US 20140223052A1

(19) United States
(12) Patent Application Publication (10) Pub. No.: US 2014/0223052 A1
Chavali et al. (43) Pub. Date: Aug. 7, 2014

(54) SYSTEM AND METHOD FOR SLAVE-BASED MEMORY PROTECTION

(71) Applicant: **TEXAS INSTRUMENTS INCORPORATED**, Dallas, TX (US)

(72) Inventors: **Balatripura Sodemma Chavali**, Sugar Land, TX (US); **Karl Fredrich Greb**, Sugarland, TX (US); **Rajeev Suvarna**, Bangalore (IN)

(73) Assignee: **TEXAS INSTRUMENTS INCORPORATED**, Dallas, TX (US)

(21) Appl. No.: **14/015,690**

(22) Filed: **Aug. 30, 2013**

Related U.S. Application Data

(60) Provisional application No. 61/762,212, filed on Feb. 7, 2013.

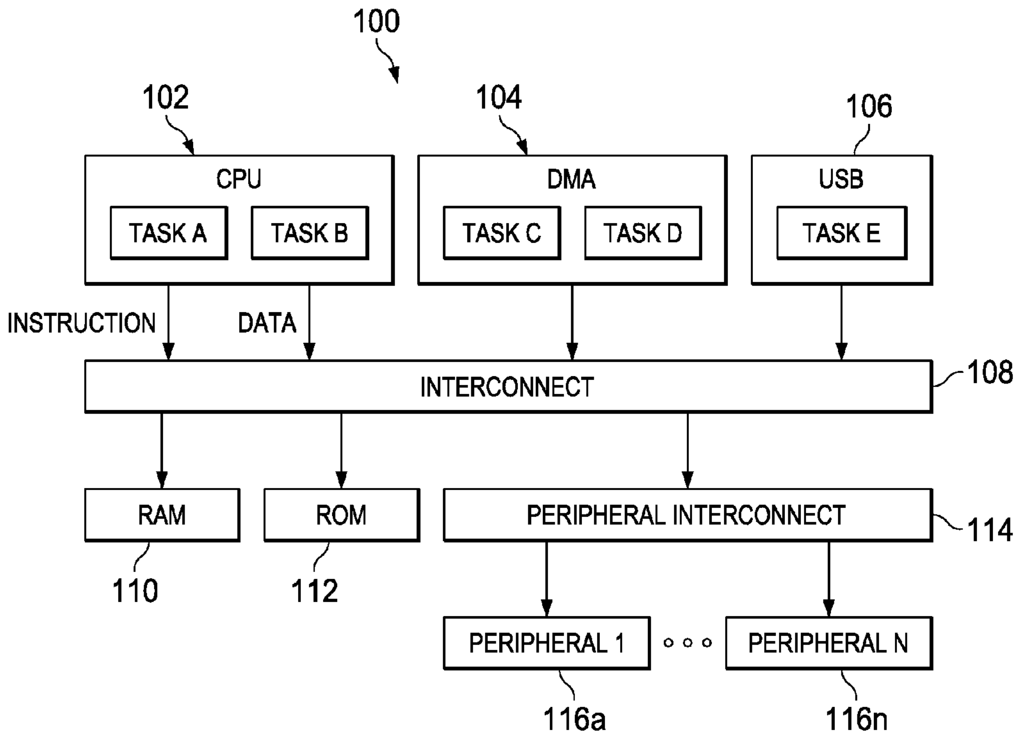
Publication Classification

(51) **Int. Cl.**
G06F 13/16 (2006.01)

(52) **U.S. Cl.**
CPC *G06F 13/1605* (2013.01)
USPC **710/110**

(57) **ABSTRACT**

A system includes a bus slave coupled to a plurality of bus masters via one or more interconnects. The system also includes a memory protection unit (MPU) associated with the bus slave, the MPU having a set of access permissions that grants access to the bus slave from a first bus master and denies access to the bus slave from a second bus master. The MPU generates an error response as result of a transaction generated by a task on the second bus master attempting to access the bus slave.



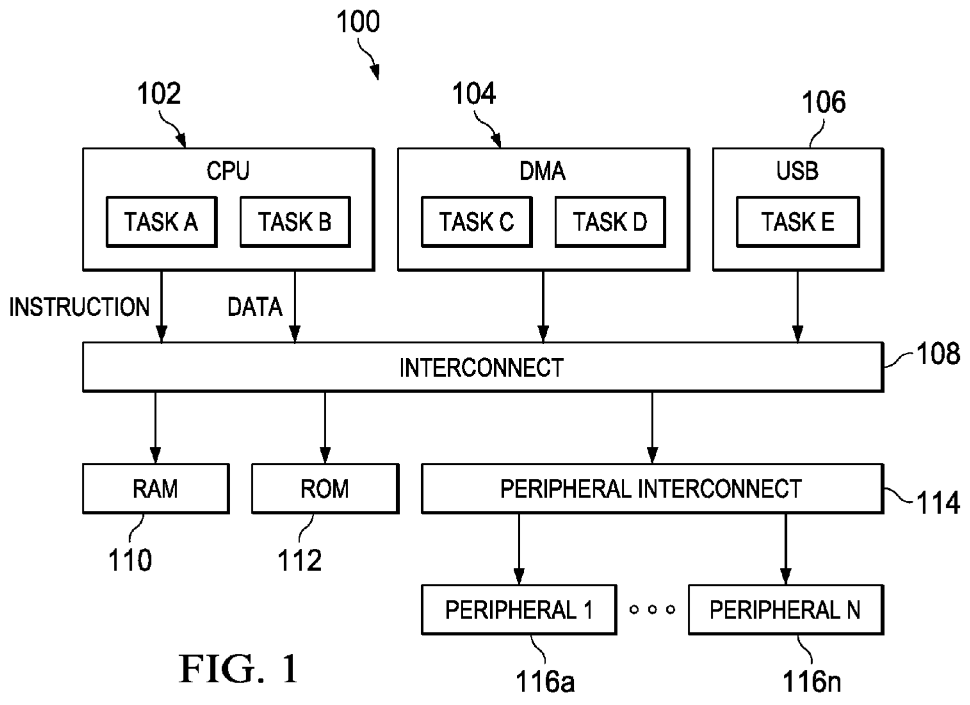


FIG. 1

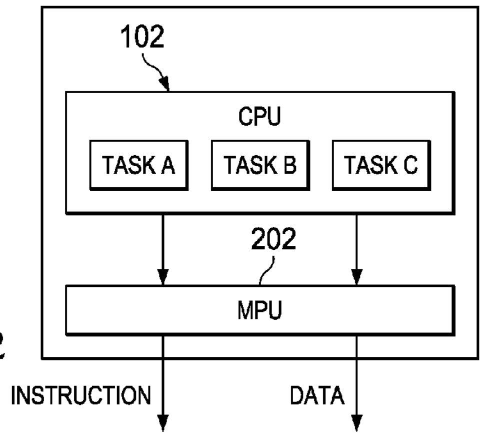


FIG. 2

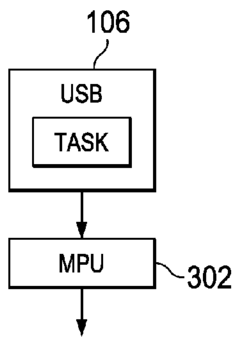


FIG. 3

FIG. 4

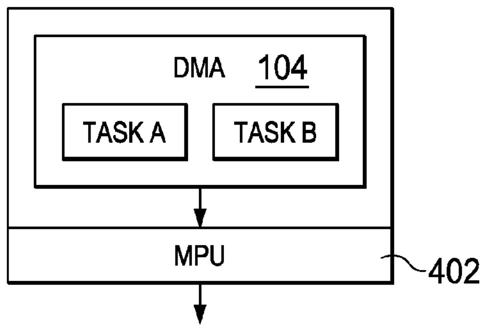
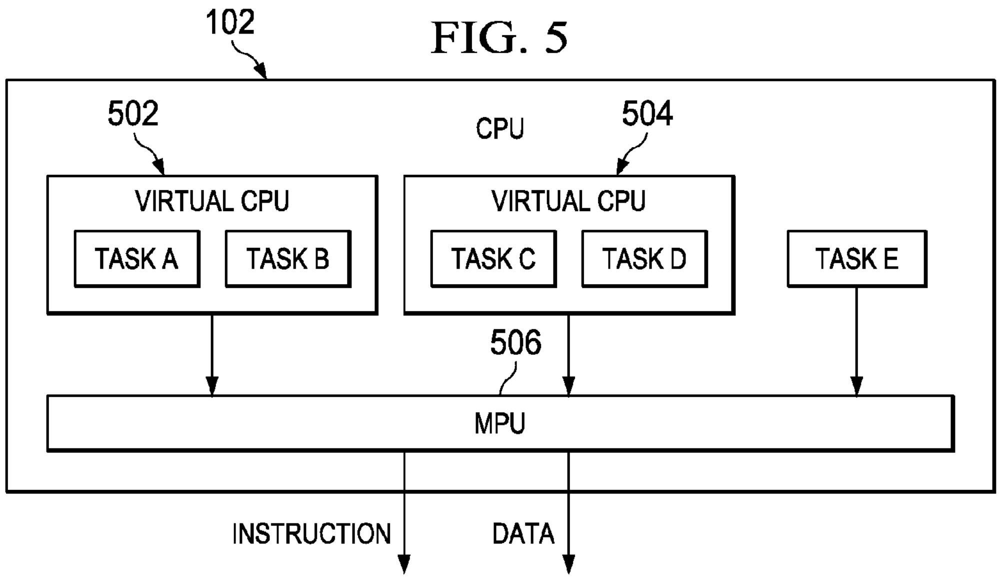

DMA 104
TASK A
TASK B
MPU
402

FIG. 5


102
502
CPU
504
VIRTUAL CPU
TASK A
TASK B
VIRTUAL CPU
TASK C
TASK D
TASK E
506
MPU
INSTRUCTION
DATA

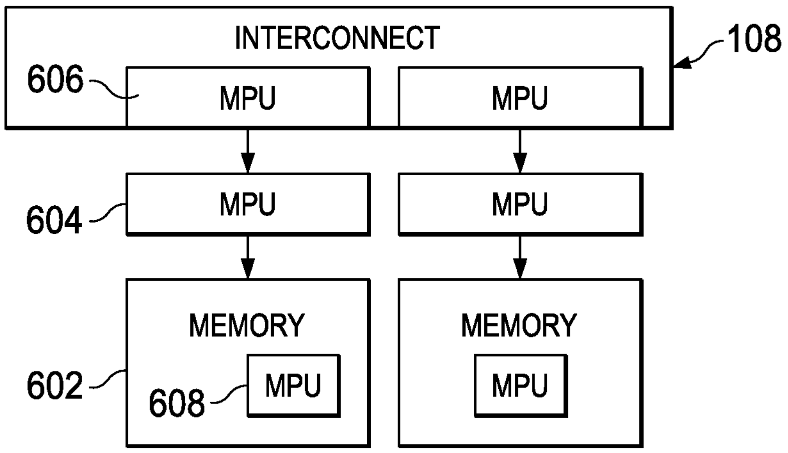

INTERCONNECT
108
606
MPU
MPU
604
MPU
MPU
602
MEMORY
608
MPU
MEMORY
MPU

FIG. 6

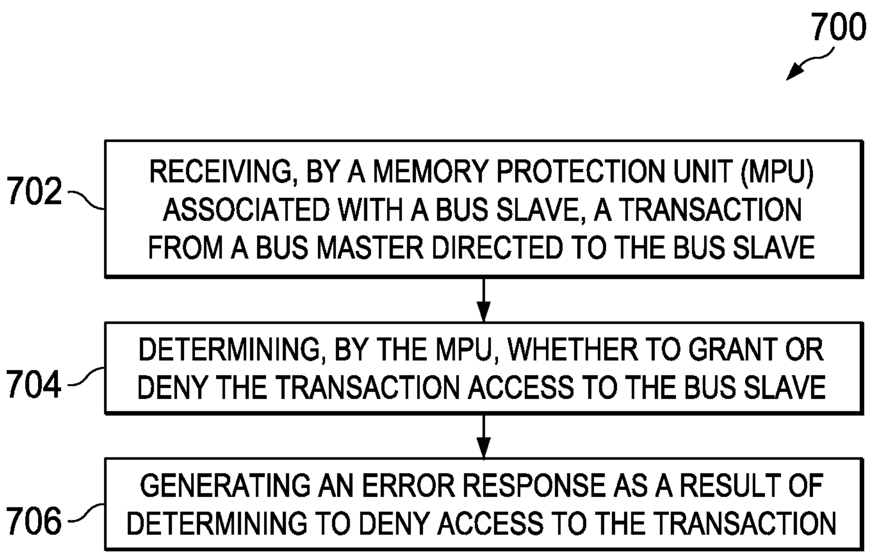

700
702
RECEIVING, BY A MEMORY PROTECTION UNIT (MPU) ASSOCIATED WITH A BUS SLAVE, A TRANSACTION FROM A BUS MASTER DIRECTED TO THE BUS SLAVE
704
DETERMINING, BY THE MPU, WHETHER TO GRANT OR DENY THE TRANSACTION ACCESS TO THE BUS SLAVE
706
GENERATING AN ERROR RESPONSE AS A RESULT OF DETERMINING TO DENY ACCESS TO THE TRANSACTION

FIG. 7

SYSTEM AND METHOD FOR SLAVE-BASED MEMORY PROTECTION

CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] The present application claims priority to U.S. Provisional Patent Application No. 61/762,212, filed on Feb. 7, 2013 (Attorney Docket No. TI-73288PS); which is hereby incorporated herein by reference. The present application is also related to co-pending U.S. patent application Ser. No. 14/015,561 (Attorney Docket No. 1962-85400, Titled "System And Method For Per-Task Memory Protection For A Non-Programmable Bus Master), which is hereby incorporated herein by reference in its entirety.

BACKGROUND

[0002] Various processes are governed by international standards relating to safety and risk reduction. For example, IEC 61508 addresses functional safety of electrical, electronic, and programmable electronic devices, such as microcontrollers or other computers used to control industrial or other safety critical processes. IEC 61508 defines Safety Integrity Levels (SIL) based on a probabilistic analysis of a particular application. To achieve a given SIL, the application, including constituent components, must meet targets for the maximum probability of "dangerous failure" and a minimum "safe failure fraction." The concept of "dangerous failure" is defined on an application-specific basis, but is based on requirement constraints that are verified for their integrity during the development of the safety critical application. The "safe failure fraction" determines capability of the system to manage dangerous failures and compares the likelihood of safe and detected failures with the likelihood of dangerous, undetected failures. Ultimately, an electronic device's certification to a particular SIL requires that the electronic device provide a certain level of detection of and resilience to failures as well as enable the safety critical application to transition to a safe state after a failure.

[0003] Another functional safety standard is ISO 26262, which addresses the functional safety of road vehicles such as automobiles. ISO 26262 aims to address possible hazards caused by malfunctioning behavior of automotive electronic and electrical systems. Similar to SILs defined by IEC 61508, ISO 26262 provides an automotive-specific risk-based approach to determine risk classes referred to as Automotive Safety Integrity Levels (ASIL). ASILs are used to specify a particular product's ability to achieve acceptable safety goals.

[0004] An electronic device that controls a process—industrial, automotive, or otherwise—may be used to perform multiple functions, some of which are "safety functions" while others are "non-safety functions." A safety function is a function whose operation impacts the safety of the process; for example, a closed-loop control system that drives an electric motor used for power steering is a safety function. A non-safety function is a function whose operation does not impact the safety of the process; for example, debug functionality built into the electronic device that is used to develop software for the control functions, but is not used when the electronic device is integrated into a vehicle, is a non-safety function.

SUMMARY

[0005] The problems noted above are solved in large part by a system including a bus slave coupled to a plurality of bus masters via one or more interconnects. The system also includes a memory protection unit (MPU) associated with the bus slave, the MPU having a set of access permissions that grants access to the bus slave from a first bus master and denies access to the bus slave from a second bus master. The MPU generates an error response as result of a transaction generated by a task on the second bus master attempting to access the bus slave.

[0006] Other embodiments of the present disclosure are directed to a method including receiving a transaction from a bus master directed at a bus slave, determining whether to grant or deny the transaction access to the bus slave, and generating an error response as a result of determining to deny access to the transaction.

[0007] Still other embodiments of the present disclosure are directed to an electronic device including a bus slave that is memory or a peripheral and first and second bus masters to execute one or more tasks. Each task generates transactions directed at the bus slave. The device also includes an interconnect to couple the bus slave to the bus master and a memory protection unit (MPU) associated with the bus slave. The MPU has a set of access permissions that grants access to the bus slave from the first bus master and denies access to the bus slave from the second bus master. The MPU generates an error response as result of a transaction generated by a task on the second bus master attempting to access the bus slave.

BRIEF DESCRIPTION OF THE DRAWINGS

[0008] For a detailed description of exemplary embodiments of the invention, reference will now be made to the accompanying drawings in which:

[0009] FIG. **1** shows a block diagram of an exemplary system on a chip (SOC) architecture in accordance with various embodiments;

[0010] FIG. **2** shows a block diagram of an exemplary memory protection unit (MPU) in conjunction with a multiple-task bus master in accordance with various embodiments;

[0011] FIG. **3** shows a block diagram of an exemplary MPU in conjunction with a single-task bus master in accordance with various embodiments;

[0012] FIG. **4** shows a block diagram of an exemplary direct memory access (DMA) controller in conjunction with a multiple-task bus master in accordance with various embodiments;

[0013] FIG. **5** shows a block diagram of an exemplary MPU in conjunction with a multiple-task bus master with a virtualized hardware scheme in accordance with various embodiments;

[0014] FIG. **6** shows a block diagram of multiple exemplary MPUs for slave-based memory protection in accordance with various embodiments; and

[0015] FIG. **7** shows a flow chart of a method in accordance with various embodiments.

NOTATION AND NOMENCLATURE

[0016] Certain terms are used throughout the following description and claims to refer to particular system components. As one skilled in the art will appreciate, companies may refer to a component by different names. This document does not intend to distinguish between components that differ in name but not function. In the following discussion and in the claims, the terms "including" and "comprising" are used in an open-ended fashion, and thus should be interpreted to mean "including, but not limited to . . . ." Also, the term "couple" or "couples" is intended to mean either an indirect or direct electrical connection. Thus, if a first device couples to a second device, that connection may be through a direct electrical connection, or through an indirect electrical connection via other devices and connections.

[0017] As used herein, the term "transaction" refers to a request to read from/write to memory or read from/write to another piece of logic or register.

[0018] As used herein, the term "bus master" refers to a piece of logic that initiates a transaction.

[0019] As used herein, the term "bus slave" refers to a component that receives a transaction; for example, a memory region or a peripheral may be a bus slave.

[0020] As used herein, the term "interconnect" refers to a component that distributes a transaction, for example between bus masters and bus slaves.

DETAILED DESCRIPTION

[0021] The following discussion is directed to various embodiments of the invention. Although one or more of these embodiments may be preferred, the embodiments disclosed should not be interpreted, or otherwise used, as limiting the scope of the disclosure, including the claims. In addition, one skilled in the art will understand that the following description has broad application, and the discussion of any embodiment is meant only to be exemplary of that embodiment, and not intended to intimate that the scope of the disclosure, including the claims, is limited to that embodiment.

[0022] Safety and non-safety function may be implemented, for example, on a system on a chip (SOC) with one or more processor cores and a memory, which may be shared among processor cores. In theory, a highest level of safety is achieved when a separate SOC carries out each of the various functions of the electronic device. In this way, the operation of a particular function cannot be impaired or corrupted by other functions since a bus master that implements a particular function cannot access any bus slave(s) other than its own. However, such an approach is cost-prohibitive.

[0023] To reduce the cost of such electronic devices, safety functions may be implemented alongside non-safety functions, for example with multiple functions carried out by a single SOC. However, to maintain an appropriate SIL, certain functions should be prevented from interfering with other functions (e.g., a function should be prevented from accessing an address region memory that is not allocated to that function or by sending a transaction to a peripheral that is not allocated to that function).

[0024] Safety functions may be associated with one of a plurality of SILs. For example, a safety function with a SIL of 3 may require a high level of safety assurance while a function with a SIL of 2 or lower requires a lower level of safety assurance, while still requiring more safety assurance than a non-safety function. That is, the function with a SIL of 3 presents a greater degree of risk relative to the function with a SIL of 2 (or lower) and as such requires greater risk reduction measures. As a result, multiple safety functions may have SILs that are independent of each other. Various standards require that functions having different SIL ratings should not interfere with one another. Similarly a non-safety critical task must not interfere with a safety critical task. Thus, while a non-safety function should be separated such that the non-safety function does not corrupt the safety function(s), a higher-SIL safety function (i.e., numerically greater) should also be separated such that the lower-SIL safety function does not corrupt the higher-SIL safety function.

[0025] FIG. **1** shows a system comprising SOC architecture **100** having multiple functions (also referred to as tasks) implemented by a number of bus masters. As explained above, the SOC architecture **100** may be part of an electronic device that controls a process and performs multiple functions. Certain of the tasks may be safety functions, in some cases having varying SILs, and other of the tasks may be non-safety functions. The SOC architecture **100** comprises a CPU **102** implementing tasks A and B, a direct memory access (DMA) engine **104** implementing tasks C and D, and a Universal Serial Bus (USB) controller **106** implementing task E. The CPU **102**, DMA engine **104**, and USB controller **106** are examples of bus masters.

[0026] The SOC architecture **100** also comprises an interconnect **108** that couples the bus masters **102**, **104**, **106** to exemplary bus slaves, such as random access memory (RAM) **110** and read-only memory (ROM) **112**. Additionally, the interconnect **108** may couple the bus masters **102**, **104**, **106** to peripherals **116*a*-116*n*** (e.g., a serial port, a general purpose input/output port, or a timer). In some cases, a peripheral interconnect **114** is inserted between the interconnect **108** and the peripherals **116*a*-116*n*** to further facilitate routing of transactions to the appropriate peripheral **116*a*-116*n***.

[0027] The SOC architecture **100** is exemplary, and it should be appreciated that multiple instances of various bus masters **102**, **104**, **106** may exist within an application-specific SOC. Regardless of the particular implementation, maintaining freedom from interference between various tasks at the bus slave level is important to assure that the device that carries out the various tasks achieves an acceptable level of risk. Additionally, as shown in FIG. **1**, certain bus masters **102**, **104**, **106** implement multiple tasks, some of which may be safety functions and others of which may be non-safety functions, and so maintaining freedom of interference between tasks operating on a single bus master **102**, **104**, **106** is important as well. Further, preventing a bus master **102**, **104**, **106** from improperly accessing a certain bus slave (e.g., a bus slave for which the bus master is not entitled to access) may provide further security from interference between tasks executing within the SOC architecture **100**.

[0028] Turning to FIG. **2**, the CPU **102** is shown with a local memory protection unit (MPU) **202**. The MPU **202** comprises hardware logic (not shown) that determines whether to grant or deny access to a bus slave on a per transaction basis. The hardware logic may comprise various comparators, encoders, decoders and the like that utilize information contained in a transaction to determine whether to grant or deny access to a bus slave. For example, a transaction may be an instruction fetch or data access request. The MPU **202** may transmit an instruction fetch to an instruction bus, transmit a data access request to a data bus, or transmit either to a mixed instruction and data bus, along with a control signal to identify whether the transaction is an instruction fetch or a data access request. This is shown in FIG. **2** by way of the MPU **202** transmitting the instruction fetch and data access requests separately. The interconnect **108** represents the various bus implementations.

[0029] Information contained in the instruction fetch and/or data access request may be used to determine whether to grant or deny access to a bus slave. Additionally, the determination by the MPU **202** of whether to grant or deny access to a bus slave may be based on one or a combination of a number of factors.

[0030] In some cases, transactions may be isolated based on the address of memory to which the transaction is directed. For example, certain addresses may be protected while other addresses are non-protected. A transaction originating from a safety function may be granted access by the MPU **202** to an address that is either protected or non-protected, while a transaction originating from a non-safety function is granted access to an address that is non-protected and denied access to an address that is protected. Additionally, in certain embodiments there may be multiple levels of address protection and a higher-level safety function is granted access to any address, while a lower-level safety function is only granted access to certain levels of protected addresses and a non-safety function is only granted access to non-protected addresses.

[0031] In other cases, transactions may be isolated based on a privilege level associated with the function or task that generates the transaction. For example, certain functions may be "privileged" and other functions may be "non-privileged." Transactions originating from a privileged function may be granted access by the MPU **202** to bus slaves that require a privileged level and transactions originating from a non-privileged function may be denied access to bus slaves that require a privileged level. Similarly, transactions may be isolated based on a security level where some functions comprise trusted code while other functions comprise non-trusted code. Transactions originating from trusted code are granted access by the MPU **202** to secure bus slaves and transactions originating from non-trusted code are denied access to secure bus slaves.

[0032] Additionally, transactions may be isolated based on a task identification (ID) associated with the function or task that generates the transaction. For example, the bus master or a CPU **102** may assign a task ID to each task that is running, which can be used by the MPU **202** to discriminate permissions on a per task basis. Alternately, transactions may be isolated based on whether the transaction originated from a function or task executed by a bus master that is a "functional unit" or executed by a bus master that is a "debug unit." The MPU **202** may grant access to certain bus slaves for tasks originating from a functional unit and deny access to those bus slaves for tasks originating from a debug unit.

[0033] Referring to FIGS. **1** and **2**, address regions of the RAM **110** and/or ROM **112** have associated permissions. If various attributes of a particular function or task satisfy the permission level of the address region, the MPU **202** grants access to a transaction originating from that function or task. If the attributes do not satisfy the permission level of the address region, access is denied. For certain components that support the execution of more than one task (e.g., CPU **102**), the associated MPU **202** is reconfigured when the task being executed changes to support task-based isolation. Configuration of the MPU **202** refers to the access permissions that are applied to the currently-executing task. For example, a memory buffer may belong to a first task. When the CPU **102** is executing the first task, the MPU **202** is configured to grant access to the memory buffer; however, when the CPU **102** switches to a second task, the MPU **202** is reconfigured to prevent access to the memory buffer. The MPU **202** may have many stored configurations corresponding to different tasks executed by the CPU **102**. In some embodiments, the MPU **202** may switch configurations based on a different received task ID for a transaction. In other embodiments, such as where the bus master is the CPU **102**, software executing on the CPU **102** that changes the task also reconfigures the MPU **202**.

[0034] In the event of an attempted violation of access rules implemented by the MPU **202**, various actions may be taken. For example, the MPU **202** may report the attempted access violation to a system-level monitoring task executing on the CPU **102**. In some cases, the MPU **202** blocks the transaction from occurring, while in other cases the MPU **202** tags the transaction as having an error. Further, in security-sensitive applications where a transaction tagged as having an error may provide useful information to a malicious entity attempting to gain access to secure memory, a response may be generated that mimics a normal response, but which contains false data.

[0035] FIG. **3** shows the USB controller **106**, which is an example of a single-task bus master. In the case of a single-task bus master, a MPU **302** similar to the MPU **202** is implemented, although on a simplified basis. For example, the USB controller **106** typically accesses only two regions—a transmit buffer and a receive buffer. Additionally, it is not necessary that the MPU **302** implement task-based discrimination since only one task is implemented by the USB controller **106**.

[0036] In the above examples, a MPU **202**, **302** facilitates protection of certain regions of memory and/or certain peripherals by limiting access by lower-level or non-safety functions where appropriate. As a result, an acceptable level of safety is achieved by the overall device on which the SOC architecture **100** is implemented while reducing the cost of the device by implementing many functions on a single SOC.

[0037] In the event of an attempted violation of access rules implemented by the MPU **202**, various actions may be taken. For example, the MPU **202** may report the attempted access violation to a system-level monitoring task executing on the CPU **102**. In some cases, the MPU **202** blocks the transaction from occurring, while in other cases the MPU **202** tags the transaction as having an error. Further, in security-sensitive applications where a transaction tagged as having an error may provide useful information to a malicious entity attempting to gain access to secure memory, a response may be generated that mimics a normal response, but which contains false data.

[0038] In accordance with various embodiments, a non-programmable bus master, such as the DMA controller **104**, may implement multiple tasks to perform various functions. Unlike a bus master such as the CPU **102**, which may reconfigure its MPU **202** with software executing tasks on the CPU **102**, the DMA controller **104** does not execute software to optimize its performance during DMA operations, and thus is non-programmable.

[0039] Turning to FIG. **4**, the DMA controller **104** comprises an integrated MPU **402**. The DMA controller **104** is shown as able to implement multiple tasks, namely task C and task D. Each task generates various transactions to access memory **110**, **112** or peripherals **116*a*-116*n***. In accordance with various embodiments, the DMA controller **104** includes hardware logic that switches between tasks as needed to perform the required functionality of the DMA controller **104**. The MPU **402** is integrated to the DMA controller **104** such that, upon switching from one task to another, the DMA controller **104** causes a configuration of the MPU **402** to switch as well. For example, when the DMA controller **104** is executing task C, the DMA controller **104** causes the MPU **402** to operate in a first configuration, while when the DMA controller **104** is executing task D, the DMA controller **104** causes the MPU **402** to operate in a second configuration. As explained above, the MPU **402** regulates access to certain bus slaves in each configuration. The MPU **402** may have a different configuration for each task implemented by the DMA controller **104**.

[0040] In some embodiments, the DMA controller **104** implements automated task-switching by automatically changing the configuration of the integrated MPU **402** when the DMA controller **104** switches tasks. However, in other embodiments, the DMA controller **104** may provide task identification (ID) to the MPU **402** and, as a result of receiving a different task ID, the MPU **402** changes its configuration. This allows the MPU **402** to be less closely integrated to the DMA controller **104**.

[0041] As explained above, for a transaction generated by one of the tasks implemented by the DMA controller **104**, the MPU **402** determines whether to grant or deny access to a bus slave for that transaction. This determination may be based on the address of memory to which the transaction is directed, a privilege level of the transaction or the task that generates the transaction, or a security level of the task that generates the transaction.

[0042] Thus, the DMA controller **104** enables automated task-switching for the MPU **402** configurations to apply different access permissions to each task executed by the DMA controller **104**. As such, memory protection is enabled, achieving an acceptable level of risk, even in systems where a non-programmable bus master such as the DMA controller **104** implements multiple tasks, which include safety and non-safety functions.

[0043] In accordance with various other embodiments, a bus master, such as the CPU **102**, may implement multiple instances of virtualized hardware to perform various functions. Turning to FIG. **5**, the CPU **102** may contain a first virtual CPU **502** that implements a safety function and a second virtual CPU **504** that implements a non-safety function. However, since both the safety function and the non-safety function are implemented by the same physical CPU (i.e., the CPU **102**), the CPU ID for a transaction generated by either function would be the same. Additionally, in some cases a task ID for a transaction generated by either function may be the same. Thus, the MPU **202** described above would not be able to differentiate the transactions and a lower-level or non-safety function may be inappropriately granted access to a particular bus slave.

[0044] In accordance with various embodiments, a virtual CPU ID is associated with each virtual CPU **502**, **504** simulated on the physical CPU **102**. Additionally, a virtual task ID may be associated with each virtual task running on the virtual CPUs **502**, **504**. An MPU **506** associated with a bus master that implements virtualized hardware (e.g., the physical CPU **102** implementing one or more virtual CPUs **502**, **504**) grants or denies access to a peripheral, memory region, or other bus slave based on the virtual CPU ID and/or the virtual task ID. As such, memory protection is enabled, achieving an acceptable level of risk, even in systems where safety and non-safety functions are implemented in virtualized hardware.

[0045] Further, the physical CPU **102** may execute tasks (e.g., task E) independently of tasks (e.g., tasks A-D) executed by the virtual CPUs **502**, **504**. In such cases, the MPU **506** does not only grant or deny access based on virtual CPU ID or virtual task ID, but rather grants and denies access generally based on virtual CPU ID and CPU ID or virtual task ID and task ID. In this way, the MPU **506** applies an equal permission scheme to CPUs, regardless of whether they are virtual CPUs **502**, **504** or a physical CPU **102**. Similarly, the MPU **506** applies an equal permission scheme to tasks, regardless of whether they are tasks implemented by virtual hardware (i.e., tasks A-D implemented by virtual CPUs **502**, **504**) or tasks implemented by physical hardware (i.e., task E implemented by CPU **102**).

[0046] Turning now to FIG. **6**, multiple examples of a slave-based memory protection scheme are shown in accordance with various embodiments. In a first example, a MPU **604** is positioned in the datapath between a bus slave (e.g., memory **602**) and an interconnect **108** that transmits data to and from the bus slave **602**. This is simple but limited because, in some cases, the introduction of a MPU not tightly integrated into the datapath may be physically larger, consume more power, and introduce transaction latency as compared to solutions that are optimized for the particular datapath and coupling between the interconnect **108** and the bus slave **602**.

[0047] In a second example, a MPU **606** is integrated into the interconnect **108**. In this context, being integrated refers to an interconnect **108** design in which the MPU **606** is included directly into the datapath at the time of design of the interconnect **108** rather than added to the datapath design after the interconnect **108** has been designed. As a result, the MPU **606** may provide additional capability relative to the MPU **604**, such as reduced latency (i.e., improved overall performance), reduced power consumption, reduced physical size, and improved response time.

[0048] In a third example, a MPU **608** is integrated into the bus slave **602** itself. Similar to being integrated into the interconnect **108**, in this context, integrated refers to the fact that the MPU **608** is part of the base design of the bus slave **602** itself. Thus, the MPU **608** may be optimized for the behavior of the particular bus slave **602**. As a result, the MPU **608** may be optimized in particular for the bus slave **602** to which it is integrated. For example, optimization such as reduced latency, reduced power consumption, reduced physical size, and improved response time are possible.

[0049] Regardless of the particular location and implementation of the slave-based MPU **604**, **606**, **608**, the MPU **604**, **606**, **608** includes a set of access permissions that grants access to the bus slave **602** when certain conditions are met and denies access to the bus slave **602** when at least one of those conditions are not met. More particularly, granting and denying access is often determined on a transaction by transaction basis, where the transaction is generated by a task executing on a bus master. For example, the MPU **604**, **606**, **608** may deny access to the bus slave **602** based on an address to which the transaction is directed, a privilege level associated with the task that generated the transaction, a security level associated with the task that generated the transaction, or whether the transaction was generated by a functional unit of the bus master or a debug unit of the bus master.

[0050] In some embodiments, the MPU **604**, **606**, **608** grants or denies access to the bus slave **602** based on the bus master that generated the transaction. For example, transactions generated by tasks on a first bus master may be generally granted to access the bus slave **602** while transactions generated by tasks on a second bus master are denied access to the bus slave **602**. In this way, while MPUs associated with bus masters (e.g., as shown in FIGS. **4** and **5**) differentiate access permissions largely on a task-by-task basis from a single bus master while MPUs **604**, **606**, **608** associated with bus slaves may differentiate access permissions on a bus master-by-bus master basis.

[0051] In the event of an attempted violation of access rules implemented by the MPU **604**, **606**, **608**, various actions may be taken. For example, the MPU **604**, **606**, **608** may report the attempted access violation to a system-level monitoring task executing on the CPU **102**. In some cases, the MPU **604**, **606**, **608** blocks the transaction from occurring, while in other cases the MPU **604**, **606**, **608** tags the transaction as having an error and generates a bus error response via the interconnect **108**. Further, in security-sensitive applications where a transaction tagged as having an error may provide useful information to a malicious entity attempting to gain access to secure memory (e.g., bus slave **602**), a response may be generated that mimics a normal response, but which contains false data.

[0052] Thus, in some embodiments a system-wide memory protection scheme is disclosed, in which MPUs are implemented at both the bus master level and the bus slave level. As a result, an acceptable level of safety is achieved by the overall device on which the system-wide memory protection scheme (e.g., including SOC architecture **100**) is implemented while reducing the cost of the device by implementing many functions on a single SOC.

[0053] FIG. **7** shows a method **700** for bus slave-based memory protection, for example where a MPU is positioned in the data stream between a bus slave and an interconnect, integrated to the interconnect, or integrated to the bus slave, in accordance with various embodiments. The method **700** contains various steps, which may be performed in an order other than that shown in FIG. **7**. The method **700** begins in block **702** with receiving, by a MPU associated with a bus slave, a transaction from a bus master directed at the bus slave. For example, the transaction may be generated by a task executing on the bus master, and may be related to a safety function of varying levels or a non-safety function.

[0054] The method **700** continues in block **704** with the MPU determining whether to grant or deny the transaction access to the bus slave. If it is determined to deny the transaction access to the bus slave in block **704**, the method **700** continues in block **706** with generating an error response. The error response may include a bus error response (e.g., an error message is transmitted via the interconnect), transmission of false information intended to appear as a normal response, or blocking the transaction from accessing the bus slave. Denial of a transaction, and thus subsequent generation of an error response, may occur as a result of an identification of the bus master that generated the transaction, an address to which the transaction is directed, a privilege or security level associated with the task that generated the transaction, or whether the transaction was generated by a functional unit of the bus master or a debug unit of the bus master.

[0055] As explained above, in some embodiments a system-wide memory protection scheme is disclosed, in which MPUs are implemented at both the bus master level and the bus slave level. In such embodiments, the method **700** may comprise additional steps not shown in FIG. **7** for conciseness. For example, the method **700** may further comprise a non-programmable bus master (e.g., a DMA controller) executing first and second tasks, each generating transactions, where hardware logic of the non-programmable bus master switches between executing the first and second tasks. The non-programmable bus master may cause a MPU associated with the non-programmable bus master to operate in a first configuration with a first set of access permissions when the hardware logic executes the first task. Correspondingly, the non-programmable bus master may cause the MPU associated with the non-programmable bus master to operate in a second configuration with a second set of access permissions when the hardware logic executes the second task.

[0056] As another example, the method **700** may further comprise a MPU associated with a virtual CPU (implemented on a physical CPU) receiving a transaction from the virtual CPU directed at a bus slave. The transaction may be associated with a virtual CPU ID or a virtual task ID. The MPU determines whether to grant or deny access to the bus slave based on the virtual CPU ID or the virtual task ID. In either case, the virtual CPU ID or virtual task ID is different than an ID of the physical CPU on which the virtual CPU is implemented or an ID of a task executed on the physical CPU, respectively.

[0057] As a result, the method **700** enables bus slave-based memory protection, where access to a bus slave is determined at least party based on the bus master from which a transaction originates. Additionally, the method **700** facilitates a system-wide memory protection scheme, in which MPUs are implemented at both the bus master level and the bus slave level. As a result, an acceptable level of safety is achieved by the overall device on which the system-wide memory protection scheme (e.g., including SOC architecture **100**) is implemented while reducing the cost of the device by implementing many functions on a single SOC.

[0058] The above discussion is meant to be illustrative of the principles and various embodiments of the present invention. Numerous variations and modifications will become apparent to those skilled in the art once the above disclosure is fully appreciated. It is intended that the following claims be interpreted to embrace all such variations and modifications.

What is claimed is:

1. A system, comprising:

a bus slave coupled to a plurality of bus masters via one or more interconnects; and

a memory protection unit (MPU) associated with the bus slave, the MPU having a set of access permissions that grants access to the bus slave from a first bus master and denies access to the bus slave from a second bus master;

wherein the MPU generates an error response as result of a transaction generated by a task on the second bus master attempting to access the bus slave.

2. The system of claim **1** wherein the error response comprises a bus error response.

3. The system of claim **1** wherein the error response comprises false information.

4. The system of claim **1** wherein the error response comprises blocking the transaction from accessing the bus slave.

5. The system of claim **1** wherein the MPU comprises hardware logic that generates an error response based on one selected from the group consisting of:

an address to which the transaction is directed;

a privilege level associated with the task that generated the transaction;

a security level associated with the task that generated the transaction; and

whether the transaction was generated by a functional unit of the bus master or a debug unit of the bus master.

6. The system of claim **1** further comprising:
a non-programmable bus master; and
a MPU associated with the non-programmable bus master, the MPU to operate in a first configuration with a first set of access permissions and a second configuration with a second set of access permissions;
wherein the non-programmable bus master further comprises hardware logic to:
execute a first task and a second task, wherein the tasks generate transactions and wherein the hardware logic switches between executing the first and second tasks;
cause the MPU to operate in the first configuration when the hardware logic executes the first task; and
cause the MPU to operate in the second configuration when the hardware logic executes the second task.

7. The system of claim **1** further comprising:
a virtual central processing unit (CPU); and
a MPU associated with the virtual CPU comprising hardware logic to:
receive a transaction from the virtual CPU directed at the bus slave, the transaction being associated with a virtual CPU identification (ID), wherein the virtual CPU is implemented on a physical CPU; and
determine whether to grant or deny access to the bus slave based on the virtual CPU ID;
wherein the virtual CPU ID is different than an ID of the physical CPU on which the virtual CPU is implemented.

8. The system of claim **1** further comprising:
a virtual central processing unit (CPU); and
a MPU associated with the CPU comprising hardware logic to:
receive a transaction from a virtual central processing unit (CPU) directed at the bus slave, the transaction being associated with a virtual task identification (ID), wherein the virtual CPU is implemented on a physical CPU; and
determine whether to grant or deny access to the bus slave based on the virtual task ID;
wherein the virtual task ID is different than an ID of a task executed on the physical CPU on which the virtual CPU is implemented.

9. A method, comprising:
receiving, by a memory protection unit (MPU) associated with a bus slave, a transaction from a bus master directed at the bus slave;
determining, by the MPU, whether to grant or deny the transaction access to the bus slave; and
generating an error response as a result of determining to deny access to the transaction.

10. The method of claim **9** wherein the error response comprises a bus error response.

11. The method of claim **9** wherein the error response comprises false information.

12. The method of claim **9** wherein the error response comprises blocking the transaction from accessing the bus slave.

13. The method of claim **9** wherein generating an error response occurs based on one selected from the group consisting of:
an identification of the bus master that generated the transaction;
an address to which the transaction is directed;
a privilege level associated with the task that generated the transaction;
a security level associated with the task that generated the transaction; and
whether the transaction was generated by a functional unit of the bus master or a debug unit of the bus master.

14. The method of claim **9** further comprising:
executing, by a non-programmable bus master comprising hardware logic, a first task and a second task, wherein the tasks generate transactions and wherein the hardware logic switches between executing the first and second tasks;
causing, by the non-programmable bus master, a MPU associated with the non-programmable bus master to operate in a first configuration with a first set of access permissions when the hardware logic executes the first task; and
causing, by the non-programmable bus master, the MPU associated with the non-programmable bus master to operate a the second configuration with a second set of access permissions when the hardware logic executes the second task.

15. The method of claim **9** further comprising:
receiving, by a MPU associated with a virtual central processing unit (CPU), a transaction from the virtual CPU directed at the bus slave, the transaction being associated with a virtual CPU identification (ID), wherein the virtual CPU is implemented on a physical CPU; and
determining, by the MPU associated with the virtual CPU, whether to grant or deny access to the bus slave based on the virtual CPU ID;
wherein the virtual CPU ID is different than an ID of a physical CPU on which the virtual CPU is implemented.

16. The method of claim **9** further comprising:
receiving, by a MPU associated with a virtual central processing unit (CPU), a transaction from the virtual CPU directed at the bus slave, the transaction being associated with a virtual task identification (ID), wherein the virtual CPU is implemented on a physical CPU; and
determining, by the MPU associated with the virtual CPU, whether to grant or deny access to the bus slave based on the virtual task ID;
wherein the virtual task ID is different than an ID of a task executed on a physical CPU on which the virtual CPU is implemented.

17. An electronic device to control a process, comprising:
a bus slave comprising memory or a peripheral;
first and second bus masters to execute one or more tasks, each task to generate transactions directed at the bus slave;
an interconnect to couple the bus slave to the bus master; and
a memory protection unit (MPU) associated with the bus slave, the MPU having a set of access permissions that grants access to the bus slave from the first bus master and denies access to the bus slave from the second bus master;
wherein the MPU generates an error response as result of a transaction generated by a task on the second bus master attempting to access the bus slave.

18. The electronic device of claim **17** wherein the error response comprises a bus error response.

19. The electronic device of claim **17** wherein the error response comprises false information.

20. The electronic device of claim **17** wherein the error response comprises blocking the transaction from accessing the bus slave.

21. The electronic device of claim **17** wherein the MPU comprises hardware logic that generates an error response based on one selected from the group consisting of:

an address to which the transaction is directed;

a privilege level associated with the task that generated the transaction;

a security level associated with the task that generated the transaction; and

whether the transaction was generated by a functional unit of the bus master or a debug unit of the bus master.

* * * * *