



# Reducing the SoC System Management Burden using the SoC System Manager (SSM)

Authored by: Phil Casini

August 21, 2009

## Introduction:

SoC system management is becoming more difficult as complexities continue to rise. Much of the focus has been on power management. But security, error recovery, and even boot and reset sequencing are growing design problems.

This white paper discusses an architectural approach to system management which streamlines hardware and software development and integration, as well as enabling a system management scheme which can be applied to power management, error recovery, security management, and boot sequencing.

By using a system manager such as SSM, these system management tasks can be coordinated in hardware and software using a scheme similar to how interrupt service routines are written and executed. It is possible to script scenarios for system management. These scripts can be mixed and matched in real time as required. The ability to utilize scripts is a key element to easing the system management architecture as a whole. SSM becomes the key “glue” component that streamlines software and hardware integration.

## SSM Architecture Innovations

Two technology innovations are introduced in this paper that is part of the SSM product offering. These innovations enable scripted system management architectures:

1. Independent policy enforcement – Having an intelligent IP block and software combination as an independent entity on the SoC enables SSM to execute system management commands at both hardware and software levels. Thus, SSM becomes a policy enforcement agent with the necessary connectivity in hardware through a bus and in software through drivers to execute entire system management policies. Note that this does not mean SSM has to become the system host. But SSM can execute commands using any script from any where.
2. Virtualization – The ability to use drivers to connect SSM to software operating across the SoC on the various IP blocks (host processors, DSPs, media processing engines, communications stacks on dedicated hardware, etc.) allows SSM to synchronize system management changes it is performing at both the hardware and software levels with no other intervention required. This eliminates the need to perform these same functions elsewhere in the SoCs software, reducing development burden and providing a scalable scheme when product changes are made. Further, this scheme also enables software to control system

management policy enforcement using the scripting techniques. Thus, a virtualization schema forms. Software can simply use an API format to transmit its policy enforcement requests using SSM drivers as the virtualizing agent. Software is now separated from the hardware underpinnings of the specific instantiation, and by varying the scripts, software can change its management profile requests easily from chip to chip as well as from operational mode to mode.

### Using the Scripting Technique

It is often the case that SoCs require certain system management sequencing for various states of operation. Scripting is a way to predetermine the sequences for execution. For example, when booting the device, the host processor typically must boot first if it is to perform system boot for the other IP blocks. So a simple sequence list can be designed which indicates the host should boot first.

But what if the host processor must reset during operation? How is this reset sequencing effect the other IP blocks and how is the effected software elements dispersed on the other IP blocks informed and synchronized so the system is brought back to a fully operational state?

So while the former task is rather straight forward, the latter is quite complex. Now let's use scripting. Consider creating a baseline sequence for normal booting represented as a list of commands which the host can use to form a policy. In a boot sequence for a Video SoC the list may look something like this:

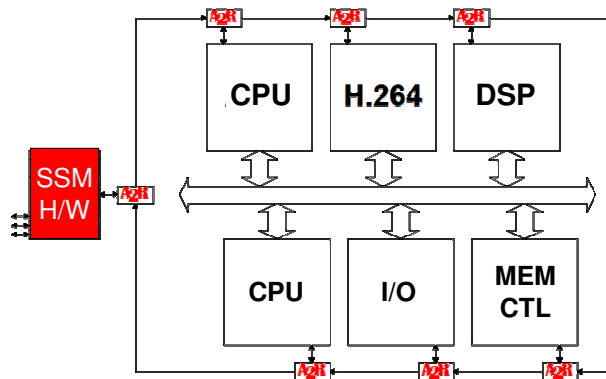
1. Boot host
2. Boot DSP – Start audio software
3. Boot Video Codec Engine – Start video codec software

Let's call this script #1. Now let's consider the case where the host needs to reset:

1. Tell DSP and MPE to suspend operation
2. Host to reset
3. Host to re-synchronize with DSP and MPE
4. DSP and MPE can resume operation

Let's call this one script #2.

Now let's assume SSM has been implemented on the SoC according to the following diagram:



The A2R bus is the hardware level connectivity between SSM and the IP blocks. SSM drivers (not shown) connect to the various software elements that operate on the IP blocks. SSM will assign a number to each of the IP blocks for sequencing. A software kernel running on SSM hardware manages the software level communications.

The A2R bus also contains a 16 bit data bus for message passing with the IP blocks. Thus a mailbox scheme can be utilized to set conditions by which SSM executes certain commands. The payload is user defined so that any message passing scheme can be used to signal SSM when it is time to execute a command. Conditional execution of certain commands in the script sequence is possible then using message passing.

SSM drivers use an API for communicating with the software elements on the SoC. The kernel that operates within SSM creates the software level of operations independence.

Now back to the example. At boot time, instead of having the host software coordinate the boot sequencing for the other IP blocks, it simply sends script #1 to SSM and commands it to execute. SSM takes care of the hardware level booting and also provides status for sequencing the software boots as well. The host need not intervene.

Now let's cover the case when the host needs to reset. Using SSM, the host sends script #2 to SSM and commands its execution. In this manner, the host is sequenced in the proper order while the states of the other SoC elements are guaranteed to be managed correctly. No other intervention is required to coordinate the sequencing. And as an aside, any element in the system can initiate script #2 (so if the host is hung up, another block can act on a system time out and issue the script). But how is this guaranteed?

Because the A2R message passing scheme is used to communicate through the hardware and up to the various software on the DSP and MPE IP blocks to coordinate their safe transitions as the host moves through its reset process (the

actual host reset is being controlled by SSM). The host is not involved. The advantages of this method are quite significant:

1. Guaranteed state synchronization across the SoC dramatically limits operational artifacts, each of which has to be modeled and dealt with, and streamlines worst case system level debug.
2. Having a third party enforce the policies dramatically simplifies software schemes other wise needed to perform the same tasks, and as a ripple effect, also simplifies the hardware schemes necessary to enforce the software policies.

From these examples it is easy to see the enormous flexibility that SSM brings to SoC system management:

1. Any IP block at any time can become a policy maker driving SSM. In the event that the host is not involved in an operations sequencing, additional scripts owned by other SoC elements are executed. This is extremely useful in distributed processing architectures whereby no one processing element is always the master. .
2. It is possible now to have subsystems not usually associated with system management become policy makers. For example, by developing intelligent external memory subsystems, they can become the policy maker for initiating power sequencing. This allows coordination of an ultra-fine grained power management scheme that is governed directly by the priorities within the queuing of the requests which are managed by the memory subsystem. These power down scenarios are based on data access priority (in addition to by processor cycle) and extend power savings.

Summary:

The ability to script system level management simplifies otherwise enormous architecture complexities when designing power management, security, error recovery, and even boot sequencing schemes. Scripts homogenize and consolidate these management tasks. Virtualization also enables these scripts to be developed in parallel with the SoC development, debugged and tested as a set of “gold” scripts that can be reused many times, and also used as a source for work-arounds, thus saving project schedule time.

Employing SSM to manage script schemes offers a massive reduction in architecture complexity and hardware software integration.