SystemC, transaction level modeling simplifies capture

By Allan Cochrane, EDA Specialist, DMIT Group, Jon Connell, Engineering Manager, System Modeling Group, Andy Nightingale, SoC Validation Manager, IP Products Division, ARM Ltd., Cambridge, England, EE Times

April 18, 2003 (9:06 AM EDT)
URL: http://www.eetimes.com/article/showArticle.jhtml?articleId=16500866

IP companies have heralded a new age in platform-based design for years - ever since semiconductor integration capacity reached the point where entire systems could theoretically be integrated into a single die. So, why haven't we seen a huge explosion in platform-based design?

The key is the scope of the platform: only now are platforms being defined which include a wide assortment of elements from System-level design (SLD): the RTL hardware definition, bus architecture, power management strategy, device drivers, OS ports, and application software.

However, to be successful, a platform will need more than this. An essential element for enabling differentiation will prove to be an advanced systems modeling and verification environment. Developers require a variety of views of the entire platform from RTL, system models, software development models, and real hardware development boards.

Each view of the platform reflects the same system architecture, and designers can use test software in any of the higher-level views, providing a high degree of confidence in the design prior to tape out. This provides a valuable environment in which to investigate system bandwidth and performance requirements.

System views must be extendible, allowing designers to exploit the advantages of a well-supported, pre-verified base platform of hardware and software IP, while differentiating their own application with their own IP. Specifically, each design task has specific requirements on methodologies and IP customers will want to make extensions to the IP during each stage of their own design.

At the system-level, availability of software becomes critical and it is no longer reasonable for the software team to wait for a prototype system. Coverification can move the integration schedule forward to the point where RTL is available, but this still delays the software integration to a point where much of the hardware design is complete. System and software designers would still be lacking a common environment.

SystemC provides a solution to this dilemma. It is the standard design and verification language that spans from concept to implementation in hardware and software and which can also be used to develop models. Prior to the introduction of SystemC, there were many proprietary C or C++ based environments. Since these environments are not based on an open standard, their usefulness is limited since model availability from IP vendors is non-existent. SystemC has become the de facto standard for system level design. As a result, IP vendors are starting to provide SystemC compatible models of their IP. As model availability increases, so too will the adoption of SystemC increase.

Consider a design flow where the system is first specified using SystemC, then partitioned into hardware and software blocks and handed to the respective teams. This executable specification is a key enabler for both teams. The software engineer not only has a platform for the development of his software, but also a C++ based simulation environment.

One of the key techniques used in this design flow is the modeling of the system at the transaction level. Transaction level modeling (TLM) is simply a higher abstraction level for modeling. Systems modeled in RTL are concerned about the hardware details such as pin-level behavior of their system. With TLM, it is possible to accurately model many aspects of a system at a higher (e.g. Read and Write) level. By using TLM we are simplifying the modeling effort and we also gain simulation speed.

The types of models available trade off performance versus accuracy and as such are suitable for different applications. Generally, the more accurate a model, the slower it is. The less accurate a model is, the faster it is, and software engineers typically use these models.

TLM seeks to bridge the gap in the current available hardware/software integration methodologies. By raising the level of abstraction of the hardware described for the system, designers can produce more efficient system simulations. Speeds of 100 kHz for cycle-accurate simulation and 1 MHz for cycle-count approximate simulation are achievable using the abstraction technique in a suitable modeling language such as SystemC.

Cycle to protocol

Cycle-accurate (transfer-level) modeling provides a cycle-by-cycle mapping from one bus cycle to its representation in a complete bus protocol. Speed increases are achieved by a combination of high-value IP models written to simulate efficiently in such environments and the reduction of a number of signal events in a protocol cycle to a single cycle operation. In TLM, the level of abstraction is raised higher still to sit on top of the protocol level such that a single transaction can be converted to cycle-based timing using an adapter. An entire ATM packet or an AMBA AHB burst can be represented by a single transaction, but the timing of these transactions will always be known because the protocol is well understood.

In SystemC, a system may be modeled as a collection of modules that contain processes, ports, channels, and even other modules. Processes define the behavior of a particular module and provide a method for expressing concurrency. A channel implements one or more interfaces, where an interface is simply a collection of method (a.k.a. function) definitions. A process accesses a channel's interface via a port on the module.

To understand the architecture of a complex TLM system, consider a simple example consisting of four IP components: two masters and two slaves are connected through a shared communications fabric supporting the AHB Multi-layer bus protocol. The architecture exists within a single clock domain. Masters, slaves and the hierarchical channel model are all clocked components. Arbiters and decoders, as part of the bus fabric, are reactive, un-clocked models.

The communication fabric is modeled by three components: a hierarchical channel model which manages state of component connectivity, an arbiter which receives requests and allocates the shared channel based upon the priority of the requester (master), and a decoder which resolves addressing into specific block connections (transfers from/to). The arbiter and decoder, though distributed in HW, are expected to be modeled through use of monolithic SystemC blocks (single interface to each).

Masters in such a system might be processor models or memory controllers, while slaves might be peripherals such as a UART. Masters are clocked models that create data structures representing bus transactions; masters pass these structures to the bus fabric. Slaves are models that are invoked by the bus fabric and are given the bus transaction data structures to operate upon.

A critical part of this methodology is the ability to migrate to RTL design and retain a verification infrastructure for the complete system. The introduction of RTL models is achieved using adapters to the transfer interfaces of the SystemC models. Since there is a known mapping between AHB transactions in the SystemC world and the signals of a physical AHB, such adapters can be generic and guarantee the correct operation of the device.

A single monitor, waveform analysis tool can connect through a monitoring interface to the arbiter component. As the arbiter maintains a list of channel requests and masters the allocation of the shared channel, this component maintains specific information about the current and pending state of system connectivity not maintained elsewhere. Simple transfer monitors may also be placed on master/slave interfaces.

In System C, an interface provides a set of method declarations, but provides no method implementations and no data fields. Interfaces are used to define sets of methods that channels must implement. Ports are connected to channels through interfaces. A port that is connected to a channel through an interface sees only those channel methods that are defined by the interface. A port is not able to access any other method or data field in the channel. SystemC 2.0 allows users to define their own interfaces.

The channel would typically be the bus fabric. To connect a component (master/slave) to a complex channel like a bus fabric, a component interface is instantiated. There are three types of components which instantiate interfaces to the bus-model: bus masters, bus slaves, and bus-fabric components (decoders/arbiters).

A transaction interface can be of two types, a single-transfer or a burst-transfer. A single-transfer (burst of length 1) is the execution of request-and-retrieval for a single bus-width word of data. A burst-transfer is the execution of request-and-retrieval for a block of data, several bus-width words in length. Burst-transfers are executed as a set of single-transfers methods, but eliminate the requirement that master and slave communicate on every clock cycle during the transfer.

Accurate simulations

With these operational caveats restricting burst-mode transfers, simulation is completely cycle accurate with the exception of support for data interleaving. In burst-transfer mode, cycle-by-cycle multiplexing of concurrent burst-requests to a single slave by multiple masters may sometimes be approximately modeled (close to cycle accurate). This approximation occurs only on the slave side when it is accessed by multiple concurrent bursts. The cycles that occur on the bus are completely accurate.

A component interface is generally assembled from transaction interfaces. Three types of basic transaction interfaces support the component interfaces: blocking, non-blocking, and direct access (or debug access). These are treated as clearly separate classes due to their different handling requirements. For IP re-use and system-validation, it is critical that the use of each transaction type be clearly delineated: Non-blocking access, blocking access and direct access.

The most obvious type of master TLM/SystemC component is a processor model. A processor model is a program that simulates the behavior of the target processor within the context of the overall SystemC simulation. In a typical embedded developer's toolkit, an integrated debug environment (IDE) contains a connection to a single ISS representing a single CPU with a simple memory system. Execution within the ISS occurs at instruction boundaries and interaction with the memory system is via an address map only.

Complex systems can easily have many processing units, made up of CPU, DSP and application-specific cores. For each of these to interact, they must present a SystemC module that has a clock port, a connection to an appropriate bus, and probably some asynchronous input ports like reset and interrupts. In the AMBA framework, a processor model is modeled as a bus master module. The processor model is cycle-synchronous with the rest of the SystemC simulation; for each cycle of the SystemC simulation there would be a corresponding cycle within the processor model. This scheme provides the greatest accuracy and, since the system is modeled at the transaction level, the overall simulation speed is extremely fast:

Our cycle-accurate processor model (CCM) exposes two important interfaces to the designer which are used to interface the CCM to a cycle-based simulation environment: a bus-transaction interface, and a remote debug interface (RDI).

A simple callable interface provides pipeline-accurate AHB transactions which can easily be translated into bus transactions at the cycle-level. The advantage of using such a generic interface is that the same processor model can be used in a variety of abstractions.

The speed of the interface is directly proportional to the level of detail extracted from the transactions provided by the processor model. Should the designer wish to re-use the model in a signal-level interface, this is still possible using the same model.

Source: Class 201 - Capturing design intent and evaluating performance w/SystemC

See related chart

Need full-chip verification?