Failure Analysis in Embedded Systems

Joshua Main-Smith

**Table of Contents**

1. Common Types of Failure in Computer Science 3

Software 3

Hardware 3

2. Failure Analysis 5

Preventative 5

Forensic 6

3. References 7

**1. Common Types of Failure in Computer Science**

**Software**

There are many ways for a computer to fail, with the most common failures being software related. This is probably due to there being a multitude of different ways in writing a successful program, albeit with bugs under conditions that the software engineer nor the quality assurance engineer had considered. Failure in software that most people experience tends to be related to an operating system issue. There are a multitude of ways in which the operating system can crash a computer system, generating the infamous blue screen of death (BSOD). Although seeing a BSOD with todays operating systems aren’t nearly as bad as a few decades ago, they can still provide a headache for many people. When the operating system fails, it’s usually due to a programmable issue (how the OS was written), an incompatibility issue (an application or driver added to the system which is affecting the OS in a detrimental fashion), or when the OS is subject to excessive heat[1]. The reason why an OS would be responsible for so many failures leads back to several factors during the development process, such as a lack of communication between different departments in implementation, using newer technologies without proper quality assurance testing, and deploying the software early (again, without proper quality assurance testing) [2][5].

**Hardware**

Hardware, on the other hand, leaves less room for creativity in its design. This is a necessary compromise to establish a standard so that software knows how to communicate with the hardware. Even so, there are many ways in which hardware can fail as well. Some examples of hardware failure include components such as power supply, random access memory (RAM), the central processing unit (CPU), integrated fans, disk drives, and much more [1]. Most of these issues stem from some environmental factor, most notably heat. If the RAM, CPU, or disk drives were to receive excessive heat it could potentially cause permanent damage. This is especially true for disk drives if the operating system is located on the drive, leading to the consumer being forced to buy a new drive with an operating system installed and starting from scratch if the files located on the drive weren’t saved anywhere else. [1][3]

Further, there’s another area of special interest to a wide array of companies and agencies that is concerned with both the hardware and software aspects of system failure: embedded systems. Embedded systems are the interface between hardware and software providing a possibility for the hardware to communicate with the software, and vice versa. Due to the very nature of how embedded systems works, there are a plethora of potential problems with the system, making diagnosis difficult. This is an important field due to its use in everything from television sets to financial systems to medical device. Figure 1 below shows where embedded systems fits in the development process (T. Noergaard, 2013, pp. 5).

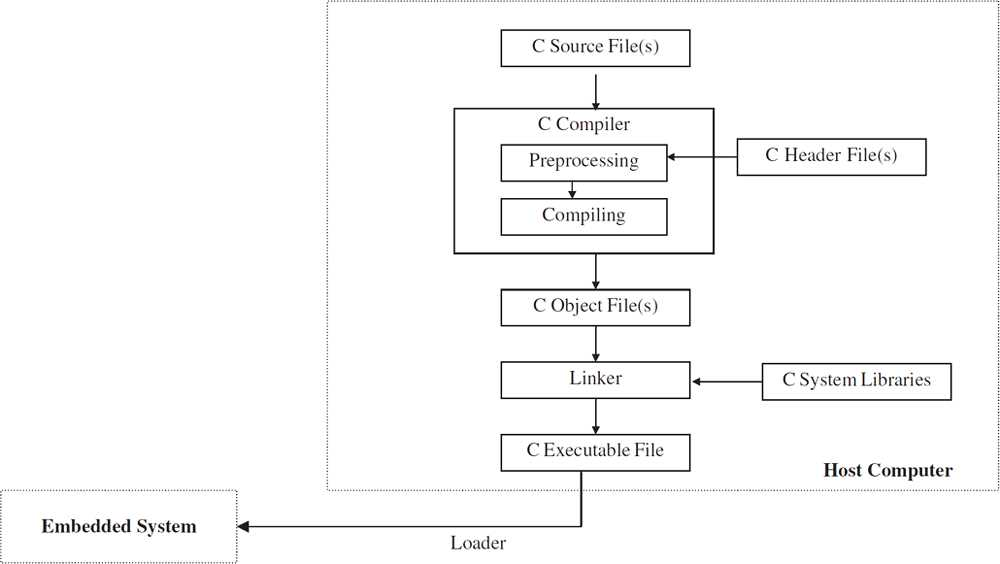


Figure 1. *Embedded Systems in the Development Process*

The following will be a discussion on a few failure analysis methods in testing for performance.

**2. Failure Analysis Methods**

**Preventative Methods**

One method in testing for and preventing structural damage to SRAM cells is to subject the cells to destructive read failures. This is essentially the task from the processor switching in the middle of a read function, thereby corrupting the data. The result from this can potentially be bad, destroying all memory files.

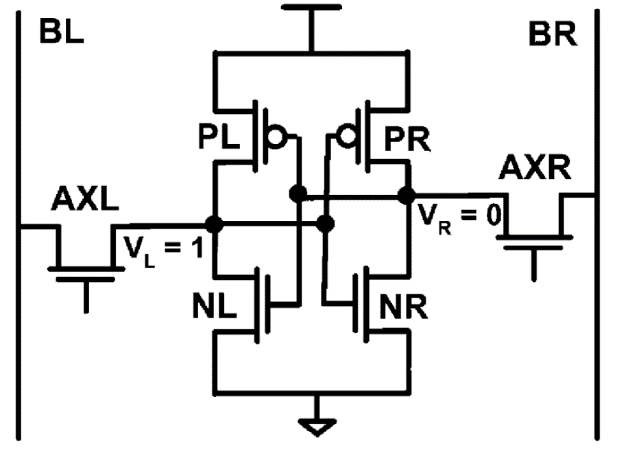


Figure 2. *SRAM Voltage Cell.*

To test SRAM cells for this flaw, researchers create an artificial flip in the SRAM cell. So, the voltage (VR) in Figure 2 (A. Agarwal et al, 2005) increases (becomes relatively positive) while performing a read operation. Then, VR can be increased enough for it to be more than the voltage potential between PL and NL, causing a read failure. This checks for the durability of the chip [3][4].

**Forensic Methods**

As a non-destructive method in evaluating an embedded system, there are a few tools an engineer can use while testing for hardware and software issues. For the hardware aspect, an engineer can use an in-circuit emulator (ICE) which is a debugging solution that acts as a replacement for the microprocessor and allows for break points, modifiability and other similar features that most debuggers have, which brings me to the software component. Engineers use debuggers when checking for software issues, usually included with an integrated-development environment (IDE). It allows the engineer to step through each line of code, enter functions, check the memory on the stack memory, and more.

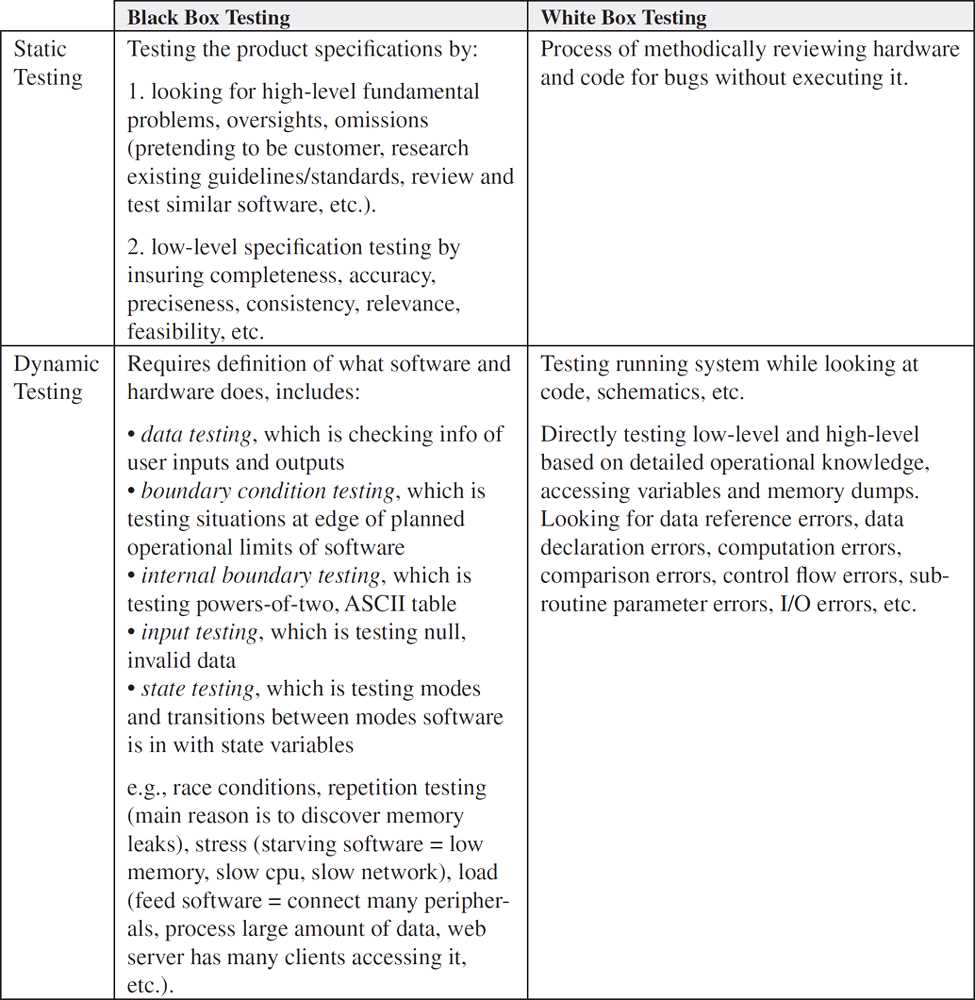


Figure 3. *Black Box and White Box Testing Methods.*

The strategy in checking for issues can be seen in Figure 3 above (T. Noergaard, 2013, pp. 18). The black box testing is used in checking for high level issues, such as completeness of files, code, readability. White box testing is looking for specific issues, or a low-level approach (for examples, there’s an issue with a function so an engineer will step into the function and debug line-by-line) [6]

**3. Resources**

[1] C. Shields, “Why Do Computers Crash?,” 2003.

<https://www.scientificamerican.com/article/why-do-computers-crash/>

[2] S. Dalal and R. Chhillar, “Case Studies of Most Common and Severe Types of Software System failure,” Int. Jou. Com. Sci & Sof. Eng. vol. 2, no. 8, pp. 341-346, Aug. 2012.

<http://ijarcsse.com/Before_August_2017/docs/papers/8_August2012/Volume_2_issue_8/V2I800209.pdf>

[3] A. Agarwal et al, “Process Variation in Embedded Memories: Failure Analysis and Variation Aware Architecture,” Sol. Sta. Cir. vol. 40, no. 9, pp. 1804-1813, Sep. 2005.

<https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1501978>

[4] J. Kloos, T. Hussain, and R. Eschbach, “Risk-based Testing of Safety-Critical Embedded Systems Driven by Fault Tree Analysis,” presented at Int. Con. On Software Testing, Kaiserslautern, German. 2011.

<https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5954386&tag=1>

[5] Leveson, Nancy G. "Software safety in embedded computer systems." *Communications of the ACM*, Feb. 1991, p. 34+. *Academic OneFile*, https://link.galegroup.com/apps/doc/A10371104/AONE?u=googlescholar&sid=AONE&xid=c964c939. Accessed 25 Feb. 2019.

<https://go.galegroup.com/ps/i.do?p=AONE&sw=w&u=googlescholar&v=2.1&it=r&id=GALE%7CA10371104&sid=googleScholar&asid=fa52d829>

[6] T. Noergaard, “The Final Phases of Embedded Design-Implementation and Testing” in *Embedded Systems Architecture: A Comprehensive Guide for Engineers and Programmers, 2*nd ed. Waltham, MA, USA: Newnes, 2013, ch. 12, sec. 1 & 2, pp. 5-19.

<https://ufl.skillport.com/skillportfe/main.action?assetid=RW$11672:_ss_book:51067#summary/BOOKS/RW$11672:_ss_book:51067>