The Lambda Chart: A Model of Design Abstraction and Exploration at System-Level

Falko Guderian and Gerhard Fettweis
Vodafone Chair Mobile Communications Systems
Technische Universität Dresden, 01062 Dresden, Germany
Email:{falko.guderian, fettweis}@ifn.et.tu-dresden.de

***Abstract*—In recent years, chip design complexity is further increasing through multi-processor system-on-chip built up from macro blocks. System-level design promises to close the growing productivity gap between hardware and software design but more sophisticated design models are needed. Therefore, we developed a new model of system-level design abstraction. Therein, three views divide the design space to reduce design complexity. A unified design process with five steps has been defined for the design views. Furthermore, we show that the model is able to formalize system-level design exploration. Finally, exploration has been considered as walk through the design views connected via inter-view links. With the proposed system-level model, the authors provide conceptual foundations to more holistic design and introduce formalization of design exploration.**

***Keywords*-system; design; abstraction; exploration;**

I. Introduction

As integration technology continues to shrink and processor clock frequency stagnates, design complexity grows through the transition from traditional system-on-chip (SoC) towards multi-processor system-on-chip (MPSoC) built up from macro blocks. The increasing complexity motivates to use modeling and simulation languages, such as SystemC, which allow for simulation of the complete design at system-level including the hardware and software. We believe that more sophisticated models are needed to close the growing productivity gap between hardware and software forecasted by the International Technology Roadmap for Semiconductors [1]. Existing models of system-level design require the application and architecture description as starting point and apply exploration in similar design views and process steps. Therefore, we developed a model of system-level design abstraction to bring them more into line. Our model defines three design views, called administration, computation and communication, wherein a unified design process is followed. We introduce the administration view due to the growing importance of management, e.g., of scheduling, power consumption, reliability etc., in future MPSoCs and Many-Core systems. The separation into three views and the reuse of the design process aims at reducing design complexity. Furthermore, system-level design exploration tools become more important but separation and integration of design views and design process steps during exploration is hardly supported in current approaches. Hence, relationships between views and steps should be considered. We show how our model is able to formalize system-level exploration. More specifically, exploration is defined as walk through the design views via inter-view links representing the relationships. During exploration, each view applies one or more design process steps. Our approach is independent on the application domain and the formalization aims at automated exploration. Moreover, we derive three exploration types (classes) from our model and present their usage based on design examples.

In this work, we describe system-level design abstraction and exploration with regard to MPSoC design. Nevertheless, the authors see a general relevance of the model for complex systems based on networks. For example, computer networks, telecommunication networks and sensor networks are also composed of computation, communication and administration. Hence, these systems could be designed and explored using our model.

In the remainder of the paper, Section II provides an overview of the model of system-level design abstraction. In Section III, we compare our model with existing work. Then, Section IV describes the system-level design views. The system-level design process is explained in Section V. Section VI presents different types of design exploration based on system-level design abstraction. Finally, Section VII concludes our work.

II. Overview of System-Level Design Abstraction

In Figure 1, the tripartite representation called λ-chart is our model to realize system-level design abstraction. We use three axes to describe the design views: administration, computation and communication. Along each of the axes, we unified the design process to five steps which are given as concentric bands. The process starts with modeling and partitioning, e.g., of application, architecture and administrative algorithm. Provisioning describes the selection and dimensioning of system components, e.g., cores and processors respectively. As depicted in Figure 1, we separate scheduling and allocation. Finally, the system is validated to decide for an additional design iteration. Hence, the axial loop indicates the iterative design process within each design view.

The example in Figure 2 illustrates the relationship between λ-chart and Y-chart which was introduced by Gajski and Kuhn [2] and refined by Walker and Thomas [3]. We refer to the Y-chart in [3] which is a model of design representation and synthesis. It defines the structural, behavioral and physical domain description. "Structural" means the abstract implementation of the design, "behavioral" describes

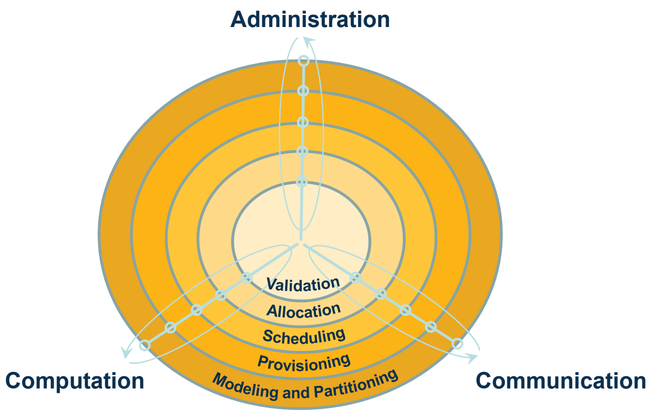

Fig. 1. The λ-chart: A Model of System-Level Design Abstraction.

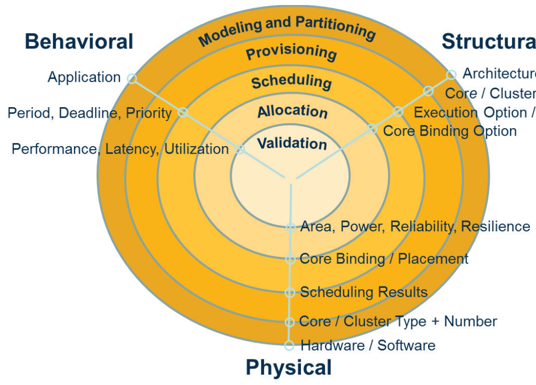

Fig. 2. Relationship between λ-chart and Y-chart [3] illustrated for the Computation View and our Unified Design Process.

the functionality of the design and "physical" relates to the physical realization of the abstract structure. Hence, synthesis is defined as transition from the behavioral model to the structural model ending in the physical realization. Figure 2 contains relevant domain descriptions for the computation view and the unified design process. In the example, not all process steps include a domain description. For example, provisioning is limited to the structural and physical description. In contrast to Y-chart synthesis, the λ-chart jointly uses structural and behavioral descriptions to realize the physical description. For example in Figure 2, the simulation of application and architecture creates scheduling and binding results to physically map application tasks to cores. In the λ-chart, each design view and process step is further characterized with the three domain descriptions. This is demonstrated in Section IV-VI. Moreover, the λ-chart can be considered as refinement of the architectural level within the Y-chart in [3]. This corresponds to the fact that our model addresses decisions at early design stages.

III. Related Work

Thomas et al. [4] present a model and methodology for mixed hardware-software design. According to given performance goals, the design process results in optimal hardware and software realizations. In contrast, our model aims at abstraction of system-level design. Therefore, we separate several views on the system to reduce design complexity. Software functionality is covered through high-level task graphs.

In [5], the authors propose interface-based design which abstracts design with decomposed components. Moreover a communication view is implicitly considered separating communication and component behavior. In addition to structural compositions we also decompose behavioral and physical descriptions independently on each other. Our approach is more generic because it uses communication as separate design view to support also less communication-centric designs.

Blickler et al. [6] define system-level synthesis as mapping of task-level specifications onto heterogeneous hardware/software architectures. They introduce a new formal definition for system-level synthesis which includes several process steps. Our model refines this approach to a unified design process applicable to several design views.

In [7], system design at different abstraction levels is proposed which reduces design complexity by separating concerns, such as function, architecture, computation and communication. The authors also use platform-based design to map applications to abstract representation of micro-architectures similar to [6]. In [8], the author proposes platform-based design as unified methodology applicable to future systems with heterogeneous subsystems, such as electronic and mechanical components. Despite our model supports platform-based design, we extend the work unifying the design process and also considering an administration view.

Gerstlauer and Gajski [9] have developed a design process starting with system specification to build the architecture by mapping communication and computation. The authors focus on model refinement between abstraction levels, e.g., system-level and algorithmic level. In addition, Kienhuis et al. [10] have developed a tripartite representation of system-level design. Therein, application models are mapped to architecture models and evaluated afterwards. We also consider modeling, mapping and evaluation in the design process. In general, our separation of design view and process allows a more generic description of system-level design and exploration.

IV. Views on System-Level Design

Our model of system-level design abstraction includes three design views: administration, computation and communication. Each view represents a separate portion of the total design space. Hence, inter-view links are necessary to synchronize with the other views. Although other divisions of the system-level design have been published, e.g., by [9] and [10], we believe that our division is the most natural. Thus, the administration view includes the design of planning, monitoring and control tasks of a system and its subsystems. In the computation view, all designs related to the code execution are covered. The design of data storage and data exchange between components is considered within the communication view. In the following, each design view is characterized in more detail via the domain descriptions.

A. Administration View

The "Administration View" considers all design tasks for planning, monitoring and control in the system and its subsystems. A structural description is the administrative architecture, e.g., central or distributed. In the behavioral description, the administrative algorithm would be realized either in a static or dynamic manner, such as static or dynamic scheduling. Considering physical descriptions, administrative units, such as hardware schedulers, are placed at a geometric position in the design. Hence, hardware-based vs. software-based administration corresponds to the physical realization.

B. Computation View

The "Computation View" considers all design tasks related to code execution. The computational architecture could be designed as central system or divided into subsystems (clusters). Further examples of structural descriptions are the decision for a heterogeneous or homogeneous set of processing elements (PEs) and subsystems (clusters). In contrast, the computational behavior is characterized by the degree of application parallelism or the number of inputs and outputs. Physical descriptions relate to binding and placement which typically take geometric and area constraints into account.

C. Communication View

The "Communication View" considers the design of data storage and data exchange between components, such as memory, router and peripherals. The topology of memory and network are examples of structural descriptions. Strategies for routing and data caching are considered as behavioral description. For the physical description, the dimensioning of links, routers (buffers) and memories are important realizations.

D. Further Characterizing the Design Views

The domain descriptions in the λ-chart include components and behavior in different levels of hierarchy. The hierarchical compositions occur in each design view and we divide them into structural, behavioral and physical compositions. This enables to further specify the design views and allows the formal representation of inter-view relationship, described further below. Figure 3-5 illustrate exemplary compositions for each design view as tree-based graph representations. Figure 3 shows an example of the administration architecture. Therein, a software-based application balancer supplies several hardware schedulers with application code. Hence, this describes structural and physical details of the design view. Referring to Figure 3, a behavioral composition is given for different scheduling strategies, such as As Soon As Possible (ASAP), Earliest Deadline First (EDF), priority-based etc. In Figure 4, structure and behavior of the computation view are depicted as hierarchical compositions. The system consists of several subsystems (clusters) which include either heterogeneous or homogeneous sets of PEs. A PE could be a RISC core, ASIC core, DSP core etc. Referring to Figure 4, the behavioral composition shows that sets of computation tasks are scheduled and that synchronization between these sets is necessary. Here, the term "task" means a computation kernel which is executable on a PE. Task examples are FFT, FIR, ENC, DCT etc. In Figure 5, the memory structure of the communication view is represented as hierarchy of main memory and local cache. In the physical composition, memory is realized as DDR3 and SRAM. An example for the behavioral composition is the selection of the switching technique, here guaranteed service, wormhole switching etc.

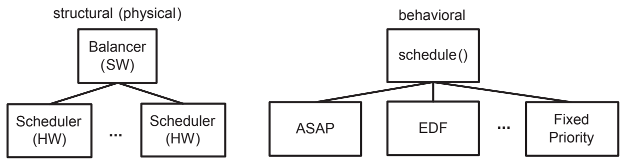

Fig. 3. Hierarchical Compositions in the Administration View.

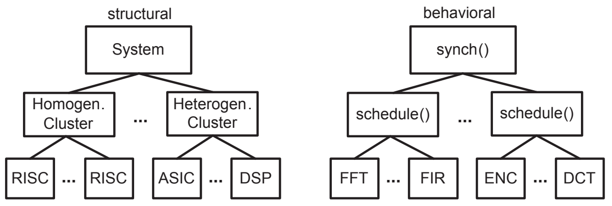

Fig. 4. Hierarchical Compositions in the Computation View.

In general, relationships between the design views exist. For example, the structure of communication and computation are closely coupled as depicted in Figure 6. Therein, all subsystems (clusters) access the main memory and each PE includes a local cache. Hence, formal representations are necessary to integrate inter-view relationships into system-level design. Referring to Figure 6, graph representations are suited to model the relationships between design views.

E. Importance of the Administration View for Dynamic Behavior

We introduce the administration view in system-level design abstraction because administrative tasks and the design of such hardware units become important in future MPSoC and Many-Core systems. For example, administration units can be used to improve reliability, power consumption, programmability, product reuse etc. In general, administration can handle static

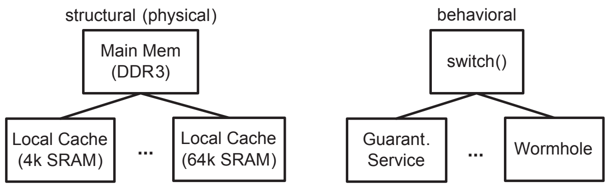

Fig. 5. Hierarchical Compositions in the Communication View.

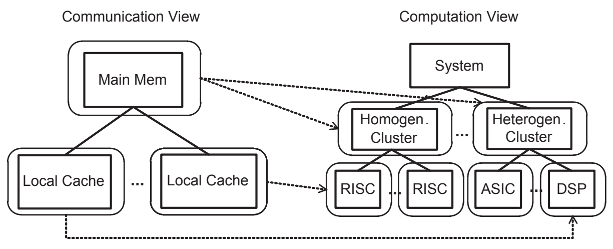

Fig. 6. Relationship between Computation and Communication.

TABLE I
LIMITATIONS FOR STATIC ADMINISTRATION OF DYNAMIC BEHAVIOR

No adaption to	Computation	Communication	Administration
Failure/ Resiliency	Processing element	Router, Link, Memory	Administrative unit
Hotspots	Thermal issue	Transfer bottleneck	Monitoring/ control bottleneck
System/ User changes	Performance scaling, Energy saving		Non-deterministic task dependency
Data dependent control structures	Non-deterministic operation		
Concurrent applications			

and dynamic behavior with static or dynamic mechanisms. If the changing conditions are known in advance, dynamic behavior can be administrated statically. Therefore, periodic applications with fix schedules are statically administrated, such as in data-flow driven communication protocols. As MPSoCs become subject to unpredictable dynamic behavior (see Table I), static administration is not suited anymore. Hence, dynamic administration increases design complexity which underlines the importance of the administration view. Table I lists limitations of static administration for dynamic behavior. It shows that components in the computation, communication and administration view can hardly adapt to unpredictable situations like failure/resiliency, hotspots, system/user changes, data dependencies or application concurrency. If components, such as a PE, memory, router, link or administrative unit, fail or require a certain fault tolerance, static administration is poorly able to react to this situations. Other examples are hotspots, such as thermal issues and bottlenecks, which can temporarily occur and would require dynamic management. Referring to Table I, high-performance or energy-saving modes are examples for unpredictable user and system changes. This can only be effectively handled via dynamic administration. Furthermore, static administration shows weaknesses in non-deterministic operation which occurs due to data dependent control structures and concurrent applications. Moreover, non-deterministic tasks dependencies can only be resolved via dynamic administration.

V. SYSTEM-LEVEL DESIGN PROCESS

The λ-chart defines five steps unifying the design process to: modeling and partitioning, provisioning, scheduling, allocation and validation. Prior to that, the design goal must be defined,

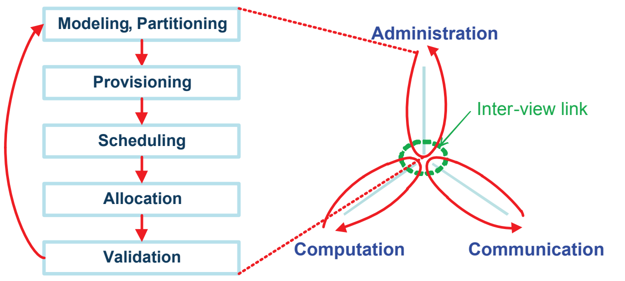

Fig. 7. Unified System-Level Design Process.

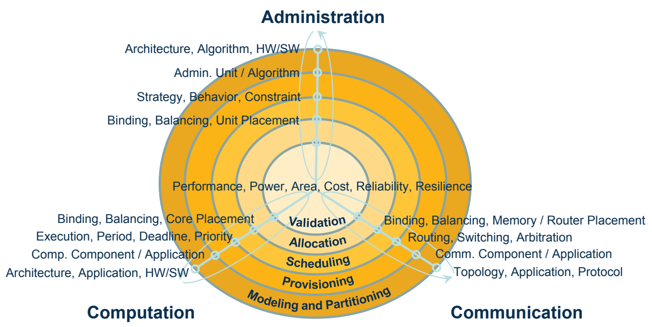
Administration
Architecture, Algorithm, HW/SW
Admin. Unit / Algorithm
Strategy, Behavior, Constraint
Binding, Balancing, Unit Placement
Performance, Power, Area, Cost, Reliability, Resilience
Binding, Balancing, Core Placement
Execution, Period, Deadline, Priority
Comp. Component / Application
Architecture, Application, HW/SW
Binding, Balancing, Memory / Router Placement
Routing, Switching, Arbitration
Comm. Component / Application
Topology, Application, Protocol
Validation
Allocation
Scheduling
Provisioning
Modeling and Partitioning
Computation
Communication

Fig. 8. λ-chart refined by Domain Descriptions for the Design Views and Design Process Steps.

e.g., low-power or high-performance. Figure 7 shows that the process steps are independently followed in each design view but the views have to be synchronized via inter-view links. This section describes all five process steps which have been further characterized by structural, behavioral and physical domain descriptions, as illustrated in Figure 8. Therein, the design views and design process are detailed by components, behavior, properties and design goals which relate to a domain description. Modeling and partitioning are limited to behavioral and structural descriptions because the process step does not intend a physical realization. Hence, provisioning, scheduling and allocation increasingly contain physical descriptions due to the elapsed design progress. Finally, validation must be restricted to physical descriptions since it represents the last design process step. Unfortunately, terminology is not consistently used amongst the researchers, e.g. the term allocation might also include provisioning for some people. Because we do not intend to confuse the reader by inventing new names, our model considers the most accepted naming to the authors best knowledge.

A. Modeling and Partitioning

Modeling and partitioning describe the first step in the design process which is to build formal representations of the system structure and behavior. In addition, partitioning underlines the importance of modeling hierarchical structures and concurrent behavior. Representations of the application and architecture should be adequate for an automated design flow. Therefore, graph representations are widely used and applied in the administration, computation and communication views. Referring to Figure 8, our model accounts for structural descriptions of system-level design by describing the topology (architecture). This also includes the modeling of hardware and software supported application functionality, known as HW/SW codesign. In addition, application and administrative algorithm represent behavioral descriptions.

B. Provisioning

Provisioning is the second step in the design process and basically means to select the type and number of components and behavior necessary to fulfill the purpose of the system under design. Hence, provisioning chooses structural and behavioral descriptions, e.g., cores, routers, application tasks and administrative algorithms, to increasingly create the physical realizations. For the administration, computation and communication view, different provisioning decisions are needed. These decisions include the reuse of existing IP, the design from scratch or an intermediate solution. Provisioning considers also different design hierarchies, such as number and type of subsystems (clusters). Usually, the decisions are determined by the availability and experience of designers, the cost of IP and the reuse of existing designs. Nevertheless, other criteria such as tool support or user and system requirements might influence the choice.

C. Scheduling

The scheduling step represents the temporal planning of the application and component behavior, such as execution, communication, monitoring and power mode. Referring to Figure 8, structural and behavioral descriptions include components, behavior and properties related to scheduling. A physical description is the realization either in software or hardware. Administration, computation and communication design could be closely coupled because administrative units are responsible for the scheduling of computation and communication behavior. For example, a scheduling algorithm jointly aims at improved computation performance and low communication latency. In that case, additional monitoring and control functionality is required in all three design views. Furthermore, priority based scheduling represents a widely used technique to account for real-time behavior. In that case, priorities must be considered as behavioral description.

D. Allocation

The allocation step focusses on spatial planning of the application, architecture and administrative algorithm. Allocation is considered in all design views. It requires structural and behavioral descriptions, such as units and algorithms for component binding and application balancing. A physical description is the assignment of application code to components, such as tasks to cores. Furthermore, balancing tries to level component usage to improve performance and reliability. Referring to Figure 8, physical description is also represented by the geometric placement of components, such as of cores, memories, routers or administrative units. Allocation also accounts for techniques aiming at power reduction, such as power/clock gating or frequency scaling. In general, scheduling and allocation must be closely coupled to improve the design goals.

E. Validation

Validation represents the last process step and it proves whether the system fulfills the previously defined purpose or not. As prerequisite of the validation, design execution is either based on an analytical, simulative or hybrid approach. During execution, adequate evaluation data is aggregated for further analysis. Finally, design validation is performed according to the given objectives and constraints. In case the current system misses the design goal, relevant representations and design decisions are adapted by means of an additional design iteration. This can be separately done for each design view. Validation is applied to the physical realizations. Hence, structural and behavioral descriptions can only be examined by means of their physical realization. For example, insufficient computation performance leads to changes in prior design steps and relate to one or more design views.

VI. System-Level Design Exploration

The model of system-level design abstraction is also suited to describe system-level design exploration. We introduce the following exploration types to be able to operate both with separated and integrated design views:

- single view,
- view-to-view and
- all views integrated.

The term design exploration denotes to systematically alternate design parameters with the goal of finding systems that fulfill the intended purpose. In Figure 9-11, we present the design exploration types based on three design examples.

Figure 9 illustrates single view exploration which is not necessarily limited to information of the explored view. In our case, administration delay and transfer time has been additionally provided after modeling and partitioning. Referring to Figure 9, all design process steps are followed in the computation view. In contrast, only "modeling and provisioning" is rudimentarily considered in the remaining views. In general, focusing to a single view simplifies exploration at the cost of limited coverage of the overall design space.

In Figure 10, an example of view-to-view exploration is shown. Therein, the computation view provides a fixed configuration which serves as basis for the exploration of the communication. The administration view delivers only basic information, such as strategy/behavior of scheduling and routing. View-to-view exploration realizes limited interaction between the views during the design process. This allows the exploration to improve design space coverage within a specific design view. Nevertheless, observation of design space in the residual views remains limited.

Single view and view-to-view exploration are either suited for designs dedicated to a specific view or to designs with a small

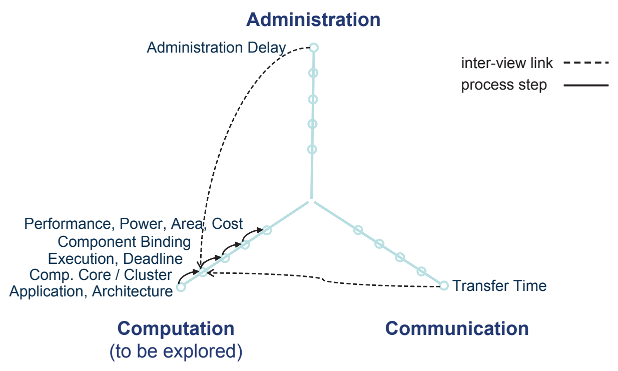

Fig. 9. System-Level Design Exploration - Single view.

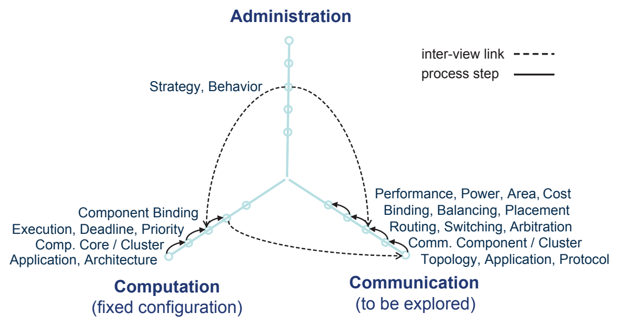
Administration
inter-view link
process step
Strategy, Behavior
Component Binding
Execution, Deadline, Priority
Comp. Core / Cluster
Application, Architecture
Performance, Power, Area, Cost
Binding, Balancing, Placement
Routing, Switching, Arbitration
Comm. Component / Cluster
Topology, Application, Protocol
Computation
(fixed configuration)
Communication
(to be explored)

Fig. 10. System-Level Design Exploration - View-to-view.

amount of interacting components and system functionality. An example would be to limit exploration to the performance analysis of IP cores assuming a single shared bus architecture. The design of complex systems, such as MPSoC, imply wider functionality, more interaction and dependencies between the three design views. A MPSoC is based on an on-chip network and different memory types. It has often several administration components, such as scheduler and performance monitor. Moreover, computation is enabled by an arbitrary number of cores. In addition, an MPSoC structure could be composed of several hierarchies, such as subsystems (clusters) for computation, communication and administration. Therefore, exploration integrating all views, as shown in Figure 11, is needed to create feasible MPSoC designs. Therein, the three design views are closely coupled. Several inter-view links allow the synchronization of results in each view and process step. Referring to Figure 11, exploration starts in the computation view which serves the communication view. In the example, the administration view requires input from the other views. But it also provides design decisions from the scheduling step towards the other views, such as execution schedule, routing strategy etc.

The exploration types are combined to form different exploration strategies. For example, exploration starts with the integrated type to initially create feasible design solutions. Afterwards, single-view or view-to-view exploration allow the optimization towards certain design goals within a design view and process step. Hence, the refinement should be guided with

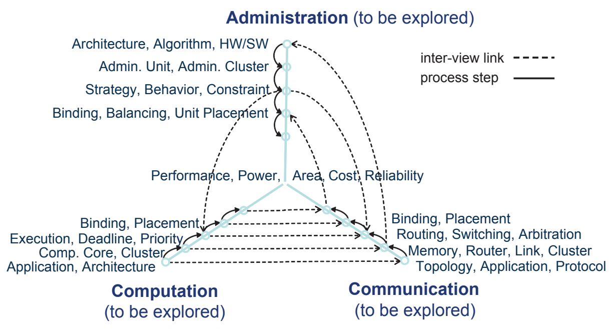
Administration (to be explored)
Architecture, Algorithm, HW/SW
Admin. Unit, Admin. Cluster
Strategy, Behavior, Constraint
Binding, Balancing, Unit Placement
inter-view link
process step
Performance, Power, Area, Cost, Reliability
Binding, Placement
Execution, Deadline, Priority
Comp. Core, Cluster
Application, Architecture
Binding, Placement
Routing, Switching, Arbitration
Memory, Router, Link, Cluster
Topology, Application, Protocol
Computation
(to be explored)
Communication
(to be explored)

Fig. 11. System-Level Design Exploration - All views integrated.

the validation results.

VII. Conclusion and Future Work

Increasingly complex chip design, such as MPSoC design, motivates us to contribute to the better understanding of design and exploration at system-level. Therefore, we developed a new model of system-level design abstraction. It reduces design complexity via abstraction to three design views and relationships between views have been considered. Furthermore, a unified system-level design process was introduced which is applicable to the three design views. As exploration becomes more important, the authors show how the model is used to formalize design exploration at system-level. This paper provides conceptual foundations to more holistic system-level design and introduces formalization of system-level exploration by giving examples. In future work, we will use our model for automatic system-level exploration. Ideally, the model will be proven within a real MPSoC design project.

References

[1] ITRS. (2011, Apr.) International technology roadmap for semiconductors 2010 edition. [Online]. Available: http://www.itrs.net

[2] D. D. Gajski and R. H. Kuhn, "New vlsi tools," *Computer*, vol. 16, pp. 11–14, Dec. 1983.

[3] R. A. Walker and D. E. Thomas, "A model of design representation and synthesis," in *Proc. of DAC*, 1985.

[4] D. Thomas, J. Adams, and H. Schmit, "A model and methodology for hardware-software codesign," *Design Test of Computers, IEEE*, vol. 10, no. 3, pp. 6 –15, Sep. 1993.

[5] J. A. Rowson and A. Sangiovanni-Vincentelli, "Interface-based design," in *Proc. of DAC*, 1997.

[6] T. Blickle, J. Teich, and L. Thiele, "System-level synthesis using evolutionary algorithms," *Design Automation for Embedded Systems*, vol. 3, pp. 23–58, 1998.

[7] K. Keutzer, A. Newton, J. Rabaey, and A. Sangiovanni-Vincentelli, "System-level design: orthogonalization of concerns and platform-based design," *CAD of ICs and Systems*, vol. 19, no. 12, pp. 1523 –1543, Dec. 2000.

[8] A. Sangiovanni-Vincentelli, "Is a unified methodology for system-level design possible?" *Design Test of Computers, IEEE*, vol. 25, no. 4, pp. 346 –357, Jul.-Aug. 2008.

[9] A. Gerstlauer and D. Gajski, "System-level abstraction semantics," in *Proc. of System Synthesis*, 2002.

[10] B. Kienhuis, E. F. Deprettere, P. v. d. Wolf, and K. A. Vissers, "A methodology to design programmable embedded systems - the y-chart approach," in *Proc. of SAMOS*, 2002.