2022 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION INDIA

A scalable framework to validate interconnect-based firewalls to enhance SoC security coverage

Ashutosh Mishra
ashutosh.mishra@intel.com
Suresh Vasu
suresh.vasu@intel.com
Intel Technology India Pvt Limited, No 23/56-P, Outer Ring Road, Bellandur, Bengaluru – 560103

***Abstract*- Access control in IOT based SoCs is typically implemented using firewall design architecture in NoCs. Firewall structures in SoC are broadly used to control the access permissions of various initiators to different memory regions (both DRAM and On-chip SRAM) and different target peripherals in the MMIO space. Often these firewalls are positioned closer to the target interfaces and assets that needs to be protected. In the SoC under test the firewalls were uniquely positioned near every initiator IP. This gave rise to a lot of permutations and combinations to be tested. Complicated programming rules and frequent design updates added to the complexity. As hardware security bugs can become catastrophic, to ensure a complete security coverage we came up with a scalable framework from scratch to address these issues.**

I. INTRODUCTION

Emerging technologies have an enormous capacity to enrich our lives and change the world. As we get connected through devices more than ever before, privacy and security of the data becomes extremely crucial. But these technologies must be built on a strong foundation of security and trust. Securing technology products is one of the industry's most pressing and challenging goals. As hackers come up with new ideas to breach software security and most importantly hardware assets through adverse side channel attacks [1], it becomes very crucial to ensure that these new security technologies do not pose any risk. As a system validator one must analyze and predict all possible attack scenarios for the product. An area that is extremely critical to validate is access control-based hardware security features.[2]. Current SoCs or SoCs built on the backbone of arm architecture implement network-on-chip (NoC) based firewall design architecture to overcome these security vulnerabilities. But the design itself is so complex that there are a lot of possible permutations and combinations to be validated and it becomes a bottleneck for security verification. To achieve hundred percent security coverage there is dire need to develop a scalable validation framework.

The proposed solution is the first of its kind, built from scratch to address the above shortcomings. The automated framework is scalable across multiple validation platforms. It has enabled quicker turn-around time to develop tests, detection of quality hardware bugs thereby increasing the security robustness of the product.

.

II. VALIDATION CHALLENGES

Verification and validation are a big bottleneck in designing a System-on-Chip (SoC) which consumes a considerable amount of design efforts. The enormous impact of verification came from the fact that an SoC is required to be validated against several objectives, such as correct functionality, timing, power, energy consumption, reliability, and security in pre-silicon and post-silicon stages before it can be used in hardware devices. The huge complexity of

SoCs (tens of billion transistors are involved), as well as aggressive time-to-market, also contribute to even more growth of verification/validation efforts.

Figure 1: Motivation for a scalable security framework.

When it comes to security verification and validation [3], not only all the above-mentioned challenges are still in the picture, but the problem becomes even more challenging due to several reasons. The reason is that security is a generic term, and it is unclear how to achieve a secure design. There is no security specification or security verification plan to check the implementation against it. There are several security vulnerabilities, including information leakage, side-channel leakage, access control violations, malicious functionality, etc., that a security verification engineer should check Modern computing systems increasingly rely on hardware level protection to provide secure environments for critical software components. Hardware security architectures such as ARM TrustZone [4], Intel SGX [5], and IBM SecureBlue [6] aim to protect software even when the operating system is malicious or compromised. All these security architectures ensure that access control policy is not violated and only the secure agents have the authorized access to the protected assets on the platform.

The fundamental motivation behind realizing the framework is to address the following concerns as highlighted by Figure 1:

- Hardware security verification is becoming increasingly demanding and complicated.
- Hardware security bugs pose great risk and can be catastrophic.
- Foundational security – Like access control through firewalls becomes crucial.
- Frequent design updates make validation taxing.
- Lack of a scalable framework to validate across multiple platforms.

III. SoC Firewall Architecture

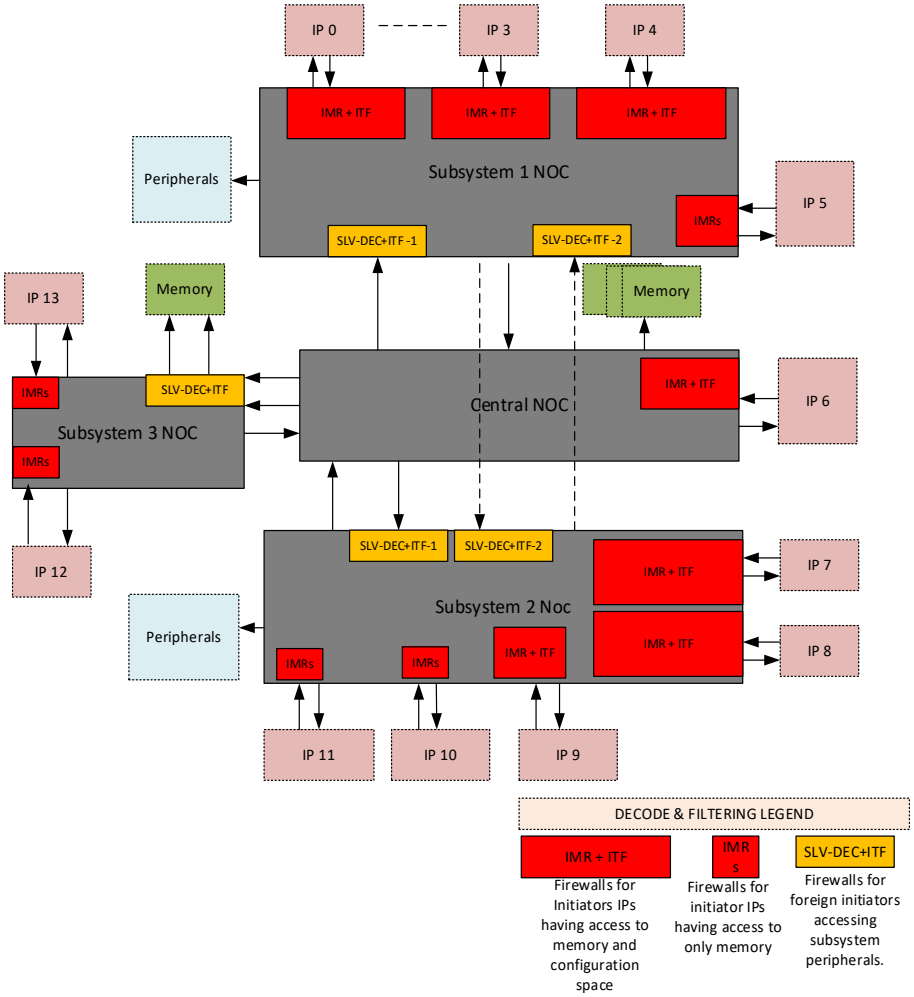

Figure 2: NoC firewall's architecture in SoC

Firewall structures in SoC are broadly used to control the access permissions of various initiators to different memory regions (both DRAM and On-chip SRAM) and different target peripherals in the MMIO space. The exact logic structure, positioning and programming details of the firewalls are dictated by the NOC architecture. The SoC Fabric as shown in Figure 2 is split into different hierarchical NOC based on physical positions of the initiators. Each NOC caters to different set of initiator IPs. Access permissions are implemented using combination of decode functions and firewalls in the master bridge and firewalls in the slave port. IMR (Isolated Memory Regions) split the memory space (DRAM and SRAM) into different regions along with specific access permission. Access from all memory initiators go through this permission check there by allowing or denying a particular access. To access permission, <IP>-Secure and <IP>-Normal are considered as different initiators though the memory access from them come out of the same physical interface. ITF (Initiator Target Firewall) structures are used to allow/deny access to the configuration regions by various initiators. Configuration regions are managed by some highly privileged initiators (Example CPU core). Configuration access from these initiators will go through the ITF block which will perform the filtering rules and then allow/deny the access.

It can be clearly concluded from the above architecture that positioning firewalls uniquely near the initiator IPs creates a lot of permutations and combinations of test scenarios. If we take a conventional approach and start creating directed scenarios to validate the firewalls it will obviously take a lot of time and the aggressive SoC tape out timelines cannot be met. The programming rules that governs the NoC firewalls also make the life of the validator messy, and it adds to the complexity.

IV. AUTOMATION FRAMEWORK

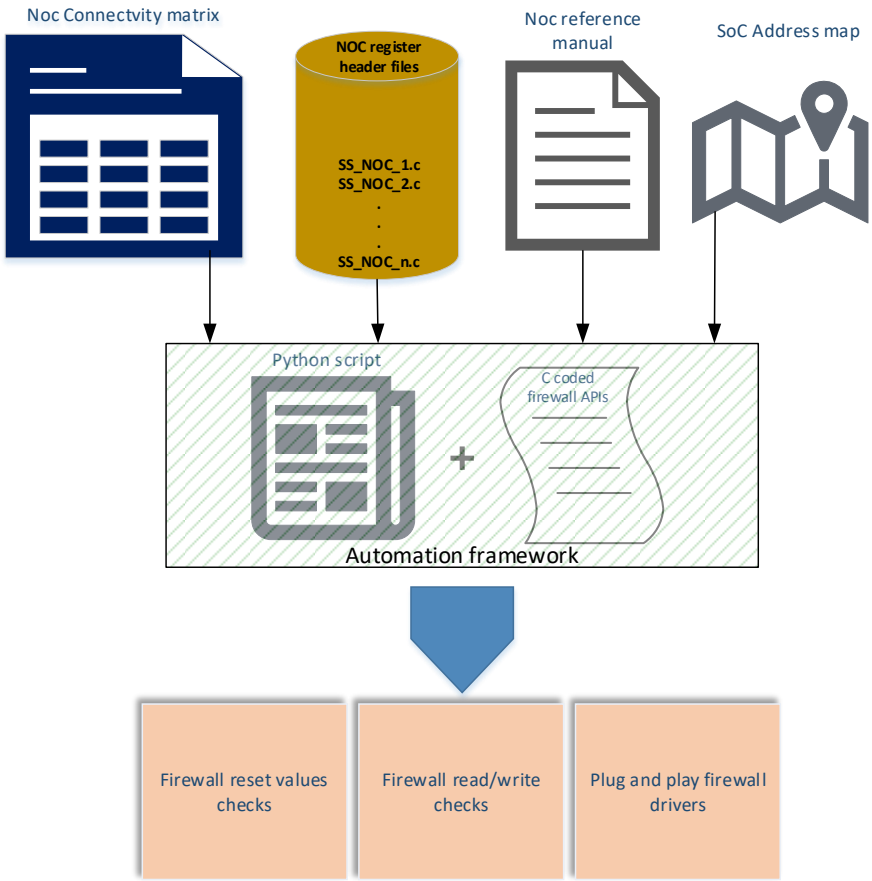

Figure 3: Scalable automation framework

The Automation framework as shown in Figure 3 is built from scratch which is a blend of python script and some uniquely defined C based APIs. The python script is responsible for checking frequent updates made into the NoC connectivity matrix, NoC register header files, address map, and finally prepare the list of all firewall registers present in different NoC register files. Once all the firewall registers are identified, the script develops automated tests like firewall reset checks and firewall register read/write checks. The framework also addresses the complex programming rules that governs the NoC firewalls for which we have coded the C based APIs. Although it was just a one-time effort in bringing up the APIs, it proved to be very helpful in covering a lot of unique scenarios in a very short span of time leading to a better coverage. The best thing about the C based APIs is that it can be used across multiple validation

platform – Pre silicon (Subsystem and SoC), Emulation, Platform Software and Post silicon. It can be easily extrapolated and used in SV/UVM environments with the help of DPI-C constructs. Below are some of the highlights/features of the framework:

- Automated firewall reset checks
- Automated firewall read/write checks
- Plug and play drivers
- Auto detects frequent design/address map updates through scripts
- One time effort in building C based APIs
- Drivers are easily extrapolated to SV-UVM environment through DPI-C export method

A. Examples:

1. If the validator wants to run a reset value check of all firewalls on NOC SS1, all he needs to do is call a simple C based function that is generated by the python script:
   - *SS_NOC1_register_reset_check ()*
2. If the validator wants to run a register read/write value check of all firewalls on NOC SS3, all he needs to do is call a simple C based function that is generated by the python script:
   - *SS_NOC3_register_reset_check ()*
3. If the validator wants to program a 4 MB region in SRAM starting from 0x180000000 for <IPx>initiator as secure read/write:
   - *firewall_config_FG_IMR (<IPx>, 0x180000000, 0x400000, SECURE_RW)*
4. If the validator wants to program a 2 MB region in DRAM starting from 0x200000000 for <IPy>initiator as secure read/write:
   - *firewall_config_FG_IMR (<IPy>, 0x200000000, 0x200000, SECURE_RW)*
5. If the validator wants to program the <Target_x> peripheral inside SS_NOC2 from <IPz> initiator port as write only:
   - *SS_NOC2_firewall_config_ITF (IPz, Target_x , WRITE_ONLY)*

All in all, the above framework makes the life of validator easy, saves a lot of time, helps in focused debugging, and improves the SoC security coverage.

B. Results:

Working on SoCs where the firewalls [7] were positioned near the target entities, we were easily able to create the directed scenarios and test the firewalls. But the same approach would not have been successful here because of the total number of combinations of scenarios to be tested in the current SoC. Moreover, it would have consumed a lot of time for developing all the scenarios, executing, and debugging them and closing the coverage. The estimated time that would have taken to complete the execution using the conventional approach is shown below in Figure 5.

When the automation framework was used, it significantly reduced the time taken to develop all the new scenarios as the python script was used to auto generate the sequences and C based APIs were used as a plug and play drivers in the already existing positive test scenarios which would now make the test as a negative scenario. With this approach we were able to hit some quality bugs and we were able to spend focused time in debugging the details as shown in Figure 4 This paved the path for a cent percent security coverage and were able to successfully close our security development lifecycle process [8] for the product on time.

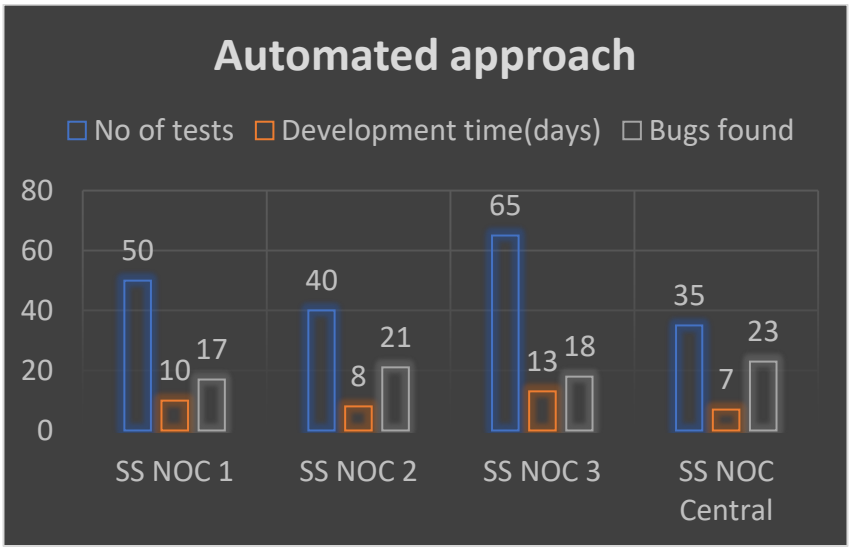

Figure 4: Results with Automated approach

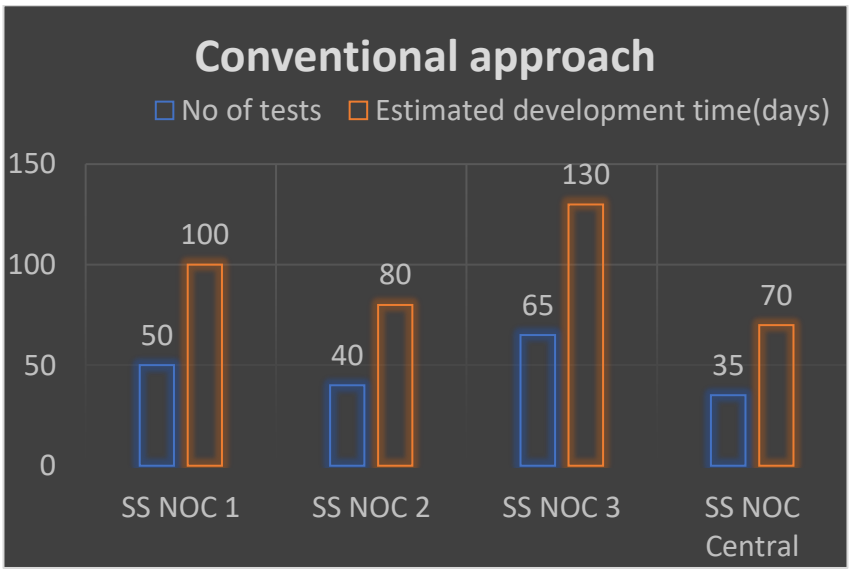

Figure 5: Estimated results with Conventional approach

The following can be inferred from the above graphs:

- Significant reduction in time taken to develop tests (90 %)
- Quality hardware bugs detected
- Enabled focused and detailed debugging
- Better security coverage and SDL closure.

V. Conclusion and Impact

The automation framework proved to be an effective tool to validate all possible firewall scenarios and enhance our SoC security coverage. It is scalable across multiple validation environments – Pre-silicon (Sub-system and SoC), Emulation, Post-silicon, and Platform-software. It paved the way for an effortless validation as lesser time was spent on developing test code and more time on debugging leading to detection of quality hardware bugs. The framework/codes can be easily referenced as was done in ROM code development to setup the initial firewalls pre-OS boot. The environment can easily be adopted and ported to future projects with similar NoC firewall architecture.

Acknowledgment

The work was greatly encouraged and supported by Nithin. M Kumar (Director of Engineering at Intel) right from the beginning to the end. We would like to extend our gratitude to members of the Intel Security team and Suresh Vasu (Validation architect at Intel) who provided insights and expertise that greatly assisted our work. Last but not the least, we would like to thank DVCON India committee for selecting our abstract and for providing us an opportunity to publish our work in this field.

References

[1] H. Ju, Y. Jeon, and J. Kim, "A Study on the Hardware-Based Security Solutions for Smart Devices," *2015 International Conference on Computational Science and Computational Intelligence (CSCI)*, 2015, pp. 833-834, doi: 10.1109/CSCI.2015.105.

[2] https://www.linkedin.com/pulse/hardware-security-verification-example-cwe-1267-policy-nordstrom/

[3] J. D. Guttman and H. -P. Ko, "Verifying a hardware security architecture," *Proceedings. 1990 IEEE Computer Society Symposium on Research in Security and Privacy*, 1990, pp. 333-344, doi: 10.1109/RISP.1990.63862

[4] Rick Boivie. SecureBlue++: CPU Support for Secure Execution. http://researcher.watson.ibm.com/researcher/ view_group.php?id=7253, 2012

[5] Intel Corporation. Intel Software Guard Extensions Programming Reference, 2014

[6] ARM Ltd. ARM Security Technology: Building a Secure System using TrustZone Technology, 2009

[7] https://www.arl.wustl.edu/projects/fpx/cs536/Soc_Firewall.pdf

[8] H. Khattri, N. K. V. Mangipudi and S. Mandujano, "HSDL: A Security Development Lifecycle for hardware technologies," *2012 IEEE International Symposium on Hardware-Oriented Security and Trust*, 2012, pp. 116-121, doi: 10.1109/HST.2012.6224330.