Software/Hardware Hybrid Network-on-Chip Simulation on FPGA

Youhui Zhang, Peng Qu, Ziqiang Qian, Hongwei Wang, Weimin Zheng

Software / hardware Hybrid Network-on-Chip Simulation on FPGA

Youhui Zhang[1,*], Peng Qu[1], Ziqiang Qian[1], Hongwei Wang[1], Weimin Zheng[1]

[1] Department of Computer Science and Technology, Tsinghua University
100084 Beijing, China
zyh02@tsinghua.edu.cn

Abstract. In this paper, a software-and-hardware hybrid simulation method for CMP (Chip-MultiProcessor) system is designed, as well as its performance model. In detail, the NoC (Network-On-Chip) module is totally simulated by the FPGA resource; a software-and-hardware interaction interface of this module is provided so that the simulation software running on the on-chip soft core can cooperate with the NoC to complete the whole simulation. In other words, the most time-consuming and relatively-fixed part is implemented by hardware and others are implemented by software, which maintains simulation flexibility and high performance owing to the compact on-chip design. We implement this design on the Xilinx's Virtex 5 155T chip and the work frequency is 100Mhz. Compared with a typical software counterpart, the simulation speed of NoC is more than 3000 times faster; and the advantage is widened further with the increasing injection rate. Moreover, compared with another hybrid method executing the software part on the host CPU, it is still fairly faster although the host performance is much higher than the on-chip core.

Keywords: Network on chip, FPGA, Simulation

1 Introduction

Networks-on-Chip (NoC) [1][2] is an approach to designing the communication subsystem between IP cores in a chip, which separates the on-chip communication from computing and storage to improve scalability. For multi-core architectures, NoC has been regarded as one of the crucial and common components. NoC interacts with other functions on-chip, like the cache-coherency (CC) mechanism, the cache-line distribution and so on, to affect the whole system efficiency. It means system architects and researchers must include detailed NoC simulation as part of any complete system simulation.

Detailed NoC simulation is a time-consuming process [3] [4]. Thus quite a few existing projects [5][6][7][8] have used FPGA resource to emulate NoC for high speed, and the reconfigurable feature of FPGA provides a certain degree of flexibility. Fur-

thermore, to simulate the system rather than the NoC itself is the final destination. Thus, NoC simulation should interact with other simulation modules efficiently.

This paper presents a software/hardware hybrid design for such requirements. The design principle is: the detailed and time-consuming NoC simulation is totally accomplished by the hardware resource (FPGA) and a simple but general purpose interface between the software and hardware simulations has been provided. To reduce the roundtrip transfer latency between SW and HW, we use the on-chip soft processor to execute simulation software to drive the NoC.

This paper gives the following contributions:

1. A general SW/HW hybrid simulation framework of CMP has been designed and implemented. The kernel is a configurable NoC simulator on the FPGA and some parameters such as data width, network topology, channel FIFO depth, virtual channel options, and packet length and so on can be assigned without recompiling / resynthesizing. Now in a Xilinx Virtex 5 FPGA chip, a NoC to scale to 16 nodes can be simulated with one MicroBlaze CPU on-chip and the work frequency is 100Mhz. A simple interface between SW and HW is also designed that the simulation software running on the on-chip soft core can cooperate with the NoC.
2. Performance analysis is given to show under what circumstance such a design is preferred. There are two ways to run the simulation software, on the on-chip soft CPU or on the host general-purpose CPU. The former usually works on a much lower frequency (1/10 ~ 1/30 of the latter) but its interaction speed with the hardware module is very high because they co-exist on the same chip. For the latter, the transfer trip between the FPGA and the host application is fairly long. We present a sequential and quantitative performance model for such situations
3. The running efficiency is compared with its software counterpart. Result shows that, the NoC speed is 3000 times faster than the 100% software method for the simulation of a 4X4 NoC. Moreover, two usage examples are presented: For the first, two directory-based CC protocols for the multi-core architecture design have been simulated on this framework. In this case, all parts other than the NoC are simulated on the on-chip soft core. Compared with the counterpart that executes the software part on the host CPU, its speed is much higher because the HW/SW transfer latency has been reduced remarkably. The second is a trace-driven case. Compared with the counterpart that executes all parts (including the equivalent NoC simulation implemented by software) on the host CPU, our method is much faster, too.

2 Related Work

Many NoC simulators are achieved by software. Some are standalone tools, like BookSim [3] and DARSIM [4] while others are used as the interconnection module of full-system simulators (GEMS [9], SimFlex[10], etc.). Software implementations are very easy for reconfiguration, fast to compile and deterministic. However they are

quite slow, so that users often have to maintain reasonable running speed at the expenses of simulation accuracy.

Several FPGA-based NoC simulators have been designed [5], [6], [7], [8], [11] that increase simulation performance by 10 times and more. [5] presents an emulation environment implemented on an FPGA that is suitable to explore a wide range of NoC design-space. It gets a speed-up of 4 orders of magnitude with respect to a SystemC simulation of the same network. But re-generating is required when changing the configuration of the network.

DRNoC [6] solves this problem based on the re-configurability of FPGAs. Speedups of hundreds of time have been achieved in the presented use case compared with a non-reconfigurable approach (synthesis based). However, partial reconfiguration requires a special design flow and incurs area overheads; and only a few select devices are available. NoCem [7] improves emulation density over [4] and implements a 9-node mesh network on a single FPGA. Moreover, an example memory architecture exploration platform based on this tool has been provided. [8] virtualizes a single router on an FPGA, allowing the simulation of a NoC with multiple routers. An off-chip ARM processor stores N contexts for the router model and orchestrates the emulation of the N-node network. However, the off-chip ARM/FPGA communication is a performance bottleneck.

Further, DART [11] virtualizes NoC totally on the chip while no help of any off-chip CPU is needed. It provides a flexible FPGA-based NoC simulator platform by decoupling the simulator architecture from the architecture of the simulated system. This technology has been also used in some multi-core CPU simulations, like Protoflex [12], RAMPGold [13] and so on.

Our work can be regarded as an extension of the above-mentioned projects. We design a straight and efficient HW/SW interface to control and stimulate the FGPA NoC module, and give a performance model of such a hybrid design to judge in what case it is beneficial in terms of performance.

A similar hybrid work is [14]. It is focused on how to design the FPGA platform efficiently, which is used as an emulation platform for the cache and NoC designers to verify their designs in cooperation with the full-system software simulator. In contrast, our work provides a flexible NoC simulator on FPGA and all other modules, including the cache system, are implemented by software on-chip. Then, researchers can explore the design space more flexibly as less RTL (Register Transfer Level) codes are needed, and the SW/HW interaction is more efficient owing to the compact on-chip design.

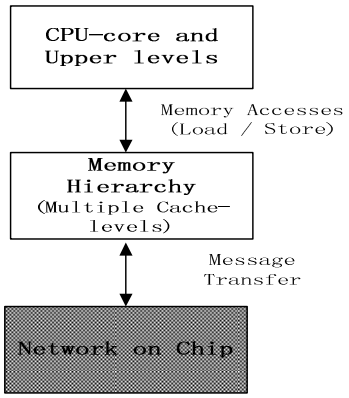

Fig. 1. Levels of the micro-architecture of CMPs.

3 System Design

From the viewpoint of the system architects or researchers, a chip multi-core processor (CMP) contains the following levels (Fig.1).

- Network on Chip: Most NoCs are packet-switched. The router usually implements wormhole routing with virtual channel (VC) flow control. Each packet in the network consists of the header flit to setup a route, an arbitrary number of data flits that contain the packet's information and one tail flit that will free the router's resources. All communications, including explicit accesses issued from CPU cores or the others (like control messages used by the CC protocol), are transferred in this level.
- Memory hierarchy: It includes all of cores' local caches and / or on-chip memories. Access requests from CPU cores will be handled by this level before entering into the NoC. If CC is maintained by hardware (which is true for most CMP designs today), the corresponding components (for example, the CC directory) are also regarded as part of this level.
- CPU cores and upper-level components.

In our design, the whole NoC is emulated by FPGA and quite a few of parameters can be configured without recompiling. Detailed descriptions of parameters are listed in Table 1. Of course, the simulation scale is limited by the available on-chip resources.

Further, we wrap the hardware kernel to provide control signals for the simulation software. From the viewpoint of software, the NoC module works like a function call: when all of the upper level's architecture-states in one simulated cycle have been updated, the software simulator invokes the SW/HW interface to promote the NoC simulation a cycle. Because the HW running frequency is much higher than the software simulation[1], this sequential design does not affect the simulation speed.

Table 1. Parameters of the NoC simulator

Configurations	**Valid Options**
Topology	Mesh,Torus,2D-Torus
Data width	1-256 bits
Packet Length	2,4,8,16 words
Pipeline latency of the router	Arbitrary (> 3)
VC Number	0/2/4
FIFO Length	2,4,8,16

[1] In our design, MicroBlaze and NoC both work at 100Mhz and a machine instruction on the MicroBlaze will take at least 3 cycles to complete. Therefore, during a whole NoC cycle, there is almost no progress for software.

3.1 The Internal Design

From the physical view, the NoC simulator contains multiple nodes connected by a crossbar, which allows all-to-all communication mode. Because it can restrict the communication pattern through this interconnect, this design is able to model the connectivity of the target topology. Then from the viewpoint of simulation, arbitrary topologies can be simulated. In the current design, we only introduce 5 entry points into one NoC node (*North, South, West, East* and *Local Access Point*), which limits the number of types that can be simulated.

Each node is a timing model, including the channel FIFOs, a router and a traffic generator: The first sends and receives data-flits to and from the node through those points; the second consists of arbitration blocks used to judge what is transmitted and when. The traffic generator connects to the *Local Access Point* to inject data flits into NoC. Each node has parameters that can be configured to match the properties of the component they simulate, without modifying the RTL codes (using VHDL generics).

- Traffic Model

Each data-flit in the NoC is described by a 32-bit value. It contains the metadata such as the source / destination addresses, the packet length, a timestamp that indicates when the flit should be forwarded, and the injection time to compute latency at the destination. The bit-width is dependent on the range of configuration parameters: For a NoC containing 16 nodes, 8 bits are used to represent the source and destination IDs, and we reserve extra 4 bits for expansion. Now the maximum of packet length is 16, so that 4 bits are used here. In addition, the timestamp occupies 6 bits and 10 bits for the latency computation.

For flow control, the credit-based mechanism is used: The upstream router keeps a count of the number of free flit buffers in each virtual channel downstream. Then, each time the upstream router forwards a flit, thus consuming a downstream buffer, it decrements the appropriate count. If the count reaches zero, no further flits can be forwarded until a buffer becomes available. Once the downstream router forwards a flit and frees the associated buffer, it sends a credit to the upstream router, causing a buffer count to be incremented. The 12-bit credit descriptor contains a timestamp (6 bits) and a virtual channel identifier.

- Timing Model

The channel FIFO models the timing information of latency of a wire link; each FIFO contains multiple virtual channels (VCs) and a RTL parameter controls the number of VCs to incorporate (up to 4 for this design). For each incoming flit, it will be queued into its VC and the internal timing logic can compute the corresponding de-queue timestamp.

The router models a four-stage wormhole VC router with credit-based flow control. Each router connects to four channel FIFOs and a traffic generator. The deterministic routing (X-Y routing) is supported because this simulator is mainly used to simulate the mesh (or mesh-like) topologies now. Router latency is modeled by incrementing

the flit timestamp (it is also configurable) when it leaves. Contention in VC and switch allocation is also modeled by adjusting the timestamp appropriately.

3.2 The Configuration and Result-collection Interface

The interface contains the following types of signals or modules:

- Packet Injection: Packet control is written to the channel FIFO for the local access point. It will contain the metadata of packet such as source / destination addresses, packet length, and so on, which will be used to route the packet.
- Statistics Export: There are three counters per traffic generator to record the number of injected and received packets and the cumulative packet latency.
- Router Metadata: These lines are used to collect running status of each router, which can be used to locate and analyze the hot spot(s). Now they mainly consist of various VC status signals (empty or full).
- Clock Signal：The clock signal of NoC is connected to a special strobe register. Once this register has been written, a clock signal of high frequency will be issued to promote the NoC simulation a cycle.
- Reset Signal: It is a synchronous signal and the configuration is lost during the reset.
- Simulation configurations: As mentioned earlier, each node has configurable parameters (packet length, VC Number, FIFO length, etc.). These parameters are chained in a 16-bit shift register. A software tool sends the configuration bits over an RS232 serial interface.

From the system point of view, the NoC is a device attached to the processor bus. Thus software can access the above-mentioned signals through the memory-address mapping mechanism.

3.3 Performance Model

A sequential simulation model is presented here. In another word, the simulation software works with the hardware NoC sequentially. It is also a common case for quite a few widely-used full-system simulators, like SIMICS+GEMS and so on. Therefore, the elapsed time for one simulated cycle, E, can be represented by Equation 1.

$$E = T_{sw} + T_{hw} + T_{interaction} \quad (1)$$

T_{sw} denotes the elapsed time of software execution and T_{hw} is the hardware time. The last one represents the roundtrip latency of HW/SW interaction.

There are two modes of sequential simulation.

- Both software and hardware simulations are completed in the same chip.

In this case, both of $T_{interaction}$ and T_{hw} are of the order of magnitude of 10 ns, while T_{sw} is much larger. For example, the test of this design shows that MicroBlaze and

NoC both work at 100Mhz and a machine instruction on the MicroBlaze will take averagely 3 cycles to complete. It means the software simulation will consume tens or hundreds of instructions to finish a simulated cycle. Then, in this case E mainly depends on T_{sw}.

- The software is running on the host.

Here T_{hw} is still of the order of magnitude of 10 ns and $T_{interaction}$ is much larger. Tests show that, in our design the average roundtrip latency between the host software and the NoC is about 0.3ms (through the GB Ethernet cable). On the other hand the software execution time is smaller: for an n-core CPU to simulate m target cores (m >= n), it is about 1/(30*n) of the previous version[2] if we assume the frequency of the host CPU is 3Ghz.

In this case, E equals with (300000ns + T_{sw} / (30*n)). The conclusion is straightforward: if the operation in one simulated cycle is too complex, for example, it takes the core on-chip more than 30000 cycles (10ns per cycle) to complete, using the host CPU is preferred (just like [14] did); otherwise, the on-chip mode is better.

4 Implementation and Evaluation

4.1 Implementation

We use the open source implementation of NoC emulator [7] as the foundation, which is a body of VHDL code configurable by a top-level package file that can create a variety of Network on Chips on parameters of data-width, virtual channel implementations, topology, and in-network buffering lengths.

We wrap this emulator as described in Section 3.1 to provide the SW/HW interface. As mentioned in Section 3.2, because the NoC simulation is actually used as a timing model, the original design of in-network buffers has been simplified. In detail, the packet header is necessary to be stored and forwarded node by node; for data flits, only the corresponding arbitration and flow control behaviors should be simulated while no real data-transfer is needed.

The FPGA platform used is BEE3. One BEE3 module consists of four large Virtex-5 155T FPGA chips. In addition, up to 4 Gigabit Ethernet interfaces allow a full-duplex data communication between each BEE3 module and a host server.

This NoC simulator is created by Xilinx ISE 12.4. For a 4X4 mesh, about 60% of FPGA slices have been used and its running frequency is 100Mhz. The remaining resources are used to occupy the MicroBlaze CPU.

[2] Because the detailed simulation of one core is usually implemented as a large and tightly-coupled state machine, it is difficult to be parallelized. On the other side, more than one target core can be simulated in parallel on several host CPU cores as they only interact with each other through the NoC.

BEEcube Platform Studio (BPS) is employed, too, which is a system-level IDE specially designed for BEE3. We use soft registers provided by BPS to connect the software simulation to the NoC. Such a register works like a normal hardware register; the difference lies in that it is used as a bus device that software can access. In order to improve the HW/SW interaction performance, several 128-bit-wide soft registers are used to supply data as much as possible once.

Moreover, external DRAM has been mapped into the MicroBlaze's address space so that the simulation software on-chip can access enough memory space. Thus, the work flow of one simulated cycle is described as follows:

Step 1) Software reads all output signals of the hardware NoC;

Step 2) Software completes all simulated events (of the memory hierarchy and / or the above level) in the current cycle.

Step 3) Software updates all input signals of the SW/HW interface accordingly.

Step 4) Trigger the reg_clk_strobe register.

4.2 Usage Examples and Tests

Table 2. Configurations of the comparison

Topology	2D-Mesh
NoC Scale	4*4
FIFO length	8
Packet Length	1
Link latency per flit	1 cycle
Router pipeline latency per flit	4 cycles

- Running Performance Comparison

We use a well-known software NoC simulator, BookSim [15], as the counterpart and compare its performance with the FPGA version under different flit-injection-rates (configuration time is excluded). The flit-injection-rate defines the rate at which packets are injected into the simulator; for example, setting flit-injection-rate = 0.25 means that each node injects a new packet in one out of every four simulator cycles.

The configurations used by these two simulators are listed in Table.2.

One Linux server with a 3.2GHz Intel Xeon processor is used to run BookSim; multiple flit-injection-rates, from 0.05 to 0.5, are configured respectively to show the simulation performance under different loads.

The speedup (in the right part of Fig.2) is the ratio of the number of cycles simulated per second in FPGA to that in software. We observe that the software's simulation speed decreases with increasing injection rate (in the left part of Fig.4) while the hardware speed is constant. As a result, our simulator achieves greater speed-up at higher packet injection rates: The least is more than 3000 as the scale is 4X4 and the

flit-injection-rate is 0.05; when the flit-injection-rate is set as 0.5, the ratio is about 6000. In addition, as the NoC scale increases, the speed gap will expand further.

- Comparison with other FPGA-based solutions

We have collected some information of existing FPGA-based NoC simulators, including the running performance, the consumed on-chip resources and the NoC scale that can be simulated, and presented them in Table 3. Because the FPGA chips used by these works are different, the comparison is for reference only.

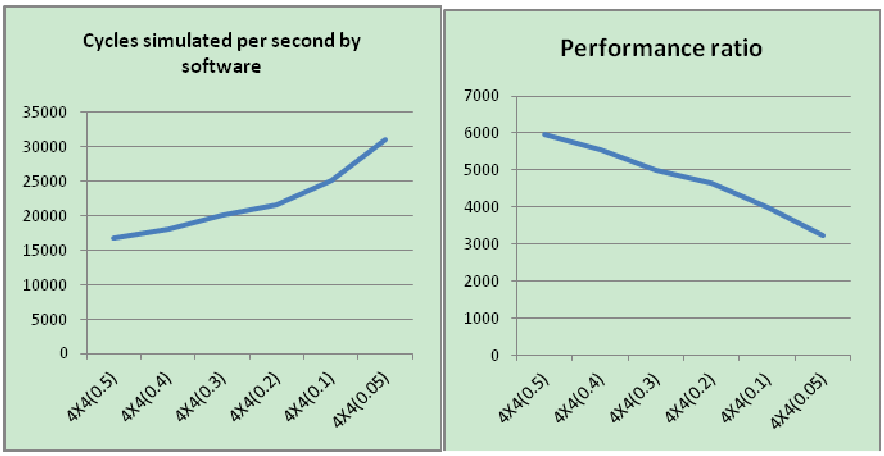

Fig. 2. Performance of the FPGA NoC simulator. In both figures, the X-axis stands for the NoC scale and flit-injection-rates. For example, 4X4(0.5) means there 16 nodes and each node injects a new packet in one out of every two simulator cycles. In the left part, the y-axis stands for the simulation speed measured in cycles per second. In the right part, the y-axis indicates the ratio of the number of cycles simulated per second in FPGA to that in software.

Table 3. Comparison with other FPGA-based simulators

	[5]	[7]	[8]	[11]	**Ours**
Running Frequency(Mhz)	50	70	6.6	50	**100**
On-chip resources (slices)	7387 (Virtex II Pro V20)	16394 (Virtex II Pro (XC2VP30))	N/A (Virtex-II 8000)	13050 (Virtex II Pro (XC2VP30))	**11005**
NoC Scale	6	9	N/A[3]	9[4]	**16**

[3] It supports the time-division multiplexing technology, so that one physical node can simulate multiple target nodes (while the simulation speed will be degraded). For example, if a 4X4 network is simulated, the maximum simulation frequency is 206 kHz.

- CC Examples

For the 4X4 mesh NoC, we simulate some directory-based CC protocols of CMP at the memory-hierarchy level to verify its function. The software is written in C and running on the soft processor on-chip. Details are presented as follows:

- The whole memory space is shared by all cores. The CC feature is kept by the distributed-directory-based hardware, which is the main simulated subject by software.
- No real CPU core is simulated. Instead, read / write memory accesses from the upper-level are created randomly.
- On the memory-hierarchy level, a distributed and shared L2 cache including 16 banks and 16 L1 private caches have been simulated; all connect with each other through the 4X4 mesh. From the system point of view, each core has a L2 bank and a L1 private cache; the former is also the home directory of the corresponding memory-address range assigned to it. All distributed directories construct a global table that keeps track of what memory is held and where.

Now, two variants of the MESI protocol have been implemented:

Case 1: When one cache-line is to be modified, its owner directory will invalidate all copies in L1 caches.

Case 2: When one cache-line has been modified, its owner directory will replace all copies in caches with the new content as well as the copy in the memory.

A fixed sequence of 10000 memory accesses has been simulated in these two cases; the results are presented in Table 4. We can see that the number of elapsed cycles of Case 2 is about 164.5% of that of Case 1, although its L1 cache hit-ratio is 160.9% of Case 1. The reason lies in that, for Case 2, the communication times is much larger, about 229.5% of the other; and the average transfer-latency of a message through the NoC is increased from 3.8 cycles to 4.1.

Table 4. Results of CC Examples

	Case 1	**Case 2**
Simulated cycles	83627	137536
Cache-hit ratio	16.31%	26.24%
Number of NoC communications	44131	101300
Average transfer-latency of a message	3.8 cycles	4.1 cycles

For Case 1, the whole simulation time is about 2.5s. According to the performance model in Section 3.2, if running the CC simulation on the host CPU that interacts with the FPGA, the time spent on the roundtrip communications will be more than 25.1s

[4] DART supports the time-division multiplexing technology, too. Here only the number of physical nodes are given.

(83627 *0.3ms), which is much more than the on-chip version. Therefore, this mode is preferred for such relatively simple simulations.

In addition, if both simulations (CC and the NoC) are completed by software on the host, the estimated running-time is also longer although we take the fastest simulation speed of BookSim (about 31000 cycles per second in Fig.4).

- Trace-driven Example

This is a real usage. To compare NoC designs with different configurations, we have to simulate a few NoC architectures and use the real running trace as the input to judge the better design. The trace is collected through the following way: We use the Pin [16] to instrument the real target process; in the corresponding callback code, we simulate a 16-core CMP's memory hierarchy with the given configurations but no NoC-communication has been simulated. Till now we can get the access-trace for NoC and the collected trace contains about 3,200,000 records.

Then, we use the GB Ethernet cable to transfer records into the chip when a proxy program is running on the soft core to receive data to drive the NoC. Although the transfer latency of the Ethernet is relatively high, its throughout is enough so that the transfer is not the bottleneck.

It has consumed 29,000,000 cycles or so to complete all records, in 127s. In contrast, we use the 100% software solution and the running time is about 710s.

5 Conclusions

We have presented a method that allows multi-core system designers to simulate NoC and other layers with flexibility and fast enough speed. By providing common SW/HW interfaces, the time-consuming and relatively-fixed part, NoC, is implemented by hardware while others are implemented by software. We synthesize this design on the Xilinx's Virtex 5 155T FPGA and some example has been completed to show its availability. In addition, performance analysis is given to show under what circumstance such a design is preferred: if the operation in one simulated cycle is too complex, using the host CPU for software simulation is preferred although the slow HW/SW interaction exists; otherwise, the on-chip mode is better.

Acknowledgement
The work is supported by the High Technology Research and Development Program of China under Grant No. 2013AA01A215. The authors wish to thank Mo Tao, Li Xiaoxiao and Jiang Linhao for their tireless support of the simulation environment.

6 References

1. W.J. Dally, and B. Towles, Route packets, not wires: on-chip interconnection networks. Proceedings of Design Automation Conference (DAC'01), Las Vegas, USA, 2001, pp. 684-689.

2. L. Benini, and G.D. Micheli, Networks on chips: a new SoC paradigm. IEEE Computer, Vol. 35, 2002, pp.70-78.
3. William Dally and Brian Towles. Principles and Practices of Interconnection Networks. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.
4. M. Lis, K.S. Shim, M.H. Cho, P. Ren, O. Khan, and S. Devadas. DARSIM: a Parallel Cycle-Level NoC Simulator. Proceeding of 6th Annual Workshop on Modeling, Benchmarking and Simulation, June 2010.
5. N. Genko, D. Atienza, G. De Micheli, J. Mendias, R. Hermida, and F. Catthoor, A complete network-on-chip emulation framework. Proceeding of Design, Automation and Test in Europe, DATE, Mar 2005.
6. Y. Krasteva, F. Criado, E. de la Torre, and T. Riesgo, A Fast Emulation-Based NoC Prototyping Framework. Proceedings of International Conference on Reconfigurable Computing and FPGAs, Dec 2008.
7. G. Schelle and D. Grunwald, Onchip interconnect exploration for multicore processors utilizing FPGAs. Proceedings of 2nd Workshop on Architecture Research using FPGA Platforms, 2006.
8. P. Wolkotte, P. Holzenspies, and G. Smit. Fast, Accurate and Detailed NoC Simulations. Proceedings of First International Symposium on Networks-on-Chip (NOCS 2007), May 2007.
9. Milo M. K. Martin, Daniel J. Sorin, Bradford M. Beckmann, etc. Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset. ACM SIGARCH Computer Architecture News, Volume 33(2005) 92 - 99.
10. Nikolaos Hardavellas, Stephen Somogyi, Thomas F. Wenisch, Roland E. Wunderlich, Shelley Chen, Jangwoo Kim, Babak Falsafi, James C. Hoe, and Andreas G. Nowatzyk. SimFlex: a fast, accurate, flexible full-system simulation framework for performance evaluation of server architecture. SIGMETRICS Perform. Eval. Rev., 31(4):31–34, 2004.
11. Danyao Wang, Natalie Enright Jerger, J. Gregory Steffan. DART: a programmable architecture for NoC simulation on FPGAs. Proceedings of the Fifth ACM/IEEE International Symposium on Networks-on-Chip, 2011.
12. Eric S. Chung, Michael K. Papamichael, Eriko Nurvitadhi, James C. Hoe, Babak Falsafi, and Ken Mai. ProtoFlex: Towards Scalable, Full-System Multiprocessor Simulations Using FPGAs. ACM Transactions on Reconfigurable Technology and Systems, Volume 2 Issue 2, June 2009.
13. Asanović, K., Patterson D., Tan Z., Waterman A., Avizienis R., & Lee Y. RAMP Gold: An FPGA-based Architecture Simulator for Multiprocessors. Proceedings of Design Automation Conference (DAC-2010).
14. G.X. Liu, G.H. Li, P. Gao, H. Qu, Z.Y. Liu, H.X. Wang, Y.B. Xue, D.S. Wang. Cycle-Accurate 64+Core FPGA-Based Hybrid Simulator. Proceedings of 5th Annual Workshop on Architectural Research Prototyping, 2010.
15. Nan Jiang, George Michelogiannakis, Daniel Becker, etc. BookSim 2.0 User's Guide. Avalaible at https://nocs.stanford.edu/cgi-bin/trac.cgi/raw-attachment/wiki/Resources/BookSim/manual.pdf. 2003.
16. Sion Berkowits. Pin - A Dynamic Binary Instrumentation Tool. Available at http://software.intel.com/en-us/articles/pin-a-dynamic-binary-instrumentation-tool.