A Fast Timing-Accurate MPSoC HW/SW Co-Simulation Platform based on a Novel Synchronization Scheme

Mingyan Yu, Junjie Song, Fangfa Fu, Siyue Sun, and Bo Liu

***Abstract*—Fast and accurate full-system simulation is needed for MPSoC design space exploration to achieve tight time-to-market design goals. In the field of full-system simulation, transaction level modeling with SystemC and traditional instruction set simulators (e.g. M5) based on C/C++ have their own advantages, separately. In this paper, a novel method for synchronizing M5 and SystemC modules is proposed to achieve fast and timing-accurate co-simulation. This method adopts event-driven scheduling and object-oriented programming technology. With this method, an MPSoC full-system co-simulation platform, which allows modeling the architecture at multiple levels of abstraction, is presented. The fast abstract model of M5 and timing-accurate model of SystemC has been used for hardware framework. On the other hand, a lightweight MPI library is implemented for communication of software applications. The simulation result shows that the novel scheme can obtain a fast simulation speed with no expense on simulation precision. Additionally a parallel MUSIC algorithm is designed which evaluates the validation of the MPSoC platform.**

***Index Terms*—Co-simulation; MPSoC; Synchronization; Timing-Accurate.**

I. Introduction

It has been widely accepted that Multiprocessor System-on-Chip (MPSoC) is the most promising way to keep on exploiting the high level of integration provided by the semiconductor technology and matching the constraints imposed by the embedded system market in terms of performance and power consumption [1]. However, as more processors and hardware components are integrated, designing and programming these complex multiprocessor architectures has become a major challenge [2]. In this context, fast and accurate full-system simulation is necessary to efficiently explore different implementations and design parameters in order to achieve a cost-efficient MPSoC implementation [3].

Nowadays, many research institutes have presented their MPSoC full-system simulators written in SystemC [6]–[8]. But for each simulator the details of components are quite different, and most of them are not open source. In this condition, new scheduling schemes or communication protocols for MPSoC will not be convictive, if they only experiment with their own simulator. Moreover, it is a waste of time to implement a simulation platform from scratch for a new institute to launch. Hence a wide-accepted open-source MPSoC platform is needed for design exploration.

Fortunately, several full-system simulators for computer system architecture research (e.g. M5 [4], [5]) are open source, and most of them have excellent infrastructure for function extension. Thus the platform for MPSoC design exploration will be implemented with great efficiency and speed, if these full-system simulators are utilized wisely. But considering institutes who have been engaged on SystemC simulator for several years, lots of modules written in SystemC have been made at different abstraction levels, which can be inserted into the simulator mentioned above to achieve a cost-efficient implementation. However, most of these open-source simulators are written in C++, so a synchronization scheme is necessary for fast and accurate co-simulation.

In this paper, a novel scheme for synchronizing M5 and SystemC module is proposed, which uses the local time of M5 and SystemC time to achieve timing-accurate co-simulation. This scheme can guarantee a fast simulation speed with no expense on simulation precision. With this scheme, an MPSoC full-system simulation platform is presented which unites the advantage of M5 and SystemC. Any module of our platform can be implemented at system level with M5 or transaction level with SystemC. With the support to full-system simulation, our platform can fully analyze a parallel application to find out the performance bottleneck at the early stage of design flow. Additionally, for inter-process communication, we implement a lightweight MPI library [10] which contains a minimal working subset.

The rest of the paper is organized as follows. Section 2 presents an overview of related work, while Section 3 presents the scheme of co-simulation for M5 simulator and SystemC. Section 4 describes the architecture of the MPSoC platform and the parallel programming models for inter-process communication. Section 5 gives the experiment result for this co-simulation scheme, and Section 6 evaluates the performance of parallel MUSIC algorithm with our MPSoC platform. We conclude in Section 7.

Manuscript received November 25, 2009, revised January 28, 2010.

M. Yu is with the Micro-Electronics Center, Harbin Institute of Technology, Harbin, China. He is the director of Micro-Electronics Center. His current research interests are in computer architecture, System-on-chips and IP design (e-mail: myyu@hit.edu.cn)

J. Song, F. Fu, S. Sun, and B. Liu are with the Micro-Electronics Center, Harbin Institute of Technology, Harbin, China (e-mail: jjsong7@163.com, fff1984292@163.com, sunmissmoon@163.com, liubohit@gmail.com).

II. Related Work

Instruction Set Simulator (ISS) and Transaction Level Modeling (TLM) with SystemC are both commonly adopted technologies for modeling of MPSoC simulation platform. Several researchers have proposed methodologies for co-simulation between ISS and SystemC. An overview of these related work shows that existing methodologies can be grouped in three broad categories: cycle-accurate co-simulation [9], timing-accurate co-simulation [12] and untimed co-simulation [11].

Boukhechem et al. [9] proposes an MPSoC platform cycle-accurate co-simulation methodology based on an open source ISS OR1Ksim within SystemC as simulation environment. In this platform, the ISSs are wrapped under SystemC. So a SystemC wrapper interface is added to the ISS C model. At every positive clock edge, the SystemC wrapper interface calls the corresponding C function inside the ISS via IPC. Cycle-accurate computation simulation slows down the overall simulation significantly, which is the main disadvantage of this simulator.

Cordibella et al. [12] presents a HW/SW co-simulation framework consisting of a timing-accurate interaction of a SystemC simulator with an array of ISSs. The synchronization mechanism of this platform follows an asymmetric scheme, where one of the two simulators (the master) explicitly controls the execution of the other (the slave). To achieve this, they modify the SystemC kernel, which compares ISS time and SystemC time in the simulation loop. A weakness of this mechanism regards the case in which SystemC simulator is ahead of time with respect to the ISS simulator; in this case the result of a read operation or the behavior of SystemC after a write operation may depend on the delay of ISS with respect to SystemC.

Benini et al. [11] implements two alternative schemes to allow a transparent integration of ISSs within the SystemC simulation framework. The former embeds the ISS within the SystemC simulator, which is just like [9]. The latter uses a wrapper called gdbAgent whose main function is executing the gdb and controlling its execution. The main disadvantage of this scheme is that synchronization between the SystemC time and the ISS simulated time can not be implemented completely.

To cope with the problems mentioned above, a novel timing-accurate co-simulation scheme is presented which implements synchronization between the SystemC time and the ISS simulated time, and keeps a fast simulation speed simultaneously.

III. Technology of Co-simulation

M5 provides a full-system simulation platform and an excellent object-oriented infrastructure, while SystemC permits different abstraction levels and combination of different levels in one model. Therefore building a co-simulation platform will unite the advantages. But as mentioned above, synchronization and simulation speed are the most two important problems we should figure out.

M5 is an event-driven simulation which uses a global event queue to simulate execution of each instruction and

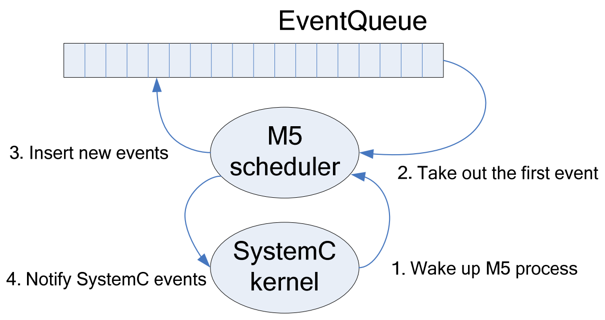


Figure 1. Technology of synchronization.

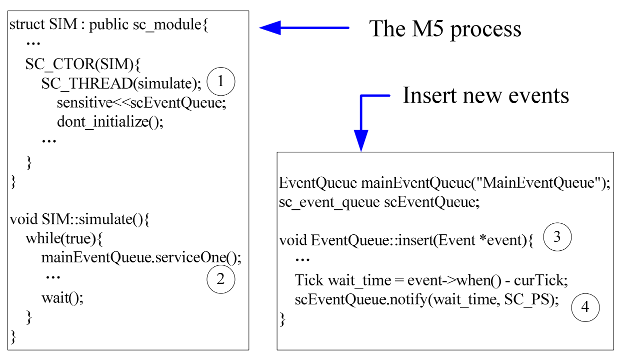
move forward the simulation time. During the execution of current event, new events will be notified, and it calls the method 'insert' of class EventQueue which will insert the events into the global event queue at a proper position according to their execution time. When the current event finishes, the scheduler will take out the first event in the event queue to execute, whose execution time is the earliest, and then the scheduler assigns the simulation time with the execution of this event.

Similarly, SystemC is event-driven. After sc_start routine is called in main function, all SystemC processes are placed into a ready pool. Then, the scheduler takes out processes from ready pool one by one. During execution, a process may invoke immediate or delayed event notification (i.e. sc_event.notify()) and possibly cause one or more waiting processes to be placed in the ready state.

In order to achieve synchronization, these two schedulers must be unified. The principle of co-simulation is synchronizing the local time of M5 and SystemC time. As mentioned in [12], an intuitive method is that one of the two simulators (the master) explicitly controls the execution of the other (the slave). The simulation time of the master goes forward by itself. The master is told when to schedule the slave. During the execution of the slave process, the slave updates its local time to achieve synchronization, and it tells the master when to schedule it again according to the local time. Not like [12], we only modify the M5 scheduler, and do not touch the SystemC kernel, which will improve the portability of the co-simulation platform.

As shown in Fig. 1, SystemC is chosen as the master, and M5 scheduler is regarded as a SystemC process. When a SystemC event related with M5 happens, SystemC kernel will wake up M5 scheduler. Then M5 scheduler takes out the first event in EventQueue to execute. During the execution, new M5 events may be notified. When an M5 event is inserted into EventQueue, the method 'insert' will also notify a SystemC event. Then the SystemC event will wake up the M5 process at the expected time once again.

It should be noticed that more than one SystemC event may be notified at a time, so class sc_event does not satisfy this condition. For sc_event, only the nearest time notification can be executed, so only one M5 event can trigger a SystemC event. Class sc_event_queue is chosen to solve this problem, which is added in SystemC version 2.1. An sc_event_queue object can be scheduled multiple times even for the same time. With this method, we achieve synchronization between the SystemC time and the ISS simulated time, and ensure that all of the M5 events and SystemC processes are executed one by one according to their execution time.



```
struct SIM : public sc_module{
    ...
    SC_CTOR(SIM){
        SC_THREAD(simulate);  (1)
            sensitive<<scEventQueue;
            dont_initialize();
        ...
    }
}

void SIM::simulate(){
    while(true){
        mainEventQueue.serviceOne();
         ...                   (2)
        wait();
    }
}
```

```
EventQueue mainEventQueue("MainEventQueue");
sc_event_queue scEventQueue;

void EventQueue::insert(Event *event){   (3)
    ...
    Tick wait_time = event->when() - curTick;
    scEventQueue.notify(wait_time, SC_PS);   (4)
}
```

Figure 2. Segments of the SystemC-master scheme for synchronization.

Fig. 2 shows some essential segments for synchronizing between M5 and SystemC. The M5 scheduler is wrapped in a SystemC thread called 'simulate' which is sensitive to a sc_event_queue called 'scEventQueue'. The numbers shown in Fig. 2 just correspond to those in Fig. 1.

M5 has excellent simulation speed which benefits from its high abstraction level. When TLM methodology is used for SystemC modules, the co-simulation speed is approaching to M5 with the low cost of schedule of SystemC kernel. By modeling only the necessary details, we can realize huge gains in modeling accuracy as well as simulation speed.

IV. MPSoC Co-simulation Platform

With the method mentioned above, an MPSoC full-system simulation platform is presented. The platform unites M5 and SystemC, and it is convenient for any M5 module in the platform to be replaced by timing-accurate SystemC model and vice versa. The platform supports full-system simulation, so a parallel application can be fully analyzed through simulation to find out the performance bottleneck at the early stage of design flow.

As shown in Fig. 3, the whole platform can be divided into two parts: HW which is composed of two major components (i.e. PEs and NoC), and SW which includes OS, MPI library and parallel applications.

Each PE includes CPU, local memory and some peripherals. A PE is connected to NoC through network interface (NI). All of the modules of PEs derive from M5, and the peripherals of M5 are all connected to memory bus. NI communicates with CPU via DMA, which can free up processor cycles. NoC module which is composed of routers is part of a NoC simulator written in SystemC. The topology and routing algorithm of this module can be configured during elaboration. Each router in the NoC module connects to a NI module respectively.

NI can be considered as a transactor between M5 and SystemC. It is a type of channel specialized to translate between modules with different interfaces. In M5 all of the device modules inherit from class MemObject, and in SystemC all of modules inherit from class sc_module. Therefore NI must inherit from both of them. The physical-level communication between M5 and SystemC can be implemented by the Send FIFO and Receive FIFO in NI module. NI reads data from local memory via DMA, and pushes the data into Send FIFO. Then a SystemC thread in NI module sends the data to routers. On the other hand, another SystemC thread in NI module receives data from routers, and pushes

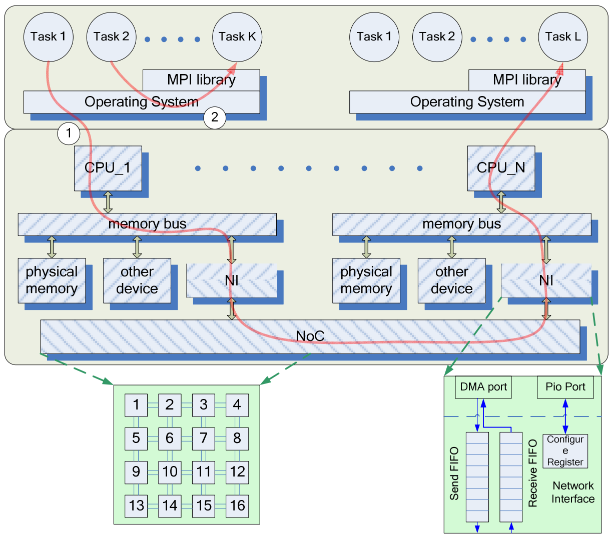


Figure 3. MPSoC co-simulation platform.

the data into Receive FIFO. Then an M5 event in NI writes the data to local memory.

For data transfer and synchronization between processes of the parallel application, the platform must provide a set of communication primitives. The primitives must support two kinds of inter-process communication: inter-processor communication (path 1 in Fig. 3) which calls device driver of NI to transmit data and intra-processor communication (path 2 in Fig. 3) which uses shared memory to communicate with each other. A lightweight MPI library is implemented to satisfy the request. The library contains six MPI functions, and other communication protocols can be developed based on these functions.

- MPI_Init(): initializes the MPI execution environment.
- MPI_Finalize(): terminates MPI execution environment.
- MPI_Comm_size(): determines the size of the group associated with a communicator.
- MPI_Comm_rank(): determines the rank of the calling process in the communicator.
- MPI_Send(): blocking send.
- MPI_Receive(): blocking receive.

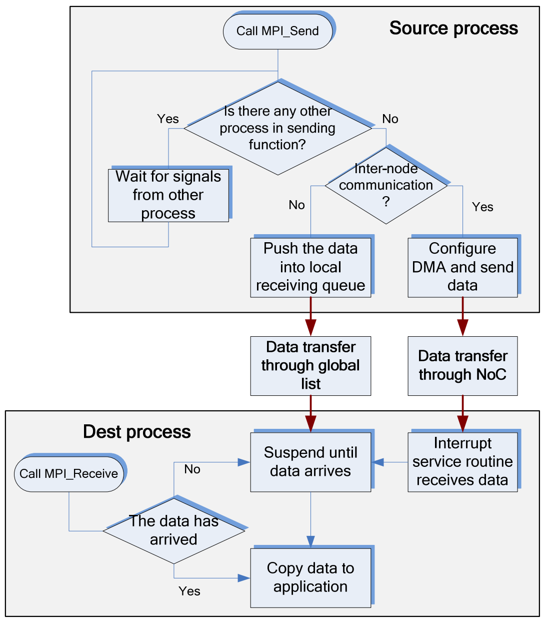

Source process
Call MPI_Send
Is there any other process in sending function?
Yes
No
Wait for signals from other process
Inter-node communication ?
No
Yes
Push the data into local receiving queue
Configure DMA and send data
Data transfer through global list
Data transfer through NoC
Dest process
Call MPI_Receive
No
Suspend until data arrives
Interrupt service routine receives data
The data has arrived
Yes
Copy data to application

Figure 4. Process of data transfer.

```
SimLoopExitEvent * simulate(Tick num_cycles){
   …
   while(1){
      …
      mainEventQueue.serviceOne();
      …
      while(mainEventQueue.nextTick()>curTick){
         if(mainEventQueue.nextTick() - curTick>CYCLE){
            sc_time sc_t(CYCLE, SC_PS);
            sc_start(sc_t);
            curTick+=CYCLE;
            }
         else{
            Tick t = mainEventQueue.nextTick() - curTick;
            sc_time sc_t(t, SC_PS);
            sc_start(sc_t);
            curTick+=t;
            }
         }
      }//end while(1)
}//end simulate
```

Figure 5. Segments of the M5-master scheme.

The implementation of path 1 and path 2 in MPI library is shown in Fig. 4. When a process wants to send data to other process, it calls MPI_Send routine. Because only one process can enter MPI_Send routine at a time, the process will be suspended until it gets the mutex. Then, the routine checks whether the destination process is in the same PE. If so, the routine inserts data to a global list from which receiving routine can obtain expected data (path 2). If not, the routine will configure DMA register, and then start a DMA transfer (path 1). On the other hand, when a process calls MPI_Receive routine, it will be blocked until the expected data is inserted in the global list by interrupt service routine or by MPI_Send routine in the same PE.

V. Performance Analysis

In order to evaluate the performance of the co-simulation scheme presented above, another intuitive scheme is implemented, which can also ensure synchronization between SystemC and M5. In this scheme, M5 is chosen as master, and SystemC processes are executed every time at the end of scheduling loop. During each iteration of the loop, SystemC simulator will keep running until the execution time of next M5 event comes. The simulation time of calling sc_start routine each time is not more than a constant which ensures that the start time of all M5 events notified by SystemC is later than the finish time of the simulation. If not, a new event may be inserted at the head of the event queue of M5, which will break down the synchronization between SystemC and M5. The implementation of this scheme is shown in Fig. 5.

These two schemes are compared in our co-simulation platform, and the same application is executed in the platform. The simulation environment is shown in Table II, and a parallel MUSIC algorithm is run on a four-core platform. The result is shown in Table I, which indicates SystemC-master algorithm is 7.6 times faster than the other.

TABLE I. COMPARISON OF SIMULATION TIME

	SystemC-master scheme	M5-master scheme
Simulation time (sec.)	332.07	2520.45

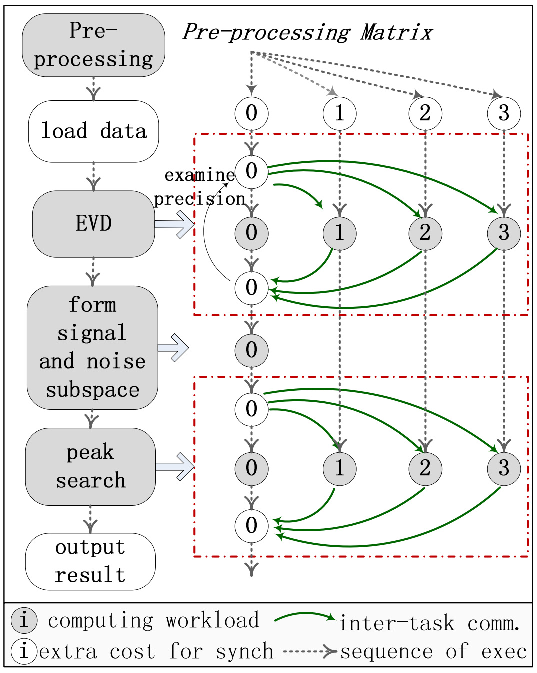


Figure 6. Parallelization of MUSIC algorithm.

VI. Case Study

In order to validate the MPSoC platform, and evaluate the performance of the parallel programming models we implement, a parallel MUSIC algorithm is designed with the lightweight MPI library.

MUSIC, which is an acronym for MUltiple SIgnal Classification, is a classic spatial spectrum estimation technique. It is based on partitioning the estimated multi-channel covariance matrix from a linear or planar array into a noise and signal subspace and finding a direction vector which is orthogonal to the noise subspace. The algorithm can be divided into three stages: pre-processing, EVD and peak-search, which is shown in Fig. 6. In the parallel MUSIC algorithm, EVD and peak-search are parallelized. MPI routines are used for communication among processes.

According to the flow graph in Fig. 6, the MPSoC co-simulation platform can be configured as shown in Table II.

TABLE II. PARAMETER OF MPSOC PLATFORM

network topology	Size of network	class of service	route algorithm
2D-mesh	2x2	best effort	XY-routing

Speedup is commonly used to evaluate the performance of parallel computation. Speedup for p processors is the ratio of execution time for the serial program running on single processor to execution time for the parallel program running on p processors. As shown in Table II, the parallel MUSIC algorithm is executed on a four-core platform, and the speedup we gain is 2.6. The result is compared with speedup in [13] which uses fore-core DSP for computation and shared memory for communication. With our platform, the speedup is enhanced by 9%.

VII. Conclusion

In this paper, a novel method for synchronizing M5 and SystemC module is proposed, and an MPSoC full-system simulation platform is presented which unites the advantage of M5 and SystemC. The simulation result shows that our scheme is 7.6 times faster than M5-master scheme. A parallel MUSIC algorithm is used to validate the MPSoC platform, and the speedup is 2.6. In the near future, we will extend our MPI library to support more complicated applications. Moreover, we will develop an interface for co-simulation between our MPSoC platform and Verilog modules.

References

[1] J. Ceng et al., "MAPS: An Integrated Framework for MPSoC Application Parallelization," DAC 2008, June 8–13, 2008, Anaheim, California, USA, pp.754–759.

[2] K. Huang et al., "Simulink-Based MPSoC Design Flow: Case Study of Motion-JPEG and H.264," DAC 2007, June 4–8, 2007, San Diego, California, US, pp.39–42.

[3] E. Cheung, H. Hsieh and F. Balarin, "Fast and Accurate Performance Simulation of Embedded Software for MPSoC," ASP-DAC 2009, January 19–22, 2009, pp. 552–557.

[4] http://www.m5sim.org/wiki/index.php/Main_Page.

[5] N. Binkert, R. Dreslinski, L. Hsu, K. Lim, A. Saidi and S. Reinhardt, "The M5 Simulator: Modeling Networked Systems," IEEE Micro, Volume 26, Issue 4, July-August, 2006, pp. 52–60.

[6] N. Pouillon, A. Becoulet, A. Mello, F. Pecheux and A. Greiner, "A Generic Instruction Set Simulator API for Timed and Untimed Simulation and Debug of MP2-SoCs," Rapid System Prototyping, June 23–26, 2009, pp. 116–122.

[7] S. Boukhechem and E. Bourennane, "TLM Platform Based On SystemC For STARSoC Design Space Exploration," Adaptive Hardware and Systems, June 22–25, 2008, pp. 354–361.

[8] C. Jalier, D. Lattard and G. Sassatelli, "A Flexible Modeling and Simulation Framework for Design Space Exploration," System-on-Chip, November 5–6, 2008, pp. 1–4.

[9] S. Boukhechem and E. Bourennane, "TLM Co-simulation for an Open Source MPSoC Platform under STARSoC Environment," System-on-Chip, November 5–6, 2008, pp. 1–6.

[10] "MPI: A message passing interface," Supercomputing, November 15–19, 1993, pp. 878–883.

[11] L. Benini, D. Bertozzi, D. Bruni, N. Drago, F. Fummi and M. Poncino, "Legacy SystemC Co-Simulation of Multi-Processor Systems-on-Chip," Computer Design: VLSI in Computers and Processors, September 16–18, 2002, pp. 494–499.

[12] S. Cordibella, F. Fummi, G. Perbellini and D. Quaglia, "A HW/SW Co-Simulation Framework for the Verification of Multi-CPU systems," High Level Design Validation and Test Workshop, November 19–21, 2008, pp. 125–131.

[13] H. Liu, P. Wei and X. Xiao, "Parallel Implementation of MUSIC Based on a Special Parallel Processing Machine," Systems Engineering and Electronics, Vol.23 ,No.1, 2001, pp. 86–89.