Host Based System C-TLM2.0 Simulation Model for Memory Device Driver Verification

A. Pallavi, C. Kanagasabapathi, Siva S Yellampalli

Abstract*: With increasing complexity in SoC architecture in today's embedded systems demand for more software applications is increasing. Time to market plays an important role for any product to win the market. Device drivers and stability pre-silicon (prior to silicon arrival) plays a very important role. To develop, integrate, system test and debug device drivers like Memory, I2C, PCIE, USB etc. with real world use cases, availability of hardware board is necessity. Availability of hardware board traditionally happens towards the later stages of the project life cycle. The best solution is using SystemC-TLM2.0 simulation models to verify device drivers early. This ensures the readiness of device drivers even before the hardware availability and effectively reduces time to market (TTM). Aim of this thesis is to develop Host based SystemC-TLM2.0 simulation model for Memory device driver verification.

***Index Terms*: System C, TLM2.0, TTM, Memory device drivers**

I. INTRODUCTION

The scope of the paper is to develop interconnect bus, memory model using SystemC-TLM2.0, API(Application Peripheral Interface) functions for host based the CPU, C based memory driver tests and perform the validation by running driver tests. Interconnect bus design is modeled efficiently which can be ported with minor changes to reuse to perform the design with different master slave numbers. Scope of the thesis limits to only Memory drivers, additional drivers like I2C, PCIE, USB etc. can be verified by developing respective hardware models in SystemC-TLM2.0 and plugging into one of the interconnect bus slaves.

II. DESIGN OF SYSTEMC-TLM2.0 SIMULATION MODEL AND DEVICE DRIVER

A. Design and verification procedure

Design and Verification procedure flow is as shown in the Figure.1. and steps are mentioned below:

1) Design and unit level verify the Memory model using SystemC-TLM2.0. The same model is instantiated twice for Memory0 and Memory1 instance.
2) Design the interconnect bus model and Verify the design with unit level verification using SystemC TLM2.0.
3) Design the Host based CPU APIs for Memory device driver interactions using C/C++.
4) Integrate Host Based CPU, Memory and Interconnect model in SystemC and generate an executable.
5) Develop the Memory driver infrastructure, API supporting common functions.
6) Develop C based memory access tests and ISR related test cases using APIs.
7) Verify the design and capture the simulation results

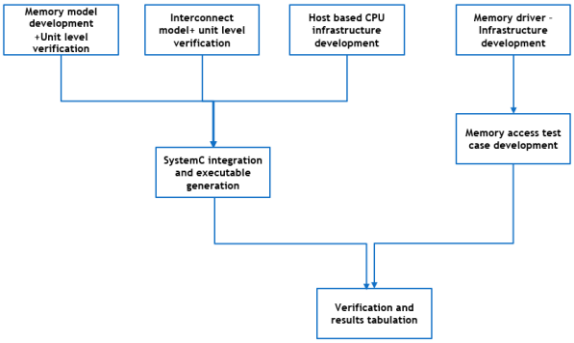

Figure 1: Flow chart of design and verification procedure

Design and flow and verification flow can be planned in parallel to speed up the procedure to get the faster results. For example, Memory driver development can go in parallel while the simulation model development is ongoing.

B. Block diagram of the design

Figure.2. shows the block diagram of SystemC-TLM2.0 simulation model for Memory device driver verification. Simulation model mimics the hardware components in SystemC-TLM2.0 and Memory device driver - test code shows the C based test code abstracted at software HAL (Hardware Abstraction) layer.

Below is the brief description of each component of the block diagram:

• **Host based CPU**: It implements the Application Program Interface (API) functions to interface Memory device driver calls to the SystemC-TLM2.0 simulation models.

• **Interconnect**: It is an interconnect bus model developed using SystemC-TLM2.0. It decodes the incoming transaction address and sends the transaction to Memory1 or Memory2. It also generates bus error for the invalid transactions.

• **Memory:** Memory model is an abstracted SystemC-TLM2.0 model. Supports normal memory read/write operations through b_transport interface and direct memory access through direct memory interface (DMI).

• **Memory Device Driver** - Test Code: It is a C based test code infrastructure having memory access test cases and interrupt test cases.

Integration of all the blocks is done using

Revised Manuscript Received on July 22, 2019.

A. Pallavi, VLSI design and Embedded Systems, UTL Technologies Ltd, Bangalore, Karnataka, India.
C. Kanagasabhapati, VLSI design and Embedded Systems, UTL Technologies Ltd, Bangalore, Karnataka, India.
Dr. Siva S Yellampalli, VLSI design and Embedded Systems, UTL Technologies Ltd, Bangalore, Karnataka, India.

SystemC-TLM2.0. There are two different levels of integrations. One is for the unit level verification and another is for the Memory device driver verification. Executable is generated to perform the verification.

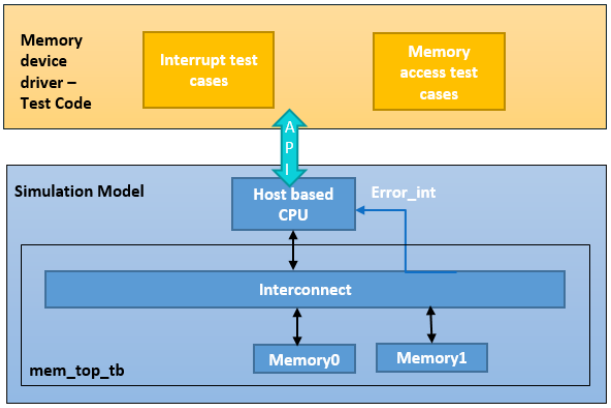

Figure 2: Block diagram of SystemC-TLM2.0 simulation model

C. Integration and executable generation

Integration is the process of binding sub components and making it into one system. Procedure involved in integration of design blocks of simulation model for unit level verification and for Memory device driver verification is explained below.

A. Integration for Unit level verification

Unit level verification is sub system level verification performed using SystemC-TLM2.0 test bench. This requires the development SystemC-TLM2.0 mem_top test bench (mem_tob_tb). Interfaces of mem_tob_tb will be same as the mem_top module but with the ports polarity reversed.

B. Integration for Memory device driver verification

Memory device driver verification is C based verification. The required models for integration are mem_top, host based cpu model. This integrated block is called the simulation model (sim_top). Memory device driver test cases are written using C. Test cases and test case description remains same for the unit level verification or Memory device driver verification. Generic flow for Memory device driver test case development is explained below

Register an ISR:

- write an isr_handler function to handle the error_int interrupt : void isr_handler(){ };
- register an ISR using register_isr API: register_isr (unsigned int_number, *ISR function pointer*)

Write transaction:

- Define the target write address (addr) and write value (data)
- Use API for write transaction write_port (unsigned int addr , unsigned int data)

Read transaction:

- Define the target read address (addr) and read value (data)
- Use API for read_port (unsigned int addr, unsigned int &data)
- Check for the received data, if read data and write data is same test is reported as Pass else test is reported as Fail.

C. Executable generation

The steps involved for executable generation unit level verification and Memory device driver verification are explained below.

Step1: In case of Memory device driver verification - Open the host_based_cpu solution in MSVC. Build the MSVC project solution which will generate executable host_based_cpu-Debug.exe. And in case of unit level verification - Open the mem_top_tb solution in MSVC. Build the msvc project solution which will generate executable mem_top_tb-Debug.exe.

Step2: Build the Memory device driver test code as dll (dynamic linked library) and copy the memorydriver_dll_if.dll to the host_based_cpu solution folder. (Note: device driver test code is developed in C, hence any C compiler can be used to build the solution, in this project GCC compiler has been used). Step2 is skipped in case of Unit level verification.

Note: Linking errors can be caught only during execution, make sure all API definitions are defined in host based CPU model.

Tools overview

Tools used:

- Microsoft Visual Studio 2013 version (installation can be customized only for C/C++)
- Additional libraries: SystemC 2.3 (from open source SystemC initiative web source)
- GTKWave 3.3 - waveform viewer

Languages used:

- SystemC/TLM2.0 - For Simulation model development
- C - For Device driver test code development

III. SIMULATION RESULTS

Aim of the work is to verify Memory device driver test code using the SystemC-TLM2.0 simulation model. Figure. 3. Shows the Console window pop up to select options for test case to run. It provides the provision to either to run individual test case or run the regression (run all test case one by one in a single loop). Make a selection of intended test cases and observe the test results.

A. Results interpretation

- After running each test cases console window prints the test Pass or Fail results.
- SystemC-TLM2.0 also provides the options for signal tracing. We can add the required signals for tracing from the test bench code. After the simulation run is complete, it will generate the VCD file which can be opened using any waveform viewer. In this project GTKWAVE has been used. TABLE I. shows the sc_trace signals and its description used for the test results interpretations.

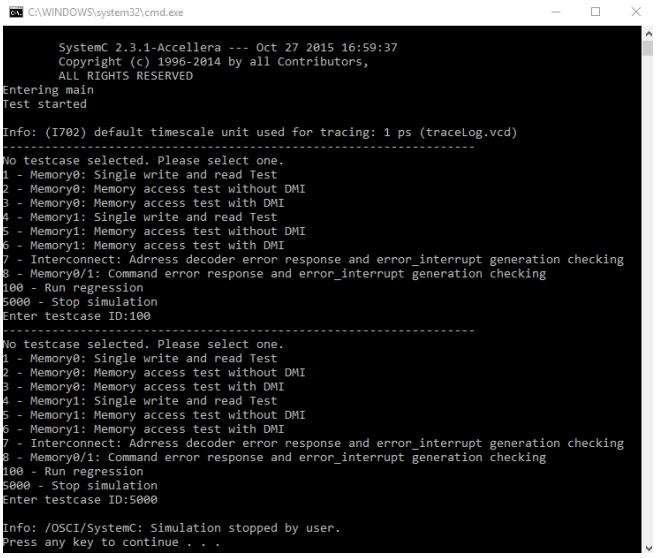

Figure 3: Snapshot of console output window running MSVC solution

Table 1: List of sc_trace signals

Signal name	Type	Description
Clk	sc_time	26MHZ clk
Rst	bool	Active low reset
Address	unsigned int	random 32 bit address
Data	unsigned int	random 32 bit data
Cmd	unsigned int	TLM_READ_COMMAND = 0x0 TLM_WRITE_COMMAND = 0x1 TLM_IGNORE_COMMAND=0x2
response	Int	TLM_OK_RESPONSE = 1, TLM_INCOMPLETE_RESPONSE = 0, TLM_GENERIC_ERROR_RESPONSE = –1, TLM_ADDRESS_ERROR_RESPONSE = –2, TLM_COMMAND_ERROR_RESPONSE = –3, TLM_BURST_ERROR_RESPONSE = –4, TLM_BYTE_ENABLE_ERROR_RESPONSE = –5
error_int	bool	Interrupt generation indication

B. Memory access tests results

➢ Memory0: Single word Write and Read test:

Figure. 4. are the waveform for the single word read and write. As shown in waveform, WRITE is done to address 0x00003F28 with the data 0xABABABAB. On reading with the address 0x00003F28 the read data is same as written data 0xABABABAB.

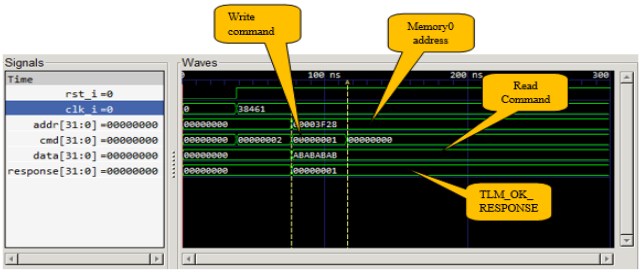

Figure 4: Snapshot of waveform for Single word write and read test

➢ Memory0: Memory access test without DMI:

Figure. 5. is the memory access with the transfer length of 128 bytes. In this test write and read is performed to consecutive aligned address (0x00003f20, 0x00003f24, 0x00003f28 and 0x00003f2c) without DMI request. As expected READ data from the WRITE address is same. Observe in the waveform, there is a new b_transport call initiated from cpu for every aligned address transaction. All b_transport calls will go through all transport interfaces (interconnect bus model). For a transfer of bigger data chunks like 1000MB this behavior will be a simulation bottle neck.

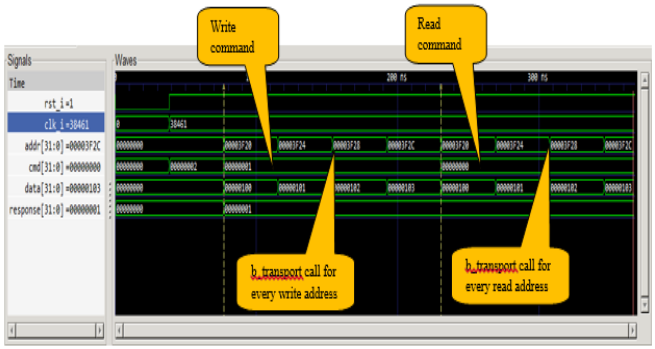

Figure 5: Snapshot of waveform for memory access test without DMI

➢ Memory0: Memory access test with DMI:

Figure. 6. is the memory access with the transfer length of 128 bytes. In this test write and read is performed to consecutive aligned address (0x00003f20, 0x00003f24, 0x00003f28 and 0x00003f2c) with a DMI request. As expected READ data from the WRITE address is same

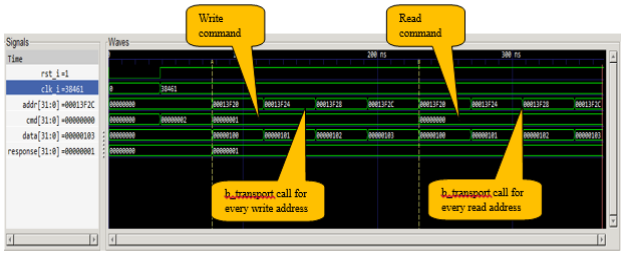

Figure 6: Snapshot of waveform for memory access test with DMI

Observe in the waveform, there only one b_transport call initiated from cpu for all aligned address transaction with DMI request. This b_transport call will go through all transport interfaces (interconnect bus model). As target Memory0 model supports DMI, it acknowledges the request and sends DMI pointer. Subsequent DMI call bypass the interconnect components and does the memcpy with respect to received direct memory pointer.

➢ Interconnect: Default slave read and write

This test is for checking TLM_ADDRESS_ERROR_RESPONSE and error interrupt generation. Figure. 7. shows the waveform of error_int generation for an invalid address. Interconnect bus model checks the incoming address 0x00023F80 which does not belong to Memory0 or Memory1 address ranges and it generates an error interrupt and updates its response status as TLM_ADDRESS_ERROR_ RESPONSE. Handling of interrupt differs for unit level verification and driver level variation.

At unit level SystemC SC_THREAD callback sensitive to error_int handles the interrupt whereas at driver level ISR (Interrupt Service Routine) function callback handles the interrupt. ISR will be registered in the beginning of test initializations.

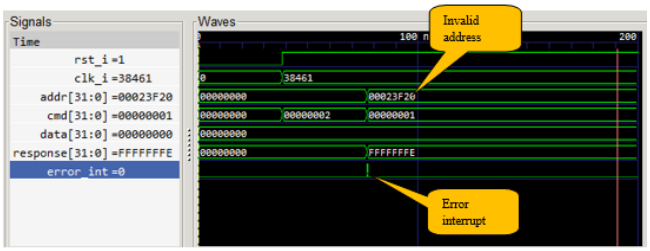

Figure 7: Snapshot of waveform of default slave read write test

- Memory0/1:TLM_COMMAND_ERROR_RESPONE checking

This test is for checking TLM_COMMAND_ERROR_RESPONSE and error interrupt generation. Figure.8 shows the waveform of error_int(error interrupt) generation for an invalid command. Target Memory0/1 model checks the command received TLM_IGNORE_COMMAND which is not either TLM_WRITE_COMMAND or TLM_READ_COMMAND and hence it cancel the transaction by setting its response status as TLM_COMMAND_ERROR_RESPONSE. Interconnect bus model checks received command error response and generates and interrupt. Handling of interrupt differs for unit level verification and driver level variation. At unit level SystemC SC_THREAD callback sensitive to error_int handles the interrupt whereas at driver level ISR (Interrupt Service Routine) function callback handles the interrupt. ISR will be registered in the beginning of test initializations.

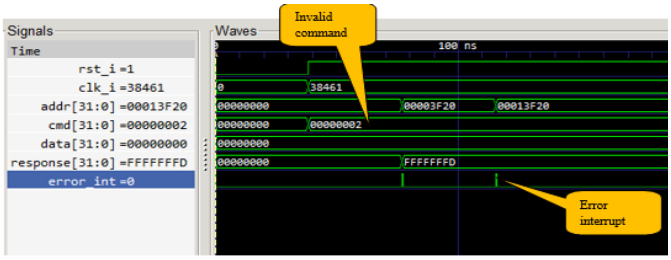

Figure 8: Snapshot of waveform of error_int generation for invalid command test

IV. CONCLUSION

Developed Host based SystemC-TLM2.0 simulation model. Developed unit level test bench and verified the simulation model. Developed test infrastructure and C test cases for Memory device driver. Performed driver verification by running intended test cases and recorded the test results. This project has demonstrated host based simulation model flow and the integration of Memory device driver earlier in the design process. Current thesis just provides the frame work for Host based SystemC-TLM2.0 simulation model for the Memory device driver verification. The frame work can be reused and enhanced for other device drivers like I2C, PCIE, and USB etc. This requires the availability of SystemC-TLM2.0 IP models for I2C, PCIE and USB etc. as per the product specification.

REFERENCES

1. Jon Connell, ARM , Bruce Johnson, Synopsys, Inc , "Early Hardware/Software Integration Using SystemC 2.0" , Class 552, ESC San Francisco 2002
2. Luca Fossati, Politecnico di Milano (Italy): 2009-2010 , European Space Agency/ESTEC: 2010-2012, "Development of the SystemC model of the LEON2/3 Processor", ESA contract 20921/07/NL/JD
3. Claude Helmstetter, Vania Joloboff, "SimSoC: A SystemC TLM integrated ISS for full system simulation", INRIA
4. Sammidi Mounika and Renuka B S, "An Approach to Reduce Time to Market in today's Mobile Market", IEEE Computer Society, IEEE Standard for Standard, SystemC® Language Reference Manual, IEEE Std 1666™-2011
5. M. Caldari*, M. Conti*, M. Coppola**, S. Curaba**, L. Pieralisi*, C. Turchetti*, "Transaction-Level Models for AMBA Bus Architecture Using SystemC 2.0", 1530-1591/03, IEEE 2003
6. David C.Black, Jack Danovan, "SystemC: From the Ground Up"©2004 Kluwer Academic Publishers,Eklectic Ally,Inc

AUTHORS PROFILE

Ms. Pallavi A obtained her M.Tech from VTU Extension Centre, UTL Technologies Ltd, Visvesvaraya Technological University. She has worked on broad range of research topics including pre silicon hardware modelling in SystemC/TLM2.0, Virtual prototyping, System level Validation, SW debugging on Virtual platforms.

Mr. C. Kanagasabapathi obtained his M.Tech from Visvesvaraya Technological University. He is currently with VTU Extension Centre, UTL Technologies Ltd. He worked on a broad range of research topics including Very Large Scale Integration (VLSI), High voltage Instrumentation, EMI & EMC and high performance computing. He has published multiple journal papers & IEEE Conference papers in these areas of research.

Dr. Siva Yellampalli obtained his MS & Ph.D from Louisiana State University. He is currently with VTU Extension Centre, UTL Technologies Ltd. He worked on a broad range of research topics including Very Large Scale Integration (VLSI), mixed signal circuits/systems development, micro electromechanical systems (MEMS), and integrated carbon nanotube based sensors. He has published a book in the area of mixed signal design, and edited two books on carbon nano tubes. He also published multiple journal papers & IEEE Conference papers in these areas of research.