

IP/SOC 2006

Session : Reconfigurable IP/SoC and power consumption

Fast Virtual Prototyping for early software design and verification

Amit Garg, CoWare

Noida, India

Fast Virtual Prototyping for early software design and verification

Abstract: Continuous advance in the fabrication technology has exponentially increased the amount of logic that can be put in a SoC or an ASIC. With this increased logic the demand for functionality and performance expected from the SoC is also increasing exponentially. The traditional design cycle time for a SoC varies from six months to two years for the first prototype and at least six months for subsequent spins with an average cost of a million dollars per spin. Given these constraints, the viable solution is to increase the programmability of the SoC; as this allows newer features to be delivered as software upgrades, which is resulting in the explosion of software content on the SoC. With this explosion of software content in the SoC, software development and validation has become the bottleneck for time to market. In the traditional design flows, software validation could start only after the first prototype of the silicon was available. Software validation cycles range from six months to a year. In current scenarios, where the product life span itself is six to twelve months, this paradigm is clearly unacceptable. As a result of this shift, the industry has been resorting to various simulation techniques that aid early integration and verification. These simulation technologies range from cycle-accurate RTL simulations, to purely instruction accurate ISS based simulations, which are purely functional and usually provide for processor centric verification. Both these approaches have their advantages and disadvantages. However, neither of them scales to provide software developers with a platform where they can do meaningful development in the context of the entire platform. These limitations have constrained the penetration of these approaches in the industry. Also, generally, these simulations lose their significance and applicability, once the first cut silicon is available. Thus, the effort proves expensive, being limited to the initial product only, with no or limited reuse capabilities.

This paper describes a simplified and novel approach for fast and easy virtual prototyping which provides a fully functional and accurate platform for the embedded software developer. This platform enables the software developer to execute the same binary image on the virtual platform as he would on the real hardware. The

virtual platform, thus developed, can also be used for meaningful development of newer software features and upgrades, and will carry a lot of value for software development.

I. Introduction

Virtual prototyping is a technique where a software simulation of the entire hardware platform is created using simulation models of the various blocks in the system. These models are provided either by the IP vendors, or developed in-house for custom IP blocks. For software development these models are required to be functionally accurate, and also contain all software visible interfaces. Subsequently, the platform is assembled either to mirror the real architecture with the entire bus hierarchy, bus bridges and multiple clock domains etc. or alternatively at a higher abstraction level, but still maintaining the accurate view for the software developer. In this paper, we take the latter approach. Since these models are at a higher abstraction level creating them is faster and simpler than the corresponding RTL blocks. Moreover, some of these models are already available for popular, off-the-shelf IP blocks, and can be reused. For example, models for standard components like UART, USB etc. can be bought either from the silicon IP vendor or from various ESL vendors.

This initial platform can be adopted separately by the hardware and software teams to suit their needs. Interface accurate, functionally correct and partially timed models that are discussed in this paper are sufficient for software functional verification. For hardware development and verification this platform can be further refined towards a more cycle accurate version. The rest of the paper discusses how a higher abstraction level of platform modeling can yield substantial gains in software development life cycle of a SoC or ASIC based system.

II. Requirements

Virtual prototyping can be performed at various abstraction levels to match the requirements of a specific stage in the design flow. The trade-off at any given abstraction level is between the simulation speed and the accuracy desired. The accuracy desired is different for different users. For software development the model should be correct on the following aspects:

1. Software visible interfaces should be modeled correctly. This includes the register interface of the block, which should be bit true and functionally accurate.
2. Accurate modeling of timing visible to software. This would include things like modeling the baud rate of a serial device correctly, software timeouts etc.
3. Correct modeling of the platform architecture that is visible to software. This includes artifacts like memory map of the system.
4. Modeling bus communication artifacts visible to software indirectly. For example, things like accesses to invalid addresses.
5. Architecturally correct to the extent needed for software development. For example, things like bus bridges, clock domain bridges etc. may not be of interest.

Following things are not necessary for software development and verification hence is not modeled in the virtual platform:

1. Timing of the bus communication and different phases of the bus protocol. Access between various blocks can happen as purely blocking untimed function calls across a simple read/write interface.
2. Timed transfer of data and control across various external interfaces. For example, from a USB model that wants to transfer a certain amount of data, there is no need to transmit it serially but can be done at a higher level and the entire data can be transferred in a blocking untimed function call, but the model ensures that the next logical quantum of data is transferred only after the time required to transfer the current data has expired, because this timing will be visible to software.
3. Instruction timing and pipelining in the processor are neither visible to software nor does software has any control on it. Due to this reason, if the ISS is modeled with an approximate throughput rate then the system timing is sufficient for most of software development needs. For example, if the processor is executing at 200 MHz and the throughput of the architecture is 1 instruction per clock cycle, then it is

possible to model the processor with an approximate timing of 5ns per instruction.

For hardware design and verification this top level functional simulation can serve as an executable functional specification. This executable specification can be refined further by replacing the functional models with either the cycle accurate simulation models or the corresponding RTL blocks as they become available.

III. Case Study

3.1 SoC Architecture

This section will illustrate the approach of this simplified virtual prototyping with the help of a simple case study. Figure 1 is the top level architecture diagram of the SoC that is the target for virtual prototyping. There are two sets of software development that is the target of this project:

1. Development of various device drivers.
2. Control and interface applications on the control processor.
3. Development of the interface layer between the DSP core and the application processor.

Note: DSP algorithm development is not the focus of the project.

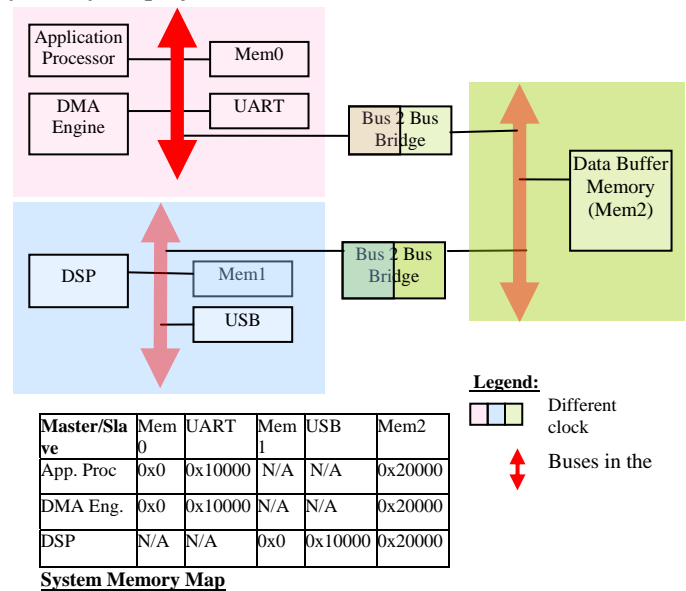


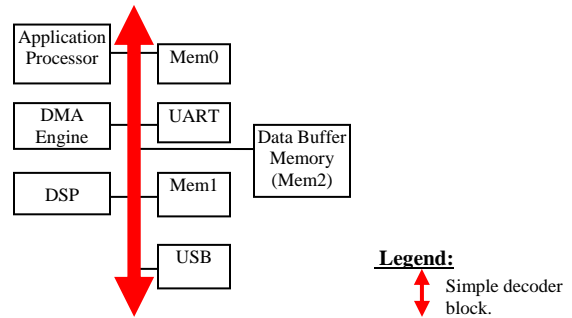
Figure 1: Physical Architecture

3.2 Virtual Prototype Requirements and Architecture

For the above illustrated platform as the reference design following components need to be modeled with the described abstraction levels for effective software development.

- 1. Application Processor and DSP:** Loosely timed and interface accurate instruction accurate models are sufficient for these blocks. There is no requirement for a strict timing accuracy.
- 2. Memories:** These will be simple data stores which have to expose an interface through which the bus model can access them. These can be completely un-timed blocks. (*Note: For memory blocks like FLASH this cannot be done and they will be modeled like peripherals*).
- 3. Buses:** These blocks have to be simple decoders and ensure that the access originating from the required master is being sent to the desired peripheral or memory. In addition, they should have an error detection mechanism for cases where a bad address is floated. These blocks do not have to be cycle accurate blocks and there is no need for them to understand the various phases of a bus transaction for that matter.
- 4. Peripherals (UART/USB):** These blocks have to model the entire register interface in a bit accurate manner, also with this bit accurate interface the corresponding behavior for all the registers needs to be modeled. Also, these blocks should have some notion of time in them which will help them determine things like baud rate and data rates, as this is needed for software to validate things like buffer overruns/under run scenarios.
- 5. DMA:** This again can be a simple data transfer component that has to be modeled with functionally correct interfaces like bus interface, interrupt etc. and also has to have a limited notion of time, which will help it determine things like the time required to complete a data transfer etc. This block is loosely timed and ensures that there is some notion of time between the beginning and end of transfer which is proportional to the configured data rate.

Given the above paradigm, the virtual prototype for this problem can be reduced to:



Master/Slave	Mem0	UART	Mem1	USB	Mem2
App. Proc	0x0	0x10000	N/A	N/A	0x20000
DMA Eng.	0x0	0x10000	N/A	N/A	0x20000
DSP	N/A	N/A	0x0	0x10000	0x20000

System Memory Map

3.3 USB Model Details

This section will highlight the various features and artifacts of the USB model that will be modeled for this kind of virtual platform.

- 1. Bus Interface:** A simplified bus interface will be required that will allow the simple bus model to forward all the USB accesses to this model. This can be a simple read/write interface and it does not need to model the exact bus protocol that will be implemented in the real system.
- 2. Registers:** All the registers in the model that are software accessible will need to be modeled in a bit accurate manner. Also, for each of these registers the corresponding behavior that will be triggered within the USB device will need to be modeled too. In addition, it is required that all the status and control registers that the driver will have access to also reflect the correct status information of the model.
- 3. Timing:** A limited notion of time will need to be introduced in the model that will allow it to model conditions like under-runs and overruns, and also it will allow the model to update its status registers and interrupts in a timing approximate manner.
- 4. Control Interfaces:** USB model will need to model all the other control and access interfaces in a simplified abstract manner so that it can signal interrupts and get other control information (for

example, trigger to start a transfer from an external peripheral).

5. **Data Flow model:** USB model will have to model the data flow with a limited notion of timing. Depending on the requirements it will be required to model either an additional data port that will allow the model to be connected to the test bench for driving the stimulus or this can be built into the model itself and then we do not need any additional test bench data ports. For the data flow in to the simulation, the bus interface can be used by the DMA controller to pump in the data and also for the USB model to put data into the data buffer memory.

3.4 Sample Software Development Flow in Virtual Prototype

This section will highlight how the real life software validation can be done for a USB driver given the virtual platform that has been modeled at the abstraction level highlighted earlier.

Requirements for the USB driver development:

1. There should be an RTOS available to run on the application processor. Porting of the RTOS can easily be accomplished on the loosely timed and interface accurate IA ISS of the application processor that is present in the virtual prototype, which is attached to a simple timer block (not depicted in the architecture above). This timer block is needed for scheduler of the RTOS, and is expected to fire an event at the desired intervals of time which usually is in the order of a few milliseconds.
2. The device driver should be able to access the USB device in the same manner as it would on the real platform. This requirement is fulfilled by having the correct memory map of the system and by having a bit accurate register interface in the USB model.
3. There should be a way to simulate various scenarios like buffer overruns and under runs in the device. This can again be achieved with the underlying simulation technology that provides a notion of time. The granularity of time

required here is lot smaller than for running a cycle accurate simulation. A simple example is, for the driver to test a data transfer of 1024 bytes at a baud rate of 2MBps. For this a simple timeout will be required at every 5 ms of simulation time, as opposed to an event required at 5 ns for a processor running at 200 MHz. With this limited notion of time it is possible to develop the virtual prototype at a speed that will be acceptable for software development.

4. An interrupt is notified to the processor on the completion of the specified data transfer by the USB. This can be achieved by building on top of any of the simulation technologies available like SystemC, which allow for asynchronous communication between different components.

3.5 Effort and time estimates

Following table gives an approximate effort and time lines for creating such a virtual prototype that can be used for software development and validation.

Block Name	Feature Description	Effort
App. CPU	a) Instruction accurate ISS. b) Interface accurate for control interfaces like resets/interrupts etc. c) Processor resources like MMU and cache functionality that is visible to software.	Available from IP vendor or ESL vendor. Off the shelf component.
DSP	a) Instruction accurate ISS. b) Interface accurate for control interfaces like resets/interrupts etc. c) Processor resources like MMU and cache functionality that is visible to software.	Available from IP vendor or ESL vendor. Off the shelf component.
DMA engine	a) Programming mode support. b) Accurate register interface c) Interrupt/notification functionality.	1.5 weeks.

UART	a) Accurate register interface. b) Limited notion of time. c) Interface for external stimulus. d) Interrupt notification.	Off the shelf IP block, or developed in house. Estimated effort is 1 week.
USB	a) Support for register accurate interface. b) Support for different modes. c) Support for recognizing external stimulus. d) Interrupt notification.	3 weeks.
Memories	Simple data store memories.	2 days
Platform assembly & configuration	Assemble the platform in the simulation environment for use.	2 days.

The total time spent on the creation and assembly of this simple SoC platform before it is ready for use is approx. five weeks. So five weeks from the moment the top level specification of the SoC is frozen software validation can start, as against the conventional flow where this would not have been possible at least for six months from the completion of specification.

IV. Conclusion

This approach for virtual prototyping where the architecture and functionality is modeled to be just sufficient for software development and validation provides a virtual platform for software development at a very early stage of SoC design.

Following table summarizes the advantages and gain of this approach against the currently prevalent industry approaches:

Feature	Eval. board	RTL Sims	ISS	Cycle Accurate	Fast Virtual Sims
Simulation speed	High	Very Low	High	Low	High
Availability of virtual prototype	Very Late	Late	Early On	Mid of hardware dev.	Early on
Development time	Very high	Very High	Low	Medium	Low
Software changes for simulation	None	None	High	None	None
Visibility in platform internals	Very low	Very low ¹	Limited	Limited	High
Setup and execution time	High ²	Very high	Low	Medium	Low
Usability for post silicon development and revisions	Limited	None	Almost None	Limited.	Very high.
Additional development cost	Very High	None	Very Low	High	Low
Extent of software verification possible	100%	~5-10% ⁴	~60%	~20-30% ³	80-90%

1. Very limited because software developers are not expert at looking RTL wave traces and using other hardware trace tools.

2. This includes board layout and fabrication.

3. Due to limited availability of other platform components like the peripherals etc.

4. Due to extremely slow simulation speeds

Additional advantages:

1. No additional cost to have SoC boards for each software developer.
2. Easier management and bug fixes to the virtual platform for corresponding fixes to be available in the real SoC whereas for an SoC based board it might be a few hour to a few weeks job.
3. Executable specification that can be enhanced and upgraded with silicon revisions and increase the time window available for validation of new software features.

The summary above and results clearly indicate that this simplified approach to virtual prototyping can lead to significant advantages both in terms of usability and also in terms of cost savings and time to market window for a SoC or ASIC based product development cycle.

V. Limitations

The approach highlighted in this paper is sufficient for doing the majority of functional verification of the software and it also allows the software developer with a platform very early on in the design cycle to accomplish the task, but since there are certain trade-offs that have been made to aid faster development with regards to the accuracy there are a few potential issues that may not be covered. Following is a list of kind of problems that might not be validated using this virtual platform:

1. Race conditions between hardware blocks and software can be masked because the hardware software concurrency is modeled in an approximate manner as illustrated above.
2. Memory controller timings and software configuration of such parameters cannot be validated. Since the memories for most cases are modeled as simple read/write arrays, it is not possible to validate memory timing parameters configured by low level software (for example, timing parameters for DRAM).
3. Corner cases in software/hardware interaction that are triggered by timing differences between the approximate models and the real cycle accurate behavior. Such, cases should not be present in the properly designed software but they are a fact of life anyways.

V. Acknowledgment

The author would like to acknowledge the contributions of Sanjay Chakravarty and Tom De Schutter for their valuable review comments and help.