My encounters with Virtual Platforms

Praveen Wadikar

Introduction

My tryst with SystemC started while working with VaST tools (Now part of Synopsys). My job involved modeling functionally accurate peripherals in SystemC which could work with VaST infrastructure. SystemC had been around for sometime and these were the pre-TLM2 days. So every vendor had their own flavors of ports. Although the end goal was to develop the SoC platform to be used by software teams and tier 1 customer but my main focus was proving the equivalence of SystemC model with the corresponding RTL.

Now lets take a step back. Before entering the SystemC world I was an embedded software engineer. One of the things we used to verify the firmware before the Silicon arrival was FPGA boards. We started with 2 fpgas then moved to 4 and quickly realized the design doesn't really fit in 4 FPGAS also. Lot of time and effort went in "modifying" the RTL so that it fits FPGA and that design partitioning doesn't really affect the functionality. Also the FPGA boards were limited and also could not shipped to customer's software engineers. But we could get away with these because the amount of software was far less compared to the today' SOCs. These were the days where software was a poor cousin of hardware. However there has been great progress in the FPGA prototyping world and things are not the same when I first worked on a FPGA prototype.

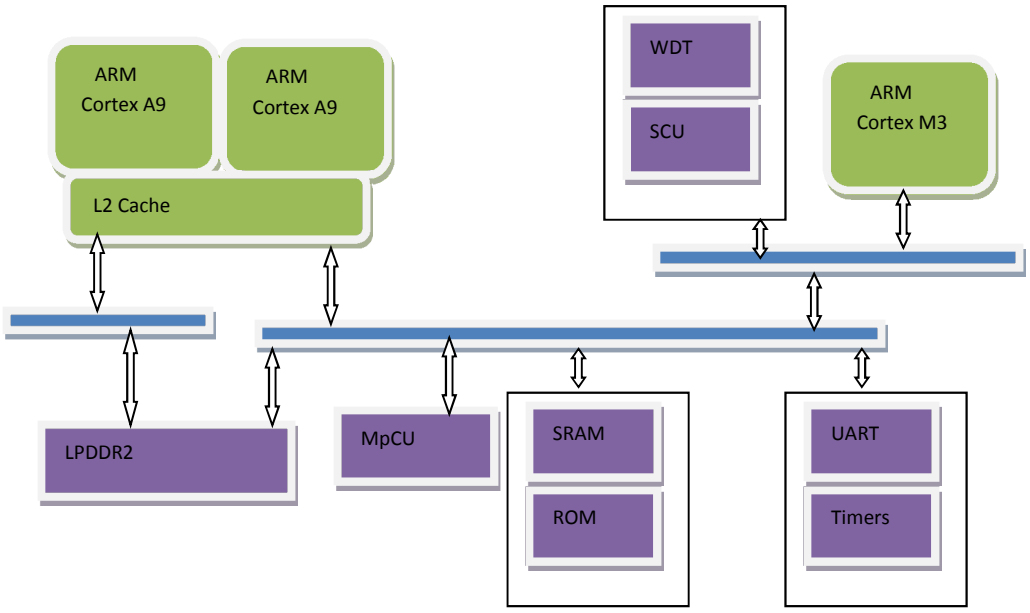
So working on fast functionally accurate simulation model opened up a new perspective. You could model the entire SoC and could ship it to the customer's software engineers. Its like each software engineer has his/her own board. This made more and more sense as the software content increased and suddenly with Android and its brothers, the need to have software running before Silicon arrives became of prime importance.

Virtual platform for Software bringup

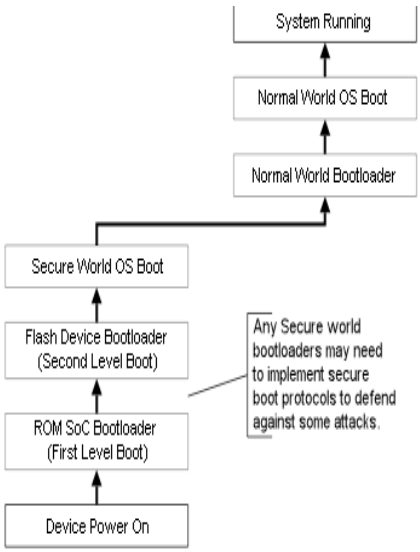
As a result of this, Virtual prototypes (VP) are being increasingly used to develop and test software. The basic building blocks that are used in creating a VP are Transactional Level Models (TLM). With SystemC and TLM2 developing a TLM model which could work across multiple vendors became a possibility. This also enabled IP reuse.

Now lets look at a multi-core platform and look at how a virtual platform can be useful for software *(This was presented in the ARM Techcon: video link below)*. I will explain the multi boot process and look at some of the problems faced and solved.

The platform is shown in below figure



The platform contains two on-die Boot ROMs. One is dedicated for M3 and the other one is for the MP-core A9. These ROMs are mapped at 0x0 (the reset vector) in the M3 and A9 address space. Booting from on-die ROM guarantees that the boot image cannot be changed which is an essential requirement for TrustZone based systems. Hence the boot loader for the example platform will involve multi stage boot. This is shown in diagram below



In order to explain the hardware software debug we consider a stage of the boot loader the where it copies this level bootloader image into A9 secure SRAM2 and do not execute it. Instead, the A9 and M3 handshake with each other to copy boot loader image to M3 secure SRAM, where the M3 finally executes it. After the copy is complete the A9 subsystem is shut off. This can be done by M3 subsystem as it has ability to control the reset and power domain settings of A9 subsystem. M3 then executes the bootloader that was copied to its SRAM. Once this is complete it reboots A9 subsystem. The A9 subsystem then starts the Normal world Bootloader. One of the first problem we saw was that the execution failed at Normal world Bootloader (refer figure above). In order to debug this scenario let us look at the potential causes that can lead to this situation. These are

1) Problem in the synchronization of access to shared RAM that is used by A9 and M3 for bootloader copy.
2) Problem in Multiprocessor communication unit (MpCU)
3) Problem with interrupt configuration
4) M3 – A9 reset architecture

In order to debug such a situation a good debugger is necessary. The debugger should be able to provide all the features that a standard embedded software debugger provides. These include breakpoints, watchpoints and memory viewer to name a few. Additionally we should get the advantage of using a hardware debug environment which can be connected to the virtual system platform. Typically this needs to support hardware debug environment like waveform tracing, thread level breakpoints, event tracing, transaction tracing. Also there should be a tight synchronization between the hardware style debugger with the software style debugger. Now such debuggers are available from EDA partners.

Now that we know of these debuggers let us try to see how these can help us in solving our problem. Let us take each of the potential causes starting with Problem in the synchronization of access to shared RAM that is used by A9 and M3 for bootloader copy. This is quite easily achievable only with software style debugger. We can set a breakpoint at the start of the copy. We can then use the memory viewer and observe the contents of shared RAM and observe the contents of the M3 SRAM. Visual inspection confirmed that contents look same. So this confirms that both A9 and M3 are able to access the shared RAM. Assuming if this was a problem we could use the transaction breakpoint (a feature wherein the execution breaks when access is made to the programmed address). This is basically like tracing the bus protocol.

Now lets focus the attention to the next potential cause which is interrupt generation. The simplest way to debug this is to use the hardware style debugger and use the waveform capture capability. The MpCU block generates interrupts to A9 and M3 to achieve synchronization. We can add the interrupt ports of MpCU block. From the waveform it was clear that the MpCU was indeed toggling the interrupt lines. Additionally tracing the events we confirmed that the MpCU block was configured correctly. However one of the observations we made in the waveform was that from the MpCU we saw only one set of synchronization interrupt. However we expect atleast three sets of synchronization interrupt. This is

because synchronization happens at every page copy (4kb). The M3 bootloader was more than 4kb. So now we know that there is some problem in the A9 bootloader code that is involved in this copy. Now to debug this we set breakpoint in the hardware thread of MpCU that generates the interrupt. Once the execution is stopped at this point, we shift our focus to the software debugger. We now use the regular 'debug techniques' like single stepping, breakpoints. We find that A9 bootloader waits for the interrupt from MpCU which indicates that M3 has copied the bootloader from shared memory to its SRAM. Once this interrupt is received the A9 starts the copy of the next page.

From our earlier waveform observation we had seen that M3 was sending that interrupt using the MpCU. So now the focus shifts to GIC. Probing the interrupt line shows that GIC was indeed getting the interrupt. Now when we probe the status register it shows that the particular bit for the GIC is still showing "interrupt not generated". We then looked at the GIC spec and find that he GIC works on interrupt IDs and not the physical IRQS lines. Interrupt ID 0 to 15 is usually used for S/W interrupts. In most of the cases the physical line IRQS[0] will correspond to interrupt ID 32. This is essential because all the enable/disable and other configuration are based on the function ID. So when we do this adjustment for GIC register configuration and rerun the software we see the expected set of synchronization interrupt and normal world OS boot. Additionally the model could generate log file logging specific events and states. This helps in quickly identifying the control flow in the model.

One of the most common requests from software engineers running software on a Virtual Platform is to be able to profile the executing software. We can use the ARM Profiler included in DS5 with the Fast Models from ARM.

Next Generation System Development

Once we developed the virtual platform, the software team was able to use this platform. There was a new requirement from the software team that for certain IPs they wanted to verify the system level tests with the RTL. I was lucky enough to work on various approaches to solve this problem. The first approach we adopted was the particular IP under consideration was converted to SystemC (from RTL using tools). This enabled us to construct the platform with some of the IPs converted from RTL. The next approach that we used was to co-simulate. In this the SystemC and Verilog/VHDL kernel are synchronized. There is no need of converting the RTL model. The software now runs in this mixed platform. The main idea behind these mixed platforms is that the software under test runs on RTL model. For example you are developing a driver for the Ethernet. This Ethernet is not a legacy but has many new features. The initial software development can happen with pure virtual platform. Once we are confident of the driver functionality it can now be tested with the platform having Ethernet as RTL and rest of the system in SystemC. The speed of the platform decreased but it met the need of verifying the software on the RTL.

As more and more system level tests were performed there was need of having more and more IPs in RTL. Now this brings back to the original problem. When the amount of RTL increases in the platform the speed of the platform decreases. To overcome this limitation next generation platform aka hybrid platforms came into existence. These platforms used SystemC+Emulation (in case of Cadence VSP+PXP).

Emulation platform are really fast compared to pure RTL platform. However with the processors in the virtualized world and all critical IPs in emulation the best of both worlds (systemc simulation and emulation) is achieved.

Summary

Growing multicore design poses new challenges in the way software is developed and debugged. Virtual system prototypes provide an excellent platform for software development (VP). The use of hardware style and software style debuggers attached to VP provide a deeper view inside the system which is usually not available on real silicon or may need additional configurations (like ETM etc) to achieve the debug capabilities that is provided in VP. With the SystemC and TLM2 standards it is now possible to develop interoperable VP. These platforms can not only be used by internal teams but can also be shipped to customers. Coupled with advanced use cases of hybrid these help in achieving more software validation coverage compared to traditional platforms.

ARM Techcon Presentation

1) https://www.youtube.com/watch?v=4vA_gFZPI64