Methodology for Rapid Development of Loosely Timed and Approximately Timed TLM Peripherals

Mukundan Kadambi Narasimhachar, MindTree Ltd
Bangalore India

Abstract :

This paper proposes a methodology to develop SystemC TLM 2.0 peripheral models and a technique to incorporate Loosely Timed (LT) and Approximately Timed (AT) modes in them. This methodology uses simple_target_socket to model slave interface, since a single transport function can be used for both blocking and non-blocking transport call. An input Boolean port controls the addition of timing logic in AT mode of operation.

INTRODUCTION

With increase in System on Chip (SoC) complexity, virtual prototyping is gaining popularity as a method to architecture a system, as an executable specification across teams and as surrogate hardware for early software development. For this, high performance IP models at a higher abstraction level than Register Transfer Logic (RTL) are used to develop the virtual prototype. Open SystemC Initiative (OSCI) has provided the SystemC community with the TLM 2.0 standard to increase interoperability of models from different vendors.

OSCI TLM 2.0 documents propose two model coding styles: Loosely-Timed (LT) models and Approximately-Timed (AT) models. The LT models are used for early software development including complete Operating System (OS) and applications, hence their simulation performance needs to be high. For this, these models have lesser timing details and use the temporal decoupling technique. Temporal decoupling allows initiators to run ahead in a local time warp and perform multiple transactions every time the SystemC kernel passes control to them. The temporally decoupled initiators sync with the rest of the system at identified configurable timing boundaries. For the masters to complete multiple transactions on every run the target blocks must execute their target functions without using wait calls. In LT modeling style the transaction order and resource contention for different initiators will not reflect the SoC’s hardware behavior. But this is an acceptable tradeoff for the simulation speeds that will be attained, provided the functionality of the SoC for software development is not compromised. The AT models on the other hand will be used for performance analysis and architecture exploration. For this, its timing behavior and resource contentions must be similar to the actual hardware that is modeled. Here, initiators have to move in sync with other initiators in the SoC at the transaction boundaries.

MOTIVATION

The OSCI TLM 2.0 documents recommend developers to first create LT models and later add timing details to use them as AT models. In the methodology proposed in this paper, delays and resource contention that will be encountered when multiple initiators access a single target block will be modeled. This methodology was used to develop System Packet Interface Level 4 (SPI-4) and PCI Express (PCIe) SystemC TLM models for virtual prototyping applications.

METHODOLOGY

For SPI-4, the complete functionality was modeled as a C++ class and instantiated in a SystemC module wrapper that takes care of communication with the external world. For PCIe, the Data Link Layer and Transaction Layer require timing information as they have to timeout for certain operations. Hence, these blocks are modeled as SystemC modules. The data packets flowing through them internally are purely functional and do not carry timing information. The PCIe functional blocks are integrated in a SystemC module and instantiated in a wrapper that handles the communication with the external world. Delays to receive and transmit the packets are modeled in the communication wrappers for both the modules.

The block diagram of the SPI-4 SystemC TLM model is shown in Figure 1. Here, TLM 2.0 target socket is used to model the configuration slave interface that will be a slave on the memory mapped bus. The TLM 2.0 initiator and target sockets are also used to model the interfaces to accept packets from rest of the system and to model the interface that performs the SPI Transmit/Receive operation. Though these interfaces are not part of the memory mapped bus, the OSCI TLM 2.0 provided sockets can be used to model them. This decreases the development time and simplifies data packet exchange between memory-mapped and non-memory mapped interfaces. The target transport functions associated with these sockets compute delays for the transaction, based on transaction length and the interface’s data-width and clock period. The data-width and clock period information for each of the interfaces are configurable parameters and length of the transaction is obtained from the generic payload. The slave interfaces are modeled using the convenience socket simple_target_socket, thus making it possible to use a single transport function for both blocking and non-blocking transport calls. The transport function is registered as the blocking transport function with the simple_target_socket since SystemC wait would be called in AT mode. To control the mode of operation of the model, a Boolean input port is provided. The value of this input port is used in the transport function to control the path of execution in the current mode.

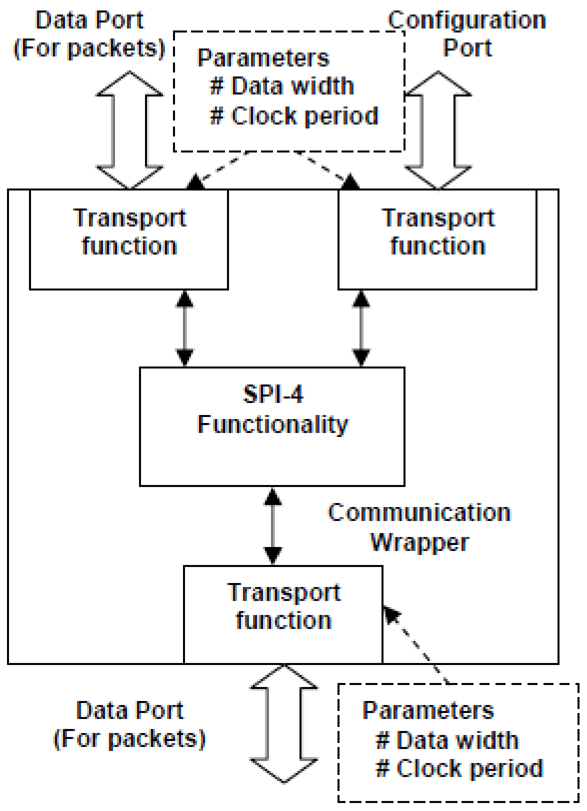

Figure 1 SystemC TLM Model of SPI-4

In the case of LT mode, the computed delay is added to the annotated time received by the transport function of the target. This function executes without any wait statements. This allows temporally decoupled initiators to complete multiple transfers on every simulation run given to them. If multiple initiators are accessing a slave in this manner, the slave increments the annotated time by a delay equal to the time consumed for current access. It does not take into account that another master could be accessing the target in the same time duration. Figure 2 shows the simulation flow for the LT mode for multiple masters accessing a single slave. The path from initiator to slave shows the transaction length and the annotated time. Path from target to initiator is the return from the transport function call with incremented annotated delay. The transaction length, data width of target interface and clock period are used to compute delay of current transaction. Since, only the annotated delay of the accessing initiator is considered, the simulation time at the end of the four transactions for both initiators is 900 ns. In case of AT models, the computed delay will be added to the annotated time input and wait for the total time will be performed before returning from the transport function. Also, additional logic is added to take into account that a previous transaction of the initiator may be in progress. Hence, the target socket timing behavior will be closer to the behavior of the target interface in the real system. This makes it a good candidate for use in architecture exploration and system performance analysis. The addition of extra logic for AT mode is controlled by the Boolean value of the input port as mentioned earlier. Figure 3 provides the flowchart for this logic. Figure 4 shows the simulation flow in AT mode for a target accessed by multiple initiators. The forward calls from initiator to target contain transaction length and annotated delay. The target adds annotated delay, the delay for previous transaction that is in progress if applicable and the current transaction delay. The target transport function performs wait for this total duration before the call is returned to the initiator.

CONCLUSION

Our proposed methodology is to use the TLM sockets for non-memory mapped interfaces. The use of simple_target_socket for modeling slave interfaces allows a single transport function to be utilized for both AT and LT models. A control can be used to select between AT and LT modes. Inter-initiator timing related logic can be enabled only during AT mode of operation. These steps will decrease the development time required for many of the IP peripheral blocks. The above approach was used in the development of SPI-4 and PCIe blocks and the time required to convert the LT models to AT models was well within required time limit.

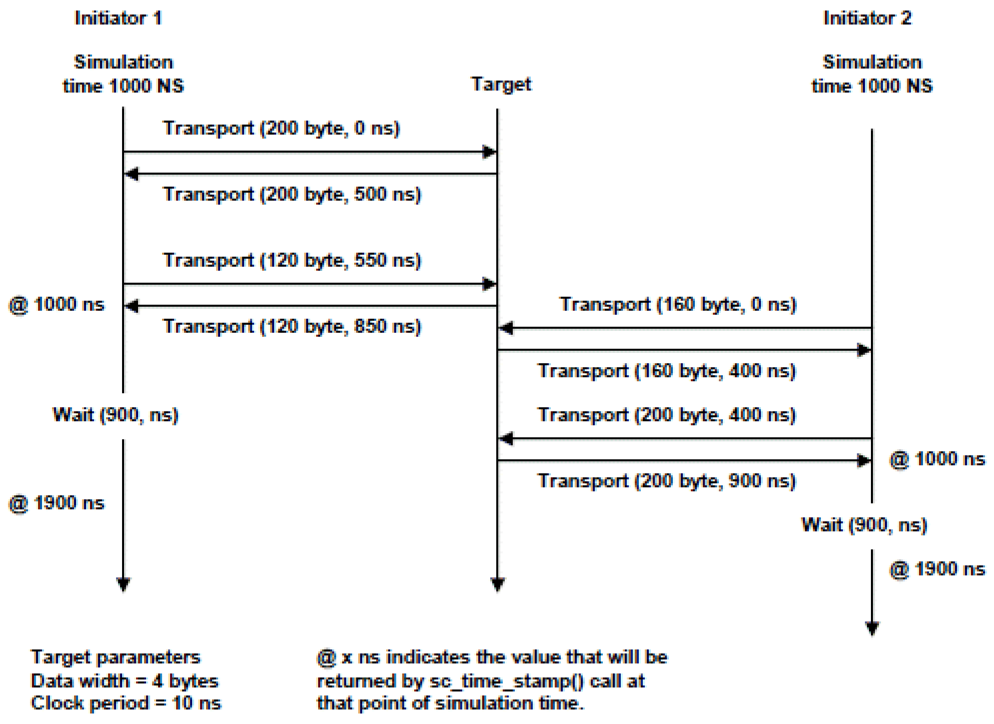
Initiator 1
Simulation time 1000 NS
Target
Initiator 2
Simulation time 1000 NS
Transport (200 byte, 0 ns)
Transport (200 byte, 500 ns)
Transport (120 byte, 550 ns)
@ 1000 ns
Transport (120 byte, 850 ns)
Transport (160 byte, 0 ns)
Transport (160 byte, 400 ns)
Wait (900, ns)
Transport (200 byte, 400 ns)
@ 1000 ns
Transport (200 byte, 900 ns)
@ 1900 ns
Wait (900, ns)
@ 1900 ns
Target parameters
Data width = 4 bytes
Clock period = 10 ns
@ x ns indicates the value that will be returned by sc_time_stamp() call at that point of simulation time.

Figure 2 Simulation Flow in LT mode

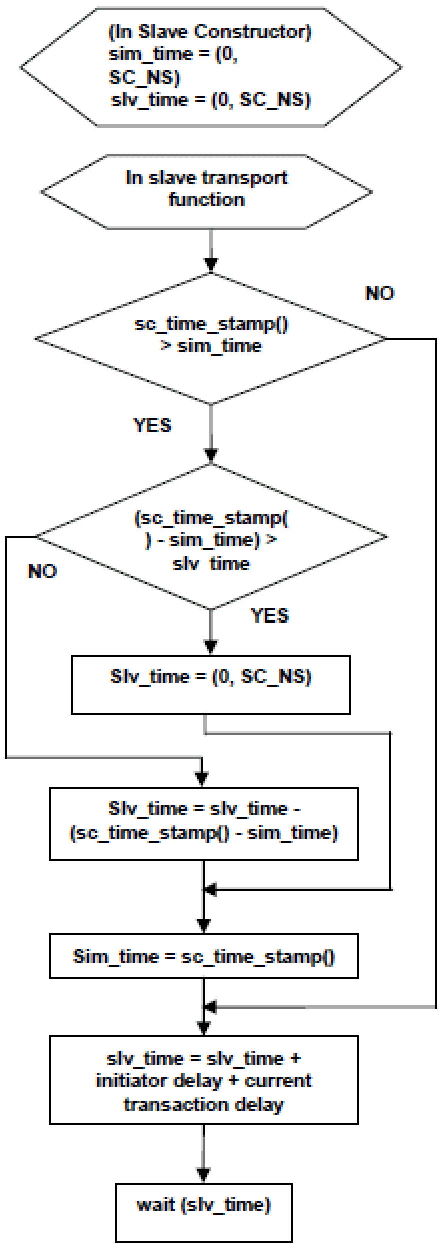
(In Slave Constructor)
sim_time = (0, SC_NS)
slv_time = (0, SC_NS)
In slave transport function
sc_time_stamp() > sim_time
NO
YES
(sc_time_stamp( ) - sim_time) > slv time
NO
YES
Slv_time = (0, SC_NS)
Slv_time = slv_time - (sc_time_stamp() - sim_time)
Sim_time = sc_time_stamp()
slv_time = slv_time + initiator delay + current transaction delay
wait (slv_time)

Figure 3 AT mode timing logic Flowchart

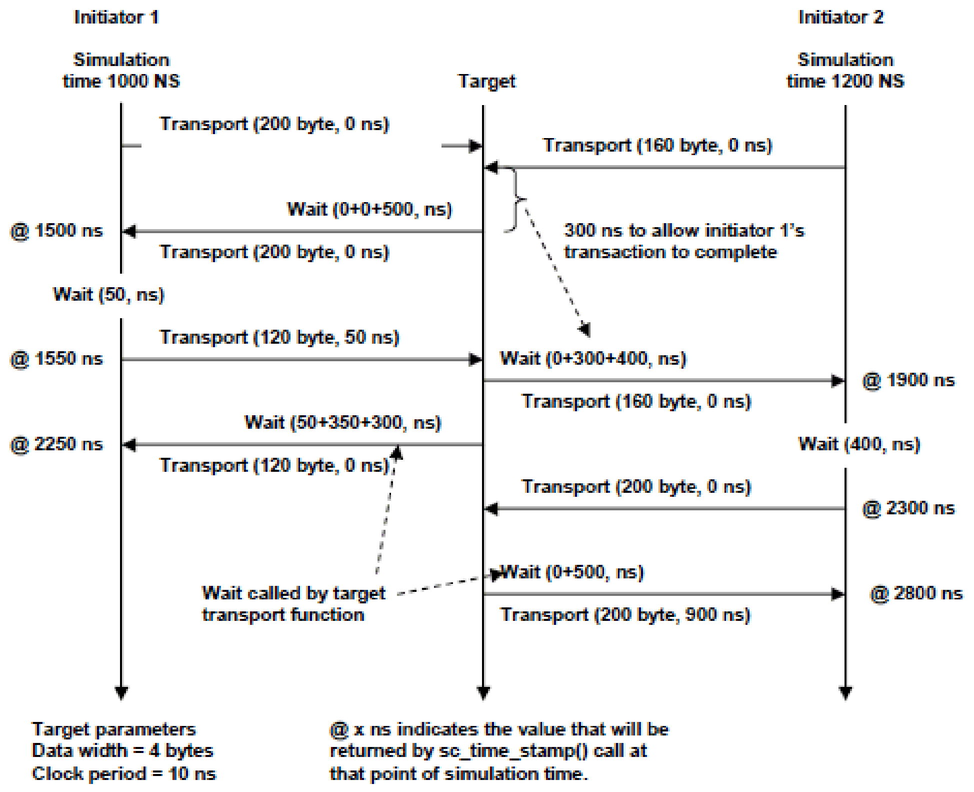
Initiator 1
Simulation time 1000 NS
Target
Initiator 2
Simulation time 1200 NS
Transport (200 byte, 0 ns)
Transport (160 byte, 0 ns)
Wait (0+0+500, ns)
@ 1500 ns
Transport (200 byte, 0 ns)
300 ns to allow initiator 1's transaction to complete
Wait (50, ns)
Transport (120 byte, 50 ns)
@ 1550 ns
Wait (0+300+400, ns)
@ 1900 ns
Transport (160 byte, 0 ns)
Wait (50+350+300, ns)
@ 2250 ns
Wait (400, ns)
Transport (120 byte, 0 ns)
Transport (200 byte, 0 ns)
@ 2300 ns
Wait (0+500, ns)
Wait called by target transport function
@ 2800 ns
Transport (200 byte, 900 ns)
Target parameters
Data width = 4 bytes
Clock period = 10 ns
@ x ns indicates the value that will be returned by sc_time_stamp() call at that point of simulation time.

Figure 4 Simulation Flow in AT mode