A Low Maintenance Infrastructure to Jumpstart CPU Regression and Performance Correlation

Thomas Soong
thomas.w.soong@intel.com

Chenhui Huang
chenhui.huang@intel.com

Christopher Browne
christopher.browne@intel.com

Intel Corporation
1300 S MoPac Expy,
Austin, TX 78749

***Abstract*- Surge is a low-maintenance RTL jumpstart mechanism that works on any architectural boundary for CPU simulation. Surge's design makes it resilient to frequent RTL rewrites and design changes. It has a small code base to maintain, supports multicore, and is compatible with all soft IP configurations. Legacy RTL jumpstart mechanisms relied on a specialized SVTB module with a detailed mapping of every architectural state to its microarchitectural (µArch) counterpart. The legacy jumpstart module must understand every relevant signal path, register shadow copy, and array configuration; this results in monolithic code that frequently breaks due to µArch changes. Surge utilizes the CPU's functional high-level model and legacy CPU power state feature to abstract out the µArch dependencies. Under Surge, any test will first run very fast under E-core's [1] (formerly known as Intel Atom) functional model to the point of interest then seamlessly migrate to RTL simulation. As a result, Intel E-core validation skips the RTL simulation cycles spent on test setup and E-core performance team can more reliably complete performance correlation simulation.**

I. Introduction

Pre-Silicon RTL simulation is comprehensive and consequently infamous for being slow. During the development of Intel E-core processor, we created Surge, a new versatile mechanism to jumpstart the CPU RTL simulation into any architectural boundary. Surge has many immediate benefits. First, the Pre-SI validation team can considerably shorten the total CPU simulation cycles by jumpstarting directly into the critical section of any test content. This ability is especially useful for IA32 features that have long preamble routines, such as Intel's Software Guard Extension (SGX) [2] and Virtual Machine Extension (VMX) [3]. Second, Surge provides a more streamlined jumpstart to maximize efficiency for correlating E-core's RTL performance data against our high-level performance model.

The biggest challenge of jumpstarting any RTL simulation is to map the architectural states with their corresponding microarchitectural equivalent and inject them into the simulator. Traditional jumpstart mechanism previously used by E-core team has a hardcoded mapping of architectural states to µArch signals or arrays. This complicated jumpstart tool requires routine updates and regular debug to keep up with E-core's frequent design changes. For example, mundane design optimization that introduces new shadow copy of control register will break the legacy jumpstart unless it is made aware of the change. The maintenance cost can be further exacerbated when RTL undergoes significant rewrite in ambitious project even for legacy features. The high maintenance cost also takes bandwidth away from traditional validation work.

Under Surge, any architectural test will run very quickly under E-core's functional model to the point of interest, then seamlessly migrate to RTL simulation. As a result, E-core validation skips the slow RTL simulation cycles spent on test setup and the E-core performance team can more reliably complete performance correlation simulation. Surge is resilient to frequent RTL and firmware changes. Surge's resiliency stem from its small low-maintenance code base that takes advantage of existing processor ISA features technologies, such as power management sleep states and system management interrupt as well as validation collateral, such as our system agent (uncore) BFM and CPU functional model. Surge can migrate a high-level functional simulation run to a low-level RTL simulation without maintaining a complex architectural to microarchitectural mapping. It is the current tool used by E-core to increase simulation performance by up to 40% on pre-silicon regression run. It is used by E-core to conduct performance correlation on captured silicon trace without any disruption caused by RTL changes during project development.

II. Building Surge For Faster CPU Simulation

There are four main ingredients necessary to successfully jumpstart a trace mid simulation onto a CPU RTL simulation:

1) A fast high-level functional model of the CPU
2) Ability to capture and migrate all architectural and microarchitectural states from each core
3) Ability to capture and migrate all states in the uncore BFM
4) Synchronize the ISA golden reference model with the jumpstart state

E-core maintains a fast functional high-level model of its CPU design called FastSim for prototyping, development, and testing microarchitectural features, which facilitates efficient sharing of design, debug, and validation components across E-core projects. Jumpstarting CPU RTL simulation to any macro boundary from FastSim has always been a goal on our projects. This would significantly shorten pre-silicon simulation time by shortcutting the setup code instructions common in our test generators.

To easily capture all architectural and microarchitectural states from FastSim and restore them on CPU simulation, Surge takes advantage the CPU design's native power management C6 sleep feature [4], where the processor state is saved into a dedicated Static RAM or SRAM before voltage to core is reduced to zero. When exiting C6, the processor state is restored back from the SRAM. By injecting a C6 power state request into FastSim, Surge can initiate a transition into the C6 power state on any architectural boundary. As part of the C6 power state specification, all the necessary IA32 architecture states and microarchitectural states are saved into the C6 SRAM. The data in the C6 SRAM is later used to perform a CPU state recovery in the Surge jumpstart flow.

Both FastSim and CPU simulation shares the same uncore BFM. Migrating the uncore states from FastSim to core simulation is done through a save and restore of uncore states across both environments. The uncore BFM was updated to produce an external log containing all its transaction request. To restore, the CPU simulation's uncore BFM states, Surge replays FastSim's request log before the start of the simulation. This simple addition of logging and replaying uncore transactions allowed us to migrate uncore states across both environments.

A key component for CPU core validation is the ability to check architectural states against the golden ISA reference simulator, Archsim. Unlike FastSim, which models the E-core microarchitecture, Archsim models only the x86 instruction set. Both Archsim and CPU RTL simulation need to be synchronized after Surge jumpstart. Surge triggers Archsim's software save subroutine when the C6 power event is delivered in FastSim. After CPU jumpstart is completed, Archsim's states are recovered with its software restore routine to keep both simulators consistent.

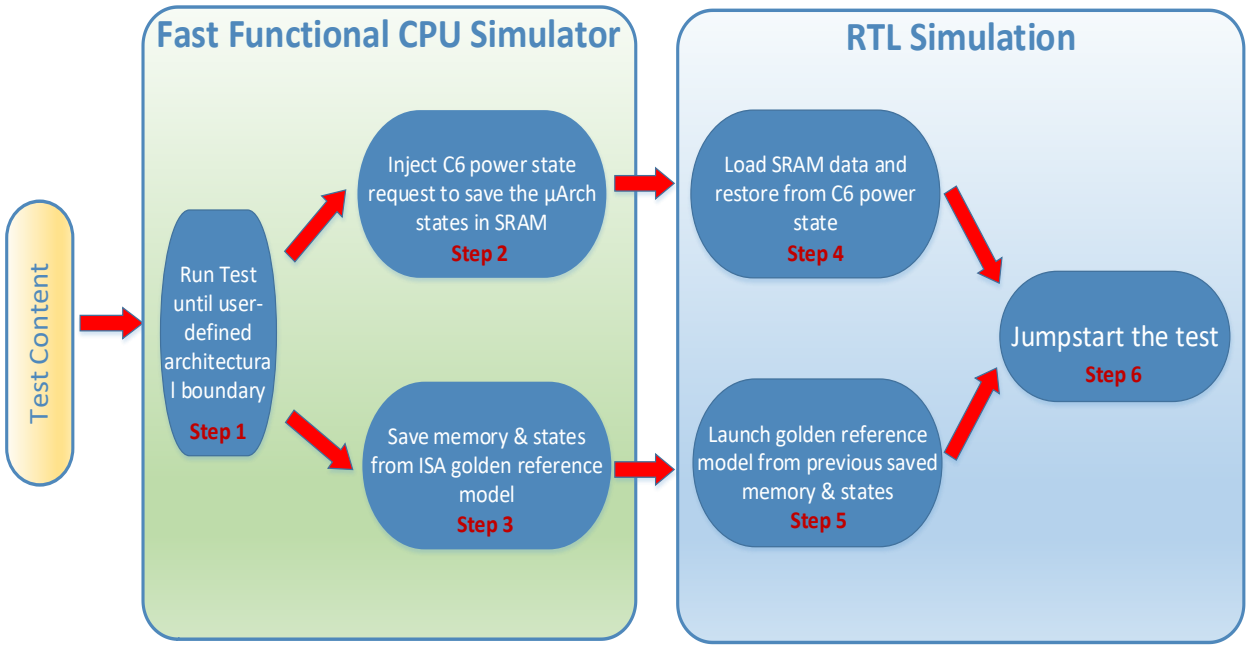

Figure 1: Flow chart of how SURGE migrates FastSim state to jumpstart CPU simulation

To use Surge, the user first identifies an architectural boundary point, normally a specific linear instruction pointer (LIP) from the test content. FastSim simulates the test content up until the boundary point and jumpstarts the RTL simulation afterwards. All the preparation work for the jumpstart is done in FastSim by reusing existing validation collateral and repurposing legacy architectural features. As shown in Figure 1:

Step 1: The test content is run until the user-defined boundary point.
Step 2: After the boundary point, FastSim injects a C6 power state entry command. The CPU model executes the C6 save flow, which saves all critical states into the C6 SRAM. Upon C6 save completion, FastSim exports the C6 SRAM contents to a file.
Step 3: In parallel with ***Step 2***, Surge invokes Archsim's save routine to snapshot its architectural image. This will allow Surge to keep RTL and Archsim in sync after CPU jumpstart is completed.
Step 4: In RTL simulation, Surge invokes SystemVerilog testbench code to inject the previously saved C6 SRAM data into RTL's physical C6 SRAM. Subsequently, an injection power management module triggers a C6 restore which loads all the architectural and microarchitectural core states back into the CPU simulation. A monitor event is triggered at the end of the C6 restore, and CPU simulaton will start executing from where FastSim left off.
Step 5: In parallel with the C6 restore, Archsim is restored using the saved Archsim image.
Step 6: After this point, the test will continue on like a normal CPU VCS run.

The deployment of Surge allowed the E-core validation team to easily migrate long IA32 tests to the CPU simulation model. More important, only the critical sections of the test content are exercised on the CPU model, allowing the user to expedite simulation results by skipping the time-consuming test setup code especially for complex IA32 features such as Intel's Trusted Execution Technology (TXT) [5].

III. Extending Surge for Architectural Trace

To validate the CPU's performance in pre-silicon, E-core's performance team runs "snippets" of real-world content on the RTL model as well as the performance timing model to compare their respective IPC. These snippets are small windows of instructions taken from larger benchmarks or customer workloads. Because they necessarily start in the middle of execution, they are unlike normal validation content and require us to preload both architectural (e.g. the instruction pointer) and microarchitectural state (e.g. cache contents) into the RTL model prior to simulation.

The initial architectural state for the snippet is available in an ArchXML file. Historically, a homegrown script called Perf_JS was used to inject the architectural state into the model. The legacy script parses the ArchXML file and performs signal injections into the RTL at runtime. However, this requires Perf_JS to encompass all the complexity of mapping architectural state to project-specific microarchitecture. Whenever the RTL changes, Perf_JS needs to be updated to reflect new structures and locations of architectural state. The onus is put mostly on RTL designers, who must update things like signal paths and bit swizzling schemes in Perf_JS before they can commit RTL edits.

Surge replaced Perf_JS to save this cost. Instead of maintaining a mapping of architecture states to their corresponding RTL signal or shadow copies for injection, Surge recovers these states from the ArchXML file. ArchXML contains only architectural states which are too high level to be loaded directly into FastSim. A more robust method was needed to simplify the complexity of mapping architectural state to processor-specific microarchitecture which FastSim can understand. Surge solved it by adopting an Archsim run followed by a FastSim run to create the C6 SRAM image needed to jumpstart the CPU simulation from the snippet initial states.

System Management Interrupt [6], SMI, allows IA32 processors to enter System Management Mode (SMM). This transition saves most of the important architectural states into the reserved memory state save area known as System Management RAM or SMRAM. However, some architectural states, which are critical for performance correlation, are not present in the SMRAM. Surge utilizes a custom SMI handler to save these extra states through simple x86 assembly instructions, such as RDMSR and XSAVE. Figure 2 illustrates a sample SMI handler which contains both a save and a restore routine. The save routine is executed in standalone Archsim after loading the snippet and saves all important extended architectural states into an unused memory region. The restore routine is executed in FastSim and does the opposite, by restoring all the extended states back to the processor as well as everything stored in the SMRAM. The restore routine reads the MSR value from memory and uses WRMSR instructions to restore these states. The XRSTOR instruction restores all the extended states back the processor. Finally, the SMI restore handler executes the RSM instruction to recover everything stored in SMRAM back to the

processor. The restore routine is executed in FastSim. Figure 2 details the steps for Surge to jumpstart from the architecture snippet to the CPU VCS simulation.

Step A: Surge combines the snippet memory image with the SMI handler to generate a new memory image. Archsim loads the memory image together the ArchXML file in standalone mode. A SMI event is injected into Archsim immediately after the memory image and ArchXML file are loaded, which triggers the SMI handler's save routine. At the end of save routine, Archsim will dump its entire memory containing both SMRAM image and additional save states from the PSMI save routine into the "*smi.ami*" file.

Step B: FastSim loads *smi.ami*, triggers a SMI immediately, and executes the SMI handler's restore routine. The restore routine enables FastSim to bootstrap its architectural states from the ArchXML file while having all of its microarchitecture states consistent and properly initialized. Upon exiting the SMI restore handler, a C6 sleep event is injected to trigger the C6 save flow.

Step C: The final step is the same as the second half of Figure 1. In short, SVTB code loads FastSim's C6 SRAM data into the CPU. The CPU simulation initiates C6 restore and continues with the trace simulation.

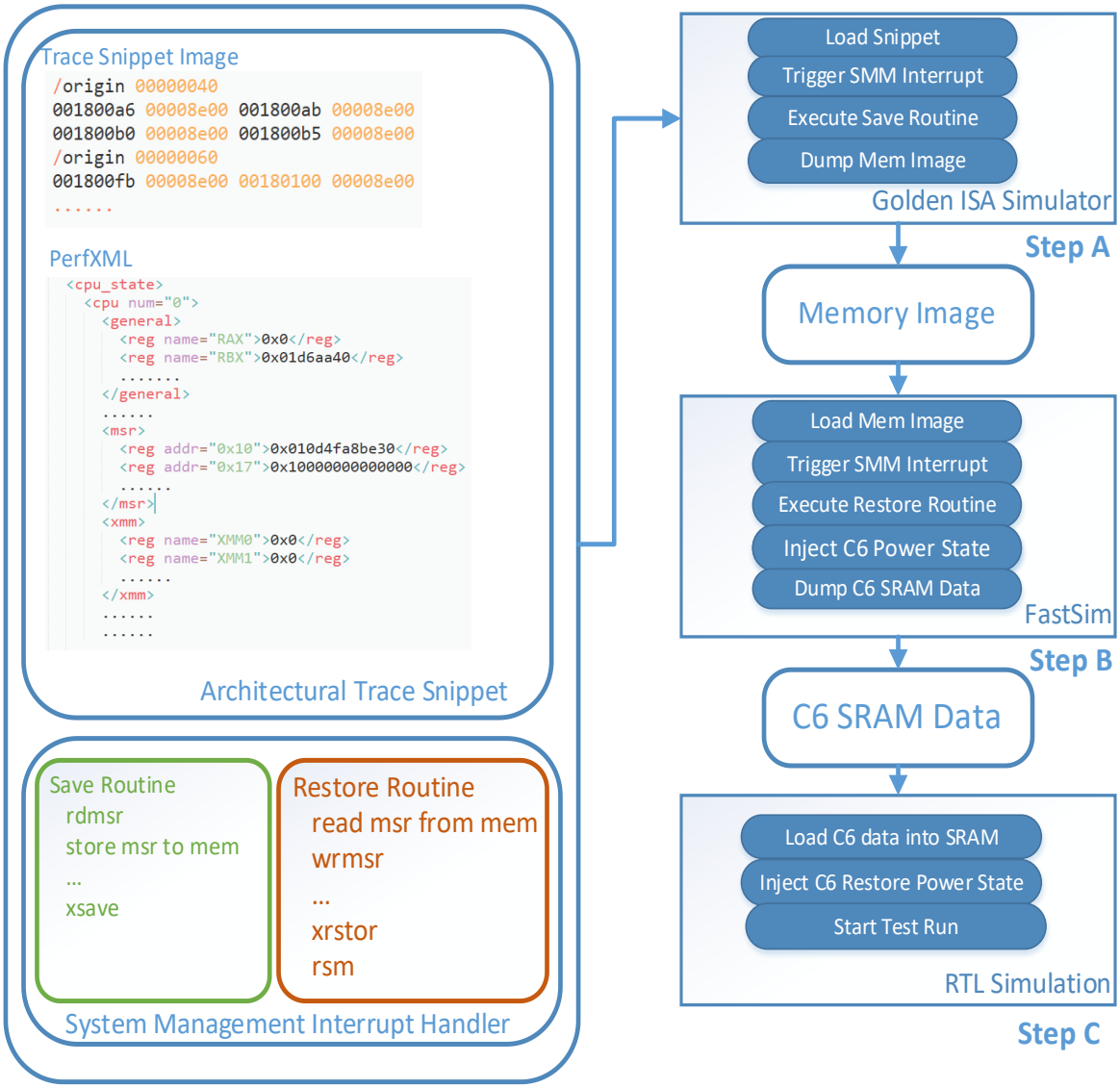

Figure 2: Surge jumpstart of CPU simulation with architectural snippet

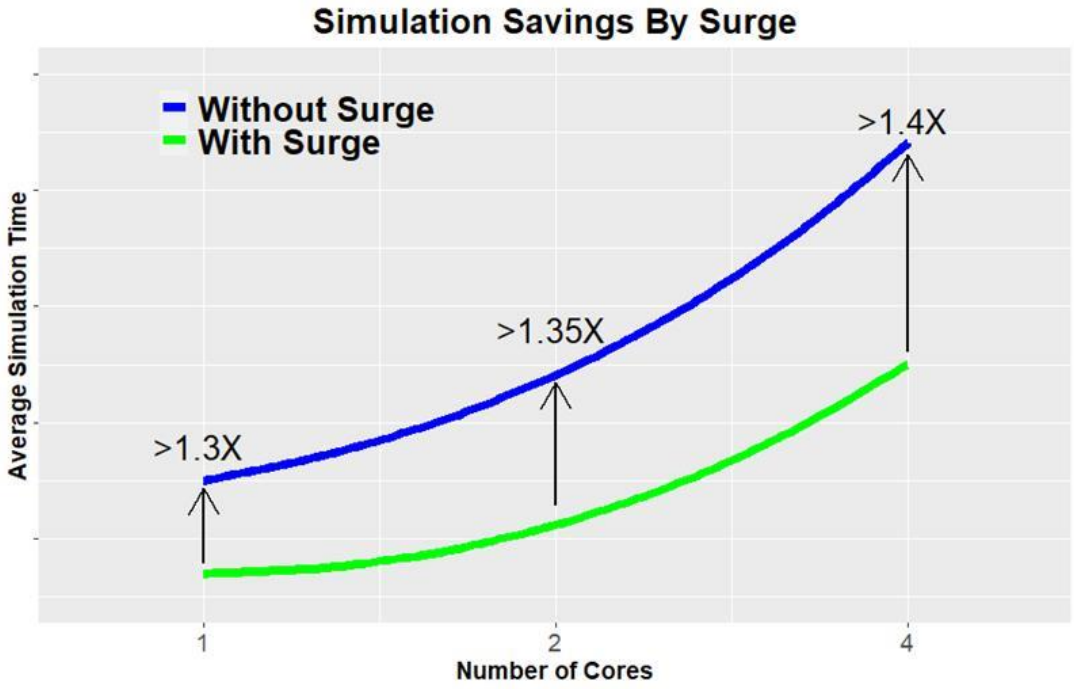

Figure 3: Surge average simulation savings

IV. RESULT

In general, Surge provided significant increase to the overall test throughput of E-core's pre-silicon simulation. As mentioned before, Surge allows RTL simulation to skip over all the uninteresting test configuration routines, such as memory initializing, mode transitions, I/O programming, etc., and start immediately executing the random code block. Figure 3 illustrates the simulation savings compare to traditional full end-to-end test runs. The simulation result in Figure 3 with Surge also includes the time spent in FastSim, which is insignificant given that FastSim runs thousands of times faster than RTL simulation. The data has been abstracted to protect and not reveal detail about E-core's proprietary testbench environment; however, the general trend and relative performance in the graph remains accurate. On average, E-core's simulation performance increased by 40% on exercising the four core CPU module. The percentage of savings from Surge increases with the number of cores being simulated. This is expected as multicore content contains various barriers and cross-core synchronization loops that increasingly slows test progression as core count rises. Surge allows the testbench to jumpstart pass the bulk of those test setup sequences.

The E-core validation team uses a variety of proprietary architectural test generators to stress all areas of the processor design. While the specific details of E-core's test generators are outside the scope of this paper, it is worth noting that Surge is agnostic to the architectural instruction sequences being skipped and as result works with all variety of architectural test generators. Each test generator simply needs to define the boundary point where Surge need to trigger the migration from FastSim to RTL simulation. The boundary point can be as complex as a combination of architectural test sequence/state or as simple as a particular assembly jump label.

E-core's performance team have been running with Surge enabled for almost 2 years and have not had to look back. Surge is the current state-of-the-art tool to conduct E-core's performance correlation, replacing Perf_JS. For scale, E-core runs 3 performance regressions per week with a rotating set of ~2700 trace snippets per regression. Throughout the years only minor tool bugs were discovered and quickly fixed with no disruption to pre-silicon validation and performance correlation work despite constant changes in E-core's RTL and feature set.

While most discussion in this paper has focused on simulation, Surge has proven to be easily translatable to emulation as well. Given the ability to preload the C6 SRAM through ordinary methods and by injecting a single signal to spoof the cold boot state to C6 restore the same Surge flow from pre-silicon simulation also works on emulation platforms. This enabled our performance correlation team to run real-world content at high-speed allowing us to practically compare performance and power estimates for full-duration sections of benchmarks on our performance timing model versus RTL.

V. Understanding Surge's Tradeoffs

As with any jumpstart mechanism, the validator should only utilize Surge on content that make sense and understand the tool's limitation. Any critical test sequence should not be skipped over with Surge. Surge relies on FastSim to correctly emulate the RTL behavior; however, the validator should understand that FastSim is not a cycle accurate CPU model. Any test content that relies on CPU counter for program ordering or any test sequence attempting to align a microarchitectural race condition cannot expect accurate result from Surge jumpstart. In general, when utilizing Surge to expedite simulation performance it is best matched with random test generators and test content that are microarchitecturally agnostic.

Surge's heavy reuse of FastSim, SMM, and C6 power state allows the tool to have a very small footprint and negligible maintenance cost. Aside from the two dozen lines of SVTB code utilized to inject the C6 SRAM and trigger C6 restore from the power management unit nothing else in Surge is tied to project's RTL implementation. Of course, FastSim's modelling needs to be accurate as well as SMM and C6 functionality needs to be healthy for Surge to be operational. From our experience, these dependencies have not been an issue. Both SMM and C6 are legacy features with robust regression suites that guard against new bugs. Meanwhile, FastSim modelling has always been a high priority task during the initial phase of all E-core projects.

VI. Closing Remarks

Surge gives validators a straightforward way to shorten CPU simulation through a robust jumpstart mechanism. While the saving on CPU simulation varies by test content, on average 30% ~ 40% runtime savings is achieved. Surge is E-core's state-of-art tool for launching performance correlation validation on multiple projects. It streamlines E-core's performance correlation tool through its support of code snippets and multicore support. By reusing legacy architectural features, C6 and SMI, as well as leveraging existing validation behavior models kept the code base simple and resilient to frequent RTL changes. As a result, throughout the years only minor tool bugs were discovered and quickly fixed with no disruption to performance correlation work despite constant changes in E-core's RTL. Surge's low maintenance and small code base allows it to be easily deployed across the E-core projects.

VII. Acknowledgements

The authors would like to thank the people who encouraged Surge's development and many others who provided technical guidance including Loucas, Stephen, Ramya, Riccardo, Ben, and Chris.

VIII. References

[1] "Intel Atom® Processor Family," [Online]. Available: https://www.intel.com/content/www/us/en/products/details/processors/atom.html

[2] "Intel® Software Guard Extensions (Intel® SGX)," [Online]. Available: https://www.intel.com/content/www/us/en/architecture-and-technology/software-guard-extensions.html

[3] "Intel® Virtualization Technology (Intel® VT)," [Online]. Available: https://www.intel.com/content/www/us/en/virtualization/virtualization-technology/intel-virtualization-technology.html

[4] "Power Management - Technology Overview," [Online]. Available: https://builders.intel.com/docs/networkbuilders/power-management-technology-overview-technology-guide.pdf

[5] "System Management Module (SMM) User Guide," [Online]. Available: https://www.intel.com/content/dam/support/us/en/documents/server-products/server-systems/R2600SR_System_Management_Module_User_Guide.pdf

[6] "Intel® Trusted Execution Technology (Intel® TXT) Overview," [Online]. Available: https://www.intel.com/content/www/us/en/support/articles/000025873/technologies.html