Implementation of a DMAC using SystemC

Young-Jin Oh, Byeong-Deok Kim, Myoung-Keun You, Gi-Yong Song

School of Electrical and Computer Engineering
Chungbuk National University, Cheongju Chungbuk, 361-763, Korea
E-mail : goodmen913@chungbuk.ac.kr, haracan@chungbuk.ac.kr, mkyou77@chungbuk.ac.kr, gysong@chungbuk.ac.kr

Abstract : Recently, SystemC has been stressed in system-level design methodology because of the capability of system architectural model description and co-design hardware and software design. Also SystemC has many features on modeling as well as verification. DMA is an essential feature of modern systems. It improves performance of system and decreases CPU overhead. This paper describes an implementation of a DMAC using SystemC and compares performance in each channel and each system-level. By comparison of performances, we found out a proper abstraction level model for system-level design.

1. Introduction

As the design of a system level gets more complex and larger, the expressing design and verification on a high level becomes important than ever. The disparity between chip complexity, measured by the number of transistors on a silicon chip and design productivity, measured by the number of transistors for designers to handle per day indicates that semiconductor industry will be able to manufacture complex chip beyond our ability to design those chips in any reasonable time to market. The most obvious solution for narrowing the productivity gap is to implement a SoC(system-on-a-chip) by raising the level of abstraction in design [1-2].

The importance of SystemC which models the system on a high abstraction level has recently been stressed in system design. SystemC provides an event-based simulation kernel and C/C++class library for hardware modeling, and co-design environment for software and hardware modules. Also it provides the concepts such as channels, interfaces, ports and events for communication between modules [3-4].

Being built on standard C/C++ language, the SystemC describes functionality and communication at various abstraction and allows modeling range from system level to RTL(Register Transfer Level). The target system and sub-system communication and functionality can be developed and refined independently in un-timed, approximately-timed, or cycle-timed model.

This paper designs a DMAC(direct memory access controller) using SystemC in several abstraction levels and compare performances of DMAC in each level. Sub-modules of DMAC are progressive refined from system architectural model to finite state machine in the viewpoint of functionality, and interconnections between sub-modules are refined from a sc_fifo channel to a bus channel which implements a communication protocol using sc_buffer channels.

2. Preliminary

2.1 SystemC

SystemC [1-6] is a C/C++ class library and a methodology that we can use to effectively create a cycle-accurate model of a given algorithm, hardware architecture, and interface of SoC and system-level design. So system designer can write the design and verify it using the same language, and further refine it all the way to the implementation level.
The SystemC design flow is shown in Fig. 1.

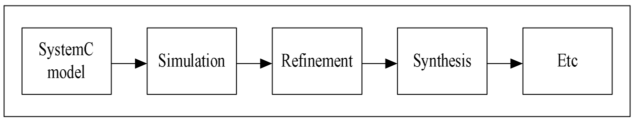


Fig. 1. SytemC design flow

SystemC supports RTL modeling simulation and synthesis, a major reason for using the language is to design at higher abstraction level than RTL, hardware-software co-design, and description of the architecture of complex systems consisting of both hardware, software, and interfaces in a C++ environment. A system designer need only write a model using SystemC.

SystemC provides several advantages that it can understand the design/system specification, create performance models of the system and validate the system performance and refine and reuse testbench at the higher level down to the implantation level. We iteratively refine the executable specification down to the RTL prior to synthesis, which is still in SystemC. The testbench which is written in SystemC is reused to ensure that the iterative refinements of the SystemC model did not introduce errors.

SystemC supports co-design and description of the architecture of complex systems consisting of both hardware and software. It allows a hierarchical structure because a module can have other modules or processes in it. Process communicate with other via interfaces, channels, and ports, and synchronize via events. Also SystemC has event-driven simulation kernel which offers fast simulation.

2.2 DMAC

The processor of a system must communicate with memory and with a wide range of I/O devices, from slow devices to high-speed devices. Although interrupt-driven I/O frees the processor until the device requires service, the processor still responsible for making the data transfer. DMA removes the processor from the path, and thus relieve congestion on the system bus [7]. DMA is an essential feature of all modern systems, as it allows devices to transfer data without subjecting the CPU to a heavy overhead. DMA services are provided by a DMA controller, which is itself a specialized processor whose specialty is transferring data directly to or form I/O devices and memory. Datapath of DMA is shown in Fig.2.

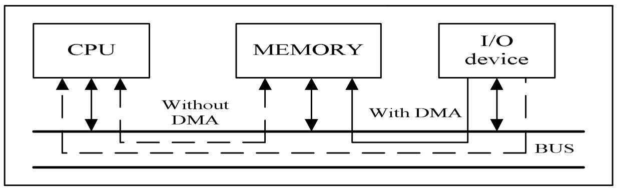


Fig.2. Datapath of DMA

A DMA transfer essentially copies a block of memory from one I/O device to another. DMA can be used to offload expensive memory operations, such as large copies or scatter-gather operations, from the CPU to a dedicated DMA controller [8].

3. Implementation of a DMAC using SystemC

The DMA controller needs a circuit of an interface to communicate with the processor and I/O device. The structure of a DMAC is shown in Fig. 3.

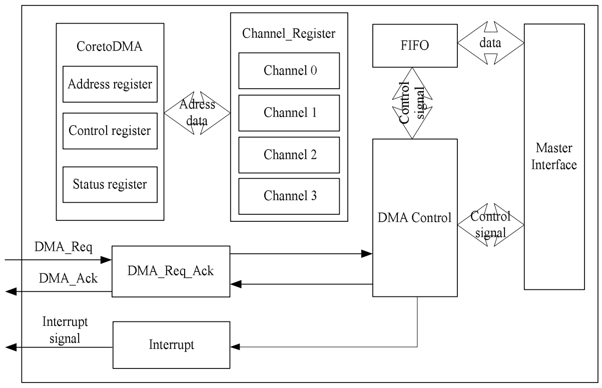


Fig. 3. The structure of a DMAC

DMAC consists of 7 modules: Core_to_DMA module, Channel_Register module, DMA_Req_Ack module, Interrupt module, DMA_Control module, FIFO module, and Master_Interface module. The function of each module is as follows.

Core_to_DMA module : This module stores initial information of data transfer.

Channel_Register module : This module divides channel for DMA operation and temporarily stores changed data by DMA_Control module.

DMA_Req_Ack module : This module performs handling a request of external devices.

Interrupt module : This module is only responsible for delivering interrupt signal from DMAC to CPU.

DMA_Control module : This module controls DMAC

FIFO module : This module stores temporarily data of I/O device or memory

Master_Interface module : This module provides coordination of the communication between I/O device or memory and DMAC

The processes of a DMAC consist of 4 steps.

Step 1: Core_to_DMA module stores the information of data burst transfer.

Step 2: If a request happens from outside to DMA_ReqAck module, this module checks a status of DMA_Control module.

Step 3: If DMA_Control module is active, load the data from memory in source address through an available channel in Channel_Register module. The loaded data is temporarily stored in FIFO module and DMA_Control module stores data in FIFO module into memory in destination address.

Step 4: For next data transfer, DMA_Control module confirms values of Descriptor_register in Channel_Register module. Repeat step3 if positive, otherwise finish DMAC operation.

The DMAC has four independent programmable channels allowing four different contexts for DMA operation. If requests from two channels become active at the same time, channel with the highest priority is serviced first. Each channel has a specific hardware priority DMA channel 0 has the highest priority and channel 3 has the lowest priority.

The DMAC enables the transaction: memory-to-memory, memory-to-peripheral, peripheral-to-memory, and peripheral -to-peripheral. We program the DMAC by writing to the DMA control registers over the Core-to-DMA module. Burst size of the implemented DMAC in this paper can be programmed to make data transmission of 8, 16, or 32-bit.

4. Simulation and Analysis

This paper designs a DMAC from high-level abstraction to low-level abstraction and compares performances of DMAC in each abstraction level. The simulation and verification system roughly consists of three main components as shown in Fig. 4.

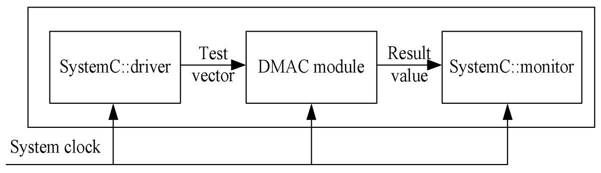


Fig. 4. Simple structure of verification system

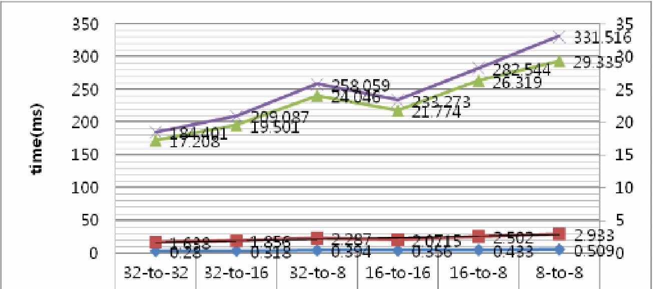
The operation of each module is as follows:

SystemC::driver : This module generates test_vector using random() function which will be sent to DMAC module.

SystemC::DMAC : This module performs data transmission mentioned above.

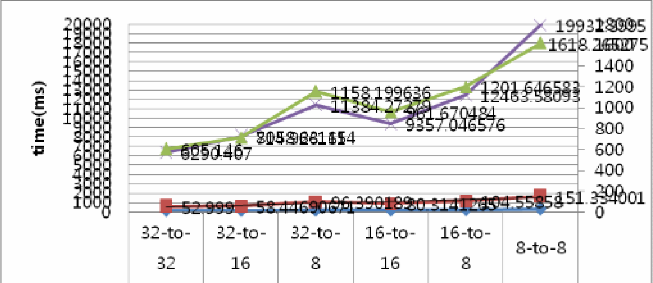
SystemC::monitor : This module receives the results from DMAC module and reports the execution time on SystemC console monitor.

Comparisons of performances on each data burst size in TLM model and in RTL model are shown in Fig. 5 and Fig. 6 respectively.



	32-to-32	32-to-16	32-to-8	16-to-16	16-to-8	8-to-8
10000	184.401	209.087	258.059	233.273	282.544	331.516
10	0.28	0.318	0.394	0.356	0.433	0.509
100	1.638	1.856	2.287	2.0715	2.502	2.933
1000	17.208	19.501	24.046	21.774	26.319	29.335

Fig. 5. Comparison of performance in TLM



	32-to-32	32-to-16	32-to-8	16-to-16	16-to-8	8-to-8
10000	6290.4	8058.6	11384.	9357.0	12463.	19932.
10	10.856	13.918	17.639	16.650	21.984	29.914
100	52.999	58.446	96.390	80.314	104.55	151.33
1000	605.14	714.92	1158.2	961.67	1201.6	1618.2

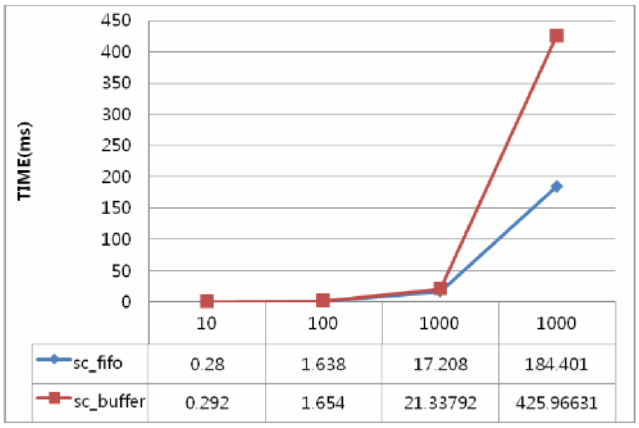
Fig. 6. Comparison of performance in RTL

As shown in Fig. 5 and Fig. 6, execution time of TLM is around 40 times shorter than one of RTL model. The difference of total execution time of a whole DAMC module is made by active process switching time of event-based simulation kernel even though the operation times of only System::DMAC module in each level are almost same.

RTL model needs much operation time for data transfer because it is descriptive of all in/out port of module for operation. TLM model is read/write or transmit/receive operations operating on complete data structures that is being exchanged between the functional units. The TLM level model effectively creates an executable system model that simulates order of magnitude faster than a RTL model.

The SystemC has port that it refers to channel through interface and is an object for connection between modules. The channel of SytemC is container for communications protocols and synchronization. Channel provides paths to exchange information in a predefined protocol and implements one or more interface.

SystemC provides various channels for connection between modules or processes. We designed DMAC module using sc_fifo and sc_buffer channel. Comparison of performance on each channel is shown in Fig. 7.



	10	100	1000	1000
◆ sc_fifo	0.28	1.638	17.208	184.401
■ sc_buffer	0.292	1.654	21.33792	425.96631

Fig. 7. Comparison of performance on each channel

The more the amount of processing data is, the bigger time difference on each channel is as shown in Fig. 7. The time difference between sc_fifo and sc_buffer channel is caused by the internal scheme of communication method between each modules even if we design same system.

4. Conclusions

As a system modeling language, the importance of SystemC has been stressed in system design. It has a lot of features on modeling as well as verification. SystemC describes functionality and communication at various level of abstraction. In other words, it is possible to explore design space from high-level abstraction to low-level abstraction.

The implementation of a DMAC using SystemC is presented in this paper. We implemented a DMAC module in several abstraction-levels and compared performances of each level DMAC. We confirmed the difference of between abstraction levels, and between channels.

REFERENCES

[1] Frank Ghenassia, *Transaction Level Modeling with SystemC*, Springer, 2005

[2] David C. Black, Jack Donovan, *SystemC:From The Ground Up*, Eklectic Ally, Inc., 2004.

[3] Thorsten Grotker, Stan Liao, Grant Martin, Stuart Swan, *System Design with SystemC*, Kluwer Academic Publishers, 2002.

[4] J.Bhasker, *A SytemC Primer*, Star Galaxy Publishing, 2002.

[5] OSCI, *SystemC version 2.0 User Guide*, Synopsys, Inc.,2001.

[6] http://www.systemc.org.

[7] Miles J.Murdocca, Vincent P.Heuring, *PRINCIPLES OF COMPUTER ARCHITECTURE*, Prentice Hall.,2000.

[8] ARM, *PrimeCell DMA Controller*, Arm,Inc., 2005.