International Journal of Scientific Research and Engineering Development-– Volume 4 Issue 3, May -June 2021

Available at www.ijsred.com

RESEARCH ARTICLE OPEN ACCESS

Challenges to Embedded System Debugging and Analysis of Debugging Approaches

Sumukha K R*, Nagaraj Cholli**

*(Dept. of Information Science & Engineering, R. V. College of Engineering, Bengaluru
Email: sumukhakr.is17@rvce.edu.in)

** (Dept. of Information Science & Engineering, R. V. College of Engineering, Bengaluru
Email:nagaraj.cholli@rvce.edu.in)

----------**********----------

Abstract:

With a surge in the number and type of electronic devices, in both consumer and enterprise markets, there has been an observable increase in the complexity of processors, i.e. SoC (System on Chip) in embedded systems. With this, debugging of electronic devices effectively can be complicated and troublesome for embedded system programmers due to various hardware units, peripherals, signals, and corresponding protocols.

This paper examines all the different parameters that affect the process of debugging in various types of electronic devices. It lists out the prevalent challenges to debug such systems. It also compares and analyzes various approaches employed towards addressing the challenges during debugging of embedded systems.

Keywords **—Debug, Embedded Systems, SoC, Electronic Devices, Firmware.**

----------**********----------

I. INTRODUCTION

Debugging is the process of detecting and eliminating existing or potential errors(bugs) from a system which involves computation. The electronic industry has been focused to provide more functionality in embedded systems for better consumer satisfaction over the past decade. This has led to more complex SoCs and more convoluted firmware written. Hence the operability of debugging designs of electronics has been struggling to keep up with the advancements in processing cores, firmware programs, and related peripherals.

Software developers dedicate around 35-50 percent of their time towards validating and debugging software [1]. The cost of debugging, testing, and verification is estimated to account for 50-75 percent of the total budget of development. Considering the previous factors, this time and money dedicated towards debugging embedded systems is expected to be significantly higher [2].

The increasing design complexity, integration level, and SoCs getting more intricate, have created new debug challenges which can have an adverse effect on system reliability in case of occurrence of errors [3]. This makes it all the more important to design a robust debugging design which doesn't overwhelm developers and supports development pro-actively.

Development of an electronic device is already an expensive undertaking, involving designing, programming, prototyping, manufacturing and testing in various phases [4]. On top of this, debugging has to be done throughout the device's lifetime for maintenance which is also the least efficient process of all.

The fact that electronic devices require interaction between various hardware components like processor, chipset, memory and peripherals for

their operation make them complex and their debugging process more complicated.

II. Architecture of Embedded System

In order to understand the challenges to debugging an embedded system and analyse the approaches towards debugging in hardware, there is a need to understand the architecture of embedded systems. In embedded systems, hardware and software process input signals to generate an application-specific output. Embedded systems have firmware as in the flash memory chip. This firmware is used for system boot-up, to control various specific functions and configure components of the electronic device. There are mainly two main types of architectures in embedded systems: Von-Neumann architecture, where the program and data are kept in the same memory; and Harvard architecture, instructions and data are kept in separate memory locations. An embedded system has the following primary components: processor, power supply, clock, memory, bus, I/O ports.

- Processor: the brain of an embedded system. It can be microprocessor, microcontroller, digital signal processor (DSP) etc.
- Clock / Oscillator Circuits: for system timers, machine clock cycles for CPU and scheduling tasks.
- Memory: RAM, Cache, Flash, ROM, PROM

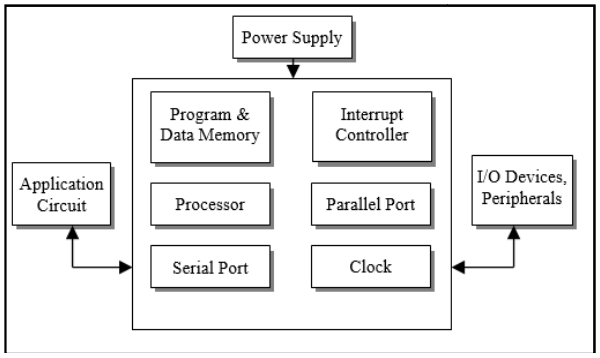

Fig.1 Architecture of a typical embedded system.

III. Debugging Modes

For SoCs, there mainly exists two basic modes of debugging, depending on use-case.

A. Invasive Debug

Invasive debug is a debug procedure where the processor can be controlled and observed. There is also provision to modify its state, registers, and memory, which is done by setting up breakpoints and watchpoints. This can be advantageous and useful when we want to see real-time process states at run-time. However, since this causes the processor to halt, it may affect the performance of the system.

B. Non-Invasive Debug

Non-invasive debug is the debug procedure where the processor can be observed, but cannot be controlled at run-time. The processor's flow of execution is recorded and stored in a file or a predefined memory location. This file is then analyzed later for gaining insights at instructions executed, data transferred etc.

IV. Challenges to Electronic Device Debugging

The electronic devices prevalent currently in the industry cannot guarantee that all errors in system firmware and software are removed before released into the market. Errors unavoidably show up when these systems are put into practical use. This calls for the need of debugging. Even when the system shows deviation from normal functioning or unexpected behaviour, Debugging must be brought into operation. At present, the industry spends on average more than 50% of the total project duration on post-silicon validation and debugging. However, there are numerous challenges to debugging, in embedded systems which are discussed in this section.

A. Intrinsic Debugging Challenges

The steps of the debugging process consist of scan the error symptoms, recognize the cause, and eventuallyfix the error. However, the intrinsic challenges are impossible to be eliminated due to the very nature and definition of debugging. This process is cumbersome due to the following reasons:

- *Error symptoms might not give direct indications about the cause*. The cause of error can be in program source code and/or can arise during execution.
- *There may be an interrelation between errors*, i.e. symptoms of error can change during the course of debugging.
- There is always a probability of occurrence of new errors on fixing the old ones.

B. Categorization

Problems can range from catastrophic to mild. They can cause a complete shut down or produce erroneous results only under specific operating conditions. When problems arise, the first challenge is to categorize them [4]. However, there aren't many indicators except for severe bugs which allow for instant categorization of the problem for a quick debug.

C. Limited Observing Capacity

Limited on-chip storage and off-chip debug bandwidth restricts the amount of data that can be monitored at a time with respect to specific state of the system. Hence it is a challenge at runtime to keep an eye on the many simultaneous branching at run-time [5].

D. DUT (Device Under Test) Visibility

While debugging embedded systems, the debugging process is often observed to be directly proportional to the complexity of its components. When the component undergoing debug and test is not observable, it is difficult to what is going on in and when it fails. These types of problems usually lead to production delays and product cancellations [6].

E. Tightly Coupled Processors

Electronic Systems have gone on to move to System-on-Chips (SoCs) from printed circuit boards (PCBs) for greater efficiency and less power consumption. However, this has caused a decrease in system observability, controllability, and has consequently arisen new debug challenges for embedded systems [7].

F. Debugging Instruments

Previously, testing and debugging could be done through connecting external instruments to PCBs like logic analyzers. Now, with advancement of technology, components have been integrated into a single chip. This makes it even more difficult to connect and use external debugging devices [8].

G. Clocking

Most SoCs have components based on clocks and clock ratios. Some components have clock frequencies which are non-multiples of each other, which results in different phase relations over time [6]. This scenario gets even more complicated when variations from peripherals are introduced, leading to asynchronous clocks. This results in accumulation of small errors in timing and process-sensitive bugs.

H. Power Control Circuitry

Electronic devices are usually low electric power consumption systems, with large stand-by times. To implement this, there is a complex circuitry which optimizes power usage, by changing voltages, or switching off many times per millisecond [6]. All this leads to generation of logic and circuit bugs which are difficult to debug due to barriers between power-on and power-off components.

I. Internal Buses

To observe the point of failure, the state and purpose of internal buses must be known to the debugger. Debugging of devices may also involve measurement of performance to identify potential bottlenecks [6]. The visibility of busses is the main concern here, while debugging which may require tracking of the data flow simultaneously through multiple busses.

J. Security Concerns

Embedded Systems contain numerous physical connections and components in the IC (Integrated Circuit) such as:

- CPU for computation
- Memory Controller to regulate access to memory

- Input/Output Controller to regulate CPU access to peripherals
- DRAM for storage
- Interrupt Controller to handle and signal various types of interrupts
- Flash to store BIOS Firmware
- Power controller

In addition to this, there are also communication protocols for connecting peripheral interfaces. Some examples of these peripheral interfaces are USB, SATA, PCIe. There are also buses which are responsible for transport of data between various components [9].

Due to the above various hardware components, ensuring security is a challenge while debugging. Security of the device can be summarized as the security of individual components which must be kept strong. However, the same security of different individual components can prove to be a hassle while gaining access during debugging.

V. APPROACHES TO EMBEDDED SYSTEM DEBUGGING

In order to address the above discussed challenges of debugging, there are numerous approaches towards addressing some of the challenges during debugging embedded systems which are analyzed in this section. These approaches are compared on parameters like real time execution support, peripheral implementation support, Provision for viewing and modifying RAM and peripherals, I/O requirements for debugging and occurrence of data loss. The analytic comparison of various approaches of hardware debugging is given in **Table 1** based on the above parameters.

A. Simulation by Software

Simulators provide simulation of program execution, with access to internal processor registers. These are able to read, write, set up breakpoints and show program flow history. But this is still challenging in embedded systems as they have peripherals, clock variation (as they run slower than actual) and interrupts. They can overcome the challenges of visibility, component accessibility up to some extent. Many electronic devices can be debugged using proper knowledge of use of simulation software. However, there is a parity in the performance of simulation vs the actual processor itself. These are debuggers focused on determining what the code executed or what it does. However, they do not provide much component visibility. They provide software debug capabilities using standard interfaces like GNU Debugger (GDB) and Cycle-Accurate Debug Interface (CADI) to connect software debuggers to virtualized hardware.

B. On-Board Emulators

These are emulators which are implemented in-circuit, which can be plugged into a system in place of the processor under test. During the design process, hardware emulators can be used to test chips under development. This tackles errors at initial stages of design, and usually has better debug capabilities than FPGA prototyping. It is still limited by the ability to replicate custom devices accurately. They provide peripheral emulation, breakpoint set up capacity and profiling of execution history. Some on-board emulators have specialized ASICs (Application specific Integrated circuit) or FPGA (field-programmable gate array) to mimic core processor in terms of execution and interaction with peripherals, with option for direct I/O interaction with actual peripherals. This is accomplished by using RAM for emulation where the firmware for the processor is stored and is connected through a separate socket to enter in debug mode. They can provide run-time as well as non-invasive debug support. However, the cost of such implementation in every embedded system is high.

C. Build and Download Approach

This process is most utilized during firmware development. The process is as follows: Source code is changed after looking for logical or syntactical errors, executable file is rebuilt and downloaded onto the device memory. This process is repeated till the device functions properly.

Program flow and debugging information can be retrieved through serial port and I/O pins. This process is still cumbersome and tedious.

D. In-Circuit Simulation

This is the approach combination of a simulator with a module for communication with the processor. The simulation stimulated the processor's input to configure some values on the microcontroller. The simulator gets information about the state of hardware, state of input pins or branching condition through this stimulus. This process is still slow compared to actual processor speed due to simulation, hence it is difficult to keep a check on time-critical errors, e.g. UART. Support for complex peripherals is also limited.

E. In-Circuit Debugger

These are development tools with extra features to support custom silicon features. This communication occurs between host and target to facilitate code debug. RAM on the host or dedicated flash memory is used for this purpose, with a customized protocol.

F. Run-Time Monitoring

Run-time monitoring is a step forward from simulation separation from hardware. The host issues commands to the processor to perform debug functions, read memory at run-time. Code execution control is also possible and is done through a communication channel like UART. This approach has features like setting up watchpoints at compile-time, periodic data gathering, and memory state retrieval. Debugging devices are connected to the product through JTAG boundary scan, which provides a mechanism of communication with on-board chips. Accessing the depth of hardware remains a challenge and is limited by the level of on-chip support and instrumentation. For example, it is useful in debugging FPGA Prototypes of chips oncircuit board.

TABLE 1
ANALYTIC COMPARISION OF HARDWARE DEBUGGING APPROACHES

Debugging Approaches	Parameters of Evaluation				
	Real-time execution support	Peripheral implementation support	Provision for viewing and modifying RAM and peripherals	I/O requirements for debugging	Loss of program memory or RAM data
Simulation by Software	No	Very Limited support	Yes	None	No
On-Board Emulators	Yes	Complete support	Yes	None	No
Build and Download Approach	Yes	Complete support	No	None	Yes
In-Circuit Simulation	No	Limited support	Limited	Serial ports, I/O pins required	Yes
In-Circuit Debugger	Yes	Complete support	Yes	Few I/O requirements	Yes

VI. FRAMEWORKS AVAILABLE FOR HARDWARE DEBUGGING

With increase in functionality and performance of SoCs, it has become a challenge to build an infrastructure to have corresponding debug capability.

To address some of the challenges observed in debugging electronic devices, a significant amount of effort has been dedicated towards developing tools, frameworks and environments to ease the process of debugging embedded systems. Considering the discussed challenges, a debug framework or design for embedded systems should support four core debug functionalities: Real-time run control, internal memory access and debug file generation and set up trigger points for observation and monitoring.

The industry has come up with tools which overcome some of the challenges to provide a comprehensive SoC debug support targeted towards electronic devices. Some of which are listed below:

- Arm Debugger (Debugging Framework for ARM based SoCs)
- GDB (GNU Debugger)
- KDB (Kernel Debugger/KGDB (Kernel GDB server)
- OpenOCD (On-chip Debugger)

VII. CONCLUSION

Debugging remains one of the most difficult aspects of applied computer science. Efficiency in debugging can be obtained through continued learning, malleable approach towards problems, and effective use of debugging tools and frameworks. The process of debugging requires finesse and expertise due to its inherent nature, which is why it is regarded more of an art than an exact science.

Current debug frameworks are typically limited by shortage debug support facilities within the SoC. Upcoming generations of SoCs will be equipped with more processors, with inbuilt scalable communication infrastructure, with numerous threads of program instances running simultaneously and support for more specific functional capabilities in electronic devices. The current debug frameworks are not capable of handling such towering demands. We have discussed a few high-quality debug approaches for

embedded systems which can ease problem fixing and reduce time taken for development. However, with the discussed approaches, a combination of simulation and monitoring on a host system, with in-circuit emulation would be the path to progress, when it comes to debugging embedded systems. There is scope for research to develop novel debug techniques concerning firmware and embedded systems. Although a panacea can't be expected to solve all the challenges discussed, more effort must be put into developing more utilitarian frameworks with higher debugging prowess and focusing on ease of operation.

ACKNOWLEDGMENT

We thank R.V. College of Engineering, Bengaluru and its faculty for the support and guidance in writing this paper.

REFERENCES

[1] Britton, T., Jeng, L., Carver, G., Cheak, P., Katzenellenbogen, T. 2013. Reversible debugging software. Cambridge Judge Business School;

[2] RTI. 2002. The economic impacts of inadequate infrastructure for softwaretesting;

[3] Gang, Wang and Zhang Shengbing. "On-chip debug architecture for MCU-DSP Core based system-on-chip." 2011 IEEE International Conference on Computer Science and Automation Engineering 2 (2011): 605-608.

[4] Rob Aitken and Erik Jan Marinissen. 2008. Guest Editors' Introduction: Addressing the Challenges of Debug and Diagnosis. IEEE Des. Test 25, 3 (May 2008), 206–207

[5] B. Vermeulen, "Design-for-debug to address next-generation soc debug concerns," 2007 IEEE International Test Conference, 2007, pp. 1-1, doi: 10.1109/TEST.2007.4437683.+

[6] K. Schultz and K. Paranjape, "SOC Debug Challenges and Tools," 2006 IFIP International Conference on Very Large Scale Integration, 2006, pp. 385-390, doi: 10.1109/VLSISOC.2006.313266.

[7] Yi-Ting Lin, Wen-Chi Shiue, and Ing-Jer Huang, "An Embedded Infrastructure of Debug and Trace Interface for the DSP Platform", IEEE/ACM Design Automation Conf. (DAC), p.866 - 871, San Diego, California, USA, June 2007

[8] A. Mayer, H. Siebert and K.D. McDonald-Maier, "Boosting Debugging Support for Complex Systems on Chip", Computer, v.40 n.4, p.76-81, April 2007

[9] N. Potlapally, "Hardware security in practice: Challenges and opportunities," in 2011 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST), San Diego CA, 2011 pp. 93-98

