



(51) International Patent Classification:
G06F 9/44 (2006.01)

(21) International Application Number:
PCT/IB2014/002221

(22) International Filing Date:
25 June 2014 (25.06.2014)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
61/860,842 31 July 2013 (31.07.2013) US

(71) Applicant: MARVELL WORD TRADE LTD. [BB/BB];
L'horizon, Gunsite Road, Brittons Hill, St. Michael,
BB14027 (BB).

(72) Inventors: AKDEMIR, Kahraman, D.; 8 Independence
Way, Apt. 211, Franklin, MA 02038 (US). AYTEK,
Tolga, Nihat; 6 John Mcquinn Circle, Framingham, MA
01701 (US).

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, LT, 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, SZ, TZ,
UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ,
TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK,
EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV,
MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM,
TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW,
KM, ML, MR, NE, SN, TD, TG).

Published:

(81) Designated States (unless otherwise indicated, for every
kind of national protection available): AE, AG, AL, AM,

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

[Continued on next page]

(54) Title: PARALLELIZING BOOT OPERATIONS

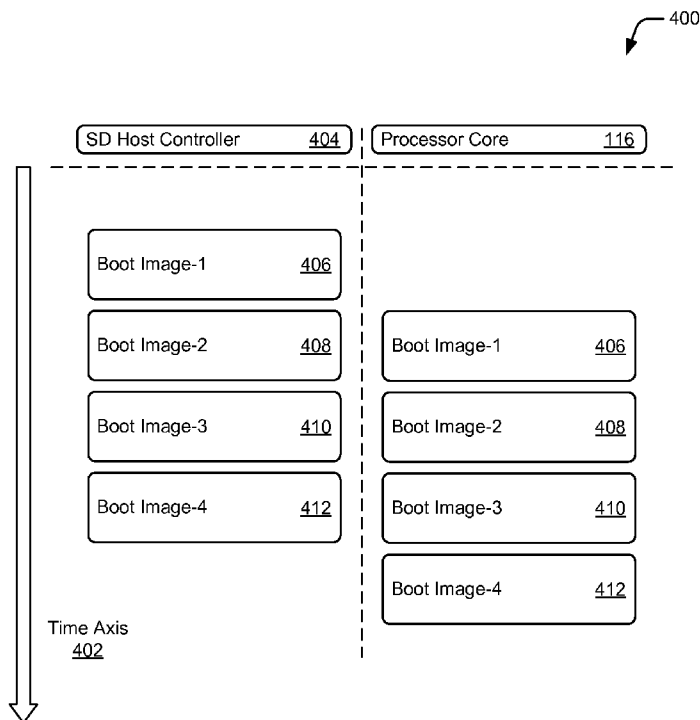


Fig. 4

(57) Abstract: The present disclosure describes apparatuses (102, 202, and 700) and techniques (300, 500, and 600) for parallelizing boot operations. In some aspects, an operation transferring a boot image from a non-volatile memory to a volatile memory is initiated prior to completion

[Continued on next page]



-
- *before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments (Rule 48.2(h))*

an operation validating another boot image previously-transferred into the volatile memory. This can be effective to enable transfer operations and validation operations of boot images to be performed in parallel. By so doing, delays between the transfer and validation operations can be minimized thereby reducing device boot times.

PARALLELIZING BOOT OPERATIONS

CROSS REFERENCE TO RELATED APPLICATION

[0001] This present disclosure claims priority to U.S. Provisional Patent Application Serial No. 61/860,842 filed July 31st, 2013, the disclosure of which is incorporated by reference herein in its entirety.

BACKGROUND

[0002] The background description provided herein is for the purpose of generally presenting the context of the disclosure. Work of the presently named inventors, to the extent it is described in this background section, as well as aspects of the description that may not otherwise qualify as prior art at the time of filing, are neither expressly nor impliedly admitted as prior art against the present disclosure.

[0003] Computing and electronic devices often execute an operating system or applications to implement functionalities of the device. The operating system or applications of a device, however, are unable to execute until boot code is used to configure and boot the device for operation. When booting a device, portions of the boot code are copied from a non-volatile memory of the device in the form of boot images. Each of the boot images is typically copied to a volatile memory of the device for validation or other boot operations before a next boot image is copied from the non-volatile memory. Waiting for a processor to complete validation or other boot

operations for each boot image, however, can delay the loading of other boot images and introduce significant delays into the boot process.

SUMMARY

[0004] The details of one or more implementations are set forth in the accompanying drawings and the description below. Other features and advantages will be apparent from the description and drawings. As such, this Summary should not be considered to describe essential features nor used to limit the scope of the claimed subject matter.

[0005] A method is described for initiating, via a direct memory access (DMA) controller, a first transfer operation of a first boot image from a first memory into a second memory. After completion of the first transfer operation, the method then initiates a validation operation of the first boot image in the first memory to verify that the first boot image is secure. Prior to completion of the validation operation, a second transfer operation is initiated to transfer a second boot image from the first memory into the second memory.

[0006] Another method is described for retrieving a header image from a non-volatile memory that stores multiple boot images, which are described by the header image. The method then constructs, based on the header images, a structure and descriptors that describe source locations of the multiple boot images and destination locations of the multiple boot images in a volatile memory. The structure and descriptors are then issued effective to cause a memory controller to transfer at least

one of multiple boot images into the volatile memory while a processor validates a previously-transferred one of the multiple boot images.

[0007] A System-on-Chip is described that includes a boot manager that is configured to implement parallel boot operations. In some aspects, the boot manager initiates, via a memory controller, a first direct memory access (DMA) transfer operation that transfers a first of the multiple boot images from a non-volatile memory to a volatile memory. After completion of the first DMA transfer operation, the boot manager causes a processor to initiate a validation operation to validate the first boot image in the volatile memory. Prior to completion of the validation operation, the boot manager initiates a second DMA transfer operation that transfers a second of the multiple boot images from the non-volatile memory to the volatile memory.

[0008] The details of one or more implementations are set forth in the accompanying drawings and the description below. Other features and advantages will be apparent from the description and drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

[0009] In the figures, the left-most digit of a reference number identifies the figure in which the reference number first appears. The use of the same reference numbers in different instances in the description and the figures indicate like elements.

Fig. 1 illustrates an operating environment having computing devices in accordance with one or more aspects.

Fig. 2 illustrates an example system architecture in accordance with one or more aspects.

Fig. 3 illustrates a method for initiating parallel transfer and validation operations during a boot sequence.

Fig. 4 illustrates an example boot image loading sequence in accordance with one or more aspects.

Fig. 5 illustrates a method for constructing a command structure and transfer descriptors to enable parallel boot operations.

Fig. 6 illustrates a method of booting a computing device with parallelized boot operations.

Fig. 7 illustrates a System-on-Chip (SoC) environment for implementing aspects of the techniques described herein.

DETAILED DESCRIPTION

[0010] Conventional techniques for loading boot images sequentially transfer each boot image after a previously-transferred boot image is validated. For example, once a memory controller has transferred a boot image into volatile memory, the memory controller waits until a processor has validated the boot image before starting a transfer of a next boot image. Further, boot images are typically large in size due to the complex nature of devices, which results in validation operations for each boot image consuming a considerable number of processing cycles and time. This not only delays the transfer

and validation of each subsequent boot image, but has a cumulative effect of significantly slowing an entire boot up sequence.

[0011] This disclosure describes apparatuses and techniques for parallelizing boot operations, which enable boot images to be transferred into volatile memory while other boot images are being validated. Once a boot image is transferred from non-volatile memory into volatile memory, validation of the boot image and transfer of a next boot image are performed in substantially concurrent fashion. By so doing, subsequent boot images can be transferred without waiting for validations of previously-transferred boot images to complete. This can be effective reduce delays in the boot process and reduce an overall boot time of a device by as much as 40 percent to 50 percent.

[0012] The following discussion describes an operating environment, techniques that may be employed in the operating environment, and a System-on-Chip (SoC) in which components of the operating environment can be embodied. In the discussion below, reference is made to the operating environment by way of example only.

[0013] Operating Environment

Fig. 1 illustrates an example of an operating environment 100 having computing devices 102, each of which are capable of communicating, accessing, presenting, or processing various data. Computing devices 102 include smart-phone 104, tablet computer 106, network-attached storage (NAS) device 108, multi-function printer 110, internet-protocol enabled television 112 (IP TV 112), and wireless router 114. Although not shown, other configurations of computing devices 102 are also contemplated such as a desktop computer, server, mobile-internet device (MID), camera, personal

navigation device, set-top box, netbook, ultra-book, gaming console, appliances, mobile hotspot, networked media player, and so on.

[0014] Generally, computing devices 102 have operational states ranging from an “off” state to an “on” state. These states may include a fully off state (mechanical off), suspend state, sleep state, hibernation state, idle state, active state, and the like. When transitioning from a lower operational state to a higher operational state (*e.g.*, from an off state to an active state), computing device 102 is booted. Booting computing device 102 includes executing low-level code (*e.g.*, boot code) to configure components of computing device 102 for operation and control by higher-level software, such as firmware, operating system, or applications.

[0015] Each computing device 102 includes processor core 116 (*e.g.* an application processor core) and computer-readable storage media 118 (CRM 118). Although shown as a single core, processor core 116 may be one of any suitable number and/or type of processing cores, which may be configured in any suitable manner (*e.g.*, a heterogeneous multi-core application processor). CRM 118 includes volatile memory 120 and non-volatile memory 122, which may include any suitable type, combination, or number of memory media or devices. Note, that CRM 118 are implemented at least partially in hardware or physical devices that are configured to store data (*e.g.*, digital or analog values) and do not include transitory signals or carrier waves.

[0016] In this particular example, volatile memory 120 includes static random access memory 124 (SRAM 124) and synchronous dynamic random-access memory 126 (DRAM 126). Alternately or additionally, volatile memory 120 may include other

suitable types of memory, such as random-access memory (RAM), asynchronous dynamic RAM, double-data-rate RAM (DDR), and the like.

[0017] Non-volatile memory 122 includes serial-peripheral interface (SPI) Flash 128 and Flash memory 130, which may be configured in any suitable way, such as standalone NAND Flash memory or an embedded multimedia card (eMMC) device comprising Flash memory, memory interface, and embedded memory controller. Other non-volatile memories contemplated include non-volatile RAM (NVRAM), read-only memory (ROM), electronically-erasable programmable ROM (EEPROM), NOR Flash, single-level cell (SLC) Flash, multi-level cell (MLC) Flash, and the like.

[0018] Non-volatile memory 122 stores data of computing device 102 persistently when powered-down or suspended. Boot code of computing device 102 can be stored in one or more non-volatile memory components, such as SPI Flash 128 and Flash memory 130. In some cases, the boot code of computing device 102 is stored as boot images that are individually addressable and transferrable. Operating systems, applications, or user data (not shown) of computing device 102 may be stored by, or executed from, volatile memory 120, non-volatile memory 122, or any suitable combination thereof. Alternately or additionally, operating systems and/or applications of computing device 102 may be embodied as firmware or other processor-executable instructions, binaries, or code. These operating systems and applications of computing device 102 are executable by processor core 116 to provide a user interface, various functionalities, applications, or services of computing device 102.

[0019] CRM 118 also contains boot manager 132, which in this particular example, is embodied as processor-executable instructions that are executable by processor core 116 to implement various functionalities. Alternately or additionally, boot manager 132 may be implemented, in part or whole, via firmware or hardware (not shown) or any suitable combination thereof. Boot manager 132 may enable concurrent transfers of boot images from non-volatile memory 122 to volatile memory 120 and validations of boot images in volatile memory 120. The implementation and use of boot manager 132 varies and is described below.

[0020] Computing device 102 also includes memory controller 134 that enables data to be transferred, copied, or otherwise moved between various memory locations. Memory controller 134 may be implemented as any suitable type of memory controller, such as a Flash memory controller, secure digital (SD) controller, SD input/output (SDIO) controller, SD combo controller, multimedia card (MMC) controller, SD host controller (SDHC), and the like. Alternately or additionally, memory controller 134 may be implemented separate from or integrated with CRM 118 or a memory thereof. Memory controller 134 also includes direct memory access (DMA) controller 136, which enables DMA transfer operations between various memories of computing device 102, such as Flash memory 130 and DRAM 126.

[0021] Generally, a DMA transfer operation, once initiated, is able to move data between memory locations without involving a processor (*e.g.*, processor core 116) or other higher-level controller. These DMA operations may include single operation DMA algorithms or advanced DMA (ADMA) operations, such as scatter gather ADMA

algorithms, which are described in version 2.0 and later revisions of the SD Host Controller specification. Note that memory controllers described herein may support any suitable type of ADMA operation, such as ADMA1 for transfers up to 4 KB or ADMA2 for transfers of data at any size or location. Alternate or additional aspects of ADMA transfer operations are also described in version 4.5 and later revisions of the Joint Electron Device Engineering Council (JEDEC) Embedded Multimedia Card (eMMC) standard.

[0022] The ADMA transfer algorithm is a scatter gather algorithm and operates via a set of transfer and link descriptors. The descriptors may be organized by a descriptor table (not shown), which may be programmed with descriptors describing a series of data transfers between Flash memory 130 and other memories of computing device 102. The descriptors may be executed from the descriptor table without interrupting software executing on processor core 116 (*e.g.*, a memory host driver). Memory controller 134 and DMA controller 136 may also support various boot modes, such as an alternate boot mode capable of implementing ADMA transfers. Accordingly, boot manager 132 may be configured to construct ADMA descriptors effective to cause DMA controller 136 to transfer boot images from non-volatile memory 122 to volatile memory 120 when computing device 102 is booted. Alternately or additionally, descriptors can be configured to trigger interrupts when execution of the descriptor completes or at the end of a transfer operation on a data bus.

[0023] Computing device 102 may also include I/O ports 138, display 140, or network interface(s) 142. I/O ports 138 allow computing device 102 to interact with

other devices or users. I/O ports 138 may include any combination of internal or external ports, such as USB ports, audio ports, Serial ATA (SATA) ports, PCI-express based ports or card-slots, secure digital input/output (SDIO) slots, and/or other legacy ports. Various peripherals may be operatively coupled with I/O ports 138, such as human-input devices (HIDs), external computer-readable storage media, or other peripherals.

[0024] Display 140 may present a user interface or rendered graphics associated with an operating system or application of computing device 102. Display 140 may include a touch-input sensor (not shown), such as a touch screen or touch-sensitive overlay. Network interface 142 provides connectivity to one or more networks and other devices connected therewith. Data communicated over network interface 142 may be packetized or framed depending on a communication protocol or standard by which computing device 102 is communicating. Network interfaces 142 may include wired interfaces, such as Ethernet or fiber optic interfaces for communication over a local network, intranet, or the Internet. Network interfaces 142 may also include wireless interfaces that facilitate communication over wireless networks, such as wireless LANs, cellular networks, or wireless personal-area-networks (WPANs).

[0025] Fig. 2 illustrates an example system architecture of a computing device 102 generally at 200. In this particular example, some components of computing device 102 are illustrated as being implemented as part of System-on-Chip 202 (SoC 202), such as processor core 116, DRAM 126, and memory controller 134. These and other components of SoC 202 communicate via system bus 204, which may be configured as

a data bus, memory bus, system crossbar, interconnect fabric, or any combination thereof. In some cases, system bus 204 is implemented as an Advanced eXtensible Interface (AXI) bus, Advanced High Performance Bus (AHB), or Open Core Protocol (OCP) bus.

[0026] System architecture 200 also includes an external memory in the form of embedded multimedia card (eMMC) device 206, which communicates with SoC 202 via external memory interface 208. EMMC device 206 includes header image 210 and boot images 212, which are useful to boot and configure components of SoC 202 for operation and control by a higher-level operating system. Generally, boot images are transferred from eMMC device 206 to DRAM 126 via system bus 204 when computing device 102 is booted. EMMC device 206 may also include an internal memory controller and data interface (not shown) that enable communication over external memory interface 208.

[0027] Header image 210 includes information describing locations, destinations, or sizes of boot images 212 that are part of a boot process for computing device 102. In some cases, header image 210 is located at a predefined or default address within eMMC device 206 that is accessed when computing device 102 boots. When computing device 102 implements a secure boot process, a security or authenticity of header image 210 and boot images 212 is determined before code of the images is executed by processor core 116, memory controller 134, or other components of computing device 102. In some cases, a hash value for a boot image is generated by a

processor and then compared with a predefined hash reference value to verify the security of the boot image.

[0028] The information of header image 210 can be useful to construct commands and transfer descriptors (*e.g.*, ADMA descriptors) for loading boot images into volatile memory for validation or execution. In some cases, groups of commands are assembled to form a packed command structure, which can be sent to a memory controller to enable multiple data transfers. For example, information describing respective sizes and locations of boot images in a flash memory can be used to construct a packed command structure (*e.g.*, read_multiple_block commands) that is configured to cause the boot images to be read or streamed from the flash memory. Note here that execution of the packed command structure is treated as a single transfer operation rather than a series of individual transfers. The packed commands described herein can be implemented in accordance with version 4.5 and later revisions of the JEDEC eMMC standard.

[0029] Alternately or additionally, information describing respective sizes and destinations of boot images can be used to construct multiple descriptors (*e.g.*, ADMA descriptors) for transferring the boot images into non-volatile memory. Transfer descriptors typically include a destination address, a length indicator specifying an amount of data to transfer, and various attribute fields (*e.g.*, descriptor type, validity, end, interrupt action). In some cases, the descriptors are configured to trigger an interrupt in response to execution of the respective descriptors or in response to completion of a data transfer. These interrupts can be useful to signal or synchronize

other boot operations performed by various entities, such as initiating validation operations once a boot image is loaded into volatile memory.

[0030] Techniques of Parallelizing Boot Operations

The following discussion describes techniques of parallelizing boot operations. These techniques can be implemented using the previously described environment or entities, such as boot manager 132, memory controller 134, or DMA controller 136 of Fig. 1 embodied on a computing device 102. These techniques include methods illustrated in Figs. 3, 5, and 6, each of which is shown as a set of operations performed by one or more entities. These methods are not necessarily limited to the orders shown for performing the operations. Further, these methods may be used in conjunction with one another, in whole or in part, whether performed by the same entity, separate entities, or any combination thereof. In portions of the following discussion, reference will be made to operating environment 100 of Fig. 1 and entities of Figs. 2 and 4 by way of example. Such reference is not to be taken as limited to operating environment 100 but rather as illustrative of one of a variety of examples.

[0031] Fig. 3 depicts a method 300 for initiating parallel transfer and validation operations during a boot sequence, including operations performed by boot manager 132 of Fig. 1.

[0032] At 302, a first transfer operation is initiated to transfer a first boot image from a non-volatile memory of a device. The first transfer operation transfers the first boot image to a volatile memory of the device. The first boot image may include any suitable

amount of data, such as multiple blocks or pages of boot code or binaries. In some cases, the non-volatile memory includes a header image describing the first and other boot images. In such cases, the transfer operation can be initiated based on information included in the header image, such as a location, destination, or size of the first boot image.

[0033] The first transfer operation may be a DMA transfer implemented by a DMA-enabled memory controller executing descriptors to copy the first boot image from the non-volatile memory. In some cases, the DMA transfer is initiated by setting up the transfer or other commands, and may include configuring a DMA command or transfer descriptor that specifies the non-volatile memory as a source of the first boot image and the volatile memory as a destination. Alternately or additionally, the first boot image may be read from the non-volatile memory in response to execution of a packed command sequence by a memory controller associated with the non-volatile memory.

[0034] As an example, consider a user powering-on IP TV 112 of Fig. 1. Assume here that IP TV 112 is transitioning to an “on” state from an “off” state in which power was removed from DRAM 126 and other various components of IP TV 112. In the context of this example, also assume that IP TV 112 is configured similar to the system architecture of Fig. 2 in which the boot images 212 are stored by eMMC device 206. Here, boot manager 132 constructs, based on header image 210, transfer descriptors and packed commands to enable the transfer of boot images 212 from eMMC device 206 to DRAM 126. Boot manager 132 then issues the transfer descriptors and packed commands to memory controller 134 and eMMC device 206, respectively.

[0035] In response to executing the transfer descriptor, memory controller 134 initiates a DMA transfer of the first boot image into DRAM 126. Fig 4. illustrates temporal aspects of this and other boot operations performed in association with volatile memory (*e.g.*, DRAM 126) generally at 400 and with respect to time axis 402. Additionally, entities that are capable of performing each boot operation are shown above each of the various boot operations. In this example, processor core 116 performs validation operations and a memory controller, which is illustrated as SD host controller 404, transfers the boot images into DRAM 126 from eMMC device 206.

[0036] Returning to the context of operation 302, eMMC device 206 reads boot image-1 406 from internal flash in response to executing one of the packed commands. SD host controller 404 then loads boot image-1 406 into DRAM 126 via a DMA transfer operation in response to executing a transfer descriptor. By so doing, involvement of processor core 116 is unnecessary during the transfer of boot image-1 406 into DRAM 126.

[0037] At 304, a first validation operation for the first boot image is initiated once the transfer of the first boot image into the volatile memory is complete. The validation operation is performed to ensure security or authenticity of the first boot image or contents thereof. In some cases, a processor generates a hash value or checksum based on the contents of the first boot image. The generated hash value or checksum is then compared with a predetermined reference value to verify that the content of the boot image are unaltered or authentic. Alternately or additionally, the validation operation can be initiated in response to an indication that the first boot image is loaded into the

non-volatile memory. For example, a memory controller may trigger an interrupt on completion of the transfer of the first boot image or in response to execution of a descriptor associated therewith.

[0038] In the context of the present example, a DMA interrupt is triggered at the end of the transfer descriptor's execution by SD host controller 404. Responsive to the DMA interrupt, boot manager 132 signals processor core 116 to start validating boot image-1 406 in DRAM 126. Here, processor core 116 generates a hash value for boot image-1 406 that, when complete, is compared with a predefined reference value to determine the security of boot image-1 406.

[0039] At 306, a second transfer operation is initiated to transfer a second boot image from the non-volatile memory of the device. The second transfer operation transfers the second boot image to the volatile memory of the device. As with the first boot image, the second boot image may include any suitable amount of data, such as multiple blocks or pages of the boot code or binaries. In some cases, the first and the second boot images are stored in a same partition of the non-volatile memory. Additionally, the transfer operation can be initiated based on information included in the header image, such as a location, destination, or size of the second boot image.

[0040] The second transfer operation may be implemented in similar fashion to the first transfer operation of the first boot image, the details of which are described above and elsewhere herein. For example, the second transfer operation can be a DMA transfer implemented by a DMA-enabled memory controller executing descriptors and a non-volatile memory device executing a packed command sequence. By so doing,

the second boot image can be transferred into the volatile memory of the device without involving other entities, such as a processor core that is performing other operations.

[0041] Continuing the ongoing example, eMMC device 206 reads boot image-2 408 from internal flash in response to executing another one of the packed commands. SD host controller 404 then loads boot image-2 408 into DRAM 126 via another DMA transfer operation in response to executing another transfer descriptor. As illustrated in Fig. 4, boot image-2 408 is loaded into DRAM 126 while processor core 116 validates boot image-1 406. Alternately or additionally, sizes of each boot image may differ, which can result associated boot operations consuming differing amounts of time.

[0042] As noted with respect to operation 302, boot manager 132 can issue the packed commands and transfer descriptors at a start of the boot sequence. Execution of the respective packed commands or respective transfer descriptors can then continue throughout the boot sequence with little or no involvement from a processor or other higher-level controller. This can be effective to permit processor core 116 to perform validation operations on previously-transferred boot images while a next boot image is loaded into DRAM 126.

[0043] At 308, a second validation operation for the second boot image is initiated once the transfer of the second boot image into the volatile memory is complete. The validation operation is performed to ensure security or authenticity of the first boot image or contents thereof. The second validation operation may be implemented in similar fashion to the first validation operation of the second boot image, the details of which are described above and elsewhere herein. In the context of the present example,

another DMA interrupt is triggered at the end of the other transfer descriptor's execution by SD host controller 404. Responsive to the other DMA interrupt, boot manager 132 signals processor core 116 to start validating boot image-2 408 in DRAM 126. Here, processor core 116 generates a hash value for boot image-2 408 that, when complete, is compared with a predefined reference value to determine the security of boot image-2 408.

[0044] From operation 308, method 300 may return to operation 306 to initiate transfer of an additional boot image from the non-volatile memory of the device. Note that because operations 306 and 308 may be initiated in approximately concurrent fashion, method 300 may initiate the transfer of the additional boot image while a previously-transferred boot image is being validated. In some cases, transfer of a second boot image may complete before validation of a first boot image, such as when a first boot image (or any previously-transferred image) is larger than a second boot image. In these cases, a memory controller may initiate transfer of a third boot image without waiting for the validation of the first boot image to complete, which is effective to prevent validation operations from delaying any of the transfer operations. In the ongoing example, boot manager 132 initiates a transfer of boot image-3 410 from non-volatile memory and initiates validation of boot image-2 408. Boot manager 132 may then continue performing operations of method 300 to transfer and validate boot image-3 410 and boot image-4 412 as shown in Fig. 4.

[0045] At 310, control of the device is transferred to an executing boot image. In some cases, control is transferred from a first executing boot image to a next executing

boot image until control is transferred to an operating system of the device. By validating each successive boot image, a chain of trust is established throughout the boot process to ensure that unauthorized or malicious code, operating systems, or applications are prevented from executing on the device. Concluding the present example, boot manager 132 transfers control of the device to an operating system of IP TV 112 in response successfully validating and initiating execution of boot image-4.

[0046] Fig. 5 depicts a method 500 for constructing a command structure and transfer descriptors to enable parallel boot operations, including operations performed by boot manager 132 of Fig. 1.

[0047] At 502, a header image is read from a boot partition of a non-volatile memory. The header image can be located at a default location or address that is accessed in response to a device booting up. In some cases, a processor or controller of the device is configured to access the non-volatile memory, and the boot partition thereof, by default to boot the device. The header image may also be validated to ensure security or authenticity of the header image. Alternately or additionally, the header image may have a signature indicating that the header image is a first boot image of the boot partition.

[0048] At 504, the header image is interpreted to extract information that describes other boot images stored in the non-volatile memory. This information may include a location, destination, or size for each of the other boot images. For example, the information of the header image may indicate an address of a boot image in the non-volatile memory, a destination for the boot image in a volatile memory of the device,

and a size of the boot image. This information describing the boot images is useful to transfer the boot images from the non-volatile memory during a boot process of the device.

[0049] At 506, a packed command structure is constructed based on the information of the header image. By using a SET_BLOCK_COUNT command (CMD23) with a PACKED flag set and a block count of one, the packed command structure may include multiple read commands configured to transfer the boot images from the non-volatile memory. A WRITE_MULTIPLE_BLOCK command (CMD 25) can follow the block command with a header, which is followed by multiple READ_MULTIPLE_BLOCK commands (CMD18) to read the boot images. Issuing the packed command structure (*e.g.*, to an eMMC device) at the start of a boot sequence may enable multiple data transfers to be initiated over an SD bus without issuing individual read commands. This can be effective to reduce command overhead on the SD bus and reduce delays associated with boot image transfers.

[0050] At 508, transfer descriptors are constructed based on the information of the header image. The transfer descriptors can be ADMA2 descriptors configured for execution by an SD host controller. Similar to the packed command structure, the transfer descriptors enable a memory controller (*e.g.*, SD host controller) to treat the transfer of the boot images as one large transfer, rather than several individual transfers.

[0051] The descriptors may also be configured to trigger different types of interrupts, such as interrupts triggered at an end of a descriptors execution (*e.g.*, DMA interrupts) or when an entire transfer of a boot image is complete (*e.g.*, last descriptor

in a chain). Interrupts that trigger in response to complete execution of a transfer descriptor can be used to signal a processor or higher-level software to begin validation operations on a boot image transferred by the transfer descriptor. Note that for boot images spanning two or more transfer descriptors, the interrupt is configured to trigger when a last of the transfer descriptors executes for that boot image. Setting a “last descriptor” attribute of the last descriptor triggers a transfer complete interrupt that notifies the processor that the transfers on the bus are complete.

[0052] At 510, a boot sequence of the device is initiated using the packed command structure and the transfer descriptors. In some cases, this includes issuing the packed command structure to a memory controller integrated with the non-volatile memory and issuing the transfer descriptors to a system level memory controller. By so doing, transfer operations of the boot process can be performed in parallel with validation operations performed by a processor, which can significantly reduce delays in the boot sequence and improve device boot times.

[0053] Fig. 6 depicts a method 600 of booting a computing device with parallelized boot operations, including operations performed by boot manager 132 and processor core 116 of Fig. 1.

[0054] At 602, an electronic or computing device is started. Starting the device initiates a boot process during which components of the device are initialized and configured for use. The device may start responsive to a user input (*e.g.*, power button), power event, alarm, timer, and the like.

[0055] At 604, a security interface of the device is initialized. The security interface is configured to ensure the boot operations and other functionalities of the device are secure. In some cases, the security interface generates a hash value or checksum based on contents of a file or image. The security interface may then compare the generated hash value or checksum with a reference value to validate an authenticity or security of the file or image.

[0056] At 606, boot media of the device is configured. In some cases, the boot includes volatile memory of a device, such as DRAM or SRAM as described herein. Configuring the boot media may include applying power to the boot media, flushing the boot media, configuring address ranges of the boot media, and the like. Alternately or additionally, configuring the boot media includes configuring the boot media to interface with a system bus or memory controller, such as a DMA-enabled memory controller.

[0057] At 608, a header image is loaded from non-volatile memory of the device. The header image can be loaded into the boot media or another volatile memory of the device from which the header image can be processed. In some cases, the header image is located at a predefined or default location of the non-volatile memory that is accessed responsive to the device starting up.

[0058] At 610, the header image is validated to ensure security of the contents of the header image. This can be effective to prevent execution of unauthorized commands, which may be configured to load alternate or insecure boot images into the boot media. In some aspects, a hash value is generated based on the code of the header image, which

is then compared with a reference hash value to verify that the header image is secure. In cases in which the header image fails validation, the boot process the device can be halted or the device can be locked until a valid header image is loaded.

[0059] At 612, the header image is interpreted to construct commands and descriptors useful to boot the device. The commands may include a set of packed commands that are suitable for transmission to a memory controller and be configured to cause the memory controller to transfer multiple boot images. The descriptors may include ADMA descriptors that are configured to load multiple boot images into the boot media.

[0060] At 614, a boot image is loaded into the boot media from non-volatile memory, such as Flash memory or an eMMC device. In some cases, the boot image is sent by a memory controller of the non-volatile memory in response to execution of a first packed command. Alternately or additionally, the boot image can be loaded into the boot media via a DMA transfer operation in response to execution of a descriptor configured to implement the DMA operation. From operation 614, method 600 may proceed to operation 616 and to operation 618 in substantially parallel fashion.

[0061] At 616, it is determined whether another boot image is to be loaded from non-volatile memory. In some cases, this is indicated by the presence of another packed command awaiting execution. For example, the packed commands may include a sequence of READ_MULTIPLE_BLOCK commands configured to read multiple boot images. If another boot image is to be loaded from non-volatile memory, method 600 returns to operation 614, at which point a transfer operation of the other boot image is

initiated. Note here that operations 614 and 616 may, independent of other concurrent operations of method 600 (*e.g.*, 618 and 620), continue to repeat until any remaining boot images are loaded into the boot media.

[0062] At 618, the boot image in the boot media is validated to ensure security of the contents of the boot image. This can be effective to prevent execution of an unauthorized or insecure boot image, such as an image for operating system that permits malicious operations and programs. In some aspects, a hash value is generated based on the code of the boot image, which is and then compared with a reference hash value to verify that the boot image is secure. If the boot image fails validation, the boot process of the device can be halted or locked until another boot image is loaded. The validation of the boot image may be performed by a processor while a memory controller to loads a next boot image into the boot media. By so doing, various operation of the boot process can be parallelized, which may result in reduced boot times for the device.

[0063] At 620, it is determined whether another boot image in the boot media is to be validated. In some cases, this is indicated by an interrupt triggered in response to a descriptor completing execution or in response to completion of a DMA transfer operation. For example, at the end of descriptors execution by a memory controller, an interrupt may be triggered to alert the processor of another boot image's presence in the boot media. If another boot image in the boot media is to be validated, method 600 returns to operation 618, at which point a validation operation for the other boot image is initiated. Note here that operations 618 and 620 may, independent of other concurrent

operations of method 600 (*e.g.*, 614 and 616), continue to repeat until the remaining boot images in the boot media are validated. By so doing, various operation of the boot process can be parallelized, which may result in reduced boot times for the device. Once the boot images are validated, method 600 proceeds to operation 622.

[0064] At 622, the boot process transfers control of the device to one of the executing boot images. In some cases, control is transferred from a first executing boot image to a next executing boot image until control is transferred to an operating system of the device. By validating each successive boot image, a chain of trust is established throughout the boot process to ensure that unauthorized or malicious operating systems are prevented from executing on the device.

[0065] **System-on-Chip**

Fig. 7 illustrates another example System-on-Chip (SoC) 700, which can implement various aspects of parallelizing boot operations. A SoC can be implemented in any suitable device, such as a video game console, IP-enabled television, smart-phone, desktop computer, laptop computer, access point, wireless router, cellular broadband router, tablet computer, server, network-enabled printer, set-top box, printer, scanner, camera, picture frame, home appliance, thermostat, home automation device, and/or any other type of electronic device.

[0066] SoC 700 can be integrated with electronic circuitry, a microprocessor, memory, input-output (I/O) logic control, communication interfaces, other hardware, firmware, and/or software needed to provide functionalities of a device, such as any of

the above-listed devices. SoC 700 can also include an integrated data bus (not shown) that couples the various components of the SoC for data communication between the components. A memory storage device that includes SoC 700 can also be implemented with many combinations of differing components. In some cases, these differing components may be configured to implement concepts described herein over various internal or external data interfaces.

[0067] In this example, SoC 700 includes various components such as an input-output (I/O) logic control 702 (*e.g.*, to include electronic circuitry) and microprocessor 704. SoC 700 also includes memory 706, which can be any type and/or combination of RAM, SRAM, DRAM, Flash, low-latency nonvolatile memory, ROM, one-time programmable (OTP) memory, and/or other suitable electronic data storage. Alternately or additionally, SoC 700 may comprise data interfaces (not shown) for accessing additional or external off-chip memory, such as external SRAM, DRAM, or Flash memory. Although not shown, the external memory (or memory 706) may include multiple non-volatile memory devices storing multiple respective portions of boot code or boot code images, as described with reference to Fig. 4. SoC 700 can also include various firmware and/or software, such as operating system(s) 708, which can be computer-executable instructions maintained by memory 706 and executed by microprocessor 704. SoC 700 may also include other various communication interfaces and components embodied as hardware, firmware, software, or any suitable combination thereof.

[0068] SoC 700 also includes boot manager 132 and DMA controller 136 (either of which may embodied as disparate or combined components). Examples of these various components, functions, and/or entities, and their corresponding functionality, are described with reference to the respective components of the environment 100 and various configurations as illustrated by Figs. 2 and 4 (implemented as part of SD host controller 404). Boot manager 132, either independently or in combination with other entities (*e.g.*, DMA controller 136), can be implemented as computer-executable instructions maintained by memory 706 and executed by microprocessor 704 to implement various embodiments and/or features described herein. Boot manager 132 may also be provided integral with other entities of the SoC, such as integrated a memory controller associated with memory 706 or another suitable software, firmware, or hardware component within SoC 700. Alternatively or additionally, boot manager 132 and the other components can be implemented as hardware, firmware, fixed logic circuitry, or any combination thereof that is implemented in connection with the I/O logic control 702 and/or other signal processing and control circuits of SoC 700.

[0069] Although the subject matter has been described in language specific to structural features and/or methodological operations, it is to be understood that the subject matter defined in the appended claims is not necessarily limited to the specific features or operations described above, including orders in which they are performed.

CLAIMS

What is claimed is:

1. A method comprising:

initiating (302), via a direct memory access (DMA) controller, a first transfer operation transferring a first boot image from a first memory into a second memory;

initiating (304), after completion of the first transfer operation, a validation operation of the first boot image in the first memory effective to verify that the first boot image is secure; and

initiating (306), via the DMA controller and prior to completion of the validation operation, a second transfer operation transferring a second boot image from the first memory into the second memory.

2. The method as recited in claim 1, wherein the validation operation is a first validation operation and further comprising:

initiating (308), after completion of the second transfer operation, a second validation operation of the second boot image in the first memory; and

transferring (310), after completion of the second validation operation, control to the first boot image or the second boot image.

3. The method as recited in claim 1, wherein the validation operation comprises calculating a hash value for the first boot image and comparing the hash value with a predetermined reference value.
4. The method as recited in claim 1, further comprising triggering an interrupt to indicate completion of the first transfer operation.
5. The method as recited in claim 1, wherein the first memory is a Flash memory and the second memory is a random-access memory (RAM).
6. The method as recited in claim 5, wherein the Flash memory comprises one or a serial peripheral interface (SPI) Flash memory, NAND Flash memory, embedded Flash memory, or embedded multimedia card (eMMC) device.
7. The method as recited in claim 1, wherein the first boot image and the second boot image are located in a same partition of the first memory.

8. A method comprising:

retrieving (502), from a non-volatile memory, a header image describing multiple boot images stored in the non-volatile memory;

constructing (506), based on the header image, a structure describing respective locations of the multiple boot images stored in the non-volatile memory;

constructing (508), based on the header image, direct memory access (DMA) descriptors describing respective destinations of the multiple boot images in a volatile memory; and

issuing (510) the structure and DMA descriptors to a memory controller effective to cause the memory controller to transfer the multiple boot images from the non-volatile memory to the volatile memory, the transfer transferring at least one of the multiple boot images while a processor validates a previously-transferred one of the multiple boot images.

9. The method as recited in claim 8, wherein the non-volatile memory is Flash memory and structure is a packed command structure that describes respective sizes of the multiple boot images and respective addresses of the multiple boot images in the Flash memory.

10. The method as recited in claim 8, wherein the DMA descriptors are advanced DMA (ADMA) descriptors that describe respective sizes of the multiple boot images and respective destination addresses for the multiple boot images in the volatile memory.

11. The method as recited in claim 10, wherein an ADMA descriptor associated with transferring a first of the multiple boot images is configured to trigger an interrupt that indicates complete transfer of the first boot image.

12. The method as recited in claim 11, further comprising, responsive to the interrupt indicating completion of the transfer, causing the processor to initiate a validation operation of the first boot image in the volatile memory.

13. The method as recited in claim 10, wherein a sequence of multiple ADMA descriptors are associated with transferring one of the multiple boot images and an ADMA descriptor at an end of the sequence is configured to trigger an interrupt that indicates complete transfer of the boot image.

14. A System-on-Chip (202, 700) comprising:

a processor (116, 704) configured to implement validation operations and other functionalities of a device;

an interface (208) to a non-volatile memory storing multiple boot images;

a volatile memory (126, 706) configured to receive the multiple boot images;

a memory controller (136) configured to enable access to the non-volatile memory and the volatile memory; and

a boot manager (132) configured to:

initiate (302), via the memory controller, a first direct memory access (DMA) transfer operation transferring a first of the multiple boot images from the non-volatile memory to the volatile memory;

cause (304), after completion of the first DMA transfer operation, the processor to initiate a validation operation to validate the first boot image in the volatile memory; and

initiate (306), via the memory controller and prior to completion of the validation operation, a second DMA transfer operation transferring a second of the multiple boot images from the non-volatile memory to the volatile memory.

15. The System-on-Chip as recited in claim 14, wherein the validation operation is a first validation operation and the boot manager is further configured to:

cause (308), after completion of the second DMA transfer operation, the processor to initiate a second validation operation to validate the second boot image in the volatile memory; and

transfer (310), after completion of the second validation operation, control of the device to the first boot image or the second boot image.

16. The System-on-Chip as recited in claim 15, wherein the boot manager is further configured to:

access a header image stored in the non-volatile memory;

interpret the header image to provide information describing a size, destination, or location of the multiple boot images; and

perform the first and second DMA transfer operations based on the information provided by interpreting the header image.

17. The System-on-Chip as recited in claim 16, wherein the boot manager is further configured to construct advanced DMA (ADMA) descriptors based on the information provided by the header image and cause the memory controller to perform, based on the ADMA descriptors, the first and second DMA transfer operations.

18. The System-on-Chip as recited in claim 14, wherein the respective DMA transfer operations transfer the first and second boot images over an Advanced eXtensible Interface (AXI) bus, Advanced High Performance Bus (AHB), or Open Core Protocol (OCP) bus.

19. The System-on-Chip as recited in claim 14, wherein the memory controller operates in compliance with version 4.5 of the Joint Electron Device Engineering Council (JEDEC) Embedded Multimedia Card (eMMC) standard.

20. The System-on-Chip as recited in claim 14, wherein the multiple boot images are stored in a same partition of the non-volatile memory.

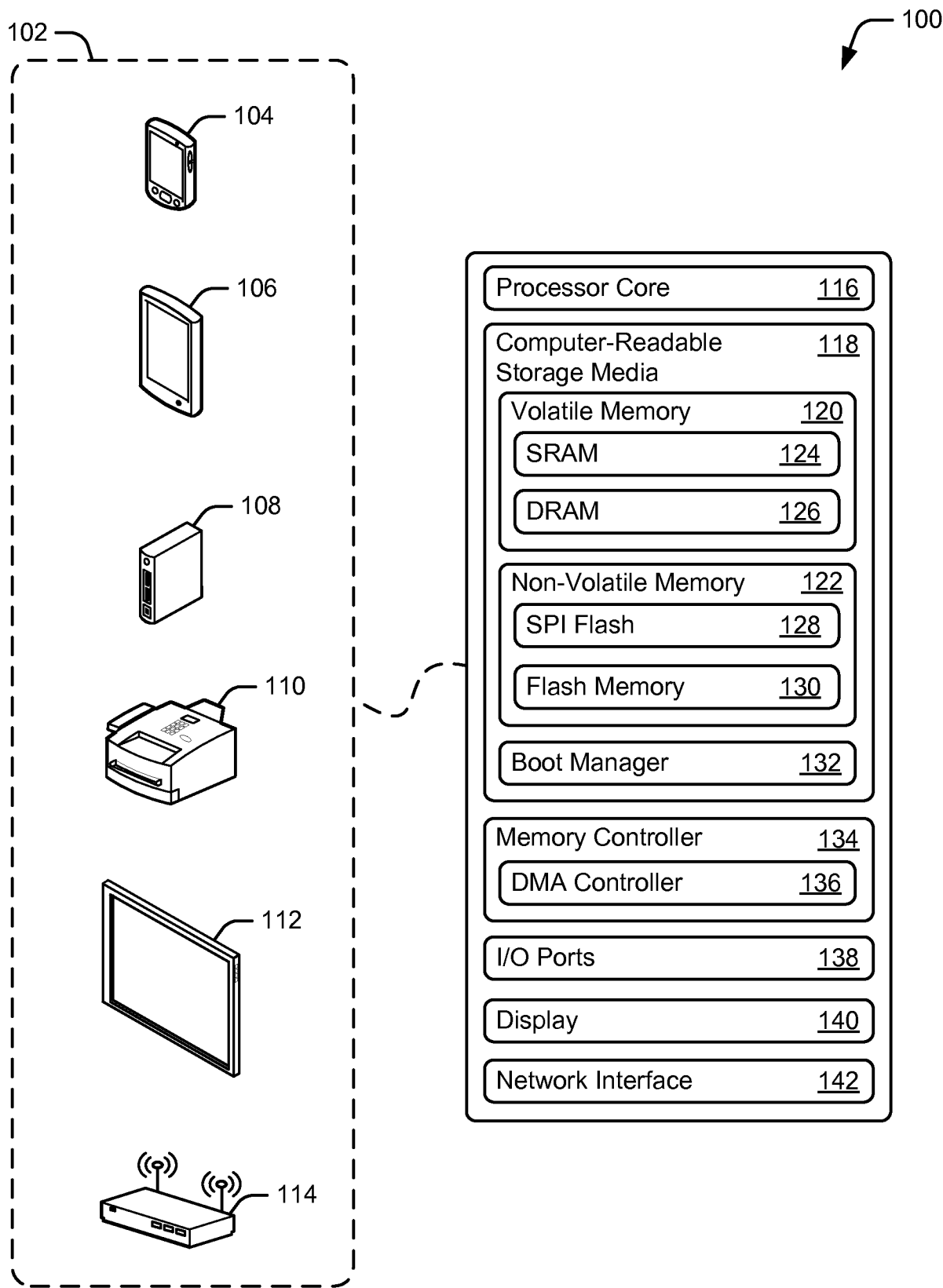


Fig. 1

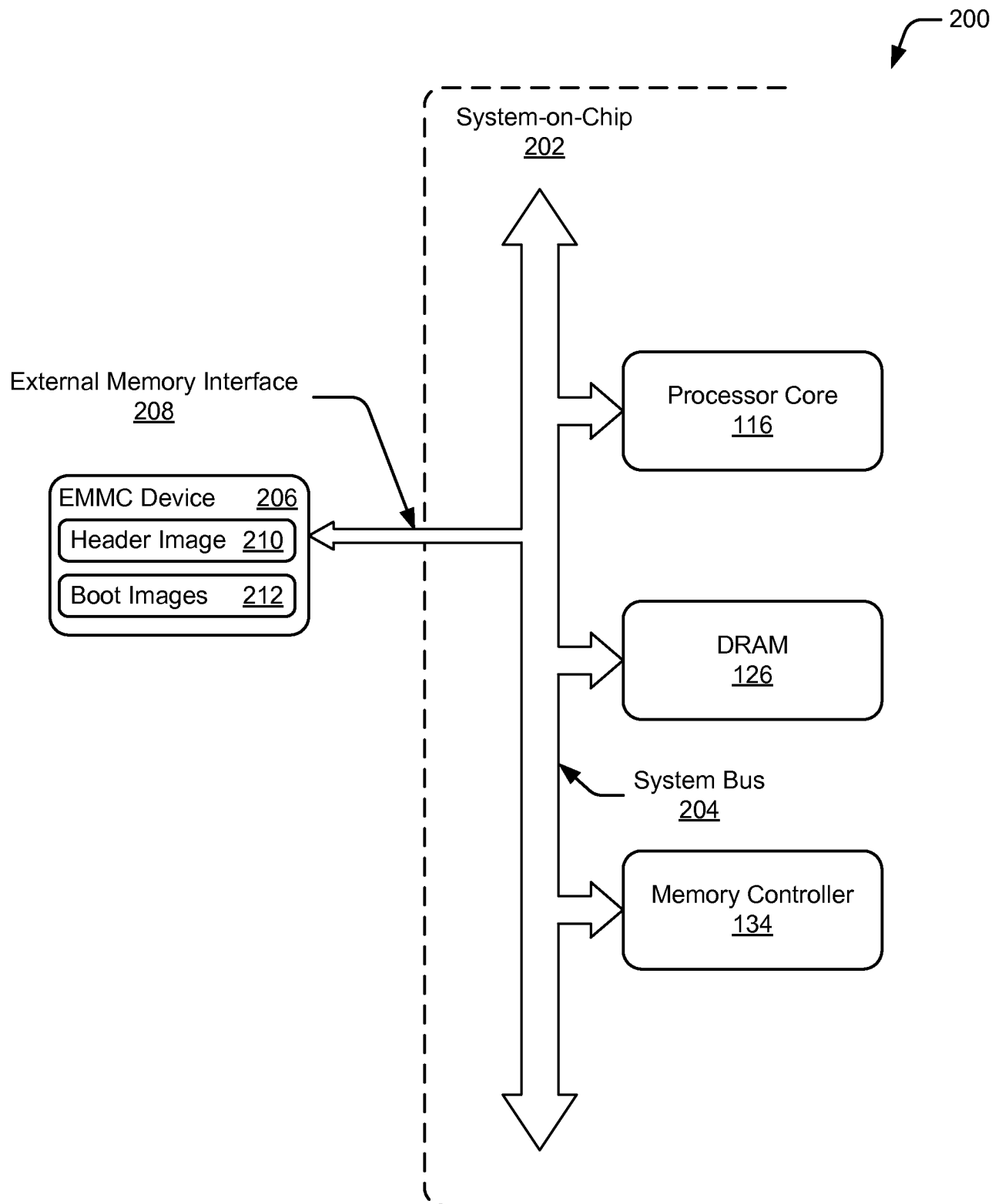
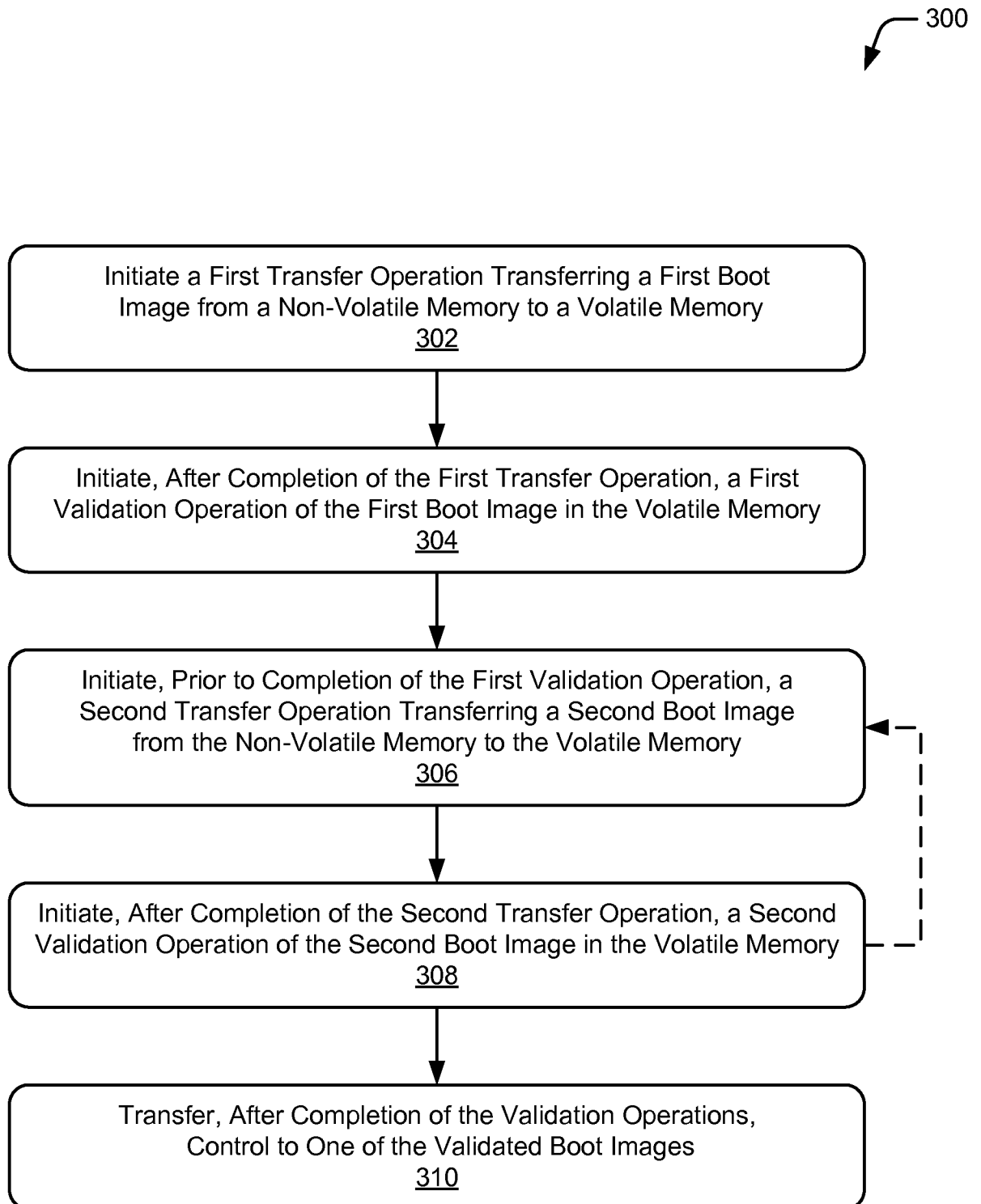


Fig. 2

**Fig. 3**

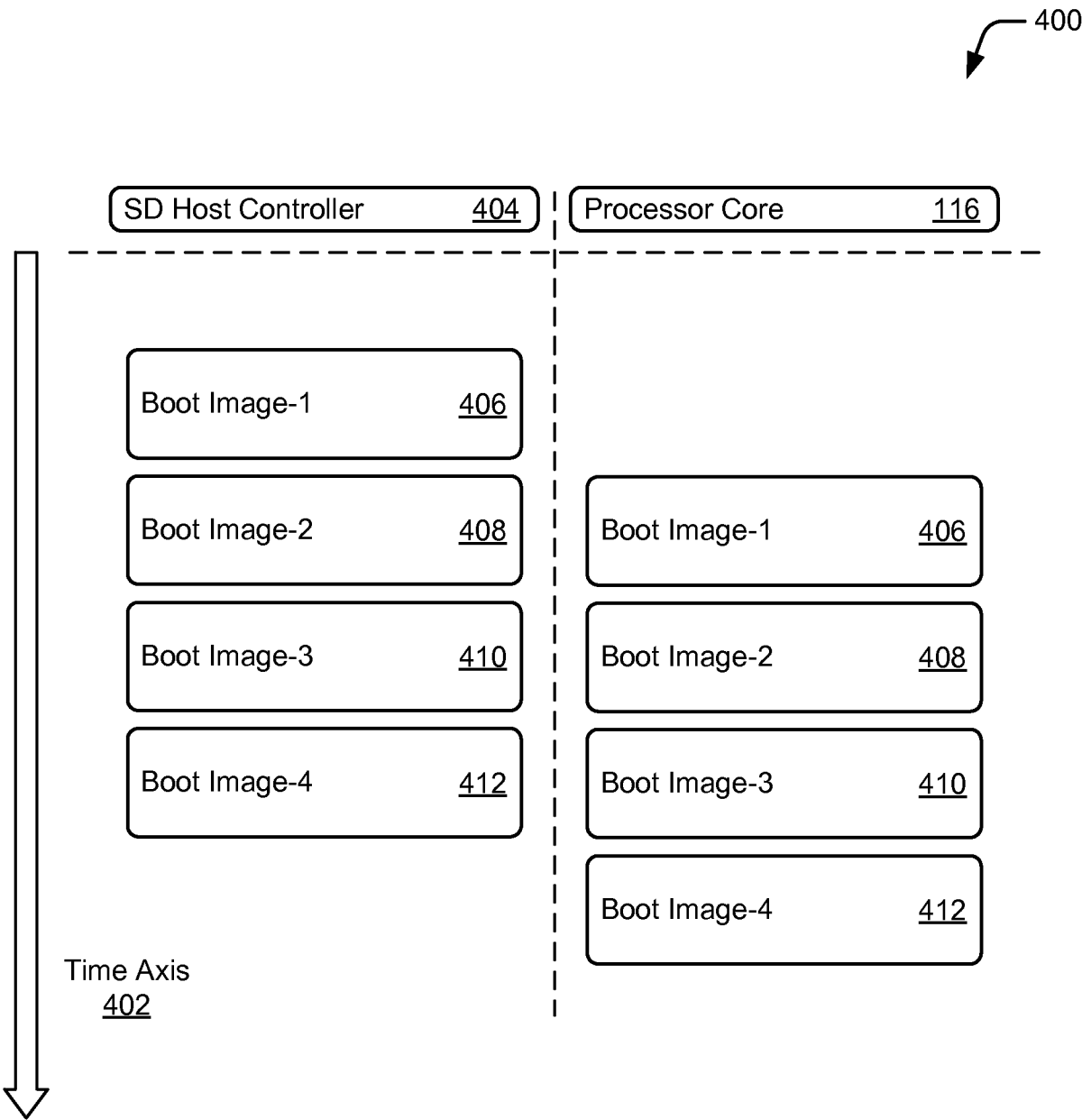
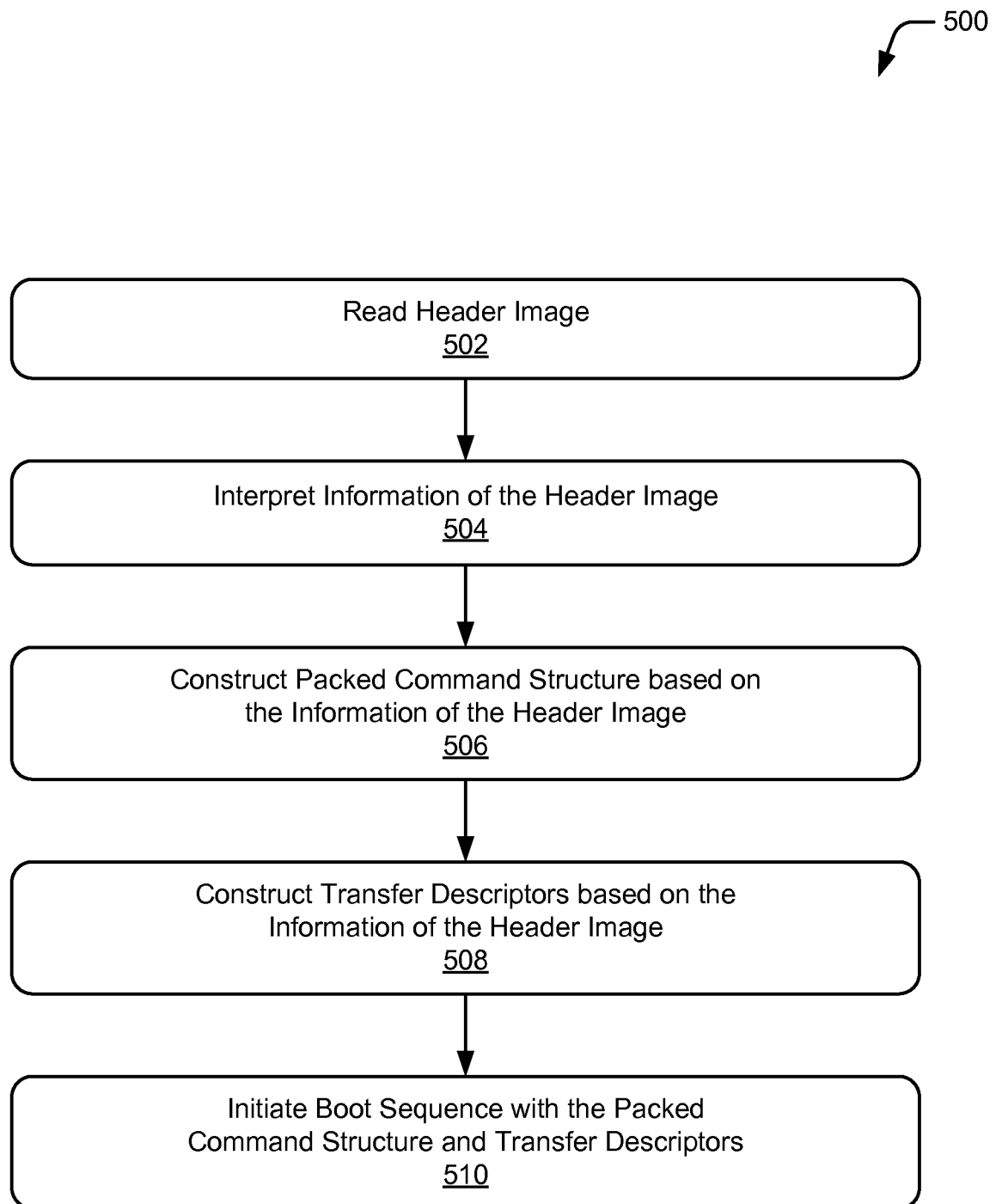


Fig. 4

**Fig. 5**

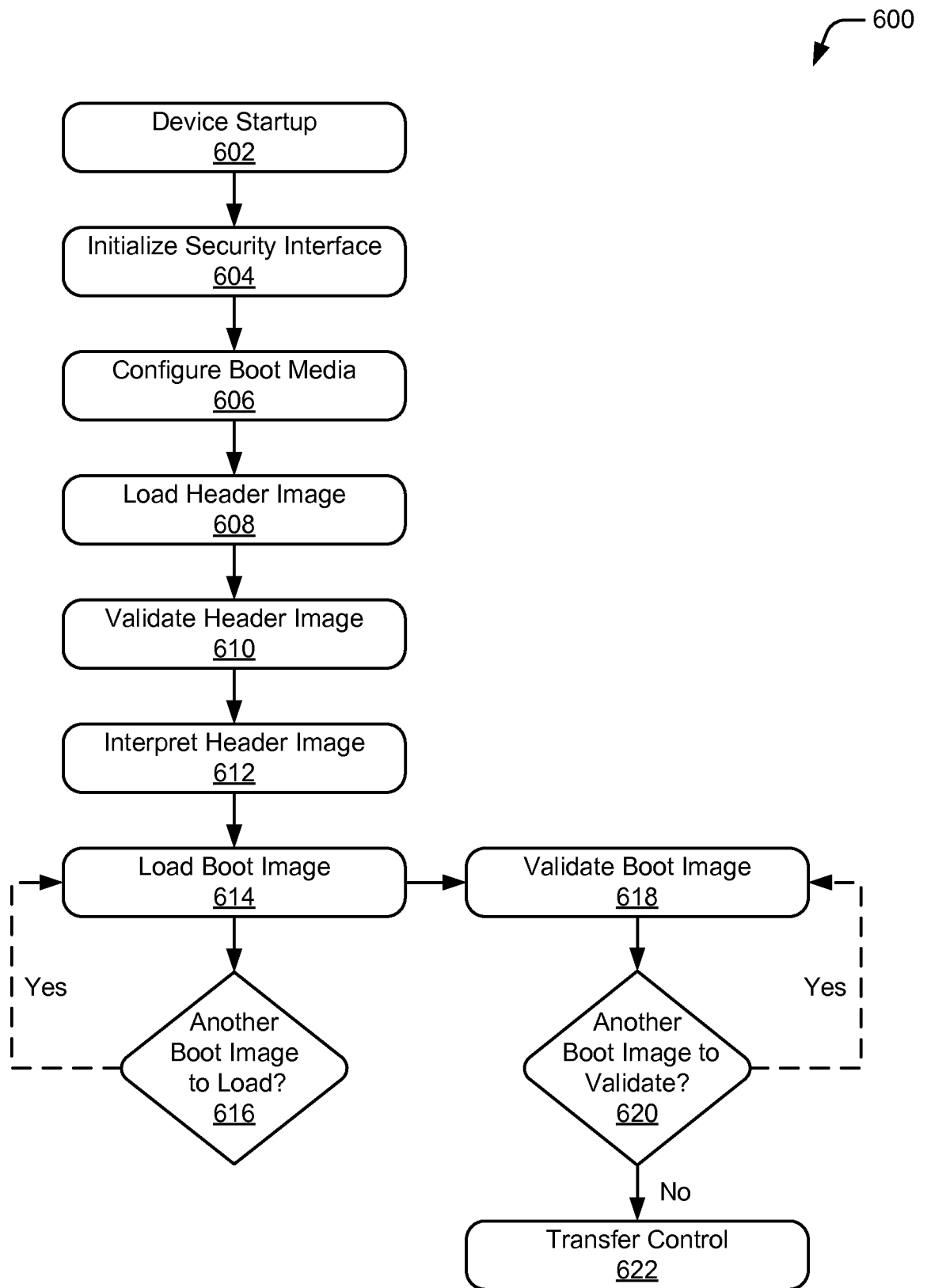


Fig. 6

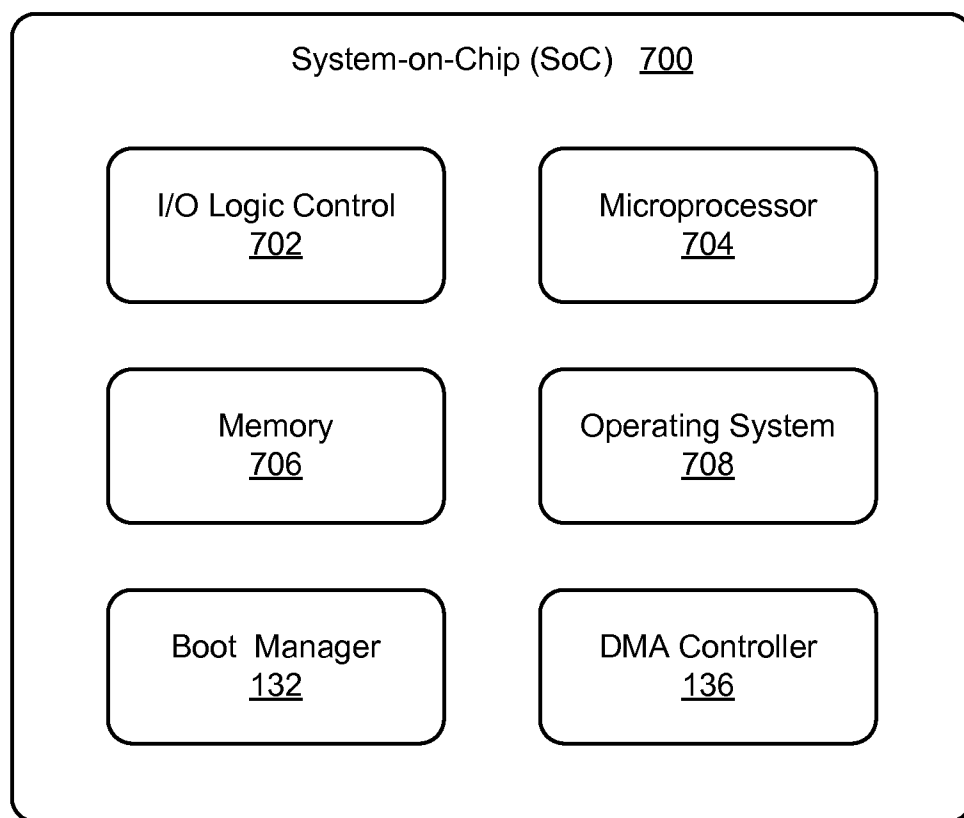


Fig. 7

INTERNATIONAL SEARCH REPORT

International application No

PCT/IB2014/002221

A. CLASSIFICATION OF SUBJECT MATTER

INV. G06F9/44

ADD.

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

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

G06F

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

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

EP0-Internal

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	JP 2006 285738 A (CANON KK) 19 October 2006 (2006-10-19) paragraph [0005] paragraph [0006] paragraph [0016]	1-20
A	----- US 2012/084547 A1 (JUNG MYUNG-JUNE [KR] ET AL) 5 April 2012 (2012-04-05) paragraph [0045] -----	1-20

☐ 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

28 November 2014

Date of mailing of the international search report

11/12/2014

Name and mailing address of the ISA/

European Patent Office, P.B. 5818 Patentlaan 2
NL - 2280 HV Rijswijk
Tel. (+31-70) 340-2040,
Fax: (+31-70) 340-3016

Authorized officer

Ebert, Werner

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No

PCT/IB2014/002221

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
JP 2006285738 A	19-10-2006	NONE	

US 2012084547 A1	05-04-2012	KR 20120035388 A	16-04-2012
		US 2012084547 A1	05-04-2012
