Review on Firmware

Chu Jay Tan
School of Electrical & Electronic Engineering
Universiti Sains Malaysia
Pulau Pinang, Malaysia.
chujay91@hotmail.com

Junita Mohamad-Saleh
School of Electrical & Electronic Engineering
Universiti Sains Malaysia
Pulau Pinang, Malaysia.
jms@usm.my

Khairu Anuar Mohamed Zain
CEDEC[1]
Universiti Sains Malaysia
Pulau Pinang, Malaysia.
anuar@usm.my

Zulfiqar Ali Abd. Aziz
CEDEC[1]
Universiti Sains Malaysia
Pulau Pinang, Malaysia.
eezulfiq@usm.my

ABSTRACT

This paper presents a review on firmware and the process of firmware development including firmware development model, current trend in firmware development, task scheduling, debugging, documenting the source code and discussed some information related to embedded microcontroller. Firmware was introduced by Ascher Opler in 1967, to solve confusion on defining the software code residing in the hardware Read Only Memory (ROM). Moore's law (1965) is rumored to coming to an end, but undeniable it has been envisioning semiconductor industry for the past 5 decades. Improvement in semiconductor manufacturing technology for integrated chip has at least 10 to 100-folded in term of performance. Embedded device which was then resource-constrained is now capable to run full fledge operating system (O/S). Firmware had been used to define the software code reside in ROM for the Basic Input Output System (BIOS) in our computer motherboard used for initialization of the hardware peripheral. Firmware is also used to define the software code reside in microcontroller and embedded system. Embedded microcontroller is used in various application due to its low cost, small size, reliability, less power consumption and lighter weight. The improvement of technology in term of both hardware and software development had introduce microcontroller like Arduino to allow people to carry out firmware coding relatively ease compared with previously available embedded system. The increased capacity of embedded memory had allowed O/S stack to reside with the embedded system firmware which has blurred the border between firmware and software as how the software code stored in the hardware ROM happened back in 1967. The definition of firmware should be redefined over time to further avoid confusion. Although the borderline between firmware and software is rather blur, undoubtedly firmware is a piece of software with close relation to the hardware within. This paper gives an overview about firmware and provides a clearer view on firmware in personal computer and embedded systems.

ICISPC 2017, July 26–28, 2017, Penang, Malaysia

© 2017 Association for Computing Machinery.
ACM ISBN 978-1-4503-5289-5/17/07…$15.00
DOI: https://doi.org/10.1145/3132300.3132337

CCS Concepts

• Computer systems organization→Embedded and cyber-physical systems→Embedded systems→Embedded software.

Keywords

Firmware, BIOS, Embedded, Microcontroller, Software

1. INTRODUCTION TO FIRMWARE

What is firmware? A simple Google search returned with Android Nougat, iOS 11, BIOS or embedded system. Which one is correctly defined? Before the introduction of the term "firmware", microprogramming is the equivalent word with the same meaning. In 1951, Prof. Maurice Wilkes proposed microprogramming as a way to design a computer in which the control of control register and arithmetic register can be made systematic to perform calculation via order code instead of hardwired means. For further understanding on what is microprogramming, there exists varied definitions [1]. One of the definition is "software in read-only control store that makes one computer (the host machine) response as if it were another computer (the target machines)". Microprograms contain information controlling hardware at primitive level, stored in memory and executed as a program, is a software as well as hardware flavor so given an appellation of "Firmware" [2]. Ascher Opler coined the term firmware in 1967 by saying "I believe it is worthwhile to introduce a new word into our vocabulary "firmware", I use this term to designate microprograms residing in a computer's control memory" [3]. Firmware is a term used to denote software code residing in ROM which is no longer "soft" enough to be modified, but still functionally software, and executable [4]. In [5], Amdahl had reported that, "hardware/software interface problems disappear only to be replaced by a more complex hardware/firmware/software interface". Firmware had since then widened to denote anything ROM-resident including macroinstruction-level routine for Basic Input Output System (BIOS), bootstrap loaders or specialized application [6].

[1] CEDEC : Collaborative Microelectronic Design Excellence Centre

In early history of digital computing, software is considered as pure information. It is recorded on tangible medium such as punch cards or magnetic tapes (hard-drive/solid-state drive equivalent in current terminology). BIOS is a software residing in BIOS chip on the motherboard of Personal Computer (PC). It is first executed during system startup to perform Power-On Self-Test (POST). Reference [7] defined BIOS as a firmware that controls a computer before the O/S begins its execution. It is typically stored in ROM or Programmable Read Only Memory (PROM). Improvement in semiconductor technology has led to the invention of EPROM, EEPROM and flash. Due to cost advantages, some manufacturer has turned to use flash implementation in their system instead of ROM. The main jobs of BIOS include testing of the machine hardware, to boot the O/S from bootable devices, as well as to provide support for interactive hardware and software debugging. To have a better understanding of BIOS, reference [8] is a good source to star reading.

BIOS system was introduced by the International Business Machines Corporation (IBM) in 1982 [9]. In 1952, Compaq reversed engineered the BIOS chip and created fully IBM compatible personal computer[10]. Reference [9], indicated that Intel and Microsoft had planned a major change to BIOS with Extensible Firmware Interface (EFI) to accelerate boot-up process, ease the addition of improvement in peripheral support to PC, cutting manufacturing cost and improve remote management of devices. In 2006, to standardize the interface and simplifies and secure the platform initialization and firmware bootstrap operation, Unified Extensible Firmware Interface Forum (UEFI) initial release had been created [11]. It consists of data tables containing platform-related information, boot service calls and runtime service calls available to loader and O/S to provide a standard environment for running pre-boot application and booting operation. The current BIOS has to initialize system management function such as power and thermal control through Advanced Configuration and Power Interface (ACPI).

1.1 Firmware Versus Operating System (O/S)

BIOS is a glue between computer hardware and O/S. Right after system startup, BIOS is the first piece of code that will be executed to initialize and test all the hardware peripheral on the motherboard before passing the control of hardware to the O/S [7]. O/S is an intermediate program interfacing a computer user and computer hardware as illustrated in Figure 1. It eases the usage of a computer by allowing user to execute application program on top of it. Application program like compiler, video game, and other programs can run on top of O/S without users acknowledge about the hardware underneath since all the hardware resource will be controlled by the O/S. In short, an O/S manages and allocates usage of hardware resources available to the application program, and controls the execution of program and the operation of I/O devices [12]. Kernel is a program that runs all the time and it is part of the O/S itself. A good O/S should be able to utilize the hardware resources efficiently to perform and execute user application program. Moreover, an O/S need to have a security support utility to ensure that unwanted applications such as viruses or WannaCry Ransomware will not be executed without user intention. Firmware resides in BIOS chip on a PC motherboard contains microcode (opcode + operand) for performing all the initialization sequence, peripheral drivers and test vector. An O/S on the other hand, is normally installed in hard drive or solid state drive and is booted after BIOS passes the control of the hardware to the O/S. An application running on an O/S can be easily installed or uninstalled, but BIOS firmware is normally not meant to be easily changed by end-user. As can be seen in Figure 1, firmware provides a layer of interface between hardware and the O/S. Figure 2 shows that firmware is a piece of software, on top of the hardware backbone, providing a means of signal, control and communication for data exchange between the CPU (MCU) and peripheral (IPs).

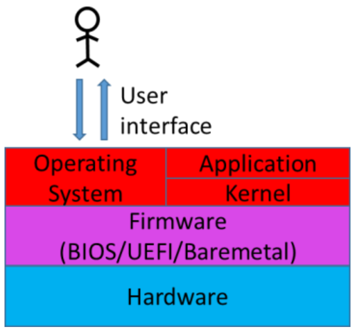

Figure 1 Structure levels of hardware, firmware and operating system

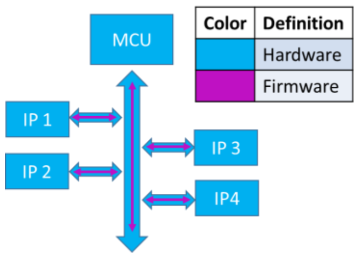

Figure 2 An illustration of hardware and firmware relation

1.2 Without Firmware What Happens?

The easiest explanation for the absence of firmware would be a computer system not able to boot up. Without BIOS, hardware will be unable to initialize in a proper way and an O/S will not be able to boot up due to missing interface between the hardware and O/S. Hence, without firmware, an O/S will not be as portable as it is. It would be an exception if a custom coded startup sequence is present with the booting O/S. Every processor has different implementation, startup sequence and interrupt table, thus O/S need to be custom coded, compiled and linked for proper memory map to suit different sets of hardware configurations. With current availability of vast variant of computer hardware peripheral, it will cause a big issue regarding total effort needed in customizing the O/S especially for computer industry just to ensure a proper boot-up of an O/S. For the microcontroller, without firmware, it would be just a piece of silicon hardware performing nothing when powered up as there is no instruction to be executed to perform a task.

1.3 Firmware in Embedded Microcontroller

Microcontroller is defined by [13] as single integrated circuit consisting basic element like Central Processing Unit (CPU), Clock oscillator, Memory and Input/ Output (I/O) port. It can be considered as one type of embedded system based on definition in [14] where embedded is defined as control of a device by a completely encapsulated special-purpose computer performing pre-defined task with very specific requirement. Microcontroller consists of two major parts which are hardware and software respectively [7]. Hardware is tangible, while software is an intangible list of instructions residing in the hardware memory.

Software residing in an embedded system is called firmware and its definition is currently still debatable. IEEE standard Std. 12207-2008 defines firmware as “a combination of hardware device and computer instructions or computer data that resides as a read-only software on the hardware device.” [15]. This definition is challenged by microcontroller supporting in-field firmware update ability [16] where firmware update can be performed under Software Control.

The breach of security in the Stuxnet case, and the CIH virus ability to target BIOS firmware [17], show that our current system is not secured. With the rise of Internet of Thing (IOT) using embedded system and current predicted IOT devices to be around 6.4 to 9 billion and increasing [18], it raise concern regarding security in embedded devices, which has an impact on firmware implementation.

2. FIRMWARE DEVELOPMENT

Reference [1] discussed firmware development as an analogy to software development, and suggested firmware development at around 1978 to be considered proprietary that many techniques had never been published. Earlier computers are programmed in binary followed by development of machine-dependent assembler to increase productivity in terms of programming. The creation of machine-independent Higher-Level Language (HLL) has ease both the software and firmware development. The term “Software Engineering” has been used for the software design method, specification languages, verification method and etc. Borrowing Boehm’s definition, [1] defined firmware engineering as “the practical application of scientific knowledge in the design and construction of micro-program and the associated documentation required to develop, operate, and maintain them”.

Previous firmware development was often divided into 4 main parts which are : (1) Design and specification; (2) construction / coding; (3) verification, testing and debugging; and (4) maintenance. In [19], the author defined a good programming practice and suggested that code should be conform with specification and standard, to ensure ease of use, and having systematic stages namely: (1) Specification; (2) Planning; (3) Coding; (4) Debugging; (5) Testing; and (6) Documentation. These steps have been followed for firmware development.

2.1 Specification

BIOS standard and UEFI Special Interest Group have defined standard to be followed as well as chain of trust and root of trust to ensure the security and the peripheral driver is running as intended by the manufacturer. Device driver in BIOS firmware development for the PC hardware initialization will be very hard to be realized by normal user. To comply with all the standard, user will need to study and understand all the documentation as well as hardware specification. The UEFI Platform Initialization (PI) Specification, consists of 5 volumes and for only volume 1, a total of 1627 pages is present [20]. To ease the discussion, this paper limits the firmware development discussion within the microcontroller. The first thing to do when developing firmware is to list down the specifications, objectives and requirements of what a microcontroller needs to accomplish. As a general ideal, an embedded firmware specifications need to be timeliness (real-time behavior), deterministic, predictability, secure, reliability and reusable [21]. Embedded microcontroller is used in various applications, there is some standard available to be adhere to especially when it is used in safety critical application like automotive Engine Control Unit or Collision detection system for airbag deploy. Coding standard like ISO 26262 [22] or MISRA C can be followed to ensure the quality and functionality of the firmware in term of safety. In addition, the standard use for programming language need to be decided during specification phase since there are many languages available and for example with just C programming language, various standard like C89, C11 and C14 are listed in the ISO/IEC standard.

2.2 Planning

Embedded software design had changed drastically since the introduction of the embedded microcontroller. Shorter time to market and better embedded hardware with more peripherals are pushing the complexity of firmware development. Current trend in embedded microcontroller usage in Internet of Thing (IOT) is pushing further the software complexity with the communication protocol stack, database involved and system security [23]. Currently, Real Time Operating System (RTOS) is used in some embedded microcontroller for task scheduling and different firmware thread is run in time sliced sequence. Traditional way of task scheduling is done by bare metal looping scheduling or a scheduling scheme is developed in house by the programmer. With the use of RTOS scheduling, different thread execution might change the register value of embedded unit and to prevent it, it may slow down and make the firmware complex [24]. This problem needs proper planning when distributing the task between the RTOS scheduler as well as hardware and software engineering team. Besides, software design process is needed to ensure that the software can be completed on-time.

2.2.1 Software design model [25]

In 1970, Waterfall model was introduced by Dr. Win Royce at Lockheed to graphically represent establishing requirement for software development in cascaded form. It is a great step forward in software development and might work satisfactorily if design requirement could be perfectly addressed. But it is almost impossible to guarantee program correctness with more than 169 lines of code by any process as rigorous as mathematical proof. For comparison, current smart phone might require more than million lines of code.

Agile is a method for quick software development in an environment with fast changing of requirement. Close collaboration with customer is encouraged to accept software requirement changes with frequent release to customer for continuous design improvement. eXtreme Programming is an agile software development model with high flexibility that puts client in the driver’s seat and it works well when the client’s requirement is continually changing. Agile had been used in software and hardware development project in [26, 27].

2.3 Coding

The increasing in software complexity and shorter time to market have pushed embedded software development to shift left by having hardware/firmware co-development [28]. It is achievable using simulation, emulation, Field Programmable Gate Array or Virtual Platform but additional effort is needed to realize the co-development process. The use of different method in the project co-development will introduce different debugging complexity between hardware and software. Increasing complexity in firmware and decreased time to market had promoted code reuse in firmware industry. Properly designed Low-level Hardware abstraction layer (LHAL) can benefit developer with portable and reusable code, abstraction, lower cost and fewer bug. C programming language is current most used programming language for embedded software development.

2.3.1 Task scheduling

Software execution on the hardware need CPU resources to allow processor to carry out operation for completing a given task. Most common task scheduling in embedded microcontroller development are realized using bare metal scheduler or RTOS. Bare Metal scheduling mean running the software on top of the embedded microcontroller without a layer of O/S in between. The advantages of bare metal are that no resources is used in the execution of the OS itself. Bare metal scheduling can be realize in 3 different methods which are round robin, round robin with interrupts or develop a own bare-metal scheduler. RTOS offer firmware developer to create semi-independent program that can run concurrently on microcontroller and synchronizing task execution. It also eases process scheduling, system integration and allowing tasks switching based on event and priority given [29]. Moreover, RTOS have additional feature like mutexes to protect shared resources, semaphores to signal and synchronize task as well as message queues to transfer data among task. Improperly used of RTOS can cause tragedy like priority inversion may cause system lock-out and the additional feature in RTOS will introduce some learning curve in the firmware development.

2.4 Debugging, Testing and Validation

Previously, debugging was done using assembler and hardware simulator [30]. In the current day, most debugging can be done by using Integrated Development Environment (IDE). Current tool like Keil µVison can debug the firmware code, support hardware debugger like JTAG debugger, Serial Wire Debug as well as software simulation using the build-in simulator to simulate read/write access to memory in the ARM Cortex-M microcontroller. Since the debug adapter is supported by the IDE, it can be used to download compiled firmware into the development board available from the vendor. The debug adapter allow programmer to carry out hardware debug and testing. For the Hardware in Loop testing [31], the microcontroller is loaded with firmware and tested in the real environment to check for its performance in real life. It would then be tested against it specification to ensure that all the requirement for the firmware development is met including the functional operation, timing and system response to stimuli. Besides that, Open On Chip Debug (OpenOCD) [32], an open source debugging tool is available for use in debugging with features like execution control, breakpoints, data watchpoint, code single-stepping and fault detection.

2.5 Documentation

To ensure the software reuse, proper source code commenting is a must. Doxygen can be used to generate the technical documentation needed from the source commenting [33]. Software version control should be applied to ensure that the desired piece of code is used when various code revision reside in the coding repository. From Agile software development model, project documentation simply need to be good enough and should be documented with purpose while waterfall model is the opposite where it emphasis on documentation of requirement, design and etc. With the pressure of shorter-time to market and more complex hardware and software development, too much emphasis on the documentation may burden engineer with their work and documenting task.

3. FIRMWARE AFFECTING OVERALL SYSTEM PERFORMANCE

As discussed in [2], microprogramming can improve the execution speed compared with software implementation. A well-designed firmware algorithm can speed up the booting and initialization process of the hardware peripheral, which in turn reduce the boot time of the O/S. Since part of the BIOS firmware provides peripheral driver for O/S to access the hardware, all the driver coding should be fully optimized to ensure all driver function can be executed in shortest time. Memory chips used in storing the BIOS chip have their rated read and write speed [34]. If a low read speed BIOS chip is used to store the BIOS firmware program, it will limiting the speed of fetch-cycle during system startup and slow down the startup speed of a PC system.

4. NEW TREND

Recent research trends in the field of computer firmware development include but not limited to security and digital forensic due to issue like Stuxnet [35] and the CIH virus damaging 60 million personal computer in 1998 [17]. BIOS hardware design can be revised to prevent changes made to the BIOS firmware by virus or rootkit. Since BIOS firmware is stored inside the BIOS chip [36], implementation of one hardware switch to enable or disable write access to the BIOS chip can be done. Since the firmware in the BIOS chip is seldom updated in normal use case, this will not give a huge impact to the overall functionality of the hardware/firmware/software interface. To update BIOS firmware, user can refer to user guide to enable write to BIOS chip. A software mechanism can then be used to detect the write enable switch status and alert user if the write enable is enabled to prevent rootkit or virus in altering the BIOS firmware at all. In [37], the author proposed a novel speed-up coding method in flash memory. Perhaps, the coding for the BIOS firmware can utilize this coding technique to reduce time delay in the verification of lockout operation. This would in-turn speed up the boot up time of the BIOS firmware. Moreover, with the rise of Artificial Intelligence (AI), it might not be far to see the application of the AI in controlling the power delivery system using the ACPI interface. For example, advancement in semiconductor technology had bring down the power consumption, cost and silicon area of microcontroller, it is now possible to implement Artificial Neural Network (ANN) using microcontroller to learn the power usage of the computer user and optimize the power delivery dynamically. In addition, the microcontroller can be used to perform the CRC calculation of the BIOS firmware to enable just in time verification without the main processor interference.

5. ACKNOWLEDGEMENTS

The authors would like to express special thanks to TalentCorp Sdn. Bhd., CEDEC and School of Electrical and Electronic Engineering, Universiti Sains Malaysia for the 3D Program to allow the study and preparation of this firmware review paper.

6. REFERENCES

[1] Davidson, S. and Shriver, B. D. 1978. An Overview of Firmware Engineering. *Computer*, 11 (5). 21-33.

[2] Rauscher, T. G. and Adams, P. M. 1980. Microprogramming: A Tutorial and Survey of Recent Developments. *IEEE Transactions on Computers*, C-29 (1). 2-20.

[3] Opler, A. 1981. Fourth-generation software. *Datamation*, 13 (1). 22-24.

[4] Reilly, E. D. 2003. *Milestones in Computer Science and Information Technology*. Greenwood Press. Westport, Conn.

[5] G. M. and Amdahl, L. D 1967. Fourth Generation Hardware *Datamation*, Cahners Publishing Company.

[6] Smotherman, M. 2000. Microprogramming and Microarchitecture. Kent, A. and Williams, H.G. eds. *Encyclopedia of Microcomputers*, Marcel Dekker, 258.

[7] IEEE. 1994. IEEE Standard for Boot (Initialization Configuration) Firmware: Core Requirements and Practices. *IEEE Std 1275-1994*. 1-262.

[8] Ortiz, S. 2003. Enter the BIOS *Maximum PC*, Future US, inc, 34-38.

[9] Paulson, L. D. 2004. New technology beefs up BIOS. *Computer*, 37 (5). 22.

[10] Skrabec, Q.R. 2012. *The 100 Most Significant Events in American Business: An Encyclopedia*. Greenwood.

[11] UEFI SIG. About UEFI Forum, Assessed Online - June 2017 http://www.uefi.org/about.

[12] Silberschatz, A., Galvin, P. B., Gagne, G., and Silberschatz, A.1998. *Operating system concepts*. Addison-wesley Reading.

[13] Bannaty ne, R. and Viot, G. 1998. Introduction to microcontrollers. I. in *Wescon/98. Conference Proceedings (Cat. No.98CH36265)*, 350-360.

[14] Kenaz, T., GarriI, K., and Aissani, M. 2016. A study law level Embedded System attacks. in *The 2nd Conference on Computing System and Applications*, Ecole Militaire Polytechnique, Algiers, Algeria, 168.

[15] IEEE. 2008. ISO/IEC/IEEE Standard for Systems and Software Engineering - Software Life Cycle Processes *IEEE Std 12207-2008*, c1-138.

[16] Kim, J. and Chou, P. H. 2009. Remote progressive firmware update for flash-based networked embedded systems *Proceedings of the 2009 ACM/IEEE international symposium on Low power electronics and design*, ACM, San Fancisco, CA, USA, 407-412.

[17] Gratzer, V. and Naccache, D. 2007. Alien vs. Quine. *IEEE Security & Privacy*, 5 (2). 26-31.

[18] Nordrum, A. 2016. Popular Internet of Things Forecast of 50 Billion Devices by 2020 Is Outdated, Assessed Online - June 2017 http://spectrum.ieee.org/tech-talk/telecom/internet/popular-internet-of-things-forecast-of-50-billion-devices-by-2020-is-outdated.

[19] Mills, J. 1980. Good programming in assembly language. *IEE Proceedings E - Computers and Digital Techniques*, 127 (6). 241-248.

[20] UEFI. 2017. Platform Initialization (PI) Specification *Vol 1: Pre-EFI initialization Core Interface*, UEFI.

[21] Sun, J., Jones, M., Reinauer, S., and Zimmer, V. 2015. Firmware Stacks for Embedded Systems. in *Embedded Firmware Solutions: Development Best Practices for the Internet of Things*, Apress, Berkeley, CA, 13-24.

[22] Kleeberger, V. B., Rutkowski, S. and Coppens, R. 2015. Design &amp; verification of automotive SoC firmware. in *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, 1-6.

[23] The Eclipse Foundation. 2016. Eclipse IoT White Paper - The Three Software Stacks Required for IoT Architectures, Assessed Online - June 2017 https://iot.eclipse.org/white-paper-iot-architectures.

[24] Fechser, D. 2017. Making hardware modules firmware friendly, Assessed Online - June 2017 http://www.eetimes.com/document.asp?doc_id=1277091&print=yes.

[25] Jayaswal, B. K. and Patton, P.C. 2006. *Design for Trustworthy Software: Tools, Techniques, and Methodology of Developing Robust Software*. Pearson Education.

[26] Greene, B. 2004. Agile methods applied to embedded firmware development. in *Agile Development Conference*, 71-77.

[27] Lee, Y., Waterman, A., Cook, H, Zimmer, B., Keller, B., Puggelli, A., Kwak, J., Jevtic, R., Bailey, S., Blagojevic, M., Chiu, P. F., Avizienis, R., Richards, B., Bachrach, J., Patterson, D., Alon, E., Nikolic, B., and Asanovic, K. 2016. An Agile Approach to Building RISC-V Microprocessors. *IEEE Micro*, 36 (2). 8-20.

[28] Abarbanel, Y., Singerman, E., and Vardi, M. Y. 2014. Validation of SoC firmware-hardware flows: Challenges and solution directions. in *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*, 1-4.

[29] Beningo Embedded Group. 2017. Tips and Tricks – 7 Reasons to Choose an RTOS, Assessed Online - June 2017 http://beningo.com/tips-and-tricks-7-reasons-to-choose-an-rtos/.

[30] Texas Instruments Incorporated. Semiconductor Group. 1975. *TMS 1000 Series MOS/LSI One-chip Microcomputers: Programmer's Reference Manual*. Texas Instruments.

[31] Gruszczynski, B. 2006. An overview of the current state of software engineering in embedded automotive electronics. in *2006 IEEE International Conference on Electro/Information Technology*, 377-381.

[32] Pomante, L., De Cesaris, I., and Dell'Osa, S. 2015. OATS: An automated test system for OpenOCD. in *2015 IEEE 1st International Forum on Research and Technologies for Society and Industry Leveraging a better tomorrow (RTSI)*, 502-507.

[33] Dimitri van Heesch. 2016. Doxygen : Generate documentation from source code, Assessed Online - June 2017 http://www.stack.nl/~dimitri/doxygen/index.html.

[34] Tal, A. 2002. Two Flash Technologies Compared: NOR vs NAND, Assessed Online - June 2017 https://focus.ti.com/pdfs/omap/diskonchipvsnor.pdf.

[35] Kushner, D. 2013. The real story of stuxnet. *IEEE Spectrum*, 50 (3). 48-53.

[36] Gershteyn, P., Davis, M., and Shenoi, S. 2006. Forensic Analysis of BIOS Chips. in Olivier, M.S. and Shenoi, S. eds. *Advances in Digital Forensics II: IFIP international Conference on Digital Forensics, National Center for Forensic Science, Orlando, Florida, January 29– February 1, 2006*, Springer US, Boston, MA, 301-314.

[37] Jin, X., Xiao, H., Wu, D., Deng, N., Wu, H., Cao, K., and Qian, H. 2016. A novel speed-up coding method in quadruple-level-cell 3D NAND flash memory. in *2016 5th International Symposium on Next-Generation Electronics (ISNE)*, 1-2.