Creating Virtual Platform using The OCP-IP Modeling kit

Puneet Arora, Ruchir Bharti, CircuitSutra,
Noida, India
Mark Burton, Greensocs
UK

Abstract :

An important use case for modeling of hardware IPs is to use the models to create Virtual Platforms. In a Virtual Platform, models of different IPs are stitched together, to be used to simulate the functionality of entire SoC (rather than mere components of a larger system). Different IPs have varied interface requirements, therefore connecting software models together can become a time consuming and error prone task, especially if the individual models use non-standard interfaces. The OCP-IP modeling kit provides a comprehensive resource that standardizes the way all IP that uses the OCP-IP bus interconnect is modeled. Hence greatly simplifying the difficulty in building virtual platforms.

This paper presents work that has used the OCP-IP modeling kit[1], and shows the benefits that the kit brings.

I. Introduction

Many methodologies and techniques are being developed in the ESL domain, with the objective of enabling models re-use, and to facilitate the construction of Virtual Platforms, providing the functionality of an entire SoC. This complete Virtual Platform, can be run on a host machine(PC) to emulate the behavior of different modeled architectures. It can be used to boot the complete software stack as would the actual physical platform and thus can be used for architecture exploration, hardware-software co-development and verification.

The essential requirements of the individual models is to provide fast simulation, and to make use of standard interfaces to ease integration : in both cases overall turn around time for verification is greatly reduced. Both model speed, and, importantly, more integration times are becoming important and key factors while designing large models. As a complete SoC consists of many hardware IPs, a VirtualPlatform consists of various models connected together.

Providing standard and generic external interfaces to the individual IP models becomes crucial to their reusability and interoperability. For this purpose we present work based on the OCP-IP modeling kit[1] which provides a set of rich and configurable features for the interfaces of systemC modules. The OCP modeling kit is built on top of OSCI's TLM-2.0 technology, adding support for the OCP protocol features. This Kit facilitates the creation of models at various abstraction levels (TL1, TL2, TL3, TL4) and supports all the use cases including verification, architecture exploration and software development. The TL1 Abstraction corresponds to fully cycle-accurate modeling and the TL4 abstraction level is equivalent to the base protocol defined in OSCI TLM2.0 with loosely timed modeling style. In this project we have created a virtual platform at the TL4 abstraction level, using the TL3/TL4 APIs provided in the OCP Kit.

For our setup, we have taken the processor model from QEMU, which is a 'C' based processor emulator. Then SystemC models of IPs like DMA, UART and InterruptController, which form the guts of both i386 and ARM based platforms are connected. These models have been developed and plugged into the Virtual Platforms, which includes the QEMU CPU model. The interface of these models uses the features from OCP kit. These models can be accessed by applications running on the virtual platform. Using this approach, SystemC models of other IPs can also be plugged into the VP and can be easily accessed by the application/device driver running on the system. This provides for the use case of early software development and analysis.

We have also used the GreenSocs infrastructure which is an open source infrastructure for developing standards based tool independent SystemC models.

This virtual platform will be donated to OCP-IP and will be available for download through the OCP-IP website in near future.

This project demonstrates how easy it is to create a complete virtual platform of a SoC by using the standard modeling kits (OCPIP TL Kit / OSCI TLM2.0) and the opensource infrastructure (GreenSocs, QEMU, Openmoko etc..).

This platform can be used by a SoC company as a base to create the virtual platform for its own SoC. It is becoming a standard practice for the IP vendors to provide a TLM model of their IP blocks. Our platform can be used by IP vendors to try out the models of their IP blocks and run the software stack / device driver. They can simply create the TLM model of their IP using the OCP-Kit, plug in their model into this virtual platform, and access it through the software running on top of the virtual platform.

The benefits of creating the virtual platform using SystemC/TLM2.0 have become well known by now in the industry. However, many companies seem to struggle to benefit from this new technology. There is confusion and mistrust, will it really work for their SoC, how much investment will be required etc. This project is an attempt to increase the awareness in the industry, and to accelerate the adoption of SystemC, TLM and ESL in general. Atleast for creating the initial proof of concept, no heavy investment is required and no commercial ESL tool is required to be purchased.

II. Starting Point : Qemu and Openmoko

As mentioned earlier, Qemu is a 'C' based generic and open-source processor emulator which supports emulating several hardware platforms including x86, ARM, SPARC and MIPS. It lets you run OSes and programs made for one machine on another machine. Before using Qemu, its source code needs to be configured(for a specific target architecture) and compiled on host machine.

For our setup, we have used a particular flavor of Qemu called qemu-neo1973[3] which provides ARM based hardware platform for mobile devices.

Openmoko is a hardware and software project, for creating open source mobile phones. Openmoko software is linux kernel based opensource OS which can be run on Openmoko hardware platforms.

The qemu-neo1973 flavor, provides emulation of Samsung microprocessor(S3C2410A)[2]which is integrated with peripherals like TouchScreen, bluetooth headset, I2C devices, NAND flash etc. The Samsung microprocessor consists of ARM core and necessary

peripherals including DMA, interruptController, Timer, RTC, NandController etc. It minimizes overall system costs by providing a complete set of system peripherals around its ARM920T RISC processor.

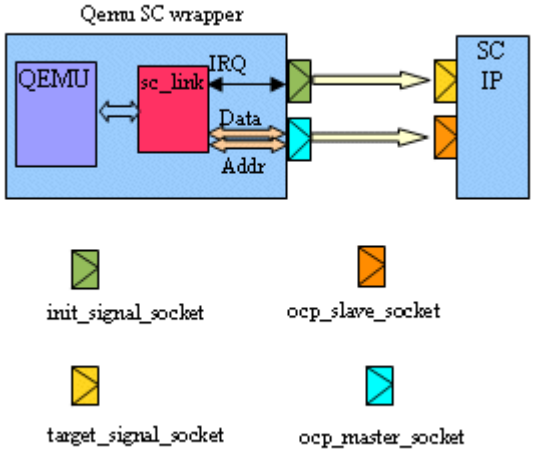
III. Integrating SystemC models in Qemu

The native 'C' code of qemu, has its own simulation infrastructure, which is invoked when the qemu is launched. It connects the processor core to the model of peripherals through 'C' callback functions and accesses the registers of the models through them. We have replaced the existing models of the DMA, UART and InterruptController IP with corresponding SystemC models.

To provide access to these SystemC modules we have used a methodology, “Qemu InABox”, devised by GreenSocs to connect the two simulation environments. Qemu contains an endless loop that performs all the steps necessary for a proper system simulation. In the InABox approach, Qemu code is wrapped in a SystemC wrapper. So Qemu can be used as a standard TLM2.0 initiator and can be instantiated in any TLM2.0 simulation. This wrapper acts as a SystemC master to which the modules of IPs can connect as slaves[8].

The accesses to SystemC devices is managed by registering the devices and capturing read and writes to them (i.e. predefined address spaces). This link is provided in a separate 'C' file(sc_link_arm.c). In this file , read/write calls for specific IPs are registered and these get forwarded to proper functions in Qemu SC_wrapper during simulation. These functions then perform the TLM2.0 transaction through the socket to the TLM2.0 target device. When the SystemC models need to communicate back to the qemu world, for example when the InterruptController raises an interrupt to CPU, then the reverse path is followed, i.e. the model sends the signal to the Qemu SC_wrapper which raises the appropriate IRQ(interrupt request) line of qemu.

Following diagram depicts the generic connectivity through Qemu SystemC wrapper:



So, sc_link receives calls from Qemu and breaks them into data and address which can then be used by SC_master to transmit on its socket connected to IPs.

When the Qemu is running, applications can access the underlying peripherals (SystemC models) as if they were connected to system bus. Here the models need to use a standard interface, so that all of them can easily connect to the same SC_master.

The essence of using standard interfaces becomes more relevant when the different models are being sourced from different teams, which would use different interfaces while development and optimizations, leading to a major rework activity during integration.

IV. Building Blocks(IPs)

The models use standard modeling kits(OCP IP TLM and OSCI TLM2.0) and opensource infrastructure from GreenSocs, Qemu and openmoko. Using the approach described above we have connected SystemC models of DMA, UART and InterruptController, and models of other IPs and peripherals can be similarly connected. The specifications for these IPs are given in [2].

The virtual platform is created at the PV abstraction level, keeping in mind the use-case of embedded software development. The blocking transport APIs are used for TLM transactions, through TL3 level OCP sockets. The OCP kit provides for maximum interoperability with OSCI BaseProtocol. At PV level the detailed timing information and bus specific features are not modelled. The OCP kit provides support to model these and also at cycle accurate level, but for initial early software development, this level of detail is not required.

DMA

The DMA controller can serve up to four channels in three different modes. Requests can be initiated by hardware as well as software and each channel has nine config and status registers. TLM2.0 base protocol is used through ocp_slave_socket to access these registers. Further, since DMA is required to read/write physical memory to serve the requests, it acts as an initiator also.

InterruptController

The interrupt controller can handle requests from various sources like DMA, UART, IIC etc and it uses two interrutp lines(FIQ and IRQ) to the CPU to serve multiple internal and external requests. The various control registers in the IC can be accessed through its ocp_slave_socket using TLM2.0 base protocol.

UART

The S3C2410A UART provides three independent asynchronous serial devices, each of which can operate in Interrupt based or DMA based mode. Since, by default, TLM2.0 Base Protocol provides for memory-mapped bus transfers only, so it was extended for non-memory mapped serial communication to model the data transfer through UART. This TLM extension for serial protocol is covered in [7].

The OCP sockets, that are used to access the memory-mapped internal registers of models, also provide memory management to the user. The memory pool is associated with Master sockets and APIs are provided for Transaction memory management and Data and Byte enable array memory management. The user can give the desired allocation scheme to the socket during its construction. For our purpose we have used the transaction memory management provided with the socket.

Besides memory management, the OCP sockets also provide run time bindability checks. These checks are provided by means of configurations that are attached to

them. So if two sockets with conflicting configurations are bound to each other then at run time (i.e. elaboration phase) this conflict will be reported to user and has to be fixed before proceeding. Certain extensions are defined in the OCP-IP, and have attributes like Phase association, Mutability, Bindabaility and Extension types assocaited with them[1].

All extensions have Bindability levels(BL) associated with them, which are defined in conjunction with the role(Master/Slave) of the socket. BL can be one of : mandatory, optional or rejected. Mandatory means that the extension is necessary for the functioning of module, Optional means that the module can work correctly with or without the extension, and Rejected means that the module is unable to handle the extension.

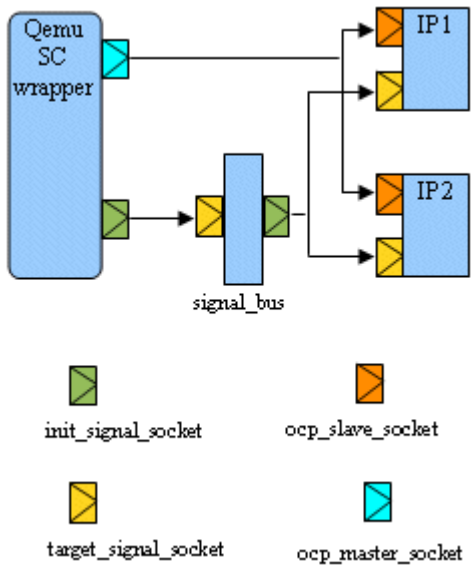
For modeling IPs (DMA, IC and UART) at PV level, the extensions were not required therefore the BL of extensions was kept as rejected or optional. When such models are used in other environments then the config of master_sockets on the other end should be set to appropriate bindability level. Hence if the master is modelled at some other abstraction level and uses some extensions then the slave models will give the configs mismatch error at runtime.

The SC_master contains an ocp_master_socket_tl3 (multi socket) which is connected to the ocp_slave_socket_tl3 of each of the model. This socket-binding is used to access memory mapped registers of the models.

The models also interact with the external world by means of interrupt lines. For example, DMA has four input DRQ(data request lines) and four output IRQ(interrupt request to CPU for bus access) lines. To speed up the simulation, sc_in/sc_out ports are not used because they use event based communication mechanism which has its overheads due to context switching in the kernel.

Instead, The GreenSocs SignalSockets[4] are used to communicate the values of signals across IPs. They communicate by setting appropriate extensions in the payload, and then by simply making transport calls on the SignalSockets.

These SignalSockets are TLM-2.0 based and provide many of the same features as the OCP-IP kit, but just for signals. They are extremely easy to use, and cover all system internal signals. The signal sockets are not connected directly to each other,instead they connect through a SignalBus[5], which acts as a router for signals. So even if a model drives signals for more than one target, it need only contain a single init_signal_socket. And similarly even if a model receives signals from more than one sources, it need only contain a single target_signal_socket. These are then connected to the init and target sockets of signal bus. At the time of elaboration, SignalBus generates an internal map by reading configuration of the targets to which it connects and uses this map at the simulation time to appropriately route signals to targets. The following diagram shows the connectivity of the entire setup:



The memory mapped registers inside the SystemC modules are modelled using GreenReg[6]. Using greenreg provides for a well tested method of handling pre-read, post-read, pre-write and post-write functionalities on registers. The entire structure of building models using standard

components like GreenReg, and connecting them using OCP interfaces and SignalSockets is very easy to use and can be easily extended to plug-in SystemC models of more peripherals.

V. Conclusion:

Virtual Platforms at PV abstraction level serve an important use case of software modeling i.e. early software development and analysis. This allows software developers to start developing applications long before the actual hardware is available in the market.

The SystemC Modules created in this exercise use sockets, memory management and run-time bindability checks from the OCP-IP kit. They use TL4 abstraction level and can be refined to lower levels where other features of OCP-IP kit like pre-defined TLM Extensions, TimingConfigurations can be used.

We have demonstrated the ease with which it is now possible to create a Virtual Platform using SystemC modules by using standard and open source infrastructure already available. Usage of standards like the OCP-IP modeling kit makes connecting models from different teams very easy and also greatly enhance the reusability and interoperability of models.

VI. References:

1. OCP kit manual (**www.ocpip.org**)
2. S3C2410A Users Manual **http://wiki.openmoko.org/wiki/Samsung_S3C2410**
3. Openmoko under qemu **http://wiki.openmoko.org/wiki/Qemu**
4. SignalSockets documentation **http://greensocs.com/en/Projects/GreenSocket/SignalSocket**
5. GreenRouter documentaion and SignalBus examples **http://greensocs.com/en/Projects/GreenParts/GreenRouter**
6. GreenReg documentation and examples **http://greensocs.com**

7. TLM extension for Serial Protocol **https://svn.greensocs.com/public/packages/serialsocket/docs**
8. Qemu InABox **http://greensocs.com/en/Projects/QEMUSystemC/docs/QEMUSystemC/QEMUInABox**
9. OSCI TLM2 User Manual(version JA22)