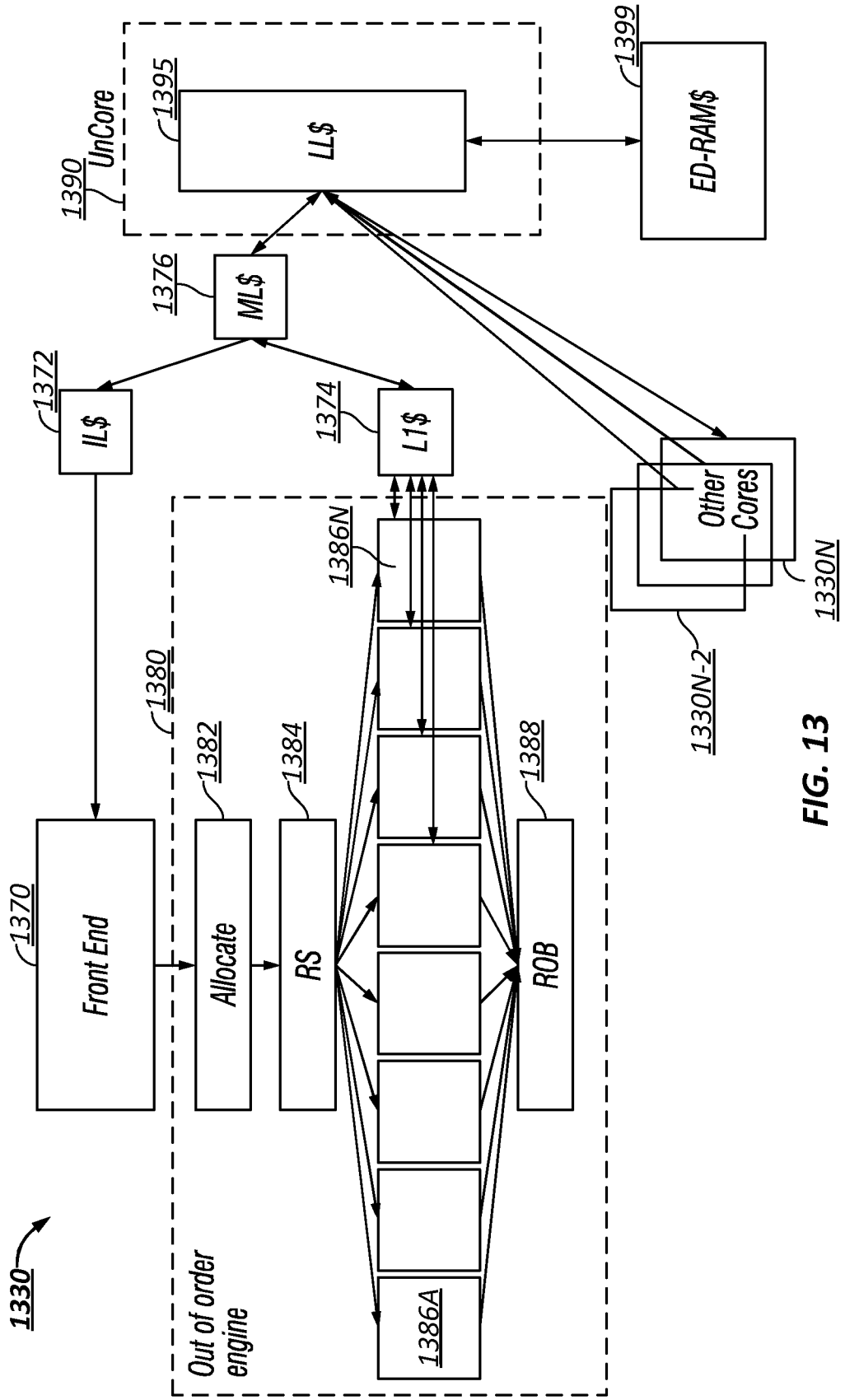

1330
Out of order engine
1370
Front End
1380
1382
Allocate
1384
RS
1386A
1386N
1388
ROB
1372
IL$
1374
L1$
1376
ML$
1390
UnCore
1395
LL$
1399
ED-RAM$
Other Cores
1330N
1330N-2

FIG. 13

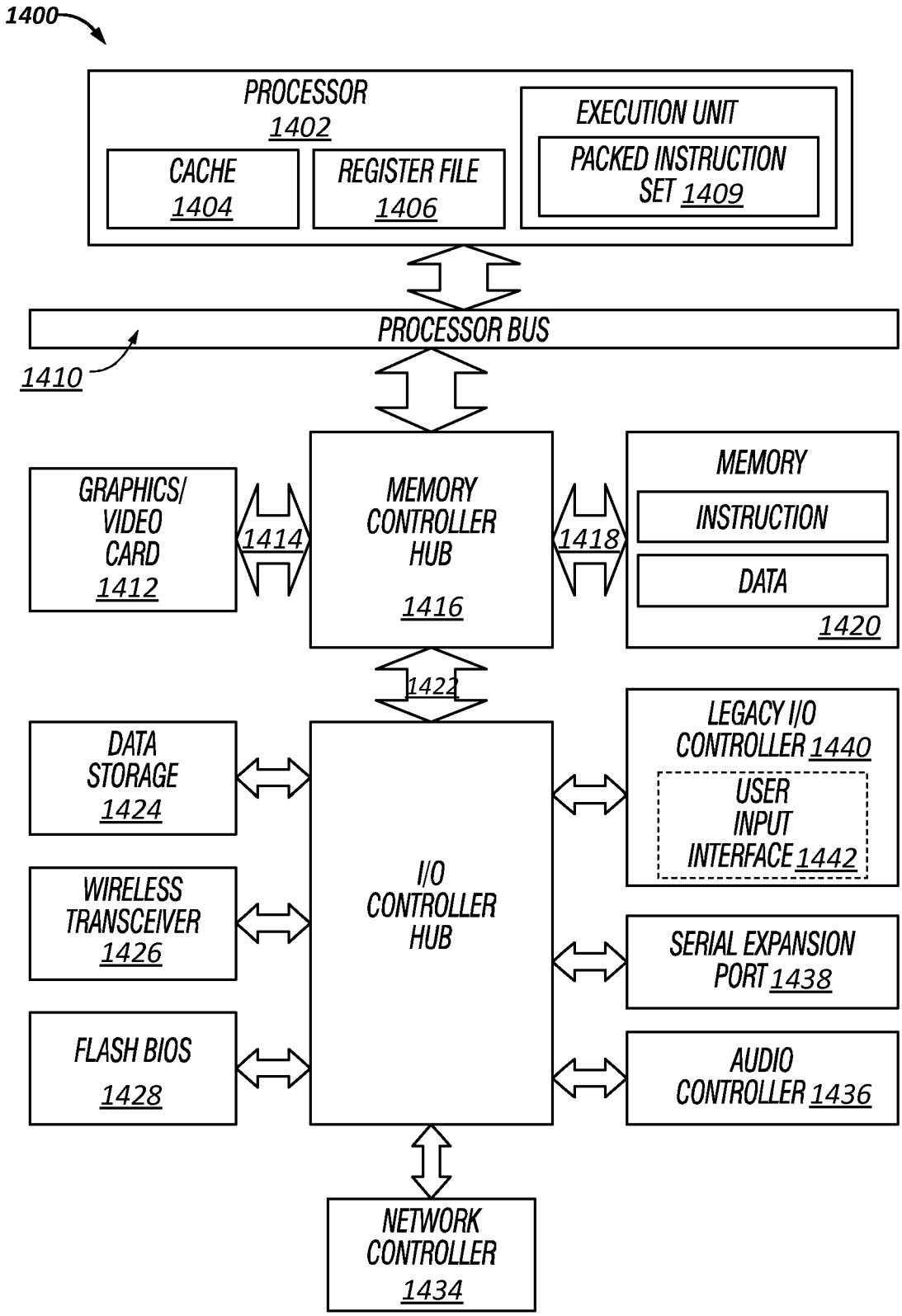

1400
PROCESSOR
1402
CACHE
1404
REGISTER FILE
1406
EXECUTION UNIT
PACKED INSTRUCTION SET 1409
PROCESSOR BUS
1410
GRAPHICS/ VIDEO CARD
1412
1414
MEMORY CONTROLLER HUB
1416
1418
MEMORY
INSTRUCTION
DATA
1420
1422
DATA STORAGE
1424
WIRELESS TRANSCEIVER
1426
FLASH BIOS
1428
I/O CONTROLLER HUB
LEGACY I/O CONTROLLER 1440
USER INPUT INTERFACE 1442
SERIAL EXPANSION PORT 1438
AUDIO CONTROLLER 1436
NETWORK CONTROLLER
1434

FIG. 14

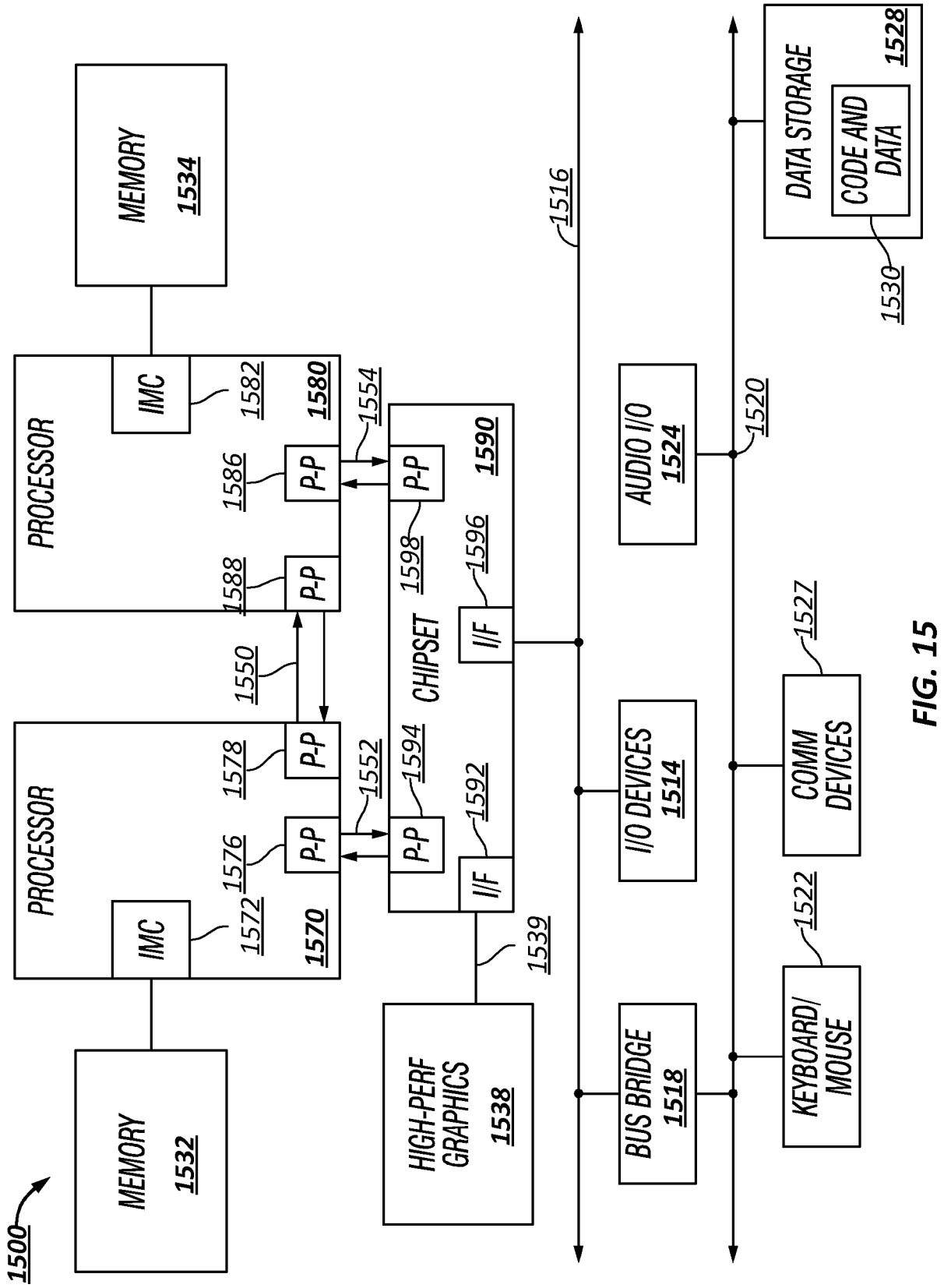

1500
MEMORY 1532
MEMORY 1534
PROCESSOR
IMC
1572
1570
1576
P-P
1578
P-P
1580
IMC
1582
1586
P-P
1588
P-P
1550
1552
1554
CHIPSET
1590
1594
P-P
1592
I/F
1598
P-P
1596
I/F
1539
HIGH-PERF GRAPHICS 1538
1516
BUS BRIDGE 1518
I/O DEVICES 1514
AUDIO I/O 1524
1520
KEYBOARD/ MOUSE
1522
COMM DEVICES
1527
DATA STORAGE
CODE AND DATA
1528
1530

FIG. 15

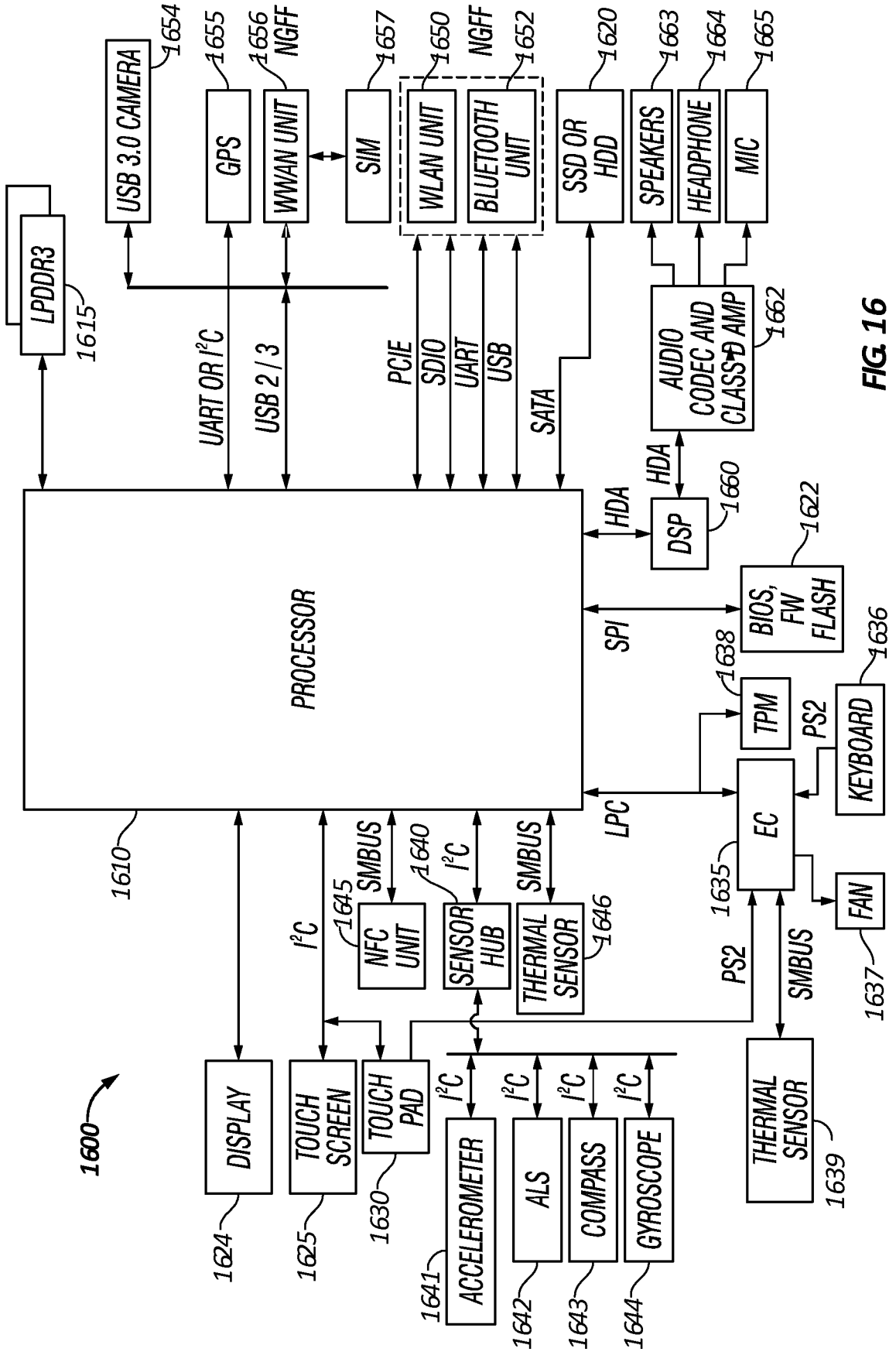

1600
PROCESSOR
1610
LPDDR3
1615
USB 3.0 CAMERA
1654
GPS
1655
WWAN UNIT
1656
NGFF
SIM
1657
WLAN UNIT
1650
NGFF
BLUETOOTH UNIT
1652
SSD OR HDD
1620
SPEAKERS
1663
HEADPHONE
1664
MIC
1665
AUDIO CODEC AND CLASS D AMP
1662
DSP
1660
BIOS, FW FLASH
1622
TPM
1638
KEYBOARD
1636
EC
1635
FAN
1637
THERMAL SENSOR
1639
NFC UNIT
1645
SENSOR HUB
1640
THERMAL SENSOR
1646
DISPLAY
1624
TOUCH SCREEN
1625
TOUCH PAD
1630
ACCELEROMETER
1641
ALS
1642
COMPASS
1643
GYROSCOPE
1644
UART OR I²C
USB 2 / 3
PCIE
SDIO
UART
USB
SATA
HDA
HDA
SPI
LPC
PS2
PS2
SMBUS
SMBUS
SMBUS
I²C
I²C
I²C
I²C
I²C

FIG. 16

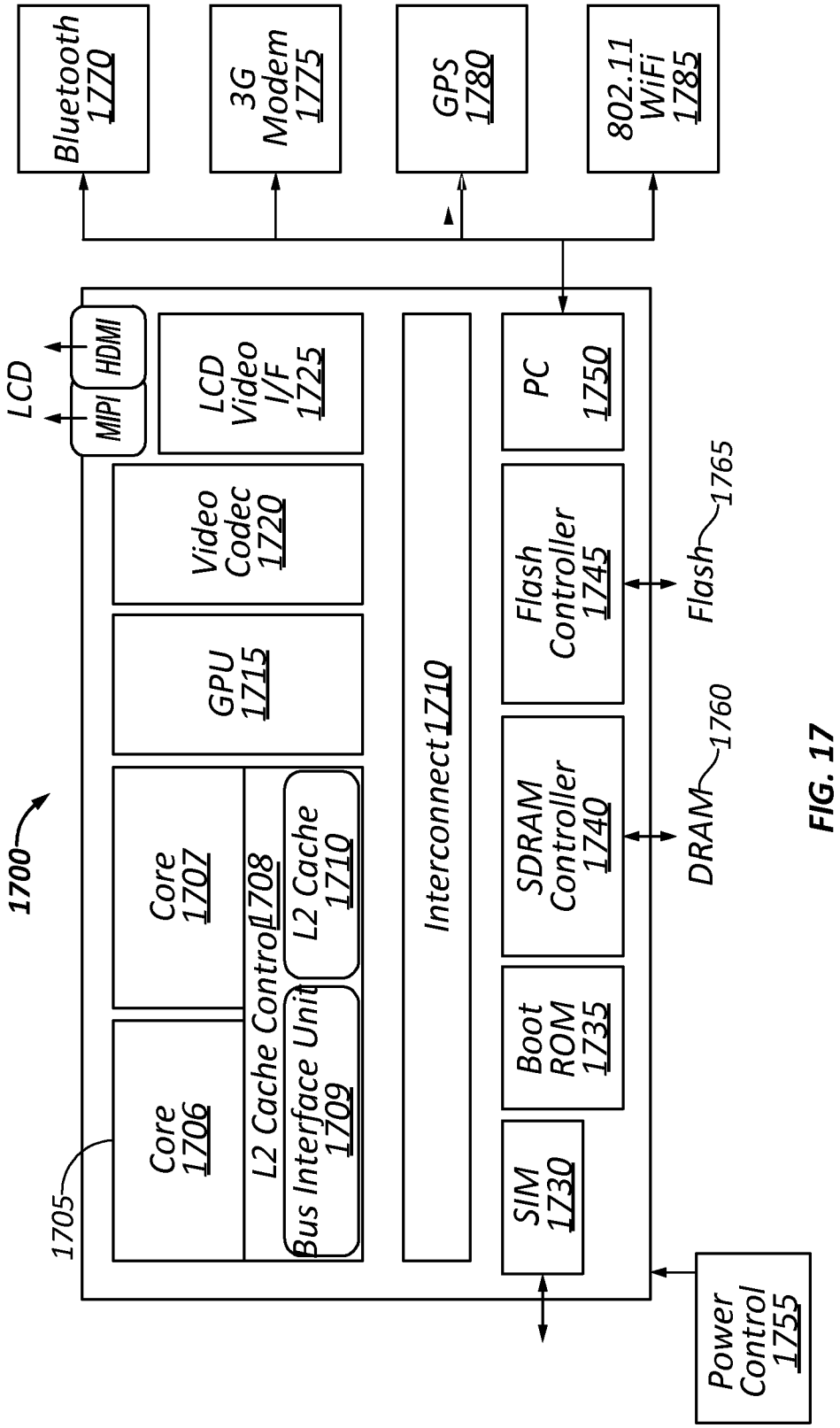

Bluetooth 1770
3G Modem 1775
GPS 1780
802.11 WiFi 1785
LCD
HDMI
MIPI
LCD Video I/F 1725
PC 1750
Video Codec 1720
Flash Controller 1745
Flash
1765
GPU 1715
Interconnect 1710
SDRAM Controller 1740
DRAM
1760
Core 1707
L2 Cache Control 1708
L2 Cache 1710
Bus Interface Unit 1709
Core 1706
Boot ROM 1735
SIM 1730
1705
1700
Power Control 1755

FIG. 17

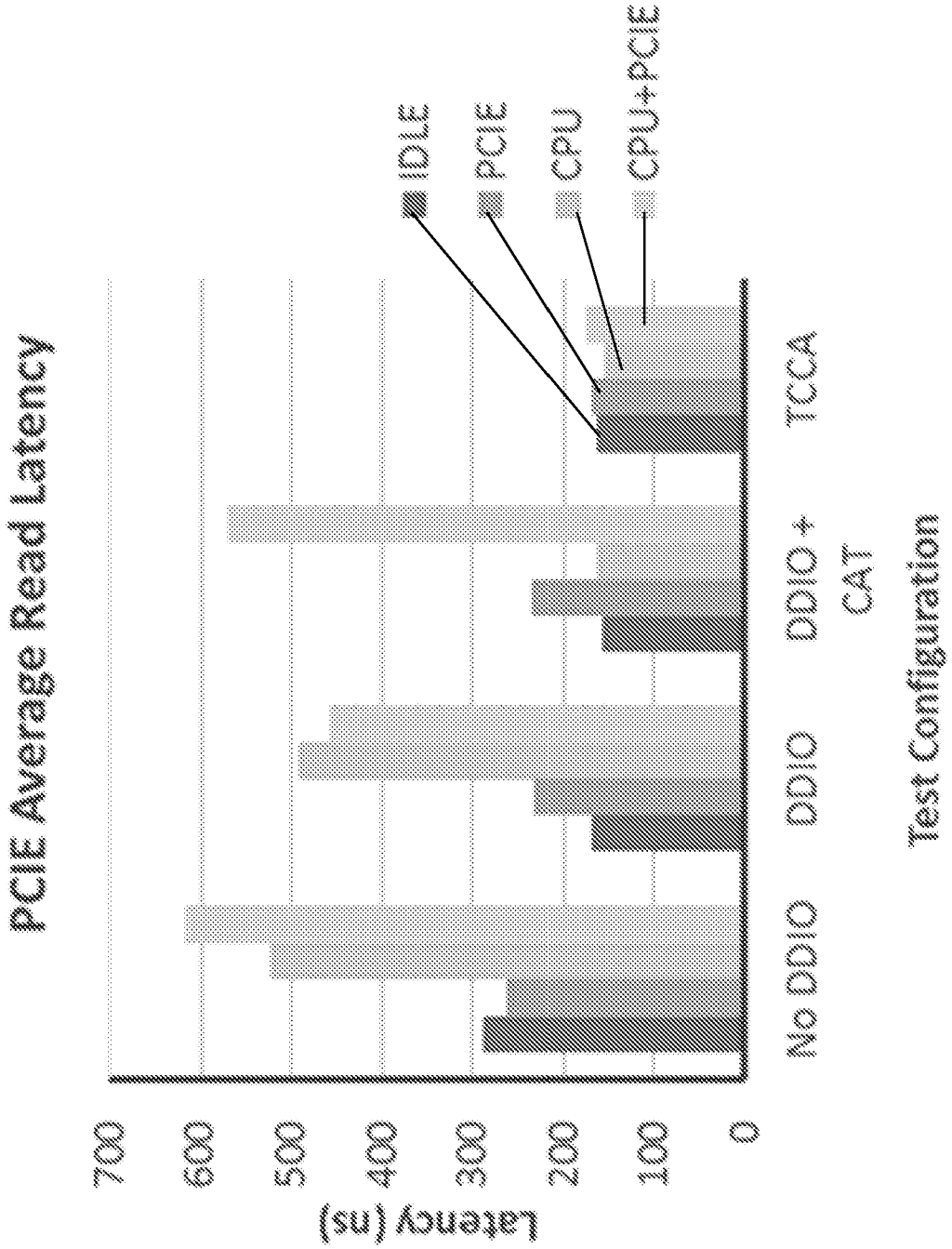

PCIE Average Read Latency
IDLE
PCIE
CPU
CPU+PCIE
Latency (ns)
0
100
200
300
400
500
600
700
No DDIO
DDIO
DDIO + CAT
TCCA
Test Configuration

FIG. 18

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US2016/048256

A. CLASSIFICATION OF SUBJECT MATTER

G06F 13/42(2006.01)i, G06F 13/40(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 13/42; H04L 12/26; G06F 13/14; G06F 13/36; G06F 13/24; H04L 12/56; G06F 13/40

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched
Korean utility models and applications for utility models
Japanese utility models and applications for utility models

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
eKOMPASS(KIPO internal) & Keywords: data packet, traffic class, cache memory, system memory, allocate, priority, pcie, and similar terms.

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	US 2008-0063004 A1 (KEVIN D. HIMBERGER et al.) 13 March 2008 See paragraphs [0008], [0010]-[0012], [0018], [0020]-[0022], and [0025]-[0028]; and figures 3-4.	1-8,10-19
A		9
Y	US 2014-0233383 A1 (IFAT NAAMAN et al.) 21 August 2014 See paragraphs [0006], [0028]-[0029], and [0034]-[0039]; claim 1; and figures 1 and 4.	1-8,10-19
A	US 2007-0112995 A1 (BRIAN EDWARD MANULA et al.) 17 May 2007 See paragraphs [0006]-[0007] and [0090]-[0101]; and figure 12.	1-19
A	US 2009-0161684 A1 (SREENIVAS VORUGANTI et al.) 25 June 2009 See paragraphs [0004]-[0006] and [0025]-[0027]; and figures 3-5.	1-19
A	WO 2012-134638 A1 (INTEL CORPORATION) 04 October 2012 See page 2, line 26 - page 4, line 14 and figure 5.	1-19

☐ 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
28 November 2016 (28.11.2016)	**28 November 2016 (28.11.2016)**

Name and mailing address of the ISA/KR
International Application Division
Korean Intellectual Property Office
189 Cheongsa-ro, Seo-gu, Daejeon, 35208, Republic of Korea
Facsimile No. +82-42-481-8578

Authorized officer
NHO, Ji Myong
Telephone No. +82-42-481-8528

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No.

PCT/US2016/048256

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2008-0063004 A1	13/03/2008	None	
US 2014-0233383 A1	21/08/2014	US 9455913 B2	27/09/2016
US 2007-0112995 A1	17/05/2007	US 7424566 B2	09/09/2008
US 2009-0161684 A1	25/06/2009	US 2012-275464 A1 US 8160085 B2 US 8848724 B2	01/11/2012 17/04/2012 30/09/2014
WO 2012-134638 A1	04/10/2012	CN 103460199 A CN 103460199 B US 2012-0254492 A1 US 8566494 B2	18/12/2013 17/02/2016 04/10/2012 22/10/2013