SoCBase-DE : A SystemC-based Multimedia System Design Environment

Sanggyu Park, Sangyong Yoon, Dosun Hong, Soo-Ik Chae

{sanggyu, syyoon, dshong, chae}@sdgroup.snu.ac.kr

CoSoC - Seoul National University – Korea

http://sdgroup.snu.ac.kr

Abstract

SoCBase-DE is a SystemC-based design environment that defines 8 channels such as FIFO and array for the TLM modelling and provides several templated implementations for the architecture exploration.

1. Introduction

Multimedia systems such as the H.264 codec and the 3D graphics engine are complex systems that contain much concurrent hardware, software, and memory components to achieve high performance, low power, and small area in short design time. To design such a complex system, the system-level functions should be partitioned into a set of concurrent components which should be designed manually or reused from previous designs.

In the ASIC design methodology which has been adopted in the design industry, the micro-architecture and the external interface of a component are closely related to other components or system-level architectures. Therefore, several important system-level decisions such as the hardware-software partitioning, allocation of on-chip memories, and communication architectures should be decided before the component authoring.

Since estimation and analysis of the system is limited in early design time, however, such decisions are often made by intuition which can be evaluated only after the system integration. In many cases, some components should be modified spending much effort and time. The design re-spin of ASIC is very risky because a modification of a component is often propagated to other related components.

The 'orthogonalization of concerns' explained in [1] is a good way to alleviate this problem by separating the computational cores from the rest parts which are closely affected by the system-level decisions or design changes

In our design experiences, implementations of the communication, memory, and synchronization are closely affected by the system-level decisions and later design changes. Therefore, it is preferable to implement these functions during in the system integration not in the early design time. Furthermore, the computational core parts become more reusable than bus-based IP cores because their communication, memory, and synchronization parts can be customized to be best-fit to the target system [2].

We defined several functional patterns related to the communication, memory, and synchronization functions such as FIFO, array, and broadcast, etc. Furthermore, we constructed various reusable implementations of these functional patterns by integrating several primitive components such as buses, on-chip or off-chip memories, flip/flops. These architectural patterns have different performance, area, and power characteristics to be selected and reused during the architecture exploration.

In this demonstration, we present the SoCBase-DE which is a SystemC-based design environment for multimedia applications that exploits the communication-memory-synchronization (CMS) library. SoCBase-DE consists of various point design tools. The Template Instance Generator (TIG) outputs source codes of transaction level, RT-level, and software implementations. The DE eXecution (DEX) platform enables the verification of progressively refined systems by seamless transaction-level and RT-level simulation, FPGA emulation and silicon. The Architecture Synthesis Tool (AST) is an interactive user interface tool for the convenient function modelling and template-based architecture exploration. It can invoke various tools such as TIG to generate the mixed-level source codes of the system, GNU gdb for the software simulation, Mentor Graphics ModelSim® for the RT-level simulation, and Synopsys Design Compiler® for the logic synthesis.

This paper organized as follows. In section 2, we briefly explained the design flow and the CMS library. In section 3, we summarized several point tools. In section 4, we present the H.264 decoder design results as a case study.

2. CMS Library and Design Flow

The collection of reusable functional and architectural patterns related to the communication, memory, and synchronization is the CMS library.

A system functions can be modelled as a network of computational cores which are connected to functional patterns defined in the library. Then, the computational cores can be refined into either hardware or software independently, and the instantiated functional patterns can be refined by selecting appropriate architectural patterns.

Since the concept is very similar to those of SystemC, the functional patterns defined in the library are called channel and the computational cores are called modules, and the implementations of channels are called the channel architecture templates (CATs).

We defined 14 CMS interfaces listed in Table 1. For each interface, we defined a transaction level interface (TLMIF), a RT-level interface (RTLIF), and a software programming interface (SWIF). TLMIFs and SWIFs are derived from the SystemC sc_interface class. Figure 1 shows the TLMIF and RTLIF of the sequential put interface (SPI). A channel has two or more interfaces. For example, the FIFO channel has one SPI and one SGI while the broadcast pattern has only one SPI and two or more SGI.

Table 1: CMS Interfaces

Interface Names	Methods	
Sequential Put (SPI)	void put(T†)	void sync()
Indexed Put (IPI)	void put(int, T†)	void pclear()
Sequential Get (SGI)	T get(), T peek(), void gclear()	
Indexed Get (IPI)	T get(int), T peek(int), void consume()	
Single Write (SWI)	void write(T), void sync()	
Indexed Write (IWI)	void write(int, T) , void sync()	
Single Read (SRI)	T read()	
Indexed Read (IRI)	T read(int)	
Event Notify (ENI)	void notify(), void wait_ack()	
Event Accept (EAI)	void wait_req(), void ack()	

†T: data type

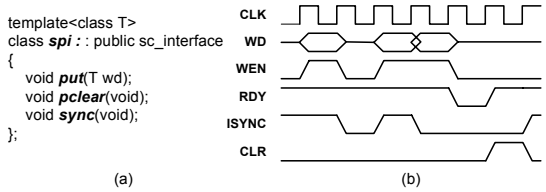

Figure 1: TLMIF and RTLIF of sequential put interface.

Currently, FIFO, Array, Variable, Event, Broadcast, Scatter, Gather, Conflux channels are defined. For each channel, we constructed various CATs of different performance, area, and power characteristics. For all channels and their CATs, SystemC-based TLMs for the transaction level simulation, HDL-based RTLs for the RT-level simulation and logic synthesis were developed.

Figure 2 shows the FIFO channels and its CATs. When the producer and the consumer are both hardware and located closely, the register FIFO CAT shown in Figure 2(a) or the Array-based CAT shown in Figure 2(b) can be used. The later is better choice when the internal memory size is large, and vice versa. When the producer and consumer are located distant or one is hardware while another is software, bus-connected versions shown in Figure 2(c)-(d) can handle these situations. When the producer is software, the bus-master-sequential-put (mSP) adapter is assigned to the embedded processor as a device driver. Since we provide software implementations of various adapters, device drivers can be automatically constructed according to the hardware-software partitioning decisions. No manual device driver description is needed.

3. Tools

3.1 Template Instance Generator

TLM models in the CMS library are described in C++ language overriding the sc_module and associated interface classes. The most TLM descriptions are template classes that have several parameters such as data types, memory sizes and priorities. Some TLM descriptions which have

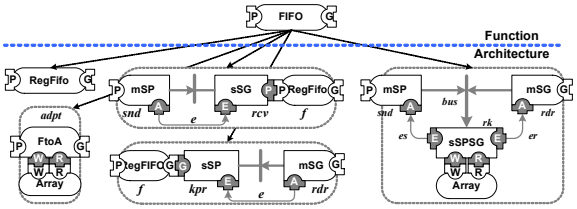

Figure 2: FIFO channel and its CATs

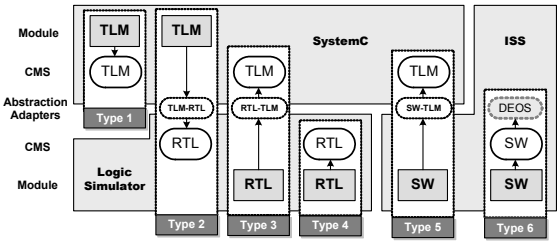

Figure 3: The DEX Platform

multiple producer and consumer ports are generated by the template instance generator (TIG) because current SystemC does not support multiple ports of the same type.

All RTL models are generated by the TIG because HDL does not support parameterized entity or module descriptions. Although VHDL provides the generic construct, Synopsys Design Compiler cannot synthesize parameterized descriptions.

3.2 DE eXecution Platform

The DEX platform[3] is a mixed abstraction level simulation or execution environment for the CMS-based function modeling and progressive architecture refinement. Six execution types can be mix-used for the verification of partially refined systems as shown in Figure 3.

The DEX platform provides various abstraction adapters that bridge different execution domains. DEOS is another important tool that can execute SystemC-based TLM descriptions as software without any modification.

4. Case Study: H.264 decoder

We designed a baseline H.264 decoder using the SoCBase-DE[4]. First, we captured the system-level function at the SystemC-based transaction level. The system TLM consists of 13 computation modules and 53 CMS channels. 12 array, 2 variable, 2 conflux, 1 broadcast channels were used and the rest were FIFO channels. The captured H.264 decoder TLM was refined for different decoding targets by selecting different CATs as summarized in Table 2.

Table 2: CMS Interfaces

		VGA	720p HD
Specification		30fps @ 50MHz	30fps @ 100 MHz
Cycles / MB		1,833	800
Area	Modules	98K gates	
	CMS	144 K gates	214 Kgates
On-chip SSRAM		2.8 KB	5.7 KB

5. References

[1] K. Keutzer, A. Sangiovanni-Vincentelli, “System level design: Orthogonalization of concerns and platform-based design”, Trans. on computer aided design of integrated circuits and systems, Vol. 19, No. 12, Dec. 2000

[2] Sanggyu Park, Sangyong Yoon, Soo-Ik Chae, “ Reusable component IP design using refinement-based design environment”, proc. of ASPDAC 2006.

[3] Sanggyu Park, Sangyong Yoon, Soo-Ik Chae, “A mixed-level virtual prototype environment for refinement-based design environment”, proc. of RSP, June, 2006.

[4] Sangyong Yoon, Sanggyu Park, Soo-Ik Chae, “Implementation of a H.264 decoder with template-based communication refinement”, proc. of Asia-Pacific Conference on Circuits and Systems, Dec. 2006.