Review Article

iMedPub Journals
http://www.imedpub.com

American Journal of Computer Science and Information Technology

2021
Vol. 9 No. 9: 109

Scaling Data Intensive Testing of Embedded System with a Slight Look at Mobile Devices

Ayemowa Matthew[1*], Ajayi W[2], Emmanuel Adediran[3], Iyanuoluwa Fatoki[4] and Alonge Opeyemi[5]

Department of Computer Science, Lead City University, Faculty of Basic Medical Sciences, Toll Gate, Ibadan, Nigeria

Corresponding author:
Ayemowa Matthew, Department of Computer Science, Lead City University, Faculty of Basic Medical Sciences, Toll Gate, Ibadan, Nigeria

E-mail: ayemowaodunayo@gmail.com

Citation: Matthew A, Ajayi W, Adediran E, Fatoki I, Opeyemi A (2021) Scaling Data Intensive Testing of Embedded System with a Slight Look at Mobile Devices. Am J Compt Sci Inform Technol Vol.9 No.9:109.

Abstract

Embedded systems have a highly increased penetration around the globe. Nearly every 1 out of 5 persons deal with embedded systems not knowing. Innovations are increasingly triggered by software embedded in automotive, transportation, medical-equipment, communication, energy, and many other types of systems. To test embedded software in an effective and efficient manner, a large number of test techniques, approaches, tools and frameworks have been proposed by both practitioners and researchers in the last several decades . This paper is a research made on real time embedded system testing. This paper explains what is meant by testing, embedded systems, faults in embedded systems, testing of the embedded systems and an example of one of the embedded system- mobile embedded applications. A little study is done on Mobile Embedded Systems as well as its challenges, Future works on how Embedded Systems can still work not just with Mobile phones now, but with other computer gadgets. This study is a brief and concise one, not going into details about each segment of an embedded system.

Keywords: Software testing; embedded systems; embedded software; mobile embedded systems; systematic mapping

Received: August 30, 2021; **Accepted:** September 13, 2021; **Published:** September 20, 2021

Introduction

Embedded Systems are ubiquitous. They appear in cell phones, microwave ovens, refrigerators, automobiles and a veritable array of consumer products. Some of these embedded systems have potentially safety or security critical consequences. Embedded systems exist in contexts where failure can be profound. Consider fly by wire, chemical factories, nuclear power plants and even offshore oil wells. Though embedded systems are already ubiquitous, the adoption trajectory of the technology continues unabated. Similar observations exist within the military realm. From smart sensors, software fuses to the evolution of the battle space to being network centric.

Software Testing is a process which is carried out to verify the performance features of a Software product whether it matches the expected requirement it was designed for, and to ensure it is free from failures, bugs or defects. It involves various testing components such as the Manual Testing and the automate testing. Our study shall focus on Testing with automated tools to evaluate one or more properties of interest. Real Time Most, if not all, embedded systems are "real-time". The terms "real-time" and "embedded" are often used interchangeably. A real-time system is one in which the correctness of a computation not only depends on its logical correctness, but also on the time at which the result is produced.

Embedded Systems are electronically controlled systems where both the hardware and the software of a computer device are combined together to perform a specific task. Embedded Systems are used also in both large and small systems. Embedded systems basically consists of the hardware and application software and also for larger devices, it has or makes use of Real Time Operating System RTOS that supervises the application software as well as providing mechanism allowing the processor runs a process per schedule by following a stipulated guideline. RTOS defines the way the system works. It sets the rules during the execution of application program. When talking about the small scale systems of embedded systems, it may not have RTOS. Hence, an embedded system is a Microcontroller based, software driven, reliable, real-time control system that performs a designed task [1].

Objective

In this paper, our objective is to enable the black-box, automated testing of RTES based on environment models. More precisely, our goal is to make such environment modeling as easy as possible, and allow the testers to automate test case and oracle generation without any knowledge about the internal design of the RTES.

While embedded systems are computing systems, they can range from having no User Interface (UI)-for example, on devices designed to perform a single task to complex Graphical User Interfaces (GUIs), such as in mobile devices. User interfaces can include buttons, LEDs (Light Emitting Diodes) and touchscreen sensing. Some systems use remote user interfaces as well.

Literature Review

Embedded systems

Embedded systems are used to control a wide variety of dynamic and complex applications, ranging from non-safety-critical systems such as cellular phones, media players, and televisions, to safety-critical systems such as automobiles, airplanes, and medical devices. Embedded systems are also being produced at an unprecedented rate, with over four billion units shipped in 2006. An embedded system is a microcontroller or microprocessor based system which is designed to perform a specific task. For example, a fire alarm is an embedded system; it will sense only smoke [2].

Systems components of embedded systems

Embedded systems vary in complexity but, generally, consist of three main elements:

Hardware: The hardware of embedded systems is based around microprocessors and microcontrollers. Microprocessors are very similar to microcontrollers and, typically, refer to a CPU (central processing unit) that is integrated with other basic computing components such as memory chips and digital signal processors (DSPs). Microcontrollers have those components built into one chip.

Software and firmware: Software for embedded systems can vary in complexity. However, industrial-grade microcontrollers and embedded IoT systems usually run very simple software that requires little memory.

Real-time operating system: These are not always included in embedded systems, especially smaller-scale systems. RTOS (es) define how the system works by supervising the software and setting rules during program execution. RTOS supervises the application software and provide mechanism to let the processor run a process as per scheduling by following a plan to control the latencies. RTOS defines the way the system works. It sets the rules during the execution of application program. A small scale embedded system may not have RTOS [3].

So we can define an embedded system as a Microcontroller based, software driven, and reliable, real-time control system.

Structure of embedded systems

Basic structure of an embedded system **(Figure 1)**.

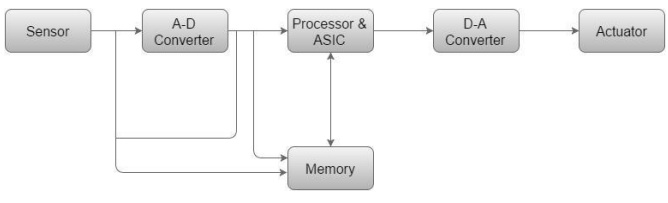

Figure 1: The basic structure of an embedded system.

Sensor: It measures the physical quantity and converts it to an electrical signal which can be read by an observer or by any electronic instrument like an A2D converter. A sensor stores the measured quantity to the memory.

A-D Converter: An analog-to-digital converter converts the analog signal sent by the sensor into a digital signal.

Processor and ASICs: Processors process the data to measure the output and store it to the memory.

D-A Converter: A digital-to-analog converter converts the digital data fed by the processor to analog data

Actuator: An actuator compares the output given by the D-A Converter to the actual (expected) output stored in it and stores the approved output [4].

Types of embedded systems

There are a few basic embedded system types, which differ in their functional requirements. They are:

Mobile embedded systems: A small-sized systems that are designed to be portable. Digital cameras are an example of this.

Networked embedded systems: They are connected to a network to provide output to other systems. Examples include home security systems and point of sale (POS) systems [5].

Standalone embedded systems: They dont reliant on a host system. Like any embedded system, they perform a specialized task. However, they do not necessarily belong to a host system, unlike other embedded systems. A calculator or MP3 player is an example of this.

Real-time embedded systems: They give the required output in a defined time interval. They are often used in medical, industrial and military sectors because they are responsible for time-critical tasks. A traffic control system is an example of this.

Embedded systems can also be categorized by their performance requirements:

Small-scale embedded systems: They often use no more than an 8-bit microcontroller.

Medium-scale embedded systems: They use a larger microcontroller (16-32 bit) and often link microcontrollers together [6].

Sophisticated-scale embedded systems: They often use several algorithms that result in software and hardware complexities and may require more complex software, a configurable processor and/or a programmable logic array.

There are several common embedded system software architectures, which become necessary as embedded systems grow and become more complex in scale.

Simple control loops call subroutines: They manage a specific part of the hardware or embedded programming.

Interrupt controlled systems: They have two loops: a main one and a secondary one. Interruptions in the loops trigger tasks.

Cooperative multitasking: This is essentially a simple control loop located in an application programming interface (API).

Preemptive multitasking or multithreading: This is often used with an RTOS and features synchronization and task switching strategies.

Functional types of embedded system

Single-functioned: An embedded system usually performs a specialized operation and does the same repeatedly. For example: A pager always functions as a pager [7].

Tightly constrained: All computing systems have constraints on design metrics, but those on an embedded system can be especially tight. Design metrics is a measure of an implementation's features such as its cost, size, power, and performance. It must be of a size to fit on a single chip, must perform fast enough to process data in real time and consume minimum power to extend battery life.

Reactive and real time: Many embedded systems must continually react to changes in the system's environment and must compute certain results in real time without any delay. Consider an example of a car cruise controller; it continually monitors and reacts to speed and brake sensors. It must compute acceleration or de-accelerations repeatedly within a limited time; a delayed computation can result in failure to control of the car.

Microprocessors based: It must be microprocessor or microcontroller based [8].

Memory: It must have a memory, as its software usually embeds in ROM. It does not need any secondary memories in the computer.

Connected It must have connected peripherals to connect input and output devices.

HW-SW systems: Software is used for more features and flexibility. Hardware is used for performance and security.

Examples of embedded systems

Embedded systems are used in a wide range of technologies across an array of industries. Some examples include:

Automobiles: Modern cars commonly consist of many computers (sometimes as many as 100), or embedded systems, designed to perform different tasks within the vehicle. Some of these systems perform basic utility functions and others provide entertainment or user-facing functions. Some embedded systems in consumer vehicles include cruise control, backup sensors, suspension control, navigation systems and airbag systems.

Mobile phones: These consist of many embedded systems, including GUI software and hardware, operating systems (Oases), cameras, microphones, and USB (Universal Serial Bus) I/O (input/output) modules [9].

Industrial machines: They can contain embedded systems, like sensors, and can be embedded systems themselves. Industrial machines often have embedded automation systems that perform specific monitoring and control functions.

Medical equipment: These may contain embedded systems like sensors and control mechanisms. Medical equipment, such as industrial machines, also must be very user-friendly so that human health isn't jeopardized by preventable machine mistakes. This means they'll often include a more complex OS and GUI designed for an appropriate UI.

Characteristics of embedded systems

The main characteristic of embedded systems is that they are task-specific

Additionally, embedded systems can include the following characteristics:

- It consists of hardware, software and firmware.
- It can be embedded in a larger system to perform a specific function, as they are built for specialized tasks within the system, but not various tasks.
- It can be either microprocessor-based or microcontroller-based -- both are integrated circuits that give the system compute power.
- Embedded systems are often used for sensing and real-time computing in internet of things (IoT) devices, which are devices that are internet-connected and do not require a user to operate.
- It can as well vary in complexity and in function, which affects the type of software, firmware and hardware they use.
- They are often required to perform their function under a time constraint to keep the larger system functioning properly [10].

Challenges in embedded software testing

Some of the challenges that one can face during embedded software testing:

Hardware dependency: Hardware dependency is among the main difficulties faced during embedded software testing because of limited access to hardware. However, Emulators and Simulators may not precisely represent the behavior of the actual device and could give a wrong sense of system performance and application's usability.

Open source software: The majority of the embedded software components are open source in nature, not created in-house and absence of complete test available for it. There is a wide range of test combinations and resulting scenarios.

Software vs. hardware defects: Another aspect is when software is being developed for a freshly created hardware, during this process high ratio of hardware defects can be identified. The found defect is just not limited to software. It may be related to hardware also.

Reproducible defects: Defects are harder to reproduce/recreate in the case of the embedded system. That enforces the embedded testing procedure to value every defect occurrence substantially higher than in a standard case, other than to gather as much data as could sensibly be required to alter the system to find the foundation of the defect.

Continuous software updates: Embedded systems require regular software updates like the kernel upgrade, security fixes, different device drivers, etc. Constraints identified with the software updates influence make bug identification difficult. Additionally, it increases the significance of build and deployment procedure [11].

Faults in embedded systems

Incorrectness in hardware systems may be described in different terms as defect, error and faults. These three terms are quite bit confusing. We will define these terms as follows

Defect: A defect in a hardware system is the unintended difference between the implemented hardware and its intended design. This may be a process defects, material defects, age defects or package effects.

Error: A wrong output signal produced by a defective system is called an error. An error is an "effect" whose cause is some "defect". Errors induce failures, that is, a deviation from appropriate system behavior. If the failure can lead to an accident, it is a hazard.

Fault: A representation of a "defect" at the abstraction level is called a fault.

Faults are physical or logical defects in the design or implementation of a device.

Debugging embedded systems

Some programming languages run on microcontrollers with enough efficiency that rudimentary interactive debugging is available directly on the chip. Additionally, processors often have CPU debuggers that can be controlled -- and, thus, control program execution -- via a JTAG or similar debugging port.

In many instances, however, programmers need tools that attach a separate debugging system to the target system via a serial or other port. In this scenario, the programmer can see the source code on the screen of a general-purpose computer, just as would be the case in the debugging of software on a desktop computer. A separate, frequently used approach is to run software on a PC that emulates the physical chip in software. This is essentially making it possible to debug the performance of the software as if it were running on an actual physical chip.

Broadly speaking, embedded systems have received more attention to testing and debugging because a great number of devices using embedded controls are designed for use, especially in situations where safety and reliability are top priorities [12].

Automated testing

The scope of automation is the area of your Application under Test which will be automated. Following points help determine scope:

- The features that are important for the business
- Scenarios which have a large amount of data
- Common functionalities across applications
- Technical feasibility
- The extent to which business components are reused
- The complexity of test cases
- Ability to use the same test cases for cross-browser testing

Types of automated testing

Smoke testing

- Unit testing
- Integration testing
- Functional testing
- Keyword testing
- Regression testing
- Data Driven testing
- Black box testing

Benefits of automation testing

- 70% faster than the manual testing
- Coverage of more application features
- Reliable in results
- Ensure consistency
- Saves time and cost
- Improves accuracy
- Less human dependent
- Increases efficiency
- Better speed in executing tests
- Re-usable test scripts
- Test frequently and thoroughly
- More automation cycle in execution

Testing of the embedded os and application for embedded os

- Model-based testing
- Functional testing of the OS/embedded software
- Regression of the OS/embedded software
- Unit testing
- Mocking (emulation of external data sources and signals)
- Testing of the command stack (including testing floating point operations)
- Testing components based on protocols (CAN, I2C, RS232, RS245, TCP, REST)
- Testing of drivers for elements of embedded systems with respect to specific OSes (Linux, RTOS, QNX)

Direct testing of systems (hardware testing)

Model-based testing using a model for PLD and microcontrollers

• Spec-based testing

• Test-bench testing (testing systems on a test bench using external carrier boards and sensors)

• Testing communications with external systems at the level of interfaces and signals (sampling and analysis of input/output signals)

• Testing based on modeling external signals (black box – SCPI, VXI-11, or other wave form generator)

• White-box testing based on a component's specification

• Load testing using an input stream of prepared data (measurement of the performance of an MCU or cryptographic coprocessors)

• Fault testing (supplying edge-case data , modeling crosstalk and voltage surges for power circuits – from a carrier board)

When testing we use our internal tools to store and analyze data (a Neuron-R system), but we use enterprise systems to generate data. Storing and analyzing the test results using a Neuron-R allows us to generate reports, view past results, and assess aggregate metrics in real-time on a single chart, which makes it possible to see the correlation between input and output parameters. This is an important ability when testing embedded systems and devices.

Test case representation

Test Case Representation In our context, a test case execution is akin to executing the environment simulator. The domain model represents various components in the RTES environment. As mentioned earlier, there can be multiple instances for each of these environment components during simulation. For example, in a gate controller RTES, we can have an environment component representing trains in general. And then, during simulation, we can simulate multiple trains where each simulated train will be represented by an independent running instance of the train environment component

On-line testing

On-line testing addresses the detection of operational faults, and is found in computers that support critical or high-availability applications11 The goal of on-line testing is to detect fault effects, that is, errors, and take appropriate corrective action. On-line testing can be performed by external or internal monitoring, using either hardware or software; internal monitoring is referred to as self-testing. Monitoring is internal if it takes place on the same substrate as the circuit under test (CUT); nowadays, this usually means inside a single IC—a system-on-a-chip (SOC).

There are four primary parameters to consider in the design of an on-line testing scheme:

Error Coverage (EC): This is defined as the fraction of all modeled errors that are detected, usually expressed in percent. Critical and highly available systems require very good error detection or error coverage to minimize the impact of errors that lead to system failure [13].

Error Latency (EL): This is the difference between the first time the error is activated and the first time it is detected. EL is affected by the time taken to perform a test and by how often tests are executed. A related parameter is fault latency (FL), defined as the difference between the onset of the fault and its detection. Clearly, FL ≥ EL, so when EL is difficult to determine, FL is often used instead.

Space Redundancy (SR): This is the extra hardware or firmware needed to perform on-line testing.

Time Redundancy (TR): This is the extra time needed to perform on-line testing.

An ideal on-line testing scheme would have 100% error coverage, error latency of 1 clock cycle, no space redundancy, and no time redundancy. It would require no redesign of the CUT, and impose no functional or structural restrictions on the CUT. To cover all of the fault types described earlier, two different modes of on-line testing are employed: concurrent testing which takes place during normal system operation, and non-concurrent testing which takes place while normal operation is temporarily suspended. These operating modes must often be overlapped to provide a comprehensive on-line testing strategy at acceptable cost [14].

Non-concurrent testing

This form of testing is either event-triggered (sporadic) or time-triggered (periodic), and is characterized by low space and time redundancy. Event-triggered testing is initiated by key events or state changes in the life of a system, such as start-up or shutdown, and its goal is to detect permanent faults. It is usually advisable to detect and repair permanent faults as soon as possible. Event-triggered tests resemble manufacturing tests.

Time-triggered testing is activated at predetermined times in the operation of the system. It is often done periodically to detect permanent faults using the same types of tests applied by event triggered testing. This approach is especially useful in systems that run for extended periods, where no significant events occur that can trigger testing. Periodic testing is also essential for detecting intermittent faults. Periodic testing can identify latent design or manufacturing flaws that only appear under the right environmental conditions [15].

Concurrent testing

Non-concurrent testing cannot detect transient or intermittent faults whose effects disappear quickly. Concurrent testing, on the other hand, continuously checks for errors due to such faults. However, concurrent testing is not by itself particularly useful for diagnosing the source of errors, so it is often combined with diagnostic software. It may also be combined with non-concurrent testing to detect or diagnose complex faults of all types.

A common method of providing hardware support for concurrent testing, especially for detecting control errors, is a watchdog timer. This is a counter that must be reset by the system on a repetitive basis to indicate that the system is functioning properly. A watchdog timer is based on the assumption that the system is fault-free—or at least alive—if it is able to perform the simple task of resetting the timer at appropriate intervals, which implies that control flow is correctly traversing timer reset points [16].

Interaction testing technique between hardware and software in embedded systems

In embedded system where hardware and software are combined, unexpected situation can occur owing to the interaction faults between hardware and software. As the functions of embedded system get more complicated, it gets more difficult to detect faults that cause such troubles. Hence, Faults Injection Technique is strongly recommended in a way it observes system behaviors by injecting faults into target system so as to detect interaction faults between hardware and software in embedded system.

The test data selection technique discussed in first simulates behaviors of embedded system to software program from requirement specification. Then hardware faults, after being converted to software faults, are injected into the simulated program. And finally, effective test data are selected to detect faults caused by the interactions between hardware and software.

Conclusion

Embedded Systems are good but they come with their diver's problems, this study has helped shed more light to their management and testing. Modular test techniques for digital, mixed-signal, and hierarchical SOCs must develop further to keep pace with design complexity and integration density. The test data bandwidth needs for analog cores are significantly different than that for digital cores; therefore unified top-level testing of mixed-signal SOCs remains major challenge.

References

1 Abbors F, Aho VM, Koivulainen J, Teittinen R (2017) Applying Model-Based Testing in the Telecommunication Domain and Dragos Truscan. In Model-Based Testing for Embedded Systems. 19: 515-552.

2 Galster M, Weyns D, Tang A, Kazman R, Mirakhorli M (2018) From Craft to Science: The Road Ahead for Empirical Software Engineering Research.IEEE/ACM International Conference on Software Engineering: New Ideas and Emerging Technologies Results (ICSE-NIER) 27: 77-80.

3 Kitchenham B, Charters S (2007) Guidelines for performing systematic literature reviews in software engineering.

4 Ebert C, Jones C (2009) Embedded software: Facts, figures, and future. Computer 42: 42-52.

5 Murphy C, Zoomkawalla Z, Narita K (2013) Automatic test case generation and test suite reduction for closed-loop controller software.

6 Lo D, Nagappan N, Zimmermann T (2015) How Practitioners Perceive the Relevance of Software Engineering Research. Joint Meeting on Foundations of Software Engineering 30: 415-425.

7 Petersen K, Vakkalanka S, Kuzniarz L (2015) Guidelines for conducting systematic mapping studies in software engineering: An update. Inf Softw Technol 64: 1-8.

8 Burford MA, Belli F (1984) CADAS: A tool for designing reliable embedded software and supporting testing "in the large". In Fehlertolerierende Rechensysteme 12: 101-112.

9 van Schooenderwoert N, Morsicato R (2004) Taming the embedded tiger-agile test techniques for embedded software. In Agile Development Conference 9: 120-126.

10 Offutt J, Ammann P (2008) Introduction to software testing. Cambridge: Cambridge University Press.

11 Doğan S, Betin-Can A, Garousi V (2014) Web application testing: A systematic literature review. J Syst Softw 91: 174-201.

12 Dyba T, Dingsoyr T (2009) What do we know about agile software development? IEEE Software 26: 6-9.

13 Hall T, Sharp H, Beecham S, Baddoo N, Robinson H (2008) What do we know about developer motivation? IEEE Software 25:92-94.

14 Yusifoğlu VG, Amannejad Y, Can AB (2015) Software test-code engineering: A systematic mapping. Inf Softw Technol 58: 123-147.

15 Garousi V, Elberzhager F (2017) Test automation: Not just for test execution. IEEE Software 34: 90-96.

16 Lin YD, Chu ET, Yu SC, Lai YC (2013) Improving the accuracy of automated GUI testing for embedded systems. IEEE Software 31: 39-45.