DVCon Europe 2022

A novel approach to expedite MCU verification and enabling efficacious inter-processor communication in a multiprocessor SoC

Harshal Kothari, Staff Engineer, Samsung Semiconductor India Research, Bangalore, India (harshal.k1@samsung.com)

Manishadevi Satyanarayana Cheernam, Senior Engineer, Samsung Semiconductor India Research, Bangalore, India (c.manisha@samsung.com)

Vignesh Adiththan, Associate Staff Engineer, Samsung Semiconductor India Research, Bangalore, India (vignesh.a1@samsung.com)

Sriram Kazhiyur Sounderrajan, Associate Director, Samsung Semiconductor India Research, Bangalore, India (sriram.k.s@samsung.com)

Somasunder Kattepura Sreenath, Director, Samsung Semiconductor India Research, Bangalore, India (soma.ks@samsung.com)

***Abstract*— In today's world, as the data complexity is increasing rapidly, there is a need for processors with high efficiency and high performance to manage SoCs that perform a plethora of functionalities in different subsystems. With increase in reusable software and microcontroller IP cores along with shrinking development times combined with multiple processors used across SoC for different systems has resulted in huge challenge to complete verification with due quality in a short span of time. This calls for a novel streamlined approach to verify the processors at different level of abstractions and leverage the closure by divide and conquer approach. The approach includes verification at standalone subsystem and SoC level abstraction with use cases and with the inter-processor communication. To achieve the goal of leveraging and combining results from different abstractions, a Modular Reusable MCU Testbench (MRMT) architecture was developed resulting in 41% savings of overall verification effort and to detect ~10 critical Silicon breaking issues. The paper details on how the enormous workload needed to functionally verify multiple microcontroller subsystems and their inter processor communication was achieved using MRMT architecture and how this can be leveraged as a generic architecture for MCU based design verification across industry.**

Keywords—MCU, microcontroller, verification, security, NVIC, Interprocessor communication, SoC, FIFO, Interrupt

I. Introduction

The XR co-processor SoC provides acceleration for computer vision, audio, and display compression. It runs an RTOS and uses several ARM Cortex microcontroller class MCUs to control the various subsystems and a Root of Trust microcontroller handling the SoC security. The SoC is attached to an Application Processor which is responsible for running the high level OS. The interface between the SoC and the application processor is achieved with PCIe Gen 4 interface or USB 3.0 for high bandwidth communications and LSIO (UART, I3C, SPI) for low bandwidth communications. It hosts a PCIe root port for connection to external endpoint devices. The presence of multiple independent subsystems like Always-ON, security, BUS/NOC, Cache, DRAM Memory Interface, Display Serial Interface, PCIe, USB, Camera Serial Interface, computer vision, audio and compression, making it imperative to verify MCU functionality with typical use-cases and inter-processor communication.

The SoC consisted of ARM Cortex M0 performing the SoC boot with preloaded ROM Code along with a third party crypto manager Root of Trust engine for handling security features. Multiple instances of ARM Cortex M7 were responsible for controlling the sub systems across SoC. The cores are pre-verified IP for SoC abstraction. The major challenge at higher abstractions is to check unique features like Memory Protection Unit (MPU), Error Correcting Code (ECC), floating point arithmetic with single and double precision, faster interrupt responses using Nested Vectored Interrupt Controller (NVIC), and low power features like clock gating, sleep mode and wake up interrupt controller (WIC). The major challenge is to understand the behavior of the core, power controller and subsystem and to ensure that the handshake between them is meeting the expectation of the design requirement and customer requirements. The early sub system abstraction resulted in uncovering at least 3 issues resulting due to design and customer requirement mismatch. The ease of integration, verification and firmware development makes them one of the go to processors for a wide variety of applications.

II. MRMT Architecture

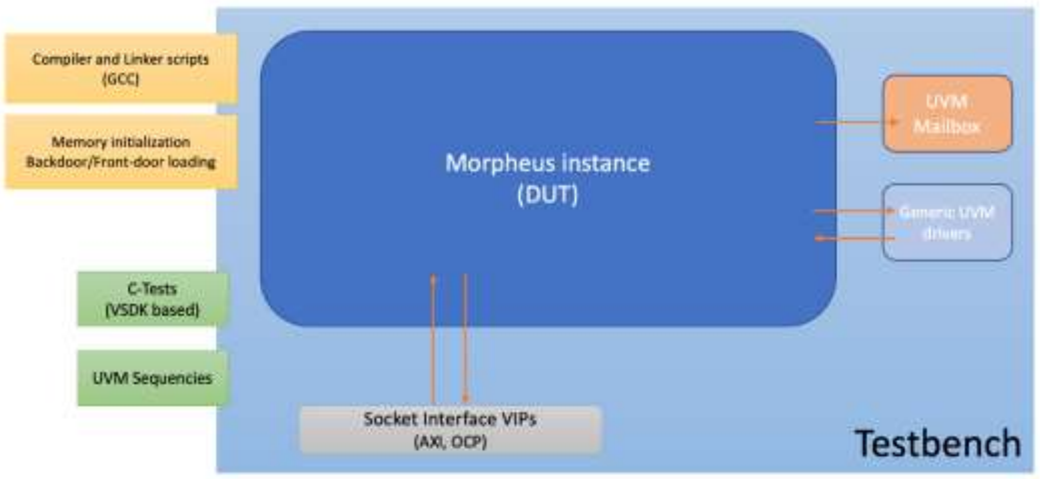
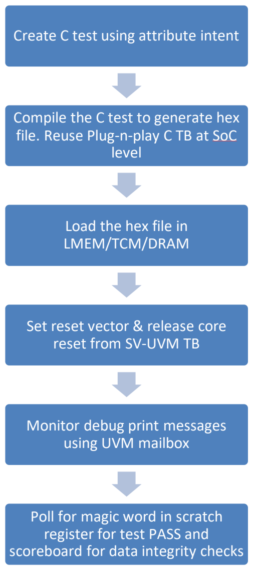

Compiler and Linker scripts (GCC)
Memory initialization Backdoor/Front-door loading
C-Tests (VSDK based)
UVM Sequencies
Morpheus instance (DUT)
UVM Mailbox
Generic UVM drivers
Socket Interface VIPs (AXI, OCP)
Testbench

Figure 1 MRMT testbench architecture

Development of MRMT begins at the subsystem level where Verification IPs (VIPs) are integrated at all the socket AXI, AHB and APB master and slave interfaces. Since SoC architecture is still in early stages of development at this point, the address space is extrapolated so as to not hinder the early verification of the MCU block. Verification typically involves basic hello world boot, memory and DMA access, interrupt and power scenarios. RTL simulations run at subsystem level are 10x-15x faster than SoC. It helps to flush the MRMT flow and find configuration and integration bugs at the MCU level. Everything except Socket Interface VIPs are reusable component from MRMT. This enhanced MCU testbench can then be seamlessly plugged into SoC abstraction level where real logic and memories replace erstwhile used VIPs and real world use-case implementation begins. Simulations for the MCU are run in both subsystem and SoC context reusing the common test-bench for program generation. The verification steps encompass boot image generation from a C based test-bench, using compiler and linker scripts, loading of the low level hex code image into a particular memory (either local SRAM memory/Tightly Coupled Memory/system scratch memory/external flash/external DRAM), setting of reset vector address for boot code fetch and performing the boot followed by use-case operations. All transactions onto other IP interfaces are driven by MCU, not driver or Verification IP transactors. The various components of this test bench involve ARM GCC compiler setup, common set of reusable compile and library files, make-file, linker file and C test code which can be prudently used for all MCU instances across the SoC. (1)

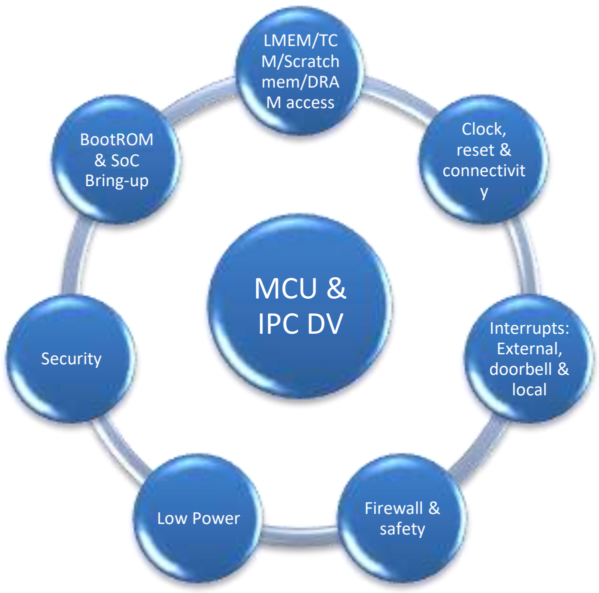
III. Execution



Create C test using attribute intent
Compile the C test to generate hex file. Reuse Plug-n-play C TB at SoC level
Load the hex file in LMEM/TCM/DRAM
Set reset vector & release core reset from SV-UVM TB
Monitor debug print messages using UVM mailbox
Poll for magic word in scratch register for test PASS and scoreboard for data integrity checks

Figure 2 MRMT Verification Steps

The MRMT hybrid environment combines both C and System Verilog (SV) with Universal Verification Methodology (UVM) to perform the verification of the MCU using dynamic simulations. The boot process completion is determined by the successful write of the magic word register at end of the test based on the C flow. The compiler converts the high level code into assembly level code resulting in a System Verilog readable hex file using $readmemh System Verilog task that initiates the behavioural RAM model via backdoor. Considering the impact on the simulation time, the load via front door using the SoC boot or secure processor is avoided in resulting simulation time saving up to 35%. To provide a flexibility during run time, the load mechanism can be controlled as a simulation parameter with System Verilog construct testplusargs. The reset vector is positioned at the base address of this memory location and core reset is released resulting in the execution of the fetched instructions. In order to qualify successful boot, the value written in the scratch register from the C environment is polled for using AMBA transactors on SV environment. A matching read back implies successful boot and test sequence of MCU. The processor generates the stimuli by running the program stored in its memory. External stimuli and response checks are coded in the testbench ensuring the sanity of the functionality.



LMEM/TCM/Scratch mem/DRAM access
Clock, reset & connectivity
Interrupts: External, doorbell & local
Firewall & safety
Low Power
Security
BootROM & SoC Bring-up
MCU & IPC DV

Figure 3 MCU & IPC SoC DV attributes under MRMT

Critical use-case features are verified at SoC level after MRMT ensures a bug-free subsystem (3). This plug and play approach was inculcated and tested for both in-house developed testbench and customer delivered third-party testbench and it ensured smooth and early verification with impeccable turn-around time. In order to help in debugging the C code execution, UVM mailbox monitor is implemented. This component enables printing the display statements present in the C code and indications like main function start and end and interrupt service routines into the simulation log files. This handy utility provides handshake between C test running on MCU thread and SV test-bench environment. Once the reset is released and the boot starts to proceed, the mailbox monitor starts printing useful information from the C test which helps in debugging the C code as well logs the progress of the program. Mailbox base address is set in such a manner that it does not overlap with the executable boot code instructions. This mailbox discerns the status of software program progress, whenever the new message comes from C environment, it prints in simulator log based on ASCII value converter. At the end of C test, it prints whether the C test execution passed or failed depending on error flag increment coded in the C test.

Another nifty benefit of using this verification methodology manifests during interrupt verification. At SoC level, up to 256 interrupts can be routed to each MCU. Across the 9 instances in current implementation, close to 860 interrupts are routed which are ingeniously verified along with their functional data-path and real usage scenario. Common interrupt task is implemented to set, generate and service the interrupt, which other IP owners can use as part of their test bench environment. Earlier, to do away with the need of booting MCU to execute verification of other IPs, a Verification IP (AMBA AXI transactor) was hooked up at the M7 AXI interface. However, NVIC cannot become operational without booting up the MCU and setting interrupt service routine environments with real boot code. This is why real software hex code is loaded in MCU memory in conjunction with other IP data path running from SV side test to generate and service the interrupt similar to real use-case. For example, due to chip over-heating, a temperature management unit test generating a trip threshold interrupt actually triggered primary SoC MCU to divide the clock frequencies inside clock manager to bring power dissipation down and reducing temperature below critical threshold. This illustrates how this methodology enabled DV engineers to combine pre-existing SV-UVM based verification environment of other IPs with real MCU operation traversing all important system level scenarios. This mimicked the way Silicon was actually going to perform in the final product and helped identify bugs in concurrent traffic scenarios and corner cases.

NVIC interrupt functional coverage model was newly developed to track the enable, assertion, service and de-assertion of interrupt at the controller boundary. An MCU to Power controller FSM was also integrated and verified. It maps the ARM low power modes to the custom power controller to enable power saving techniques like clock gating, power gating, memory retention etc. via multiple low power states namely idle, active sleep, light sleep, deep sleep, standby and off state. This can be reused to plug-and-play any microcontroller IP core into the SoCs to leverage the Silicon proven power manager to throttle power consumption of the subsystem and implement robust low to ultra-low power saving techniques.

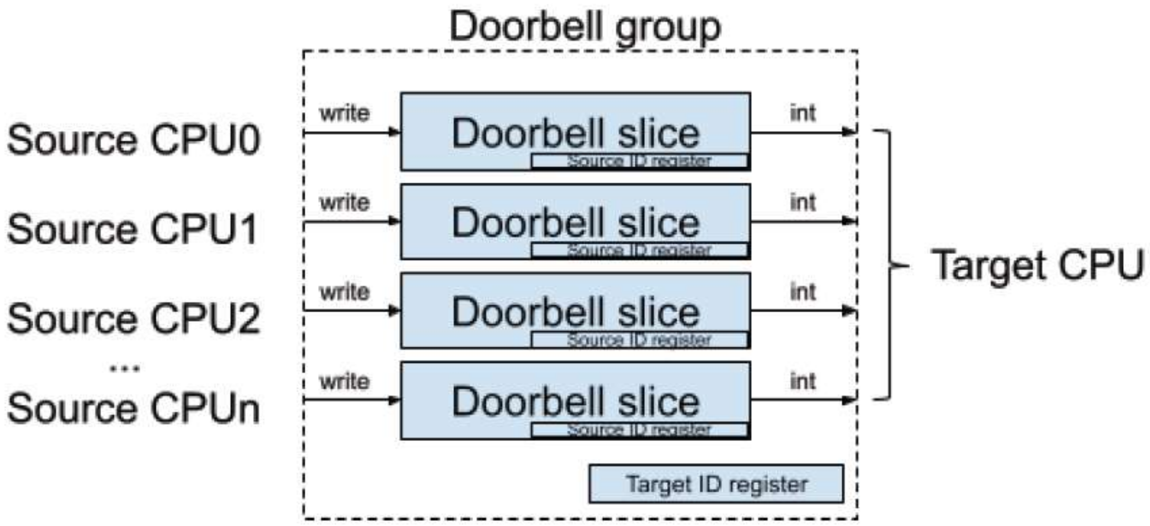


Figure 4 Inter Processor Communication Via Doorbell

Since the SoC is a multi-processor based chip running multiple parallel threads in varying subsystems, the inter processor communication is ensued with the help of an inter processor communicator IP. Its internal components 1) Mutex for resource sharing and 2) Doorbell for inter processor communication via interrupts are verified as part of the IPC test sequences. Doorbell module has doorbell slices which has one target CPU and one source CPU. Up to 16 doorbell slices contribute to one doorbell group. This is how each doorbell group has multiple source CPUs and a single target mapped CPU, to which the interrupt is finally triggered. Each processor is associated with a Role ID and based on it the processor could ring the doorbell which in turn raises the interrupt of the target MCU. This Role ID is transferred over SoC fabric over AxUSER ID field to the IPC. The doorbell is cleared only when the target CPU acknowledges the message by clearing the doorbell register. Both target CPU ID and source CPU IDs are stored in respective registers and they are validated during ringing the doorbell and clearing the doorbell. If any mismatch is found, it raises the doorbell error interrupt. At a time, only a single source CPU can ring the doorbell of a particular doorbell slice. In this way the multiprocessor system could communicate in order to carry out its functionalities in an orderly fashion. These interrupts are also used as wake events by the target CPU. Hence in this context, the wake from low power features like power gating or clock gating is also additionally exercised and verified via these doorbell interrupts.

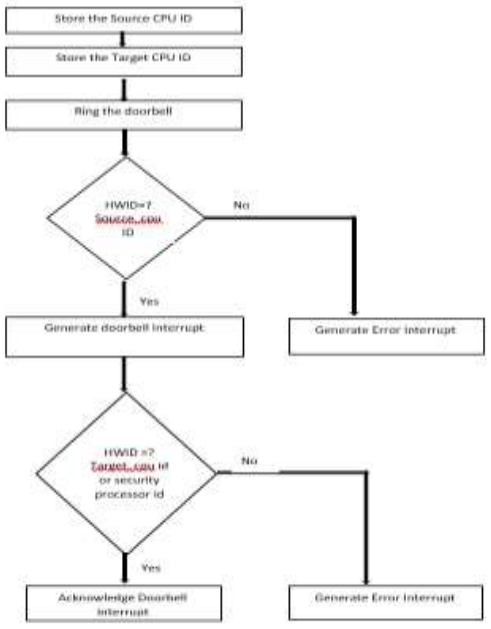


Figure 5 IPC Verification flow

Along with the normal doorbell, FIFO doorbell is also verified wherein the messages are stored in a FIFO (implemented by external SRAM) and once the doorbell is rung, the messages that are stored in the FIFO from the source CPU is transferred to the target CPU. This is more efficient as more information could be transferred from source CPU to target CPU in a shorter time and the execution time taken by target CPU also reduces.

Mutex component in the IPC IP is a shared device that helps to achieve mutually exclusive ownership of certain resources, e.g. memory range, IO device, etc. It works by capturing the initiator's unique hardware ID, carried on request information bits and a unique software ID when a locking write is performed. A subsequent read will return the same unique software if the lock was successful. To release the mutex, the same initiator that performed the lock must unlock the mutex by writing the same unique software ID. This ensures that only one initiator can have access to the mutex at any given time. This is verified thoroughly to enhance security aspect of IPC. An unlocking write to a locked mutex that fails due to an incorrect software or hardware ID returns an error. Since there are multiple processors in the SoC which has limited access to different blocks present, the Security processor sitting in Always-On block is the main master at the core of this feature. It has access to all the blocks of the SoC and can set secure accesses for other subsystems and IPs. The security processor has a unique ID associated with it which is also stored in the mutex registers. When a processor acquires a mutex lock, the lock can also be released by the security processor. When the security processor tries to release the mutex lock the ID is matched with the stored register value and then granted access so as to supersede the security privileges. Even in the case of doorbell, once the bell is rung for inter processor communication by a particular processor, the acknowledgement can be done by the security processor too instead of the target processor. This provides an added advantage as the security processor could be used to resolve any unforeseen issue that occurs.

In addition to adding a security layer for IPC, security processor also manages test and debug access of the SoC. It fetches firmware blobs of other microcontrollers / CPU units and performs signature verification. The SoC communicates with PCIe EP and Application Processor and both of these flows are encrypted. Diffie Hellman Key exchange mechanism is used for key exchanges which is also handled by security processor. The data transfer to/from DRAM is also encrypted similarly and verified using a ruby script. Fabric Firewalls are further used to isolate and protect the physical address space of the SoC also controlled by security processor. In addition to multiple MCUs operating in tandem to achieve the various use cases, our extensive verification scenarios covered all the security aspects of the SoC. The cold boot is performed via BOOTROM and every branch of the ROM and firmware-controlled boot is verified at RTL, Power Aware RTL, unit-delay and timing Gate Level Simulations and

Power Aware Gate Level Simulations. The ecosystem discussed here was extended to the booting and security processors as well, which expedited the MCU verification by leveraging the concept that was developed and proven with multiple CM7 instances. This alternative approach facilitated in implementing all major real world usage data paths within stipulated stringent timelines with dynamic simulations resulting in a first-pass Silicon.

IV. CONCLUSION

The entire MRMT functionality is encapsulated in a single set of scripts and makefile, which also allows automation of complete MCU regression. This niche approach has enough modularity to be used as is for emulation platforms. As part of future scope, MRMT developed in the DV stage is planned to be extensively reused to extend data size for mem to mem copy scenarios, DMA operation, interrupt service routines and high run-time SoC level scenarios with an FPGA image. This facilitates early driver development, which shifts left the software bring-up for the final SoC.

Using this generic approach, we were able to create an innovative methodology to scale the verification for multiple MCU instances in an SoC while facilitating robust inter-processor communication amongst them. This has been actualized with 3 different classes of MCU in 2 XR SoCs with 16+ instances. Future scope further involves automating the subsystem environment and testbench porting to any SoC, where preliminary results are showing a tremendous 75% save in terms of simulation time and manual effort.