Transaction level modeling of SoC with SystemC 2.0

Sudeep Pasricha
Design Flow and Reuse/CR&D
STMicroelectronics Ltd
Plot No. 2 & 3, Sector 16A
Noida – 201301 (U.P) India

Abstract

System architects working on SoC design have traditionally been hampered by the lack of a cohesive methodology for architecture evaluation and co-verification of hardware and software. These activities are crucial and must be addressed at an early stage to prevent costly redesign effort later in the design cycle which can adversely affect time-to-market. SystemC 2.0 facilitates the development of Transaction Level Models (TLMs) which are models of the hardware system components at a high level of abstraction. System architects can quickly develop these models and be ready with an executable specification of the hardware blocks as soon as the initial functional specifications of the system are decided. The high speed of simulation of these TLMs allows early development and verification of hardware dependent application software. Timing details can be incorporated into these models to allow performance estimation and architecture exploration. The modular nature of SystemC also promotes reuse of developed components from one system to another. This paper elaborates on the concepts mentioned above and introduces an example SoC TLM platform.

1. Introduction

A system-on-a-chip comprises of many components such as processors, timers, interrupt controllers, busses, memories and embedded software. It is a complete system, which would have been assembled on a board a few years back, but can now be fit entirely in a single circuit because of advances in semiconductor technology. The traditional RTL to layout design and verification flow proves inadequate for these multi million gate systems which have the added complexity of embedded software running on them to cope with. At STMicroelectronics, we are moving towards extending this flow by concentrating our design and verification efforts before the RTL to layout flow comes into the picture. We call this the System-to-RTL flow.

Systems can be modeled at various levels o abstraction. While the terminology may differ slightly from paper to paper, the distinguishing concepts remain the same. In this paper, the micro architecture level of abstraction refers to a cycle accurate models that include complete pin and signal descriptions that verify cycle and system behavior at a very low level. The architecture level of abstraction is less detailed but these models are still implementation dependent. It is useful for software developers who can use the instruction set of the processor that is made available at this abstraction to run and debug their code. Finally there is the functional level of abstraction, which captures the functional behavior of the system, without much concern for implementation details. These models are generally architecture independent.

Our extended flow for system-on-a-chip introduces the concept of Transaction Level Modeling. These models are not as detailed, nor are they concerned with the micro-architecture like the RTL models. Rather, they correspond to the architecture level of abstraction. This is a natural extension of the high-level design process since SoC designs are actually conceived at the transaction level. System architects do not start out thinking about relationships between pins and address busses. Rather they start out by mapping out data flow details - the type of data that flows and where it is stored.

2. What is a Transaction?

In the SoC world, the term transaction has several meanings. In our context, the term refers to the exchange of a data or an event between two components of a modeled and simulated system. Here we are not interested in the protocol that realizes this exchange, as we are not verifying the micro-architecture. A data transaction can be a single word, a series of words or a complex data structure that is transferred over a bus between system components. For example, a DMA master can request to read data from a memory. To do so, it issues a read transaction specifying the address in the memory to read data from. Another case could be a write transaction issued by the embedded software when it wants to write to the registers of the DMA controller. An event transaction models synchronization aspects that ensure correct operation of the SoC model. Interrupts between components can be considered to be an example of an event transaction.

3. TLMs for eSW development

One of the major areas of interest for Transaction Level Modeling is embedded software (eSW). Since most SoCs contain at least one programmable processor, software is an essential part of a SoC. TLM models ease the development of eSW by enabling high-speed simulation of quickly developed models early in the SoC development lifecycle. The speeds required for this purpose vary around 1/1000th to 1/100th of real simulation time of the final product. This means a simulation speed of at least 100k bus transactions per second, which is possible with TLM models but not with the detailed RTL models which tend to be naturally much slower. These TLM models can be built as soon as the architectural specification is available, and even before the time consuming RTL code development commences. This means that eSW development, which is a very lengthy activity, takes place in parallel with the RTL development and not after it. Tasks closely related to the hardware implementation such as low level software development will still have to wait for the RTL model to be completed, but there is still a considerable saving of time which can cut off several valuable months from the development cycle. For instance, the MPEG4 IVT team in STMicroelectronics used TLM models for eSW development 6 months before the top-level netlist was made available.

4. TLMs for architecture exploration

Untimed TLM models, which include the correct ordering of events with no notions of physical time or duration, provide the first level of analysis which is useful for eSW developers. System architects are more interested in timed TLM models, which they can utilize for architecture exploration. One can argue that cycle accurate models in RTL provide a more precise basis for analysis. But this is only partially true. These cycle accurate models require many times the effort that goes into the development of TLM models. The detailed models are also much more difficult to change than TLM models when, for example, HW/SW tradeoffs are being explored. Using TLM models for the purpose of architecture exploration is still being studied. Precision issues essential to issue modeling guidelines for developers of high level TLM models targeted at architecture exploration need to be further understood before being accepted by system architects. In an experiment done by the System Architecture group (CR&D) STMicroelectronics [1], a complex dual processor SoC platform at the TLM and RTL levels was compared and it was found that the TLM model had less than a 15% error margin for most figures (such as interrupt latencies and bus utilization) against transactions observed in RTL SoC simulation. This is an encouraging result that is already being used as the basis for new and additional comparisons using other SoC models. The aim is to gain the confidence of RTL architects and designers by showing that decisions made at the timed TLM level are also valid at the cycle accurate reference RTL platform level.

5. SoC lifecycle and consistency issues

According to the approach outlined above, the SoC lifecycle will require at least three models - one for each of the three levels of abstraction. Since the functionality of the SoC is independent of the architecture, its functional model can be started at an early stage of product specification. Once the SoC architecture specification is made available, work on RTL code development and the SoC TLM model starts. The TLM model is built quickly with a much shorter development time than the detailed RTL model. This means that eSW development and architectural exploration can begin almost as soon as the first architecture specification is released. While the software and architecture teams are working on the SoC TLM model, the RTL development takes place culminating in a SoC RTL platform. At this stage, hardware implementation dependent tasks like low level software development and validation can begin. These tasks are conducted concurrently with the synthesis and back end implementation using the standard ASIC design flow. By the time the first hardware emulator board is available, the eSW has been developed and validated thoroughly so that chances of first time silicon success are high. One problem that would have to be addressed in this flow is that of maintaining consistency between the three views of the same system - functional, architectural and micro-architectural. This issue can be addressed by reusing the same system test vectors across all views, therefore ensuring conformance to expected functionality.

6. SystemC 2.0

We have used SystemC 2.0 for our Transaction Level Modeling effort. SystemC is a C++ library aimed specifically at system level modeling. It has all the benefits that C++ possesses - it is an object oriented design language that makes full use of data encapsulation and generic programming concepts. SystemC 2.0 defines primary channels for communicating transactions but leaves it to the user to define higher-level SystemC channels suited to their design needs.

Communication in the TLM platform is ensured by using a primitive channel, while the synchronization is based on events. We have developed our own channel, as proposed in [3]. Our channel is an example of a communication channel that suits the needs of fast simulation for eSW development. The necessary building blocks for process synchronization and communication refinement are (user-defined) interfaces, ports, and channels. An interface defines a set of methods, but does not implement these methods. It is a pure virtual object without any data in order not to anticipate implementation details. A channel implements one or more interfaces. A port enables a module, and hence its processes, to access a channel interface. A port is defined in terms of an interface type, which means that the port can be used only with channels that implement that interface type. The use of interfaces enables a scheme called interface-method-call [3] IMC refers to a process calling an interface method of a channel. The interface method is implemented in the channel, but it is executed in the context of the caller (the process). An example of an interface method is a blocking read method of a FIFO. When calling this interface method, the caller (process) can be suspended if there is not enough data available.

The channel we have developed has the following features:

- ❑ Master/slave oriented transactions: a master initiates a transaction (a read or write operation) to be served by a slave
- ❑ Multi master / multi slave support: An arbitrary high number of masters is supported, with good scalability of performances
- ❑ Registration facilities: Masters can register and get information about the slaves of the platform for specialized exchanges
- ❑ Synchronization ensured by events: this avoids implementing an ad hoc scheduling policy, and offers a scalable platform
- ❑ Decoding is done on the slave side

7. EASY platform: an example SoC TLM platform

We now come to an example SoC TLM platform developed by the System Architecture group (CR&D) in STMicroelectronics. This TLM platform is a subset of ARM Ltd. Micropack Easy SoC. It has been written in C++ using SystemC 2.0 for system level transaction handling.

Based on an analysis for system design needs in ST, we have developed a higher level SystemC communication channel that offers high level (e.g. read and write) primitives to IP TLM modeling engineers. The source code of IP blocks of the Easy TLM platform has been used as an example from which other platforms have been derived. The only tools required to develop such TLM IPs are the free-of-charge open source SystemC 2.0 kernel and GNU compiler & debugger.

The platform is composed of:

- ❑ a timer with two counters (compliant with the EASY functional specification)
- ❑ an interrupt controller (compliant with the EASY functional specification)
- ❑ a memory
- ❑ a traffic generator, intended to run a compute

function and handle interrupts

The generator loads a value into the timer (counter 1 and 2). When the timer reaches 0, it raises an interrupt to the interrupt controller. The interrupt controller manages the interrupt and propagates it to the generator. The latter suspends the execution of the computing function, handles the interrupt, clears the interrupt source, and resumes its execution (see Figure 1).

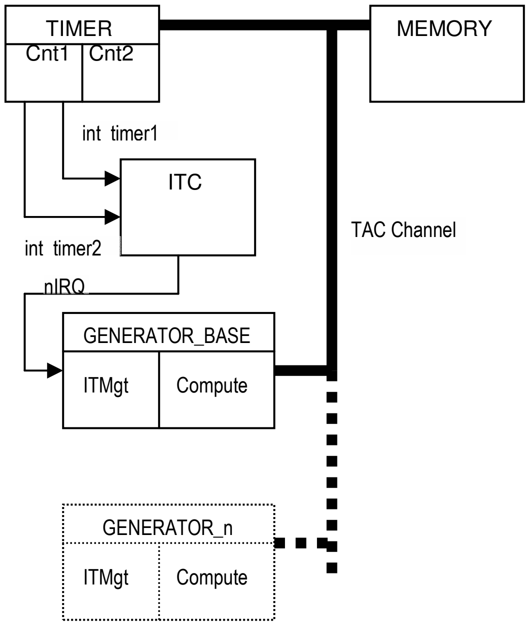

Figure 1: EASY platform with generator

The channel is instantiated as follows:

```
tac_channel<DATA_TYPE>
tac_channel_inst("TAC_CHANNEL");
```

Each block is modeled as a SystemC module, instantiated in the main function, and connected to the channel:

```
MEMORY = new
memory("MEMORY",MEMSPACESIZE,MEMBASE);
MEMORY->slave_port(tac_channel_inst);
TIMER = new timer
("TIMER",TIMERSPACESIZE,TIMERBASE);
TIMER->slave_port(tac_channel_inst);
TIMER->int_timer1(int_timer1);
TIMER->int_timer2(int_timer2);
ITC = new itc
("ITC",INTSPACESIZE,INTBASE);
ITC->slave_port(tac_channel_inst);
ITC->int_timer1(int_timer1);
ITC->int_timer2(int_timer2);
ITC->nIRQ(nIRQ);
GENE = new
generator("GENERATOR",0,MEMBASE,true);
GENE->nIRQ(nIRQ);
GENE->master_port(tac_channel_inst);
```

Below is a couple of examples dealing with the write and read primitives, extracted from the traffic generator code.

- Timer configuration (Write operations):

```
master_port.write(TIMER1LOAD,701);
master_port.write(TIMER1CONTROL,TIMER_
ENABLED |
TIMER_PERIODIC_MODE |
TIMER_PRESCALE_16);
master_port.write(TIMER2LOAD,500);
master_port.write(TIMER2CONTROL,TIMER_
ENABLED |
TIMER_PERIODIC_MODE |
TIMER_PRESCALE_256);
```

- Read a block of data from memory:

```
master_port.read(addr,verif_mem,
BLOCK_SIZE);
```

- Dealing with interrupts (another thread is managing the interruption):

```
if ( IRQ_Handled ) {
wait(IRQ_End);
}
```

In the version of the platform described above, the generator module contains instructions that manipulate the components in the platform. In a subsequent version, the generator was replaced by eSW running on an ARM ISS (Figure 2). In that case it was the eSW that manipulated the platform. Here we use the generator for simplicity to demonstrate the transactions taking place in the system. Note that the generator can replicate the functionality of the eSW running on the ISS and it presents a similar interface to the rest of the system. Hence it can be used instead of the eSW and ISS to validate and examine the rest of the system components.

For our EASY platform, simulation speed with all transfers being single-word, plus interrupts processing, is 120 to 170 K bus transactions per second, depending on the platform version. Simulation speed with some transactions being blocks rather than single-word transfers, shows speeds of several million bus transactions per second (platform version with ISS as master would limit that speed). This benchmark corresponds to the measures made on a SUN Ultra 10 work station, running at 450 Mhz, with

256 MB RAM.

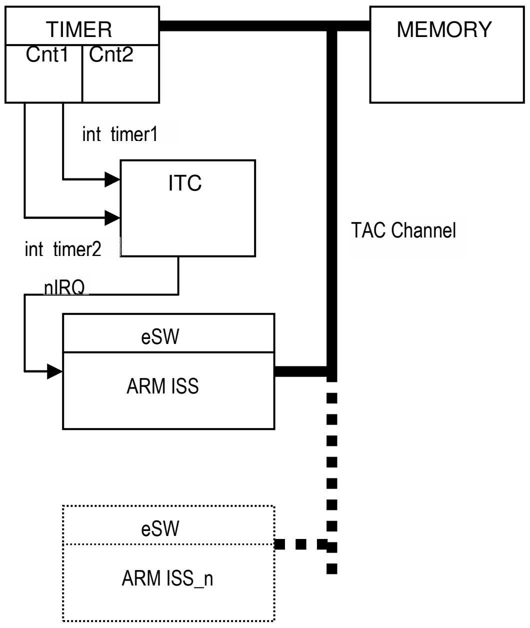

Figure 2: EASY platform with ISS/eSW

8. Design Reuse

To reduce cost and development time, reuse of designed components is a must. Traditionally, reuse of components close to the final implementation has proved effective. However it is not always desirable to reuse components at this level since slight variations in specification can result in different implementations and a lot of remodeling effort. However, moving higher in abstraction can eliminate the differences among designs, so that the higher level of abstraction can be shared and only a minimal amount of work needs to be carried out to achieve final implementation. This is the first step towards building a library of hardware and software implementations at a high level, which will tend to be stable across platforms. Of course it is also important to have a multilevel library, including the lower level abstractions close to the physical implementation that change with advances in technology. But the importance of reuse at a higher level (system to RTL flow) should not be ignored. Many system designers have yet to embrace the idea of a reusable high-level system library. They have to realize that design reuse in every shape and form will be necessary to cope with increasingly complex embedded systems that have become a reality now.

9. Conclusion and future work

System architects and embedded software developers are accepting transaction level modeling into their design flow because it addresses their need for early architecture exploration and eSW development. SystemC 2.0 lends itself to TLM modeling and is thus increasingly becoming the language to propagate the TLM paradigm. However work still needs to be done to formalize the methodology for architecture exploration and for adopting a common set of modeling guidelines to promote interoperability. It is forecasted that in the next few years, most of the content of SoCs will be pre-designed. This will occur along with a move to platforms in which many elements of an architecture are predetermined. The modular approach used by SystemC will allow libraries of system components to be developed and reused for different platforms, thus reducing time-to-market without compromising on SoC quality.

10. Acknowledgements

I would like to thank Srikant Reddy Modugula for his valuable advice and support throughout this work. Thanks to Alain Clouard and Frank Ghenassia for their cooperation in the area of TLM modeling. Special thanks to Sudhanshu Chadha, Rohit Jindal, Digvijay Singh, Kshitiz Jain and Suraj Bharech for their feedback.

11. References

[1] A. Clouard, G. Mastrorocco, F. Carbognani, A. Perrin, F, Ghenassia. *“Towards Bridging the Precision Gap between SoC Transactional and Cycle Accurate Levels”,* DATE 2002

[2] A. Ferrari and A. Sangiovanni-Vincentelli, System Design. *“Traditional Concepts and New Paradigms”.* Proceedings of the 1999 Int. Conf. On Comp. Des, Oct 1999, Austin

[3] *“Functional Specification for SystemC 2.0”,* Version 2.0-P, 0ct 2001

[4] Frank Vahid, Tony Givargis. *“Embedded System Design: A Unified Hardware /Software Introduction”.* John Wiley & Sons, Inc