Fast Exploration of Parameterized Bus Architecture for Communication-Centric SoC Design

Chulho Shin, Young-Taek Kim, Eui-Young Chung, Kyu-Myung Choi, Jeong-Taek Kong, Soo-Kwan Eo
CAE Center, Samsung Electronics Co., LTD.

Abstract

For successful SoC design, efficient and scalable communication architecture is crucial. Some bus interconnects now provide configurable structures to meet this requirement of an SoC design. Furthermore, bus IP vendors provide software tools that automatically generate RTL codes of a bus once its designer configures it. Configurability, however, imposes more challenges upon designers because complexity involved in optimization increases exponentially as the number of parameters grows. In this paper, we present a novel approach with which effort requirement can be dramatically reduced. An automated optimization tool we developed is used and it exploits a genetic algorithm for fast design exploration. This paper shows that the time for the optimizing task can be reduced by more than 90% when the tool is used and, more significantly the task can be done without an expert's hand while ending up with a better solution.

Index Terms— Platform-based design, Bus Configuration, Optimization, SoC design, genetic algorithm.

I. Introduction

Platform-based design has become one of the trends of system-on-chip (SoC) design as the complexity of design grows. With a billion transistors or a thousand cores expected to be integrated on a single chip by 2010, how to interconnect those components will be a more challenging problem not only because there are many components but also because performance requirements are increasingly demanding. For instance, prevalence of embedded multimedia applications requires frequent accesses to memories and as a result, how to utilize shared memory bandwidth becomes a critical issue. Moreover, such requirements have to be met while maintaining or lowering power consumption.

Momentum is toward establishing standard on-chip interconnects and reusing them for different applications after reconfiguration. AMBA, Sonics SiliconBackplane™ MicroNetwork, LOTTERYBUS, National Semiconductor's GeodeLink™ and IBM CoreConnect™ are examples of the standard on-chip system interconnect [1][2][5][6][7][16].

Some of them are tool-configurable to allow designers to adapt interconnects to a new application while meeting a new set of performance requirements. Designers should perform optimization of the configurable buses in addition to employed IP blocks that are attached to the buses as either a master or a slave to meet all specified requirements. The trend of IP (Intellectual Property) reuse and socketization results in elevated complexity because more IP components are designed with flexible configuration in mind.

Unfortunately, optimally configuring interconnects is a time-consuming task because reconfiguring a bus usually involves repeating manual refinement of RTL codes. Some interconnect vendors now offer tools for automatic generation of the bus and configuration. For instance, the Sonics SiliconBackplane™ parameterizes most aspects of the bus and its arbiter and offers software tools to allow designers to configure the bus, automatically generate the corresponding RTL codes, and simulate the generated RTL codes. (Similar functionality is available for AMBA with a third party tool.) The parameters that can be configured include bus pipeline latency, bus width, the number of masters and arbitration policies. The number of parameters configured in Sonics, however, exceeds a hundred [2].

Configuring a complex system with tens of parameters or more is an inherently difficult and tedious task. Although an expert can speed up the process with intuition and experience, a human being is liable to make mistakes and get stuck in a local optimum. In order to avoid it, a more systematic optimizing approach is needed. To avoid ending up with a local optimum solution, the configuration space needs to be explored with all parameters as candidates for variation. For this reason, this task needs an automated tool. We developed a tool called *ABC* (Automatic Bus Configurator) to address the issue.

Our automated approach reduced the configuration time by more than 90% with no expert's ado at all. Our approach was evaluated on our testbed that employs the Sonics MicroNetwork bus and the MemMax™ memory scheduler as the bus and memory backbones. The work presented in this paper, however, is generic enough to be readily applied to other similar systems or general configuration problems. We plan to extend this tool for optimization of the mapping of multiple hardware IP's to multiple interconnects employed in an SoC design.

This paper is organized as follows: in the next section, we discuss issues concerned with bus configuration. In section III, we describe the core of our work, automated configuration of a system bus. Experimental results are discussed in section IV. Conclusion and future work appear in section V.

Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE'04)
1530-1591/04 $20.00 © 2004 **IEEE**

II. Optimizing System Interconnects

A work [4] on an optimal configuration of a parameterized system was recently performed. In the work, performance and power were the dual objectives of its Pareto-optimization. To reduce the space of design exploration, they used a novel pruning technique that works on an optimal dependency graph of parameters. The technique was attractive because it could significantly reduce complexity while obtaining an exact solution if only an optimal parameter dependency graph had been available. Unfortunately, the limitation of the study is that in a typical system, a dependency graph cannot easily be obtained because very commonly there is no way of intuitively determining optimal dependency between each pair of parameters. Our system is not an exception. In such cases, the only way to determine a dependency graph may be through sensitivity analysis of each pair of parameters; which is an NP complete problem.

Some studies [25][26] were performed to find out an optimal bus configuration of in an SoC design. Those studies, however, did not focus on optimizing a specific interconnect and automating the task. Instead, the main interest was to investigate effects of different topologies. Our study can be applied once a topology is chosen through such studies.

The main interest of our paper lies in how to optimize the communication backbone of a system rather than an entire system. We claim that the best way of optimizing an SoC is to optimize it in three phases. First, a system interconnect is optimized based on specifications with pseudo masters that characterize the behavioral requirements of them. Then, the mapping of multiple IP's to multiple interconnects is to be optimized. (Hardware/software partitioning is also performed in this phase.) Finally, each IP is fine-tuned to fully exploit the optimized interconnect. For more accuracy, these phases may be reiterated until desired analysis result is obtained. In this paper, we limit the scope to optimizing communication architecture of an SoC.

Of various communication topologies, buses are most commonly used as SoC interconnects. A bus is popular because it is area-wise inexpensive and simpler to design compared to other topologies in spite of such disadvantages as inefficient power consumption, bandwidth bottleneck and complex protocol. Among various system buses, we selected Sonics SiliconBackplane™ MicroNetwork as our test vehicle due to its relatively higher acceptance and richer set of configuration features.

With more and more IP's integrated on a single chip, it is likely that the bus interconnect expected to become no longer effective after some critical point. The *networks on chip* might be one of the solutions to replace the bus interconnect [24]. This will, though, impose more burdens in the design of the IP's. Even with the newer interconnect schemes, configuration of them will not be a problem that can rely on the intuition and expertise of a few engineers.

III. Automated configuration of an on-chip Bus

The SoC design of our interest is for a digital TV set-top box application as shown in Figure 1. The system contains multiple masters each of which covers the functions related to audio/video codec, system control, graphics management, graphics/video scaling and enhancement, etc. A master in the diagram actually represents a functional subsystem that may contain one or more processor cores. The master models we use in this study are bus-cycle-accurate and represent worst-case scenarios of the real operations. (Optimally configuring each master is not within the scope of this work.) After obtaining the bus-cycle accurate stimuli through characterization of each master, we performed an investigation to reduce configuration space.

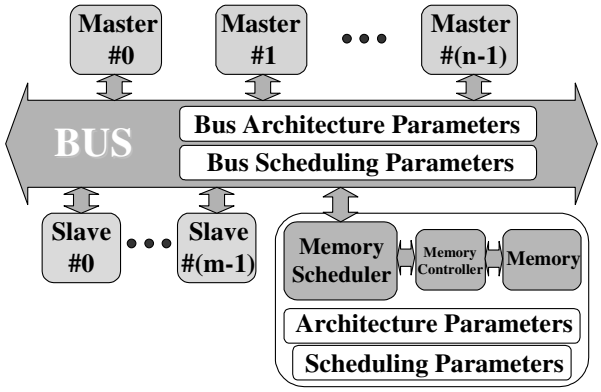


Figure 1 Our system configuration and the parameters involved.

A. *Configuration Space*

As shown in Table 1, exhaustive exploration is not possible even after a significant reduction in the space for exploration. With more than hundred parameters available, optimally configuring a system can be a daunting task even with an automated method because complexity is exponential. Thus, in the beginning, an effort must be made to eliminate parameters from the list of candidates. Examples of these parameters include the ones that make the objective function value monotonically increase or decrease, the ones that do not have significant effect on the function and those that designers would never vary for design reasons. One example is the data bus width; whether the data bus should be 32-bit or 64-bit is not likely to be an issue of configuration.

After examining parameters and eliminating many of them, in our target SoC where five masters and eight threads are employed, we ended up with a configuration space of 2^{45}. Because each simulation in our case took between 1 minute and 5 minutes, it would take at least 67 million years for complete exploration.

Parameter Class	Bus	Bus	Memory Scheduler	Memory Scheduler	Total
	Arbitration	Others	Scheduling	Others	
Parameters	Per-master time slice allocation	Pipeline Latency	Bandwidth allocation per thread, etc.	Prefetch limit, direction change limit, etc.	
Original Exploration space	$1 \sim 6^{256}$	6	2^{136}	2^{26}	$2^{165} \sim 2^{933}$
Reduced Exploration space	847	6	324	2^{26}	2^{45}

Table 1. Even after a significant reduction in the space for exploration, exhaustive exploration is not possible.

B. Basic Flow of Operation

An expert, using a GUI tool (provided by Sonics), can vary configurations of the bus arbitrator and memory scheduler until she/he can find a satisfactory one (Figure 2). With the Sonics bus, once a configuration is determined, an RTL design for the bus, bus arbiter and memory scheduler are generated, compiled and simulated.

C. ABC's Flow of Operation

In ABC, the flow of operation starts with creating a population of individuals for one generation for a genetic algorithm. The genetic algorithm (GA) [14] creates individuals by performing basic operations such as reproduction, crossover and mutation. Once a population is formed, the parallel launcher invokes simulation of individuals concurrently. (See Figure 3.)

For each individual, a configuration file is formed based on the parameters being used. The file is then used for generation of the RTL codes. The RTL model is compiled and simulated to produce simulation result files. The result files are analyzed to compute the value of the fitness function of each individual. A result file is generated for each master and shows how much bandwidth has actually been allocated to it. Each master is modeled as a bus functional model and its simulation is several orders of magnitude faster than that of the RTL counterpart.

D. GA Implementation

In the beginning, we considered simulated annealing as our optimization engine. Simulated annealing (SA) [3][22] is a commonly used combinatorial optimization algorithm that resembles a metallurgical phenomenon. Its algorithm is controlled by three parameters: temperature, equilibrium condition, and cooling schedule, which are related to the total simulation time needed. To address SA's problems such as long simulation time, stochastic evolution (SE) was introduced [11]. SE offered better control over simulation and stopping condition making the exploration faster. However, if parameters are not tuned properly, simulation can often be prematurely aborted ending up with a local optimum.

Like SA, SE and other optimization algorithms, GA has a few parameters that need to be fine-tuned for correct operation: population size [13], the number of generations, crossover rate and mutation rate. GA was selected as our optimization engine because it indeed outperformed other algorithms as shown in section IV. It finds an optimal solution faster and more reliably while SA heavily depends on the quality of the initial configuration and SE often prematurely converges. Another significant reason is its inherent parallelism. Fitness function of individuals that belong to a generation can be obtained fully independently. In other words, parallelism is only limited by the population size. Although SA can be parallelized [12], it needs direct modification or extension of the algorithm. GA does not need any modification except supporting parallel invocation of a fitness function.

E. Table-based bit string formation for GA

Scheduling-related parameters usually have a huge space of

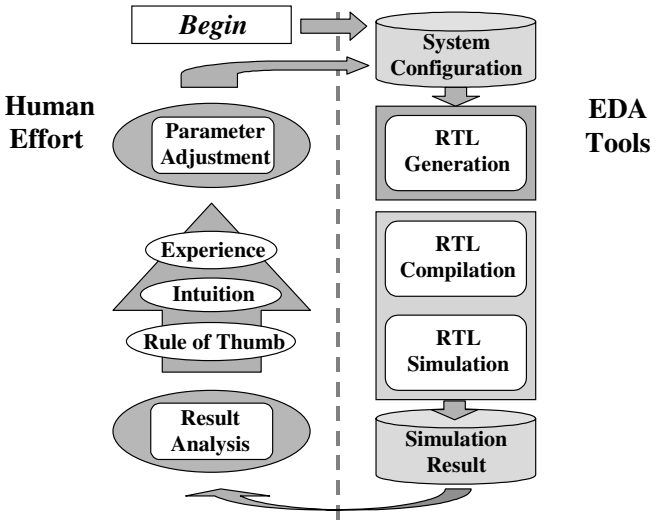


Figure 2 Conventional work flow of bus optimization

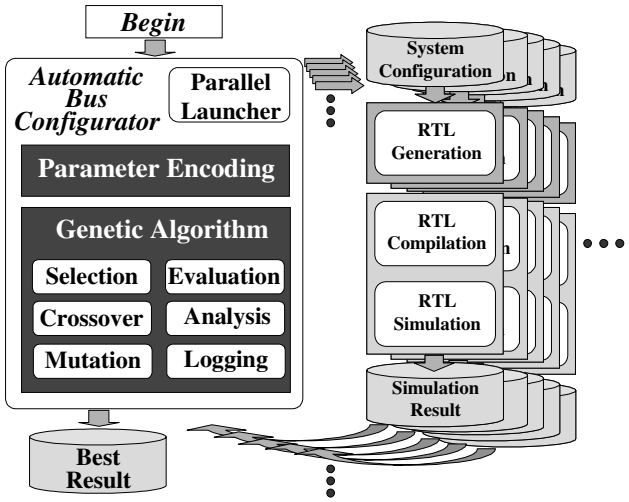
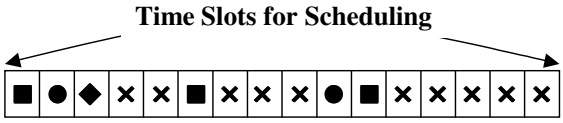


Figure 3 New workflow of bus optimization offered by our automatic bus configurator (*ABC*).

configuration where valid solutions are scarcely found because of the restrictions forced by the software tool. We contrived a way to handle scheduling-related parameters by using a table. Based on the requirements imposed on the masters and guidelines available from the manuals, we found a generic way of forming a table. In this approach, a bit string is formed out of the values of each entry's index (Figure 4). The number of entries for the bus arbitration-related parameters and memory

scheduler's scheduling-related parameters are both less than a thousand. This approach can be used to form bit strings of hard-to-represent problems. Using this approach we were able to reduce the length of a bit string to 45 as shown in Table 1. Even after such significant reduction in exploration space, note that exhaustive exploration is still not possible.

Figure 4 illustrates how a bit string is formed. For example, a master (M0) needs to be guaranteed 20% bandwidth for a fixed period. Of 16 scheduling slots, we allow it to be scheduled from 2 to 4 slots. Though two slots are less than 20%, it is not excluded because unscheduled slots can also be allocated the master if there are no requests. In these experiments, we did not consider permutations for scheduling because we found that the simulation result was not sensitive to the ordering of the schedule.



Master	Symbol	Required Bandwidth	# of Scheduled Slots / Total Slots
M0	■	20%	(2 or 3 or 4) /16
M1	●	15%	(1 or 2 or 3) /16
M2	◆	10%	(0 or 1 or 2) /16
don't care	×	55%	remaining slots

Index	M0–M1–M2	Scheduling
00000	2 – 1 – 0	■●××××××■×××××××
00001	2 – 1 – 1	■●◆××××××■×××××××
00010	2 – 2 – 0	■●××××××■●×××××××
⋮	⋮	⋮ ⋮
11010	4 – 3 – 2	■●◆×■×●×■×◆●■×××

Figure 4 Table-based bit-string formation

F. Objective function

Our objective for optimization is to make each bus master meet its own throughput requirement. To avoid solving a multi-objective optimization problem, we reduced all our objectives to one single function as shown in Figure 5. Here, the smaller the value is, the better the configuration is. Until every master meets its requirement, the focus is on simply bringing all masters within requirement based on their weight. In this phase surplus throughput does not count at all. Once all masters meet requirement, surplus becomes important and exploration proceed toward higher surplus throughput.

We tried another objective function that (Figure 6) focuses on the bus bandwidth and latency. In this case, we used the weight factors to trade off between the bandwidth and quality of service. The weight was also used for normalization. This objective function was introduced based on a request from the designer after delivering the result for the first objective function.

We found that it took about a week or so for an expert to come up with an optimal configuration when manually going through the entire configuration process putting aside the time taken for educating the expert for the tools. In this study, the goal put on our tool was to reduce the time at least to three days because most designers believed more time could not be afforded. With three days as the limit, if an iteration of simulation takes one to ten minutes, thousands of iterations can be completed in the given time. Due to this time limitation, a careful choice of algorithmic parameters had to be made. Eventually, we were able to reduce simulation time dramatically by running simulations in parallel.

Objective Function := $\sum w_i \Delta_i$

where $\Delta_i = R_i - Q_i$ if $R_i - Q_i > 0$

or

$\Delta_i = 0$ if $R_i - Q_i \le 0$

R_i : Required Throughput for Master i.

Q_i : Observed Throughput for Master i.

W_i : Weight for Master i.

Final Fitness Value := $\sum w_i \Delta_i$ if $\sum w_i \Delta_i > 0$

$\sum w_i (Q_i - R_i)$ if $\sum w_i \Delta_i = 0$

Figure 5 Objective Function focusing on the throughput of the individual master

Objective Function := $w_B \Delta + w_L \sigma$

where $\Delta = B_{req} - B_{obv}$ if $B_{req} - B_{obv} > 0$

or

$\Delta = 0$ if $B_{req} - B_{obv} \le 0$

where $\sigma = L_{obv} - L_{cst}$ if $L_{obv} - L_{cst} > 0$

or

$\sigma = 0$ if $L_{obv} - L_{cst} \le 0$

B_{req} : Required Bandwidth

B_{obv} : Observed Bandwidth

L_{cst} : Required Latency

L_{obv} : Observed Latency

W_B : Weight for Bandwidth

W : Weight for Latency

Figure 6 Objective Function focusing on the total bandwidth and latency

IV. Experimental Results

The ABC was used for architecture exploration of a commercial SoC design. As said earlier, it was a chip for digital television set-top box applications. An abstract diagram of the system architecture is shown in Figure 1. Our goal was to reduce design exploration time spent in the bus and memory scheduler configuration to minimum. We achieved our goal as shown in Figure 7. An expert, for about eight days, tried out only 800 configurations in series. The ABC tried more than 2,000 configurations in 8 hours. With 80 machines available, the time could be reduced to 2 hours, achieving 99% reduction in

To determine the algorithm most suitable to our problem, we formed a discrete configuration space where a subset of points from the original space are included. The discrete space consists of 2^{14} configurations. We once obtained values for all points and the objective function was obtained by reading the table. That way, simulation is faster resulting in efficient comparison of the three algorithms. Table 2 shows the results of the comparison.

In Table 2, each is obtained by averaging over 100 different simulation runs in which up to 5,000 iterations are allowed. The number of iterations means how fast the known optimum has been found on the average. If the optimum value were not found, the number of iterations becomes 5,000 and a sub-optimal value would be recorded. From Table 2 the followings can be inferred. The simulated annealing is not good at handling too many parameters and it is not consistent for various initial configurations of parameters. The stochastic evolution tends to make a decision prematurely with a bad local optimum solution. The genetic algorithm's results are not always the best but obviously it is faster and consistent. In this experiment, we were sure that GA was finding good optimal solutions because we had a list of the actual measured values and we knew where the genuine optimal value was located.

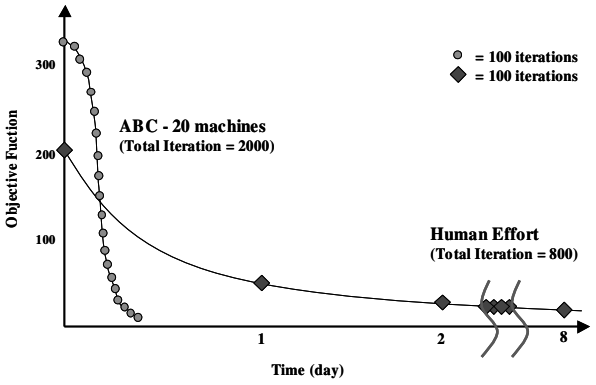


Figure 7 ABC (20 machines) vs. Human Effort

	SA	SE	GA
Objective Func. (avg.)	8.81	8.5	8.21
Iterations (avg.)	4231.6	3663.7	1664.2

Table 2. Comparison of SA, SE, and GA

		Cross-over rate		
		60%	70%	80%
Cross -over Method	1-point	1726.6	1664.2	1764.2
	2-point	2110.6	1762.6	1881.8
	Uniform	2309.8	2313.8	1868.2

Table 3. Effect of crossover method and crossover rate on the average number of iterations

For optimization of our GA approach, we performed simulations to find optimal parameters. One of important parameters is the crossover method. We evaluated three different methods based on how to determine the boundary of a crossover operation: 1-point, 2-point and uniform. It turned out 1-point was optimal for our problem as shown in Table 3. The best crossover rate was 70%. We also evaluated different mutation rates to conclude that the mutation rate of 25% works the best for our problem.

We also conducted simulations to gauge the effect of population size of GA (Figure 8). The number of iterations (equivalent to search speed) varied and the highest speed was observed with the population size of 80. The average objective function values were barely affected by the population size.

With the parallel ABC, we were able to find a solution slightly better than the one found by an expert. The expert took more than eighty hours for the task while our approach took less than eight hours using the load-balanced workstation clusters. The value of this tool is in the following facts:

(1) ABC finds a solution better than the one found by an expert.

(2) ABC finds a solution in less than 1/10 of the time spent by an expert.

(3) ABC can be reused for different system configurations without an expert's help.

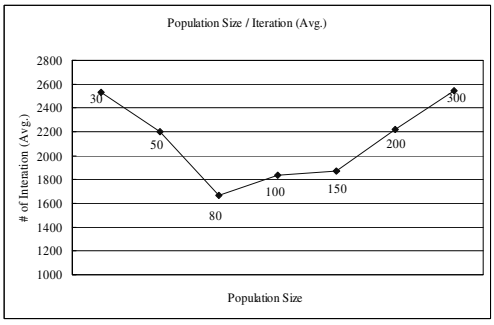


Figure 8 Effect of Population Size in GA

V. Conclusion

In this paper we demonstrated how an optimal configuration of a parameterized on-chip system bus could be found using a software tool we developed. Without the tool, an experienced engineer should undergo a difficult and tedious task of repeating the two phases of work; tweaking the configuration and waiting for a long simulation to end. We found this process could last more than eighty working hours.

We used a genetic algorithm to quickly find an optimal configuration solution. Various population sizes were tested to find out the optimal value. We contrived an efficient way of representing scheduling-related parameters where table indices are used as the bit strings of the genetic algorithm.

In our case study, we corroborated the value of the tool by reducing the optimization time from about eighty hours to eight hours. It should be noted that, during the eight hours, the engineer could do other more valuable works while the s

is running. In fact, exploiting more parallelism could further reduce the time.

As long as IP reuse and socketization remain as the keywords of SoC design approaches, the trends of making IP components configurable will continue and software tools like ABC demonstrated in this paper will be indispensable.

The work presented in this paper is being extended for more general use. The configurator software is being extended for use in a SystemC-based virtual platform environment under development. Once the virtual platform is ready, the tool will be used for configuration of its interconnect. The SoC design we are modeling exploits two identical interconnects and placement of the hardware IP's is a crucial design decision. The virtual platform will be used for fast simulation of various choices and the optimal solution will be found using the extended tool. The indices used in this paper will represent the mapping of a hardware IP to one of the two interconnects.

The virtual platform under development will offer faster simulation speed because the models for hardware IP's and processor cores are transaction-level and the simulation environment will not require slow event-based logic simulation. It will still give accurate results because the models are cycle-accurate. We are planning to support high-level power estimation in this environment as well by adding table-based power metrics in each hardware IP model.

References

[1] K. Lahiri, A. Raghunathan, and G. Lakshminarayana. "LOTTERYBUS: A new high-performance communication architecture for system-on-chip designs," in *Proc. of 38th Design Automation Conference*, 2001.

[2] *SiliconBackplane µNetwork Reference Document Revision 2.3b*, Confidential Document, Sonics, Inc.2002.

[3] K. Choi and S. Levitan, "Exploration of Area and Performance Optimized Datapath Design Using Realistic Cost Metrics", in *Proc. of International Symposium on Circuits and Systems (ISCAS-95)*, 1995

[4] T. Givargis, F. Vahid, J. Henkel. "System-Level Exploration for Pareto-Optimal Configurations in Parameterized System-on-a-Chip," *IEEE Trans. on Very Large Scale Integration Systems (TVLSI)*, vol. 10, no. 4, Aug. 2002.

[5] IBM Microelectronics: CoreConnect™ bus architecture, http://www-3.ibm.com/chips/products/coreconnect/

[6] The GeodeLink™ System Architecture, http://www.national.com/appinfo/solutions/files/geodelink_white_paper.pdf

[7] S. Furber. ARM System Architecture. Addison Wesley, 1996.

[8] J. Peng, S. Abdi and D. Gajski, "Automatic Model Refinement for Fast Architecture Exploration," in *ASP-DAC/VLSI Design,* pp. 1–6, 2002.

[9] M. Hashempour, et. al, "Rapid Design Space Exploration of DSP Applications using Programmable SoC Devices – A Case Study," in *SoC Conference*, 2002.

[10] A. Naeemi, R. Venkatesan, and J. Meindl, "System-on-a-chip Global Interconnect Optimization," in *SoC Conference*, pp. 399-403, 2002.

[11] Y. Saab and V. Rao, "Stochastic Evolution : A Fast Effective Heuristic for Some Generic Layout Problems," in *Proc. of 27th ACM/IEEE Design Automation Conference*, pp. 26-31, 1990.

[12] R. Azencott, *Simulated Annealing : Parallelization Techniques*, Wiley, New York, 1992.

[13] J.T. Alander, "On optimal population size of genetic algorithms," in *Proc. of CompEuro92*, 65-70, IEEE Computer Society Press, 1992.

[14] J. Koza, *Genetic Programming : on the programming of computers by means of natural selection,* Massachusetts Institute of Technology, Namco Ltd.1998.

[15] A. Goel and W. Lee, "Formal verification of an IBM CoreConnect processor local bus arbiter core", in *Proc. of the 37th conference on Design automation*, p.196-200, June, 2000.

[16] ARM, Amba Specification, available from www.arm.com

[17] J. Koza, *Genetic Programming : on the programming of computers by means of natural selection,* Massachusetts Institute of Technology, Namco Ltd.1998.

[18] K. Lahiri, A. Raghunathan, and S. Dey, "Efficient Exploration of the SoC Communication Architecture Design Space," in *Proc. IEEE/ACM Intl. Conf. on Computer Aided Design*, pp.424-430, San Jose, California, November 2000.

[19] K. Jong and W. Spears, "An Analysis of the Interacting Roles of Population Size and Crossover," in *Proc. of the International Workshop Parallel Problem Solving from Nature*, Springer-Verlag, pp. 38-47, 1990

[20] F. Polloni, L. Mazzoni, S. Matteo, "Fast System-Level Design Space Exploration for Low Power Configurable Multimedia Systems-on-Chip," in *SoC Conference*, pp. 150–154, 2002.

[21] A. Brinkmann, et. al, "On-Chip Interconnects for Next Generation System-on-Chips," in *SoC Conference,* pp. 211–215, 2002.

[22] D. Wong, H. Leong, and C. Liu, *Simulated Annealing for VLSI Design*, Kluwer Academic Publishers, Boston, 1988.

[23] G. Harik, E. Cantu-Paz, D. E. Goldberg, and B. L. Miller, "The gambler's ruin problem, genetic algorithms, and the sizing of populations," in *Proc. of the 1997 IEEE International Conference on Evolutionary Computation*, pp. 7--12, 1997.

[24] L. Benini and G. De Micheli, "Networks on chips: a new SoC paradigm", *IEEE Computer*, volume 35, pp. 70--78, January 2002.

[25] K.Lahiri, A.Raghunathan, S.Dey, "System-Level Performance Analysis for Designing On-Chip Communication Architectures", *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no.6, pp.768-783, June 2001.

[26] K. Ryu and V. Mooney, "Automated Bus Generation for Multiprocessor SoC Design," *Proceedings of the Design Automation and Test in Europe Conference (DATE'03)*, pp. 282-287, March 2003