A Novel Hardware and Software Architecture Sequencing & Validation Platform

Authors:
Lashminarayan Venkatesan
Raghav Tenneti
Preetika Tandon
Vishal Shah
Edward Riegelsberger

Context – SoC Boot

Boot of an SoC is a sequential process that brings an uninitialized device to a functional state. It starts at chip reset-release and ends with an OS running.

Boot is complicated and difficult to specify: extremely cross-functional, with many conflicting constraints

- Must be water-tight w.r.t. security – extremely high bar. Ever rising.
- Must complete quickly – boot latency matters in many product spaces
- Must be functionally compatible with all IP initialization needs
- Must be parametrizable with platform constraints and dependencies.
- Must support all boot variants – cold-boot, warm-boot, etc.

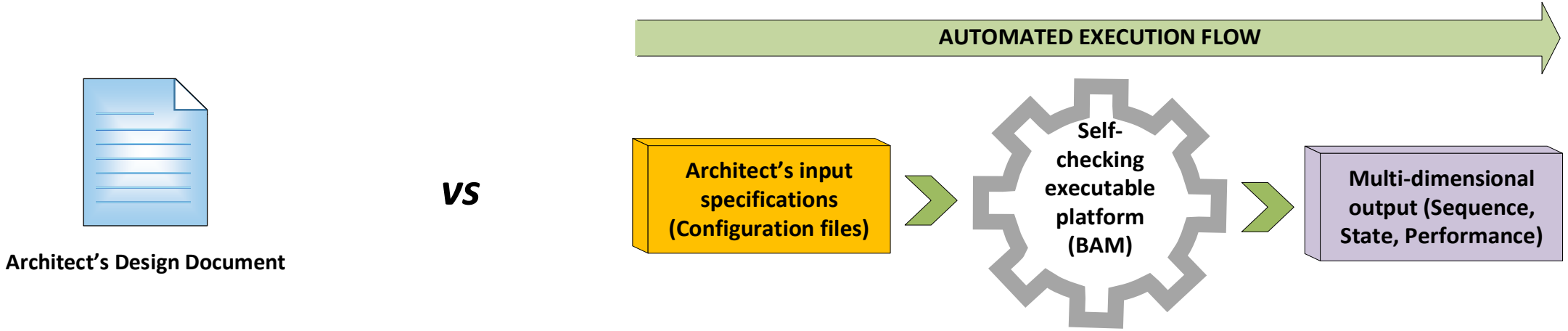
Motivation

System architects typically write software sequence specifications as a set of (subjective) documents.

Concerns:

- Documents alone are error-prone and hard to maintain.
- Difficult to ensure production software loyally executes the architectural intent
- SW testing often limited to ensuring a graceful exit of code to a limited set of directed inputs.
- Static performance analysis for varying constraints proves cumbersome

Our Solution: A self-checking, executable platform architecture specification



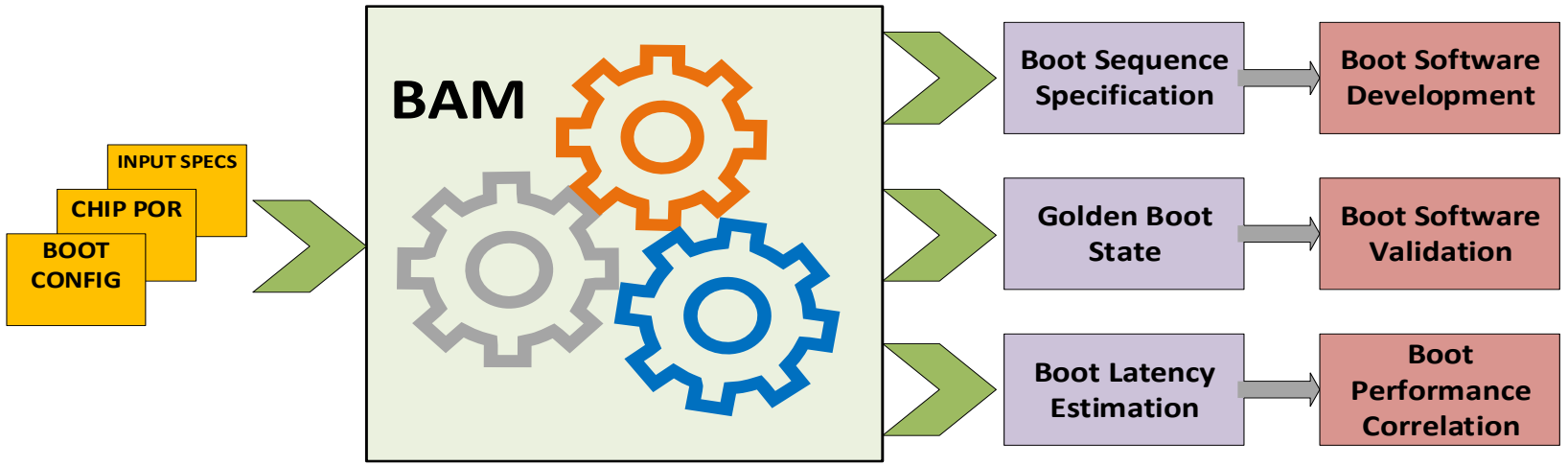
Boot Architectural Model (BAM)

An 'executable specification' for Boot, comprising

- A hierarchical state model
- Task definitions that act on state
- Parametrized SW task-lists, per boot processor
- Functional, security and latency assertions, self-checking

Inputs

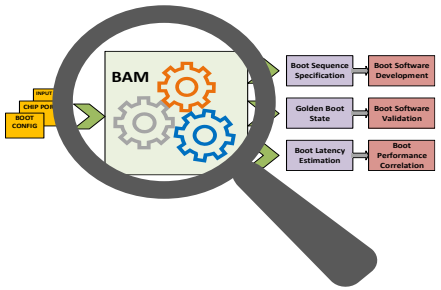
- Boot configuration files



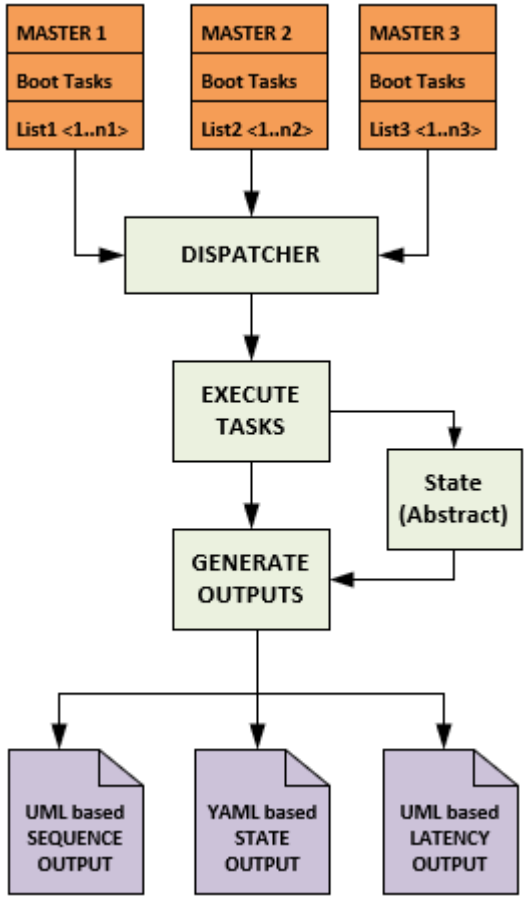
Outputs

- Auto-generated normative spec
  - SW Task sequence (steps and state changes)
  - System state snapshots
  - Boot sequence diagrams
  - Visualization extensions
- Boot latency estimations

Implementation



BAM TASK
Function
Boot Mode
Master ID
Master dependency
SW Latency Overhead



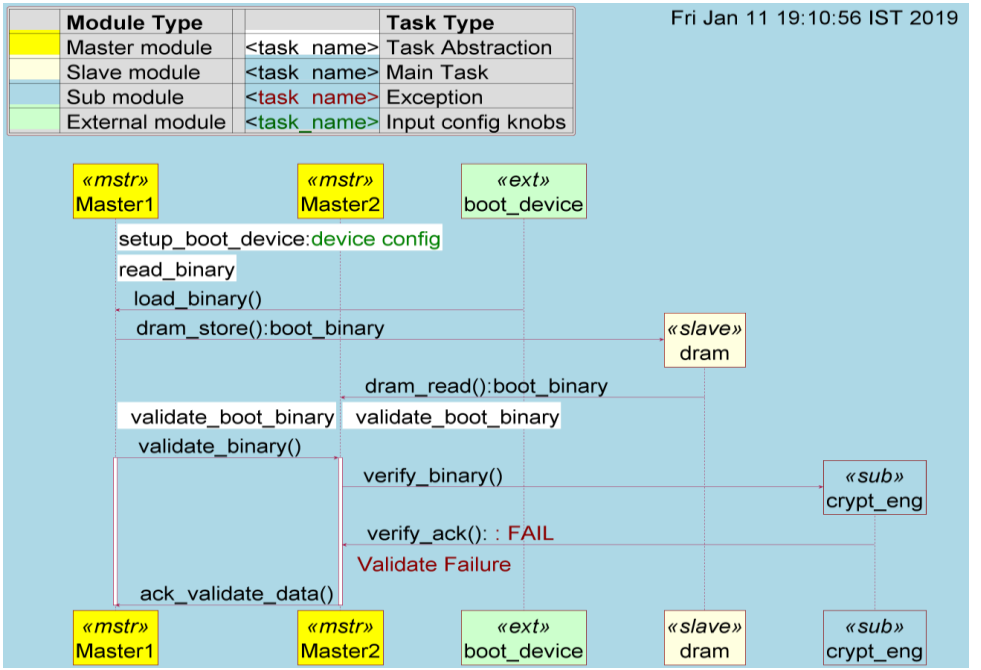
- **Task:** A boot step or SW function, defined by Boot Function and Boot Mode.
- **Task List:** List of Tasks executed by Boot Master.
- **Boot Master:** A programmable processor that executes task lists and changes system state
- **State:** Device state impacted by boot sequence
  - Abstract. Avoid register-level details.
- **Dispatcher:** determines the next task executed on the model
  - Emulate multiple Boot Masters running in parallel
  - Aware of time taken per task
  - Features for inter-processor synchronization, handshaking, and wait tasks
- After a task gets dispatched, it executes the **Boot Function.**
  - Updates state of all modules involved (in order)
  - Triggers assertion checks
  - Maintains elapsed time for latency reporting based on state and task parameters.
  - Adds to state_update reports and sequence diagrams

Applications

Use-case 1 : Automated Boot Specification

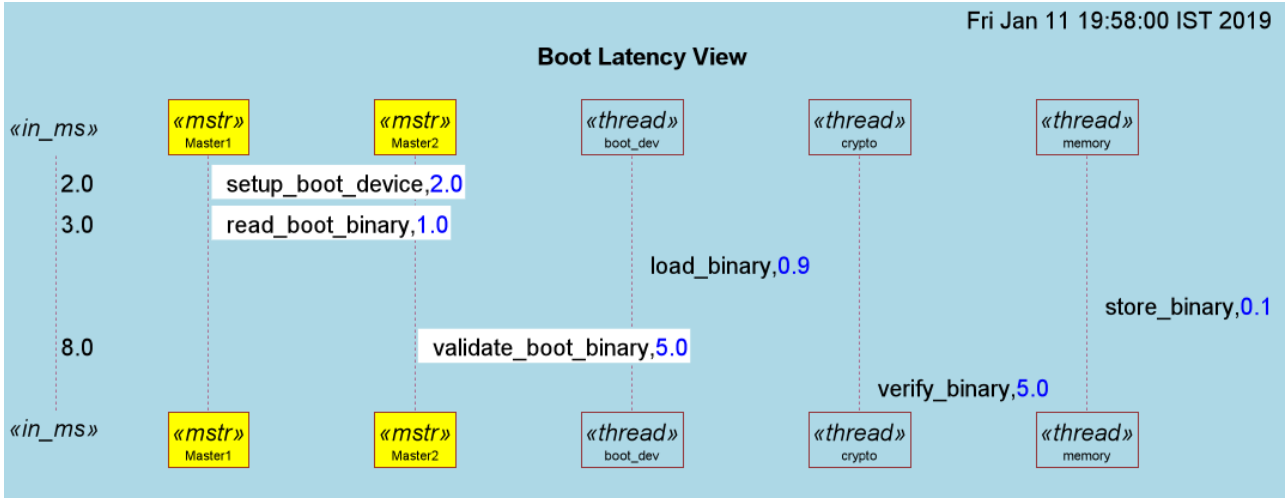
A sequence specification for SW/HW designers, including the functions and sub-functional names executed per task along with the interaction with different Boot masters and the slave modules.

- Passes all assertion checks

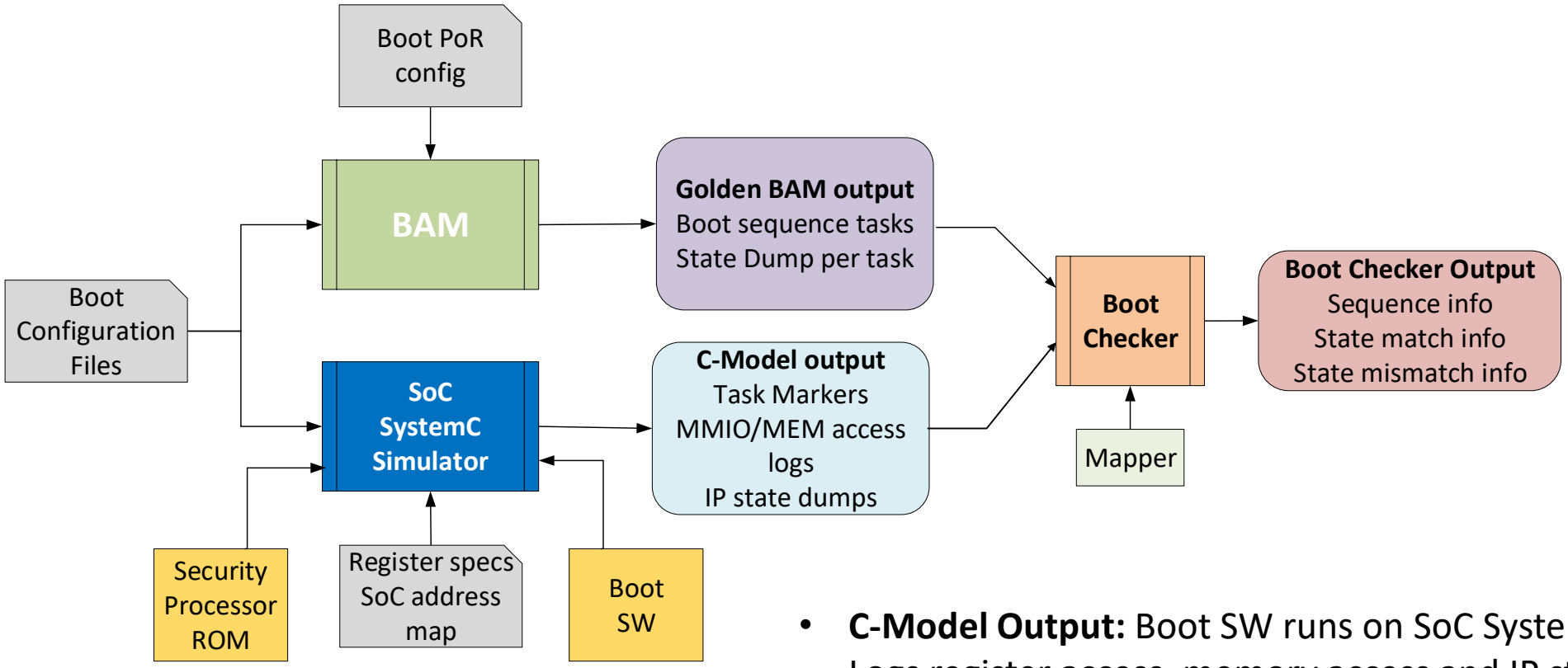


Use-case 2 : Boot Latency Computation

It provides a clear breakup of time-taken in each boot step and per hardware operation which helps in correlation with latency numbers obtained from the software running on the silicon.



Application to Pre-silicon SW Validation



- **C-Model Output:** Boot SW runs on SoC SystemC simulator. Logs register access, memory access and IP state updates.
- **Offline Boot-Checker:** compares logs against the BAM golden reference for same boot configuration.
- **Mapper:** functional translation from unit register accesses to abstract boot state for comparison with goldens.

Conclusions

- Self-checking assertions and auto-generated visualization of sequence helped in catching architectural bugs in early stages of Boot ROM code development
- It provides an automated spec comparison utility replacing code reviews.
- Boot SW validation is 10x faster on System SoC simulator using BAM golden output reference compared to RTL simulations.
- Automotive markets require Boot SW to be safety complaint (meet ASIL-D certification). BAM validation flow helps create trackable SW flow from specification to implementation.
- BAM helped in projecting expected latency per operation. Also the automated implementation helped in reducing boot latency by doing “what-if” analysis on boot sequences.
- Boot Latency correlation with silicon via latency trials on BAM PoC platform closed in 5x faster man-days than in previous chips.