Safety and Security Aware Pre-silicon Concurrent Software Development and Verification

Frank Schirrmeister
Cadence Design Systems, Inc.
2655 Seely Avenue, San Jose, CA 95134

Joe Fabbre and Max Hinson
Green Hills Software LLC
30 W Sola Street, Santa Barbara, CA 93101

***Abstract*- Simultaneously designing and testing both software and SoC designs before silicon is an enticing goal to reduce the time and cost of building embedded devices. The earlier that applications can run on a reliable representation of the SoC, the better. Merging the software and SoC paths before silicon saves time and money and creates a virtuous cycle that gets the embedded product to market faster. This paper describes a new pre-silicon continuum for concurrent SoC and software development and verification for safety and security critical systems, composed of a safety-certified RTOS and advanced C/C++ development tools that continuously support SoC verification from the earliest functional simulator (virtual prototype with Arm Fast Models) through RTL emulation and FPGA prototype stages on the path to first silicon. The continuous convergence brings more safety, security and verified reliability while establishing a more mature software enablement foundation for lead customers and partners at the time of first silicon.**

I. Introduction

As we approach a world with a trillion connected devices in a not too distant future, security and safety are crucial criteria for hardware/software systems. Studies [1] have shown that the relative cost to fix an error highly depends on the phase in which it is found. Compared to the requirements phase, the cost can be 3-6 times as high in the design phase, 10 times in the coding phase, 15-40 times in the development and testing phase, 30-70 times in acceptance testing, and 40-1000 times in the operational phase. Figure 1 [2] overlays the impact of software rework on system cost over the V-Diagram, which is often used in the automotive and aero-defense domains.

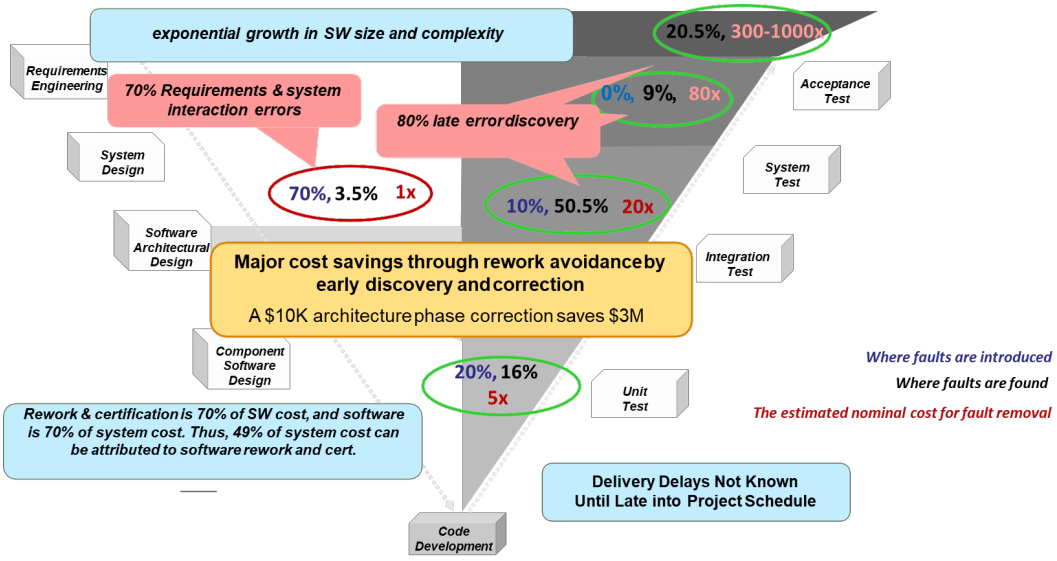

Figure 1: Impact of Software Rework on System Cost

Silicon-related verification and software development costs are growing rapidly, and are now more than 60% of the overall system cost. This is especially true at smaller geometry nodes of 16nm and below [3]. Additionally, changing industry dynamics between silicon IP providers, chip suppliers, system integrators and system houses are requiring more intricate co-development across company borders and across hardware and software.

For software projects, industry average bug defect rates are around 1-25 per 1,000 lines of code, 10-20 in internal QA and 0.5 in shipped code at Microsoft, and 3 bugs per 1,000 lines in internal QA in cleanroom development and 0.1 in shipped code [4]. Additionally, as there are a non-zero number of vulnerabilities per 1,000 lines of code in applications like Google Chrome, FireFox, the Linux Kernel, OpenSSL, Python and PHP, many defects and vulnerabilities are not discovered.

These bug rates may sound small, but they add up very quickly in large, complex systems, such as a modern automobile. As an example, let's assume that over 5 years there are 0.05 discovered vulnerabilities per 1,000 lines of code and 0.15 undiscovered vulnerabilities per 1,000 lines of code. Since a modern automobile can have over 50 million lines of code, this means that up to 2,500 vulnerabilities will be discovered in the platform over 5 years. Additionally, there may be as many as 7,500 undiscovered vulnerabilities, a significant opportunity for zero-day attacks.

In many industries, such as automotive, production dates are fixed. When projects fall behind schedule, it is often difficult to cut features or increase staffing to make up the difference. A simple solution is to cut corners on safety and security. Unfortunately, this is quite common, and can open the door for attackers. In the automotive industry, for example, the last few years have brought to light a number of security holes that allow attackers to remotely control a vehicle. The highest levels of safety and security are difficult to achieve but are attainable through proper system design and development processes.

II. Safety and Security Considerations for Software

In an era of ubiquitous connectivity and increasing system complexity, it is no longer possible to attain any reasonable level of safety without strong security. A well-designed system architecture can account for this complexity and increase safety and security.

The most effective approach to developing a safe and secure system begins with decomposing the system into separate components. Each component is analyzed for its size, complexity, resource needs, and safety and security requirements. The components that will be certified for safety or security must be designed to be very small, simple, and isolated from other components in the system. In turn, these components will be much simpler to test and certify. Non-critical components, on the other hand, may be large and complex, as they will not need to be certified.

Proper separation of the safety or security-critical components from the non-critical components is critical. Without this isolation, the system design can be compromised. A Separation Kernel guarantees resource isolation between application-level programs, allowing a system architect to separate critical device drivers and applications from applications that may be less safe or secure. Additionally, a Separation Kernel provides the isolation required to safely and securely run guest operating systems alongside critical real-time applications. Figure 2 shows an example of such an architecture.

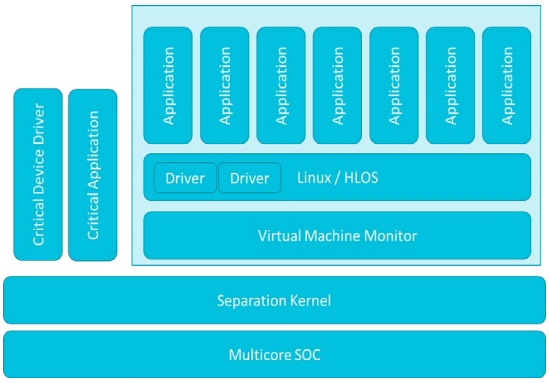

Figure 2: Separation architecture, including virtualization

Security is multi-faceted and must address more than just architectural design. Functions such as secure software boot, key storage and infrastructure, software update systems, and cryptography for data at rest and in transit are critical parts of a safe and secure system. Figure 3 shows how the chain of trust is built across the software stack from bootloader, through operating systems, device drivers, middleware and user applications.

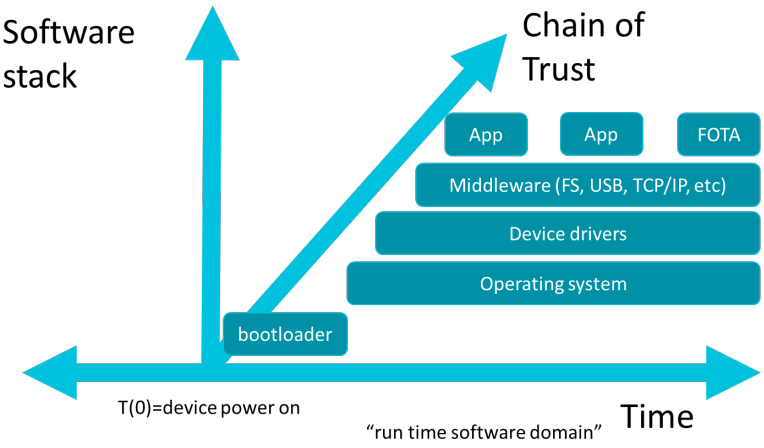

Figure 3: Chain of trust in a typical software stack

III. Safety and Security Considerations for Hardware

For hardware, safety and security have to be linked to the traditional design/verification and implementation flows, as shown in Figure 4, to include safety mechanisms and meet the hardware metrics according to ASIL standards. Safety metrics, Power Performance Area (PPA), verification time and automation have to be considered.

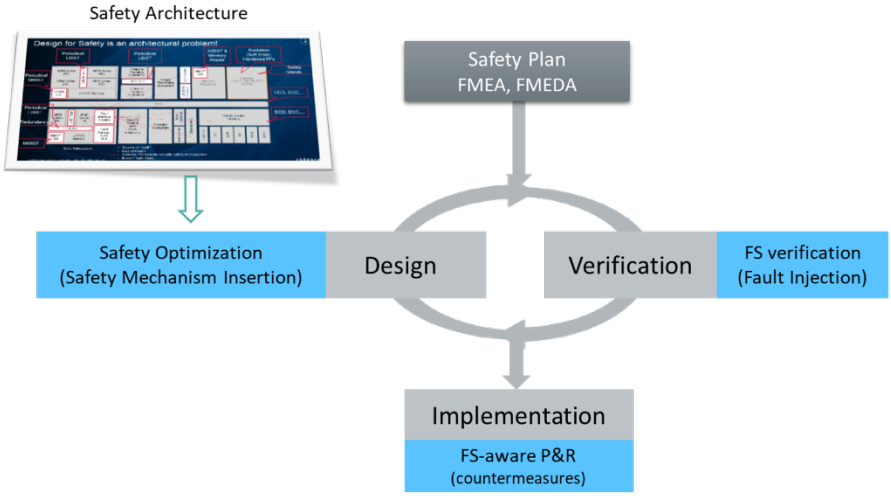

Figure 4: Hardware considerations for safety and security

A safety plan captures functional requirements and failure modes, estimates and distributes FIT, and maps failure modes to safety goals. Requirements, planning, optimization, and execution of failure campaigns interacts with safety optimizations in design and functional safety aware place & route in implementation.

For further illustration, Figure 5 shows an example of campaign management execution and analysis for hardware faults. From a central cockpit, the flow of campaigns is controlled, campaign sessions and steps are executed. The

result of fault session is collected from actual execution in dynamic engines like simulation and emulation, and accumulated, illustrating to the user the various attributes like classification of an detection.

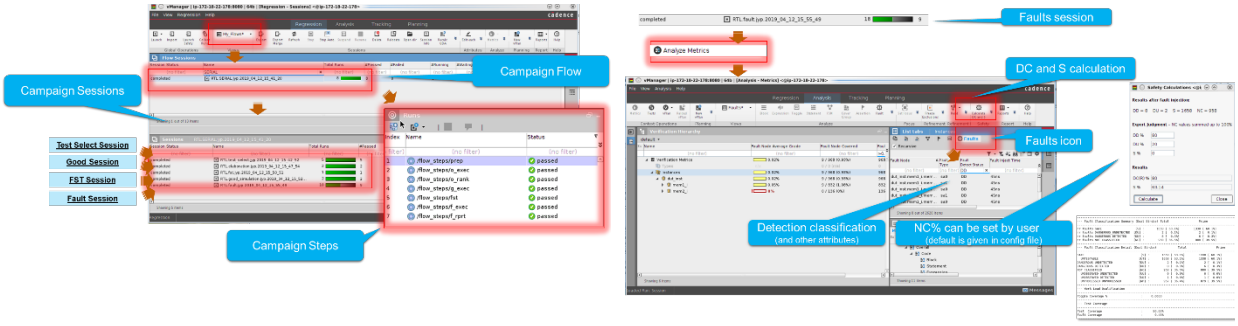

Figure 5: Fault Campaign Management execution and analysis example

There are many opportunities to optimize for safety, security, and performance when software and hardware designers work together early in the process of system design. There are some functions that may be performed in hardware, rather than software, such as lock step cores, ECC memory, Spectre mitigations, and built-in self test (BIST). During the development phase, this reduces the amount of software that needs to be designed, written, tested, and certified, and reduces the number of bugs. It also reduces the software's need for hardware resources.

IV. Achieving Shift Left with Early Hardware-Software Integration

Figure 6 and 7 illustrate how during the development cycle, designs with compute sub-systems, application specific components, interconnect and peripherals of various kinds are mapped into dynamic verification engines for simulation, emulation and prototyping.

Early in the design flow, virtual prototypes at the transaction level can execute the same software that later is loaded on the board using register accurate descriptions in SystemC. They are functionally accurate but do not represent timing in full detail. For performance analysis, timing information or even cycle accuracy can be added, but users have to carefully consider tradeoffs between accuracy and performance of the model.

At the register transfer level (RTL), simulation, emulation and prototyping provide different trade-offs between execution performance, debug insight, accuracy and bring-up time (i.e. the time to when the execution model runs functionally correctly). Simulation and emulation are initially focused on hardware verification, with prototyping focused on software development. Once chips are back from production, software development can proceed on the actual silicon.

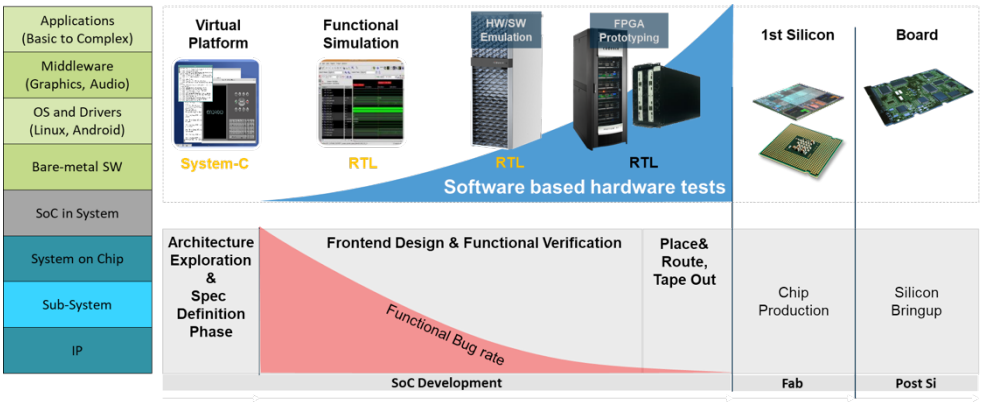

Figure 6: Dynamic execution engines representing hardware for software development

As illustrated in Figure 7, several layers of software can be executed on the compute subsystem, starting from hardware abstraction layers, to operating systems, virtualization layers, drivers, middleware and user applications. This allows for simultaneous and detailed debug and analysis of simulated or emulated hardware, as well as early production software.

Some safety standards, such as ISO 26262 for the automotive industry, recommend using fault injection to improve safety. For example, we may inject a hardware fault and analyze how the software responds. After silicon is available, however, the options for fault injection become more limited. Moving production-grade software development earlier in the hardware design process can make fault injection much easier. Running a production-grade RTOS on a simulated model of the hardware can simplify the fault injection process, as well as increase the range of tests that can be run.

It is also important that hardware verification is performed with a safety and security critical RTOS that will be used in production. It is often the case that prior to silicon being available, the only software that is run on a new SoC design is either bare metal or a simple Linux distribution. While this can check that the hardware is working at a basic level, there are some issues that may evade this type of testing. Because an RTOS will exercise the hardware in a different way than Linux will, running an RTOS on pre-silicon simulators and emulators can help hardware designers uncover unique issues.

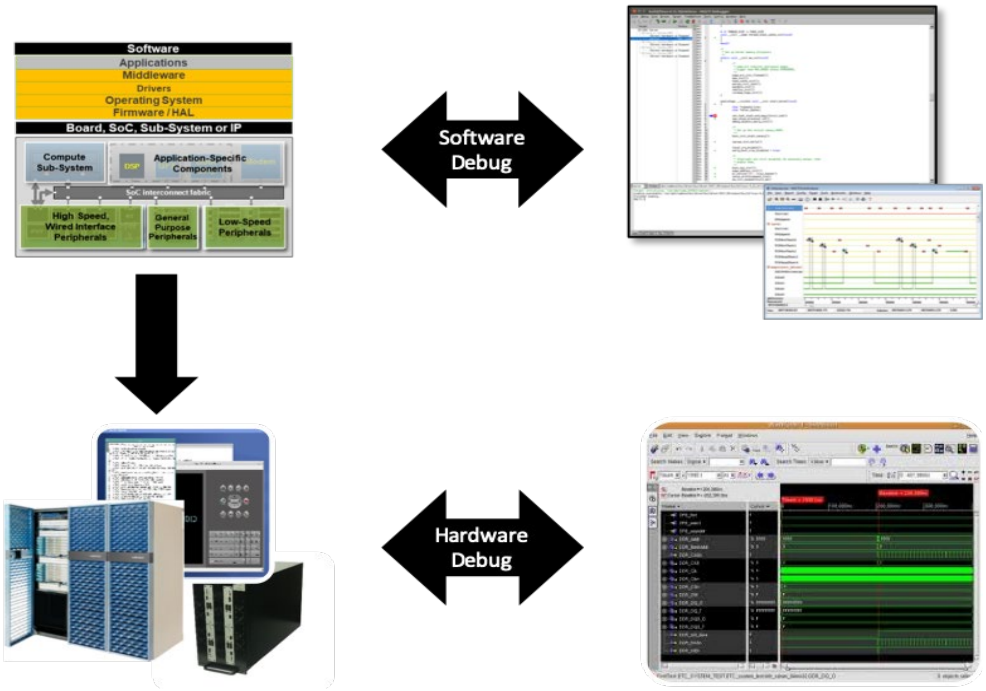

Figure 7: Hardware/software co-verification and debug

V. RESULTS

Cadence Design Systems and Green Hills Software have been working together to enable SoC and system designers with the tools necessary for hardware and software co-development and co-verification. Figure 8 shows an Arm-based virtual platform using a Cadence Virtual System Platform based on Xcelium SystemC simulation. The virtual platform is executing an OS – in this case Linux – for early software bring-up. The virtual platform is connected to the Green Hills Multi IDE and debugger for software development.

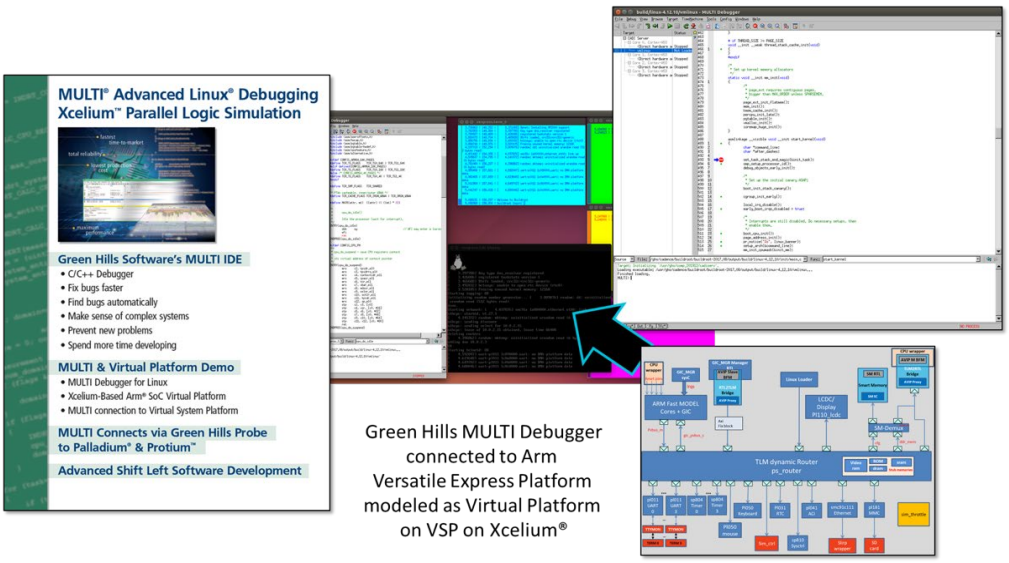

Figure 8: Example virtual platform booting Linux, connected to Green Hills Software's MULTI debugger

This environment allows software teams to use production-grade C/C++ development tools and RTOS on verification engines prior to silicon availability. As a result, the entire software schedule can experience a "shift-left", where pre-silicon development engines, virtual prototyping, RTL simulation, emulation and FPGA-based prototyping can be used before first silicon to develop and debug software applications, middleware, RTOS and device drivers. They even enable more developers to simultaneously develop/test code after first silicon, as early-revision boards are often very scarce. Safety certified RTOS and C/C++ compilers allow system designers to take into account safety and security from the very beginning of the design process, allowing more complete software enablement on first silicon once it is available. Processor manufacturers can launch first silicon pre-tested devices with production-grade RTOSs.

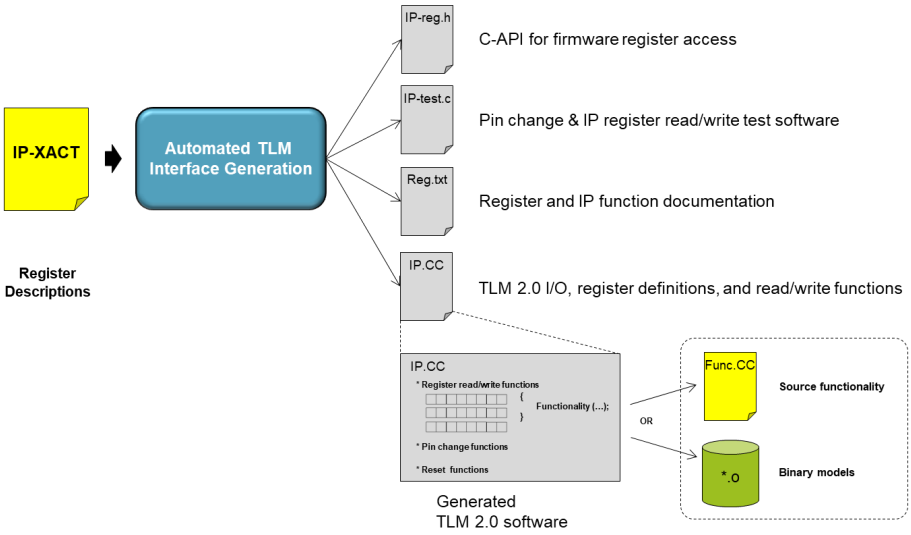

Figure 9: Automated TLM Model Creation

A key challenge in the development of such a virtual platform environment is the development of Transaction-level Models (TLM) representing the blocks in the design on which the software runs, as well as the assembly of those blocks reflecting the design topology of processors, peripherals and interconnect. The industry is offering a variety of tools that allow efficient model generation and assembly. Figure 9 shows the flow to automatically generate the code skeleton for SystemC TLM models from IP-XACT descriptions. Items like C-APIs, test software, interface and

functional documentation and I/O, register definitions and read/write functions are automatically created, allowing the user to focus on adding the actual functionality of a block that has to be developed.

VI. Outlook

By engaging in early hardware/software integration, SoC design teams benefit from deeper visibility into software/hardware interactions before silicon is available. This will result in increased quality of hardware designs, potentially saving millions of dollars by reducing silicon re-spins, as well as improving performance, safety, and security of the final hardware design.

Improvements in hardware quality, performance, safety, and security provide major benefits to a system designer. The more functionality that is implemented in hardware and the smaller the errata, the less software needs to be developed. This, coupled with an earlier software development start time, can assist system designers in having safe, secure, production-grade systems ready as early as possible.

References

[1] Error Cost Escalation Through the Project Life Cycle, NASA, https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20100036670.pdf, downloaded November 2019

[2] US Army Aviation

[3] IBS 2018, Volume 27, No. 7, Design Activities and Strategic Implications.

[4] Code Complete, Second Edition by Steve McConnell