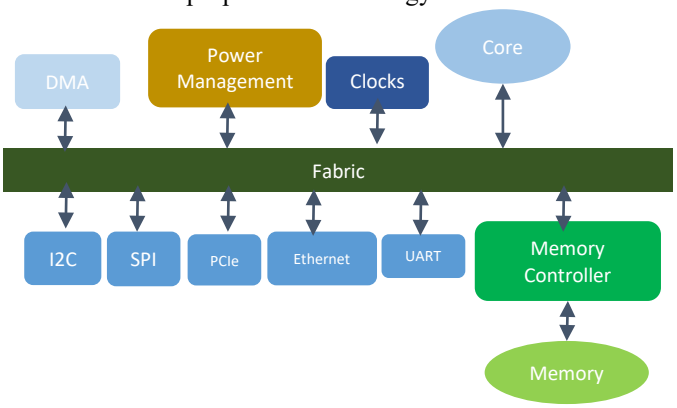
reach" coverage is cumbersome, and one needs to run a full regression and then analyse the coverage report. This method may result in several iterations and wastage of compute resource and time. This methodology uses a static probability calculator which iterates over constraints and gives probability of a scenario under all given constraints without having to run any regression thus saving compute resource and time. Using this proposed solution, the time required for bringing up a new flow significantly reduced from ~3days to a one-day effort. Modifying stimulus for hitting complex coverage and debugging a test failure becomes methodical which results in improved speed of execution. This methodology can be used effectively to solve the problem of subsystem random verification with reusability across projects with different configurations.

2. Overview of the proposed methodology

This paper will use a typical Microcontroller based Subsystem (depicted in Fig. 1.) as an example, to elaborate on the proposed methodology.



)LJ:SLFDO 0LFURFRQWURROHUEDVHG6EVWHP

Architecturally, a typical Sub System would have a single/ dual core that acts as the Main master, along with a DMA which offloads some of the traffic from the core and is the secondary Master. The subsystem would support multiple peripheral IPs which can serve both as masters or slaves. And one or more system memories that sit on a network fabric. There will be a Clocks and reset unit and a Power Management unit to control all the low power states supported by the subsystem.

The verification state space of such a subsystem will entail ability to setup requirements and configurations to enable all the valid masters to generate traffic and enable access to all slaves, concurrently. If the subsystem supports one or more low power states, the verification state space increases multi folds where it needs to cross all the system traffic with low power scenarios as well.

Just randomly kicking off all the traffic generators may provide functional coverage w.r.t concurrency of traffic in the subsystem but will not provide the ability to the user to control the traffic streams of interest or the ability to control the entry/exit to/from desired low power states within different windows of interest. It will also not provide fine control to the user to generate wake up at interesting windows in the subsystem to exit the low power states. That fine control can be achieved only if there is a framework that keeps track of state of all the components of the subsystem and has handles to control entry/ exit of the subsystem in and out of active and low power state at desired windows of interest.

This paper proposes such a methodology to develop a framework which provides ample control to the validator of a complex subsystem handling multiple low power states, to achieve desired validation coverage with quality in minimal duration. It also ensures reusability and portability of the different components of the subsystem Random engine to serve different SoCs or even different generations of the same SoC [1].

The fundamental aspect of this proposed methodology is to use constrained random verification along with scalable dashboard mechanism [2]. Fig. 2 gives the architectural view of the main components used in this methodology. The two main components are called, the C-Engine and the SV-Engine. Few of the sub-components may vary a little from one subsystem to another however the overall architecture would remain the same.

Fig. 2. Architectural view of the proposed methodology

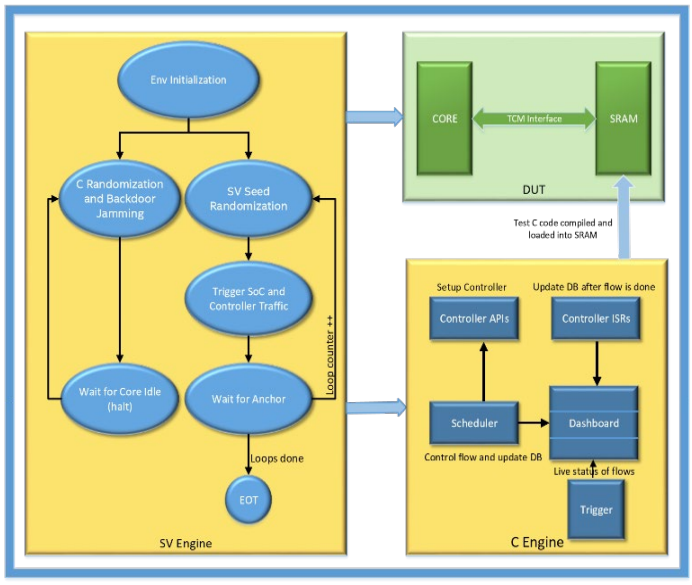


Fig. 2. Architectural view of the proposed methodology

A. C Engine

C-Engine as name suggest is a random test case software written in C-language that runs on microcontroller core of the subsystem for configuring IPs, initiating low power flows and respond to host-initiated flows. This component is the backbone for scalability and consists of Dashboard (DB), Scheduler and APIs and Interrupt Service Routines (ISR) for controllers within subsystem.

Dashboard is a simple C Data structure where in each row corresponds to one controller or low power flow and it has as many rows as number of controllers and low power

features within subsystem. Adding a new controller is equivalent to increasing dashboard size by one and adding API and ISR within C-engine for a new controller. Typical dashboard structure is shown in Fig. 3.

Project Specific Static bits

Flow Dynamic status bits

AGENT Name	Vnn Dependency	TCG Dependency	Task Enable	Task Schedule	Vnn Req	Vnn Ack	TCG Lock	API Pointer
I2C0	0	1	1	1	0	0	0	API_I2C0
I2C1	0	1	0	0	0	0	0	API_I2C1
DMA0	0	1	1	0	0	0	0	API_DMA0
IPAPG	0	1	0	0	0	0	1	API_IPAPG

Fig. 3. Dashboard Structure with Controller flows

Each column of the DB together gives the status of any flow corresponding to each row and consists of static (project specific) and dynamic (run time) bits. The DB is initialized once, setting project specific bits with appropriate values and dynamic bits with default values.

For the purpose of discussion in this paper, the example assumes two project static bits: "Vnn Dependency" and "TCG Dependency". "Vnn dependency" bit indicates if the specific Controller/ agent requires Vnn signal to be set before it can generate traffic. And the "Trunk Clock Gating (TCG) dependency" bit indicates if the controller/ agent needs to be in quiescence if the subsystem must enter a low power state. Task Enable (TE) bit if set, means a flow should be enabled. Task Scheduler (TS) bit when set indicates a flow is already triggered. TCG lock bit is used for preventing triggering of a new flow so that the system can enter clock gated state. TE bits are programmed randomly based on a trigger controlled by SV-engine. DB acts as a book keeping and used by scheduler explained below, to schedule tasks based on status bits. DB also contains API pointers that simplifies job of a scheduler. By setting a TE bit for any one agent this fully random test can used as a directed IP test.

Scheduler is a Finite State Machine (FSM) that iterates over DB in random order and calls controller API after its dependency bits are satisfied. Scheduler is completely agnostic to different entries in the DB and does a mechanical job of updating DB and calling API's based on current state of DB. Scheduler uses API pointer in a DB to call the correct API. Due to this implementation scheduler need not change with respect to addition or removal of rows in the DB. Fig. 4. Shows the scheduler FSM.

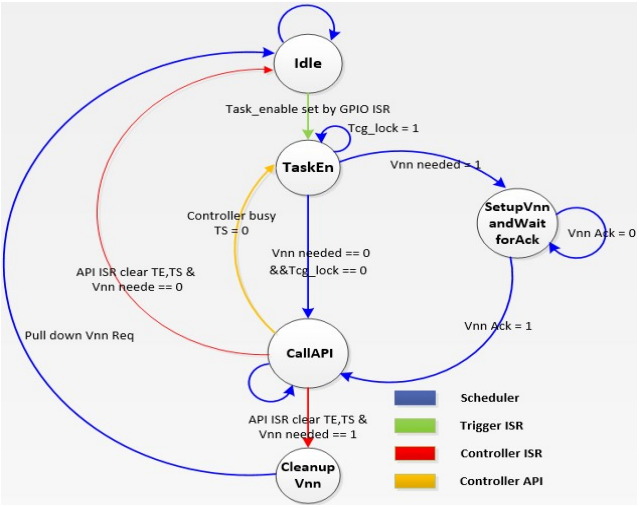


Fig. 4. Scheduler FSM

Consider for example a DMA controller entry in DB. Initial state for DMA after DB initialization would be Idle with its Vnn and TCG dependency bits set. When SV engine sets TE bit, state of the entry changes to TaskEn. Scheduler reads this entry and request for Vnn by setting Vnn_req bit as DMA would need Vnn for DRAM data transfer which is indicated by Vnn dependency bit. This changes entry state from TaskEn to SetupVnnandWaitforAck. Once Vnn_ack is received scheduler sets TS bit and invokes corresponding API for configuration of DMA and state changes to CallAPI. When the DMA transfer is over and an interrupt is asserted, ISR clears TE and TS bits and DMA entry enters Cleanup Vnn state. Scheduler now de-asserts Vnn, clearing Vnn_req and Vnn_ack bits in DB and state goes back to Idle.

B. SV Engine

SV engine handles 3 main tasks of managing all the randomization needed for covering the subsystem state space:

1. Generating all the legal traffic from external world to the subsystem
2. Creating quiescence condition needed for entering clock gate
3. Generate random constraints for 1 and 2

All the randomization needed by the subsystem, both for internal traffic within the sub-system and external traffic, is achieved by SV engine using weighted constrained randomization. The random data generated by SV engine is backdoor jammed in pre-assigned locations in SRAM and systematically picked up by C engine and used for all internal controller traffic thus saving a lot of microcontroller cycles which would otherwise get consumed in creating randomization through microcontroller.

The SV traffic generator manages triggering of all traffic external to the sub-system, like GPIO or IPC that can wake the subsystem. Relaunching of traffic threads is introduced in order to maintain high density of traffic throughout traffic phase. It depicts a system level Manager which triggers and controls all the valid traffic threads on a platform [3].

The SV engine achieves quiescence for Low Power (LP) states by using “anchor mechanism”. Anchors are typically, certain microcontroller states, FSM states within the subsystem or some RTL signals of interest. The SV engine lets the subsystem reach a state (anchor) of interest before triggering off external traffic causing the wakeup of the system from the LP state.

C and SV engines work together to make subsystem random test framework. SV engine randomizes agent configuration which is loaded it into the SRAM and initiates trigger for randomly setting TE bits in DB. Scheduler iterates over DB and takes appropriate action as explained above for DMA entry. Once programmed number of loops are done SV engine triggers end of the test.

Debug Efficiency

Debugging test failure at chip level can be complex due to concurrent traffic from different controllers. In order to achieve HVDM, reducing debug time becomes critical. To improve debug time, C-engine and SV engine monitors were developed that gives overall picture of concurrent traffic from both microcontroller side and host-initiated traffic and helps in quickly identifying the source of failure. Fig. 5. below Shows C and SV engine monitors dump.

```
|-->DASHBOARD MONITOR<--|
|------------PROJECT SPECIFIC ENTRIES---------|--------------DASHBOARD DYNAMIC ENTRIES-----------|
|------------+--------------------+-----+-----+-----+-----+-----+-----+-----+--------------------|
|    TIME    |      TASKNAME      | VNN | TCG | TASK| TASK| VNN | VNN | TCG |        DB          |
|            |                    | DEP | DEP | EN  | SCH | REQ | ACK | LOCK|       STATE        |
|------------+--------------------+-----+-----+-----+-----+-----+-----+-----+--------------------|
|     1366857|         AGENT_HPET0|    0|    0|    0|    0|    0|    0|    0|                IDLE|
|      105492|AGENT_ISH2HOST_HOST0|    1|    1|    0|    0|    0|    0|    0|            IDLE_VNN|
|     1232242|         AGENT_HPET1|    0|    0|    0|    1|    0|    0|    0|             TASK_EN|
|      104974|  AGENT_DMA_CH0_INTR|    0|    1|    1|    1|    0|    0|    0|            CALL_API|
|      105772|   AGENT_ISH2NONHOST|    1|    0|    1|    0|    1|    1|    0|  SETUP_VNN_ACK_RCVD|
|      105140|  AGENT_DMA_CH1_EXTR|    1|    1|    1|    1|    1|    1|    0|         CLEANUP_VNN|
|------------+--------------------+-----+-----+-----+-----+-----+-----+-----+--------------------|

|---------+---------+---------+---------+------------------+------------------+---------+---------+---------+--------------|
|->SV Traffic DashBoard updated<-|-->Iteration Number: 1<--|-->Relaunch Enabled: 0<--|-->Max Relaunch cntr: 3<--
|---------+---------+---------+---------+------------------+------------------+---------+---------+---------+--------------|
|Thread num  | Thread en|  Th Delay|Th relaunch|      Start TS      |      End TS        | Duration |  Status  |Done Once |Relaunch cntr|
|---------+---------+---------+---------+------------------+------------------+---------+---------+---------+--------------|
|-------->Primary Traffic Threads<----------------------------------|-----------------------------------------------------------|
|       GPIO|        1|       29|       NO |            791626|            791626|        0|        0|        1|             1|
|      IPIAPG|        0|      629|      YES |                 0|                 0|        0|        0|        0|             0|
| HOST_POSTED|        0|     1587|      YES |                 0|                 0|        0|        0|        0|             0|
|   HOST_SRAM|        1|      296|      YES |           8132552|                 0|        0|        0|        0|             0|
|    HOST_AON|        0|      581|      YES |                 0|                 0|        0|        0|        0|             0|
|-------->Secondary Traffic Threads<--------------------------------|-----------------------------------------------------------|
|-------->HOST SRAM Traffic Threads<--------------------------------|-----------------------------------------------------------|
|          0|        0|        3|      YES|                 0|                 0|        0|        0|        0|             0|
|          1|        1|        5|      YES|           8132667|                 0|        0|        1|        0|             0|
|          2|        0|        8|      YES|                 0|                 0|        0|        0|        0|             0|
|---------+---------+---------+---------+------------------+------------------+---------+---------+---------+--------------|
|                                  All Enabled Threads Done atleast once: 0                                                  |
|---------+---------+---------+---------+------------------+------------------+---------+---------+---------+--------------|
```

Fig. 5. C and SV engine monitors dump

Coverage Goal

With the increase in the number of Controllers in the subsystem, supported low power states and number of wake sources, the complexity of the subsystem can grow rapidly. This results in large number of functional Cover Groups, cross coverage and coverage for corner case scenarios. To address this need of meeting complex coverage, finer control is required on the stimulus. With large number of inter dependent constraints, predicting the occurrence of a scenario, becomes extremely essential and difficult at the same time. This problem was addressed by innovating the idea of "Probability Calculator" (PC). PC can be used to statically calculate the probability of a random variable, without running a single cycle of simulation.

By using probabilities of all the random variables including their weightage, it calculates probability of a certain scenario for e.g. "A successful entry into Deep Sleep with Memories power Gated and Logic not Power Gated and wakeup by a Ethernet Rx Magic Packet event". Traditional method has been to run regressions, generate and merge coverage and then get feedback from the coverage report to take any corrective actions. But by using this calculator as shown in Fig. 6. turnaround time is reduced from several hours to minutes. Based on the feedback from the Calculator, corrective actions can be taken in terms of either running more seeds upfront or by adjusting the weightage on the random variable constraint, affecting the probability of a scenario.

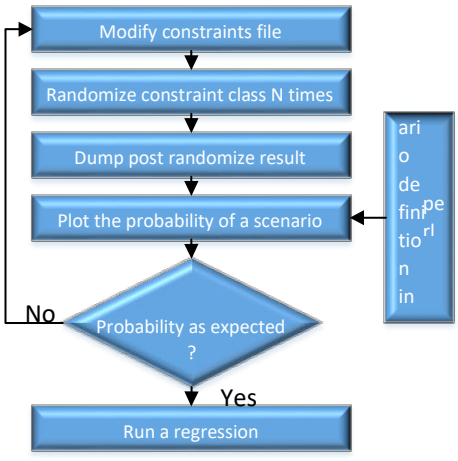


Fig. 6. Stimulus control with Probability Calculator

Results

Table 1 presents improvements measured in various areas of verification with proposed solution for a Subsystem that was delivered to multiple SoCs

Table 1. Improvements achieved with proposed solution

areas of Improvement	Traditional Method	Proposed Solution	%Improvement	Justification
Integration of new IP flow	~3days	~1day	66%	Scalable and structured framework
Debug time	~2days	~1day	50%	C and SV engine monitors
Functional Coverage Targets	~3days	~2days	33%	Flow controllability and Static probability calculator

Summary

The proposed solution helped in achieving High Velocity Development Model with quality verification. Currently no standard framework for subsystem random verification exists which can be reused across different subsystems. Structured and scalable implementation used in this methodology enabled easy integration of a new IPs and power flows into existing Random Engine. This methodology can be adapted by any microcontroller-based subsystem with little modification specific to the subsystem. Proposed static probability calculator and innovative debug solution proved very useful in reducing debug efforts and time required for achieving 100% coverage goal.

Acknowledgements.

The authors would like to acknowledge Mangesh Kondalkar for his valuable contributions towards development of this Methodology and implementation.

References

[1] Hu Zhaohui, A. Pierres, Hu Shiqing, Chen Fang, P. Royannez, Eng Pek See,Yean Ling Hoon, "Practical and efficient SOC verification flow by reusing IPtestcase and testbench", 2012 International SoC Design Conference (ISOCC),IEEE, 4-7 Nov. 2012.

[2] Sainath Karlapalem, Shashank Venugopal, "Scalable, Constrained Random Software driven Verification", 17th International Workshop on Microprocessor and SOC Test and Verification, 2016.

[3] I. Silas, I. Frumkin, E. Hazan, E. Mor, G. Zobin, "System-level validation of the intel Pentium M processor", *Intel Technol. J.*, vol. 7, no. 2, pp. 38-43, May 2003.

[1]