A System-Level Modeling Methodology for Performance-Driven Component Selection in Multicore Architectures

Ankur Agarwal, Georgiana L. Hamza-Lup, and Taghi M. Khoshgoftaar

***Abstract*—System complexity, driven by both increasing transistor count and customer need for more and more savvy applications, has increased so dramatically that system design and integration can no longer be an after-thought. With this increasing complexity of the system design, it has become very critical to enhance system design productivity to meet the time-to-market demands and reduce product development cost. Real-time embedded system designers are facing extreme challenges in the underlying architectural design selection. This involves the selection of a programmable, concurrent, heterogeneous multiprocessor architecture platform. Such a multiprocessor-system-on-chip platform has set innovative trends for the real-time systems and system-on-chip designers. The consequences of this trend imply a shift in concern from computation and sequential algorithms to modeling concurrency, synchronization, and communication in every aspect of hardware and software co-design and development. Therefore, there is a need for a high level methodology that can provide a complete system modeling and architecture/component selection platform. The proposed six-step system modeling methodology provides a process for performance driven component selection, to avoid costly iterative system design re-spins, thereby enhancing system design productivity. We demonstrate our methodology by applying it onto a network-on-chip architecture for selecting its components given certain system specification and system-level performance requirements.**

***Index Terms*—Component based design, component selection, concurrency modeling, embedded systems, performance evaluation, system level design, system modeling.**

I. Introduction

INTEGRATION of a system comprising of hundreds and thousands of components, cores with adequate memory, and communication backbone on a single silicon die is feasible but highly complex [1]. There are about 50 million transistors on a chip designed with 250 nm technology and about 400 million transistors on a chip with 90 nm technology. In such a scenario, prototyping a complete chip with the traditional *Application Specific Integrated Design* flow will take much longer, negatively impacting the system design productivity. Therefore, in order to minimize the loss in productivity we must address concerns at a higher level of abstraction—the "system architect's level."

Traditionally, in a system modeling stage, a system architect with expertise and background in all the related engineering disciplines uses a spread-sheet type of tool to conduct 'what-if' scenarios and decide on various components and technologies. Due to the increase in the system complexity, this role of the system architect is becoming more complex [2] and virtually impossible, unless of course certain system engineering principles are applied and supported *a priori*. System design architects and engineers have conceptualized several paradigms, platforms, and design methodologies that aim at enhancing system engineering productivity. Time to market is one of the leading factors among the system performance, cost, and reliability for a system design. In order to reduce the time and cost of a system design, designers now integrate predesigned components that optimally combine software (SW) and hardware (HW) to achieve certain functional and performance requirements [3], [4]. System-on-chip (SoC) [5] and multiprocessor-system-on-chip (MPSoC) [6] are examples of technologies which integrate several predesigned, developed, and verified IP blocks into a single chip. Libraries of IP blocks are available with components with varying performance parameters, such as silicon area, power consumption, and latency. A system architect must analyze various system level performance tradeoffs to select the architecture that satisfies the given system requirements and specifications.

The advancements in technologies related to IP block design have led to the development of several embedded component-based design (ECBD) methodologies. In such ECBD, the communication is separated from computation. System communication is usually described at a higher level of abstraction which is independent of the functionality of the system. This is often achieved with the help of either interfacing the standard bus protocols, as in the IBM CoreConnect [7], advanced microcontroller bus architecture [8], or by using a standard component (such as virtual socket interface alliance [9] and open core protocol-intellectual property [10]). These protocols provide an abstracted platform that grants the plug and play capability to a component being integrated into a SoC or MPSoC platform. Currently, several commercial tools are available that allow a system designer to integrate component(s) or generate interfacing protocols for these components; such as N2C from CoWare, virtual component codesign (VCC)

Manuscript received November 19, 2010; revised October 13, 2011; accepted October 13, 2011. Date of publication May 7, 2012; date of current version May 22, 2012.

The authors are with the Department of Computer and Electrical Engineering and Computer Science, Florida Atlantic University, Boca Raton, FL 33431 USA (e-mail: ankur@cse.fau.edu; ghamza@cse.fau.edu; taghi@cse.fau.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JSYST.2011.2173614

from Cadence, and Polaris from Synopsis. In several tools, the system designers emphasize the integration of SW-HW components that may be at SystemC level, such as in case of VCC. However, the integration on the entire product at a SystemC level would take a long time to complete, thereby introducing delay in product launch. For this process to be more effective system integration has to be addressed at a higher level where quick, intelligent, and informed design decisions may be taken. Lower level issues (RTL/source code), while appropriate at implementation level, would slow down the system tradeoff analysis.

Platform-based design is an example of another well-established concept to reduce time and cost and increase system design productivity. In his work [11], Vincentelli discussed the issues and challenges related to increasing system design complexity and provided an insight into the various alternatives for taming system complexity. It is a well-known fact that reusing platforms and components in a system are an effective way to managing system complexity. Platforms such as OMAP have contributed significantly to reducing the design efforts in the domain of embedded devices, such as cell phones. In such cases, the HW platform is abstracted in such a way that application specific SW sees a standard application programming interface or a SW platform [12]. These platform-based designs are also referred to as "meet in the middle" design philosophies, where the component library is abstracted from a bottom-up design flow and is integrated into the top-level abstracted platform design.

It is expected that future systems will exhibit an increasing need for design automation, reusability, and componentization, thus increasing the role of the electronic design automation (EDA) industry. In such a scenario, a system-level modeling framework for component selection should be developed that essentially supports the middle-out design philosophy to exploit reuse to the maximum in order to reduce the design effort and cut down the overall system design cycle [13].

In this paper, we propose a high level (system architect's level) modeling framework for component selection. The framework will select from a library of components, the components that integrated will satisfy given system requirements and performance characteristics. Such a framework would not distinguish between HW and SW components but would rather integrate both into the system in almost the same way.

The remainder of this paper is organized as follows. Section II discusses related work. In Section III, we present our proposed methodology for system modeling and component selection. In Section IV, we present and discuss the results of applying our methodology for selecting the components of a network-on-chip (NoC) architecture. Finally, Section V concludes this paper.

II. Related Work

Several methodologies have been introduced to address the system architect's level challenges. As the methodologies matured they evolved into EDA tools used to enhance overall system design productivity. In this section, we discuss some of these tools and methodologies.

The VCC environment by Cadence, based on SystemC, is a programming paradigm which is coupled with a clear separation among architecture, functionality, and timing. This environment introduced the concept of component reuse at a higher level of abstraction in the form of an EDA tool and allowed architectural exploration. However, executing an entire system with SystemC slows down the system analysis and architecture selection process. Similarly Ptolemy, a system modeling environment introduced by the University of California, Berkeley, provides a SW framework for modeling, simulation, and design of concurrent, real-time, embedded systems with focus on assembly concurrent components [14]. Ptolemy provides the foundation for system architectural exploration, but does not provide support for high level concurrency analysis.

Other design flows such as ROSES, N2C, and CoWare provide an integrated flow for component based design interfacing, by generating all the component interfaces and integrating them together [15]. While these environments focus their efforts on the refinement of communication channels, our proposed methodology takes a different perspective, focusing mainly on system performance evaluation and selection of the system architecture before developing a system. The modeling environment for SW and HW (MESH) developed by Carnegie Melon University, Pittsburgh, PA, is an example of another excellent system architecture mapping environment for embedded system applications [16]. MESH exploits the principle of frequency interleaving and logically groups system resources and hence enhances the overall system performance. The environment does not provide a methodology for analyzing the concurrency failures or an integrated framework for performance comparison. However, MESH allows one to analyze the impact of various components on the overall system [17].

In our methodology, the architecture selection is based on component selection algorithms designed for achieving optimized system performance. In addition, our methodology provides an integrated framework for an architect to analyze "what-if" scenarios before developing the system in order to avoid system re-spin. Furthermore, the concurrency issues are managed at a high level to avoid system synchronization issues and allow the full exploitation of multiprocessing and multithreaded system behavior.

A. System Concurrency Modeling

System designers are able to keep pace with the Moore's law by exploiting both SW and HW concurrency in a system [18]. However, concurrency issues are hard to manage and a systematic process of identifying and resolving all concurrency issues must be adopted to avoid deadlocks or livelocks. SW developers have realized the need for concurrency and have started using multithreaded programming models for embedded system designs. Multithreaded JAVA and pthreads are two examples of concurrency constructs at the SW level. However, the main issue is to be able to analyze whether after following all the steps for designing a concurrent system, the new system design still has any concurrency concerns.

Concurrency issues have been addressed through the use of various models of computation (MoC) [19], such as

communicating sequential processes (CSP) [20], pi-calculus [21], lambda calculus [22], and finite state machines (FSM) [23]. CSP, pi-calculus, and lambda-calculus offer an effective mechanism of specification and verification [24]. However, they are based on mathematical models and are not only hard to understand for a SW/HW designer but also hard to apply in practice. FSM have been extensively used in the past, but do not scale well with increased system size.

Finite state processes (FSP) and labeled transition system analyzer (LTSA) provide a framework for modeling concurrency and analyzing it exhaustively [25]. Using FSP a system designer will not have to analyze the system specification in the form of a mathematical model in order to expose concurrency issues in the system. A simplified concurrency model can be developed in FSP and concurrency issues can be analyzed with LTSA graphically. LTSA also provides an exhaustive analysis capability to uncover synchronization issues such as deadlocks and livelocks. FSP and LTSA have been widely used in SW verification of several large subsystems such as NASA's Ames K9 Rover [26]. This methodology was used by adopting a divide-and-conquer approach for carrying out compositional verification of their autonomous system development, throughout its lifecycle.

B. Component Selection

The component selection problem has been extensively researched in both the SW and the HW domains. The main challenge in component selection is that there is no standard way of describing components and available component specifications are usually heterogeneous and incomplete. Several researchers, from both the SW [27] and the HW [28] domain, attempted to alleviate this problem by proposing various formal and semiformal component specification techniques that could facilitate the automation of the component selection process.

Other research focuses on interpreting the existing component descriptions using semantics and domain knowledge. Web-based component portals like eCots, SourceForge, and ComponentSource [29], containing information about a wide range of vendor solutions, can be used when searching for components. However, they are limited to a keyword search which is inefficient since the meaning of what is searched for is ignored. In addition, they suffer from the fact that domain knowledge is constantly changing and thus it can become quite expensive to keep up to date the ontologies these portals rely upon.

Another group of research studies focus on ranking and classifying components within libraries to speed up the selection process by performing a more structured exploration of the search space. In [30], components are ranked based on a suitability value computed as a weighted sum from a set of evaluation attributes. Criticism of the weight sum method includes: the summing of differing types of data (e.g., cost plus reliability), lack of a clear process for determining the attribute weights, and the inherent problem of the formula losing dependency information between attributes. In [31], artificial intelligent classifiers are built from data generated from an ideal component specification and then used to classify components. This approach captures dependencies between component attributes, overcoming some of the limitations of existing aggregation-based approaches. In [28], similarity analysis is used to rank components based on how much their functionality overlaps with the required system functionality.

All of the previously mentioned approaches for component selection are mainly driven by system requirements (such as system functionality, reliability, cost, and delivery time) that depend directly on the attributes of the individual components. For example, in [32] the authors assumed that every component satisfies one or more functional requirements and the goal is to select the minimal set of components that together satisfy a set of given. In [33], the cost of the final system is minimized. In [34], the allowable cost is fixed while functionality is the one that needs to be maximized. In [35], the authors formulated component selection as a multiobjective optimization problem solved with evolutionary algorithms, the objectives being to minimize unsatisfied system requirements, cost, and number of components used. In [36], reliability and delivery time requirements are specified in addition to cost requirements, which again depend directly on the attributes of the system components.

In contrast to existing approaches, our proposed component selection methodology is driven by non-functional system requirements whose dependencies on the component attributes cannot be expressed by exact mathematical formulas. For example, system performance cannot be expressed in terms of the performance of individual components, because it cannot be measured for individual components in isolation but only for the integrated system. Therefore, our goal is to extract and approximate these dependencies, so that we can predict with certain accuracy the characteristics of a system, from the attributes of the individual components.

III. Proposed Methodology

Developing a system from reusable components involves searching through a component library that contains components with various functional and performance characteristics. While various researchers have provided their views on describing a component, it is important to understand that a component must be defined by keeping the HW and SW specifications in mind. This paper discusses a methodology for embedded applications and therefore the components themselves can be embedded components as opposed to SW or HW components. In such a scenario, we define a component as having the following five characteristics: 1) *Methods*; 2) *Parameters*; 3) *Attributes*; 4) *Interfaces*; and 5) *Costs*. Each of this characteristics provides an important property for an embedded component to be reusable. *Methods* represent the functions of a component. For instance an input buffer may have two main functions, of storing data and forwarding data. *Attributes* represent the input(s) and output(s) of a component. For example, the attributes of a buffer are the data ports for data to enter and exit the buffer. *Parameters* represent possible customizations for components, which can impact the quality of service (QoS) and performance of the overall system. For instance, buffer size and scheduling criteria can

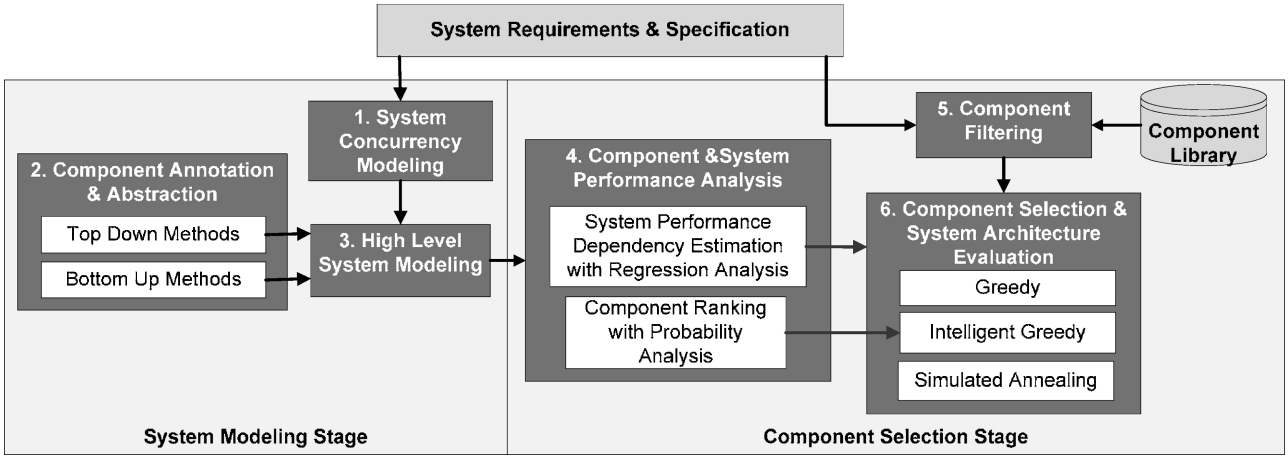

Fig. 1. Architect level framework for system modeling and component selection.

be regarded as important parameters of a buffer. *Interfaces* are the concurrent constructs that must be incorporated in a component in order to make it concurrency compliant. This will also ensure an easy integration of a component in the overall system. For instance, reading from an empty buffer can result in an overall system failure. Similarly, there can be a concurrency failure when a buffer is forwarding the data while another component may be trying to store the data in the same buffer. *Costs* are a set of QoS and performance parameters for a component. Possible costs are area, power consumption, or latency.

The proposed methodology consists of two stages: 1) system modeling, and 2) component selection. In the system modeling phase the overall system and subsystem architecture is defined and a set of candidate components are identified. High level system modeling provides architects with a rapid and effective way of measuring the performance of a system, without implementing it. Higher productivity is achieved since the down-streams error-prone and time-intensive iterative prototyping is avoided. In the component selection phase, optimal components are selected using several strategies and the final system architecture is chosen. The challenge in selecting the right component is that there are variants of each component with the same methods, attributes and interfaces, but different parameters and performance characteristics. Moreover, some performance characteristics cannot be measured for individual components in isolation but only when the components are integrated into a system. Thus, system modeling is extremely helpful to understand the impact of a component parameter on the overall system performance. Fig. 1 shows the proposed architect level methodology for system modeling and component selection. System requirements and specification are the input into the framework. The requirements/specification phase is beyond the scope of this paper.

A. System Modeling Stage

The system modeling stage includes three steps: system concurrency modeling (Step 1), component annotation and abstraction (Step 2), and high level system modeling (Step 3).

Step 1) *System Concurrency Modeling:* As the first step of this methodology, a high level abstract concurrency model of all the processes identified from system requirements and specification is developed. A system may be more prone to intermittent failures, if concurrency concerns are not addressed properly. Integration of predesigned reusable blocks may fail if these blocks execute in parallel, share resources, and/or interact with each other. Concurrency issues may lead the system into a deadlock or a livelock state. These issues, if not addressed, may be detrimental to normal functioning of the system. Multiprocessor architectures, recently introduced to extend the applicability of the Moore's law, depend upon concurrency and synchronization in both SW and HW to achieve their goal. Traditional system design integration and verification approaches will not be cost-effective in exposing concurrency failures as they are intermittent: such failures can significantly increase time-to-market and field failures. Therefore, one would have to develop abstract concurrency models and do exhaustive analysis on these models to test for concurrency problems.

There are several concurrency modeling approaches as discussed in Section II-A. In our proposed methodology we chose a concurrency modeling approach that uses the FSP modeling language and the LTSA environment. At system requirements/specifications level, we identify concurrency scenarios and model them using a concurrency modeling tool. This step also identifies and uses existing components from a component library; if no existing component exists, concurrency model development will help discover the new component that can be added to the library. Our proposed concurrency modeling approach is detailed in [37] and [38].

Step 2) *Component Annotation and Abstraction:* System modeling, analysis, and design will not be an effective solution until we have a mechanism for abstracting the parameter values from different applications and components of a system, and for representing these values in our model. A wrong representation of the performance of a subsystem or component may provide wrong estimates of the overall system performance, thereby increasing the product development time and cost rather than reducing them. Therefore, it is extremely

important to accurately estimate (annotate and abstract) the performance of subsystem(s) and its component(s). As illustrated in Fig. 1, in the proposed methodology we have as input into the high level system modeling block, component annotation and abstraction methods. There are several methods that can be used for annotating and abstracting system performance, such as literature survey, using SW emulators, SW estimation methods, extraction of HW-SW performance through rapid prototyping and subsystem modeling. The performance metrics obtained with these methods are incorporated in the component library of the modeling environment. This will facilitate high level "what-if" design tradeoffs at the architect's level. We have detailed our work on annotation and abstraction methods in [2]. In our proposed methodology, we used two of the above-mentioned methods: literature survey and subsystem prototyping.

Step 3) *High Level System Modeling:* The third step of our proposed methodology is system level modeling. We define here the five main issues that should be met for a simulation environment to be suitable for system level designs. The simulation model must allow a system architect to: 1) model the system functionality well in advance of building the actual computing system; 2) model concurrency issues among various components in the system model; 3) manipulate an abstract set of design elements simultaneously to generate different sets of QoS and performance parameters, and to fine-tune the system model; 4) integrate different models of computations (MoCs) into the modeling environment; and 5) access a well-defined library of components defined in various MoCs so that the modeling time can be substantially reduced.

A few system-level modeling tools were investigated and the details of their comparison were discussed in [39]. MLDesigner (MLD) and Ptolemy are robust system-level modeling environments suitable for embedded and other applications. Both tools satisfy the five previously specified requirements for a simulation environment. MLD is a commercial modeling environment with more integrated design libraries, it integrates other environments, has optimized memory requirements as compared to Ptolemy and most importantly, it has strong customer support. Furthermore, MLD is used for various applications, from wireless applications to embedded system design. Hence, it is not limited to a particular target application or domain. For all these reasons, MLD was the system-level design tool chosen for our methodology. MLD is a system-level modeling environment that can model various MoCs and has a well-designed library for each MoC. Therefore, it allows one to model a system at an abstract level. We can do performance analysis on this abstracted model. Such a model at the design phase will allow one to make key decisions, such as the number of processors, HW/SW partitioning, the estimated performance values for new components, and the use of existing components in SW or HW. Such design decisions have the potential to significantly enhance the productivity of system design.

One can further abstract the performance parameters for any given application on a particular architecture and capture the resource requirements in terms of processor speed, cache size, power consumption, latency, and silicon area. Since the performance analysis is being done at an abstract level, it is therefore optimized in running the simulation faster as compared to other modeling environments, which are C, C++, or SystemC-based. Further details about MLD and its contribution to the abstract system modeling domain are discussed in [39]–[41].

B. Component Selection Stage

The component selection stage involves searching through the space of possible combinations of components for those combinations that satisfy system requirements and specifications. This stage comprises of three steps: component and system performance analysis (Step 4), component filtering (Step 5), and component selection and system architecture evaluation (Step 6).

Step 4) *Component and System Performance Analysis:* As illustrated in Fig. 1, two statistical analyses were performed on the measurements resulting from the system modeling stage: 1) regression analysis was used to estimate the dependency of the system performance on the parameters of its constituent components; and 2) probability analysis was used to rank components based on their probability of satisfying certain system performance requirements. The goal was to automate the selection process by finding which variant of each component and which combinations of components best meet system requirements and specifications.

Regression is a mathematical measure of the average relationship between a response variable and one or more input variables. In our regression analysis [42], the response variable was system performance and the input variables were the parameters of the components of the system. Regression equations for each performance parameter were computed. These equations will allow us to estimate how well a certain combination of components satisfies given system requirements and therefore they will be used in the automated component selection process.

Probability analysis was used to estimate the likelihood of a component to satisfy certain system performance requirements when integrated into a system. Specifically, as detailed in [43], system performance was categorized into several classes and Bayes classifiers were trained and tested, using measurements from the system modeling step, to predict the system performance class for any given component. Then the probability of each component falling into a certain system performance class was computed and components were ranked based on this probability.

Step 5) *Component Filtering:* The component selection process begins by filtering the component library based on system specification. This is the fifth step of our methodology and it simply reduces the search space by eliminating components that do not satisfy given system constraints.

Step 6) *Component Selection and System Architecture Evaluation:* In the sixth step of our proposed methodology a selection algorithm is chosen to explore the reduced search space for possible solutions. Finding the optimal combination of components requires an exhaustive approach in which all potential combinations are evaluated. However, due to the large size of the search space, such an exhaustive approach

is usually unfeasible, as it has a very high computational cost [44]. Greedy algorithms can be used for approximation as they have lower computational complexity [32]. Artificial intelligent approaches are also good alternatives if the goal is to find an acceptably good solution in a fixed amount of time, rather than the optimal solution.

Three selection algorithms have been implemented and compared: two are greedy based [43] and [45] and the third one is a simulated annealing approach [46]. Both greedy approaches involve choosing at each step the component that "seems" (at that point) to be the most helpful in achieving the objectives, i.e., the given system requirements. If at any step we detect that the objectives (system requirements) can no longer be satisfied based on the choices of components made so far, backtracking is required in order to allow us to follow a different path in our search space. Backtracking also enables finding more than one solution. Different heuristics can be used in determining the "best" component to be chosen at each step. In our first greedy approach [45] we assume, based on previous observations, that components whose attributes have lower values are better. In our second greedy approach [43], the probabilities calculated in Step 4 are used for ranking components. Specifically, the components with the highest probabilities of satisfying system requirements are considered the best. The simulated annealing approach [46] starts its search for solutions from a random combination of components and then moves through the search space by jumping to better combinations until a solution is found.

Each of the three component selection algorithms produces a possible combination of components. The performance of each proposed combination of components is then estimated based on the regression equations computed in Step 4 and compared against the given system requirements. Last, if the proposed architecture passes validation it is recommended to the user as a solution. Otherwise, the process loops back to select and evaluate a new combination of components.

IV. Results and Discussion

A. Application Domain

To evaluate our framework we have chosen NoC as our application domain. An NoC is a multicore architecture that provides a communication infrastructure for different resources. Each component in a NoC architecture is characterized by different parameters. Data producers are characterized by packet injection rate (that is, the rate at which they produce packets to be delivered through the NoC), which can vary from 0.1 to 1.0 at equal increments of 0.1. Buffers, used for storing and forwarding data packets, are characterized by size, such as 1, 2, 3, ..., 10, and scheduling criteria, such as priority based (PB), priority based round robin (PBRR), round robin (RR), and first come first served (FCFS). Network switches, connecting resources to buffers, consist of a router and a scheduler, and are thus characterized by a routing algorithm, such as X-direction first, Y-direction first, and XY-random and scheduling criteria, such as PB, PBRR, RR, and FCFS. More details on the NoC components can be found in [39], [40], and [47]. Thus, there are many possible combinations for compiling a NoC architecture and depending on the parameters of the components integrated, the resulting system will exhibit different performance.

```
Compiled: BUFFER, Compiled: SCHEDULER, Compiled:
PRODUCER
Composition: FINAL1 = b1:BUFFER || b2:BUFFER ||
s1:SCHEDULER || p1:PRODUCER
State Space:  24 * 24 * 9 * 2 = 2 ** 15
Composing...
-- States: 351 Transitions: 882 Memory used: 3043K
Composed in 110ms
FINAL1 minimising.....Minimised States: 351 in 46ms
No deadlocks/errors. Progress Check...
-- States: 351 Transitions: 882 Memory used: 3397K
No progress violations detected. Progress Check in: 31ms
```

Fig. 2. Simulation results for the abstracted NoC model.

The main performance measure for NoC architectures is network latency, which is the time taken by a data packet to travel from one resource to another. We assume there are three types of data packets injected into the network: high, medium, and low priority packets, and there are different latency requirements for each type of packet. High priority packets are control signals such as Read, Write, Acknowledge, and interrupts; mid priority are the real-time traffic, while low priority are the nonreal time block transfers of data packets. We defined control signals as a high priority because they must reach there destination in time to manage the communication flow of the network. Equally important is to ensure the delivery of real-time data in real-time bounds and therefore, it was assigned mid priority. The rest of the data on the network belongs to the low priority class. Our component selection problem now becomes selecting the components for a 4×4 mesh-based NoC architecture that needs to satisfy certain latency, area and power consumption requirements and certain system constraints, for example, on the buffer size. Buffer size is a key parameter in a NoC architecture as buffers occupy about 60% of the area of the communication backbone.

B. Results and Discussion

Step 1) *System Concurrency Modeling:* As discussed in Section III-A1, we have used FSP and LTSA for modeling and analyzing concurrency issues. We can realize from the model implementation that there are one producer process p1, two buffer processes b1 and b2, and one scheduler process s1. The final composition process includes all above processes. These processes have been exhaustively tested. Fig. 2 shows the result for the exhaustive simulation. It can be seen that there are 351 different possible states and 882 possible transitions among these states in the system. All these states are not involved in any deadlocks and livelocks. The details of the concurrency modeling methodology and its results with respect to the NoC architecture can be found in [37], [38] and [48].

a) *Component equivalency and definition:* Our components were identified from a high level concurrency model specification as: Producer (P), Consumer (C), Input Buffer (IB), Output Buffer (OB), and Network Switch (NS) which consist of Scheduler (S) and Router (R). Table I shows the five characteristics of each of the above components. The

TABLE I
IDENTIFIED COMPONENTS AND THEIR CHARACTERISTICS

Comp.	Methods	Parameters	Attributes	Interfaces
P	produceData()	Injection Rate Distribution Pattern	dataOut	buffAvail
C	consumeData()	Consumption Rate	dataIn	buffAvail
IB OB	storeData() forwardData()	Buffer Size	dataIn dataOut counterValue	dataInBuffer sendData
R	computeRoute() getSourceAddress getDestAddress	Routing Criteria	informOutputPort dataIn dataOut	routerReady routerBusy
NS	forwardData() operateSwitch()	Switching Criteria	getData sendData	swithAvail
S	checkOutputAvail() selectOutputPort()	Scheduler Criteria	requestAck requestNAck	requestGrant confirm reqBuffAvail

details can be found in [40]. In our NoC model, a producer class corresponds to a resource and resource network interface; our input buffer corresponds to input buffer, buffer scheduler, virtual channel, and virtual channel allocator; scheduler corresponds to switch allocator; and router corresponds to the router of the NoC architecture.

Step 2) *Component Annotation and Abstraction: Literature survey* was used to extract leakage power consumption values. Vellanki *et al.* from Arizona State University, Tempe, provided results for leakage power consumption for NAND and XOR gates for 0.18 μm technology (extracted from SPICE netlist). We used these values to approximately estimate the total leakage power consumption in our NoC implementation.

b) *FPGA based subsystem prototyping:* An FPGA Xilinx-Vertex 4 platform was used to calculate the area (gate count) and leakage power consumption for various components with various parameters as shown in Table II. Note that there is no linear relationship between buffer size and gate count mainly due to two reasons: 1) the scheduling criteria of the buffer does not depend on its size, and 2) the gate count depends upon the number of configurable logic blocks (CLBs) used in an FPGA [39]. Leakage current in an FPGA may change by a factor of three based on the input to a circuit, and it depends upon the utilization factor of CLBs. At 25 °C, an FPGA consumes 4.2 μW per CLB. It is important to understand that leakage power consumption is a function of inputs test vector to the system.

Step 3) *High Level System Modeling:* A system model will have to integrate various MoCs to correctly model interactions among various components and with the system itself. Different MoCs have evolved to represent the reasoning in different areas of focus. Similarly, in the case of NoC, various MoCs must be well integrated to represent the overall system functionality. Recall that a NoC consists of several synchronous regions, however all these regions communicate between each other asynchronously. A synchronous local region of an NoC might require one or more such MoCs to co-exist and interact. A detailed discussion on NoC modeling with multiple MoC is discussed in [49] and [50].

TABLE II
GATE COUNT AND LEAKAGE POWER FOR VARIOUS NOC COMPONENTS

Components	Parameters		Gate Count	Leakage Power (W)
Buffer	Buffer Size	1	11 427	0.0020202
		2	13 381	0.0022932
		3	14 510	0.0024948
		4	14 920	0.0025242
		5	15 526	0.0025704
		10	16 108	0.0028182
Router	Scheduling Criteria	FCFS	3155	0.0008442
		RR	3674	0.0011508
		PB	3554	0.001218
		PBRR	5954	0.0014952
	Routing Algorithm	X-Y	9589	0.0032676

TABLE III
LATENCY FOR HIGH, MID AND LOW PRIORITY PACKETS

Scheduling Criteria (S)	Buffer Size (B)	Injection Rate (I)	Latency High Priority (L1)	Latency Mid Priority (L2)	Latency Low Priority (L3)
1	2	0.1	24	24	24
2	2	0.1	23.1	22.87	24
3	2	0.1	22.02	24	24
4	2	0.1	24	24	24
1	5	0.1	54	54	54
2	5	0.1	27.33	26.52	63.57
3	5	0.1	24.39	26.58	62.04
4	5	0.1	21.88	22.1	63.73

In our modeled NoC, we have used MLDesigner, which supports modeling in different domains such as the discrete event (DE), synchronous data flow (SDF), FSM, and continuous time (CT) domains, among others. All the components were modeled with SDF, FSM, and CT models. The subsystem regions of NoC were modeled with SDF only and the top level simulation was executed with DE. It is quick to analyze a system with DE MoC since it allows us to models all the signals at the change in their value thus executing the system simulation faster. The developed NoC model further supports component customization, provided through drop-down menus. This allows a design architect to change the actual system model to understand the impact of various design parameters without the need for changing the actual design.

In our proposed methodology, system modeling was used to measure three system performance parameters: latency for high, mid and low priority packets, for a limited number of combinations of NoC components. Specifically, we took measurements for the following component parameters: injection rate at three levels: 0.1, 0.5, and 1.0; buffer sizes at three levels 2, 5, and 10; scheduling criteria at four levels: FCFS-level 1, RR-level 2, PB-level 3, and PBRR-level 4. Table III shows a snapshot of our measurements.

Step 4) *Component and System Performance Analysis:* From the measurements obtained through system modeling two analyses were performed: 1) nonlinear regression analysis

TABLE IV
SNAPSHOT OF INPUT DATA AND L1 ESTIMATION AND RESIDUALS SQUARED FOR EACH OF THE FOUR MODELS EVALUATED

Input data						Model 1		Model 2		Model 3		Model 4	
S	B	I	L1	L2	L3	Predicted L1	Residuals^2	Predicted L1	Residuals^2	Predicted L1	Residuals^2	Predicted L1	Residuals^2
1	2	0.1	24	24	24	33.62	92.48	24.64	0.41	50.47	700.87	41.50	306.17
1	2	0.1	24.9	24.9	24.9	33.62	75.98	24.64	0.07	50.47	654.03	41.50	275.48
2	2	0.1	23.1	22.87	24	14.40	75.76	23.99	0.79	19.95	9.93	29.54	41.46
2	2	0.1	23.7	23.3	24.6	14.40	86.56	23.99	0.08	19.95	14.07	29.54	34.09
3	2	0.1	22.02	24	24	13.74	68.52	23.33	1.72	7.99	196.85	17.58	19.72
3	2	0.1	22	24	24.5	13.74	68.19	23.33	1.77	7.99	196.29	17.58	19.54

TABLE V
LATENCY CLASSES

Latency Values	Latency Class	Latency Values	Latency Class
10..19	1	40..49	4
20..29	2	50..59	5
30..39	3	60..69	6

was used to extract the dependencies of the system performance on the component parameters; this allowed us to estimate how well a selected combination of components satisfies given system performance requirements; and 2) probability analysis was used to extract the probability of one component to satisfy certain system performance requirements; this allowed us to rank components for the component selection process.

c) *System performance dependency estimation based on regression analysis:* Regression analysis was used to extract from the measurements obtained through system modeling an estimate of the dependency of the system performance in terms of latency for high, mid and low priority packets, (L1), (L2), and (L3) respectively, on the parameters of its constituent components: injection rate (I), buffer size (B) and scheduling criteria (S). Note that in order to simulate the noise inherent in a real experiment we introduced a noise function in our simulation model, uniformly distributed with 5% allowable error. We have also taken the simulation results twice to replicate a real scenario. A snapshot of the input data for our regression analysis is represented by the first six columns of Table IV.

Since the true functional relationships between system performance (response variable) and component parameters (input variables) were unknown, four different models were analyzed and the one with the least error was selected. These models were: 1) quadratic model, includes main effects + interaction + square terms: S, B, I, S*B, B*I, S*I, S^2, B^2, I^2; 2) model with main effects + interactions: S, B, I, S*B, B*I, S*I; 3) model with main effects + squared terms: S, B, I, S^2, B^2, I^2; and 4) model with main effects only: S, B, I. The details of the regression analysis can be found in [42]. Table IV shows the results for response variable L1. Columns 7, 9, 11, and 13 show the predicted values for L1 for each of the four models, and columns 8, 10, 12, and 14 show the corresponding residuals squared. The mean squared errors calculated for each of the four models are 48.51, 97.03, 118.89, and 167.40, respectively, indicating as expected that the quadratic model is the best. The regression equations obtained for all three response variables were as follows:

$$L1 = 49.639 - 40.96 * S + 10.529 * B + 8.30 * I - 3.03 * S * B + 0.06 * B * I - 0.438 * S * I + 9.28 * S^2 - 0.03 * B^2 - 1.63 * I^2$$

$$L2 = 49.06 - 41.434 * S + 10.58 * B + 10.43 * I - 2.97 * S * B + 0.004 * B * I - 1.24 * S * I + 9.49 * S^2 - 0.03 * B^2 - 0.86 * I^2$$

$$L3 = 68.82 - 54.23 * S + 2.29 * B - 1.77 * I + 2.65 * S * B + 0.11 * B * I + 1.49 * S * I + 9.63 * S^2 + 0.04 * B^2 + 5.66 * I^2.$$

d) *Component ranking with probability analysis:* Component ranking uses the measurements obtained from the system modeling stage to extract the probability of a given component to satisfy certain system performance requirements. For this purpose, system performance was categorized into several latency classes as shown in Table V. We have used XLMiner for our analysis and computed the conditional probabilities for each component parameter and for each latency class [43]. Table VI shows a subset of these probabilities for latency classes 1 and 2, for high priority packets. For example, if a requirement is that the latency for high priority data packets must be less than 30 ns (meaning it should fall into latency class 2 or lower), then there is a higher probability of achieving this if we select a buffer of size 1. If this is not an option, due to other system constraints, then the next best probability is obtained for a buffer of size 2, followed by a buffer of size 4, 5, or 10. Similarly, in terms of scheduling criteria, we have the best probability 0.35 + 0.28 = 0.63 of satisfying this requirement, if we select the scheduling criteria number 4 (PBRR). The next best choice is scheduling criteria number 2 (RR), with a probability of 0.25 + 0.28 = 0.53, followed by criteria number 3 (PB), with a probability of 0.33 + 0.14 = 0.47. Using these conditional probabilities we can rank components to facilitate the component selection process.

Step 5) *Component Filtering:* For evaluating the proposed component selection framework we have developed a SW prototype that allows a system architect to input desired system requirements, in terms of acceptable latencies for three types of data packets injected into the network, and

TABLE VI
CONDITIONAL PROBABILITIES FOR LATENCY CLASSES 1 AND 2

Input Variables	Latency Class 1		Latency Class 2	
	Value	Probability	Value	Probability
Injection Rate	0.1	0.5714286	0.1	0.2307692
	0.5	0.4285714	0.5	0.2820513
	1	0	1	0.4871795
Buffer Size	1	1	1	0.1025641
	2	0	2	0.2051282
	3	0	3	0.1538462
	4	0	4	0.1794872
	5	0	5	0.1794872
	10	0	10	0.1794872
Scheduling Criteria	1	0.2857143	1	0.0512821
	2	0.2857143	2	0.2564103
	3	0.1428571	3	0.3333333
	4	0.2857143	4	0.3589744

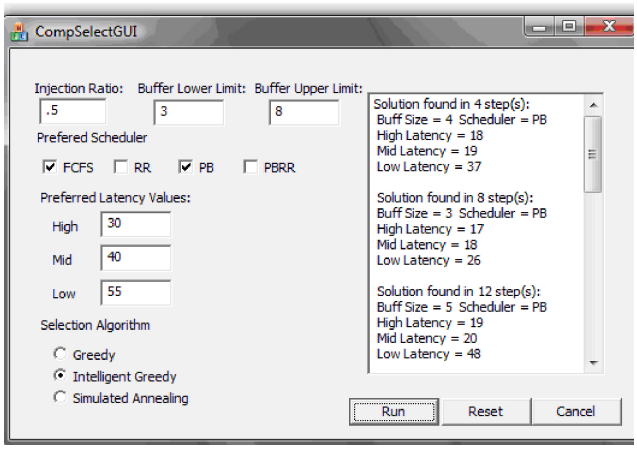

Fig. 3. Integrated framework prototype-user interface.

system constraints, in terms of injection ratio for resources and restrictions on the buffer size and scheduling criteria. Fig. 3 shows the user interface of this prototype. Three assumptions were made: 1) all the 160 buffers of the 4×4 mesh based NoC architecture have the same size; 2) all buffers and switches have the same scheduling algorithm; and 3) packet injection ratio is the same for high, mid and low priority packets. Thus, our problem becomes selecting a buffer size and a scheduling criteria for the NoC architecture such that to satisfy given system latency requirements and constraints.

In the component filtering step all components are evaluated against the given system constraints and those components not satisfying these constraints are eliminated. In all tested scenarios, the following system constraints were used in the component filtering step: 1) injection ratio was fixed to 0.5; 2) buffer size was restricted to the range 3–8; and 3) all scheduling criteria were acceptable.

Step 6) *Component Selection and System Architecture Evaluation:* We have executed component selection for several scenarios. For each scenario, different system performance requirements were given. Scenario 1, the most restricted in terms of system performance requirements, involved latency values from category 1 (10 ns to 20 ns) for high priority packets. Specifically, the latency values for this scenario were 19 ns, 29 ns, and 39 ns for high, mid, and low priority packets, respectively. For each subsequent scenario, the system performance requirements were relaxed and the latency values for all three priorities were increased by 10, until latency values from category 5 were reached. At this point, little to no change in the performance of the algorithms was observed. Each scenario was executed using each of the three component selection algorithms. In the cases of both greedy and intelligent greedy, only one execution was required for each scenario, due to the fact that both algorithms are deterministic. The simulated annealing algorithm, being nondeterministic, was executed ten times. The average performance over these ten runs was taken.

TABLE VII
BEST FIVE SOLUTIONS WITH LOWEST ESTIMATED LATENCY VALUES

Solution (Best to Worst)	Buffer Size	Scheduling Criteria	Estimated Latency		
			High	Mid	Low
Solution 1	3	PB	17	18	26
Solution 2	4	PB	18	19	37
Solution 3	5	PB	19	20	48
Solution 4	3	RR	21	21	24
Solution 5	4	RR	25	26	32

For each scenario the first five solutions found by each algorithm were recorded and the number of combinations evaluated before reaching each solution was plotted. We refer to this number as "steps" in finding a solution. For scenario 1, only two solutions were found, thus we only plotted the results obtained for each of the remaining four scenarios in Figs. 4, 6, 8, and 10, respectively. Then the best five solutions (that is, the solutions with the lowest latencies) were identified. These best five solutions and their corresponding estimated latencies are presented in Table VII. The "steps" taken by each algorithm until reaching each of these five best solutions is plotted for each scenario (except scenario 1) in Figs. 5, 7, 9, and 11, respectively.

For scenario 2 (latency category 2), Fig. 4 illustrates that both greedy and simulated annealing can find the first three solutions equally fast, but simulated annealing takes slightly longer to find the fourth and fifth solutions. Fig. 5, however, shows that the greedy methods perform the best when searching for best solutions. This is explained by the inner nature of the greedy approaches to search for the best solutions, while simulated annealing is looking into finding any solution fast. Also, both figures illustrate that intelligent greedy is slightly faster than the other greedy approach.

In scenario 3 (latency category 3), all algorithms performed at nearly the same level, when looking at how fast each algorithm found solutions (Fig. 6). Fig. 7 shows that the greedy approaches find the best solution in the lowest number of steps and simulated annealing exhibits a rather constant performance amongst all five best solutions.

In the last two scenarios, a decline in the performance of the intelligent greedy in finding the best solution was noted, going from the second best method to last in scenario 5. Intelligent greedy was able to find some solutions more quickly than others, especially in scenario 5, but constantly

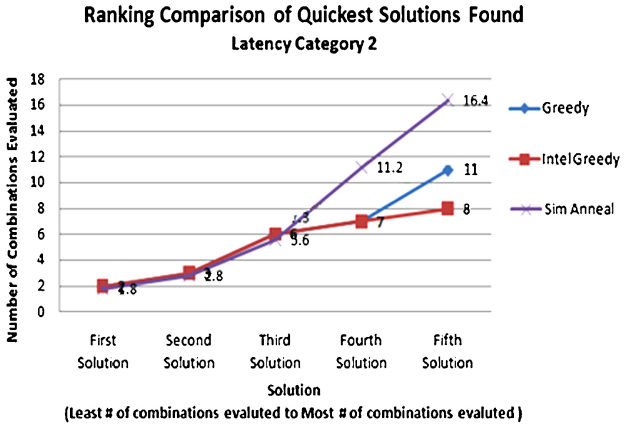

Fig. 4. Quickest solutions for latency category 2.

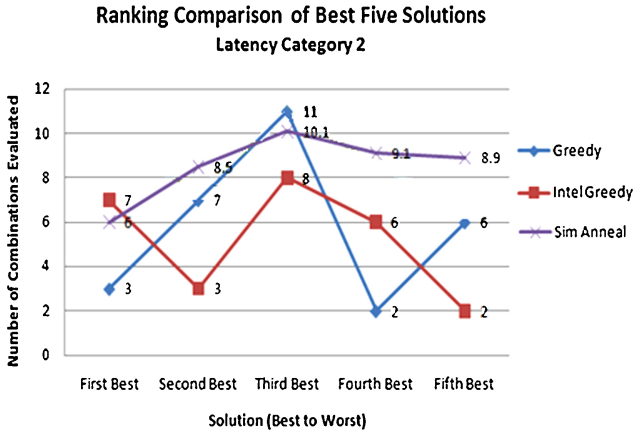

Fig. 5. Best solutions for latency category 2.

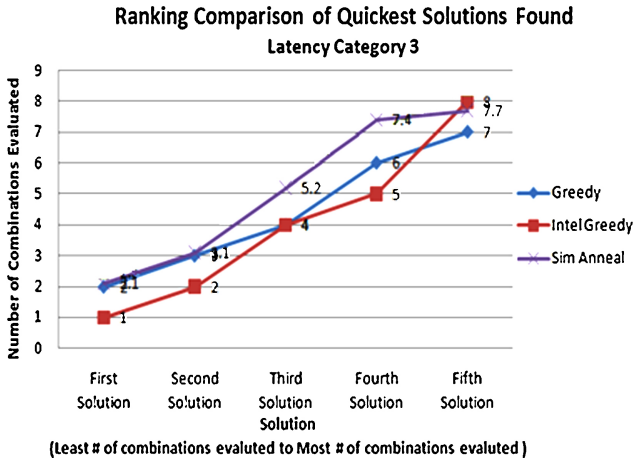

Fig. 6. Quickest solutions for latency category 3.

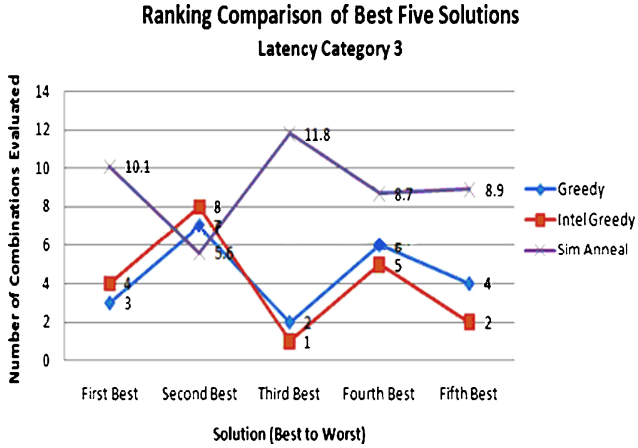

Fig. 7. Best solutions for latency category 3.

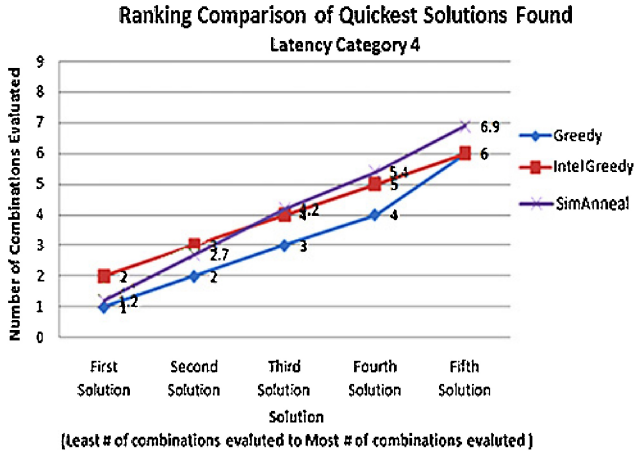

Fig. 8. Quickest solutions for latency category 4.

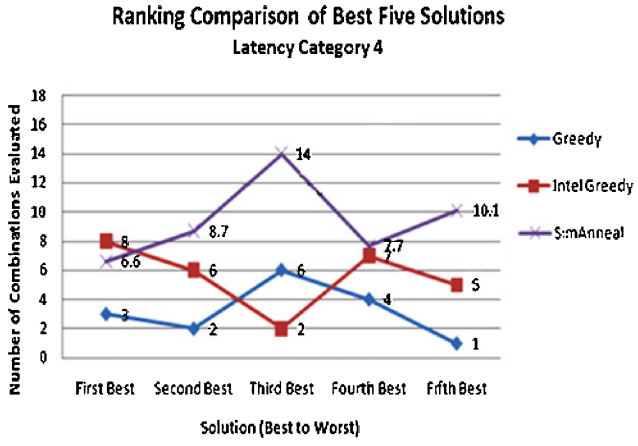

Fig. 9. Best solutions for latency category 4.

missed the best solutions by a considerable margin. The relative performance of the algorithms in terms of finding some solutions became almost identical in the last scenario. This is explained by the fact that, as the system performance requirements are relaxed more combinations of components become solutions and therefore, all algorithms are able to find solutions faster and the differences between the algorithms become unnoticeable.

The above figures illustrate that overall the three component selection algorithms had similar performance in terms of finding solutions fast, but the greedy approaches performed slightly better than simulated annealing in finding the best solutions. Intelligent greedy performed slightly better than the standard greedy implementation when finding the best solution, but fell short when system requirements were relaxed. The good performance of the standard greedy approach comes

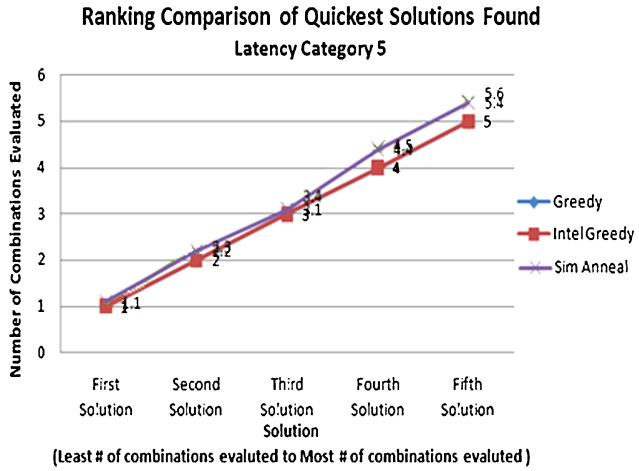

Fig. 10. Quickest solutions for latency category 5.

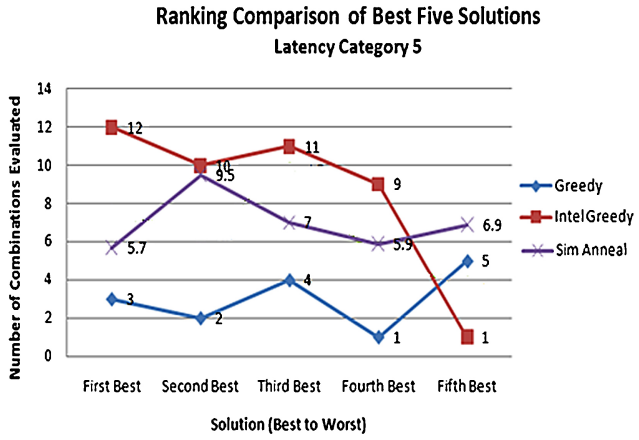

Fig. 11. Best solutions for latency category 5.

from the fact that it operated under the assumption that the lower the value of a component parameter the better that component is. While this assumption is clearly not true in general, it is a good approximation for our particular application domain, and thus it caused the greedy approach to perform unexpectedly well in our scenarios.

V. Conclusion

A framework that allows a system architect to effectively explore and provide performance driven recommendations for selecting the overall system architecture and all its components can be very useful and a powerful concept in system engineering domain. In this paper, we presented one such complete methodology for "system modeling and component selection." This methodology may be adopted for use in other multiprocessor/multicore architectures in order to manage complexity at a higher level of abstraction. The overall methodology was demonstrated by realistically modeling and optimally designing the communication backbone of a NoC. The developed NoC model is abstract enough to facilitate quick analysis. In the proposed methodology, we discussed an approach for integrating various MoCs to realistically model a system architecture such that the design complexity can be well managed and design issues can be addressed at different levels of abstraction. The methodology further discusses the approach for developing reusable performance driven components that can be abstracted using various annotation methods. This helps an architect make more informed decisions and avoid down-stream integration delays and unit cost run-ups due to the use of expensive, albeit unnecessary, components.

References

[1] C. Neeb, M. Thul, and N. Andwehn, "Network-on-chip-centric approach to interleaving in high throughput channel decoders," in *Proc. IEEE ISCAS*, May 2005, pp. 1766–1769.

[2] R. Shankar *et al.*, "Annotation methods and application abstraction," in *Proc. IEEE Int. Conf. Portable Inform. Devices*, May 2007, pp. 1–4.

[3] A. Jantsch and H. Tenhunen, "Will networks on chip close the productivity gap?" in *Networks on Chip*, A. Jantsch and H. Tenhunen, Eds. Dordrecht, The Netherlands: Kluwer, 2003, pp. 3–18.

[4] G. Karsai, F. Massacci, L. J. Osterweil, and I. Schieferdecker, "Evolving embedded systems," *IEEE J. Comput.*, vol. 43, no. 5, pp. 34–40, May 2010.

[5] L. E. M. Brackenbury, L. A. Plana, and J. Pepper, "System-on-chip design and implementation," *IEEE Trans. Educ.*, vol. 53, no. 2, pp. 272–281, May 2010.

[6] M. D. Nava, P. Blouet, P. Teninge, M. Coppola, T. Ben-Ismail, S. Picchiottino, and R. Wilson, "An open platform for developing multiprocessor SoCs," *IEEE Trans. Comput.*, vol. 38, no. 7, pp. 60–67, 2005.

[7] IBM. *32-Bit Processor Local Bus Architecture Specifications*, Ver. 2.9 [Online]. Available: http://www-3.ibm.com/chips/products/coreconnect/

[8] ARM. (1999). *Advanced Microcontroller Bus Architecture Specification* [Online]. Available: http://www.arm.com/armtech/AMBA_spec

[9] *Virtual Socket Interface Alliance* [Online]. Available: http://www.vsi.org

[10] Open-Core Protocol Int. Partnership Association Inc. (2001). *Open-Core Protocol Specification*, Release 1.0 [Online]. Available: http://www.ocpip.org

[11] A. S. Vincentelli, "Quo vadis, SLD? Reasoning about the trends and challenges of system level design," *Proc. IEEE*, vol. 95, no. 3, pp. 467–506, Mar. 2007.

[12] F. Wagner, W. Cesario, L. Carro, and A. Jerraya, "Strategies for integration of hardware and software IP components in embedded systems-on-chip," *Integr. VLSI J.*, vol. 37, no. 4, pp. 223–252, Sep. 2004.

[13] M. G. Dixit, P. Dasgupta, and S. Ramesh, "Taming the component timing: A CBD methodology for real-time embedded systems," in *Proc. IEEE Eur. Conf. DATE*, 2010, pp. 1649–1652.

[14] J. Eker *et al.*, "Taming heterogeneity: The Ptolemy approach," *Proc. IEEE*, vol. 9, no. 1, pp. 127–144, Jan. 2003.

[15] A. A. Jerraya and W. Wolf, *Multiprocessor Systems-on-Chip*. San Francisco, CA: Morgan Kaufmann, 2005.

[16] A. Cassidy, J. Paul, and D. Thomas, "Layered, multithreaded, high-level performance design," in *Proc. IEEE Int. Conf. Des. Autom. Test Eur.*, 2003, pp. 954–959.

[17] J. Paul and D. Thomas, "A layered, codesign virtual machine approach to modeling computer systems," in *Proc. IEEE Int. Conf. Des. Autom. Test Eur.*, Mar. 2002, pp. 522–528.

[18] W. Hwu, T. G. Mattson, and K. Keutzer, "The concurrency challenge," *IEEE Trans. Des. Test Comput.*, vol. 25, no. 4, pp. 312–320, Jul.–Aug. 2008.

[19] J. Burch, R. Passerone, and A. L. Sangiovanni-Vincentelli, "Overcoming heterophobia: Modeling concurrency in heterogeneous systems," in *Proc. IEEE Conf. Applicat. Concurrency Syst. Des.*, 2001, pp. 13–32.

[20] G. H. Hilderink, "Graphical modeling language for specifying concurrency based on CSP," *IEE Proc. Softw. Eng.*, vol. 150, no. 2, pp. 108–120, 2003.

[21] Chrobot, "Modelling communication in distributed systems," in *Proc. IEEE Int. Conf. Parallel Compon. Electr. Eng.*, 2002, pp. 55–60.

[22] T. Murphy, K. Crary, R. Harper, and F. Pfenning, "A symmetric modal lambda calculus for distributed computing," in *Proc. Annu. IEEE Symp. Logic Comput. Sci.*, Jul. 2004, pp. 286–295.

[23] A. Girault, B. Lee, and E. A. Lee, "Hierarchical finite state machines with multiple concurrency models," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 18, no. 6, pp. 742–760, Jun. 1999.

[24] M. Barrio and P. Fuente, "A formal model of concurrency for distributed object-oriented systems," in *Proc. IEEE Int. Comput. Sci. Conf. Softw. Eng.*, Dec. 1997, pp. 466–474.
[25] J. Magee and J. Kramer, *Concurrency State Models and Java Programs*. West Sussex, U.K.: Wiley, 1999.
[26] D. Giannakopoulou, C. Pasareanu, M. Lowry, and R. Washington, "Lifecycle verification of the NASA Ames K9 rover executive," in *Proc. ICAPS Workshop Validation Verification Planning Scheduling Syst.*, 2005, pp. 75–85.
[27] E. Mancebo and A. Andrews, "A strategy for selecting multiple components," in *Proc. ACM Symp. Appl. Comp.*, 2005, pp. 1505–1510.
[28] C. Trummer *et al.*, "Search for extended IP-XACT components in a library for power aware SoC design based on requirements similarity," *IEEE Syst. J.*, 2010.
[29] J. C. Mielnik, "Using eCots portal for sharing information about software products and the internet and in corporate intranets," in *Proc. Int. Conf. COTS-Based Softw. Syst.*, 2004, pp. 1–4.
[30] J. Guo, B. Zhang, K. Gao, H. Zhu, and Y. Liu, "A method of component selection within component based software development process," in *Proc. Int. Conf. Softw. Process*, 2004, pp. 1–3.
[31] V. Maxville, J. Armarego, and C. P. Lam, "Intelligent component selection," in *Proc. 28th Annu. Int. Comput. Softw. Applicat. Conf.*, vol. 1. 2004, pp. 244–249.
[32] M. R. Fox, D. C. Brogan, and P. F. Reynolds, "Approximating component selection," in *Proc. Winter Simul. Conf.*, vol. 1. 2004, pp. 429–434.
[33] N. Haghpana *et al.*, "Approximation algorithms for software component selection problem," in *Proc. Asia-Pacific Softw. Eng. Conf.*, 2007, pp. 159–166.
[34] P. Baker, M. Harman, K. Steinhofel, and A. Skaliotis, "Search based approaches to component selection and prioritization for the next release problem," in *Proc. 22nd IEEE Int. Conf. Softw. Maintenance*, Sep. 2006, pp. 176–185.
[35] A. Vescan, C. Grosan, and H. F. Pop, "Evolutionary algorithms for the component selection problem," in *Proc. 19th Int. Conf. Database Expert Syst. Applicat.*, 2008, pp. 509–513.
[36] P. Potena, "Composition and tradeoff of non-functional attributes in software systems: Research directions," in *Proc. 6th Joint Meeting Eur. Softw. Eng. Conf. ACM SIGSOFT Symp. Found. Softw. Eng.*, 2007, pp. 583–586.
[37] A. Agarwal and R. Shankar, "Modeling concurrency in NoC for embedded systems," in *Proc. IEEE Conf. High Performance Comput.*, Sep. 2006.
[38] A. Agarwal and R. Shankar, "A concurrency model for network-on-chip design methodology," *J. Modeling Simul.*, vol. 29, no. 3, pp. 238–247, 2009.
[39] A. Agarwal, C. Iskander, R. Shankar, and G. Hamza-Lup, "System level modeling environment: MLdesigner," in *Proc. 2nd Annu. IEEE Syst. Conf.*, Apr. 2008, pp. 1–7.
[40] A. Agarwal, "System level modeling of a network-on-chip," *Int. J. Comp. Sci. Security*, vol. 3, no. 3, pp. 154–174, 2009.
[41] A. Agarwal, H. Kalva, C. Iskander, and R. Shankar, "System level modeling in NoC based H.264 decoder," in *Proc. 2nd Annu. IEEE Syst. Conf.*, Apr. 2008, pp. 1–7.
[42] V. Gupta, A. Mazouz, A. Agarwal, and G. Hamza-Lup, "Statistical analysis for system component selection," in *Proc. 4th Annu. IEEE Int. Syst. Conf.*, Apr. 2011, pp. 1–6.
[43] G. Hamza-Lup, A. Agarwal, R. Shankar, and C. Iskandar, "Component selection strategies based on system requirements' dependencies on component attributes," in *Proc. 2nd IEEE Int. Syst. Conf.*, Apr. 2008, pp. 1–5.
[44] R. G. Bartholet, D. C. Brogan, and P. F. Reynolds, Jr., "The computational complexity of component selection in simulation reuse," in *Proc. Winter Simul. Conf.*, 2005, pp. 2472–2481.
[45] A. Agarwal, G. Hamza-Lup, R. Shankar, and J. Ansley, "An integrated methodology for QoS driven reusable component design and component selection," in *Proc. 1st IEEE Int. Syst. Conf.*, Apr. 2007, pp. 1–7.
[46] C. Calvert, G. Hamza-Lup, and A. Agarwal, "An integrated component selection framework for system level design," in *Proc. 4th Annu. IEEE Int. Syst. Conf.*, Apr. 2011, pp. 261–266.
[47] L. Benini and D. Bertozzi, "Network-on-chip architectures and design methods," *IEE Proc. Comput. Digit. Technol.*, vol. 152, no. 2, pp. 261–271, Mar. 2005.
[48] A. Agarwal, R. Shankar, and F. Kowalski, "Modeling concurrency on NoC architecture with symbolic language: FSP," in *Proc. IEEE Int. Conf. Symbolic Methods Applicat. Circuit Des.*, 2006.
[49] A. Agarwal and R. Shankar, "A layered architecture for NoC design methodology," in *Proc. Int. Conf. Parallel Distributed Comput. Syst.*, Nov. 2005, pp. 659–666.
[50] A. Jantsch and I. Sander, "Models of computation and languages for embedded system design," *Comput. Digit. Tech. IEE Proc.*, vol. 152, no. 2, pp. 114–129, Mar. 2005.

Ankur Agarwal is currently an Assistant Professor with the Department of Computer and Electrical Engineering and Computer Science, Florida Atlantic University (FAU), Boca Raton. He serves as an Assistant Director for the Mobile Technology Consortium and the Center for Systems Integration, FAU. His main research interests include system level design, embedded systems, very large scale integration design, system modeling, and analysis.

Georgiana L. Hamza-Lup is currently a Visiting Assistant Professor with the Department of Computer and Electrical Engineering and Computer Science, Florida Atlantic University, Boca Raton. She has published extensively in the areas of intelligent transportation systems, decision support systems, system modeling, and architectural optimization.

Taghi M. Khoshgoftaar is currently a Professor with the Department of Computer and Electrical Engineering and Computer Science, Florida Atlantic University, Boca Raton, and the Director of the Empirical Software Engineering and Data Mining and Machine Learning Laboratories. His current research interests include software engineering, data mining, and machine learning. He has published more than 450 refereed papers in these areas.