IN THE UNITED STATES DISTRICT COURT
FOR THE EASTERN DISTRICT OF TEXAS
MARSHALL DIVISION

HAMILCAR BARCA IP LLC,	§	
	§	
Plaintiff,	§	**Civil Action No. _____**
	§	
v.	§	**JURY TRIAL DEMANDED**
	§	
LENOVO GROUP LTD.,	§	
	§	
Defendant.	§	

COMPLAINT FOR PATENT INFRINGEMENT

Plaintiff Hamilcar Barca IP LLC ("Hamilcar" or "Plaintiff"), by and through its attorneys, for its Complaint against Defendant Lenovo Group Ltd. ("Lenovo" or "Defendant"), and demanding trial by jury, hereby alleges, on information and belief regarding the actions of Lenovo and third parties and on knowledge regarding its own actions, as follows[1]:

I. NATURE OF THE ACTION

1. This is an action for patent infringement arising under the patent laws of the United States, 35 U.S.C. § 1 et seq., including 35 U.S.C. §§ 271, 281, 283, 284, and 285, to obtain damages resulting from Defendant's unauthorized making, using, selling, offering to sell, and/or importing into the United States of products, methods, processes, services, and/or systems that have infringed Plaintiff's United States patents, as described herein.

2. Hamilcar in 2021 acquired a portfolio of patents originally developed by MediaTek relating to inventions by MediaTek Inc. in the 2000s—inventions that have since been growing in

[1] This Complaint is essentially identical to Hamilcar's Complaint filed last night in Civil Action No. 2:26-cv-00600-JRG-RSP. Counsel for Lenovo has asserted that the earlier Complaint, now dismissed, was filed prematurely. Plaintiff disagrees and reserves all rights but, to avoid unnecessary disputes, now files this new Complaint after midnight, Eastern Time.

popularity and concomitant value. Hamilcar contacted Lenovo about its infringement of the patents in 2023, offering to discuss the matter and a potential patent license. Hamilcar has sought to resolve this matter without litigation but is now left with no choice but to seek judicial relief.

3. Plaintiff seeks past damages and prejudgment and post-judgment interest for Defendant's infringement of the Asserted Patents, as defined below.

II. THE PARTIES

4. Plaintiff Hamilcar Barca IP LLC is a limited liability company organized and existing under the laws of the State of Texas, with its principal place of business located at 5204 Bluewater Drive, Frisco, Texas 75036.

5. Hamilcar is the owner of the entire right, title, and interest in the Asserted Patents, as defined below, including the right to sue for and collect all damages and to seek and obtain any other relief for infringement.

6. Defendant Lenovo Group Ltd. is a foreign company organized and existing under the laws of China, with its headquarters at 23rd Floor, Lincoln House, Taikoo Place, 979 King's Road, Quarry Bay, Hong Kong S.A.R. of China. Lenovo does business in Texas and in the Eastern District of Texas, directly or through intermediaries, such as its wholly owned subsidiaries. Lenovo is responsible for importing, making, marketing, distributing, offering for sale, and/or selling Lenovo-branded products in the United States (directly or through its wholly owned subsidiaries), including in this District.

7. Lenovo induces its subsidiaries, affiliates, retail partners, and customers in the making, using, selling, offering for sale, and/or importing throughout the United States, including within this District, of infringing products and places such products into the stream of commerce via established distribution channels knowing or understanding that such products would be sold and used in the United States, including in the Eastern District of Texas. For example, Lenovo

sells and offers to sell its products through its websites, including Lenovo.com, which may be accessed throughout the United States, the State of Texas, and this District. Additionally, Lenovo has authorized sellers and sales representatives that offer for sale and sell its products throughout the State of Texas and to consumers throughout this District.

8. Lenovo maintains a corporate presence in the United States via at least its wholly-owned subsidiaries, including Lenovo (United States) Inc. ("Lenovo US") and Lenovo Global Technology (United States) Inc. ("Lenovo Tech"). Lenovo is believed to hold 100% of the issued capital shares of each of Lenovo US and Lenovo Tech. Lenovo, Lenovo US, and Lenovo Tech are herein referred to collectively as the "Lenovo Group."

9. Lenovo US is a corporation organized and existing under the laws of Delaware, with a corporate headquarters located at 1009 Think Place, Morrisville, NC 27560. Lenovo US is registered to do business in the State of Texas.

10. Lenovo Tech is a corporation organized and existing under the laws of Delaware, with a corporate headquarters located at 1009 Think Place, Morrisville, NC 27560. Lenovo Tech is registered to do business in the State of Texas.

11. Lenovo and its U.S.-based subsidiaries (which act as part of a global network of sales and manufacturing subsidiaries) operate as agents of one another and vicariously as parts of the Lenovo Group to work in concert together. At the direction and control of Lenovo, the U.S.-based subsidiaries (including but not limited to Lenovo US and Lenovo Tech) make, use, import, offer to sell, and/or sell products that infringe the Asserted Patents, including in the State of Texas and this District. Lenovo, alone and through its U.S.-based subsidiaries, has derived substantial revenue from infringing acts in the United States, including from the sale and use of infringing products.

III. JURISDICTION AND VENUE

12. This action arises under the patent laws of the United States, Title 35 of the United States Code. This Court has exclusive subject matter jurisdiction over this action pursuant to 28 U.S.C. §§ 1331 and 1338(a).

13. This Court has personal jurisdiction over Lenovo because, directly or through wholly-owned and controlled subsidiaries, Lenovo has committed acts within the Eastern District of Texas giving rise to this action and/or has established minimum contacts with the Eastern District of Texas such that the exercise of jurisdiction would not offend traditional notions of fair play and substantial justice. Lenovo, directly or through its subsidiaries, purposefully and voluntarily placed infringing products into the stream of commerce with the knowledge and intention that they would be purchased and used by consumers in this District. District Courts in Texas have repeatedly concluded that Lenovo is subject to personal jurisdiction in the State of Texas. *See ACQIS LLC v. Lenovo Grp. Ltd.*, 572 F. Supp. 3d 291, 307 (W.D. Tex. 2021) ("this Court finds that the exercise of personal jurisdiction over [Lenovo Group Limited] is both reasonable and fair."); *see also AX Wireless LLC v. Lenovo Grp. Ltd.*, No. 2:22-cv-00280-RWS-RSP, Dkt. No. 110 (report and recommendation) (E.D. Tex. Sept. 6, 2023) ("exercising personal jurisdiction [over Lenovo Grp. Ltd.] would not offend traditional notions of fair play and substantial justice."); *Universal Connectivity Techs. Inc. v. Lenovo Grp. Ltd.*, No. 2:23-cv-00449-JRG, Dkt. No. 47 (Mem. Op. & Order) (E.D. Tex. Oct. 17, 2024) (denying Lenovo Group Ltd.'s motion to dismiss for lack of personal jurisdiction).

14. Alternatively, this Court has personal jurisdiction over Lenovo under Federal Rule of Civil Procedure 4(k)(2), because Plaintiff's claims arise under federal law, Lenovo is not subject to jurisdiction in any state's courts of general jurisdiction, and exercising jurisdiction is consistent with the United States Constitution and laws.

15. Venue is proper in this District under 28 U.S.C. § 1391(c) as to Lenovo because it is a foreign corporation organized under the laws of China, and suits against foreign entities are proper in any judicial district. *See* 28 U.S.C. § 1391(c)(3); *In re HTC Corp.*, 889 F.3d 1349 (Fed. Cir. 2018) (the patent venue statute, 28 U.S.C. § 1400(b), does not apply to foreign defendants, which may be sued in any judicial district).

IV. THE ASSERTED PATENTS

16. Plaintiff alleges that Defendant has infringed the following United States patents (collectively, the "Asserted Patents"):

United States Patent No. 8,086,938 (the "'938 Patent") (Exhibit A); and

United States Patent No. 8,407,783 (the "'783 Patent") (Exhibit B).

17. Hamilcar is the owner of all right, title, and interest in and to each of the Asserted Patents, including the right to sue for and recover all past, present, and future damages for infringement.

18. The '938 Patent expired on May 11, 2026. Accordingly, with respect to the '938 Patent, Plaintiff seeks past damages for Defendant's infringement occurring before expiration, together with prejudgment and post-judgment interest and costs, and does not seek injunctive relief as to the '938 Patent.

COUNT ONE

INFRINGEMENT OF U.S. PATENT NO. 8,086,938

19. Plaintiff incorporates by reference the allegations in all preceding paragraphs as if fully set forth herein.

20. The '938 Patent, entitled "Method for Processing Noise Interference," was filed on December 19, 2007 and claims priority to a Taiwan patent application filed on April 22, 2004. The

’938 Patent was duly and legally issued by the United States Patent and Trademark Office on December 27, 2011, and was originally assigned to MediaTek Inc.

21. The ’938 Patent is directed to patent-eligible subject matter and is valid and enforceable.

Technical Description and Background

22. The ’938 Patent is directed to a method for processing noise interference according to an error feedback mechanism of a Serial Advanced Technology Attachment (“SATA”). ’938 Patent at 1:15–26.

23. As the ’938 Patent explains, prior art SATA specifications specify a resending mechanism in which nearly all Frame Information Structures (“FIS”) are resent to prevent error caused by noises. ’938 Patent at 1:57–2:29. However, the data FIS is not protected by this resending mechanism because “the data FIS is not resent” in the event of data error. *Id*. at 2:29–33. As a result, “the error data may be received,” and in more serious instances, “some portions of primitives [(i.e., types of data control codes)] may be mistaken as data bits, thereby making the number of data sets to be sent to the device greater than it should be” and potentially halting the system. *Id*. at 2:33–37.

24. To overcome these and other problems, the ’938 Patent utilizes “a method for processing noise interference in order to prevent the system from being stopped when the data FIS is interfered by noises.” ’938 Patent at 2:48–51. The method “utilizes the ATA/ATAPI error feedback mechanism to overcome the problem of system halt caused by noise interference when the data FIS is being sent.” *Id*. at 3:41–44.

25. Claim 1, for example, includes a “type detecting step for detecting whether an FIS (Frame Information Structure) is a data type FIS” and a “responding step for asserting the CHECK

bit of the ATAPI Status Register when the FIS is data type." These steps are part of the process useful to overcome problems caused by noise interference when the FIS is a data type FIS.

Direct Infringement

26. Defendant, without authorization or license from Plaintiff, has directly infringed the '938 Patent, either literally or equivalently, as infringement is defined by 35 U.S.C. § 271, including through the making, using (including for testing purposes), importing, distributing, selling, and/or offering for sale of electronic devices and products that infringe one or more claims of the '938 Patent. Defendant is thus liable for direct infringement pursuant to 35 U.S.C. § 271.

27. Exemplary infringing products include, but are not limited to, any Lenovo device or system implementing SATA III (also known as SATA Revision 3.0) or later, including but not limited to Lenovo drives and other systems and devices having a Serial ATA Revision 3.0 or later interface, hereinafter the "'938 Accused Products."

28. Plaintiff names these exemplary infringing instrumentalities to serve as notice of Defendant's infringing acts, but Plaintiff reserves the right to name additional products, known to or learned by Plaintiff or revealed during discovery, and to include them more specifically in the definition of the '938 Accused Products.

29. As a specific, nonlimiting example, Defendant is liable for direct infringement pursuant to 35 U.S.C. § 271 for the use of devices that utilize the SATA 3.0 interface—for example, the Lenovo ThinkPad 512GB 2.5" Solid State Drive, hereinafter the "'938 Exemplary Accused Product." Use of the '938 Exemplary Accused Product met all limitations of at least claim 1 of the '938 Patent, either literally or under the doctrine of equivalents, as demonstrated in the attached claim chart marked as Exhibit C.

Indirect, Induced, and Contributory Infringement

30. Defendant, directly and/or through its subsidiaries, affiliates, agents, and/or business partners, has committed acts of indirect infringement of at least one claim of the ’938 Patent, pursuant to 35 U.S.C. §§ 271(b) and (c), by actively inducing or contributing to the acts of direct infringement performed by others in the United States, the State of Texas, and the Eastern District of Texas.

31. Defendant, through affirmative acts, has induced its distributors, manufacturers, testers, customers, and/or end users, such as designers and end users of the ’938 Accused Products, to directly infringe the ’938 Patent by using the ’938 Accused Products, with the specific intent to induce acts constituting infringement, and knowing that the induced acts constitute patent infringement, either literally or equivalently.

32. Defendant has knowingly contributed to direct infringement by its customers through affirmative acts and by having made, imported, sold, and/or offered for sale within the United States the ’938 Accused Products, which are not suitable for substantial non-infringing use and which are especially made or especially adapted for use by its customers in an infringement of the ’938 Patent.

33. The affirmative acts of inducement by Defendant include, but are not limited to, any one or a combination of: (i) designing infringing devices for manufacture according to specification; (ii) collaborating on and/or funding the development of the infringing devices and/or technology; (iii) soliciting and sourcing the manufacture of infringing devices; (iv) licensing and transferring technology and know-how to enable the manufacture of infringing devices; (v) enabling and encouraging the use, sale, or importation of infringing devices by its customers; (vi) advertising the infringing devices and/or technology; and (vii) providing data sheets, technical guides, demonstrations, software and hardware specifications, installation guides, product

specifications, user manuals, marketing materials, and instructions, including on Defendant's website.

34. Defendant has contributed to the direct infringement of the '938 Patent by its customers and other third parties; and Defendant, its customers, and other third parties have directly infringed.

35. As a result of Defendant's infringement, Plaintiff has suffered monetary damages and is entitled to an award of damages adequate to compensate it for such infringement which, by law, can be no less than a reasonable royalty, together with interest and costs as fixed by this Court under 35 U.S.C. § 284.

Willful Infringement

36. Defendant has had actual knowledge of the '938 Patent and its infringement thereof at least as of receipt of Plaintiff's notice letter dated October 16, 2023.

37. On information and belief, Defendant has numerous lawyers and other active agents who regularly review patents and published patent applications relevant to technology in the fields of the Asserted Patents.

38. Defendant or a related entity has been issued numerous patents, many of which are patents prosecuted in the USPTO in the same technology area as the '938 Patent, giving Defendant intimate knowledge of the art in fields relevant to this civil action. The timing, circumstances, and extent of Defendant obtaining actual knowledge of the '938 Patent prior to the commencement of this lawsuit will be confirmed during discovery.

39. Since at least October 16, 2023, when Defendant was expressly notified of its infringement, Defendant's infringement of the '938 Patent was either known or was so obvious that it should have been known to Defendant. Notwithstanding this knowledge, Defendant knowingly or with reckless disregard infringed the '938 Patent and continued to commit acts of

infringement despite being on notice of an objectively high likelihood that its actions constituted infringement of Plaintiff's valid patent rights, either literally or equivalently.

40. Defendant is therefore liable for willful infringement, and Plaintiff accordingly seeks enhanced damages pursuant to 35 U.S.C. §§ 284 and 285.

COUNT TWO

INFRINGEMENT OF U.S. PATENT NO. 8,407,783

41. Plaintiff incorporates by reference the allegations in all preceding paragraphs as if fully set forth herein.

42. The '783 Patent, entitled "Computing System Providing Normal Security and High Security Services," was filed on June 17, 2010 and was duly and legally issued by the United States Patent and Trademark Office on March 26, 2013, and was originally assigned to MediaTek Inc.

43. The '783 Patent is directed to patent-eligible subject matter and is valid and enforceable.

Technical Description and Background

44. The '783 Patent is directed to "computing systems providing normal security services and high security services with efficient resource utilization." '783 Patent at 1:8–10.

45. The '783 Patent explains that in many modern electronic devices, "two isolated operating environments are required for maintaining system security." '783 Patent at 1:12–16. Some services, such as "making a phone call and playing java games[,] may operate in a normal security environment." Id. at 1:17–19. However, some activities such as transacting with credit cards online or e-banking services may require higher security in order to protect sensitive personal information. Id. at 1:19–25.

46. The prior art relies on the TrustZone hardware architecture, developed by ARM, that uses a single processor core to provide both normal and high security environments, which

reduces silicon size, manufacturing costs, and power consumption compared to solutions with separate cores. ’783 Patent at 1:26–33. It switches between normal and high security states in a time-sliced manner to isolate sensitive data from potential threats. Id. at 1:33–40. However, the design at that time led to low utilization of resources dedicated to high security, as these resources are rarely used. Id. at 1:41–45. To improve resource usage, the processor may switch frequently between security states, but this increases latency and power consumption due to the switching process. Id. at 1:45–53.

47. The ’783 Patent simultaneously reduces switching of security environments while increasing utilization of hardware resources by allowing the processor to switch between normal and high security states and assigning user access rights to requests based on these states and the protection level of the computing system. ’783 Patent at 1:62–67, 6:51–57. In the high security state, user access rights grant permission to use hardware resources, which have been organized into different security levels, at a higher security level compared to the normal state. Id. at 1:67–2:3. The access right checker evaluates whether the request has the necessary authorization to access the required resources based on the assigned user access rights and the security levels of the resources. Id. at 2:3–7. If the request is authorized, it is allowed to execute; if not, an exception is triggered. Id. at 2:7–12.

Direct Infringement

48. Defendant, without authorization or license from Plaintiff, has directly infringed and continues to directly infringe the ’783 Patent, either literally or equivalently, as infringement is defined by 35 U.S.C. § 271, including through making, using (including for testing purposes), designing, manufacturing, importing, distributing, selling, and offering for sale electronic devices and products that infringe one or more claims of the ’783 Patent. Defendant is thus liable for direct infringement pursuant to 35 U.S.C. § 271.

49. Exemplary infringing products include, but are not necessarily limited to, the ARMv8-A-compliant data or central processing units, and other systems and devices made, imported, or sold by Lenovo implementing the ARMv8-A or later architecture (e.g., ARMv9-A) (the "'783 Accused Products"). These include, for example, Lenovo servers and other systems and devices implementing the ARMv8-A or later architecture, including but not necessarily limited to those with one or more ARMv8-A or later compliant CPUs.

50. Plaintiff names these exemplary infringing instrumentalities to serve as notice of Defendant's infringing acts, but Plaintiff reserves the right to name additional infringing products, known to or learned by Plaintiff or revealed during discovery, and to include them more specifically in the definition of the '783 Accused Products.

51. As another specific, nonlimiting example, Defendant is liable for direct infringement pursuant to 35 U.S.C. § 271 for the manufacture, use, sale, offer for sale, importation, or distribution of integrated circuit devices and systems implementing the ARMv8-A or later architecture, such as the Lenovo ThinkPad T14s Gen 6, hereinafter the "'783 Exemplary Accused Product." The '783 Exemplary Accused Product meets all limitations of at least claim 1 of the '783 Patent, either literally or under the doctrine of equivalents, as demonstrated in the attached claim chart marked as Exhibit D.

Indirect, Induced, and Contributory Infringement

52. Defendant, directly and/or through its subsidiaries, affiliates, agents, and/or business partners, has committed and continues to commit acts of indirect infringement of at least one claim of the '783 Patent, pursuant to 35 U.S.C. §§ 271(b) and (c), by actively inducing or contributing to the acts of direct infringement performed by others in the United States, the State of Texas, and the Eastern District of Texas.

53. Defendant, through affirmative acts, has induced its distributors, manufacturers, testers, customers, and/or end users, such as designers and end users of devices and products implementing the ARMv8-A or later architecture, to directly infringe the ’783 Patent by making, using, selling, and/or importing the ’783 Accused Products, with the specific intent to induce acts constituting infringement, and knowing that the induced acts constitute patent infringement, either literally or equivalently.

54. Defendant has knowingly contributed to direct infringement by its customers through affirmative acts and by having imported, made, sold, and/or offered for sale within the United States the ’783 Accused Products, which are not suitable for substantial non-infringing use and which are especially made or especially adapted for use by its customers in an infringement of the ’783 Patent.

55. The affirmative acts of inducement by Defendant include, but are not limited to, any one or a combination of: (i) designing infringing systems for manufacture according to specification; (ii) collaborating on and/or funding the development of the infringing systems and/or technology; (iii) soliciting and sourcing the manufacture of infringing systems; (iv) licensing and transferring technology and know-how to enable the manufacture of infringing systems; (v) enabling and encouraging the use, sale, or importation of infringing systems by its customers; (vi) advertising the infringing systems and/or technology; and (vii) providing data sheets, technical guides, demonstrations, software and hardware specifications, installation guides, product specifications, user manuals, marketing materials, and instructions.

56. Defendant has contributed to the direct infringement of the ’783 Patent by its customers and other third parties; and Defendant, its customers, and other third parties have directly infringed.

57. As a result of Defendant's infringement, Plaintiff has suffered monetary damages and is entitled to an award of damages adequate to compensate it for such infringement which, by law, can be no less than a reasonable royalty, together with interest and costs as fixed by this Court under 35 U.S.C. § 284. Plaintiff has incurred and will continue to incur substantial damages, including monetary damages.

Willful Infringement

58. Defendant has had actual knowledge of the '783 Patent and its infringement thereof at least as of receipt of Plaintiff's notice letter dated October 16, 2023.

59. On information and belief, Defendant has numerous lawyers and other active agents who regularly review patents and published patent applications relevant to the technology in the fields of the Asserted Patents.

60. Defendant or a related entity has been issued numerous patents, many of which are patents prosecuted in the USPTO in the same technology area as the '783 Patent, giving Defendant intimate knowledge of the art in fields relevant to this civil action. The timing, circumstances, and extent of Defendant obtaining actual knowledge of the '783 Patent prior to the commencement of this lawsuit will be confirmed during discovery.

61. Since at least October 16, 2023, when Defendant was expressly notified of its infringement, Defendant's infringement of the '783 Patent was either known or was so obvious that it should have been known to Defendant. Notwithstanding this knowledge, Defendant has knowingly or with reckless disregard continued to infringe the '783 Patent and has continued to commit acts of infringement despite being on notice of an objectively high likelihood that its actions constitute infringement of Plaintiff's valid patent rights, either literally or equivalently.

62. Defendant is therefore liable for willful infringement, and Plaintiff accordingly seeks enhanced damages pursuant to 35 U.S.C. §§ 284 and 285.

V. NOTICE

63. Plaintiff has complied with the notice provision of 35 U.S.C. § 287 and has not distributed, sold, offered for sale, or made any patented articles embodying the Asserted Patents. This notice provision has been complied with by all relevant persons at all relevant times.

64. Defendant has had actual knowledge of the Asserted Patents and its infringement thereof at least since receipt of Plaintiff's notice letter dated October 16, 2023.

VI. JURY DEMAND

65. Plaintiff demands a trial by jury of all matters to which it is entitled to trial by jury, pursuant to Federal Rule of Civil Procedure 38.

VII. PRAYER FOR RELIEF

WHEREFORE, Plaintiff prays for judgment and seeks relief against Defendant as follows:

A. That the Court determine that one or more claims of the '938 Patent has been infringed by Defendant, literally and/or under the doctrine of equivalents;

B. That the Court determine that one or more claims of the '783 Patent has been and continues to be infringed by Defendant, literally and/or under the doctrine of equivalents;

C. That the Court determine that one or more claims of the '938 Patent has been indirectly infringed by Defendant;

D. That the Court determine that one or more claims of the '783 Patent has been and continues to be indirectly infringed by Defendant;

E. That the Court award damages adequate to compensate Plaintiff for the patent infringement that has occurred, together with prejudgment and post-judgment interest and costs, including past damages for infringement of the '938 Patent occurring prior to its expiration on May 11, 2026;

F. That the Court find this case to be exceptional pursuant to 35 U.S.C. § 285;

G. That the Court determine that Defendant's infringements have been willful;

H. That the Court award enhanced damages against Defendant pursuant to 35 U.S.C. § 284;

I. That the Court award Plaintiff its reasonable attorneys' fees; and

J. That the Court award such other relief to Plaintiff as the Court deems just and proper.

VIII. RESERVATION OF RIGHTS

Plaintiff's investigation is ongoing, and certain material information remains in the sole possession of Defendant or third parties, which will be obtained via discovery herein. Plaintiff expressly reserves the right to amend or supplement the causes of action set forth herein in accordance with Federal Rule of Civil Procedure 15.

Dated: July 22, 2026 CT

Respectfully submitted,

/s/ Scott W. Breedlove

E. Leon Carter
lcarter@carterarnett.com
Texas Bar No. 03914300
Scott W. Breedlove
sbreedlove@carterarnett.com
Texas Bar No. 00790361
Omer Salik
osalik@carterarnett.com
Texas Bar No. 24128282
Howard L. Lim
hlim@carterarnett.com
Texas Bar No. 24092701
CARTER ARNETT STAHL + CHO HERNANDEZ PLLC
8150 N. Central Expressway, 5th Floor
Dallas, Texas 75206
Telephone: (214) 550-8188
Facsimile: (214) 550-8185

ATTORNEYS FOR PLAINTIFF HAMILCAR BARCA IP LLC

JS 44 (Rev. 03/24)

CIVIL COVER SHEET

The JS 44 civil cover sheet and the information contained herein neither replace nor supplement the filing and service of pleadings or other papers as required by law, except as provided by local rules of court. This form, approved by the Judicial Conference of the United States in September 1974, is required for the use of the Clerk of Court for the purpose of initiating the civil docket sheet. *(SEE INSTRUCTIONS ON NEXT PAGE OF THIS FORM.)*

I. (a) PLAINTIFFS

Hamilcar Barca IP LLC

(b) County of Residence of First Listed Plaintiff Denton
(EXCEPT IN U.S. PLAINTIFF CASES)

(c) Attorneys *(Firm Name, Address, and Telephone Number)*
E. Leon Carter, Scott W. Breedlove, Omer Salik, and Howard L. Lim; Carter Arnett Stahl + Cho Hernandez PLLC, 8150 N. Central Expressway, Suite 500, Dallas,

DEFENDANTS

Lenovo Group Ltd.

County of Residence of First Listed Defendant
(IN U.S. PLAINTIFF CASES ONLY)

NOTE: IN LAND CONDEMNATION CASES, USE THE LOCATION OF THE TRACT OF LAND INVOLVED.

Attorneys *(If Known)*

II. BASIS OF JURISDICTION *(Place an "X" in One Box Only)*

- [ ] 1 U.S. Government Plaintiff
- [x] 3 Federal Question *(U.S. Government Not a Party)*
- [ ] 2 U.S. Government Defendant
- [ ] 4 Diversity *(Indicate Citizenship of Parties in Item III)*

III. CITIZENSHIP OF PRINCIPAL PARTIES *(Place an "X" in One Box for Plaintiff and One Box for Defendant)*
(For Diversity Cases Only)

	PTF	DEF		PTF	DEF
Citizen of This State	[ ] 1	[ ] 1	Incorporated *or* Principal Place of Business In This State	[ ] 4	[ ] 4
Citizen of Another State	[ ] 2	[ ] 2	Incorporated *and* Principal Place of Business In Another State	[ ] 5	[ ] 5
Citizen or Subject of a Foreign Country	[ ] 3	[ ] 3	Foreign Nation	[ ] 6	[ ] 6

IV. NATURE OF SUIT *(Place an "X" in One Box Only)*

Click here for: Nature of Suit Code Descriptions.

CONTRACT
- [ ] 110 Insurance
- [ ] 120 Marine
- [ ] 130 Miller Act
- [ ] 140 Negotiable Instrument
- [ ] 150 Recovery of Overpayment & Enforcement of Judgment
- [ ] 151 Medicare Act
- [ ] 152 Recovery of Defaulted Student Loans (Excludes Veterans)
- [ ] 153 Recovery of Overpayment of Veteran's Benefits
- [ ] 160 Stockholders' Suits
- [ ] 190 Other Contract
- [ ] 195 Contract Product Liability
- [ ] 196 Franchise

REAL PROPERTY
- [ ] 210 Land Condemnation
- [ ] 220 Foreclosure
- [ ] 230 Rent Lease & Ejectment
- [ ] 240 Torts to Land
- [ ] 245 Tort Product Liability
- [ ] 290 All Other Real Property

TORTS

PERSONAL INJURY
- [ ] 310 Airplane
- [ ] 315 Airplane Product Liability
- [ ] 320 Assault, Libel & Slander
- [ ] 330 Federal Employers' Liability
- [ ] 340 Marine
- [ ] 345 Marine Product Liability
- [ ] 350 Motor Vehicle
- [ ] 355 Motor Vehicle Product Liability
- [ ] 360 Other Personal Injury
- [ ] 362 Personal Injury - Medical Malpractice

PERSONAL INJURY
- [ ] 365 Personal Injury - Product Liability
- [ ] 367 Health Care/ Pharmaceutical Personal Injury Product Liability
- [ ] 368 Asbestos Personal Injury Product Liability

PERSONAL PROPERTY
- [ ] 370 Other Fraud
- [ ] 371 Truth in Lending
- [ ] 380 Other Personal Property Damage
- [ ] 385 Property Damage Product Liability

CIVIL RIGHTS
- [ ] 440 Other Civil Rights
- [ ] 441 Voting
- [ ] 442 Employment
- [ ] 443 Housing/ Accommodations
- [ ] 445 Amer. w/Disabilities - Employment
- [ ] 446 Amer. w/Disabilities - Other
- [ ] 448 Education

PRISONER PETITIONS

Habeas Corpus:
- [ ] 463 Alien Detainee
- [ ] 510 Motions to Vacate Sentence
- [ ] 530 General
- [ ] 535 Death Penalty

Other:
- [ ] 540 Mandamus & Other
- [ ] 550 Civil Rights
- [ ] 555 Prison Condition
- [ ] 560 Civil Detainee - Conditions of Confinement

FORFEITURE/PENALTY
- [ ] 625 Drug Related Seizure of Property 21 USC 881
- [ ] 690 Other

LABOR
- [ ] 710 Fair Labor Standards Act
- [ ] 720 Labor/Management Relations
- [ ] 740 Railway Labor Act
- [ ] 751 Family and Medical Leave Act
- [ ] 790 Other Labor Litigation
- [ ] 791 Employee Retirement Income Security Act

IMMIGRATION
- [ ] 462 Naturalization Application
- [ ] 465 Other Immigration Actions

BANKRUPTCY
- [ ] 422 Appeal 28 USC 158
- [ ] 423 Withdrawal 28 USC 157

INTELLECTUAL PROPERTY RIGHTS
- [ ] 820 Copyrights
- [x] 830 Patent
- [ ] 835 Patent - Abbreviated New Drug Application
- [ ] 840 Trademark
- [ ] 880 Defend Trade Secrets Act of 2016

SOCIAL SECURITY
- [ ] 861 HIA (1395ff)
- [ ] 862 Black Lung (923)
- [ ] 863 DIWC/DIWW (405(g))
- [ ] 864 SSID Title XVI
- [ ] 865 RSI (405(g))

FEDERAL TAX SUITS
- [ ] 870 Taxes (U.S. Plaintiff or Defendant)
- [ ] 871 IRS—Third Party 26 USC 7609

OTHER STATUTES
- [ ] 375 False Claims Act
- [ ] 376 Qui Tam (31 USC 3729(a))
- [ ] 400 State Reapportionment
- [ ] 410 Antitrust
- [ ] 430 Banks and Banking
- [ ] 450 Commerce
- [ ] 460 Deportation
- [ ] 470 Racketeer Influenced and Corrupt Organizations
- [ ] 480 Consumer Credit (15 USC 1681 or 1692)
- [ ] 485 Telephone Consumer Protection Act
- [ ] 490 Cable/Sat TV
- [ ] 850 Securities/Commodities/ Exchange
- [ ] 890 Other Statutory Actions
- [ ] 891 Agricultural Acts
- [ ] 893 Environmental Matters
- [ ] 895 Freedom of Information Act
- [ ] 896 Arbitration
- [ ] 899 Administrative Procedure Act/Review or Appeal of Agency Decision
- [ ] 950 Constitutionality of State Statutes

V. ORIGIN *(Place an "X" in One Box Only)*

- [x] 1 Original Proceeding
- [ ] 2 Removed from State Court
- [ ] 3 Remanded from Appellate Court
- [ ] 4 Reinstated or Reopened
- [ ] 5 Transferred from Another District *(specify)*
- [ ] 6 Multidistrict Litigation - Transfer
- [ ] 8 Multidistrict Litigation - Direct File

VI. CAUSE OF ACTION

Cite the U.S. Civil Statute under which you are filing *(Do not cite jurisdictional statutes unless diversity)*:
35 U.S.C. § 1 et seq., including 35 U.S.C. §§ 271, 281, 283, 284, and 285

Brief description of cause:
Patent Infringement

VII. REQUESTED IN COMPLAINT:

[ ] CHECK IF THIS IS A CLASS ACTION UNDER RULE 23, F.R.Cv.P.

DEMAND $

CHECK YES only if demanded in complaint:
JURY DEMAND: [x] Yes [ ] No

VIII. RELATED CASE(S) IF ANY

(See instructions): JUDGE _____ DOCKET NUMBER _____

DATE
07-22-2026

SIGNATURE OF ATTORNEY OF RECORD
/s/ Scott W. Breedlove

FOR OFFICE USE ONLY

RECEIPT # _____ AMOUNT _____ APPLYING IFP _____ JUDGE _____ MAG. JUDGE _____

INSTRUCTIONS FOR ATTORNEYS COMPLETING CIVIL COVER SHEET FORM JS 44

Authority For Civil Cover Sheet

The JS 44 civil cover sheet and the information contained herein neither replaces nor supplements the filings and service of pleading or other papers as required by law, except as provided by local rules of court. This form, approved by the Judicial Conference of the United States in September 1974, is required for the use of the Clerk of Court for the purpose of initiating the civil docket sheet. Consequently, a civil cover sheet is submitted to the Clerk of Court for each civil complaint filed. The attorney filing a case should complete the form as follows:

I.(a) **Plaintiffs-Defendants.** Enter names (last, first, middle initial) of plaintiff and defendant. If the plaintiff or defendant is a government agency, use only the full name or standard abbreviations. If the plaintiff or defendant is an official within a government agency, identify first the agency and then the official, giving both name and title.

(b) **County of Residence.** For each civil case filed, except U.S. plaintiff cases, enter the name of the county where the first listed plaintiff resides at the time of filing. In U.S. plaintiff cases, enter the name of the county in which the first listed defendant resides at the time of filing. (NOTE: In land condemnation cases, the county of residence of the "defendant" is the location of the tract of land involved.)

(c) **Attorneys.** Enter the firm name, address, telephone number, and attorney of record. If there are several attorneys, list them on an attachment, noting in this section "(see attachment)".

II. **Jurisdiction.** The basis of jurisdiction is set forth under Rule 8(a), F.R.Cv.P., which requires that jurisdictions be shown in pleadings. Place an "X" in one of the boxes. If there is more than one basis of jurisdiction, precedence is given in the order shown below.
United States plaintiff. (1) Jurisdiction based on 28 U.S.C. 1345 and 1348. Suits by agencies and officers of the United States are included here.
United States defendant. (2) When the plaintiff is suing the United States, its officers or agencies, place an "X" in this box.
Federal question. (3) This refers to suits under 28 U.S.C. 1331, where jurisdiction arises under the Constitution of the United States, an amendment to the Constitution, an act of Congress or a treaty of the United States. In cases where the U.S. is a party, the U.S. plaintiff or defendant code takes precedence, and box 1 or 2 should be marked.
Diversity of citizenship. (4) This refers to suits under 28 U.S.C. 1332, where parties are citizens of different states. When Box 4 is checked, the citizenship of the different parties must be checked. (See Section III below**; NOTE: federal question actions take precedence over diversity cases.**)

III. **Residence (citizenship) of Principal Parties.** This section of the JS 44 is to be completed if diversity of citizenship was indicated above. Mark this section for each principal party.

IV. **Nature of Suit.** Place an "X" in the appropriate box. If there are multiple nature of suit codes associated with the case, pick the nature of suit code that is most applicable. Click here for: Nature of Suit Code Descriptions.

V. **Origin.** Place an "X" in one of the seven boxes.
Original Proceedings. (1) Cases which originate in the United States district courts.
Removed from State Court. (2) Proceedings initiated in state courts may be removed to the district courts under Title 28 U.S.C., Section 1441.
Remanded from Appellate Court. (3) Check this box for cases remanded to the district court for further action. Use the date of remand as the filing date.
Reinstated or Reopened. (4) Check this box for cases reinstated or reopened in the district court. Use the reopening date as the filing date.
Transferred from Another District. (5) For cases transferred under Title 28 U.S.C. Section 1404(a). Do not use this for within district transfers or multidistrict litigation transfers.
Multidistrict Litigation – Transfer. (6) Check this box when a multidistrict case is transferred into the district under authority of Title 28 U.S.C. Section 1407.
Multidistrict Litigation – Direct File. (8) Check this box when a multidistrict case is filed in the same district as the Master MDL docket.
PLEASE NOTE THAT THERE IS NOT AN ORIGIN CODE 7. Origin Code 7 was used for historical records and is no longer relevant due to changes in statute.

VI. **Cause of Action.** Report the civil statute directly related to the cause of action and give a brief description of the cause. **Do not cite jurisdictional statutes unless diversity.** Example: U.S. Civil Statute: 47 USC 553 Brief Description: Unauthorized reception of cable service.

VII. **Requested in Complaint.** Class Action. Place an "X" in this box if you are filing a class action under Rule 23, F.R.Cv.P.
Demand. In this space enter the actual dollar amount being demanded or indicate other demand, such as a preliminary injunction.
Jury Demand. Check the appropriate box to indicate whether or not a jury is being demanded.

VIII. **Related Cases.** This section of the JS 44 is used to reference related cases, if any. If there are related cases, insert the docket numbers and the corresponding judge names for such cases.

Date and Attorney Signature. Date and sign the civil cover sheet.

EXHIBIT A

US008086938B2

(12) **United States Patent**
Tseng et al.

(10) **Patent No.: US 8,086,938 B2**
(45) **Date of Patent: *Dec. 27, 2011**

(54) **METHOD FOR PROCESSING NOISE INTERFERENCE**

(75) Inventors: **Pao-Ching Tseng**, Hsin Chu (TW); **Shu-Fang Tsai**, Hsin Chu (TW); **Chuan Liu**, Jen Te Hsiang (TW)

(73) Assignee: **Mediatek Inc.**, Hsin Chu (TW)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 1039 days.

This patent is subject to a terminal disclaimer.

(21) Appl. No.: **11/960,555**

(22) Filed: **Dec. 19, 2007**

(65) **Prior Publication Data**

US 2008/0104480 A1 May 1, 2008

Related U.S. Application Data

(63) Continuation of application No. 11/039,948, filed on Jan. 24, 2005, now Pat. No. 7,343,545.

(30) **Foreign Application Priority Data**

Apr. 22, 2004 (TW) ..... 93111174 A

(51) **Int. Cl.**
G11C 29/00 (2006.01)
H03M 13/00 (2006.01)

(52) **U.S. Cl.** **714/769**

(58) **Field of Classification Search** 714/769
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

6,147,963 A 11/2000 Walker et al.
6,763,477 B1 7/2004 McGee
6,853,573 B2 2/2005 Kim et al.
6,854,045 B2 * 2/2005 Ooi et al. 711/202
6,948,036 B2 * 9/2005 Grieff et al. 711/151
6,961,787 B2 * 11/2005 Ooi 710/19
6,961,813 B2 * 11/2005 Grieff et al. 711/112
7,047,335 B2 * 5/2006 Schauer et al. 710/105
7,178,054 B2 * 2/2007 Seto et al. 714/5.11
7,257,163 B2 8/2007 Hwang et al.
7,284,082 B2 10/2007 Greenberger
7,313,751 B2 12/2007 Kang et al.
7,339,500 B2 3/2008 Noda
7,343,545 B2 * 3/2008 Tseng et al. 714/769
7,360,010 B2 4/2008 Ghaffari et al.
7,360,119 B1 4/2008 Stenfort et al.
7,384,082 B2 6/2008 Blake
7,406,652 B2 7/2008 Tseng et al.
7,424,628 B2 9/2008 Matsumoto et al.
7,477,649 B2 * 1/2009 Paulson et al. 370/412
7,496,691 B2 * 2/2009 Ayyavu et al. 710/5
7,523,235 B2 * 4/2009 Nemazie et al. 710/74
7,523,236 B1 * 4/2009 Nemazie et al. 710/74
7,539,797 B2 * 5/2009 Nemazie et al. 710/74
7,664,042 B2 * 2/2010 Utsunomiya et al. 370/242

(Continued)

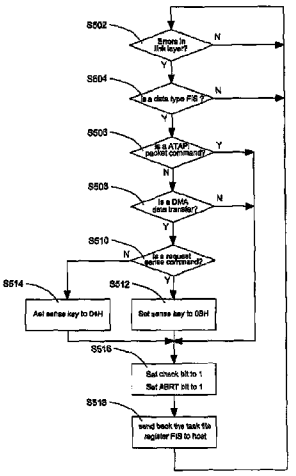
Primary Examiner — Joseph D Torres

(74) *Attorney, Agent, or Firm* — Muncy, Geissler, Olds & Lowe, PLLC

(57) **ABSTRACT**

A method for processing noise interference in a serial AT Attachment (SATA) interface. The method includes the steps of detecting whether there is an error in CRC (Cyclic Redundancy Check) checksum or whether an R_ERR primitive (reception error primitive) is received, detecting whether a FIS (Frame Information Structure) is a data type if there is any error and returning back to error state detecting step if there is no any error, detecting whether the FIS is a ATAPI packet command CDB (Command Descriptor Block) when the FIS is the data format, and writing a special tag to the CDB and returning back to the error detecting step.

9 Claims, 6 Drawing Sheets



U.S. PATENT DOCUMENTS

2003/0236952	A1*	12/2003	Grieff et al.	711/151
2003/0236953	A1*	12/2003	Grieff et al.	711/151
2004/0019718	A1*	1/2004	Schauer et al.	710/105
2004/0252716	A1	12/2004	Nemazie	
2005/0144490	A1*	6/2005	Igari	713/300
2005/0188245	A1*	8/2005	Seto et al.	714/5
2005/0240855	A1	10/2005	Tseng et al.	
2006/0095594	A1*	5/2006	Chen et al.	710/5

* cited by examiner

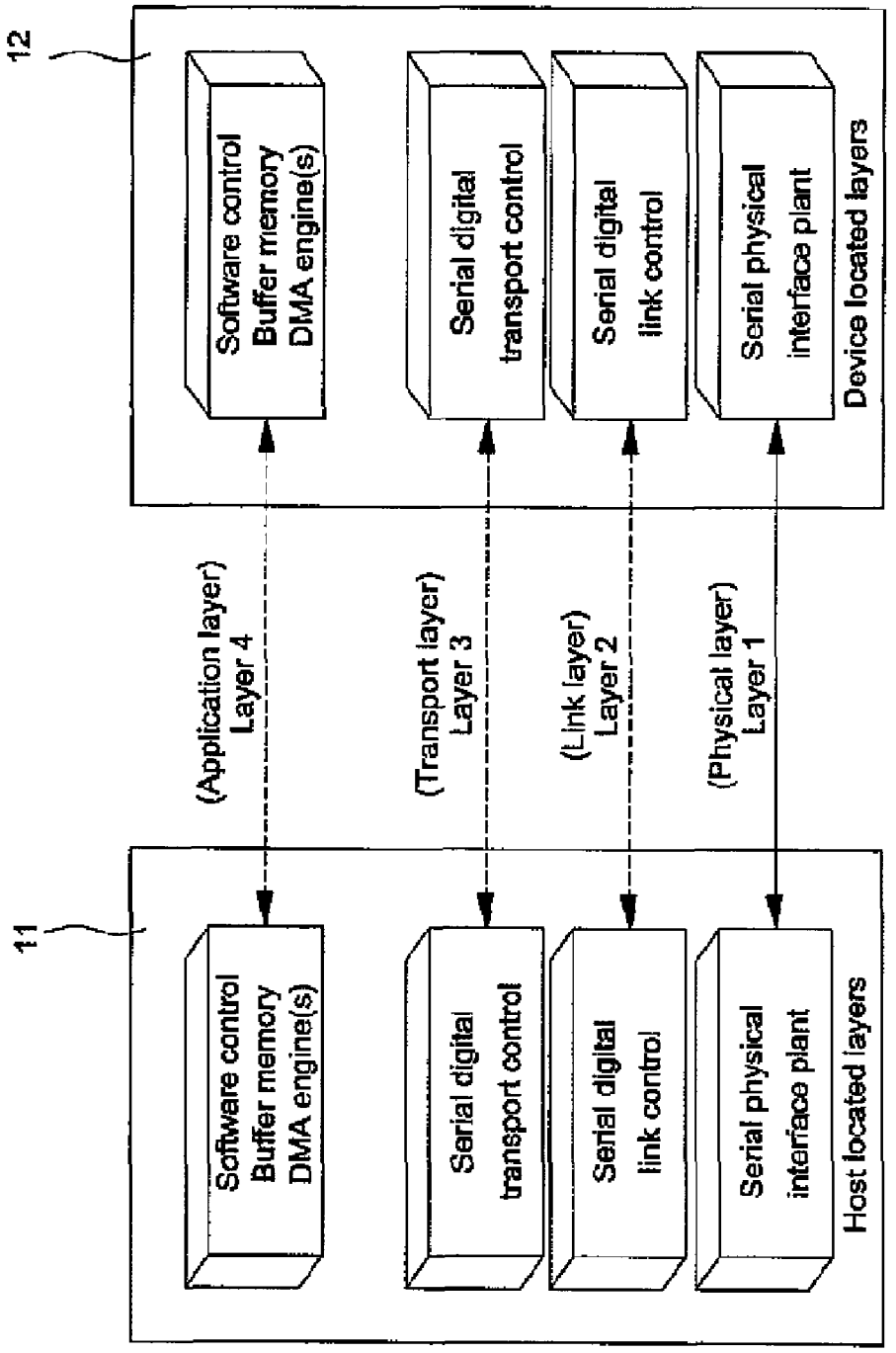


FIG. 1 (Related Art)

sending device	receiving device	description
SYNC	SYNC	
SYNC	SYNC	
X_RDY	SYNC	Ready for sending data in sending device
X_RDY	SYNC	
X_RDY	R_RDY	Ready for receiving data in receiving device
X_RDY	R_RDY	
SOF	R_RDY	Packet start
Payload 0	R_RDY	
Payload 1	R_IP	
...	...	
Payload n	R_IP	
CRC	R_IP	
EOF	R_IP	Packet end
WTRM	R_IP	
WTRM	R_IP	
WTRM	R_OK	Receive-OK
WTRM	R_OK	
SYNC	R_OK	
SYNC	R_OK	
SYNC	SYNC	

FIG. 2 (Related Art)

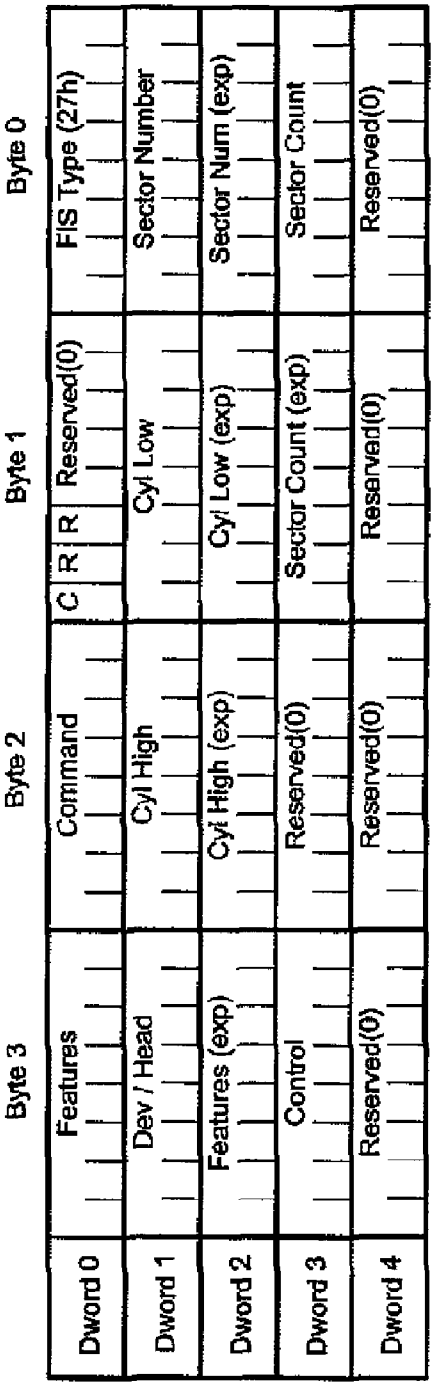


FIG. 3 (Related Art)

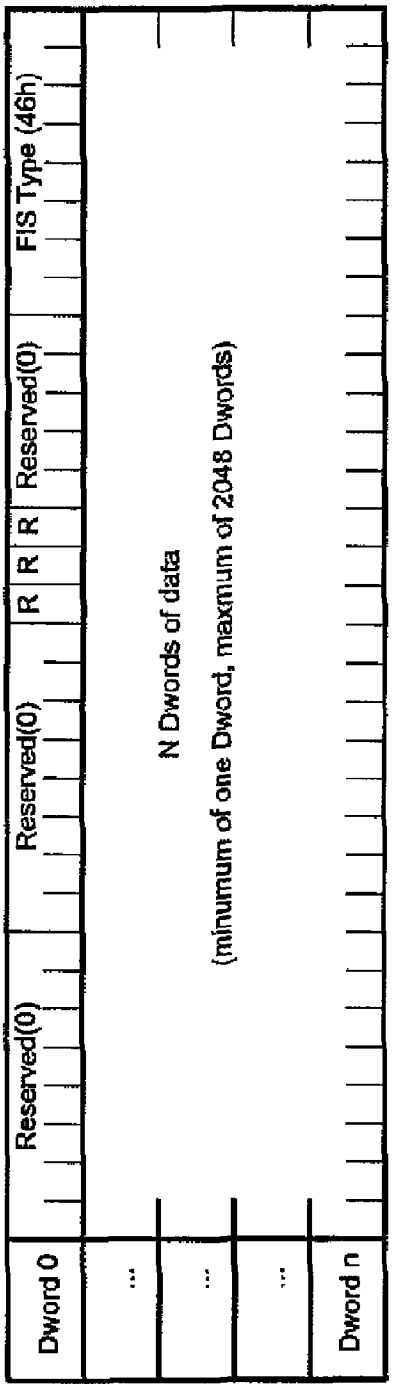


FIG. 4 (Related Art)

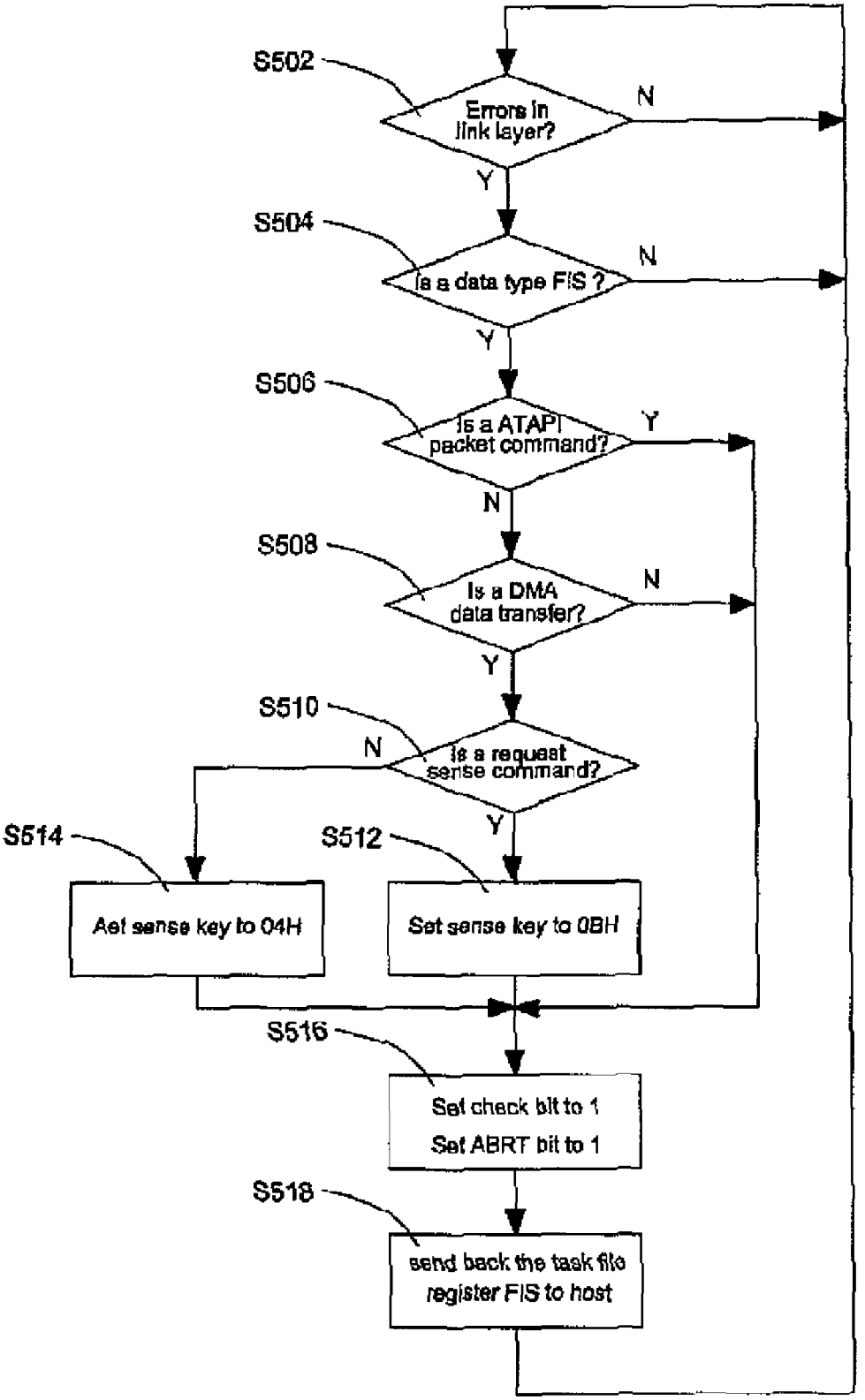

S502
Errors in link layer?
N
Y
S504
Is a data type FIS ?
N
Y
S506
Is a ATAPI packet command?
Y
N
S508
Is a DMA data transfer?
N
Y
S510
Is a request sense command?
N
Y
S514
Set sense key to 04H
S512
Set sense key to 0BH
S516
Set check bit to 1
Set ABRT bit to 1
S518
send back the task file register FIS to host

FIG. 5

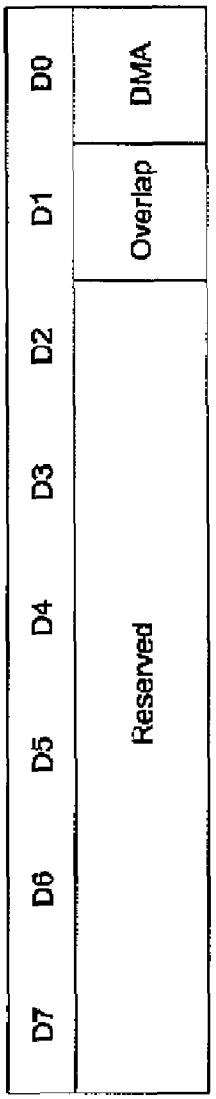


FIG. 6

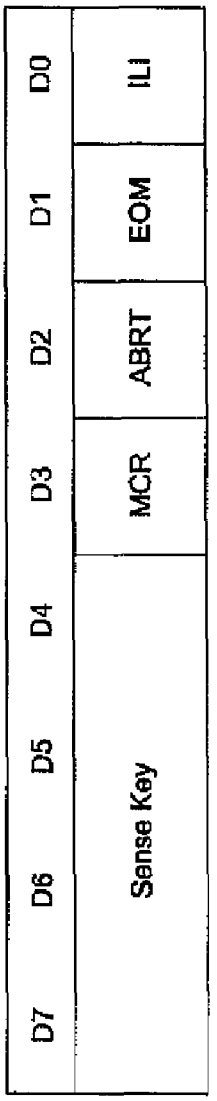


FIG. 7

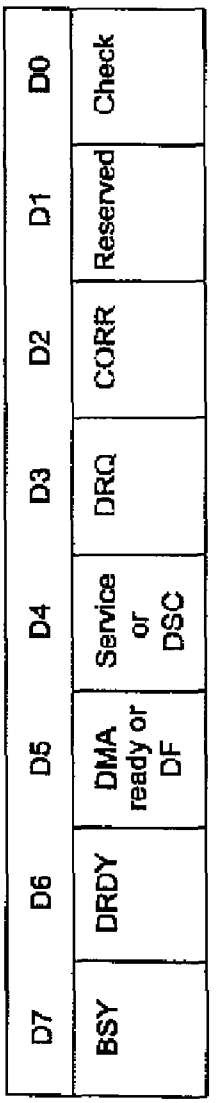


FIG. 8

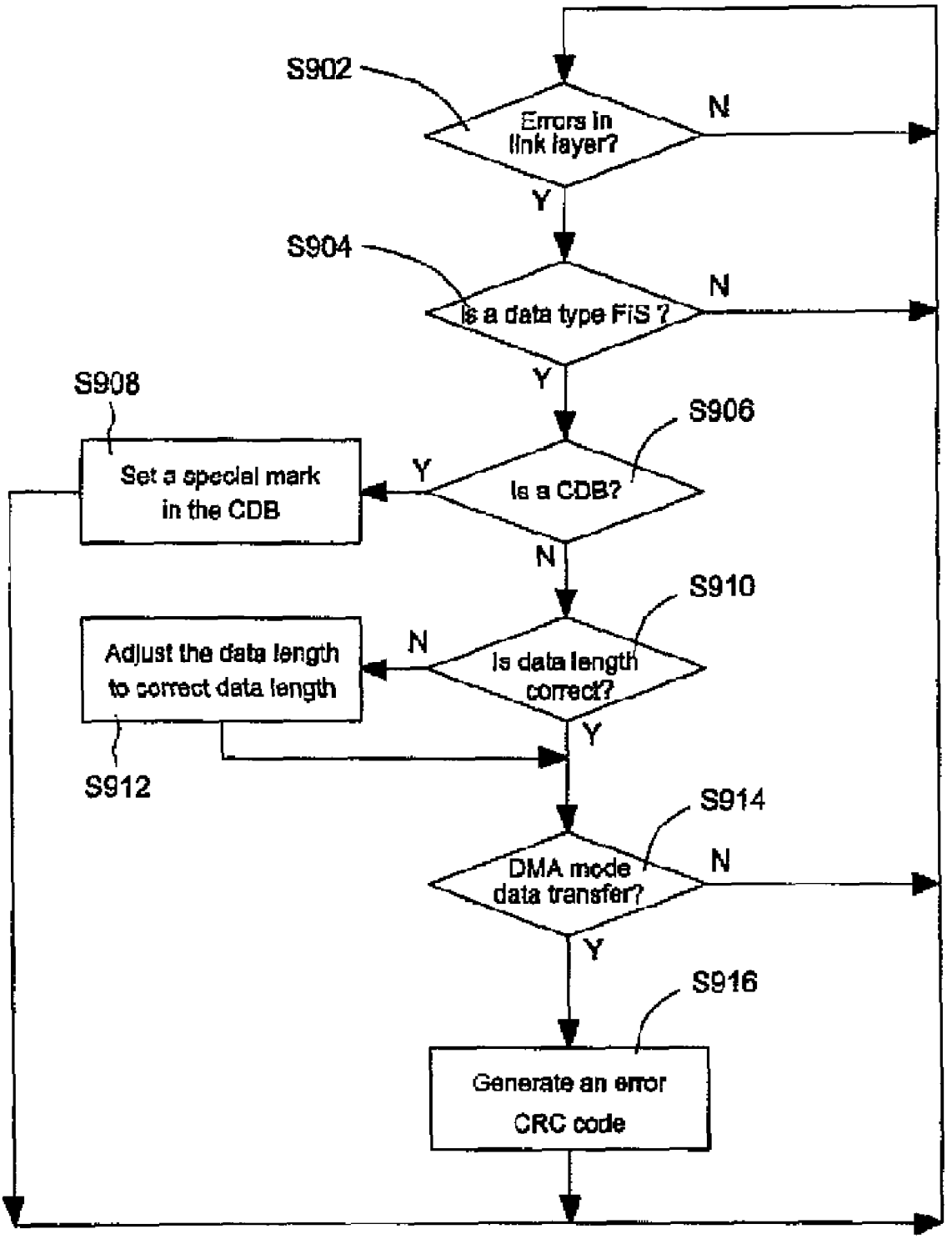

S902
Errors in link layer?
N
Y
S904
Is a data type FIS ?
N
Y
S906
Is a CDB?
Y
S908
Set a special mark in the CDB
N
S910
Is data length correct?
N
Adjust the data length to correct data length
S912
Y
S914
DMA mode data transfer?
N
Y
S916
Generate an error CRC code

FIG. 9

METHOD FOR PROCESSING NOISE INTERFERENCE

This application is a Continuation of application Ser. No. 11/039,948, filed on Jan. 24, 2005 now U.S. Pat. No. 7,343,545, and for which priority is claimed under 35 U.S.C. §120; and this application claims priority of Application No. 093111174 filed in Taiwan, R.O.C. on Apr. 22, 2004 under 35 U.S.C. §119, the entire contents of which are hereby incorporated by reference.

BACKGROUND OF THE INVENTION

1. Field of the Invention

The invention relates to a method for processing noise interference, and more particularly to a method for processing noise interference according to an error feedback mechanism of an ATA/ATAPI (AT Attachment with Packet Interface).

2. Description of the Related Art

The serial ATA (Serial Advanced Technology Attachment, hereinafter referred to as SATA) is an interface specification commonly promoted by the companies of APT, Dell, IBM, Intel, Maxtor, Seagate, etc. The SATA specification is applied to the transmission interface of a hard disk drive or an optical disk drive to replace parallel ATA/ATAPI interface that has been used for a long time. The SATA interface specification specifies two pairs of differential signal lines to replace the original 40 or 80 signal lines connected in parallel. Serializing the original data can reduce the size and voltage and increase the speed. The specification also introduces some new functions, such as flow control and error resending, to control the data stream in a simple way.

FIG. **1** is a schematic illustration showing communication layers in the SATA specification. As shown in FIG. **1**, the SATA interface connects a host **11** to a device **12**. The device **12** may be an optical storage device or a hard disk drive, or other devices with the SATA interface. The communication layers in the SATA specification include four layers, which are respectively a first layer (physical layer), a second layer (link layer), a third layer (transport layer) and a fourth layer (application layer). The physical layer is responsible for converting digital and analog signals. That is, the physical layer receives and converts a digital signal sent from the link layer into an analog signal and sends the analog signal to the other end. The physical layer also receives and converts the analog signal, which comes from the other end, into a digital signal and outputs the digital signal to the link layer. The link layer encodes and decodes the digital data. That is, the link layer encodes the data coming from the transport layer and outputs the encoded data to the physical layer. On the other hand, the link layer decodes the data coming from the physical layer and outputs the decoded data to the transport layer. The transport layer constructs and deconstructs the FIS (Frame Information Structure). The detailed definition of the FIS can be found in the SATA specification. The application layer is in charge of buffer memory and DMA engine(s).

During the serializing process, the sending device converts the parallel data (e.g., data in bytes or words) into a serial bit data stream. In addition to the typical data, the SATA specification defines some data control codes with four bytes, which are referred to as primitives, for controlling the sending and power management of the sending device and the receiving device. For example, a X_RDY primitive (transmission data ready primitive) represents that the sending device is ready to send data, and a R_RDY primitive (receiver ready primitive) represents that the receiving device is ready to receive data.

FIG. **2** is a schematic illustration showing a packet sent through the SATA interface. Two devices communicate with each other to send the packet according to the X_RDY primitive (transmission data ready primitive) and the R_RDY primitive (receiver ready primitive). Then, the sending side sends a packet content, which is packed by a SOF primitive (start of frame) and an EOF primitive (End of frame). After the packet content is sent completely, the sending side sends a WTRM primitive (wait for frame termination primitive). If there is no any error about the CRC (Cyclic Redundancy Check) check in the link layer, the receiving side responds with a R_OK primitive (reception with no error primitive) after it receives the WTRM primitive. If there is an error about the CRC check, the receiving side responds with a R_ERR primitive (reception error).

The FIS is for transferring the task file register and data. Two examples will be described in the following. FIG. **3** is a schematic illustration showing a task file register FIS from a host to a device. That is, a task file register written from the host to the device is shown in FIG. **3**. FIG. **4** is a schematic illustration showing a data FIS from the host to the device or from the device to the host. That is, the data sent from the host to the device or from the device to the host is shown in FIG. **4**.

In order to prevent the error caused by noises, the SATA specification specifies a resending mechanism. Almost all type of FISs (e.g., Register-Host to Device FIS or Set Device Bits-Device to Host FIS) are resent when a R_ERR primitive was received so as to ensure that the receiving device can receive the correct control information. However, the data FIS does not have this protection mechanism. When the data error occurs, the device cannot be protected by the resending mechanism under the condition of noise interference because the data FIS is not resent. Thus, the error data may be received. In a more serious condition, some portions of primitives may be mistaken as data bits, thereby making the number of data sets to be sent to the device greater than it should be. In some cases, this error may halt the system.

In addition, in the device (e.g., an optical storage device) using the data FIS to send the ATA/ATAPI CDB (Command Descriptor Block), the CDB cannot be protected according to the resending mechanism because the data FIS is not resent. Thus, the device receives an abnormal control command, which causes a larger influence than in the case of sending the normal data using the data FIS.

SUMMARY OF THE INVENTION

It is therefore an object of the invention to provide a method for processing noise interference in order to prevent the system from being stopped when the data FIS is interfered by noises.

To achieve the above-mentioned object, the invention provides a method for processing noise interference. The method includes the steps of detecting error states by detecting whether a CRC (Cyclic Redundancy Check) code exists error or whether an error primitive is received, detecting whether a FIS (Frame Information Structure) is a data format if there is an error state and returning back to error state detecting step if the FIS is not a data format, detecting whether the FIS is a CDB (Command Descriptor Block) when the FIS is a data format, and modifying contents of the CDB, and writing a special mark to the CDB and returning back to the error state detecting step when the FIS is not a CDB.

The method of the invention further comprises the steps of: detecting whether a data length of the FIS is correct when the FIS is not the CDB; adjusting the data length by increasing or decreasing data of the FIS so as to make the data length of the

FIS correct when the data length of the FIS is incorrect; and generating an error CRC code and jumping back to the error state detecting step.

Further scope of the applicability of the present invention will become apparent from the detailed description given hereinafter. However, it should be understood that the detailed description and specific examples, while indicating preferred embodiments of the invention, are given by way of illustration only, since various changes and modifications within the spirit and scope of the invention will become apparent to those skilled in the art from this detailed description.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention will become more fully understood from the detailed description given hereinbelow and the accompanying drawings which are given by way of illustration only, and thus are not limitative of the present invention.

FIG. **1** is a schematic illustration showing communication layers in the SATA specification.

FIG. **2** is a schematic illustration showing a FIS sent through the SATA interface.

FIG. **3** is a schematic illustration showing a task file register FIS from a host to a device.

FIG. **4** is a schematic illustration showing a data FIS.

FIG. **5** is a flow chart showing a method of the invention for processing noise interference in an ATA/ATAPI device.

FIG. **6** is a schematic illustration showing definitions of each bit in an ATAPI Feature Register.

FIG. **7** is a schematic illustration showing definitions of each bit in an ATAPI Error Register.

FIG. **8** shows definitions of each bit in an ATAPI Status Register.

FIG. **9** is a flow chart showing a method of the invention for processing noise interference in a bridge solution.

DETAILED DESCRIPTION OF THE INVENTION

The invention will be described with reference to the accompanying drawings. The invention utilizes the ATA/ATAPI error feedback mechanism to overcome the problem of system halt caused by noise interference when the data FIS is being sent. A native serial ATA device and a bridge solution may be used according to the real applications. The native serial ATA device directs to a device having a serial ATA interface serving as the data transmission interface without the conversion from the parallel ATA to the serial ATA. The bridge solution can convert the parallel ATA interface to the serial ATA interface so as to provide a solution in a transitional stage.

FIG. **5** is a flow chart showing a method of the invention for processing noise interference in a native ATA/ATAPI device. The flow chart illustrates the implementation of advanced processing when there is a CRC error in the link layer or the link layer receives a R_ERR primitive. The method of the invention for processing the noise interference will be described with reference to FIG. **5**.

Step S**502** is to detect whether the link layer has detected an error state. The error state includes the CRC error, the receiving of the R_ERR primitive, or the receiving of the improper error primitive. If the link layer has not detected any error state, step S**502** is repeated. If the link layer has detected an error state, the process jumps to step S**504**.

Step S**504** is to judge whether the FIS is a data type FIS. As shown in FIGS. **3** and **4**, the first byte (Byte **0**) of each FIS is the FIS type. If the first byte is 46H, it represents that the FIS is a data type FIS, and the process jumps to step S**506**. If the first byte is not 46H, the process returns back to step S**502**.

Step S**506** is to judge whether the FIS is an ATAPI packet command. When the host wants to send the packet command to the optical storage device, the host firstly sends the task file register FIS with the command register value of A**0**H to the optical storage device. So, when the optical storage device receives the command register value of A**0**H, it knows that there are subsequent 12 bytes of ATAPI packet commands. The first byte is the operation code, and the subsequent 11 bytes are the supplement data. If the FIS is an ATAPI packet command, the process jumps to step S**516**. Otherwise the process jumps to step S**508**.

Step S**508** is to judge whether the FIS is a DMA (Direct Memory Access) mode data transfer. FIG. **6** is a schematic illustration showing definitions of each bit in an ATAPI feature register. As shown in FIG. **6**, bit **0**(D**0**) is the DMA mode. Thus, whether the DMA mode data transfer exists may be detected as long as the data of bit **0** of the feature register is asserted. The definitions and functions of other bits may be found in the ATAPI specification. If the FIS is not a DMA mode data transfer, the process jumps to step S**516**. If the FIS is a DMA mode data transfer, the process jumps to step S**510**.

Step S**510** is to judge whether there is a request sense command (packet command with operation code 03h). When the host wants to send the packet command to the optical storage device, the host will firstly send the task file register FIS with the command register value of A0H to the optical storage device. So, after the optical storage device receives the command register value of A0H, it knows that there are subsequent 12 bytes of ATAPI packet command. The first byte is the command mode, and the subsequent 11 bytes are the supplement data. When the first byte is 03H, it represents that the command is a request sense command. So, whether there is a request sense command can be detected by recognizing whether the first byte of the ATAPI packet command is 03H. If there is not a request sense command, the process jumps to step S**512**; or otherwise the process jumps to step S**514**.

Step S**512** is to set the sense key of the task file register FIS to 0BH and then the process jumps to step S**516**. FIG. **7** is a schematic illustration showing definitions of each bit in an ATAPI error register, wherein D**4** to D**7** are the sense keys. There is a conventional way for the parallel ATA device to deal with the noise interference during the DMA mode data transmission. When the CRC has errors, the device will set the sense key to 04H to inform the host. While an exception exists, when the executing command is the request sense command, the sense key is set to 0BH. The detail definition of the sense key may be found in the associated ATAPI specification.

Step S**514** is to set the sense key of the task file register FIS to 04H and then the process jumps to step S**516**.

Step S**516** is to set the check bit of the status register to 1 and set the ABRT bit of the error register to 1.FIG. **8** shows definitions of each bit in an ATAPI status register. As shown in FIG. **8**, the bit **0** of the status register is the CHECK bit. As shown in FIG. **7**, the bit **2** of the error register is the ABRT bit. When the check bit of the status register is 1, it indicates that an error occurred during execution of the previous command. The bits in the Error Register contain the Sense Key and Code. When the ABRT bit of the error register is 1, it indicates command aborted.

Step S**518** is to send back the task file register FIS from the device to the host, and then the process jumps back to step S**502**.

Therefore, the method of the invention utilizes the ATA/ATAPI error feedback mechanism to process the noise interference according to the above-mentioned steps. Because the ABRT bit of the error register is set to 1 when the data FIS encounters the noise interference, the device requests the other side to resent whole FIS so as to effectively eliminate the problem of system halt caused by the noise interference when the data FIS is being sent.

The method of FIG. **5** is used in the accessing device that directly receives the SATA interface signal. The invention additionally proposes a method of bridge solution to connect the data accessing device of the parallel ATA interface to the SATA interface, wherein the accessing device itself only receives the parallel ATA interface data. FIG. **9** is a flow chart showing a method of the invention for processing noise interference in a bridge solution. The bridge solution is disposed between the serial ATA interface and the parallel ATA interface of a data accessing device (e.g., an optical storage device). The flow chart is an advanced implementation when the link layer detects a CRC error or receives the receive-error primitive. The method of the invention for processing the noise interference in the bridge solution will be described with reference to FIG. **9**.

Step S**902** is to detect whether the link layer has any error. The error state includes the CRC error, the receiving of the R_ERR primitive, or the receiving of the improper error primitive. If the link layer has not detected any error, step S**902** is repeated. If the link layer has detected an error, the process jumps to step S**904**.

Step S**904** is to judge whether the FIS is a data type FIS. As shown in FIGS. **3** and **4**, the first byte of each FIS is used to indicate the FIS type. That is, if the first byte is 27H, it represents that the FIS is a task file register FIS, and the process jumps back to step S**902**; and if the first byte is 46H, it represents that the FIS is a data type FIS, and the process jumps to step S**906**.

Step S**906** is to judge whether there is a command descriptor block CDB. When the host wants to send the packet command to the optical storage device, the host will firstly send the task file register FIS with the command register value of A0H to the optical storage device. So, after the optical storage device receives the command register value of A0H, it knows there are subsequent 12 bytes of ATAPI packet command. The first byte is the command mode and the subsequent 11 bytes are the supplement data. If there is a CDB, the process jumps to step S**908**. If there is not a CDB, the process jumps to step S**910**.

Step S**908** is to set a special mark in the CDB and then the process jumps back to step S**902**. For example, FFH is written into the CDB.

Step S**910** is to detect whether the data length is correct. If the data length is correct, the process jumps to step S**914**. Otherwise, the process jumps to step S**912**.

Step S**912** is to adjust the data length to correct data length and then the process jumps to step S**914**. For example, if the data length is not enough, the insufficient data is added; and if the data length is too long, the redundant data is discarded.

Step S**914** is to judge whether there is a DMA mode data transfer As shown in FIG. **6**, the bit **0** (D**0**) is the DMA mode. Thus, whether the DMA mode data transfer exists may be judged by only checking the data of bit **0** of the feature register. The definitions and functions of other bits may be found in the ATAPI specification, If there is not a DMA mode data transfer, the process jumps to step S**902**. Otherwise the process jumps to step S**916**.

Step S**916** is to generate an error CRC code at the parallel ATA interface, and then the process jumps back to step S**902**.

Thus, the existing accessing device with the parallel ATA interface may be serially connected to the bridge solution, and the control method of FIG. **9** may be adopted. The accessing device with the parallel ATA interface can be connected to the SATA interface via the bridge solution and error information is added to the parallel ATA interface when the data FIS encounters the noise interference such that the device could request the data to be to resent. Thus, the problem of system halt caused by the noise interference when the data type FIS is being sent may be effectively solved. In summary, the method of the invention for processing noise interference in the bridge solution includes the following steps.

1. When the bridge solution receives the ATAPI CDB data packet (Data FIS) with noise interference, it sends the CDB with a special mark to the parallel ATA end, as shown in step S**908**, for example. This special mark is defined as abnormal ATAPI CDB type, so the device sends back the error state such that the computer resends this data packet (Data FIS).

2. When some primitives are mistaken as data bits because the data FIS encounters the noise interference, only the desired number of data sets to be sent is outputted to the device, as shown in steps S**910** and S**912**, for example.

3. When the data is sent in the DMA mode and the data FIS encounters the interference because the SATA signal line has noises, the erroneous CRC is outputted to the device at the parallel ATA end. Thus, the host is enabled to resend the original command and information according to the error state response of the device, as shown in steps S**914** and S**916**, for example.

While certain exemplary embodiments have been described and shown in the accompanying drawings, it is to be understood that such embodiments are merely illustrative of and not restrictive on the broad invention, and that this invention not be limited to the specific construction and arrangement shown and described, since various other modifications may occur to those ordinarily skilled in the art.

What is claimed is:

1. A method for processing noise interference in a data accessing device with a SATA (Serial Advanced Technology Attachment) interface, the method comprising:

an error detecting step for detecting whether there is a CRC (Cyclic Redundancy Check) error, whether an reception error primitive (R_ERR primitive) is received, whether an improper primitive is received, or whether a LINK layer error is detected, and repeating this step if there is no any error;

a type detecting step for detecting whether an FIS (Frame Information Structure) is a data type FIS;

a responding step for asserting the CHECK bit of the ATAPI Status Register when the FIS is data type; and

sending back the response.

2. The method according to claim **1**, wherein the responding step further comprising asserting the ABRT bit of the ATAPI Error Register.

3. The method according to claim **1**, wherein the method is used in an optical storage device connected to the SATA interface.

4. The method according to claim **1**, further comprising:

an ATAPI (AT Attachment with Packet Interface) packet command detecting step for detecting whether the FIS is an ATAPI packet command when the FIS is data type, and executing the responding step if the FIS is the ATAPI packet command; and

a DMA (Direct Memory Access) mode data transfer detecting step for detecting whether there is a DMA mode data transfer when the data FIS is not the ATAPI

packet command, and executing the responding step if there is not a DMA mode data transfer.

5. The method according to claim **4**, wherein the method is used in an optical storage device connected to the SATA interface.

6. The method according to claim **4**, further comprising:

a request sense command detecting step for detecting whether the present command is a request sense command when there is a DMA mode data transfer;

setting a sense key of the ATAPI Error Register to 11 and executing the responding step if the present command is the request sense command; and

setting the sense key of the ATAPI Error Register to 4 and executing the responding step if the present command is not the request sense command.

7. The method according to claim **6**, wherein the method is used in an optical storage device connected to the SATA interface.

8. The method according to claim **1**, wherein the error state detecting step is performed in a link layer.

9. The method according to claim **8**, wherein the method is used in an optical storage device connected to the SATA interface.

* * * * *

EXHIBIT B

US008407783B2

(12) **United States Patent**
Huang et al.

(10) **Patent No.: US 8,407,783 B2**
(45) **Date of Patent: Mar. 26, 2013**

(54) **COMPUTING SYSTEM PROVIDING NORMAL SECURITY AND HIGH SECURITY SERVICES**

(75) Inventors: **Jing-Kuang Huang**, Hsinchu (TW); **Chih-Pin Su**, Hsinchu (TW)

(73) Assignee: **Mediatek Inc.**, Hsin-Chu (TW)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 358 days.

(21) Appl. No.: **12/817,387**

(22) Filed: **Jun. 17, 2010**

(65) **Prior Publication Data**

US 2011/0314538 A1 Dec. 22, 2011

(51) **Int. Cl.**
G06F 21/00 (2006.01)

(52) **U.S. Cl.** **726/19**; 713/192; 726/28

(58) **Field of Classification Search** 726/19, 726/28; 713/192

See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

2006/0265734 A1* 11/2006 Chen et al. 726/2
2008/0052534 A1* 2/2008 Harada et al. 713/190
2008/0092145 A1 4/2008 Sun et al.
2009/0293132 A1* 11/2009 Henry et al. 726/27

OTHER PUBLICATIONS

Suh et al. "AEGIS: Architecture for Tamper-Evident and Tamper-Resistant Processing", 2004, Computation Structures Group Memo 474 CSAIL Massachusetts Institute of Technology.*

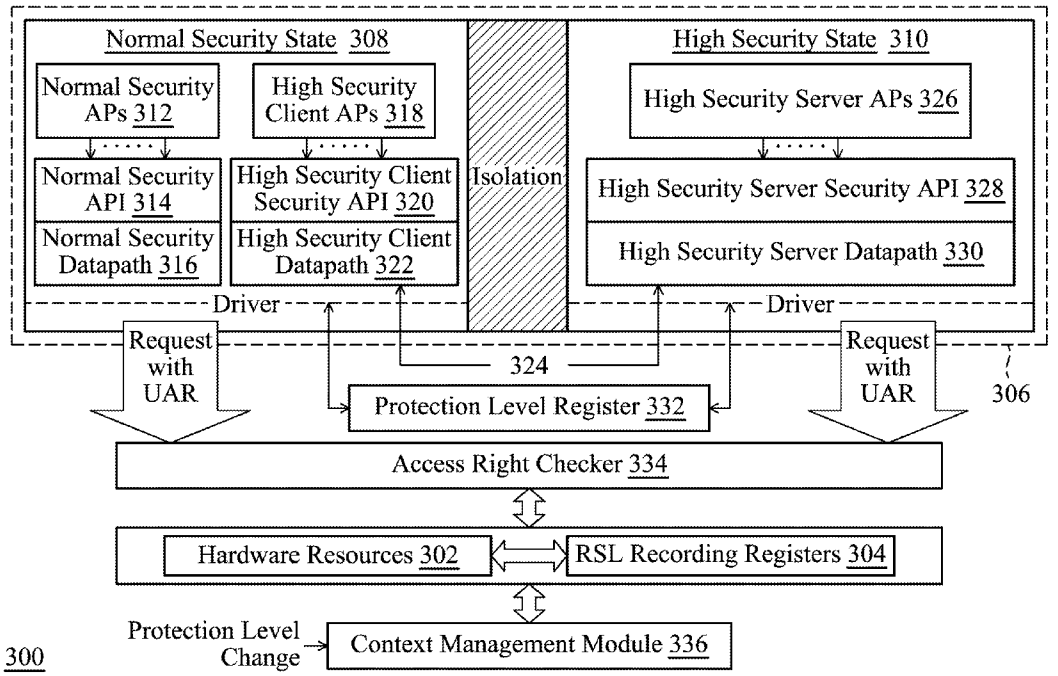
* cited by examiner

Primary Examiner — Kambiz Zand
Assistant Examiner — Michael Guirguis
(74) *Attorney, Agent, or Firm* — McClure, Qualey & Rodack, LLP

(57) **ABSTRACT**

A computing system and method providing normal security services and high security services are disclosed. The computing system includes hardware resources, a processor core and an access right checker. The hardware resources are grouped into resource security levels. The processor, switching between a normal security and a high security state, assigns a user access right to a request. In comparison with the normal security state, user access right assigned in the high security state further allows the request to use the hardware resources of a higher resource security level. According to the assigned user access right and the resource security levels of required hardware resources of the request, the access right checker determines whether the request has the authority to use the hardware resources, and thereby, the access right checker executes the request or responds the issued request with an exception.

17 Claims, 5 Drawing Sheets



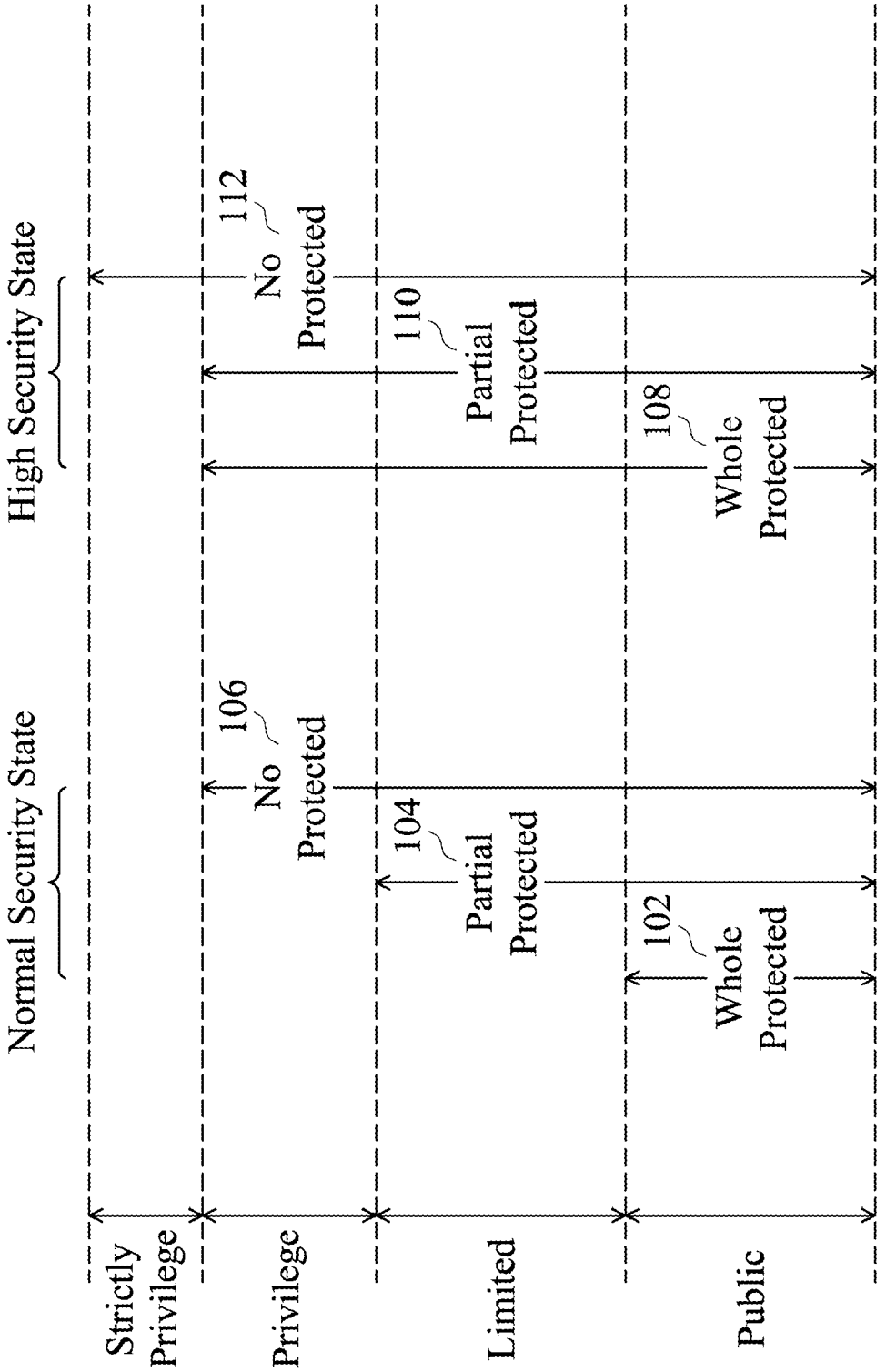

High Security State
Normal Security State
Strictly Privilege
Privilege
Limited
Public
No Protected
112
No Protected
106
Partial Protected
110
Partial Protected
104
Whole Protected
108
Whole Protected
102

FIG. 1

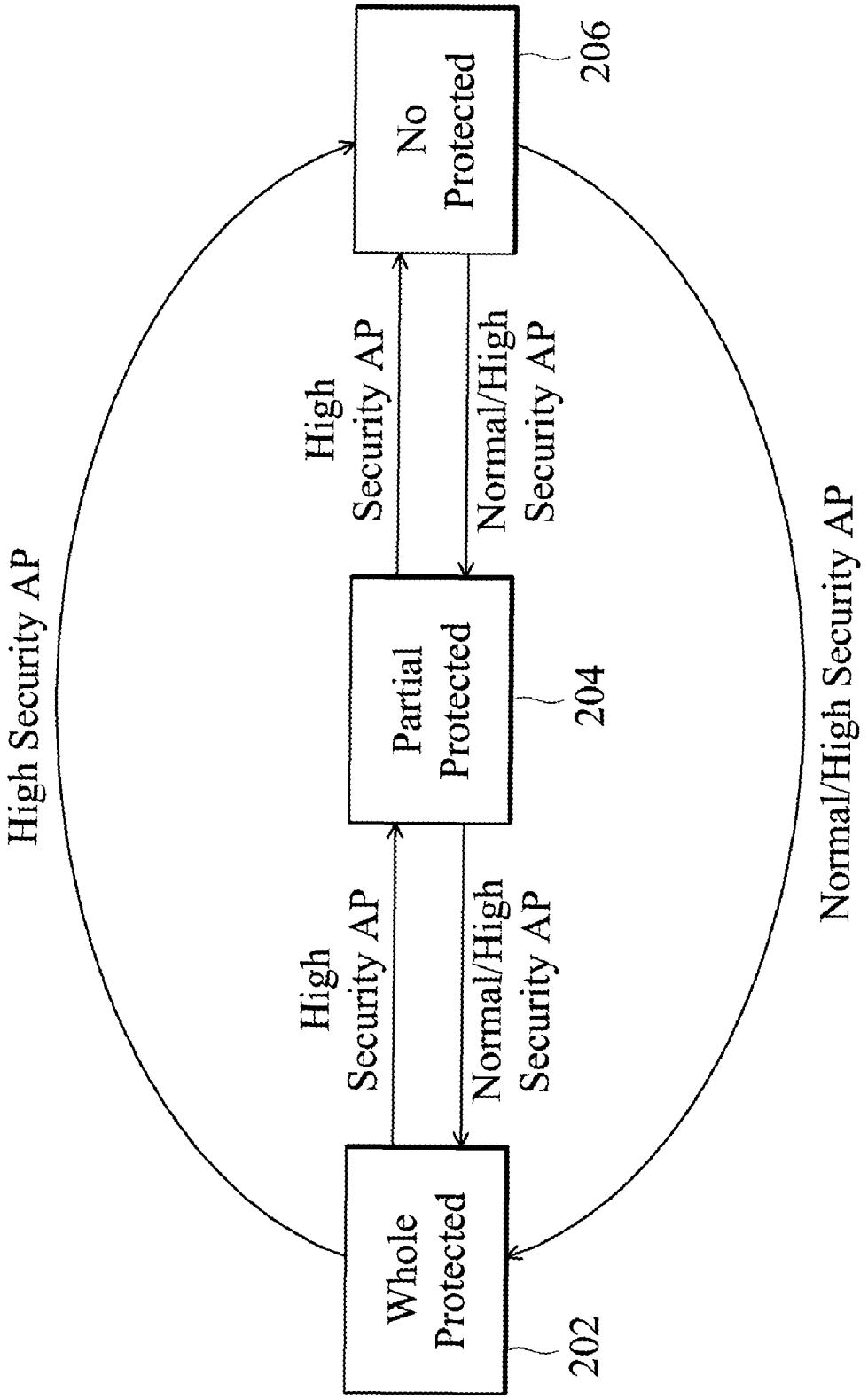

Whole Protected
202
High Security AP
Normal/High Security AP
Partial Protected
204
High Security AP
Normal/High Security AP
No Protected
206
High Security AP
Normal/High Security AP

FIG. 2

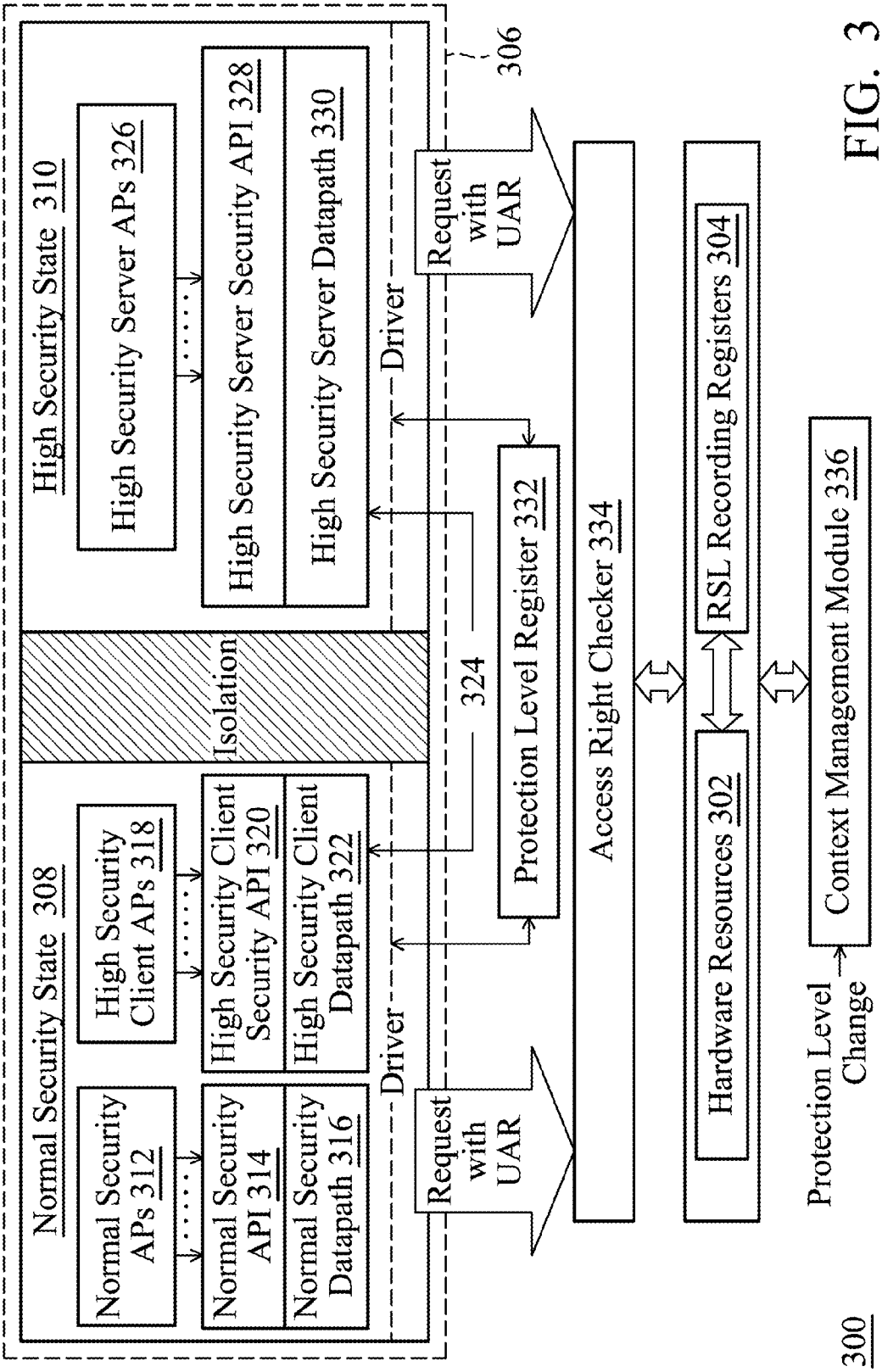

High Security State 310
High Security Server APs 326
High Security Server Security API 328
High Security Server Datapath 330
Driver
Isolation
Normal Security State 308
Normal Security APs 312
High Security Client APs 318
High Security Client Security API 320
High Security Client Datapath 322
Normal Security API 314
Normal Security Datapath 316
Driver
Request with UAR
Request with UAR
306
324
Protection Level Register 332
Access Right Checker 334
Hardware Resources 302
RSL Recording Registers 304
Context Management Module 336
Protection Level Change
300

FIG. 3

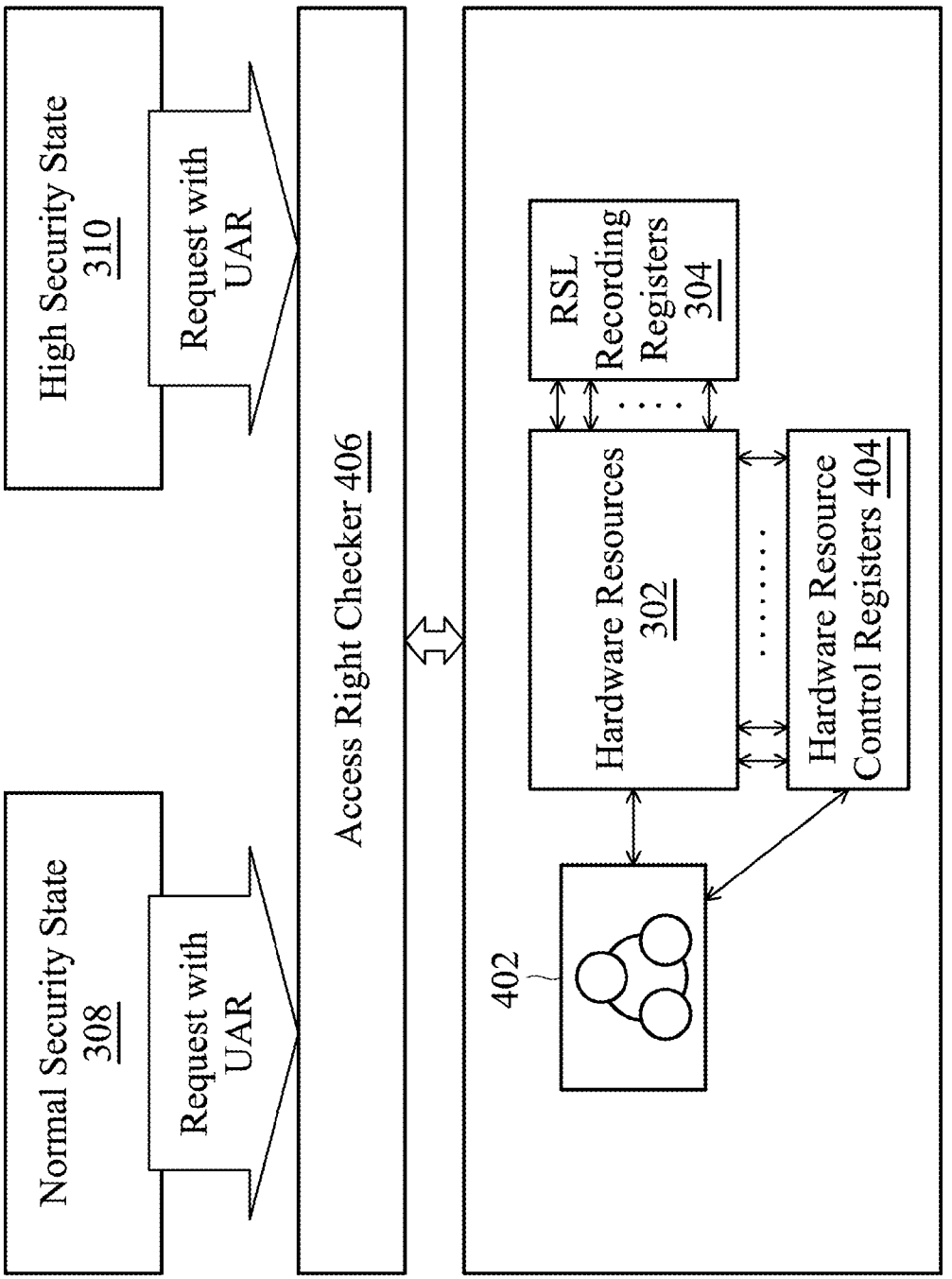

High Security State
310
Request with
UAR
Normal Security State
308
Request with
UAR
Access Right Checker 406
RSL
Recording
Registers
304
Hardware Resources
302
Hardware Resource
Control Registers 404
402

FIG. 4

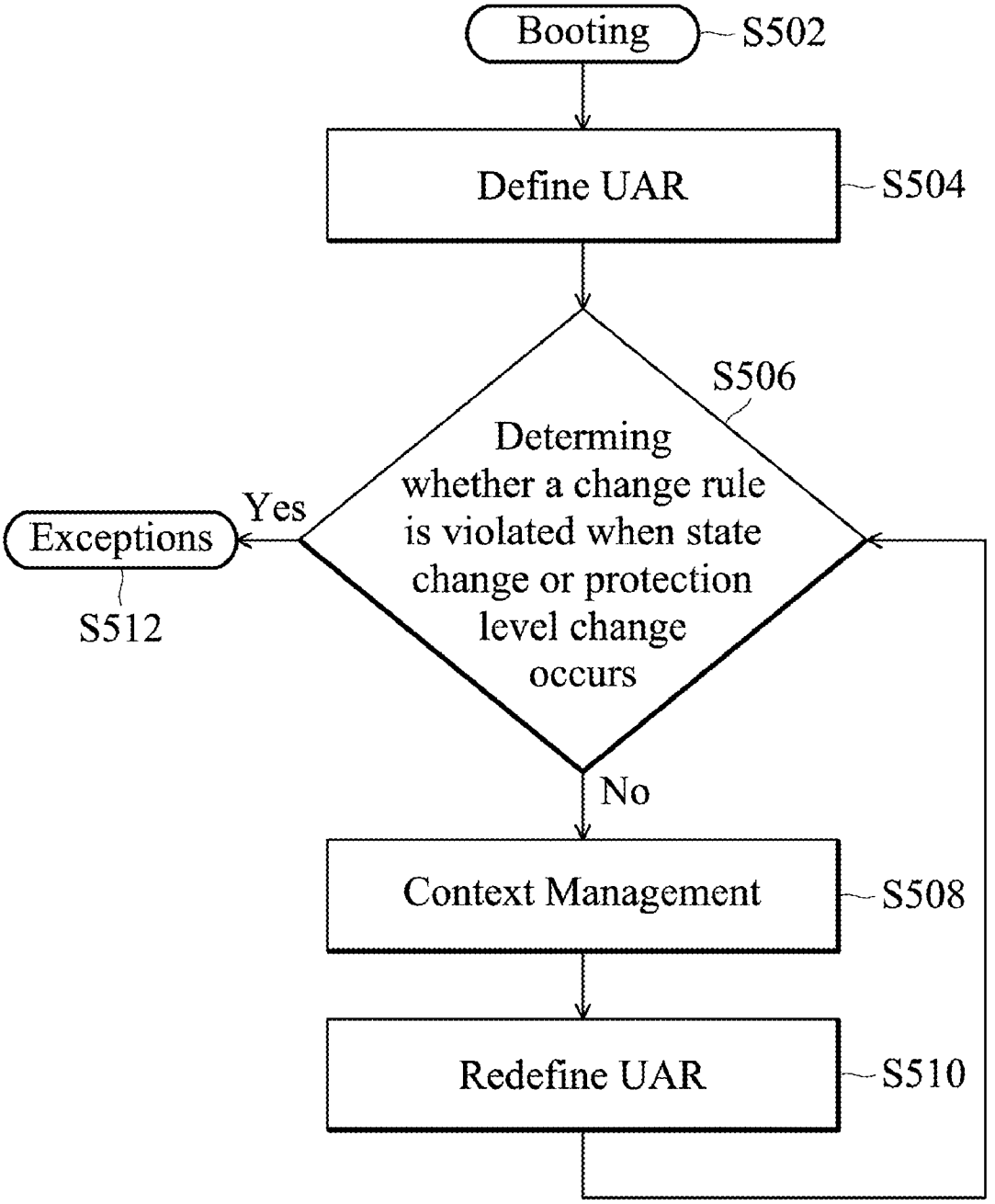

Booting
S502
Define UAR
S504
S506
Determing whether a change rule is violated when state change or protection level change occurs
Yes
Exceptions
S512
No
Context Management
S508
Redefine UAR
S510

FIG. 5

COMPUTING SYSTEM PROVIDING NORMAL SECURITY AND HIGH SECURITY SERVICES

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates to computing systems providing normal security services and high security services with efficient resource utilization.

2. Description of the Related Art

In many consumer electronics such as mobile phones, PDAs, digital cameras, digital media and music players, handheld game consoles, and computer peripherals (such as hard drives and routers), two isolated operating environments are required for maintaining system security.

Normal security services, such as making a phone call and playing java games may operate in a normal security environment. However, when high security services such as online credit card payments are requested, the normal security environment may not satisfy security requirements of e-banking services. Instead, a high security environment isolated from the normal security environment is required to handle such high security services in order to prevent information theft.

Trustzone hardware architecture, developed by ARM, provides normal security services and high security services using a single physical processor core. Because only one processor core is required in the Trustzone hardware architecture, silicon size, manufacturing costs, and power consumption of the Trustzone hardware are considerably lower when compared with solutions using two dedicated processor cores. To isolate sensitive data from the malicious software disguised in a normal security environment, the single processor core of the Trustzone technique switches between a normal security state and a high security state to provide a normal security environment and a high security environment in a time-sliced fashion, and the hardware resources are design dedicated to the normal security environment or the high security environment.

However, the design of dedicated hardware resources results in low resource utilization. Resources such as file system, memory, registers, and engine dedicated to the high security environment are seldom used as the probability of switching to the high security state is typically low. To increase the utilization of these dedicated resources, the processor may frequently switch between the two security states. For example, services involve sensitive data or require dedicated hardware resources in a high security environment will be performed in the high security environment. Frequent switching of security states occurs. Latency and power consumption caused by switching of security environments are considerable.

Thus, a way to simultaneously reduce switching of security environments and increase utilization of hardware resources is called for.

BRIEF SUMMARY OF THE INVENTION

A computing system and method providing normal security services and high security services are provided.

An exemplary embodiment of the computing system comprises hardware resources, a processor core and an access right checker. The hardware resources are grouped into a plurality of resource security levels. The processor, switching between a normal security state and a high security state, assigns a user access right to a request. In comparison with the

normal security state, user access right assigned in the high security state further allows the request to use the hardware resources of a higher resource security level. According to the assigned user access right and the resource security levels of required hardware resources of the request, the access right checker determines whether the request has the authority to use the hardware resources. When determining that the request has the authority to use the hardware resources, the access right checker allows the request to be executed. When determining that the request does not have the authority to use the hardware resources, the access right checker responds the request with an exception.

An exemplary embodiment of the disclosed method comprises the following steps. The method first provides hardware resources which are grouped into a plurality of resource security levels. Then, a user access right is assigned to a request. The processor core is switched between a normal security state and a high security state and, in comparison with the normal security state, user access right assigned in the high security state further allows the issued request to use the hardware resources of a higher resource security level. A determination is made based on the assigned user access right and the resource security levels of required hardware resources of the issued request, as to whether the request has the authority to use the hardware resources. When the request is determined to have the authority to use the hardware resources, the request is executed. When the request is determined to not have the authority to use the hardware resources, the request is responds with an exception.

A detailed description is given in the following embodiments with reference to the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention can be more fully understood by reading the subsequent detailed description and examples with references made to the accompanying drawings, wherein:

FIG. **1** illustrates an example depicting a concept of hardware resources sharing;

FIG. **2** illustrates a state machine showing a change rule of the protection level;

FIG. **3** depicts an architecture of the disclosed computing system;

FIG. **4** introduces an embodiment for implementing the architecture of the disclosed computing system; and

FIG. **5** depicts the flowchart of a UAR update procedure.

DETAILED DESCRIPTION OF THE INVENTION

The following description is of the best-contemplated mode of carrying out the invention. This description is made for the purpose of illustrating the general principles of the invention and should not be taken in a limiting sense. The scope of the invention is best determined by reference to the appended claims.

FIG. **1** illustrates an example depicting a concept of hardware resources sharing.

First, a definition of "Resource Security Level" (abbreviated to RSL) is introduced. In the computing system disclosed in the invention, the hardware resources, including the memory, registers, and hardware datapaths and so on, are divided into a plurality of groups according to resource security levels thereof. As shown in FIG. **1**, four resource security levels—"Strictly Privilege", "Privilege", "Limited" and "Public" (from highest security level to lowest security level)

are used to classify the hardware resources. Each group contains hardware resources of a corresponding security level.

FIG. **1** further introduces a concept of "User Access Right" (abbreviated to UAR). When a processor core, switching between normal and high security states to build normal and high security environments, of the disclosed computing system issues a request, user access right is assigned to the request to show the available hardware resources of the request. The scope of the assigned user access rights is dependent on the security state of the processor core, or may further depend on a protection level of the computing system. Referring to FIG. **1**, scopes **102**, **104** . . . **112** show several user access right options when the processor core is switching between the normal security state and the high security state and the protection level is switching between the "Whole Protected", "Partial Protected" and "No Protected" levels.

For example, when the computing system is in the "Whole Protected" level and the processor core is in the normal security state, user access right assigned to each request issued from software codes covers the scope **102**. Thus, only the hardware resources with the "Public" resource security level are available to the issued request. If the processor core is switched to the high security state and the protection level of the computing system is not changed (maintained at the "Whole Protected" level), user access right assigned to the issued request covers the scope **108** rather than the scope **102**, to allow the issued request to further gain access to the hardware resources of a higher resource security level (scope **108** includes any available hardware resources belonged to the "Public" resource security level as well as the "Limited" and "Privilege" resource security levels).

Comparing the scopes **102** and **108** defining the user access right, the hardware resources of the "Public" resource security level are shared between the normal security environment and the high security environment while the hardware resources of "Limited" and "Privilege" resource security levels are limited to the high security environment. Thus, sensitive data, contained in the "Limited" and "Privilege" hardware resources, are protected from malicious software of the normal security environment, and the "Public" hardware resources containing no sensitive data are efficiently shared by the normal and high security environments. The disclosed hardware resource sharing embodiments increase utilization of hardware resources.

The concept introduced in FIG. **1** also considerably reduces the redundant switching of security environments. For example, when required hardware resources of a request all belong to the "Public" resource security level, no matter what security state the processor core is in, no switching of security environments is required because the required hardware resources are available in both the normal security and high security environments. Thus, the frequency for switching of security environments can be reduced, and time latency and power consumption caused by frequent switching of security environments are eliminated.

This paragraph discusses the protection level introduced in FIG. **1**. A manufacturer may use a multi-purpose product to produce services of different security levels (for example, using the same hardware architecture to produce gaming, mobile phone call, or e-commerce services of different security levels). The introduced protection level is configured according to the security level requirement of the product, and it also affects the scope of user access right. In FIG. **1**, there are three options for the protection level setting, from the highest security level to the lowest security level, they are "Whole Protected", "Partial Protected" and "No Protected" levels. The greater the protection level is, the less the hardware resources can be accessed. For example, when the processor core is in the normal security state, the UAR scope **106** for the "No Protected" level is greater than UAR scope **104** for the "Partial Protected" level, and the UAR scope **104** for the "Partial Protected" level is greater than the UAR scope **102** of the "Whole Protected" level. Note that different protection levels may have the same scopes of UAR in some specific cases. As shown in FIG. **1**, when the processor core is in the high security state, the UAR scope **112** for the "No Protected" level is greater than the UAR scope **110** for the "Partial Protected" level while the UAR scope **110** for the "Partial Protected" level equals to the UAR scope **108** of the "Whole Protected" level.

Note that classification of the resource security levels and the protection levels and determination of the UAR scopes shown in FIG. **1** are not intended to limit the scope of the invention. The amount of the resource security levels and protection levels and the options for UAR may be different from those shown in FIG. **1**.

In an embodiment, the protection level of a computing system is extremely sensitive and should be stored in a register (named protection level register) of the "Strictly Privilege" resource security level. To avoid falsification, the setting of the protection level register may be limited during a security booting procedure of the computing system while the processor core is in the high security state. In other embodiments, the value stored in the protection level register may be changed in the run-time. FIG. **2** shows a change rule of the protection level. The disclosed protection level register is allowed to be changed from a higher security level to a lower security level only when the processor core is in the high security state, but is allowed to be changed from a lower security level to a higher security no matter what security state the processor core is in. Referring to FIG. **2**, to downgrade the protection level from the "Whole Protected" level **202** to the "Partial Protected" **204** or to the "No Protected" level **206**, or from the "Partial Protected" level **204** to the "No Protected" level **206**, the processor core has to be in the high security state to execute corresponding high security applications (APs). Oppositely, to upgrade the protection level, from the "No Protected" level **206** to the "Partial Protected" level **204** or to the "Whole Protected" level **202**, or from the "Partial Protected" level **204** to the "Whole Protected" level **202**, the processor core is not limited to be in the normal or high security environment. The protection level upgrade can be achieved by either normal security APs or high security APs.

Note that when the protection level is changed, the initially isolated hardware resources may be released to be available to the issued request so that the sensitive data contained in the released hardware resources may be exposed. Thus, context management techniques are disclosed, which store and restore context of released or isolated hardware resources when the protection level stored in the protection level register is changed. The released hardware resources were initially forbidden from the issued request before the protection level was switched to a lower security level, and the isolated hardware resources were initially available to the issued request before the protection level was switched to a higher security level. Note that in some embodiments, the protection level of a product is configured at manufactory and cannot be changed in boot time or run-time.

FIG. **3** depicts the architecture of the disclosed computing system. As shown, the hardware resources **302** of the computing system **300** are equipped with resource security level (RSL) recording registers **304**. The RSLs (for example, the "Strictly Privilege", "Privilege", "Limited" and "Public" lev-

els defined in FIG. **1**) of the hardware resources **302** are recorded in the RSL recording registers **304**.

In the computing system **300**, one single physical processor core **306** is provided, to switch between a normal security state **308** and a high security state **310** to form two isolated environments for normal security services and high security services, respectively. Normal security applications (APs) **312**, normal security application programming interface (API) **314** and firmware of normal security datapath **316** are used to provide normal security services when the processor core **306** is in the normal security state **308**. In addition, high security client APs **318**, high security client API **320** and firmware of high security client datapath **322**, performed when the processor core **306** is in the normal security state **308**, operate as a security environment switching interface. The instructions **324** are provided for switching the security state of the processor core **306**. The high security applications (APs) **326**, high security application programming interface (API) **328** and firmware of high security datapath **330** are provided for high security services and are performed when the processor core **306** is in the high security state **310**. The aforementioned software and firmware design allows the single processor core **306** to switch between the normal security state **308** and the high security state **310** in a time-sliced fashion.

Furthermore, in the firmware design of the processor core **306**, no matter what security state the processor core **306** is in, each of the issued requests is assigned a scope of user access right (UAR). The assigned UAR may be dependent on the current security state of the processor core **302**, or, in some embodiments, the protection level of the computing system (stored in the protection level register **332**) may be further taken into consideration in determining the scope of the UAR. The determination of the UAR is disclosed in FIG. **1**.

The UAR assigned to the issued request indicates available components of the hardware resources **302**. The access right checker **334** receives the issued request and the assigned UAR, determines the required hardware resources of the issued request, and determines whether the received request has the authority to use the hardware resources **302** in accordance with the assigned UAR and the required hardware resources of the issued request. When the assigned UAR covers the resource security levels of all of the required hardware resources of the issued request, the access right checker **334** determines that the issued request has the authority to use the hardware resources **302** and allows the issued request to be executed. When the assigned UAR does not cover the resource security levels of all of the required hardware resources of the issued request, the access right checker **334** determines that the issued request has no authority to use the hardware resources **302** and responds to the issued request with an exception.

A context management module **336** is introduced for aforementioned context store and restore triggered by a change of the protection level.

FIG. **4** introduces another embodiment for implementing the architecture of the disclosed computing system. As shown, in addition to the RSL recording registers **304** of FIG. **3**, a finite state machine **402** and hardware resource control registers **404** are equipped with the hardware resources **302**. The finite state machine **402** models behavior of the hardware resources **302** and controls the hardware resources **302** via the hardware resources control registers **404**. In comparison with the access right checker **334** of FIG. **3**, the access right checker **406** further takes a current state of the finite state machine **402** into consideration when determining whether the issued request has the authority to use the hardware resources **302**. When the current security state (the normal security state **308** or the high security state **310**) of the processor core meets the current state of the finite state machine **402** and the assigned UAR covers the RALs of all of the required hardware resources of the issued request, the access right checker **406** determines that the issued request has the authority to use the hardware resources **302** and allows the issued request to be executed. Otherwise, when the current security state of the processor core does not meet the current state of the finite state machine **402** or the assigned UAR does not cover the RALs of all of the required hardware resources of the issued request, the access rights checker **406** determines that the issued request does not have the authority to use the hardware resources **302** and responds to the issued request with an exception.

FIG. **5** depicts the flowchart of a UAR update procedure. After booting (step S**502**) of the disclosed computing system, step S**504** may be performed to define the UAR according to the current security state of the processor core and the protection level of the computing system. When the security state of the processor core or the protection level of the computing system is changed, the determination step of step S**506** is performed to verify whether a change rule (for example, the protection level change rule depicted in FIG. **2**) is complied with. If the change rule is not violated, steps S**508** and S**510** may be performed. The context management of the step of step S**508** is designed to store and restore the context of the hardware resources released or isolated in the protection level change. In the step S**510**, the UAR is redefined in response to the change in the security state of the processor core or change in the protection level of the computing system. The determination step of step S**506** is continuously performed for the timely update of the UAR. When the determination step of step S**506** determines that the change rule has been violated, the flowchart is directed to exceptions of step S**512** to reject the illegal security state or protection level change.

While the invention has been described by way of example and in terms of the preferred embodiments, it is to be understood that the invention is not limited to the disclosed embodiments. To the contrary, it is intended to cover various modifications and similar arrangements (as would be apparent to those skilled in the art). Therefore, the scope of the appended claims should be accorded the broadest interpretation so as to encompass all such modifications and similar arrangements.

What is claimed is:

1. A computing system providing normal security services and high security services, comprising:

hardware resources, grouped into a plurality of resource security levels;

a processor core, switching between different security states including a normal security state for providing the normal security services and a high security state for providing the high security services, and assigning a user access right to a request in accordance with the security state of the processor core and a protection level of the computing system, wherein, in comparison with the normal security state, the user access right assigned in the high security state for a particular protection level of the computing system further allows the request to use hardware resources of a higher resource security level, and, for a particular security state of the processor core, the user access right assigned for a lower protection level covers the user access right assigned for a higher protection level; and

an access right checker, determining whether the request has the authority to use the hardware resources in accor-

dance with the assigned user access right and the resource security levels of required hardware resources of the request, wherein:

when determining that the request has the authority to use the hardware resources, the access right checker allows the request to be executed; and

when determining that the request does not have the authority to use the hardware resources, the access rights checker responds the request with an exception.

2. The computing system as claimed in claim **1**, further comprising a protection level register for determining a value of the protection level.

3. The computing system as claimed in claim **2**, wherein the processor core initializes the protection level register during a security booting procedure of the computing system, and the processor core is in the security state during the security booting procedure.

4. The computing system as claimed in claim **3**, further comprising:

a context management module, storing and restoring context of released or isolated hardware resources when the protection level stored in the protection level register is changed, wherein the released hardware resources were initially forbidden from the request before the protection level was switched to a lower security level, and the isolated hardware resources were initially available to the request before the protection level was switched to a higher security level.

5. The computing system as claimed in claim **4**, wherein the processor core in the high security state is allowed to switch the protection level to the lower security level.

6. The computing system as claimed in claim **1**, wherein the access right checker determines that the request has the authority to use the hardware resources when a scope of the assigned user access right covers the resource security levels of all of the required hardware resources, or, determines that the request does not have the authority to use the hardware resources when the scope of the assigned user access right does not cover the resource security levels of all of the required hardware resources.

7. The computing system as claimed in claim **1**, further comprising resource security level registers for determining the resource security level of each of the hardware resources.

8. The computing system as claimed in claim **7**, further comprising a finite state machine modeling behavior of the hardware resources and controlling the hardware resources via control registers of the hardware resources.

9. The computing system as claimed in claim **8**, wherein the access right checker further takes a current state of the finite state machine into consideration when determining whether the request has the authority to use the hardware resources.

10. The computing system as claimed in claim **9**, wherein the access right checker determines that the request has the authority to use the hardware resources when the processor core switching between the normal and high security states meets the current state of the finite state machine and a scope of the assigned user access right covers the resource security levels of all of the required hardware resources, or, determines that the request does not have the authority to use the hardware resources when the processor core does not meet the current state of the finite state machine or the scope of the assigned user access right does not cover the resource levels of all of the required hardware resources.

11. A method of providing normal security and high security services with reduced amount of hardware resources, comprising:

providing hardware resources which are grouped into a plurality of resource security levels;

providing a processor core switched between different security states, the different security states including a normal security state for providing the normal security services and a high security state for providing the high security services;

assigning a user access right to a request in accordance with the security state of the processor core and a protection level of a computing system including the hardware resources and the processor core, wherein, in comparison with the normal security state, the user access right assigned in the high security state for a particular protection level of the computing system further allows the request to use the hardware resources of a higher resource security level, and, for a particular security state of the processor core, the user access right assigned for a lower protection level covers the user access right assigned for a higher protection level;

determining whether the request has the authority to use the hardware resources in accordance with the assigned user access right and the resource security levels of required hardware resources of the request;

executing the request when the request is determined to have the authority to use the hardware resources; and

responding the request with an exception when the request is determined to not have the authority to use the hardware resources.

12. The method as claimed in claim **11**, wherein the protection level is initialized by the processor core during a security booting procedure of the computing system, and the processor core is in the security state during the security booting procedure.

13. The method as claimed in claim **12**, further storing and restoring context of released or isolated hardware resources when the protection level is changed, wherein the released hardware resources were initially forbidden from the request before the protection level was switched to a lower security level, and the isolated hardware resources were initially available to the request before the protection level was switched to a higher security level.

14. The method as claimed in claim **13**, wherein the protection level is allowed to be switched to the lower security level by the processor core when the processor core is in the high security state.

15. The method as claimed in claim **11**, wherein the request is determined to have the authority to use the hardware resources when a range of the assigned user access right covers the resource security levels of all of the required hardware resources, or, is determined to not have the authority to use the hardware resources when the scope of the assigned user access right does not cover the resource security levels of all of the required hardware resources.

16. The method as claimed in claim **15**, wherein the step of determining whether the request has the authority to use the hardware resources further takes a current state of the finite state machine into consideration.

17. The method as claimed in claim **16**, wherein the request is determined to have the authority to use the hardware resources when the processor core switching between the normal and high security states meets the current state of the finite state machine and a scope of the assigned user access right covers the resource security levels of all of the required hardware resources, or, is determined to not have the authority to use the hardware resources when the processor core does not meet the current state of the finite state machine or the scope of the assigned user access right does not cover the resource levels of all of the required hardware resources.

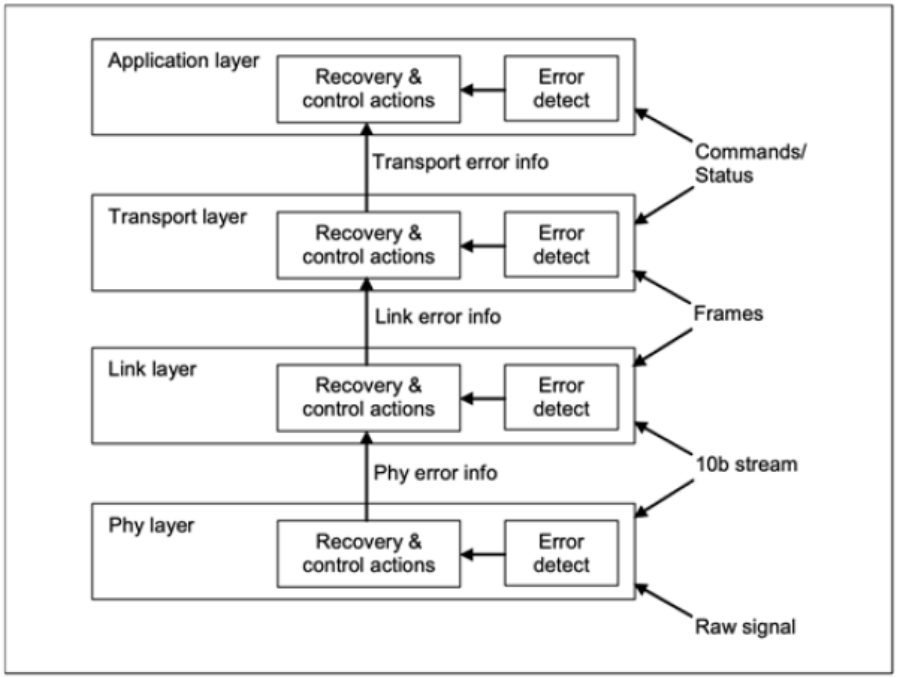
* * * * *

EXHIBIT C

U.S. Patent No. 8,086,938	NVIDIA DGX Systems
1. A method for processing noise interference in a data accessing device with a SATA (Serial Advanced Technology Attachment) interface, the method comprising:	The Lenovo ThinkPad 512GB 2.5" Solid State Drive is a representative example of the Accused Products having a SATA 3.0 interface. ThinkPad 512GB Solid State Drive - Overview and Service Parts Features and specifications ThinkPad 512GB Solid State Drive (4XB0E76671) The ThinkPad 512GB Solid State Drive option provides a speedy, light and reliable solution for your data storage and transfer. The SSD make tasks speedier and safer. Also, with 512GB high capacity, this ThinkPad 512GB Solid State Drive will be one of the best companions for your business. Technical information • 2.5" type, 7-mm height • SATA Revision 3.0 with 6Gbps interface

	Features and specifications
	
	ThinkPad 512GB Solid State Drive (4XB0E76671)
	
	The ThinkPad 512GB Solid State Drive option provides a speedy, light and reliable solution for your data storage and transfer. The SSD make tasks speedier and safer. Also, with 512GB high capacity, this ThinkPad 512GB Solid State Drive will be one of the best companions for your business.
	
	Technical information
	
	• 2.5" type, 7-mm height
	• SATA Revision 3.0 with 6Gbps interface
	• Power saving modes: HIPM, DIPM
	• Support NCQ: up to 32 depth
	• Synchronous Signal Recovery
	• Power specification
	  • Active: 3.4W (TYP)
	  • Idle: 50mW (TYP)
	
	Performance specifications
	
	• Interface: Serial ATA
	• Capacity: 512 GB
	• Sustained Sequential Read up to 540 MB/s
	• Sustained Sequential Write up to 330 MB/s
	• Shock: Operating and Non-operating 1500G/0.5msec.
	• Vibration: Operating: 7~800Hz, 3.08Grms, 30min/axis(X,Y,Z)
	• MTBF:1,500,000 hours
	
	Source: https://support.lenovo.com/us/en/solutions/acc100057

	 HIGH SPEED SERIALIZED AT ATTACHMENT Serial ATA International Organization Serial ATA International Organization: Serial ATA Revision 3.0 June 2, 2009 Gold Revision SATA-IO Board Members: Dell Computer Corporation Hewlett Packard Corporation Hitachi Global Storage Technologies, Inc. Intel Corporation Maxim Integrated Products Seagate Technology Western Digital Corporation Serial ATA Revision 3.0 - Gold Revision The Accused Products implement a method for processing noise interference in a data accessing device with a SATA interface.

	15 Error handling **15.1 Architecture** The layered architecture of Serial ATA extends to error handling as well. As indicated in Figure 250, each layer in the Serial ATA stack has as inputs error indication from the next lower layer (except for the Phy layer which has no lower layer associated with it), data from the next lower layer in the stack, and data from the next higher layer in the stack. Each layer has its local error detection capability to identify errors specific to that layer based on the received data from the lower and higher layers. Each layer performs local recovery and control actions and may forward error information to the next higher layer in the stack.  **Figure 250 – Error handling architecture** Source: SATA 3.0 specification
an error detecting step for detecting whether there is a	The Accused Products perform an error detecting step for detecting whether there is a Cyclic Redundancy Check (“CRC”) error.

CRC (Cyclic Redundancy Check) error, whether an reception error primitive (R_ERR primitive) is received, whether an improper primitive is received, or whether a LINK layer error is detected, and repeating this step if there is no any error;	**15.3.1 Error detection** There are two primary categories of errors that the Link layer detects internally: Invalid state transitions Data integrity errors Invalid state transition errors may arise from a number of sources and the Link layer responses to many such error conditions are defined in section 1. Data integrity errors generally arise from noise in the physical interconnect. **15.3.2 Error control actions** Errors detected by the Link layer during a transmission are generally handled by accumulating the errors until the end of the transmission and reflecting the reception error condition in the final R_ERR_P /R_OK_P handshake. Specific scenarios are listed in the following sections. On detection of a CRC error at the end of receiving a frame (at EOF_P), the Link layer shall notify the Transport layer that the received frame contains a CRC error. Furthermore, the Link layer shall issue the negative acknowledgement, R_ERR_P, as the frame status handshake, and shall return to the idle state. On detection of a disparity error or other 8b/10b coding violation during the receipt of a frame, the Link layer shall retain this error information, and at the close of the received frame the Link layer shall provide the negative acknowledgement, R_ERR_P, as the frame handshake, and shall notify the Transport layer of the error. The control actions are essentially the same for coding violations as for CRC errors. Source: SATA 3.0 specification The Accused Products detect whether a reception error primitive (R_ERR primitive) is received:

Primitive	Name	Description
PMREQ_P_P	Power management request to Partial.	This primitive is sent continuously until $PMACK_P$ or $PMNAK_P$ is received. When $PMACK_P$ is received, the current node (host or device) stops transmitting PMREQ_P_P and enters the Partial power management state.
PMREQ_S_P	Power management request to Slumber	This primitive is sent continuously until $PMACK_P$ or $PMNAK_P$ is received. When $PMACK_P$ is received, the current node (host or device) stops transmitting PMREQ_S_P and enters the Slumber power management state.
R_ERR_P	Reception error.	Current node (host or device) detected error in received payload.
R_IP_P	Reception in Progress.	Current node (host or device) is receiving payload.
R_OK_P	Reception with no error.	Current node (host or device) detected no error in received payload.
R_RDY_P	Receiver ready.	Current node (host or device) is ready to receive payload.
SOF_P	Start of frame.	Start of a frame. Payload and CRC follow until EOF_P.
$SYNC_P$	Synchronization	Synchronizing primitive.
$WTRM_P$	Wait for frame termination	After transmission of an EOF_P, the transmitter sends $WTRM_P$ while waiting for reception status from receiver.
X_RDY_P	Transmission data ready.	Current node (host or device) has payload ready for transmission.

9.6 Link Layer State Machine

9.6.1 Terms Used in Link Layer Transition Tables

1. LRESET: Link layer COMRESET or COMINIT signal
2. PHYRDYn: The negation of the PHYRDY signal.
3. PHYRDY: Phy status as defined in section 7.1.2.
4. DecErr: Bad decode of a 32 bit Dword transferred from Phy to Link
   - Invalid 10b pattern
   - Disparity error
   - Primitive with a control character in the first byte but not an allowed control character
   - Any control character in other than the first byte of the Dword
5. DatDword: A 32 bit pattern that is formed correctly, but does not have the primitive leading 10b pattern (K28.5 or K28.3).
6. COMWAKE: Signal from the OOB detector in the Phy indicating that the COMWAKE OOB signal is being detected.
7. AnyDword: A 32 bit pattern of any type - even one with DecErr received from Phy

Transition LT9:2: When the Link layer receives R_ERR_P from the Phy layer, the Link layer shall notify the Transport layer and make a transition to the L1: L_IDLE state.

Source: SATA 3.0 specification

The Accused Products detect whether an improper primitive is received.

If the transmitter closes a frame with EOF_P and $WTRM_P$, and receives neither a R_OK_P nor an R_ERR_P within a predetermined timeout, no Link layer recovery action shall be attempted. The higher-level layers should eventually time out, and reset the interface.

If the transmitter signals EOF_P, and a primitive other than $SYNC_P$, R_OK_P, or R_ERR_P is received, the Link layer shall persistently continue to await reception of a proper terminating primitive.

	Source: SATA III specification The Accused Products detect whether a Link layer error is detected. **15.3 Link layer error handling overview** **15.3.1 Error detection** There are two primary categories of errors that the Link layer detects internally: Invalid state transitions Data integrity errors Invalid state transition errors may arise from a number of sources and the Link layer responses to many such error conditions are defined in section 1. Data integrity errors generally arise from noise in the physical interconnect. **15.3.2 Error control actions** Errors detected by the Link layer during a transmission are generally handled by accumulating the errors until the end of the transmission and reflecting the reception error condition in the final R_ERR_P /R_OK_P handshake. Specific scenarios are listed in the following sections. **15.3.3 Error reporting** Link layer error conditions are reported to the Transport layer via a private interface between the Link layer and Transport layer. Additionally, Link layer errors are reported in the SError register as defined section 14.1.2.

	10.3.2 CRC Errors on Data FISes Following a Serial ATA CRC error on a Data FIS, if the device transmits a Device-to-Host FIS it shall set the ERR bit to one and both the BSY bit and DRQ bit cleared to zero in the Status field, and the ABRT bit set to one in the Error field. It is recommended for the device to also set the bit 7 (i.e. ICRC bit) to one in the Error field. See ATA8-ACS. There is no Device-to-Host FIS transmitted after a Serial ATA CRC error on the last Data FIS of a PIO-in command nor following a Serial ATA CRC error on the ATAPI command packet transfer. Thus, there is no mechanism for the device to indicate a Serial ATA CRC error to the host in either of these cases. The host should check the SError register to determine if a Link layer error has occurred in both of these cases. The Accused Products repeat the error detecting described above if there is no error. **L1: L_IDLE state:** This state is entered when a frame transmission has been completed by the Link layer. When in this state, the Link layer transmits $SYNC_P$ and waits for X_RDY_P from the Phy layer or a frame transmission request from the Transport layer.

15.3 Link layer error handling overview

15.3.1 Error detection

There are two primary categories of errors that the Link layer detects internally:

Invalid state transitions
Data integrity errors

Invalid state transition errors may arise from a number of sources and the Link layer responses to many such error conditions are defined in section 1. Data integrity errors generally arise from noise in the physical interconnect.

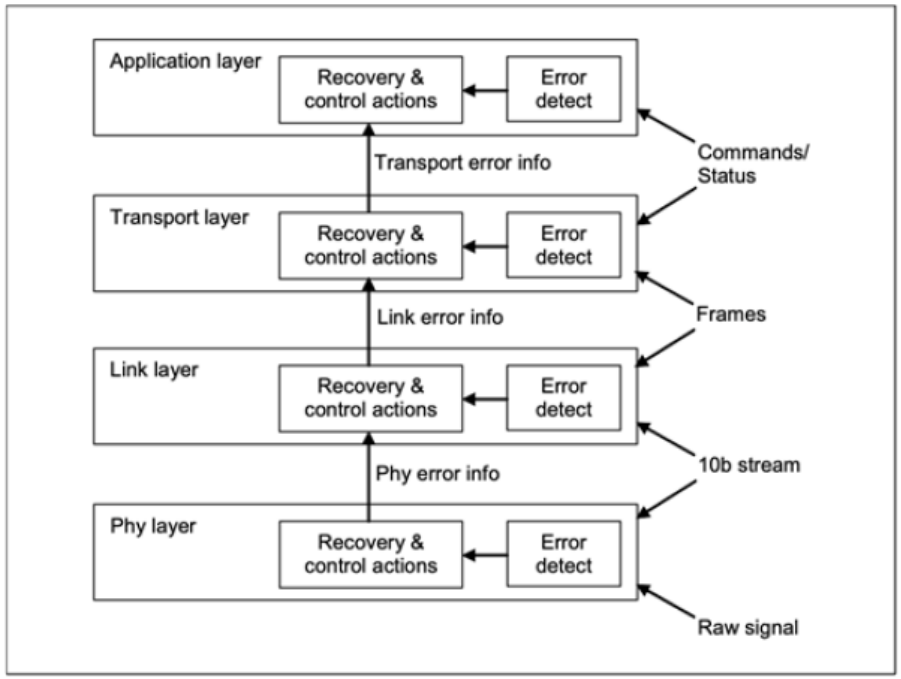
15.3.2 Error control actions

Errors detected by the Link layer during a transmission are generally handled by accumulating the errors until the end of the transmission and reflecting the reception error condition in the final R_ERR_P /R_OK_P handshake. Specific scenarios are listed in the following sections.

On detection of a CRC error at the end of receiving a frame (at EOF_P), the Link layer shall notify the Transport layer that the received frame contains a CRC error. Furthermore, the Link layer shall issue the negative acknowledgement, R_ERR_P, as the frame status handshake, and shall return to the idle state.

On detection of a disparity error or other 8b/10b coding violation during the receipt of a frame, the Link layer shall retain this error information, and at the close of the received frame the Link layer shall provide the negative acknowledgement, R_ERR_P, as the frame handshake, and shall notify the Transport layer of the error.

The control actions are essentially the same for coding violations as for CRC errors.

	15 Error handling **15.1 Architecture** The layered architecture of Serial ATA extends to error handling as well. As indicated in Figure 250, each layer in the Serial ATA stack has as inputs error indication from the next lower layer (except for the Phy layer which has no lower layer associated with it), data from the next lower layer in the stack, and data from the next higher layer in the stack. Each layer has its local error detection capability to identify errors specific to that layer based on the received data from the lower and higher layers. Each layer performs local recovery and control actions and may forward error information to the next higher layer in the stack.  **Figure 250 – Error handling architecture** Source: SATA 3.0 specification
a type detecting step for detecting whether an FIS	The Accused Products perform a type detecting step for detecting whether a Frame Information Structure (“FIS”) is a data type FIS.

(Frame Information Structure) is a data type FIS;

10.3 FIS Types

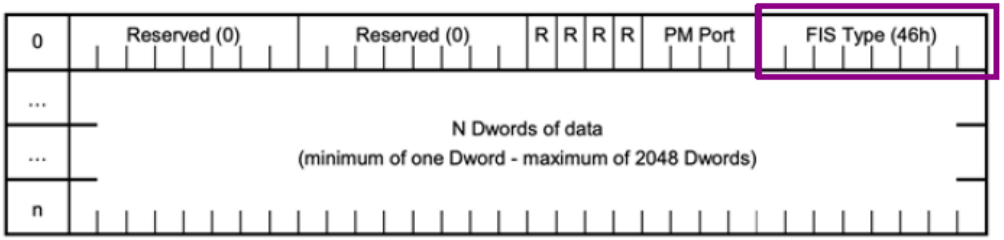
The following sections define the structure of each individual FIS.

10.3.1 FIS Type values

The value for the FIS Type fields of all FISes has been selected to provide additional robustness. In minimally buffered implementations that may not buffer a complete FIS, the state machines may begin acting on the received FIS Type value prior to the ending CRC having been checked. Because the FIS Type value may be acted upon prior to the integrity of the complete FIS being checked against its ending CRC, the FIS Type field values have been selected to maximize the Hamming distance between them.

Type field value	Description
27h	Register FIS – Host to Device
34h	Register FIS – Device to Host
39h	DMA Activate FIS – Device to Host
41h	DMA Setup FIS – Bi-directional
46h	Data FIS – Bi-directional
58h	BIST Activate FIS – Bi-directional
5Fh	PIO Setup FIS – Device to Host
A1h	Set Device Bits FIS – Device to Host
A6h	Reserved for future Serial ATA definition
B8h	Reserved for future Serial ATA definition
BFh	Reserved for future Serial ATA definition
C7h	Vendor specific
D4h	Vendor specific
D9h	Reserved for future Serial ATA definition

Figure 193 – FIS type value assignments

	10.3.11 Data - Host to Device or Device to Host (Bidirectional)  **Figure 201 – Data – Host to Device or Device to Host FIS layout** Field Definitions FIS Type - Set to a value of 46h. Defines the rest of the FIS fields. Defines the length of the FIS as n+1 Dwords. Dwords of data - Contain the actual data to transfer. Only 32 bit fields are transferred. The last Dword is padded with zeros when only a partial Dword is to be transmitted. NOTE –The maximum amount of user data that may be sent in a single Data – Host to Device or Data – Device to Host FIS is limited. See description. PM Port – When an endpoint device is attached via a Port Multiplier, specifies the device port address that the FIS should be delivered to or is received from. This field is set by the host for Host to Device transmission and this field is set by the Port Multiplier for Device to Host transmission. Endpoint devices shall set this field to 0h for Device to Host transmissions. R – Reserved – shall be cleared to zero. Source: SATA 3.0 specification
a responding step for asserting the CHECK bit of the ATAPI Status Register when the FIS is data type;	The Accused Products respond to assert the CHECK bit of the ATAPI Status Register when the FIS is the data type. Specifically, the Accused Products set the ERR bit to one when the FIS is the data type.

10.3.2 CRC Errors on Data FISes

Following a Serial ATA CRC error on a Data FIS, if the device transmits a Device-to-Host FIS it shall set the ERR bit to one and both the BSY bit and DRQ bit cleared to zero in the Status field, and the ABRT bit set to one in the Error field. It is recommended for the device to also set the bit 7 (i.e. ICRC bit) to one in the Error field. See ATA8-ACS.

Register	7	6	5	4	3	2	1	0
Error	ERROR							
Count(7:0)	na							
Count(15:8)	na							
LBA(7:0)	na							
LBA(31:24)	na							
LBA(15:8)	na							
LBA(39:32)	na							
LBA(23:16	na							
LBA(47:40)	na							
Device	na							
Status	BSY	DRDY	DF	na	DRQ	na	na	ERR

Figure 211 – WRITE FPDMA QUEUED error on command receipt

3.2 References under development

The following ANSI standards under development are referenced. Draft versions of these standards are available from http://www.T10.org or http://www.T13.org.

ATA/ATAPI-8 Serial Transport (ATA8-AST) [ANSI INCITS T13/1697-D]
ATA/ATAPI-8 Parallel Transport (ATA8-APT) [ANSI INCITS T13/1698-D]
ATA Host Adapter Standards - 2 (HBA-2) [ANSI INCITS T13/2014D]

The ERR bit in the Accused Products corresponds to the CHECK bit of the ATAPI Status Register.

	1.3 Revision 3.0 (Ratification Date: June 2, 2009) Release that incorporates errata against Revision 2.6: • ECN001 - Slimline Bump Correction • ECN002 - Bump Correction • ECN003 - State Name Corrections • ECN004 - ATA Log & Subcode reservations • ECN006 - fbaud/10 Jitter Parameter Removal • ECN008 - fbaud/500 Jitter Parameter Clarification • ECN009 - Correcting LBP references in the COMP data pattern and other locations • ECN010 - Power State Resume Speed • ECN011 - Data validity clarifications • ECN012 - Cable & Connector Retention • ECN013 - Section 13 Corrections • ECN014 - PACKET State Names • ECN016 - Long Term Frequency Accuracy and SSC Profile Tests for Transmitters • ECN017 - OOB Burst/Gap Duration Clarification • ECN018 - Figure 69 Pin Location Correction • ECN019 - Cable ISI Test Source Risetime • ECN021 - External plug height • ECN022 - Editorial cleanup – 8KB, IDENTIFY DEVICE • ECN023 - L-Key Opening Correction (Slimline Host Receptacle Connector) • ECN024 - Contact Current Rating procedure • ECN025 - Rise Time Measurements Serial ATA Revision 3.0 Gold Revision page 23 of 663 • ECN026 - Gen 3i TX TJ Measurement Location • ECN027 - Gen3i Rx Differential Return Loss Text Description and Figure Change - Clarification Only • ECN028 - Clarification of Test Patterns for Measurement Defined in 7.2 Electrical Specification • ECN029 - Addition of Pattern to the TX AC Common Mode Voltage (Gen3i) Measurement • ECN031 - Correction to ECN 004 • ECN032 - Correction to TX AC Common Mode Voltage Table Value 'Units' • ECN033 - Definition of Terms • ECN034 - Corrections to Technical Proposal 005 • ECN035 - Clarification of Words 76 to 79 • ECN036 - LIF-SATA Clarifications • ECN037 - Changes made by Technical Integration Work Group • ECN038 - Key clarification and the following new features and enhancements • TP002 - Gen3 register assignments • TP004 - NCQ Clarifications • TP005 - SATA Speed Indicator in ID String • TP007 - Automatic Partial to Slumber Transitions • TP009 - LIF Connector for 1.8" HDD for SATA Revision 2.6 • TP010 - Serial ATA NCQ Streaming Command • TP011 - Serial ATA NCQ QUEUE MANAGEMENT Command • TP012 - Gen 1 Clock to Data Jitter Definition • TP013 - Connector for 7 mm slimline drives • TP014 - Allow READ LOG DMA EXT to Clear NCQ Error • TP015 - Remove Device Register from Signature • TP016 - Add Write-Read-Verify to SSP support • TP017 - Align SATA 2.6 with ATA8-ACS • TP018 - Specification Revisions For Gen3i Source: SATA 3.0 specification
and sending back the response.	The Accused Products send back the response.

15.3 Link layer error handling overview

15.3.1 Error detection

There are two primary categories of errors that the Link layer detects internally:

Invalid state transitions
Data integrity errors

Invalid state transition errors may arise from a number of sources and the Link layer responses to many such error conditions are defined in section 1. Data integrity errors generally arise from noise in the physical interconnect.

15.3.2 Error control actions

Errors detected by the Link layer during a transmission are generally handled by accumulating the errors until the end of the transmission and reflecting the reception error condition in the final R_ERR_P /R_OK_P handshake. Specific scenarios are listed in the following sections.

Data integrity errors:
Data integrity errors are generally handled by signaling the Transport layer in order to potentially trigger a transmission retry operation, or to convey failed status information to the host software. In order to return to a known state, data integrity errors are usually signaled via the frame acknowledgement handshake, at the end of a frame transmission, before returning to the idle state.

The following paragraphs outline the requirements during specific data integrity error scenarios, and the respective Link error control actions:

On detection of a CRC error at the end of receiving a frame (at EOF_P), the Link layer shall notify the Transport layer that the received frame contains a CRC error. Furthermore, the Link layer shall issue the negative acknowledgement, R_ERR_P, as the frame status handshake, and shall return to the idle state.

On detection of a disparity error or other 8b/10b coding violation during the receipt of a frame, the Link layer shall retain this error information, and at the close of the received frame the Link layer shall provide the negative acknowledgement, R_ERR_P, as the frame handshake, and shall notify the Transport layer of the error.

The control actions are essentially the same for coding violations as for CRC errors.

15 Error handling

15.1 Architecture

The layered architecture of Serial ATA extends to error handling as well. As indicated in Figure 250, each layer in the Serial ATA stack has as inputs error indication from the next lower layer (except for the Phy layer which has no lower layer associated with it), data from the next lower layer in the stack, and data from the next higher layer in the stack. Each layer has its local error detection capability to identify errors specific to that layer based on the received data from the lower and higher layers. Each layer performs local recovery and control actions and may forward error information to the next higher layer in the stack.

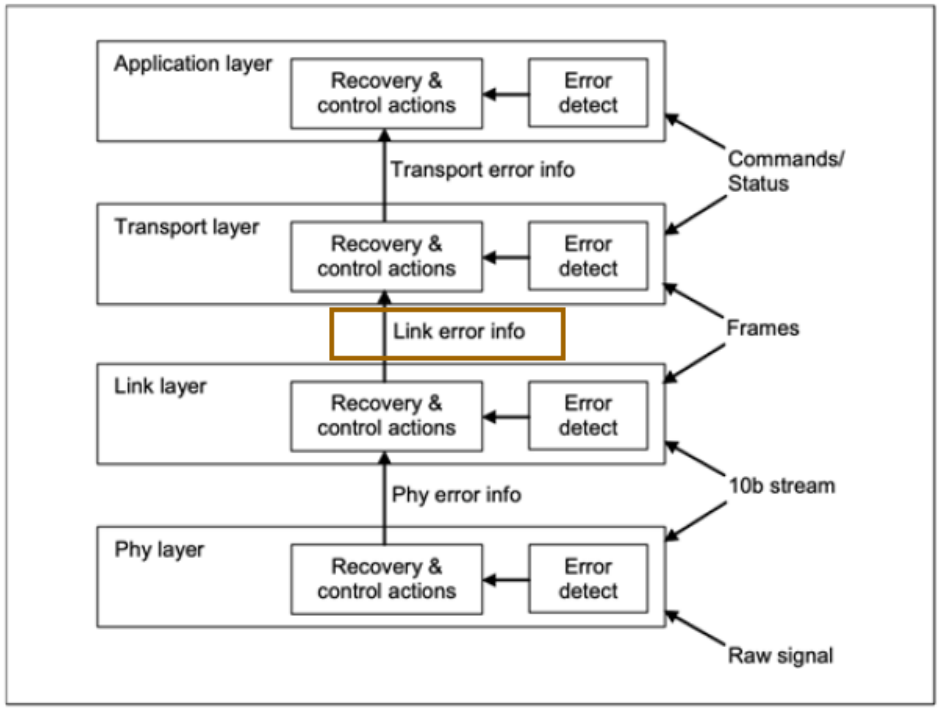


Figure 250 – Error handling architecture

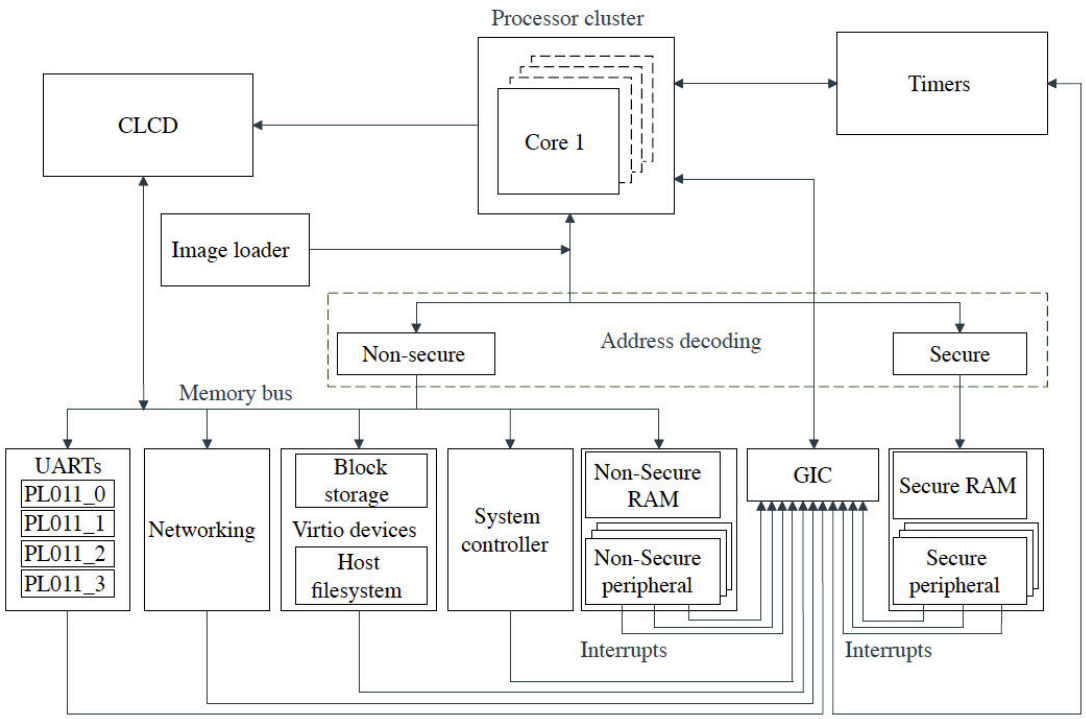
EXHIBIT D

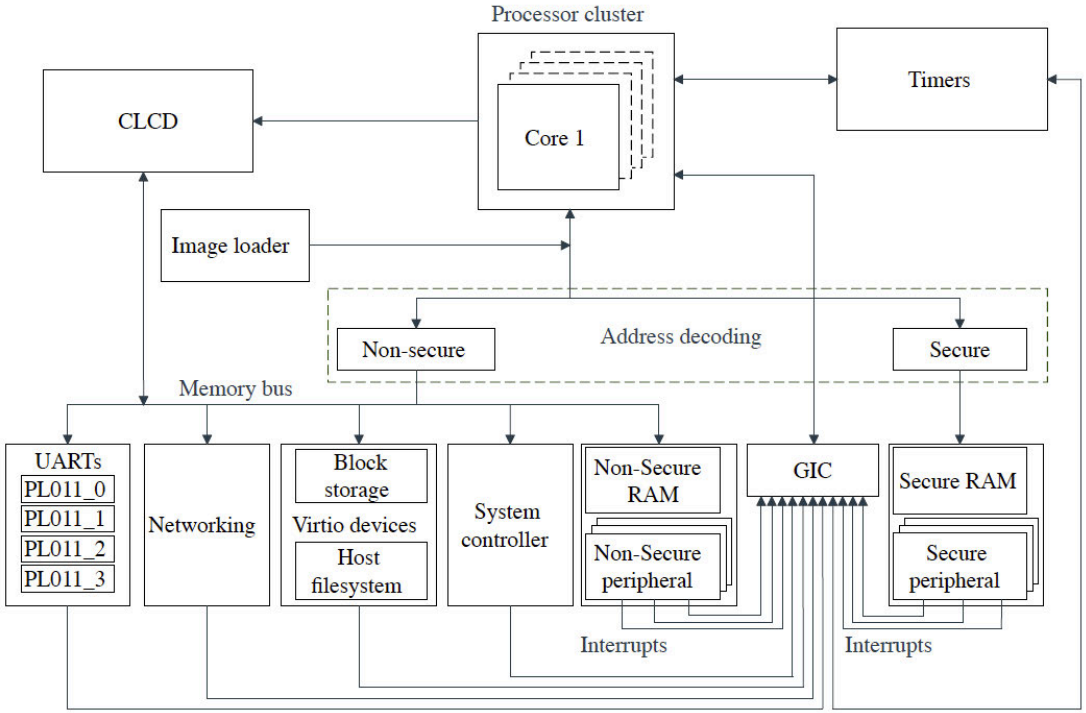
U.S. Patent No. 8,407,783	NVIDIA GRACE CPUs Incorporating Neoverse V2 Core
1. A computing system providing normal security services and high security services, comprising:	The Lenovo ThinkPad T14s Gen 6 with Snapdragon® is a representative example of the Accused Products. The Accused Products include one or more computing systems providing normal security services and high security services. The T14s Gen 6 includes a Snapdragon X Elite or Snapdragon X Plus processor. ThinkPad T14s Gen 6 (Snapdragon) Lenovo PSREF Product Specifications Reference **PERFORMANCE** **Processor** **Processor Family** Snapdragon® X Elite or Snapdragon® X Plus **Processor**** Processor Name: Snapdragon® X Elite X1E-78-100; Cores: 12; Max Frequency: 3.4GHz (multi-core); Cache: 42MB Total; Processor Graphics: Qualcomm® Adreno™ GPU; NPU: Qualcomm® Hexagon™ NPU, up to 45 TOPS Processor Name: Snapdragon® X Plus X1P-42-100; Cores: 8; Max Frequency: 3.4GHz (single-core) / 3.2GHz (multi-core); Cache: 30MB Total; Processor Graphics: Qualcomm® Adreno™ GPU; NPU: Qualcomm® Hexagon™ NPU, up to 45 TOPS Source: https://psref.lenovo.com/Product/ThinkPad_T14s_Gen_6_Snapdragon?tab=spec These processors implement the ARMv8-A architecture.

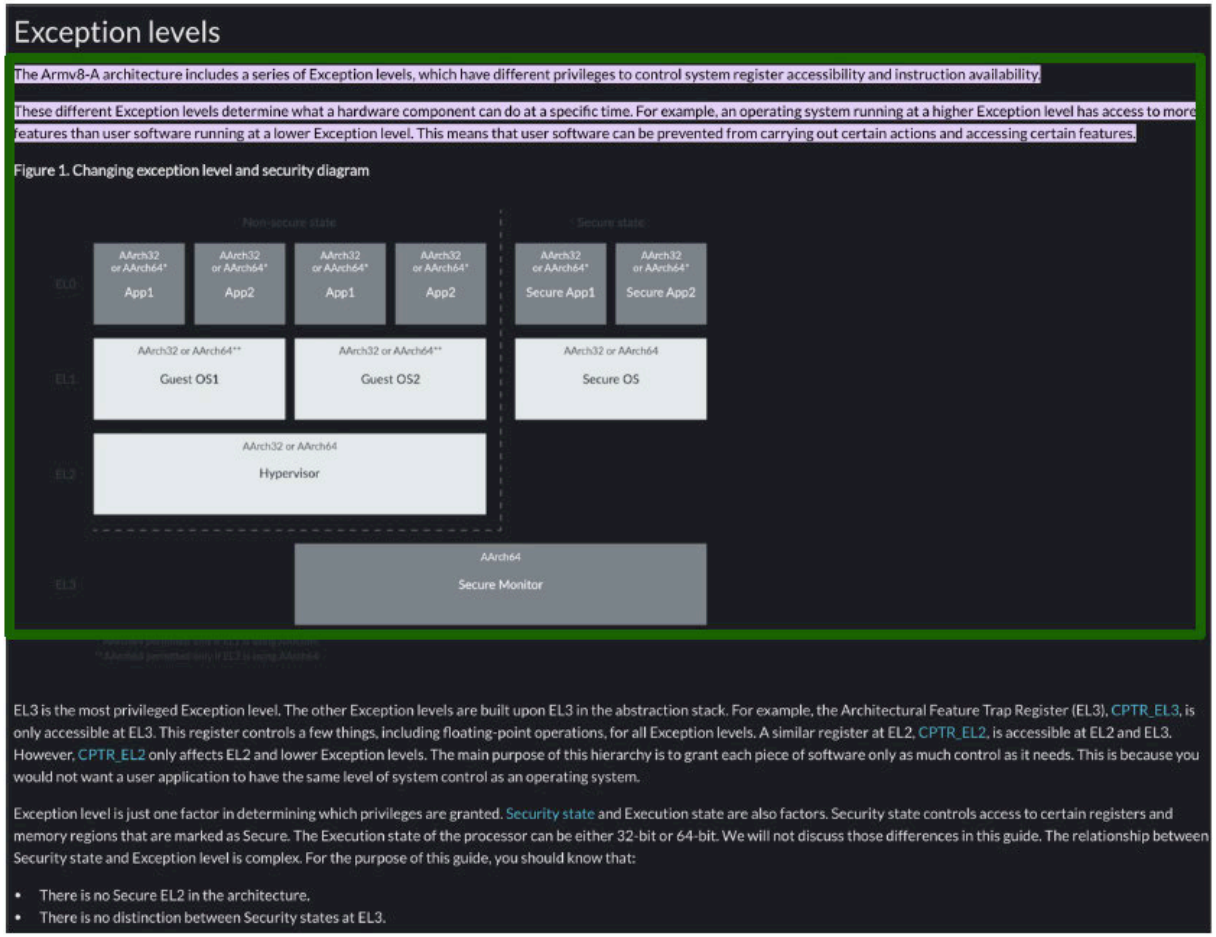
The Snapdragon X processor is ARMv8.7-A compliant, which means it implements the mandatory features of all extensions from ARMv8.0-A (the base specification) to ARMv8.7-A. Some of these notable extensions include atomic instructions, CRC instructions, and floating point to integer instructions. Some notable features NOT implemented include AArch32 support, Big-endian support, and SVE instructions.

Source: https://docs.qualcomm.com/doc/80-78185-2/topic/architecture.html

The Accused Products provide normal security (non-secure) and high security (secure) services:



	Source: https://developer.arm.com/documentation/100961/1118/Foundation-Platform-Introduction/Platform-overview/Foundation-Platform-block-diagram
hardware resources, grouped into a plurality of resource security levels;	The Accused Products include hardware resources, grouped into a plurality of resource security levels. For example, the Accused Products include non-secure RAM and secure RAM resources.  Source: https://developer.arm.com/documentation/100961/1118/Foundation-Platform-Introduction/Platform-overview/Foundation-Platform-block-diagram Additionally, the Accused Products include a series of Exception levels, which have different privileges to control the system register accessibility and instruction availability. These different Exception levels determine what a hardware component can do at a specific time.

	 Exception levels The Armv8-A architecture includes a series of Exception levels, which have different privileges to control system register accessibility and instruction availability. These different Exception levels determine what a hardware component can do at a specific time. For example, an operating system running at a higher Exception level has access to more features than user software running at a lower Exception level. This means that user software can be prevented from carrying out certain actions and accessing certain features. Figure 1. Changing exception level and security diagram Non-secure state Secure state EL0 AArch32 or AArch64* App1 AArch32 or AArch64* App2 AArch32 or AArch64* App1 AArch32 or AArch64* App2 AArch32 or AArch64* Secure App1 AArch32 or AArch64* Secure App2 EL1 AArch32 or AArch64** Guest OS1 AArch32 or AArch64** Guest OS2 AArch32 or AArch64 Secure OS EL2 AArch32 or AArch64 Hypervisor EL3 AArch64 Secure Monitor EL3 is the most privileged Exception level. The other Exception levels are built upon EL3 in the abstraction stack. For example, the Architectural Feature Trap Register (EL3), CPTR_EL3, is only accessible at EL3. This register controls a few things, including floating-point operations, for all Exception levels. A similar register at EL2, CPTR_EL2, is accessible at EL2 and EL3. However, CPTR_EL2 only affects EL2 and lower Exception levels. The main purpose of this hierarchy is to grant each piece of software only as much control as it needs. This is because you would not want a user application to have the same level of system control as an operating system. Exception level is just one factor in determining which privileges are granted. Security state and Execution state are also factors. Security state controls access to certain registers and memory regions that are marked as Secure. The Execution state of the processor can be either 32-bit or 64-bit. We will not discuss those differences in this guide. The relationship between Security state and Exception level is complex. For the purpose of this guide, you should know that: • There is no Secure EL2 in the architecture. • There is no distinction between Security states at EL3. Source: https://developer.arm.com/documentation/102437/0100/Exception-levels
a processor core, switching between	The Accused Products include a processor core, switching between different security states including a normal security state for providing the normal security services and a high security state for providing the high security services.

different security states including a normal security state for providing the normal security services and a high security state for providing the high security services, and assigning a user access right to a request in accordance with the security state of the processor core and a protection level of the computing system, wherein,	The processor core of the Accused Products assigns a user access right to a request in accordance with the security state of the processor core and a protection level of the computing system. "TrustZone" in the Accused Products provides for Secure or Non-Secure state for the processor. This is selected by the SCR_EL3.NS bit. **Changing Security state** If TrustZone is implemented then the processor can be in either Secure state or Non-secure state. This is selected by the SCR_EL3.NS bit. You may have noted in the previous diagram that EL3 is in Secure state. EL3 is the most privileged Exception level and the Security state of EL3 is fixed. EL3 is able to access all copies of a banked System register. In Armv8-A EL3 is always in Secure state. In Armv9-A, EL3 is part of Secure state unless RME is implemented. If RME is implemented, Root state is a separation of EL3 from the rest of Secure state. RME is covered more in the next section. Whenever you want to switch from one Security state to another you must pass through EL3. Software at EL3 is responsible for managing access to the different available Security states and acts as a gatekeeper, controlling access to the Security states of EL2, EL1 and EL0. The SCR_EL3.NS bit enables a change of Security state on return from EL3. **Note** At EL0, EL1, and EL2 the PE can be in either Secure state or Non-secure state. You often see this written as: • NS_EL1: Non-secure state, Exception level 1 • S_EL1: Secure state, Exception level 1 Changing Security state is discussed in more detail in TrustZone for AArch64 and Realm Management Extension. Source: https://developer.arm.com/documentation/102412/0103/Execution-and-Security-states/Security-states In the Secure state, a Processing Element in the Accused Products can access both the Secure and Non-secure physical address spaces, and the Secure copy of banked registers. In the Non-secure state, or "Normal world," a Processing Element can only access the Non-secure physical address space.

Security states

AArch64 allows for the implementation of multiple Security states. This allows a further partitioning of software to isolate and compartmentalize trusted software.

Most Cortex-A processors support two Security states:

Secure state

In this state, a Processing Element (PE) can access both the Secure and Non-secure physical address spaces, and the Secure copy of banked registers.

Non-secure state

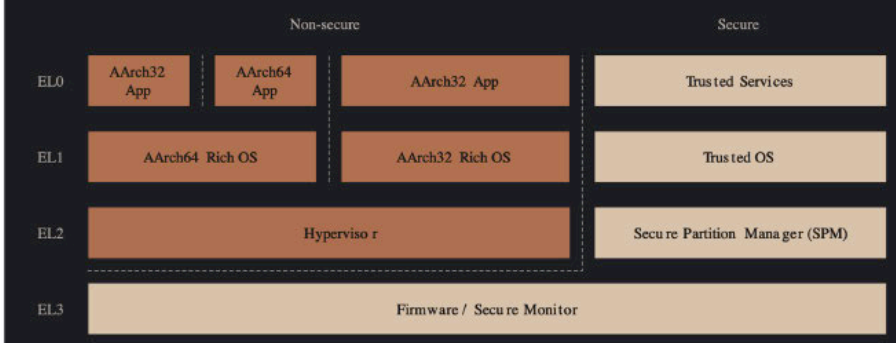
This is often also referred to as Normal world. In this state, a PE can only access the Non-secure physical address space. The PE can only access System registers that allow non-secure accesses.

The Security state defines which of the implemented Exception levels can be accessed, which areas of memory can currently be accessed, and how those accesses are represented on the system memory bus. If in Non-secure state, the PE can only access the Non-secure physical address space. In Secure state, the PE can access both the Secure and Non-secure physical address spaces.

An example of this would be to have your operating system, such as Android, running in the Normal world, with a payment or DRM system running in the Secure world. We need a higher degree of trust in the Secure world systems and need to keep them separate to protect information such as payment details and keys. Having the two security states provides this separation.

This diagram shows the Exception levels and Security states, with different Execution states being used:

Figure 1. Exception levels and Security states using different Execution states



The uses of these Security states are described in more detail in our guide TrustZone for AArch64.

https://developer.arm.com/documentation/102412/0103/Execution-and-Security-states/Security-states

Types of privilege

There are two types of privilege relevant to the AArch64 Exception model:

- Privilege in the memory system
- Privilege from the point of view of accessing processor resources

Both types of privilege are affected by the current privileged Exception level.

Memory privilege

The A-profile of the Arm architecture implements a virtual memory system, in which a Memory Management Unit (MMU) allows software to assign attributes to regions of memory. These attributes include read/write permissions that can be configured to allow separate access permissions for privileged and unprivileged accesses.

Memory access initiated when the processor is executing in EL0 are checked against the unprivileged access permissions. Memory accesses from EL1, EL2, and EL3 are checked against the privileged access permissions.

Because this memory configuration is programmed by software using the MMU's translation tables, you should consider the privilege necessary to program those tables. The MMU configuration is stored in System registers, and the ability to access those registers is also controlled by the current Exception level.

Note

In the Arm architecture, registers are split into two main categories:

- Registers that provide system control or status reporting
- Registers that are used in instruction processing, for example to accumulate a result, and in handling exceptions

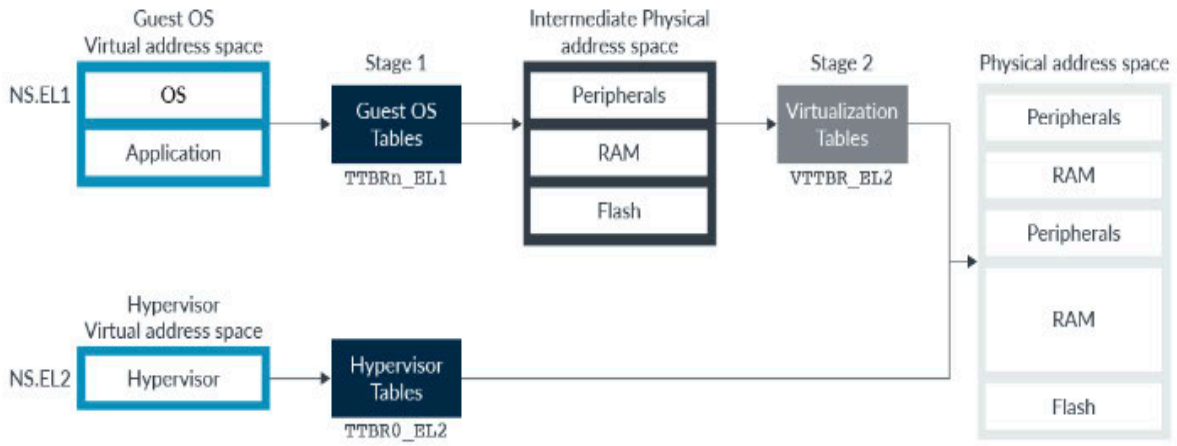
Feedback

Source: https://developer.arm.com/documentation/102412/0103/Privilege-and-Exception-levels/Types-of-privilege

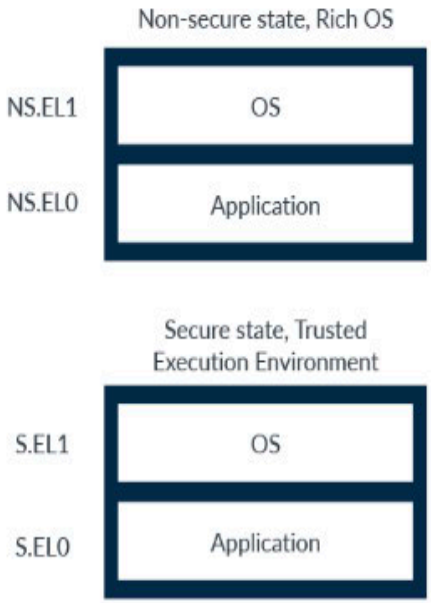
	EL3 is the most privileged Exception level. The other Exception levels are built upon EL3 in the abstraction stack. For example, the Architectural Feature Trap Register (EL3), CPTR_EL3, is only accessible at EL3. This register controls a few things, including floating-point operations, for all Exception levels. A similar register at EL2, CPTR_EL2, is accessible at EL2 and EL3. However, CPTR_EL2 only affects EL2 and lower Exception levels. The main purpose of this hierarchy is to grant each piece of software only as much control as it needs. This is because you would not want a user application to have the same level of system control as an operating system. Exception level is just one factor in determining which privileges are granted. Security state and Execution state are also factors. Security state controls access to certain registers and memory regions that are marked as Secure. The Execution state of the processor can be either 32-bit or 64-bit. We will not discuss those differences in this guide. The relationship between Security state and Exception level is complex. For the purpose of this guide, you should know that: • There is no Secure EL2 in the architecture. • There is no distinction between Security states at EL3. Source: https://developer.arm.com/documentation/102437/0100/Exception-levels Furthermore, secure state EL1 and EL0 have different virtual address spaces from non-secure state EL1 and EL0.

3.3. Virtual address spaces

The memory management guide in this series introduced the idea of multiple virtual address spaces, or translation regimes. For example, there is a translation regime for EL0/1 and a separate translation regime for EL2, shown here:



There are also separate translation regimes for the Secure and Non-secure states. For example, there is a Secure EL0/1 translation regime and Non-secure EL0/1 translation regime, which is shown here:



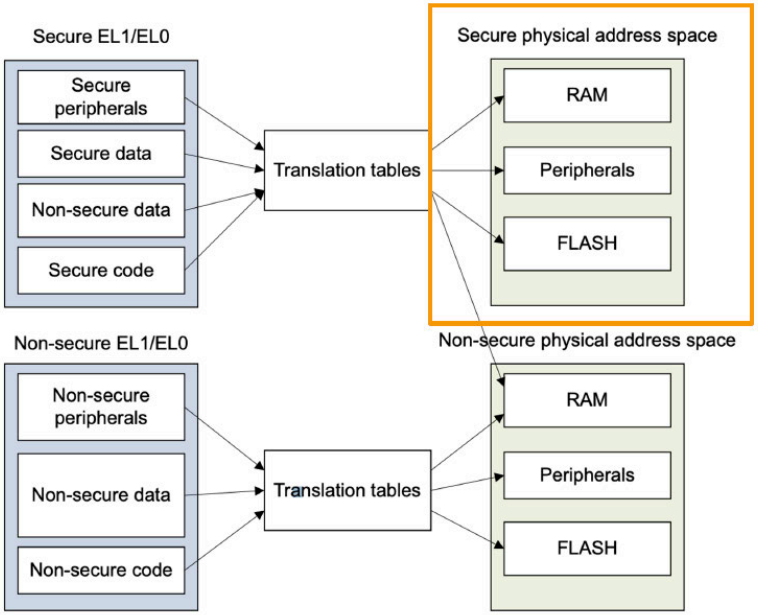
Source: https://developer.arm.com/-/media/Arm%20Developer%20Community/PDF/Learn%20the%20Architecture/TrustZone%20for%20Armv8-A.pdf

The Secure state EL1 and EL0 have a different virtual address space from the Non-secure state EL1 and EL0. This permits secure side code from the AArch32 32-bit architecture to be used in a system with a 64-bit operating system or hypervisor running on the Non-secure side.

Source: https://developer.arm.com/documentation/100935/0100/Switching-betwen-the-Normal-and-Secure-worlds-

Software running in the Normal World (non-Secure state) can only make Non-secure access to memory. Software running in the Secure world can make Secure and Non-secure accesses for specific memory mappings using the NS and NSTable flags in its page table entries.

in comparison with the normal security state, the user access right assigned in the high security state for a particular protection level of the computing system further allows the request to use hardware resources of a higher resource security level, and, for a particular security state of the processor core, the	In comparison with the normal security state (Normal World, non-Secure state) in the Accused Products, the user access right assigned in the high security state (Secure state) allows use of hardware resources of a higher resource security level. **17.1 TrustZone hardware architecture** The TrustZone architecture provides a means for system designers to help secure systems, using the TrustZone security extensions, and secure peripherals. Low-level programmers must understand the restrictions placed on the system by the TrustZone architecture, even if they do not use the security features. The ARM security model divides up device hardware and software resources, so that they exist in either the *Secure world* for the security subsystem, or the *Normal world* for everything else. System hardware ensures that no Secure world assets can be accessed from the Normal world. A Secure design places all sensitive resources in the Secure world, and ideally has robust software running that can protect assets against a wide range of possible software attacks. The *ARM Architecture Reference Manual* uses the terms *Secure* and *Non-secure* to refer to system security *states*. A *Non-secure state* does not automatically mean security vulnerability, but rather normal operation and is therefore the same as the *Normal world*. Normally, there is a master and slave relationship between Non-secure and Secure worlds, and so the Secure world is only executed when the Normal world performs a *Secure Monitor Call* (SMC), (see SMC instruction in *ARMv8-A Architecture Reference Manual*). The use of the word *world* is really used to describe not just the execution state, but also all memory and peripherals that are only accessible in that state.

user access right assigned for a lower protection level covers the user access right assigned for a higher protection level; and	 **Figure 12-9 Physical Address spaces**

	The Secure monitor EL3 has its own dedicated translation tables. The table base address is specified in TTBR0_EL3 and configured via TCR_EL3. Translation tables are capable of accessing both Secure and Non-secure Physical Addresses. TTBR0_EL3 is used only in Secure monitor EL3 mode, not by the trusted kernel itself. When the transition to Secure world has completed, the trusted kernel uses the EL1 translations, that is, the translation tables pointed to by TTBR0_EL1 and TTBR1_EL1. As these registers are not banked in AArch64, Secure monitor code must configure new tables for the Secure world and save and restore copies of TTBR0_EL1 and TTBR1_EL1. The EL1 translation regime behaves differently in Secure state, compared to its normal operation in Non-secure state. The second stage of translation is disabled and the EL1 translation regime is now able to point to both Secure or Non-secure Physical Addresses. There is no virtualization in the Secure world so that the IPA is always the same as the final PA. Entries in the TLB are tagged as Secure or Non-secure, so that no TLB maintenance is ever required when you transition between Secure and Normal worlds. In AArch64, EL3 has its own translation tables, governed by the registers TTBR0_EL3 and TCR_EL3. Only stage one translations are allowed in the Secure world and there is no TTBR1_EL3. The AArch64 EL1 translation table registers are not banked between security states and therefore the value of TTBR0_EL1, TTBR1_EL1, and TCR_EL1 must be saved and restored for each world as part of the Secure Monitor's context switching operation. This enables each world to have a local set of translation tables, with the Secure world mappings hidden and protected from the Normal world. Entries in the Secure World translation tables contain NS and NSTable attribute bits that determine whether particular accesses can access the Secure or Non-secure physical address space. Source: https://developer.arm.com/documentation/den0024/a/ (17-4) A Processing Element in the secure state can access both the secure and non-secure physical address spaces and the secure copy of banked registers. A Processing Element in the non-secure state can access only the non-secure physical address space and system registers that allow non-secure access.

Security states

AArch64 allows for the implementation of multiple Security states. This allows a further partitioning of software to isolate and compartmentalize trusted software.

Most Cortex-A processors support two Security states:

Secure state

In this state, a Processing Element (PE) can access both the Secure and Non-secure physical address spaces, and the Secure copy of banked registers.

Non-secure state

This is often also referred to as Normal world. In this state, a PE can only access the Non-secure physical address space. The PE can only access System registers that allow non-secure accesses.

The Security state defines which of the implemented Exception levels can be accessed, which areas of memory can currently be accessed, and how those accesses are represented on the system memory bus. If in Non-secure state, the PE can only access the Non-secure physical address space. In Secure state, the PE can access both the Secure and Non-secure physical address spaces.

An example of this would be to have your operating system, such as Android, running in the Normal world, with a payment or DRM system running in the Secure world. We need a higher degree of trust in the Secure world systems and need to keep them separate to protect information such as payment details and keys. Having the two security states provides this separation.

This diagram shows the Exception levels and Security states, with different Execution states being used:

Figure 1. Exception levels and Security states using different Execution states

The uses of these Security states are described in more detail in our guide TrustZone for AArch64.

https://developer.arm.com/documentation/102412/0103/Execution-and-Security-states/Security-states

In addition, for a particular security state of the Accused Products' processor core, the user access right assigned for a lower protection level covers the user access right assigned for a higher protection level.

In the Accused Products, higher Exception levels (lower protection) have the privilege to access registers that control lower Exception levels (higher protection). For example, EL2 has the privilege to access SCTLR_EL1.

Register access

Configuration settings for AArch64 processors are held in a series of registers known as System registers. The combination of settings in the System registers defines the current processor context. Access to the System registers is controlled by the current Exception level.

For example, VBAR_EL1 is the Vector Base Address Register. We will describe what it is used for later in this guide. For now, what is important is the _EL1 suffix. This tells us that software needs at least EL1 privilege to access the register.

The architecture has many registers with conceptually similar functions that have names that differ only by their Exception level suffix. These are independent, individual registers that have their own encodings in the instruction set and will be implemented separately in hardware.

The registers have similar names to reflect that they perform similar tasks, but they are entirely independent registers with their own access semantics. The suffix of the System register name indicates the lowest Exception level from which that register can be accessed.

There is a System Control Register (SCTLR) for each implemented Exception level. Each register controls the architectural features for that EL, such as the MMU, caches and alignment checking:

SCTLR_EL1

Top-level system control for EL0 and EL1

SCTLR_EL2

Top-level system control for EL2

SCTLR_EL3

Top-level system control for EL3

Feedback

Note

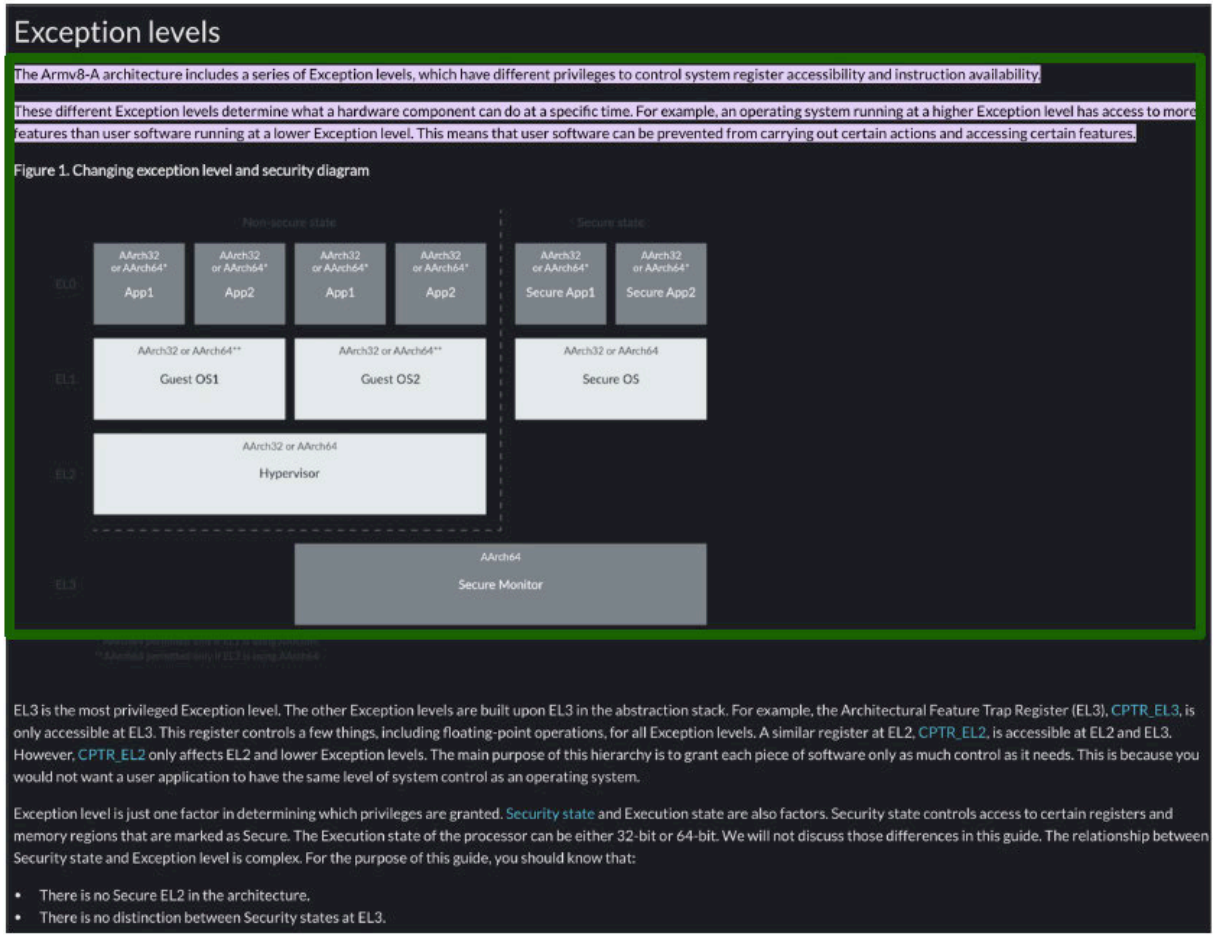
EL1 and EL0 share the same MMU configuration and control is restricted to privileged code running at EL1. Therefore there is no SCTLR_EL0 and all control is from the EL1 accessible register. This model is generally followed for other control registers.

Higher Exception levels have the privilege to access registers that control lower levels. For example, EL2 has the privilege to access SCTLR_EL1 if necessary. This register cannot be accessed from EL0, and any attempt to do so generates an exception.

In the general operation of the system, the privileged Exception levels control their own configuration. However, more privileged levels sometimes access registers associated with lower Exception levels. For example, this could be to implement virtualization features and context switching.

Source: https://developer.arm.com/documentation/102412/0103/Privilege-and-Exception-levels/Types-of-privilege

This is further demonstrated in the description of exception levels which describes how there is a hierarchy of exception levels at which privilege (and thus access) flows strictly downwards in the hierarchy.

	 Exception levels The Armv8-A architecture includes a series of Exception levels, which have different privileges to control system register accessibility and instruction availability. These different Exception levels determine what a hardware component can do at a specific time. For example, an operating system running at a higher Exception level has access to more features than user software running at a lower Exception level. This means that user software can be prevented from carrying out certain actions and accessing certain features. Figure 1. Changing exception level and security diagram Non-secure state Secure state EL0 AArch32 or AArch64* App1 AArch32 or AArch64* App2 AArch32 or AArch64* App1 AArch32 or AArch64* App2 AArch32 or AArch64* Secure App1 AArch32 or AArch64* Secure App2 EL1 AArch32 or AArch64** Guest OS1 AArch32 or AArch64** Guest OS2 AArch32 or AArch64 Secure OS EL2 AArch32 or AArch64 Hypervisor EL3 AArch64 Secure Monitor EL3 is the most privileged Exception level. The other Exception levels are built upon EL3 in the abstraction stack. For example, the Architectural Feature Trap Register (EL3), CPTR_EL3, is only accessible at EL3. This register controls a few things, including floating-point operations, for all Exception levels. A similar register at EL2, CPTR_EL2, is accessible at EL2 and EL3. However, CPTR_EL2 only affects EL2 and lower Exception levels. The main purpose of this hierarchy is to grant each piece of software only as much control as it needs. This is because you would not want a user application to have the same level of system control as an operating system. Exception level is just one factor in determining which privileges are granted. Security state and Execution state are also factors. Security state controls access to certain registers and memory regions that are marked as Secure. The Execution state of the processor can be either 32-bit or 64-bit. We will not discuss those differences in this guide. The relationship between Security state and Exception level is complex. For the purpose of this guide, you should know that: • There is no Secure EL2 in the architecture. • There is no distinction between Security states at EL3. Source: https://developer.arm.com/documentation/102437/0100/Exception-levels
an access right checker, determining whether the request has	The Accused Products include an access right checker that determines whether the user request has the authority to use the hardware resources in accordance with the assigned user access right and the resource security levels of required hardware resources of the request. For example, the Accused Products check whether a user's access rights include access to "Normal world" or "Secure world" subsystems depending on the resource security levels of hardware resources required by the request.

the authority to use the hardware resources in accordance with the assigned user access right and the resource security levels of required hardware resources of the request, wherein:	**17.1 TrustZone hardware architecture** The TrustZone architecture provides a means for system designers to help secure systems, using the TrustZone security extensions, and secure peripherals. Low-level programmers must understand the restrictions placed on the system by the TrustZone architecture, even if they do not use the security features. The ARM security model divides up device hardware and software resources, so that they exist in either the *Secure world* for the security subsystem, or the *Normal world* for everything else. System hardware ensures that no Secure world assets can be accessed from the Normal world. A Secure design places all sensitive resources in the Secure world, and ideally has robust software running that can protect assets against a wide range of possible software attacks. The *ARM Architecture Reference Manual* uses the terms *Secure* and *Non-secure* to refer to system security *states*. A *Non-secure state* does not automatically mean security vulnerability, but rather normal operation and is therefore the same as the *Normal world*. Normally, there is a master and slave relationship between Non-secure and Secure worlds, and so the Secure world is only executed when the Normal world performs a *Secure Monitor Call* (SMC), (see SMC instruction in *ARMv8-A Architecture Reference Manual*). The use of the word *world* is really used to describe not just the execution state, but also all memory and peripherals that are only accessible in that state.

	The memory system is divided by means of an additional bit that accompanies the address of peripherals and memory. This bit, called the NS-bit, indicates whether the access is Secure or Non-secure. This bit is added to all memory system transactions, including cache tags and access to system memory and peripherals. This additional address bit gives a physical address space for the Secure world and a completely separate physical address space for the Normal world. Software running in the Normal World can only make Non-secure accesses to memory, because the core always sets the NS bit to 1 in any memory transaction generated by the Normal World. Software running in the Secure world usually makes only Secure memory accesses, but can also make Non-secure accesses for specific memory mappings using the NS and NSTable flags in its page table entries. Trying to perform a Non-secure access to cached data that is marked as Secure causes a cache miss. Trying to perform a Non-secure access to external memory marked as Secure causes the memory system to disallow the request and the slave device returns an error response. There is no indication to the Non-secure system that the error is caused by an attempted access to Secure memory. Source: https://developer.arm.com/documentation/den0024/a/

Secure Configuration Register

The *Secure Configuration Register* (SCR) defines the configuration of the current state. It specifies the state of the core (Secure or Non-secure), and what mode the core branches to if an IRQ, FIQ or external abort occurs. The register also defines whether or not the F and A bits in the CPSR can be modified when NS-state==1.

The SCR is accessible in Secure Privileged modes only. An attempt to access this register in any other mode results in an Undefined Instruction exception.

Changing from Secure to Non-secure state

It is recommended that the SCR is modified only in Monitor mode. Monitor mode is responsible for switching between states.

To return to Non-secure state, set the NS-bit in the SCR and then execute a `MOVS` or `SUBS` instruction. All implementations must ensure that any prefetched instructions following `MOVS` or `SUBS` (or equivalent) instructions cannot be executed with the secure access permissions.

Caution

To avoid security holes, it is strongly recommended that:

- you do not use an `MSR` instruction to change from Secure to Non-secure state
- you do not alter the NS-bit in any mode except Monitor mode.

The usual mechanism for changing from Secure to Non-secure state is an exception return.

SCR-bit allocation

31	7	6	5	4	3	2	1	0
SBZ		nET	AW	FW	EA	FIQ	IRQ	NS

The reset value of the SCR is `0x00000000`.

Use the following instructions to alter or read the SCR:

```
MCR CP15, 0, <Rd>, C1, C1, 0 ; moves contents of <Rd> into SCR
MRC CP15, 0, <Rd>, C1, C1, 0 ; moves contents of SCR into <Rd>
```

NS-bit

The NS-bit (bit[0]), together with the mode bits in the CPSR, defines whether the core is in Secure state or Non-secure state. This is shown in Table 3.8.

NS-bit value	Monitor mode	All modes except Monitor mode
0	Secure state	Secure state
1	Secure state	Non-secure state

The value of the NS-bit also affects the accessibility of the banked CP15 registers in Monitor mode, see *Access to registers in Monitor mode*.

Additionally, the Accused Products check whether a user's access rights include, for the given exception level of the system, privilege to access hardware resources depending on the resource security levels of hardware resources required by the request.

Types of privilege

There are two types of privilege relevant to the AArch64 Exception model:

- Privilege in the memory system
- Privilege from the point of view of accessing processor resources

Both types of privilege are affected by the current privileged Exception level.

Memory privilege

The A-profile of the Arm architecture implements a virtual memory system, in which a Memory Management Unit (MMU) allows software to assign attributes to regions of memory. These attributes include read/write permissions that can be configured to allow separate access permissions for privileged and unprivileged accesses.

Memory access initiated when the processor is executing in EL0 are checked against the unprivileged access permissions. Memory accesses from EL1, EL2, and EL3 are checked against the privileged access permissions.

Because this memory configuration is programmed by software using the MMU's translation tables, you should consider the privilege necessary to program those tables. The MMU configuration is stored in System registers, and the ability to access those registers is also controlled by the current Exception level.

Note

In the Arm architecture, registers are split into two main categories:

- Registers that provide system control or status reporting
- Registers that are used in instruction processing, for example to accumulate a result, and in handling exceptions

Source: https://developer.arm.com/documentation/102412/0103/Privilege-and-Exception-levels/Types-of-privilege

Program Status Registers (PSRs)

The application level programmers' model provides the Application Program Status Register, see *The Application Program Status Register (APSR)*. This is an application level alias for the *Current Program Status Register* (CPSR). The system level view of the CPSR extends the register, adding system level information.

Each of the exception modes has its own saved copy of the CPSR, the *Saved Program Status Register* (SPSR), as shown in Figure 10.1. For example, the SPSR for Monitor mode is called SPSR_mon.

The Current Program Status Register (CPSR)

The *Current Program Status Register* (CPSR) holds processor status and control information:

- the APSR, see *The Application Program Status Register (APSR)*
- the current instruction set state, see *ISETSTATE*
- the execution state bits for the Thumb If-Then instruction, see *ITSTATE*
- the current endianness, see *ENDIANSTATE*
- the current processor mode
- interrupt and asynchronous abort disable bits.

The non-APSR bits of the CPSR have defined reset values. These are shown in the `TakeReset()` pseudocode function, see *Reset*.

Writes to the CPSR have side-effects on various aspects of processor operation. All of these side-effects, except for those on memory accesses caused by fetching instructions, are synchronous to the CPSR write. This means they are guaranteed not to be visible to earlier instructions in the execution stream, and they are guaranteed to be visible to later instructions in the execution stream.

Fetching an instruction causes an instruction fetch memory access. In addition, in a Virtual Memory System Architecture (VMSA) implementation, fetching an instruction can cause a translation table walk. The privilege of these memory accesses can be affected by changes to the mode field of the CPSR. Also, if the Security Extensions are implemented the virtual memory space of these accesses can be affected by changes to the mode field. Those mode changes take effect on the memory accesses as follows:

- A mode change by an exception entry is synchronous to the exception entry. This applies to all exception entries, including the exception entry for a synchronous exception generated directly by an instruction.
- A mode change by an exception return instruction is synchronous to the instruction.
- A mode change by an instruction other than an exception return and that is not the result of a synchronous exception generated directly by the instruction. Such a mode change can be the result of a `CPS` or `MSR` instructions, and:
  - is guaranteed not to be visible to memory accesses caused by fetching earlier instructions in the execution stream
  - is guaranteed to be visible to memory accesses caused by fetching instructions after the next exception entry, exception return instruction, or `ISB` instruction in the execution stream
  - might or might not affect memory accesses caused by fetching instructions between the mode change instruction and the point where mode changes are guaranteed to be visible.

See *Exception return* for the definition of exception return instructions.

	Security state An ARMv8 implementation that includes the EL3 Exception level provides the following Security states, each with an associated memory address space: **Secure state** In Secure state, the processor: • Can access both the Secure memory address space and the Nonsecure memory address space. • When executing at EL3, can access all the system control resources. **Non-secure state** In Non-secure state, the processor: • Can access only the Non-secure memory address space. • Cannot access the Secure system control resources. The AArch32 Security state model is unchanged from the model for an ARMv7 implementation that includes the Security Extensions and the Virtualization Extensions. When the implementation uses the AArch32 Execution state for all Exception levels, many system registers are banked to provide Secure and Non-secure instances, and: • The Secure instance is accessible only at EL3. • The Non-secure instance is accessible at EL1 or higher. • The two instances of a banked register have the same name. The *ARMv8 security model* describes how the Security state interacts with other aspects of the ARMv8 architectural state. ← → Source: https://developer.arm.com/documentation/ddi0500/e/
when determining that the request has the authority to use the hardware resources, the access right checker allows the request to be	When the access rights checker in the Accused Products determines that the user request has the authority to use the hardware resources, the access right checker allows the request to be executed. For example, in the Accused Products, if the EL0 access is enabled, a read request is allowed:

<table>
<tr>
<td>executed; and</td>
<td>

C4.4.2 DC CIVAC, Data or unified Cache line Clean and Invalidate by VA to PoC

The DC CIVAC characteristics are:

Purpose

Clean and Invalidate data cache by address to Point of Coherency.

This register is part of the Cache maintenance operations functional group.

Usage constraints

This operation can be performed at the exception levels shown below:

EL0	EL1 (NS)	EL1 (S)	EL2	EL3 (SCR.NS=1)	EL3 (SCR.NS=0)
Config-WO	WO	WO	WO	WO	WO

EL0 access is enabled when SCTLR_EL1.UCI is set to 1. When it is set to 0, this operation is UNDEFINED at EL0.

If EL0 access is enabled, this operation is available at EL0 when the VA has read access permission, otherwise it causes a Permission Fault.

Source: https://yurichev.com/mirrors/ARMv8-A_Architecture_Reference_Manual_(Issue_A.a).pdf

</td>
</tr>
<tr>
<td>when determining that the request does not have the authority to use the hardware resources, the access rights checker responds the request with</td>
<td>When the access rights checker in the Accused Products determines that the user request does not have the authority to use the hardware resources, the access right checker does not allows the request to be executed. For example, in the Accused Products, if the EL0 access is not enabled, the access rights checker responds with a Permission Fault and denies access:</td>
</tr>
</table>

an exception.	**C4.4.2 DC CIVAC, Data or unified Cache line Clean and Invalidate by VA to PoC** The DC CIVAC characteristics are: **Purpose** Clean and Invalidate data cache by address to Point of Coherency. This register is part of the Cache maintenance operations functional group. **Usage constraints** This operation can be performed at the exception levels shown below: EL0 access is enabled when SCTLR_EL1.UCI is set to 1. When it is set to 0, this operation is UNDEFINED at EL0. If EL0 access is enabled, this operation is available at EL0 when the VA has read access permission, otherwise it causes a Permission Fault. Source: https://yurichev.com/mirrors/ARMv8-A_Architecture_Reference_Manual_(Issue_A.a).pdf

EL0	EL1 (NS)	EL1 (S)	EL2	EL3 (SCR.NS=1)	EL3 (SCR.NS=0)
Config-WO	WO	WO	WO	WO	WO

Register access

Configuration settings for AArch64 processors are held in a series of registers known as System registers. The combination of settings in the System registers defines the current processor context. Access to the System registers is controlled by the current Exception level.

For example, VBAR_EL1 is the Vector Base Address Register. We will describe what it is used for later in this guide. For now, what is important is the _EL1 suffix. This tells us that software needs at least EL1 privilege to access the register.

The architecture has many registers with conceptually similar functions that have names that differ only by their Exception level suffix. These are independent, individual registers that have their own encodings in the instruction set and will be implemented separately in hardware.

The registers have similar names to reflect that they perform similar tasks, but they are entirely independent registers with their own access semantics. The suffix of the System register name indicates the lowest Exception level from which that register can be accessed.

There is a System Control Register (SCTLR) for each implemented Exception level. Each register controls the architectural features for that EL, such as the MMU, caches and alignment checking:

SCTLR_EL1

Top-level system control for EL0 and EL1

SCTLR_EL2

Top-level system control for EL2

SCTLR_EL3

Top-level system control for EL3

Feedback

Note

EL1 and EL0 share the same MMU configuration and control is restricted to privileged code running at EL1. Therefore there is no SCTLR_EL0 and all control is from the EL1 accessible register. This model is generally followed for other control registers.

Higher Exception levels have the privilege to access registers that control lower levels. For example, EL2 has the privilege to access SCTLR_EL1 if necessary. This register cannot be accessed from EL0, and any attempt to do so generates an exception.

In the general operation of the system, the privileged Exception levels control their own configuration. However, more privileged levels sometimes access registers associated with lower Exception levels. For example, this could be to implement virtualization features and context switching.

Source: https://developer.arm.com/documentation/102412/0103/Privilege-and-Exception-levels/Types-of-privilege