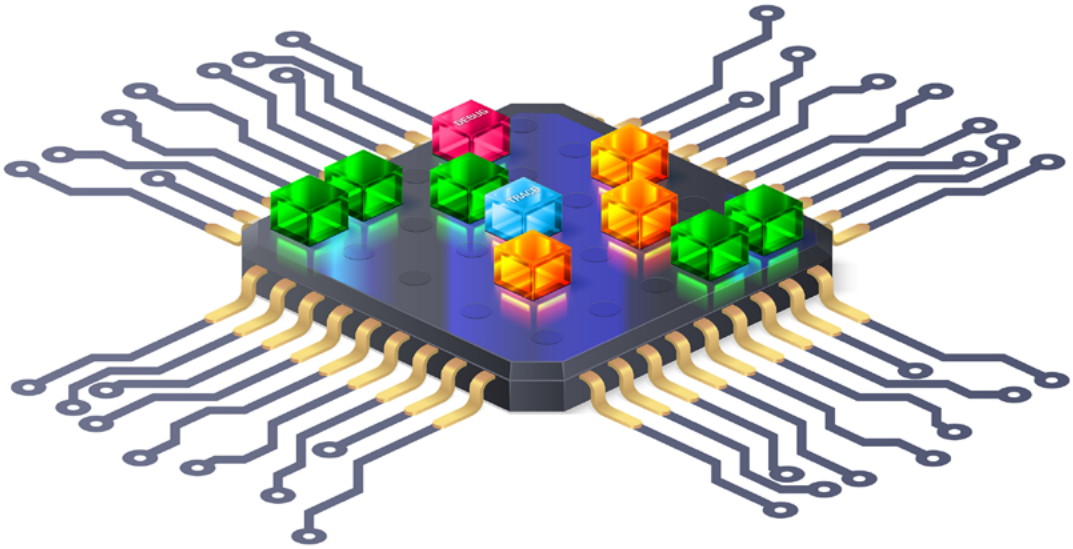
LAUTERBACH
DEVELOPMENT TOOLS

How to Design SoCs for Optimal Debuggability and Improved Time-to-Market



Introduction

Software is no longer just a tool for convenience – it's a driver of innovation. As the backbone of the digital world, software has evolved from simple commands to complex, adaptive systems. With advancements in AI and quantum computing, software will keep shaping our future, unlocking new opportunities and redefining what's possible in the digital age.

A System-on-Chip (SoC) needs embedded software to enable efficient operation and coordination of its integrated components. The software manages functions, optimizes performance, and ensures power efficiency, allowing the SoC to handle complex tasks and meet specific application needs. Without software, the SoC's components would lack coordination, limiting its potential.

Integrating software on a SoC is an important step in developing and producing embedded systems. However, errors can occur during the creation of software code, which developers have to find and eliminate at great expense using methods such as debugging.

Debugging an SoC is crucial for eliminating design errors, optimizing performance, and ensuring functionality and security in various application scenarios. For example, debugging enables the identification and elimination of security vulnerabilities that may have been caused by design or implementation errors.

Other use cases such as timing and requirement analyses – e.g. to evaluate worst execution times – or code coverage measurements for functional safety certifications are also indispensably linked to debug tools and extensions for real-time trace.

According to a study at the Judge Business School of the University of Cambridge [1], UK, developers spend almost 50 percent of their time debugging, with the rest of their time being spent on other activities such as writing code. Phaedrus claimed that 90% of most projects was spent on debug and optimization. In the early 2000's Nokia did a study over 200 projects and found that having trace saved, on average, 75% of the time it took to debug a design. This proves how important debugging is to write error-free code and develop error-free products.

WHITEPAPER

To optimally prepare SoCs for debugging, it is essential that tool suppliers and SoC designers work together early on in the development process. The requirements for the debug tools change depending on the characteristics of the SoC – and the ease of use of the SoC, which the supplier considers in advance. Only close co-operation with SoC manufacturers during the development process ensures that they meet all the requirements for modern and efficient debugging. On the other hand, thorough debugging processes offer SoC suppliers numerous advantages: reducing manufacturing costs; speeding up time to market; improving product quality and safety.

This document is intended to help SoC manufacturers to design their chips to be "simply and easily debuggable" from the outset in order to prevent potential problems from arising for their customers in later phases of the development process. A chip that is optimally designed from a debugger‘s point of view means less effort for customers, faster time to market and, above all, lower development costs as well as increased chip sales, because a failed project sells no units.

Importance of Debugging and Tracing for Time-to-Market

SoC manufacturers integrate different functional units (IP blocks) into a SoC design. They must ensure that all components work together smoothly and meet the performance requirements of the end system. Intellectual Property (IP) providers on the other hand develop and license specialized Intellectual Property (IP) blocks. These are prefabricated and tested designs that SoC suppliers can integrate into their chips. For a SoC to be debugged effectively, it must fulfill certain requirements. These requirements relate to both the hardware and software architecture of the SoC. It must provide special interfaces and debugging ports that enable access to internal components and signals.

Debug functions on an SoC come with added costs, including silicon real-estate and manufacturing resources. However, the benefits they provide make this investment worthwhile for both semiconductor suppliers and their customers. Faster debugging reduces development time and accelerates time-to-market. Integrated debugging functions are essential for maintaining consistent quality, especially in safety-critical industries such as automotive, avionics, space applications like satellites, medical technology, robotics, cloud services, and IoT, where high reliability is crucial. Modern debugging solutions also incorporate security features to prevent unauthorized access and minimize attacks on the system, protecting the SoC in the field and boosting customer confidence in the product’s security.

Debugging is the process in which developers specifically search for and fix errors or problems in the code or hardware of a system by analyzing the status and values of variables or stopping program execution at certain points (e.g. using breakpoints).

Tracing, on the other hand, is the continuous recording of the execution sequence of a program or the data flows in the SoC to analyze system performance or understand the causes of complex problems. While debugging is often carried out interactively and selectively, tracing provides a more comprehensive and temporally complete view of the processes in the system which allows the run-time behavior of multiple IPs to be monitored simultaneously. An SoC should therefore be able to capture the trace information that is required to analyze program execution. This includes:

- **Program Flow Trace:** Records the progress of program execution to see which instructions have been executed by the processor as well as monitoring which decisions in a branch have been taken and, maybe more importantly, not taken.

- **Data Trace:** Data trace refers to the tracking and recording of load and store operations during program execution. It provides insight into the data flows within an SoC, particularly about read and write operations in the memory. Data trace typically needs high bandwidth and, therefore, will not be present in most application processors or complex SoCs.

- **System Trace:** Transports system-level trace data from embedded systems, enabling real-time monitoring of processor activities, memory accesses, and inter-component interactions.

- **Bus Trace:** Bus trace refers to the capture and recording of data traffic that runs via internal communication buses in the SoC.

- **Event Trace:** Records important events in the system, such as interrupts, context switches or system calls. Whether an event trace is available depends on the respective IP.

Many modern devices include a trace port or an Embedded Trace Buffer, which is capable of providing real-time, non-intrusive information about program flow and data operations. This can be used to work out where code has been, how long things took, to analyze task switches, interrupts, and to generate code coverage reports. If one thinks of traditional debugging as a series of snapshots, then trace-based debugging is a video with timestamps. Developers use time stamps to measure the latency between events and identify bottlenecks or delays. In multi-core or distributed systems, time stamps enable the synchronization of different components to ensure that actions are timed correctly.

Debug Friendly SoC Designs – Key Considerations

An SoC must fulfill certain requirements in order to be optimally prepared for debugging. This includes integrated debug interfaces, special hardware modules for monitoring and controlling the SoC, trace mechanisms, and support from the software toolchain. These features enable developers to identify errors, analyze performance problems, and ultimately deliver a high-quality, reliable product.

PURPOSE OF THE SOC

Before the start of the development, it should be clear for which applications the corresponding SoC is to be designed. After all, this determines which peripheral components are to be accommodated on the SoC. For example, as the number of cores increases, debugging solutions must be designed in such a way that they work efficiently, even with large core numbers. This applies, for example, to parallel debugging sessions or the detailed analysis of states and interactions. SoC manufacturers should also consider how certain modes and scenarios can be debugged.

With multi-core SoCs, the trace solution must be able to synchronize data from multiple cores and subsystems to enable coherent analysis. The trace buffering must be sufficiently dimensioned to avoid data loss during data acquisition. Continuous recording is particularly important for real-time analyses.

CPU REQUIREMENTS

The processor has to integrate debug functions, for example, hardware breakpoints, which allow interrupting the execution of the program at a specific point. Watchpoints monitor memory accesses and stop the program execution flow, when certain memory addresses are accessed. Benchmark and Event Counters can either be independent or feed into the trace subsystem. Breakpoints and watchpoints are part of the provided IP. In addition, the processor must be able to execute instructions step by step and monitor the status of interrupts and their effects on system performance with interrupt handling monitoring. No code should be executed without the user‘s control. Debugging should therefore be possible from the first instruction after the reset. This is often a key requirement e.g. for Automotive applications, where debugging through reset is almost always required.

MULTI-CORE DEBUG & TRACE CAPABILITIES

Multi-core SoCs place higher demands on debugging than single-core architectures, especially due to the simultaneous execution of multiple cores and the interactions between them. Effective debugging in such architectures requires cross-triggering between the cores, advanced monitoring functions for inter-core communication, specialized mechanisms for detecting synchronization problems (e.g. race conditions and deadlocks), and comprehensive traces, like system tracing (e.g., via MIPI STP), and performance analysis to identify timing and performance problems. When debugging or tracing, it may be necessary for an event that occurs on one core to trigger an action on another core or hardware unit. Cross-triggering enables this interaction by exchanging signals or triggers between the cores.

DEBUGGING INTERFACE AND SUPPORTED STANDARDS

Low level interfaces: Different organizations have defined debug and trace standards so that not every SoC manufacturer has to develop its proprietary solution.

The JTAG (Joint Test Action Group; IEEE 1149.1) and compact JTAG (IEEE 1149.7) debug interfaces provide a comprehensive ecosystem for testing, debugging, and programming. Another example is the RISC-V® Foundation, which has defined debug and trace specifications for RISC-V® based CPUs.

A further example is the MIPI Alliance® [2], which has promoted standardized interfaces for debugging like Debug Over I3C™ or Debug Over USB℠ [3]. The MIPI Alliance also defines several interfaces for tracing, such as Debug Over USB℠ or Debug Over IPSockets [4]. MIPI provides a layered debug solution.

Arm CoreSight Debug and Trace: Arm has developed its own debug and trace ecosystem specifically for the efficient monitoring, analysis, and diagnosis of SoCs with Arm processors. It provides a comprehensive infrastructure to give developers a deep insight into how a system works.

OPEN SOURCE

When using open source tools for debugging RISC-V® SoCs, for example, there are some important points to consider. These relate to legal and security aspects as well as technical and organizational challenges, including

- Security
- Maintenance, support & warranty
- Adaptability and flexibility
- Integration into existing toolchains
- Long-term scalability
- Legal and liability, as well as commercial aspects

Open source tools offer flexibility, but often require additional integration and maintenance. It is important to use these tools responsibly and ensure their long-term sustainability by actively participating in the community and carefully monitoring updates. A company, which uses Open Source, still has to spend resources to productize and maintain the open source solution.

SECURITY

Implementing security mechanisms on an SoC is crucial for vendors. The manufacturer has to think about threats and different user types, which domains should be open for debugging and which should be locked, and whether there are any interactions between them. Also, the signals for reset and power down must be separated in the SoC. Debugging mechanisms provide deep access to internal hardware resources, which is a potential gateway for attackers. Developers must therefore deactivate debugging interfaces by default or secure them through authentication. Only authorized users should be allowed to access the system via these interfaces.

Many systems also implement secure boot to ensure that only trusted and authentic firmware is executed. In security-critical SoCs, debugging should only be permitted for non-security-relevant application areas. Certifications such as ISO 26262 or Common Criteria may require debugging interfaces to be disabled or completely removed to ensure the safety of the system.

Furthermore, there are technologies such as Arm® CoreSight™ SDC-600 secure debug channel and "Secure JTAG", which aim to make the debugging process secure and build their own ecosystem for security on their IP. Another secure approach to enabling debugging functions in production systems is to use keys or debug tokens. Intel®, for example, uses hardware-based authentication like "Disable CPU Debug" or "Delayed Authentication Mode" [5].

OFF-CHIP-TRACE CAPABILITIES AND INTERFACE

Off-chip trace enables continuous recording of program execution and data flows to analyze system behavior in detail without the trace data being restricted by limited internal memory. Sufficient pins must be available to transmit the trace data off chip at the required band width. Off-chip trace pins are often multiplexed with other features but essential features should not doubled up with trace pins. High-speed-serial and parallel are used as interfaces. Furthermore, off-chip trace requires compatible external tools that can interpret and analyze the trace data.

SAFETY

Safety is particularly important in addition to security, especially in safety-critical applications such as in the automotive industry, medical technology

or aviation. For example, in the event of an error, the system should be designed in such a way that it automatically switches to a safe state. There are several regulations for safety documented e.g.in ISO26262 for the automotive sector or DO-178 for avionics. For various requirements from these safety standards, such as code coverage measurements or timing analyses, the availability of off-chip trace is a essential prerequisite in order to be able to carry them out in a time- and resource-efficient manner. An SoC that has (also) been developed for safety-critical applications must, therefore, implement a trace interface.

DOCUMENTATION

Another big part of debugging is the availability of consistent documentation. It serves as a vital source of information and streamlines the entire process. For example, documentation provides a comprehensive overview of the system architecture, including all IP blocks, interfaces, and communication paths. It helps to identify known errors and typical problems more quickly. In safety-critical applications like space, aerospace, automotive, and medical technology, thorough documentation is essential for traceability. It ensures that information about known problems and their solutions is available for later development phases and other teams. Detailed and precise documentation is key to making the debugging process faster, more consistent, and safer.

The Perfect SoC for Optimal Debuggability

All these requirements end in an SoC, which is – from Debugging-Vendor-perspective – the „perfect SoC“ for debugging. To optimally prepare an SoC for debugging, special IP blocks must be integrated to support the development and debugging process. Below are the 10 most important IP blocks that are required in an SoC for an effective debug and trace infrastructure:

JTAG OR ANY OTHER DEBUG INTERFACE:
Provides a standard interface for accessing internal registers and memory contents. It enables the CPU to be stopped and controlled and status information to be read out.

DEBUG UNIT:
Manages the debugging functionalities of the SoC, e.g. setting and managing breakpoints, watchpoints, and single stepping.

TRACE MODULES:
Program-Flow-Trace for example, records the sequence of executed commands to trace the program execution. Important for troubleshooting complex programs. Data Trace records memory accesses and data changes. This is crucial for identifying memory errors or data inconsistencies. Event trace and System Trace on the other hand enable the tracking of events (software or hardware) and signals within the SoC to analyze time sequences and look out for synchronization problems.

TRACE BUFFER OR EXPORT INTERFACES:
Saves trace data temporarily before it is exported. These buffers must be large enough to avoid data loss, especially at high execution frequencies. Export interfaces need to be accessible and provide sufficient bandwidth for the essential usage scenarios.

EMBEDDED CROSS TRIGGERING:
Coordinates debug and trace activities between multiple cores or subsystems. It can generate and receive trigger events to enable synchronous analysis in multi-core architectures. Particularly important in systems with multiple processors to track complex interactions between cores.

SYSTEM MONITORS AND
PERFORMANCE COUNTERS:
Monitors the performance parameters of the SoC, such as CPU utilization, memory accesses, bus or cache usage, and memory access latencies. This information is useful for identifying bottlenecks and inefficient code sections.

EMBEDDED LOGIC ANALYZERS:
Enables internal signals to be sampled and analyzed directly on the chip. These analyzers work like external logic analyzers but are integrated into the SoC and allow the monitoring and diagnosis of problems at signal level without the need for external devices.

SECURITY MODULES:
Control access to debug features to ensure that sensitive information remains protected. Access controls, encryption, and authentication are often used. They are also important for protection against unauthorized access, especially in security-critical and trusted applications.

BUSES OR NETWORKS:
A separate bus or internal network that is only used for debugging and trace purposes ensures low-invasiveness of debugging. Creates simple and scalable access to various debug and trace components, also manufacturer-independent.

POWER MANAGEMENT:
Makes it possible to carry out debugging in various power-saving modes. This allows developers to analyze the state of the SoC even when parts of the system are in a low power state. All possible low-power mode scenarios and their influence on debugging must be considered. This concerns, for example, settings of the debug pins in the firmware so that they are also active in low-power modes or a possible manual selection of suitable clock generators for peripheral devices. As many chips often switch off all oscillators that are not low-speed oscillators when they are stopped, it is possible that a peripheral device used obtains its clock source from a switched-off clock generator and a low-speed oscillator has to be selected manually.

Conclusion

System-on-chips (SoC) require software as the basis for functionality. Integrating software on an SoC is an important step in the development and production of embedded systems. However, errors can occur during the creation of software code, which developers have to find and eliminate at great expense using methods such as debugging.

A comprehensive debug and trace infrastructure is crucial for the development and test processes of an SoC, as it significantly influences the efficiency and quality of the product. A well-dimensioned debug solution such as Lauterbach TRACE32® enables developers to quickly identify errors and analyze the functionality of the SoC in detail [6].

Trace functions offer seamless tracking of program execution, making critical errors and optimization potential clearly visible [7]. Particularly in complex, safety-critical applications, such as in automotive or medical technology, precise monitoring is required to ensure system integrity and detect unforeseen problems at an early stage.

In addition, special trace ports and buffers facilitate analysis without compromising performance. Targeted trace data and efficient export mechanisms enable developers to maintain an overview of all processes and memory accesses, even in highly complex systems, and ensure the reliability and stability of the SoC in the long term.

REFERENCES:

[1] https://www.jbs.cam.ac.uk/2013/research-by-cambridge-mbas-for-tech-firm-undo-finds-software-bugs-cost-the-industry-316-billion-a-year/
[2] https://www.mipi.org/
[3] https://www.mipi.org/specifications/stp
[4] mipi.org/hubfs/white-papers/mipi-Architecture-Overview-for-Debug-v1-3.pdf
[5] https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/secure-coding/intel-debug-technology.html
[6] https://www.lauterbach.com/
[7] https://www.lauterbach.com/products/trace-extensions/powertrace-system
[8] https://www.lauterbach.com/products/software/trace32-powerview