A Platform-Based Approach to Communication Synthesis for Embedded Systems

Alessandro Pinto

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2008-54
http://www.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-54.html

May 19, 2008

A Platform-Based Approach to Communication Synthesis for Embedded Systems

by

Alessandro Pinto

Laurea (University of Rome "La Sapienza") 1999
M.S. (University of California at Berkeley) 2003

A dissertation submitted in partial satisfaction of the
requirements for the degree of
Doctor of Philosophy

in

Engineering – Electrical Engineering and Computer Sciences

in the

GRADUATE DIVISION
of the
UNIVERSITY OF CALIFORNIA, BERKELEY

Committee in charge:
Professor Alberto L. Sangiovanni Vincentelli, Chair
Professor Robert K. Brayton
Professor Zuo-Jun Shen

Spring 2008

The dissertation of Alessandro Pinto is approved:

_____

Chair Date

_____

Date

_____

Date

University of California, Berkeley

Spring 2008

A Platform-Based Approach to Communication Synthesis for Embedded Systems

Alessandro Pinto

Abstract

A Platform-Based Approach to Communication Synthesis for Embedded Systems

by

Alessandro Pinto

Doctor of Philosophy in Engineering – Electrical Engineering and Computer Sciences

University of California, Berkeley

Professor Alberto L. Sangiovanni Vincentelli, Chair

As the complexity of electronic systems increases, designers adopt a re-use methodology where new products are assembled out of components. This is a common trend in many application domains. In consumer electronics, Systems-on-Chips (SoCs) integrate many different cores to provide tens of different functions. In automotive, modern cars rely on a distributed, networked embedded system that comprises hundreds of processors to provide comfort, fuel efficiency and entertainment. In large scale systems, such as avionics and building automation, networked distributed controllers are used to provide comfort, safety and energy efficiency.

Since the system behavior depends not only on the components, but also on the way in which they interact, architecting their interconnection is a critical step in the overall design flow. Being subject to tight performance and cost constraints, the design of the interconnection architecture needs to be tailored to the specific system application. This task is too complex to be done by hand, considering also the heterogeneous nature of these systems. Therefore, there is a need for com-

munication synthesis tools that, starting from a characterization of the communication constraints among the components and the library of available communication building blocks, automatically derive an optimal interconnection architecture.

In this thesis I argue that the essence of the communication synthesis problem is invariant to the application domain. I introduce a formal framework to capture the communication constraints, the library of communication building blocks, and the rules to compose them. Using this framework, I formulate a general communication synthesis problem. I show the generality of the approach by formulating and solving the problem in two different domains: system-on-chips and building automation systems.

Professor Alberto L. Sangiovanni Vincentelli
Dissertation Committee Chair

To my mother who raised me and let me go to follow my dreams.

Contents

List of Figures

List of Tables

Preface

This thesis is divided in three parts: *Introduction*, *Theoretical Background* and *Applications*. The first part is an introduction to the problem of designing complex embedded systems and motivates the necessity for communication centric design flows. In the second part, we provide a formal model on which we built a software infrastructure that allows the development of dedicated communication synthesis tools. The third part focuses on domain specific applications. By applications we do not mean simply case studies. The term ”application” refers to the use of the design methodology and software infrastructure on a specific domain like on-chip communication and building automation systems.

Each chapter contained in this thesis starts with a summary of what is presented in the chapter and provides the context to the reader. Each chapter of the last part of this thesis, “Applications”, is organized according to the following structure. The communication synthesis problem is first introduced including domain specific details. The specification of the problem is formally defined capturing the main concerns of the design. The library of communication components is described and analytical models used for synthesis are described for realistic components. The optimization algorithms are described as mapping of the specification on a composition of library elements. The chapters end with the results obtained by running the algorithms on a set of case studies.

Acknowledgments

This thesis is the result of years of dedication to understanding problems and viable solutions. Its content benefited from the technical contributions of excellent scientists and moral support of relatives and friends. My acknowledgment goes to both.

Many of the original ideas included in this thesis stemmed from inspiring discussions with my advisor Alberto Sangiovanni Vincentelli, by far one of the most energetic and acute people I have ever met. I am truly impressed by the profound meaning and implications of his statements. He has been a guide and role model for research and academic integrity, and a moral support when things seemed to be going in the wrong direction. I would like to thank Luca Carloni that collaborated with me since I came to Berkeley. The discussions with Luca have always been very detailed and passionate, and very helpful in refining rough intuitions into high quality work. Thanks for the time spent in fixing my broken English.

Thanks to my colleagues and friends of the D.O.P. center: Alvise Bonivento, Luca Daniel, Abhijit Davare, Doug Densmore, Trevor Meyerowitz, Roberto Passerone, Claudio Pinello and Alessandra Nardi, Marco Sgroi, Guoqiang Wang, Guang Yang and Yanmei Li, and Qi Zho, and to the research scientists of the Cadence Berkeley Labs, Yosinori Watanabe, Felice Balarin and Luciano Lavagno. I won't forget to thank my friends of the graduate office Ruth Gjerde and Mary Byrnes, for listening to my complaints about politics, women and life in general; and, of course, many thanks to Lorie Mariano for making every GSRC workshop a good time to have fun. Thanks to Prof. Robert K. Brayton and Zuo-Jun Shen for reviewing my thesis.

The amazing journey that from Ericsson Lab Italy brought me to U.C. Berkeley stopped at PARADES, a research lab in Rome where I spent a year and a half. I met great engineers, scientists

and good people. I would like to thank Massimo Baleani, Andrea Balluchi, Luca Benvenuti, Alberto Ferrai and Donatella Santillo, Leonardo Mangeruca and Tiziano Villa for their technical advice, for hosting me during my visits in Rome and for their friendship and support. In the last two years of my Ph.D., I had the luck and pleasure to collaborate with the United Technologies Research Center. I would like to thank Clas Jacobson and Andrzej Banaszuk for their valuable suggestions and insights and for supporting my work on building automation systems.

There are many friends I have to thank for decorating my life outside the walls of Cory Hall. The first semester of my Ph.D. has been cheered up by Alessandro Serra. I met many Italian, European and American friends through the International Italian Student Association. It would be difficult to list them all, but to them I am thankful. Many Thanks to my house-mates and friends Bruno Sinopoli and Massimiliano Fratoni, to my quasi-house-mate and Napolitan friend Massimo Franceschetti who finally got a position and stopped complaining to us, and to uncle Dino Bellugi who, being Roman, reminded me of the old good times in the eternal city. I would like to thank the members of IBERIA (the Berkeley association of Spanish students), in particular Guillermo José Rein Soto Yarritu de la Gonzalera y Pérez del Hierro Martínez de la Cosa y Carvajal de la Apelliduría, also known as Guillermo Rein.

In Berkeley, I met Rebecca Julie Abergel, a devoted chemist and interesting woman who gradually became my beloved life's companion. She poured the last few drops of catalyst into my everyday life to complete this work and step into the future. Thank you Rebechita.

Last, but definitely not least, I thank my family, Gemma, Silvio, Barbara and Daniele. They saw me growing up, they raised me well and gave me everything I needed. Selfishly, I went away to build the future I liked. They let me go, and yet they love me.

Part I

Introduction

Chapter 1

Trends in Electronics

The complexity of electronic systems is *constantly increasing*. A measure of the complexity of an electronic product is the number of functions that it provides to the user. Technology scaling allows an increasing number of transistors to be integrated on the same chip, which enables the convergence of multiple applications on the same platform. The range of devices that can benefit from the integration of multiple functions are numerous: from hand-held consumer devices to large distributed systems for societal scale applications.

On the other hand, time-to-market *keeps shrinking*, posing serious challenges to designers who are required to complete the development of new products within a limited time window. Arriving first on the market is extremely important to fix the price of new products as to maximize revenues. Moreover, a larger market share can be gained if there is no competition initially.

The strategic importance of being able to keep pace with both trends is posing serious challenges to product developers. In fact, design and verification time of electronic systems is directly dependent on the number of hardware and software components that are used to implement

them. Thus, it takes longer to bring new complex systems to the market. A traditional solution to this problem has been to increase the size of the engineering and verification teams, in hopes of a corresponding increase in productivity. Unfortunately, communication overhead among team members reduces the potential benefits of larger engineering teams. Moreover, if the design methodology being used is not based on a formal model that supports compositional design, verification must be carried out on the entire system.

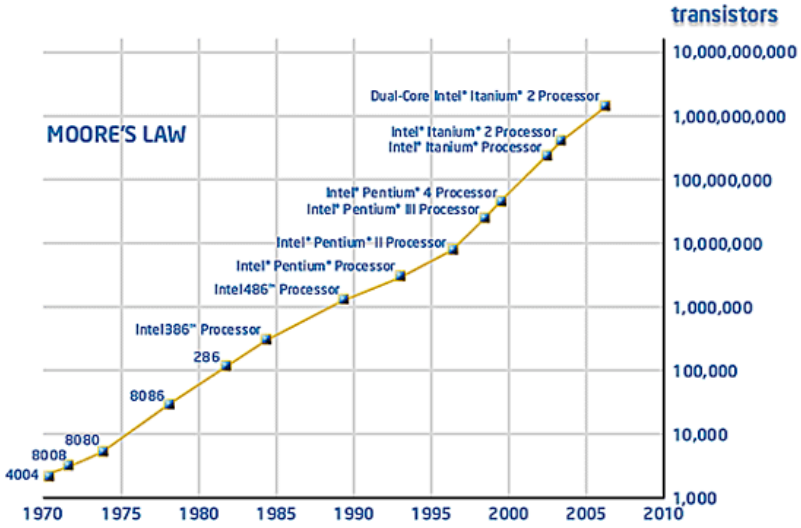
An alternative and promising way of increasing productivity is to re-use pre-designed and pre-verified components. Re-use has two advantages. Since a system is built by assembling large components, productivity in terms of number of gates per designer per year increases. Also, since each component has been pre-verified, the verification of the system is reduced to the verification of the interaction among components. However, this is not necessarily easier than verifying the entire system. To reduce, or even eliminate, the verification effort the design flow should be supported by a correct-by-construction methodology that provides the necessary tools to assemble systems automatically. Therefore, the automatic synthesis of communication systems, that we refer to as *communication synthesis*, is of primary importance in a methodology based on re-use. Notice that, whenever components are not available, abstract ones can be used instead, leaving their implementation to the subsequent design steps (at lower abstraction levels). Even in this case, communication synthesis is of great importance.

A communication synthesis tool should allow the description of the properties that a communication system must preserve. A synthesis algorithm should be able to derive the implementation of the communication systems by assembling communication components such that, *by construction*, the properties required by the specification are preserved. In an ideal system design flow,

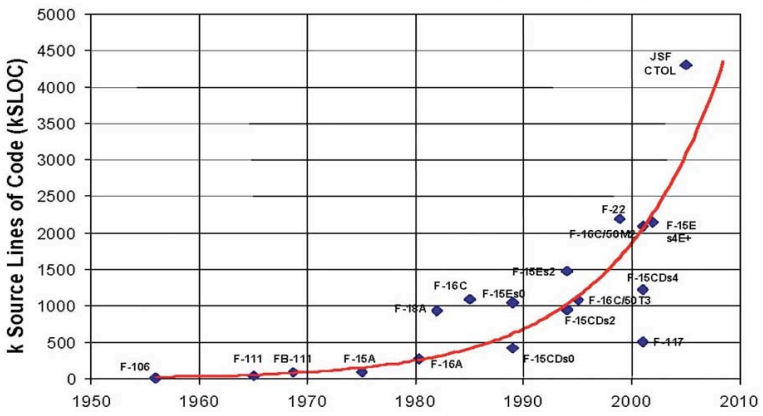
a designer would select a set of components able to implement the system functionality, and then define the properties that their interconnection must satisfy. These properties are then projected onto properties of the communication system and passed to a communication synthesis tool. Verification time is, therefore, completely eliminated because the implementation of the communication system is guaranteed to be correct with respect to the properties imposed by the designer.

1.1 System Complexity

In 1965, Gordon E. Moore observed that, for economical reasons, the number of transistors per chip followed an exponential trend [79]. This trend is known as the Moore's Law. In his original work, Moore predicted that the number of transistor integrated on the same chip would double every year. Even though this trend slowed down in recent years, it remains exponential. Figure 1.1(a) shows the number of transistors per die as a function of time in a logarithmic scale. The data points correspond to successive processor generations starting from the Intel 4004, the first example of an integrated processor on a single chip. The last data point, the Dual Core Intel Itanium, marks the beginning of the Chip-Multi-Processor (CMP) era for Intel. In [22], Borkar *et al.* explains that performance and power consumption are the main reasons to embrace parallelism. Traditionally, the demand for more computational power has been satisfied by increasing the processor clock speed. Unfortunately, memory performance has not been able to keep the same pace. In fact, memory latency is only slowly decreasing, becoming the major bottleneck in terms of computation speed. The second argument in favor of parallelism is power consumption. Having many slower processors instead of a powerful central one gives the flexibility of running part of the chip at full speed while maintaining the rest asleep. Power management becomes less flexible in



(a) Moore's law



(b) Software complexity in Avionics

Figure 1.1: System complexity: Number or transistors in Intel microprocessors as a function of time, also known as Moore's law (Source: Intel), and number of lines of code for avionic products [111].

single processor chips because running a software task, even one of limited complexity, requires the entire chip to be powered-on. Other companies like Sun and AMD have also moved from single to multi-processor platforms. This transition does have many consequences on the hardware and software design flows.

A similar revolution happened in the early 1980's. Computers with a central control unit [18, 86] were not able to satisfy the increasing computational demand coming from the scientific computation community, therefore researchers started to conceive and build parallel computers. The transition from single to multi-processor systems posed many new challenges [61] that are very similar to the ones that we also face today for multi-processor chips. The two major challenges are the programming models used to develop software for multiprocessor system and the communication network that interconnected the processors.

Programmers of multiprocessor systems need a programming model to pass data from one processing element to the other. Given the distributed nature of these systems, and the fact that a central memory would slow down computation due to memory latency, asynchronous models like message passing schemes have been adopted as successful programming models for many recent multiprocessor systems [13, 52]. Programming models (known as Models of Computation) that allow the capturing of application level concurrency have been widely studied and adopted in the design of embedded systems [64, 73]. Using a model of computation that can capture concurrency is essential since the parallelization of sequential specification is an undecidable problem [128].

The other challenge in the design of multi-processor systems is the interconnection network that provides the infrastructure for inter-processor communication. The most important requirements of the interconnection infrastructure are low latency and scalability. Latency depends

on network topology and the protocols used and impacts the overall performance of the multiprocessor system. For these reasons several regular interconnection topologies, routing algorithms and switching techniques have been proposed and characterized [33, 35, 88].

Chip Multi-Processor (CMP) systems for general purpose computing are comprised of a number of homogeneous cores. The complexity of these new chips is expected to grow exponentially in the future. An example of this kind of computing platforms is the 80-tile Polaris chip [121] from Intel. This chip features 80 processors interconnected by a mesh network. Depending on the class of applications targeted by a CMP, the number and types of cores can vary. In fact, there are many examples where the platform contains a heterogeneous set of cores like, for instance, the Cell processor [99]. This platform is comprised of 8 processing units and a master processor interconnected by a bidirectional ring network. Heterogeneous platforms are typically used in embedded systems that need to support several applications with different computation requirements and activation semantics.

An explosion in design complexity can be observed also for embedded system software. Figure 1.1(b) shows the number of lines of code of the control software in avionics products [111]. The challenges of real-time software development are related to scheduling, performance and correctness. In fact, software productivity can be as low as six lines of code per designer per day [107] if debugging time is also taken into account.

The challenges posed by complexity are also faced by the systems developer, beyond hardware and software. In consumer electronics, there is a trend to build devices that can perform multiple functions. Apple Inc.'s recent iPhone is a smart-phone offering EDGE connectivity, music and movie play-back, internet navigation, e-mail service, still image and movie capturing,

blue-tooth connectivity, calendar and address-book. The user interface is based on a touch screen. This device is battery operated, and therefore its design must be carefully tailored to provide a good customer experience while saving power. To provide all of these functions, Apple relied on third-party components that include: a central processing unit Samsung S5L8900 (which contains an ARM 1176JZF-S and 1 Gbyte of embedded DRAM), an audio codec, an accelerometer from ST Microelectronics, a power management unit, a DMA controller, a light proximity sensor from TAOS, a camera sensor from Micron, an I/O Broadcom device, an EDGE processor from Infineon that provides also the GSM radio frequency module, a display interface from National Instruments, a Marvel WLAN chip, a touch-screen from Balda, and a USB power manager and battery charger from Linear technology [43]. This is not a complete list, but it is long enough to understand that the complexity in building this device resides in assembling the components such that performance is guaranteed and power consumption is minimized.

The component integration problem is also faced by developers of large scale systems such as airplanes and building automation systems. For these systems, the design challenges arise from the scale of the system and the constraints imposed by the control algorithms. Building automation systems are highly distributed and are comprised of a large number of sensors, actuators and local controllers. Moreover, control algorithms of physical quantities, like temperature and pressure, are implemented by tasks that need to run periodically with a period that depends on the time scale of the dynamics of the variables that they control. Until a few years ago, building automation system designers could abstract computation and communication time because the time scales of the variables to control were much longer than the computation and communication delay. A common practice in designing these systems has been to over-provision the communication net-

work leading to solutions that were either grossly inadequate or over-designed and therefore much more expensive than needed. As new functionalities become either mandatory or desirable, the time scale separation property between the control algorithm and the platform does not hold any longer. Consequently the design problem becomes increasingly difficult. For example, ensuring that in case of an emergency, buildings can be evacuated safely and efficiently is an important novel requirement. The movement of people in emergency situations follows a fast dynamics so that egress control requires a sampling period that is comparable to computation and communication delays in the computing platform. In a survey [103] of 60 buildings conducted by the Lawrence Berkeley National Labs, 50% had control problems (including hardware, software and communication).

1.2 Time-To-Market and Productivity

The time-to-market is *the length of time that elapses from when a product is conceived to when it appears on the market*. This time window is a constraint for designers because the revenues generated by selling a new product are higher if the product is the first on the market. Time-to-market for electronic products is domain dependent with a wide variability. However, there is a common trend that is independent of the domain: Time-to-market continues tightening. A few years ago, the time-to-market for consumer electronics products was more than one year and it is today only six months [5]. In automotive, the time-to-market for the electronic equipment was 3 years and it is today one year [5]. Larger systems, like building automation systems, are also facing stringent time-to-market requirements.

Time-to-market is affected primarily by design and verification time. As systems become complex, not only design time necessarily increases, but verification time becomes the major bot-

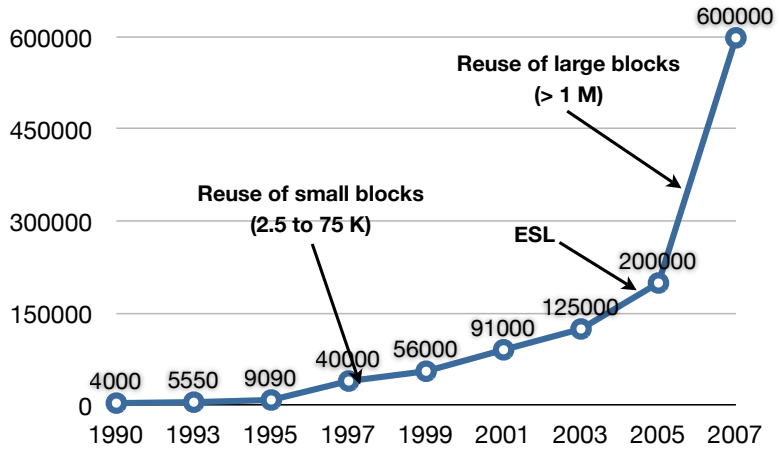
tleneck. In fact, the productivity of engineers is inversely proportional to the sum of design and verification time together.

In 2004, Synopsys reported the productivity data for hardware designers and predicted the required productivity until 2007. The chart is shown in Figure 1.2(a). Synopsys observed a major improvement in productivity in 1995 when designers started reusing small blocks of pre-designed logic (of the order of 2500 to 75000 gates). Synopsys predicted two other major improvements in productivity: one in 2005 with the introduction of system level design techniques and one in 2007 with the reuse of large blocks of logic (of the order of 1 Milion gates).

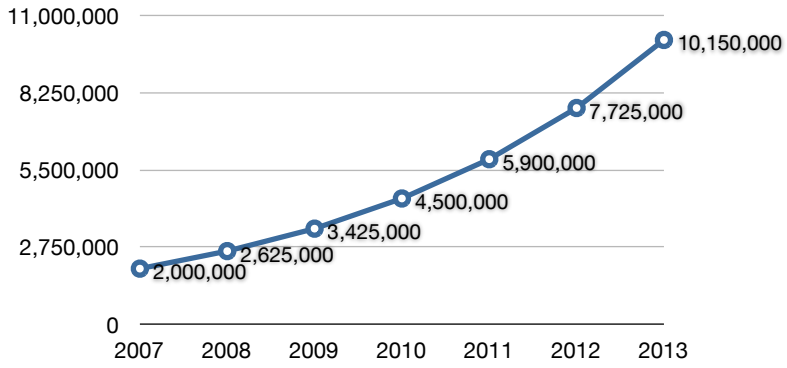
Figure 1.2(b) shows the productivity in number of gates per engineer per year predicted by the International Technology Roadmap of Semiconductors (ITRS [4]) until 2013. This chart suggests that productivity should grow exponentially with time. Given that design complexity will keep growing to sustain Moore's law, the exponential growth in productivity can be only achieved by reducing design and verification time. Increasing the size of design teams is not a viable solution to achieve this goal.

1.3 Re-Use

In recent years, re-use has been touted as a possible cure to this malaise whereby a design is the result of combining appropriately pre-designed and pre-verified components. In this case, verification amounts to checking the correctness of the interconnection among components. This is not necessarily easier, especially when components are developed by different providers, because their interfaces may be incompatible. Moreover, the communication architecture that interconnects the components affects the quality of the design in terms of performance and correctness. Therefore,



(a) Reported by Synopsys in 2004.



(b) Predicted by the International Technology Roadmap of Semiconductors (ITRS).

Figure 1.2: Hardware design productivity expressed in number of gates per designer per year

the communication architecture must be carefully designed. The verification process is further aided if the composition can be formally proven correct. There has been a flurry of activities in formal verification of interfaces that have clarified the main issues that need to be resolved [108, 97, 36, 114, 25, 96, 26, 27]. One of the key findings is that, if the components are designed with "clean" interfaces, composing the components, i.e., interconnecting them and establishing protocols that guarantee "correct" communication, can be done automatically. We call the automation of the composition of building blocks *communication synthesis*. Communication synthesis has been studied for years and, among the first pioneering work, we list Yen and Wolf [126], Ortega and Borriello [92] and an interesting approach to the synthesis of communication topologies by Gasteie, Munch and Glesner [50].

The major roadblock to effective reuse is the non-orthogonalization of computation and communication. The function implemented by a component and its interface are usually interdependent which makes a component work properly only in some specific contexts[1]. Standardization of the communication interface is one possible solution to this problem. There are currently many standardization efforts in different application domains like the Open Core Protocol (OCP) [89] for on-chip systems, and AUTOSAR [3] for automotive software systems. The building automation industry is still waiting for a definitive answer but there are at least two emerging standard communication protocols, Bacnet [24] for wired systems, and Zigbee [127] for wireless applications. Whenever components do not comply to standard interfaces, protocol adaptors need to be developed to guarantee interoperability.

Standardization does not completely solve the interconnection design problem. This problem can be stated as follows. Given the specification of a system defined by a set of functionalities

[1] By context we mean the rest of the system in which the component is used.

and a set of properties to be satisfied[2], and given a selection of components that implement the functionalities, design a communication system such that the properties are maintained. The communication system is defined by its topology, i.e. the set of links and intermediate nodes used for communication, the routing protocol, the medium access control protocol, the type of interfaces and all other parameters like transmission power and rate.

The design flow proceeds in two steps. In the first step, designers have to project the system properties to properties that the communication infrastructure must satisfy. For instance, at the specification level components communicate over point-to-point channels exchanging abstract data types called tokens. A token can be a real number of an image. The amount of data that is exchanged between them (for instance all the numbers that describe an image) translates to a certain amount of bits per token. After the execution rate of the components has been fixed, it is possible to define a minimum bandwidth requirement between the two components. This requirement is the property that the communication system must guarantee. Then, the communication system is designed by assembling communication components in such a way that all requirements are guaranteed.

Although the first step that may expose some difficulties, we focus on the second step whose complexity is due to the large number of components to be interconnected and the stringent cost and performance requirements. In this thesis we develop a Platform-Based design methodology for communication system design.

[2]Examples of properties are response time, volume of data to be transferred among components, cost and power consumption.

Chapter 2

Design Methodologies

The problem of designing complex systems can be tackled by raising the level of abstraction at which the design is carried out. Unnecessary information is hidden from the designer who is left with a limited, and therefore manageable, set of choices. Abstraction must be done in a judicious way, so as to preserve the accuracy of those metrics that are used to rank system level design solutions. System-level design means different things to different communities. For System-on-Chip designers, this term commonly refers to the design of systems conducted at a level of abstraction that is higher than RTL (for hardware) and C or assembly language (for software). For automotive companies, system-level design is conducted at even higher abstraction levels, where a system is comprised of several networked Electronic Control Units (ECU) and mechanical parts.

System-level design for SoCs became popular in the research community in the 90's. Many researchers proposed different ways of raising the level of abstraction and bringing a design from concept all the way down to implementation, using different methodologies. A design methodology is a procedure that, when followed, should lead to a design with guaranteed quality.

The metrics used to judge the quality depend on the application domain. A design flow that adopts a certain methodology and solves a design problem in a specific application domain has an input and an output. The input consists in the the *problem specification*. The output is an implementable system. Depending on the abstraction gap between the specification and the implementation, the number of free variables that need to be fixed to obtain a deterministic design can be very large. A design methodology helps in exploring the space of the free variables in an intelligent way, such that the complexity can be managed by the designers.

Two generic design methodologies are *top-down* and *bottom up*. In a top-down design methodology, the abstract specification is refined incrementally by fixing a few variables at the time in a predefined order. This methodology is not always effective. In fact, the quality of the design depends on how related are the successive design phases.

An example of top-down design methodology for circuit design is the traditional logic synthesis design flow. In this design flow, a technology independent optimization is followed by technology mapping and place-and-route. This sequence of optimization steps does not yield good results in all cases. Interconnect delay is becoming so important that logic synthesis and place-and-route cannot be completely orthogonal design steps, rather, one may want to consider the impact of routing and wire delay already during technology mapping [98, 109].

In a bottom-up design methodology, an implementation platform is designed to satisfy a class of applications. The platform is constructed mainly by experience from previous designs, relying on predefined architecture templates and selecting the set of components (that includes usually programmable ones). The specification is then satisfied by customizing the implementation. This methodology usually leads to over-designed systems.

In recent years, a novel design methodology called *Platform-Based Design* (PBD) [45, 110, 101] has been proposed and successfully adopted in many application domains. This methodology is a *meet-in-the-middle*, where constraints are propagated top-down and performances bottom-up. The implementation of a specification is obtained by a mapping step that matches the constraints and the performance while minimizing a cost function. One of the key elements of PBD is the orthogonalization of function (what a system is supposed to do) and architecture (how the system actually implements the required functionality), and computation and communication that is particularly important to allow the reuse of the same component in different contexts. Given a set of interacting components that have been pre-designed and pre-verified, a desirable feature of a design methodology is that the composition does not introduce undesired behaviors. This can be achieved by allowing the designer to state the properties that the communication system must satisfy, and provide her with a design methodology and a set of tools for communication design (*Communication-Based Design* and *Communication Synthesis*, respectively).

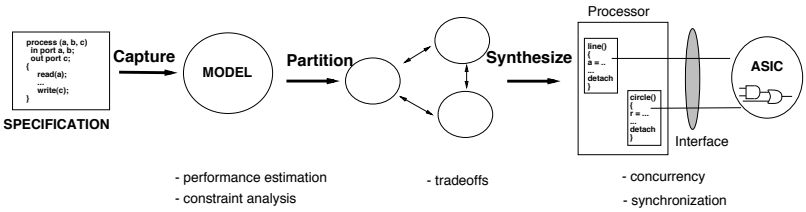
2.1 System-Level Design

A design flow starts with the specification of the functionality and constraints that must be satisfied by an implementation. Then, a synthesis process consists of using a set of primitives available to the designer to implement the specification so that the constraints are satisfied and the functionality guaranteed. The higher the level of abstraction, the easier it is to express the functionality and the constraints as well as to catch design errors early. However, quickly reaching a high-quality implementation is more difficult, due to the semantic gap between specification and implementation. Thus, researchers have either chosen to remain at high-levels of abstraction and

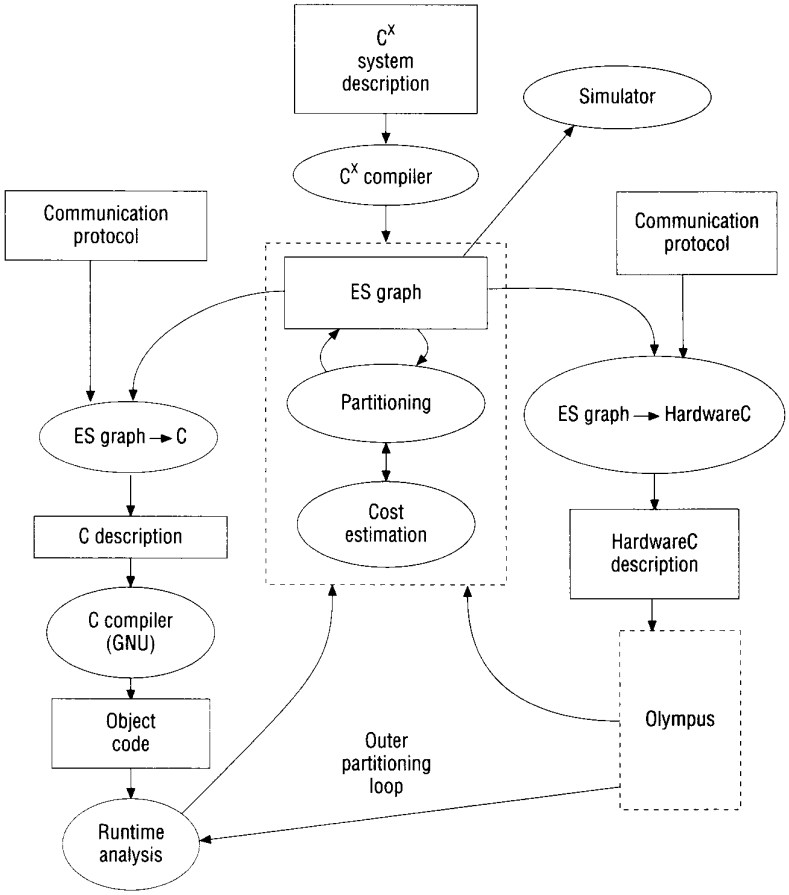
optimize high-level structures, or to begin with a low-level of abstraction that could reflect the characteristics of the implementation space. This is the case of early work in System-Level Design as in [102, 54, 44] where the specification, captured using a formal model, is partitioned in hardware and software (Hardware/Software Co-Design).

Gupta and De Micheli [54] presented a system-level design flow for hardware/software co-design shown in Figure 2.1(a). The specification is captured in a language called HARDWAREC which extends the standard C language with concurrency and is based on a formal semantics. The HARDWAREC description is then translated into an internal representation that allows verification and synthesis. The synthesis consists in partitioning the specification tasks into hardware and software while also taking into account the overhead of the interface between the two. The partition should minimize implementation cost while satisfying non-functional constraints, like rate of execution and latency, captured in the specification. The design methodology proposed by Ernst *et al.* [44] is shown in Figure 2.1(b). In this case the specification is captured in a formal language called C^X that is translated into a graph-based internal representation. The specification is partitioned into hardware and software and standard tool chains are used to generate the gate level and the assembly level descriptions, respectively.

The approach presented by Gajski [48] proposed the idea of having two abstraction layers. The specification of the problem is captured by a model called Program-State Machine (PSM) that is based on Statecharts [56] and Communicating Sequential Processes [60]. The input description is then translated into an internal representation that can be used for synthesis. The first step of the synthesis flow consists of allocating resources (processors, memories, busses and hardware) and partitioning the functionality on these resources. The second step is the refinement of the hardware



(a) System-level design flow presented in [54]



(b) System-level design flow presented in [44].

Figure 2.1: System-level design flows presented in early papers on this topic.

and software components. Other important work are related to mapping of applications to architectures taking into account different aspects like communication [106] and memory hierarchy [74].

Similar ideas have been also exploited in the Polis [15] project which already embodies some platform-based design concepts (see Section 2.2). The specification is captured using a particular model of computation called Co-Design Finite State Machine (CFSM) that can efficiently represent software and hardware. Each CFSM can be automatically refined into a software program, or into a hardware description language through synthesis algorithms. The result of the synthesis can be fed into standard refinement and synthesis flow to arrive at the final implementation of the entire system. The interface between hardware and software, as well as the real time operating system, can also be automatically generated. The Polis approach was somehow limited by the expressiveness of the model of computation that was suited for control dominated applications. Its successor, Metropolis [17], covers a broader rage of applications by providing a metamodel that can be used to described several other models of computation. Moreover, the metamodel can be used to capture architectural components, their composition and the quantities characterizing the performance and cost of each component. Mapping of a functional description on an architecture is achieved by synchronizing functional events and architectural events [16] using the Metamodel language features. Given its flexibility, the Metropolis infrastructure can be used to implement application specific design flows [100]. A further evolution of Metropolis is under development [10].

Other languages and tools for system level design exists and are offered by industry and academia. A comprehensive survey is provided by Douglas Densmore *et. al* in [39].

2.2 Platform-Based Design

We introduce Platform-Based Design (PBD) with an example. Assume we want to interconnect a set of nodes (e.g., computers) so that every node in the set can access every other node. Initial specifications may include the quality of service that each connection must be able to support, such as the required bandwidth and the maximum latency of the communication. We can solve this problem by constructing a network made of several different components such as routers, hubs, modems, protocol stacks, and links of different nature. The resources must be sized to satisfy the required constraints. However, the gap between our original, high–level, specification, and the implementation is obviously too large to be bridged in a single synthesis step: clearly, enumerating all possible topologies and interconnections is not practical, even for networks of modest complexity. A better way of approaching this problem is to divide this gap in several layers, where each layer focuses on a particular design choice. The question is then whether this division is optimal and, more importantly, how much of the entire design space can be explored. Answering these questions gives us an idea of the quality of the solutions that we obtain. Our approach consists of quantifying the design exploration process by relating the levels of abstraction corresponding to different layers. If two layers are too far apart then performance estimation will likely be poor and will not provide the necessary support for the synthesis algorithms.

In this context, a *platform* consists of a set of library elements, or resources, that can be assembled and interconnected according to predetermined rules to form a platform *instance*. One step in a platform-based design flow involves mapping a function or a specification onto different platform instances, and evaluating their performance. This step is pictorially represented by a bow tie as in Figure 2.2. The left side of the bow tie represents the application space. One particular

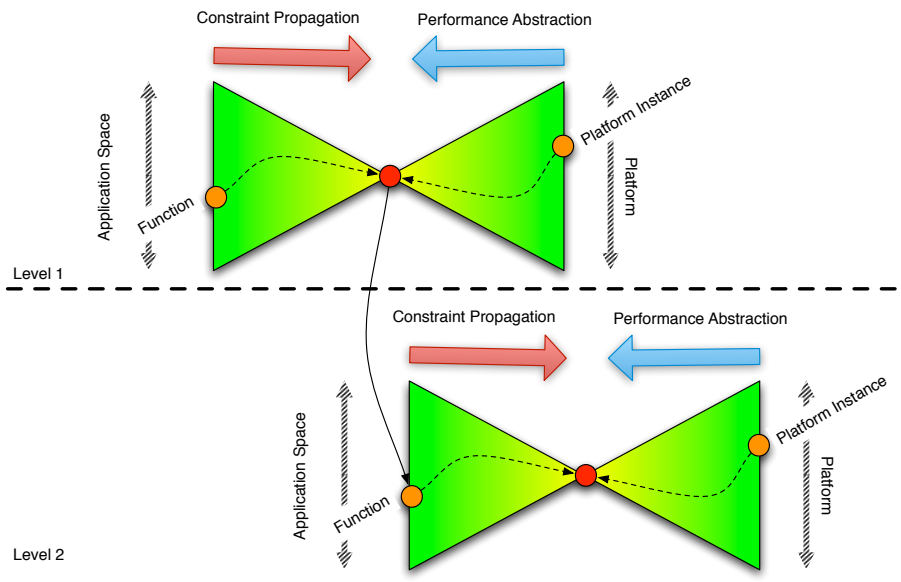


Figure 2.2: Pictorial representation of a platform-based design flow.

functional specification is a point in this space and it is called the *function*. The function is the design intent, or better, what the system is supposed to do. The right side of the bow tie is the implementation space and it is called the *platform*. A platform is the set of all valid compositions of library elements. One particular composition is called a *platform instance*. A platform instance can implement many different specification in many different ways. For instance, given a particular network, the same end-to-end constraints can be satisfied by selecting many different paths in the network for each constraint. Indeed, platform instances are characterized by performances that can be computed from the performances of the components and their relations. The mapping of the specification on the platform instance consists in selecting its parameters optimally such that the final cost is minimized and the performance matches the constraints. The result of mapping is a refinement of the original specification and of the platform instance, and plays the role of a new function at the lower level of abstraction.

Similar diagrams can be also found in early work by Gajski and Kuhn in 1987 [47], and Kienhuis *et al.* in 1997 [70]. Both diagrams are called Y-charts but they have very different meanings. The Y-chart presented by Gajski and Kuhn shows three different axes along which a design can be represented: the structural, functional and geometrical representation. For each of these axes, several abstraction levels are presented. This specific type of Y-chart is then used to show the levels of abstractions touched by several design flows. The Y-chart presented by Kienhuis is instead very close to the platform-based design view.

2.2.1 Formalizing Platform-Based Design

In this section we formalize Platform-Based Design (PBD) [45, 28, 123] using a rigorous algebraic framework to provide a methodology where we can conjugate the ease of expressing and verifying designs of high-levels of abstraction with the quality of low-level implementations. In Chapter 4 we use the same concepts developed in this section to formalize a platform-based design methodology for communication systems.

A *platform* consists of a set of library elements, or resources, that can be assembled and interconnected according to predetermined rules to form a platform *instance*.

Our formalization of the platform-based design methodology is based on the framework of Agent Algebra [95]. Informally, an agent algebra $\mathscr{Q}$ is composed of a domain D that contains the agents under study for the algebra, and of certain operators that formalize the most common operations of the models of computation used in embedded system design. Different models of computation are constructed by providing different definitions for the domain of agents and the operators. The algebra also includes a master alphabet $\mathscr{A}$ that is used as the universe of "signals" that agents use to communicate with other agents.

Definition 1. *An* agent algebra $\mathscr{Q}$ *has a domain* $\mathscr{Q}.D$ *of* agents, *a* master alphabet $\mathscr{Q}.\mathscr{A}$*, and three operators:* renaming*,* projection *and* parallel composition*, denoted by* $rename(r)$*,* $proj(B)$ *and* $\|$*. Each agent* $p \in \mathscr{Q}.D$ *is associated with an* alphabet $A \subseteq \mathscr{A}$.

The operators of the algebra are partial functions on the domain D and have an intuitive correspondence with those of most models of concurrent systems. The operation of renaming, which takes as argument a renaming function r on the alphabet, corresponds to the instantiation of an agent in a system. The renaming function is required to be a bijection, so that renaming is prevented from altering the structure of the agent interface, by for example "connecting" two signals together. Projection corresponds to hiding a set of signals, and takes the set B of signals to be retained as a parameter. Hence it corresponds to an operation of scoping. Finally, parallel composition corresponds to the concurrent "execution" of two agents. It is possible to define other operators. We prefer to work with a limited set and add operators only when they cannot be derived from existing ones. In particular, we will be mainly concerned with the operator of parallel composition. The operators must satisfy certain axioms that formalize their intuitive behavior and provide some general properties that we want to be true regardless of the model of computation. For example, parallel composition must be associative and commutative. The definitions of the operators is otherwise unspecified, and depends on the particular agent model being considered.

The notion of refinement in each model of computation is represented by adding a preorder (or a partial order) on the agents, denoted by the symbol $\preceq$. The result is called an *ordered agent algebra*. We require that the operators in an ordered agent algebra be monotonic relative to the ordering. This is essential to apply compositional techniques. However, since these are partial functions, this requires generalizing monotonicity to partial functions. This generalization is

however beyond the scope of this thesis. The interested reader is referred to [95] for more details.

It is easy to construct an agent algebra $\mathscr{Q}$ to represent the interface that components expose to their environment. In this case, the set D consists of the agents of the form $p = (I, O)$ where $I \subseteq \mathscr{Q}.\mathscr{A}$ is the set of input ports of the components and $O \subseteq \mathscr{Q}.\mathscr{A}$ the set of output ports. The alphabet of an agent p is simply $A = I \cup O$, and we require that the set of inputs and outputs be disjoint, i.e., $I \cap O = \emptyset$. The parallel composition $p = p_1 \parallel p_2$ is defined only if the sets O_1 and O_2 are disjoint, to ensure that only one agent drives each port. When defined, a port is an output of the parallel composition if it is an output of either agent. Conversely, it is an input if it is an input of either p_1 or p_2, and it is not concurrently an output of the other agent. Thus $O = O_1 \cup O_2$ and $I = (I_1 \cup I_2) - (O_1 \cup O_2)$. Given the definitions, it is clear that in this example connections are established by name.

The model can be enriched by including type information in the form of a set of possible values for each port. To ensure type consistency, parallel composition is defined only if the type of an output port is contained in the type of the corresponding input port, if one exists. For the refinement relationship, we choose to order agents $p_1 = (I_1, O_1)$ and $p_2 = (I_2, O_2)$ so that $p_1 \preceq p_2$ if and only if p_1 and p_2 have the same sets of inputs and outputs. In addition, the type of the inputs of p_1 must contain the type of the inputs of p_2, while the type of the outputs of p_1 must be contained in the type of the outputs of p_2. This ensures that p_1 can handle all the inputs that p_2 can, and does not generate outputs that p_2 does not.

The model can be further enriched with information about the nature of the signals used by the agents. For instance, in the case of agents that describe communication topologies, signals can be distinguished between those that belong to a link, denoted by the symbol l, and those that

belong to a component, denoted by the symbol n (non-link). We call this a *typed IO agent algebra*. The sets I and O of an agent p thus become sets of pairs of signals together with their type, i.e., $I \subseteq \{(a,t) : a \in \mathscr{Q}.\mathscr{A} \wedge t \in \{l,n\}\}$ and similarly for the output ports. Parallel composition can also be modified so that the operation is defined only if the ports of the agents being connected are *not* of the same type, i.e., a link must be used to connect two regular ports. Hence, $p_1 \parallel p_2$ is defined if and only if for all $i \in I_1$ and for all $o \in O_2$, if $i.a = o'.a$ then $i.t \neq o'.t$, and viceversa for p_2 and p_1.

With these definitions, it is in general not possible to derive the components from the composite. Later, we will see how this can be accomplished for a different model that we use to define architectures. There, we will also introduce non-trivial orderings of the agents.

We relate different agent algebras by means of conservative approximations. A conservative approximation from $\mathscr{Q}$ to $\mathscr{Q}'$ is a pair $\Psi = (\Psi_l, \Psi_u)$, where Ψ_l and Ψ_u are functions from $\mathscr{Q}.D$ to $\mathscr{Q}'.D$. The first mapping is an upper bound of the agent relative to the order of the algebra: for instance, the abstract agent in $\mathscr{Q}'$ represents all of the possible behaviors of the agent in the more detailed domain $\mathscr{Q}$, plus possibly some more. The second is a lower bound: the abstract agent represents only possible behaviors of the more detailed one, but possibly not all. Formally, a conservative approximations is an abstraction that maintain a precise relationship between the orders in the two agent algebras.

Definition 2. *Let $\mathscr{Q}$ and $\mathscr{Q}'$ be ordered agent algebras, and let Ψ_l and Ψ_u be functions from $\mathscr{Q}.D$ to $\mathscr{Q}'.D$. We say that $\Psi = (\Psi_l, \Psi_u)$ is a* conservative approximation from $\mathscr{Q}$ to $\mathscr{Q}'$ *if and only if for all agents p and q in $\mathscr{Q}.D$,*

$$\Psi_u(p) \preceq \Psi_l(q) \Rightarrow p \preceq q.$$

Thus, when used in combination, the two mappings allow us to relate refinement verifi-

cation results in the abstract domain to results in the more detailed domain. Hence, the verification can be done in $\mathscr{Q}'$, where it is presumably more efficient than in $\mathscr{Q}$. The conservative approximation guarantees that this will not lead to a false positive result, although false negatives are possible depending on how the approximation is chosen.

To define the inverse Ψ_{inv} of an approximation, we investigate whether there are agents in $\mathscr{Q}.D$ that are represented exactly by Ψ_u and Ψ_l rather than just being bounded. We do so by only considering those agents p for which $\Psi_l(p)$ and $\Psi_u(p)$ have the same value p'. Intuitively, p' represents p exactly in this case, and we therefore define $\Psi_{inv}(p') = p$. If $\Psi_l(p) \neq \Psi_u(p)$, then p is not represented exactly in $\mathscr{Q}'$. In this case, p is not in the image of Ψ_{inv}.

Definition 3. *Let $\Psi = (\Psi_l, \Psi_u)$ be a conservative approximation from $\mathscr{Q}$ to $\mathscr{Q}'$. For $p' \in \mathscr{Q}'.D$, the inverse $\Psi_{inv}(p')$ is defined and is equal to p if and only if $\Psi_l(p) = \Psi_u(p) = p'$.*

If the algebra $\mathscr{Q}$ is partially ordered (as opposed to preordered), the inverse of the conservative approximation is uniquely determined. Otherwise, a choice may be possible among order equivalent agents. In all cases, however, because of the defining properties of a conservative approximation, Ψ_{inv} is one-to-one, monotonic, and inverse of both Ψ_l and Ψ_u.

Assume now that for an agent p, $\Psi_{inv}(\Psi_l(p))$ and $\Psi_{inv}(\Psi_u(p))$ are both defined, It is easy to show that $\Psi_{inv}(\Psi_l(p)) \preceq p \preceq \Psi_{inv}(\Psi_u(p))$. This fact makes precise the intuition that $\Psi_l(p)$ and $\Psi_u(p)$ represent a lower and an upper bound of p, respectively.

We can use agent algebras to describe formally the process of successive refinement in a platform-based design methodology. There, refinement is interpreted as the concretization of a *function* in terms of the elements of a *platform*. The process of design consists of evaluating the performance of different kinds of instances in the platform by mapping the functionality onto its

different elements. The implementation is then chosen on the basis of a cost function. We use three distinct domains of agents to characterize the process of mapping and performance evaluation. The first two are used to represent the platform and the function, while the third, called the *common semantic domain*, is an intermediate domain that is used to map the function onto a platform instance.

A platform, depicted in Figure 2.3 on the right, corresponds to the implementation search space.

Definition 4. *A platform consists of a set of elements, called the* library elements*, and of* composition rules *that define their admissible topologies of interconnection.*

To obtain an appropriate domain of agents to model a platform, we start from the set of library elements D_0. The domain of agents D is then constructed as the closure of D_0 under the operation of parallel composition. In other words, we construct all the topologies that are admissible by the composition rules, and add them to the set of agents in the algebra. Each element of the architecture platform is called a *platform instance*.

Performance evaluation usually requires that the elements of a platform include information regarding their internal structure. Thus, an algebra such as the typed IO agent algebra described is not suitable for this purpose, since composition does not retain the structure of the agent. The IO agents can, however, be used as library elements D_0. A new domain of agents D can then be constructed as follows. If $p_0 \in D_0$ is a library element, we include the *symbol* $\mathbf{p_0}$ in the set of agents $\mathscr{Q}.D$. We then close the set D under the operation of parallel composition. However, we represent a composition $p = p_1 \parallel p_2$ in $\mathscr{Q}$ as the *sequence of symbols* $\mathbf{p_1} \parallel \mathbf{p_2}$. By doing so, we retain the *structure* of the composite, since all the previous composition steps are recorded in the representation. We call this process a *platform closure*.

Definition 5. *Given a set of library elements D_0 and a composition operator $\|$, the* platform closure *is the algebra with domain*

$$D = \{\mathbf{p} : p \in D_0\} \cup \{\mathbf{p_1} \parallel \mathbf{p_2} : \mathbf{p_1} \in D \wedge \mathbf{p_2} \in D\} \tag{2.1}$$

where $p_1 \parallel p_2$ is defined if and only if it can be obtained as a legal composition of agents in D_0.

The construction outlined above is general, and can be applied to building several different platforms, as will be shown later. The result is similar to a term algebra with the "constants" in D_0 and the operation of composition. Unlike a term algebra, however, our composition is subject to the constraints of the composition rules. For example an "architecture" platform may provide only one instance of a particular processor. In that case, topologies that use two ore more instances are ruled out. In addition, the final algebra must be taken up to the equivalence induced by the required properties of the operators. For example, since parallel composition must be commutative, $\mathbf{p_1} \parallel \mathbf{p_2}$ should not be distinguished from $\mathbf{p_2} \parallel \mathbf{p_1}$. This can be accomplished by taking the appropriate quotient relative to the equivalence relation. The details are outside the scope of this thesis.

On the other hand, the function, depicted in Figure 2.3 on the left, is represented in an agent algebra called the *specification domain*. Here the desired function may be represented denotationally, as the collective behavior of a composition of agents, or may retain its structure in terms of a particular topology of simpler functions. The denotational representation is typically used at the beginning of the platform-based design process, when no information on the structure of the implementation is available. Conversely, after the first mapping, the subsequent refinement steps are started from the mapped platform instance, which is taken as the specification. Thus, a *common semantic domain*, described below, is used as the specification domain. However, contrary to the mapping process that is used to select one particular instance among several, when viewed as a

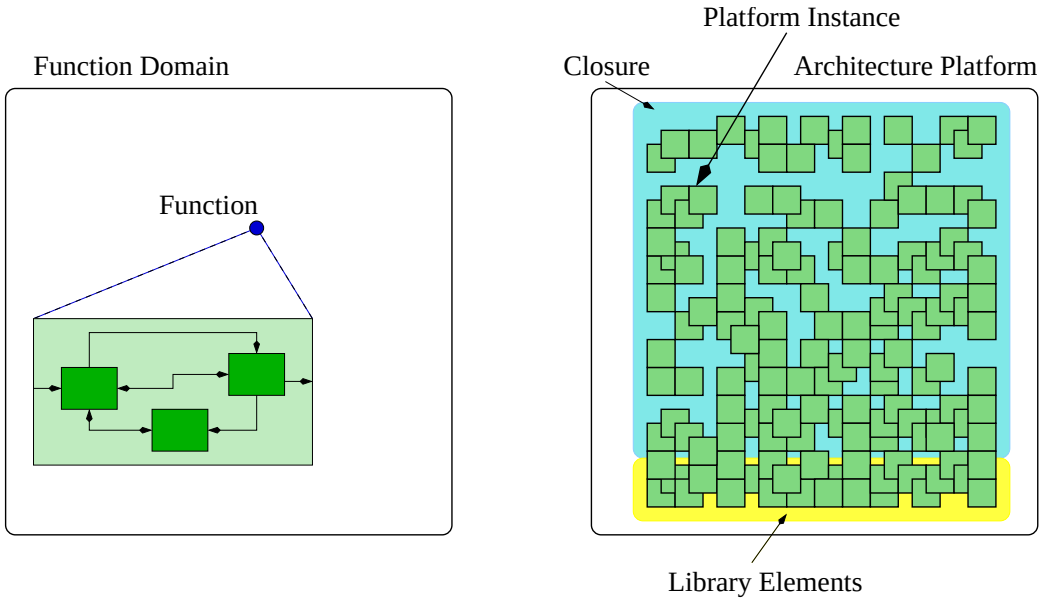


Figure 2.3: Architecture and Function Platforms

representation of a function the mapped instance is a specification, and it is therefore fixed.

The function and the platform come together in an intermediate representation, called the *common semantic domain*. This domain plays the role of a common refinement and is used to combine the properties of both the platform and the specification domain that are relevant for the mapping process. The domains are related through conservative approximations.

Definition 6. *Given a platform* $\mathcal{Q}^P$ *and specification domain* $\mathcal{Q}^S$*, a* common semantic domain *is an agent algebra* $\mathcal{Q}^C$ *related to* $\mathcal{Q}^P$ *and* $\mathcal{Q}^S$ *through conservative approximations* Ψ^P *and* Ψ^S*, respectively.*

In particular, we assume that the inverse of the conservative approximation is defined at the function that we wish to evaluate. The function therefore is mapped onto the common semantic domain as shown in Figure 2.4. This mapping also includes all the refinements of the function that are consistent with the performance constraints, which can be interpreted in the semantic domain.

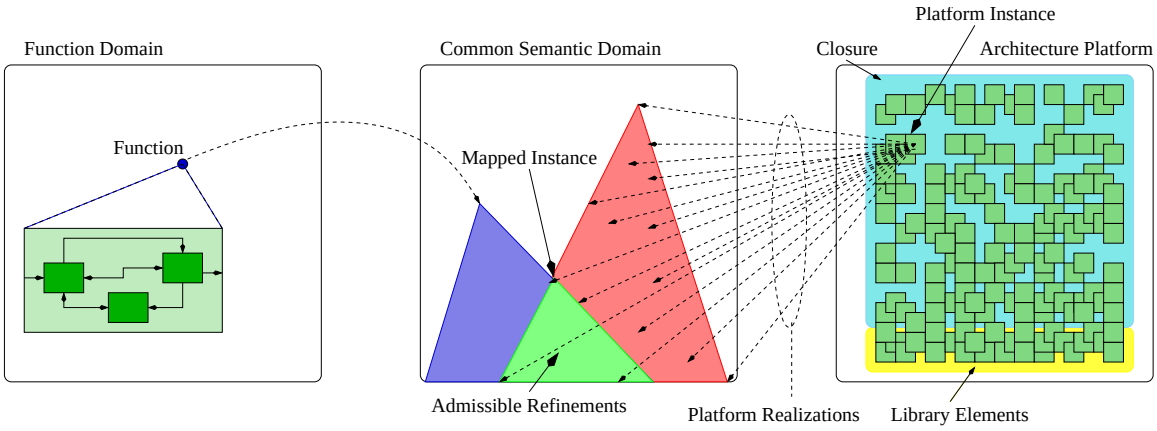


Figure 2.4: Mapping of function and architecture

If the platform includes programmable elements, the correspondence between the platform and the common semantic domain is typically more complex. In that case, each platform instance may be used to implement a variety of functions, or behaviors. Each of these functions is in turn represented as one agent in the common semantic domain. A platform instance is therefore projected onto the common semantic domain by considering the collection of the agents that can be implemented by the particular instance. This projection, represented by the rays that originate from the platform in Figure 2.4, may or may not have a greatest element. If it does, the greatest element represents the non-deterministic choice of any of the functions that are implementable by the instance.

The common semantic domain is partitioned into four different areas. We are interested in the intersection of the refinements of the function and of the functions that are implementable by the platform instance. This area is marked “Admissible Refinements” in Figure 2.4. Each of the admissible refinements encodes a particular mapping of the components of the function onto the services offered by the selected platform instance. These can often be seen as the *covering* of the

function through the elements of the platform library. Of all those agents, those that are closer to the greatest element are more likely offer the most flexibility in the implementation. Once a suitable implementation has been chosen (by possibly considering different platform instances), the same refinement process is iterated to descend to an even more concrete level of abstraction. The new function is thus the intersection of the behavior of the original function and the structure imposed by the platform. The process continues recursively at increasingly detailed levels of abstraction to come to the final implementation.

2.2.2 Example

Figure 2.5 shows and example of PBD flow applied to communication synthesis. First, we define a simple agent algebra $\mathscr{Q}$ to represent the interface that components expose to their environment. In this case, the set D consists of the agents of the form $p = (I, O)$ where $I \subseteq \mathscr{Q}.\mathscr{A}$ is the set of input ports of the components and $O \subseteq \mathscr{Q}.\mathscr{A}$ the set of output ports. The alphabet of an agent p is simply $A = I \cup O$, and we require that the set of inputs and outputs be disjoint, i.e., $I \cap O = \emptyset$. The parallel composition $p = p_1 \parallel p_2$ is defined only if the sets O_1 and O_2 are disjoint, to ensure that only one agent drives each port. When defined, a port is an output of the parallel composition if it is an output of either agent. Conversely, it is an input if it is an input of either p_1 or p_2, and it is not concurrently an output of the other agent. Thus $O = O_1 \cup O_2$ and $I = (I_1 \cup I_2) - (O_1 \cup O_2)$. Given the definitions, it is clear that in this example connections are established by name.

The model can be enriched with information about the nature of the signals used by the agents. For instance, in the case of agents that describe communication topologies, signals can be distinguished between those that belong to a link, denoted by the symbol l, and those that belong

to a component, denoted by the symbol n (non-link). We call this a *typed IO agent algebra*. The sets I and O of an agent p thus become sets of pairs of signals together with their type, i.e., $I \subseteq \{(a,t) : a \in \mathscr{Q}.\mathscr{A} \wedge t \in \{l,n\}\}$ and similarly for the output ports. Parallel composition can also be modified so that the operation is defined only if the ports of the agents being connected are *not* of the same type, i.e., a link must be used to connect two regular ports. Hence, $p_1 \parallel p_2$ is defined if and only if for all $i \in I_1$ and for all $o \in O_2$, if $i.a = o'.a$ then $i.t \neq o'.t$, and viceversa for p_2 and p_1.

The domain for each platform in Figure 2.5 is obtained following the construction of Definition 5. The set of library elements consists in all cases of appropriate subsets of the typed IO agent algebra. At the highest level of abstraction, the *connectivity platform* $\mathscr{Q}^c$ is only concerned with point-to-point connections between sources and destinations. The library elements of $\mathscr{Q}^c$ are of three types: the set of *sources* $\mathscr{S} = \{(I,O) : I = \emptyset\}$, the set of *destinations* $\mathscr{D} = \{(I,O) : O = \emptyset\}$, and the set of *point-to-point links* $\mathscr{L} = \{(I,O) : |I| = |O| = 1\}$. Furthermore, the input and output ports of sources and destinations must all have type n, while ports that belong to links have type l. Hence, given the rules of composition, it is not possible to connect sources and destinations directly. Architecture templates in the connectivity platform are simply point-to-point connections among a set of source and a set of destination agents.

A preorder in this algebra can be defined by considering substitutability. In general, an architecture that offers *more* connections can be substituted for another that offers *fewer* connections. Hence we define $p_1 \preceq p_2$ if and only if p_1 and p_2 have the same set of sources and destinations, and for each link between a source-destination pair in p_2, there is a corresponding link between the same source-destination pair in p_1. To illustrate the order, consider the simple case shown in Figure 2.5 to the right. There, one source s_1 is connected to two destinations d_1 and d_2. The most

refined architecture instance includes all links of s_1 to d_1 and d_2. Intermediate architectures include only one link to either d_1 or d_2. The greatest element is finally the architecture with no links. The connectivity platform could be used to evaluate the impact of connectivity on the performance of a system.

The function to be implemented on the architecture is represented in some suitable domain. This domain depends on the application. For instance, multimedia applications are usually described using non-deterministic Khan Process Networks (KPN). For the purpose of mapping, the function is abstracted using a conservative approximation Ψ^f from the function domain to a common semantic domain, that we call *abstract function domain*, described by a typed IO agent algebra $\mathscr{Q}^{cf}$ that includes multicommodity flow information.

A commodity is a pair of element (c_n, c_v) where c_n is the type of the commodity and $c_v \in \mathbb{R}^+$ is the commodity value (we only consider this simplified description of the communication requirements to keep the exposition simple, but other constraints could be take into account like latency and statistical properties). A port of an agent in the abstract function domain includes a commodity in addition to its type. A parallel composition is defined only if the link connected to a port carries the same commodity with a higher value. An order for this model can be defined by considering the connectivity (as for the connectivity platform) and multicommodity flow containment (flows have the order induced by the reals). The approximation Ψ^f maps the input and output ports of a function to abstract source-destination pairs and the communication channels to links. The conservative approximation also assigns commodities to ports, that are estimates of the bandwidth required by the communication. Since the abstract model has no information about the behavior, none of the processes can be represented exactly, and the inverse of the conservative approximation

is not defined.

The inverse of conservative approximation Ψ^c_{inv} maps a connectivity platform instance in the abstract function domain. If there exists an instance with the required connectivity (case 1 in Figure 2.5), then it is possible to find a set of admissible refinement in the common semantic domain. It might happen, though, that the intersection between the function instance concretization and the platform instance concretization is empty. This is the case, for example, when there is a constraint on the maximum number of links in the connectivity platform. Among all agents in the set of admissible refinement, the greatest element is the one having the minimum number of connections, each carrying the minimum commodity such that the function instance constraints are still satisfied. This agent, that we call *connectivity instance*, is selected as the function instance for the next level of abstraction. This choice is made by considering that connections and commodity values are constraints that must be satisfied in some common semantic domain at lower level of abstractions. This agent is the less constraining agent and is therefore a good candidate for cheap implementations.

Platforms used in communication synthesis, however, often include more complex topologies. To model this situation we build the *topology platform* $\mathscr{Q}^t$, which uses the same elements of the connectivity platform with the addition of a library component called *router*. The set of routers is formally defined as $\mathscr{R} = \{(I,O) : (|I| \geq 1 \wedge |O| > 1) \vee (|O| \geq 1 \wedge |I| > 1)\}$. Notice that we are not yet considering simple one-input one-output FIFOs. The ports of a router are required to be of type n, so that links must be used to connect routers to the other elements of the platform. The routers allow one to construct all the well known topologies like rings, crossbars, stars and busses.

The ordering of agents is defined by the underlying connectivity and the number of hops

on each source-destination path (number of routers among all paths from source to destination). In particular $p_1 \preceq p_2$ if and only if p_1 connects more source-destination pairs than p_2 with fewer or as many hops. We can establish a relationship between the topology and the connectivity platform by a conservative approximation Ψ^t. The upper bound ignores the routers by constructing the underlying connectivity, while the lower bound is obtained by considering only the existing point-to-point links between sources and destinations. The inverse of the approximation Ψ^t_{inv} thus maps the connectivity instance to the corresponding fully connected topology.

A mapping between connectivity instance and communication topology is realized in a common semantic domain $\mathscr{Q}^{sp}$ which contains both topology and multicommodity flow information. Point-to-point connections in $\mathscr{Q}^{cf}$ become source-destination paths in $\mathscr{Q}^{sp}$.

The algebra $\mathscr{Q}^{sp}$ is similar to $\mathscr{Q}^{cf}$ with the addition of routers as library components. The library of this platform consists of sets of elements, one for each of the element type: sources, destinations, links and routers. For instance there are several types of links depending on their commodities.

The common semantic domain $\mathscr{Q}^{sp}$ is related to the abstract function domain $\mathscr{Q}^{cf}$ and the topology platform $\mathscr{Q}^{t}$ through appropriate conservative approximations Ψ^{fs} and Ψ^{s}, respectively. In either case, the approximation ignores the information not contained in the respective abstract algebra. The lower bound defines, as usual, the conditions under which the representation is exact. For example, a fully connected topology in $\mathscr{Q}^{sp}$ can be represented exactly in $\mathscr{Q}^{cf}$. Similarly, any topology with zero flows is represented exactly in $\mathscr{Q}^{t}$. The inverses Ψ^{fs}_{inv} and Ψ^{s}_{inv} of such approximations thus establish correspondences between the elements of $\mathscr{Q}^{cf}$ and $\mathscr{Q}^{sp}$ and between $\mathscr{Q}^{t}$ and $\mathscr{Q}^{sp}$. In particular, a topology in $\mathscr{Q}^{t}$ is refined in an ordered set of topologies by Ψ^{s}_{inv}, the greatest

element being the topology with commodities all equal to zero. Similarly, a multicommodity flow connectivity graph in $\mathscr{Q}^{cf}$ is mapped by Ψ_{inv}^{fs} to all the possible topologies and their flows that satisfy the connectivity and multicommodity flow constraints imposed by the connectivity instance. These mappings are represented by the triangles in Figure 2.5, which denote all the refinements in addition to the mapped element. The intersection of these sets represents all the possible networks that satisfy connectivity and multicommodity flow requirement. It is possible that a specific topology maps to a set which does not intersect the concretization of the connectivity instance in the common semantic domain. In that case, the topology is ruled out from the search space.

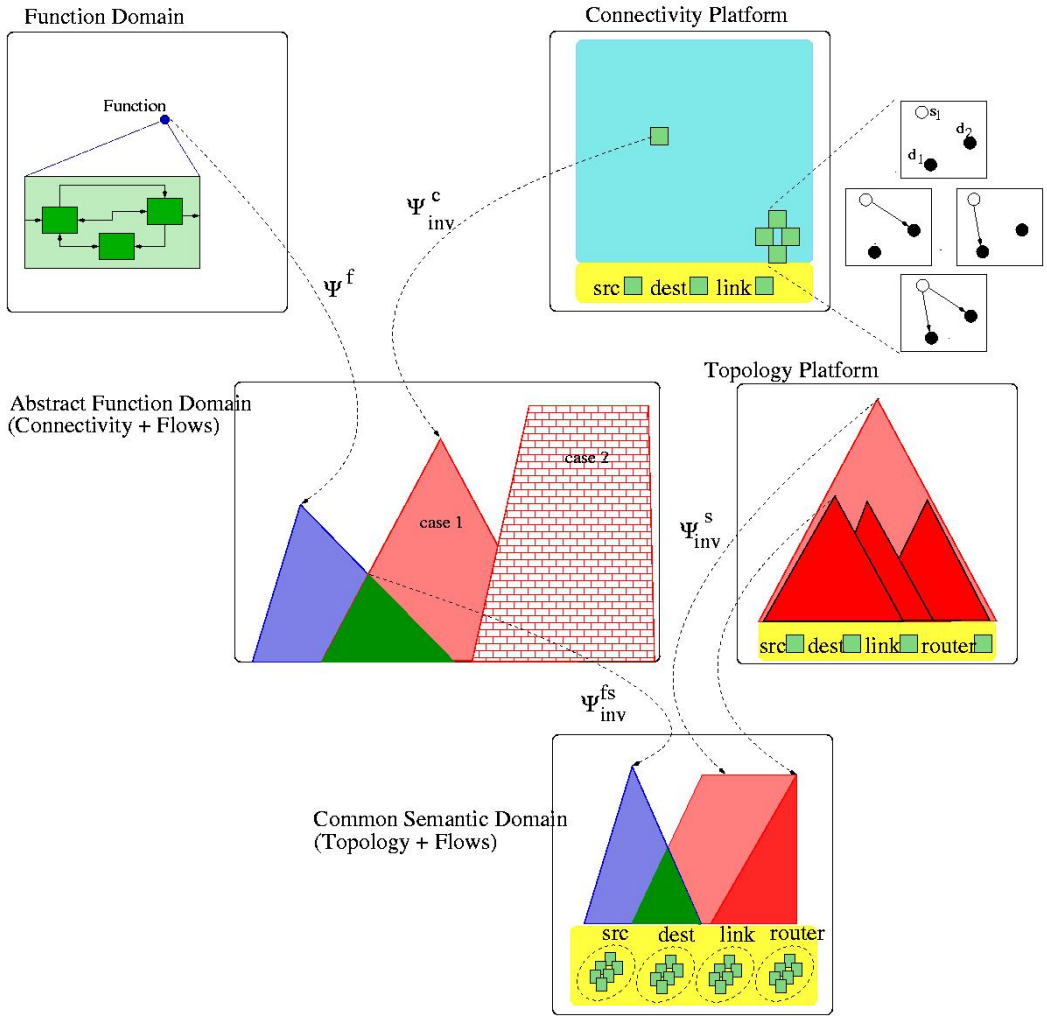


Figure 2.5: Platform-based design flow for communication synthesis

Chapter 3

Communication Synthesis

A communication system is the inner layer of a larger system composed of communicating agents. A communication system should offer the *connectivity* with a guarantee level of *Quality of Service* (QoS). Connectivity between two agents exists if it is possible for them to engage in an exchange of information. The performance guarantee associated with a connection, like for instance minimum bandwidth or maximum latency, are called Quality of Service. We refer to connectivity and QoS as *end-to-end communication constraints* [1]. Given a set of end-to-end constraints, and a set of components (i.e. nodes and links) that can be used to build a communication system, a communication synthesis flow automatically builds a network such that the constraints are satisfied and the total network cost is minimized.

The level of abstraction at which the communication synthesis problem is captured is much higher than the one in which the implementation is ultimately described. To bridge this gap, the design is split into several steps that go from the selection of the topology to the selection of the protocols and interfaces needed to derive the final implementation. In this chapter we discuss

[1] In other contexts, end-to-end constraints are referred to as demands.

some previous work that focuses on the optimization of networks. We first introduce the notion of *flows* that represent the flow of information between nodes. Then, we review some work in optimal network design that deals with networks where some particular structural properties must be satisfied. In the concluding remarks of this chapter we motivate the need for a formal model and a framework for communication design in embedded systems.

3.1 Flows

Interconnection networks are communication systems built by means of links (i.e. connections that are used to directly ship goods from one node to another) and nodes (i.e. intermediate points that serve as arrival and departure stations of goods). Networks are represented by graphs. A graph $G(V,E)$ is defined by a set of vertexes V and a set of edges E. Let $\gamma : V \times V \rightarrow \mathbb{R}$ be a function that associates to each pair of vertexes a real number called *capacity*.

Definition 7 (Flow)**.** *Let $G(V,E)$ be a graph, $s \in V$ be a source and $t \in V$ a destination, and $\gamma : V \times V \rightarrow \mathbb{R}$ the capacity function associated with G. A flow in G is a function $f : V \times V \rightarrow \mathbb{R}$ such that:*

- $0 \leq f(u,v) \leq \gamma(u,v)$ *for all* $u,v \in V$
- $\sum_{(u,v)\in E} f(u,v) = \sum_{(v,u)\in E} f(v,u)$ *for all* $v \in E \setminus \{s,t\}$

This is the classical definition of a flow [94]. If we require a certain amount of flow f_s to exit the source s and enter the destination t, we can add the following two constraints:

$$\sum_{(s,v)\in E} f(s,v) = f_s, \qquad \sum_{(u,t)\in E} f(u,t) = -f_s$$

In general, we can write all these constraints in a matrix form that can be directly used as a linear constraint in many linear programming formulation of flow related problems. Let order V and E such that $I = \{1,\ldots,|V|\}$ is the vertex index set and $J = \{1,\ldots,|E|\}$ is the edge index set. Let $A = [a_{ij}]$ be the node-arc incidence matrix of G such that:

$$a_{i,j} = \begin{cases} 1 & if\ arc\ j \in J\ leaves\ node\ i \in I \\ -1 & if\ arc\ j \in J\ enters\ node\ i \in I \\ 0 & otherwise \end{cases}$$

Let $\mathbf{f}$ be a vector such that $f(j)$ is the flow on the j-th edge. Then we can write the flow constraints as follows:

$$\mathbf{0} \leq \mathbf{f} \leq \gamma$$

$$A\mathbf{f} = \mathbf{b}$$

where $\mathbf{b}$ is a vector such that the component corresponding to s is equal to f_s, the one corresponding to t is equal to $-f_s$ and all other components are zero. The flow $\mathbf{f}$ can be also thought of as the superposition of several paths in the graph and, in fact, an alternative formulation of the flow constraints can be directly based on paths. Constraining the flows to follow only one path in the graph can be easily achieved by adding the following constraint: $f(i) \in \{0, f_s\}$ for all edges $j \in J$.

3.1.1 Maximum Flow

Given a graph $G(V,E)$, a source $s \in V$, a destination $t \in V$, and a capacity function γ, one interesting problem to solve is to determine the maximum flow that can be shipped from s to t. This problem is known as the single-source, single-sink, maximum-flow and has been extensively studied. A variant of this problem is the multiple-source, multiple-sink maximum flow problem

where the objective is to maximize the total flow that can be shipped from a set of sources $S \subset V$ to a set of destinations $T \subset V$. The problem can be formulated as a linear programming problem where the objective function is the sum of the outgoing flows from all the sources. The standard methods to solve this problem are the Ford-Fulkerson method [30] and its improvement by Edmonds and Karp [42], and the push-relabel algorithm [53]. For communication networks, this problem models the case where we are interested in achieving maximum bandwidth.

3.1.2 Minimum-Cost Flow

In the maximum-flow problem, the objective is to maximize the flow that corresponds to maximizing the amount of goods that can be shipped from one point to another in a network (e.g. bandwidth for communication systems or vehicle for transportation systems). In the minimum cost flow problem the objective is to guarantee a certain amount of flow at minimum cost.

Consider a graph $G(V,E)$ with an associated capacity function γ and let $s \in V$ and $t \in V$ be two distinct vertexes. The amount of flow that needs to be shipped from s to t is called *demand* and denoted by d. Also, let $c : E \times \mathbb{R} \rightarrow \mathbb{R}$ be a cost function that given an edge $e(u,v)$ and the flow through it $f(u,v)$ returns a cost $c(e, f(u,v))$. We want to find a flow such that the demand d can be shipped at minimum cost. The problem can be formulated as follows:

$$\min \sum_{e(u,v) \in E} c(e, f(u,v))$$

subject to

1) $\mathbf{0} \leq \mathbf{f} \leq \gamma$

2) $A\mathbf{f} = \mathbf{b}$

This problem is called single-source, single-sink, single commodity minimum-cost flow. A commodity can be best defined as a type of flow. This problem can be easily generalized to multiple sources and multiple sinks by introducing two virtual nodes: one super-source feeding all sources, and one super-sink that absorbs the flows of all sinks.

An interesting extension of the problem is to consider multiple commodities. In this variant of the problem we have a set of source-destination pairs $D = \{(s_1,t_1),\ldots,(s_k,t_k)\}$ and k demand values $d_1,\ldots,d_k$. Each demand d_i represents a different commodity to be shipped from source s_i to destination d_i. The problem can be formulated as follows:

$$\min \sum_{e(u,v)\in E} c(e, f(u,v))$$

subject to

$$1)\ f(u,v) = \sum_{i=1}^{k} f_i(u,v) \qquad \forall (u,v) \in E$$

$$2)\ \mathbf{0} \leq \mathbf{f} \leq \gamma$$

$$3)\ A\mathbf{f}_i = \mathbf{b}_i$$

where b_i is a vector such that the entry corresponding to s_i is equal to d_i, the entry corresponding to t_i is equal to $-d_i$ and all the other entries are zero. This problem is also known as the capacitated multi-commodity min-cost flow problem. Notice that the solution of this problem may very well split a flow at a node. Whenever this is not desirable, the following constraint can be added to the optimization problem:

$$4)\ f_i(u,v) \in \{0, d_i\} \qquad \forall i = 1,\ldots,k\ ,\ \forall (u,v) \in E$$

These flows are called *unsplittable*. With this additional constraint, each demand is shipped along a single path from source to destination.

Four cases can arise depending on the cost function:

Case 1 The cost function is linear, i.e for all edges $e(u,v) \in E$, $c(e, f(u,v)) = c_{u,v} f(u,v)$. In this case the problem is a standard linear program (LP) and becomes an integer LP (ILP) when constraint 4 is also taken intro account.

Case 2 The cost function is concave in $f(u,v)$ for each edge. Concave cost functions are used to model "economy of scale". In this case the problem is a concave minimization problem. One practical case is represented by piecewise-linear concave cost functions. A review of different approaches to this problem, and a new technique, can be found in [85].

Case 3 The cost function is convex in $f(u,v)$ for each edge. This class of cost functions has been used to model latency. In fact, the latency experienced by a flow depends on the total flow through an edge. There are two methods to solve this problem: the flow deviation method and the projection method (for a survey of this methods, please refer to [93]).

Case 4 The cost function is neither convex nor concave. The solution of the problem usually relies on ad-hoc heuristic methods.

3.1.3 Minimum-Cost Flow with End-To-End Constraints

The minimum-cost flow problem and all its variants are relevant for many applications in communication networks. The cost of a link can be modeled as the sum of an installation cost plus an operation cost, which is usually linear in the total flow through the link. Therefore, in many cases, the cost function is considered to be concave.

So far, the demands (i.e. the communication constraints) have been modeled by a single number that represents the value of the amount of goods to be shipped from a source to a destination (e.g. bandwidth). In many practical problems, it is also important to consider other performance

indexes. For instance, in communication networks the end-to-end delay is of extreme importance when real-time services are involved.

Consider the case of single-source, single-destination, single commodity, uncapacitated, unsplittable min-cost problem. The solution to this problem can be computed with Dijkstra's shortest path algorithm. Let $\delta(u,v)$ represent the delay on edge (u,v). The delay of a path $\pi = \{v_0,\ldots,v_n\}$ is simply $\delta(\pi) = \sum_{i=1}^{n} \delta(v_{i-1},v_i)$. The problem of finding the shortest path π^* from a source s to a destination t such that a delay constraints is met, is the constrained shortest path problem [66]. It entails to find the shortest path π^* from s to t such that $\delta(\pi^*) \leq \Delta$. This problem is NP-hard [49]. A review of techniques to solve this problem can be found in [40]. The constrained shortest path problem is a particular instance of the unsplittable multi-commodity flow problem with end-to-end constraints, which is, therefore, at least NP-hard. In resource constrained systems, like embedded systems, this class of problems is of particular interest because of their tight costs can be accommodated.

3.2 Optimal Network Design

Flow-based formulations of network design problems opened the doors to optimal route assignment and link capacity dimensioning. In real network design problems there are many other constraints to take into account including the type of equipment and the topology of the network. Excellent reviews of network design problems can be found in [69, 78, 113].

In many variants of the network design problem, the topology of a network is constrained to be a tree. Since each customer has to be connected to the network, the problem is to find a minimum spanning tree [30] or a minimum Steiner tree [63] (which is NP-complete in many of

its variants) where the customers are the leaves of the tree. Much recent work has addressed the problem of finding such optimal trees when the switches of the network have a limited number of ports. The problems are known as the minimum degree-bounded spanning tree [115] and minimum degree-bounded Steiner tree [105]. Other interesting topologies to consider are buses and rings. Busses are appealing for their low cost and effective support of broadcast and multicast transmissions. The main drawback is the throughput limitation imposed by the shared nature of the communication media. Bidirectional rings are inherently fault tolerant because the provide two paths for each source-destination pair.

In many case, communication systems are hierarchically organized. The nodes on one level of the hierarchy offer services to the nodes at the lower level. A node on one level can use the services of a node at higher level if they are connected by a communication link. The hierarchy can have two or more levels. The main problem in hierarchical network design is the location of nodes, since each node at one level of the hierarchy serves many other nodes at the lower level. A review of hierarchical network design problems can be found in [71].

Most of the network optimization problems that arise from real life applications are hard to solve, i.e. they are usually NP-hard. A general class of problems is called *bi-criteria network design* [75] that can be formulated as follows. A bi-criteria network design problem $(A, B, \mathscr{S})$ is defined by two objectives A and B and a membership requirement $\mathscr{S}$ that is a class of subgraphs. The problem is to find a network that minimizes the cost under the B objective and it is within a budget under the A objective. The network must belong to the class $\mathscr{S}$. For instance, if the class $\mathscr{S}$ is the class of trees, B is the length of the links and A is the degree of the nodes, the problem is to find a minimum length tree subject to node degree constraints. Ravi *et al.* prove in [105] that,

unless P=NP, it is not possible to find a polynomial time approximation algorithm that satisfies the budget constraints exactly, i.e. a polynomial time approximation algorithm exists if we also relax the budget constraints.

3.3 Concluding Remarks on Communication Synthesis

All the techniques reviewed in this chapter can be applied to the synthesis and optimization of interconnection networks. However, each technique represents an ad-hoc solution to a specific optimization problem of the overall communication design problem. Distributed embedded systems are the composition of heterogeneous components adopting different protocols. The problem, therefore, does not only lie in the optimization of the topology and routing of the system, but it touches all the aspects of a communication system including protocols and interfaces.

Moreover, because the applications for which they are used are cost-sensitive, the communication system that interconnects the different parts of the distributed system must be tailored to the application requirements and cannot be over-designed. Thus, a well designed communication system is likely to be heterogeneous, given the heterogeneous nature of the distributed system. Exploring heterogeneous composition of communication sub-systems is essential in the design exploration phase of a distributed embedded system.

The problem is too large to be tackled at one abstraction level. The communication synthesis problem has to be solved ad different abstraction levels. Moreover, the solution of the entire problem involves experts from different communities to contribute with optimization algorithms, communication protocols, partial designs and models for communication components. All these different aspects of the communication synthesis process should be easily mixed and matched to

build rigorous design flows leading to efficient solutions. To allow the exchange of benchmarks (i.e. specification of communication requirements), components, models, algorithms and results, the framework has to be based on a formal model that we present in Part II of this thesis.

To allow communication synthesis, we have developed COSI (COmmunication Synthesis Infrastructure) [31], a public-domain design framework for communication design that accompanies this thesis. COSI embodies a methodology based on the Platform-Based Design paradigm [45, 111]. Specifically, COSI enforces a clean separation among network specification, the library of building blocks that can be instanced and composed to derive the network implementation, the models of performance and cost associated with each of them, and the optimization algorithms that are used to explore the design space. Adopting this methodology allows comparing different communication system implementations and different building blocks thus smoothing out preconceived ideas about efficiency of particular communication technology.

Part II

Theoretical Background

Chapter 4

Communication Structures

In Chapter 2, we reviewed previous work on system-level design. Many of the framework and tools that we presented start from the description of the application in a high-level language and build an mathematical representation that is amenable to automatic manipulation. Having a formal model to express the system is very important to be able to formulate optimization problems and perform analysis and verification.

This is one of the reason to develop a formal model that helps us in reasoning about communication systems. As already mentioned in Section 3.3, our goal is to develop an infrastructure for communication design that will allow the collaboration of many experts to combine optimization algorithms, communication components, models and partial designs. To serve this purpose, we need a formal model that is shared among different plug-ins of the same tool.

The mathematical objects that we use are called *communication structures*. A communication structure is a set components $\mathscr{C}$ and a set of configurations L. Each configuration corresponds to a particular assignment of *quantities* to the components. Quantity variables are used to capture

either communication constraints or performance of components. We start with the definition of quantities and communication structures.

To make the formalism understandable, we draw examples from two different applications domains: on-chip communication and building automation systems.

4.1 Quantities

A quantity q is a variable that takes on values from a domain D_q. The domain of a quantity is partially ordered by a relation $\preceq_q$. The ordering relation captures the notion of a value being "better" than another value. We assume that $\perp$, which denotes no values, always belongs to the domain of a quantity D_q. Also, $\perp \preceq_q v$ for all $v \in D_q$. A quantity q is finite if D_q is a finite set, and it is bounded if there exists an element $\bar{v} \in D_q$ such that $v \preceq_q \bar{v}$ for all $v \in D_q$.

Quantities capture the properties of a component. They are used to specify constraints, i.e. the required services, as well as the performance, i.e. what level of service a component can offer. For instance, a quantity b with domain $D_b = \mathbb{R} \cup \{\perp\}$ is used to specify a minimum bandwidth requirement between a source and a destination. Another quantity γ with domain $D_\gamma = D_b$ is used to capture the capacity of a link (i.e. the maximum bandwidth that a link can sustain).

Given a vector of quantities $\mathbf{q} = (q_1, \ldots, q_k)$, the domain of $\mathbf{q}$ is the cross product $D_{q_1} \times \ldots \times D_{q_k}$. It is partially ordered by a relation $\preceq_{\mathbf{q}}$ point-wise induced by the relations $\preceq_{q_i}$. We use the notation $\perp^n$ to denote a n-tuple of $\perp$ values.

Example 1. (Bandwidth and Delay): Consider two quantities b and l that represent bandwidth and latency. Assume that the domains of these quantities are discrete and finite. Specifically, consider the case where $D_b = \{\perp, 10, 100\}$ (in Mb/s) and $D_l = \{\perp, 10, 100\}$ (in ns).

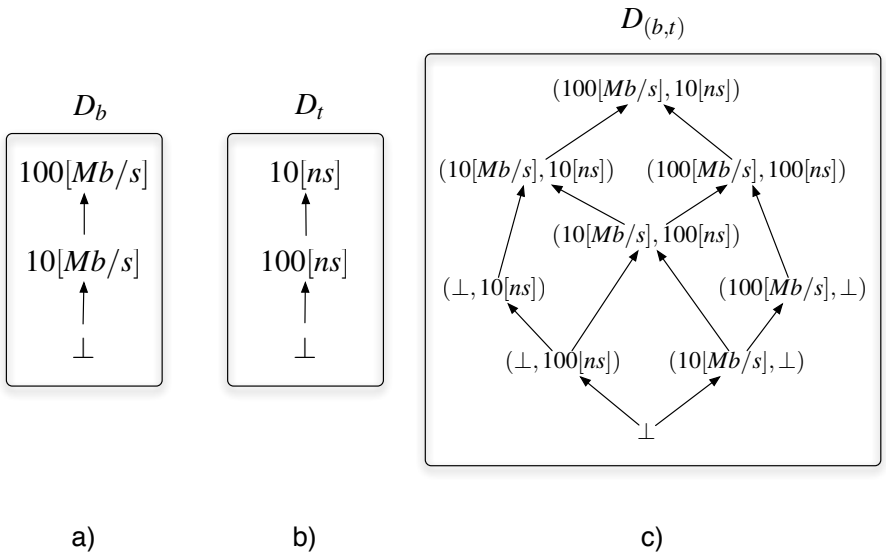


Figure 4.1: Hasse diagrams relative to the domains of three quantities: a) bandwidth, b) latency, c) the set containing both bandwidth and latency.

The two relations $\preceq_b$ and $\preceq_l$ correspond to the notion of "is better than" that we associate to the value of the quantities. To this extent, $100Mb/s$ is better than $10Mb/s$, and $10ns$ is better than $100ns$. Therefore, the two relations are such that:

$$\bot \preceq_b 10 \preceq_b 100, \quad \bot \preceq_q 100 \preceq_b 10 \, .$$

Figure 4.1 shows a representation of the ordering relations using the Hasse diagrams, where we also retain the unit of measure for the values of the quantities. Starting from the two quantities b and l, we can define the vector of quantities (b,l). The ordering relation $\preceq_{(b,l)}$ is shows in Figure 4.1-c. Notice that, even if the two domains D_b and D_l are totally ordered, the domain $D_{(b,l)}$ is only partially ordered.

4.2 Communication Structures

The basic element of our formal framework is the *communication structure*. A communication structure is meant to represent a set of *interconnected components*. Although a communication structure is a very general concept, we will often adopt terms from network and the graph theory since, ultimately, we will use our framework to solve communication problems. Therefore, the set of components of a communication structure that we are going to consider are nodes (i.e. center of computation, switching and routing) and links (i.e. communication channels that are use to interconnect nodes). Each component is associated with *quantities*. For instance, each node can be associated with a physical position, and each link with a bandwidth and a latency.

Definition 8. *A* communication structure *is a tuple* $N(\mathscr{C}, \mathbf{q}, L)$ *where* $\mathscr{C} = \{c_1, \ldots, c_n\}$ *is a set of components,* $\mathbf{q} = (q_1, \ldots, q_k)$ *is a vector of quantities, and* $L \subseteq [\mathscr{C} \rightarrow D_{\mathbf{q}}]$ *is a set of communication configurations.* [1] *Set* $\mathscr{C}$ *is partitioned into the set of nodes* $V \subseteq U_V$ *and the set of links* $E \subseteq V \times V$.

Let $\mathscr{G}_Q$ denote the set of all communication structures with quantities Q. The set U_V is called the *node universe*. Similarly, the *component universe* is $U_{\mathscr{C}} = U_V \cup U_V^2$, The set L of communication configurations captures the different ways in which quantities can be associated to components. A configuration $l \in L$ is a function that associates a tuple of quantities to each component. It is usually the case that some quantities are only meaningful when associated to nodes while others are other meaningful for links. The value $\perp$ is therefore used not only to denote that a component cannot be configured under any circumstance, but also that a quantity has not meaning when associated to that component. The *configuration universe* is $U_{\mathbf{q}} = \cup_{\mathscr{C} \subseteq U_{\mathscr{C}}} [\mathscr{C} \rightarrow D_{\mathbf{q}}]$, the union of all possible configurations for any subset of components. Let $\mathscr{G}_{\mathbf{q}}$ be the set of all communication

[1] $[X \rightarrow Y]$ denotes the set of all functions from set X to set Y.

structures with quantities $\mathbf{q}$. For a given subscript σ, and vector of quantities $\mathbf{q}$, let $N_\sigma \in \mathscr{G}_\mathbf{q}$ be a communication structure. Then, we use $\mathscr{C}_\sigma$,V_σ, E_σ and L_σ to denote the sets of components, nodes, links, and configurations of N_σ, respectively.

Example 2. (Communication structure): Consider the vector of quantities $\mathbf{q} = (x,y)$ representing the horizontal and vertical coordinates of a component. The domain $D_\mathbf{q}$ is the set of points where nodes can be placed. This domain can be described, for instance, by a discrete set of points or by union of rectangles. If there are no preferred positions, the elements of $D_\mathbf{q}$ are not comparable, therefore the order $\preceq_\mathbf{q}$ is a flat one, with $\perp$ being the minimum element. Given a communication structure $N(\mathscr{C},\mathbf{q},L)$, the set of configurations L captures all the admissible placements of the nodes in V. Since we do not assign any position to the links, for all $l \in L$ and for all links $e \in E$, $l(e) = \perp$[2]. The additional constraint that no two nodes occupy the same position requires that for all $l \in L$, and for all pair of nodes $u,v \in V$, $l(u) \neq l(v)$.

We introduce two scoping operators on configurations. Given a communication structure $N(\mathscr{C},\mathbf{q},L)$, the restriction of a configuration $l \in L$ to a subset of components $\mathscr{C}' \subseteq \mathscr{C}$, denoted by $l|_{\mathscr{C}'}$, is a function $f : \mathscr{C}' \to D_\mathbf{q}$ such that $f(c) = l(c)$ for all $c \in \mathscr{C}'$. In particular, $l|_V$ and $l|_E$ are the restrictions of a configuration l to the set of nodes and links, respectively. Given a vector of quantities $\mathbf{q}'$ obtained from $\mathbf{q}$ by projecting away some of the quantities, the projection of a configuration onto $\mathbf{q}'$ is denoted by $l[\mathbf{q}']$, and corresponds to ignoring the quantities not in $\mathbf{q}'$. We naturally extend these operators to sets of configurations. For instance, $L[(x)]|_V$ denotes the possible assignments of horizontal positions to nodes in Example 2.

As we will show in later sections, it is often necessary to compare specifications, platform instances and implementations. For instance, it is important to be able to order different specifica-

tions depending on how stringent the constraints are. Similarly, it is important to compare platform instances depending on their performance. Therefore, we define an ordering relation $\preceq_{\mathbf{q}}$ on the set of communication structures $\mathscr{G}_{\mathbf{q}}$ as follows:

Definition 9. *Given two communication structures $N_1, N_2 \in \mathscr{G}_{\mathbf{q}}$, $N_1 \preceq_{\mathbf{q}} N_2$ if and only if $\mathscr{C}_1 \subseteq \mathscr{C}_2$, and for all $l_1 \in L_1$ there exists $l_2 \in L_2$ such that for all $c \in \mathscr{C}_1$, $l_1(c) \preceq_{\mathbf{q}} l_2(c)$.*

The definition states that a communication structure N_2 "is better" than a communication structure N_1 if it contains more components and if it can perform at least as well as N_1.

Example 3. (Ordering of Communication Structures): Consider the vector of quantities $\mathbf{q} = (b,l)$ of Example 1. Figure 4.2 shows the relation between a set of communication structures. For the sake of simplicity, each communication structure in this example has only one configuration that is explicitly represented by the labels on the links in Figure 4.2. Each communication structure N_i has a set of components $\mathscr{C}_i \subseteq \{c_1, c_2, c_3, c_4, c_5\}$. The following relations are the interesting ones:

$N_7 \preceq_{(b,l)} N_4$. The two communication structure have the same set of components but $L_4 = \{l_4\}$ dominates $L_7 = \{l_7\}$, i.e. $l_7(c) \preceq_{(b,l)} l_4(c)$, $\forall c \in \mathscr{C}_4$. This is also the case for $N_8 \preceq_{(b,l)} N_6$, $N_5 \preceq_{(b,l)} N_2$, $N_5 \preceq_{(b,l)} N_3$, $N_2 \preceq_{(b,l)} N_1$ and $N_3 \preceq_{(b,l)} N_1$.

$N_7 \preceq_{(b,l)} N_5$. The components of N_7 are a subset of the components of N_5. The configuration of N_5 dominates the one of N_7 for the components that they have in common. This is also the case for $N_8 \preceq_{(b,l)} N_5$, $N_4 \preceq_{(b,l)} N_2$ and $N_6 \preceq_{(b,l)} N_3$.

N_4 and N_5 are incomparable. These two communication structures are incomparable because even if $\mathscr{C}_4 \subset \mathscr{C}_5$, the configuration of N_5 does not dominate the configuration of N_4.

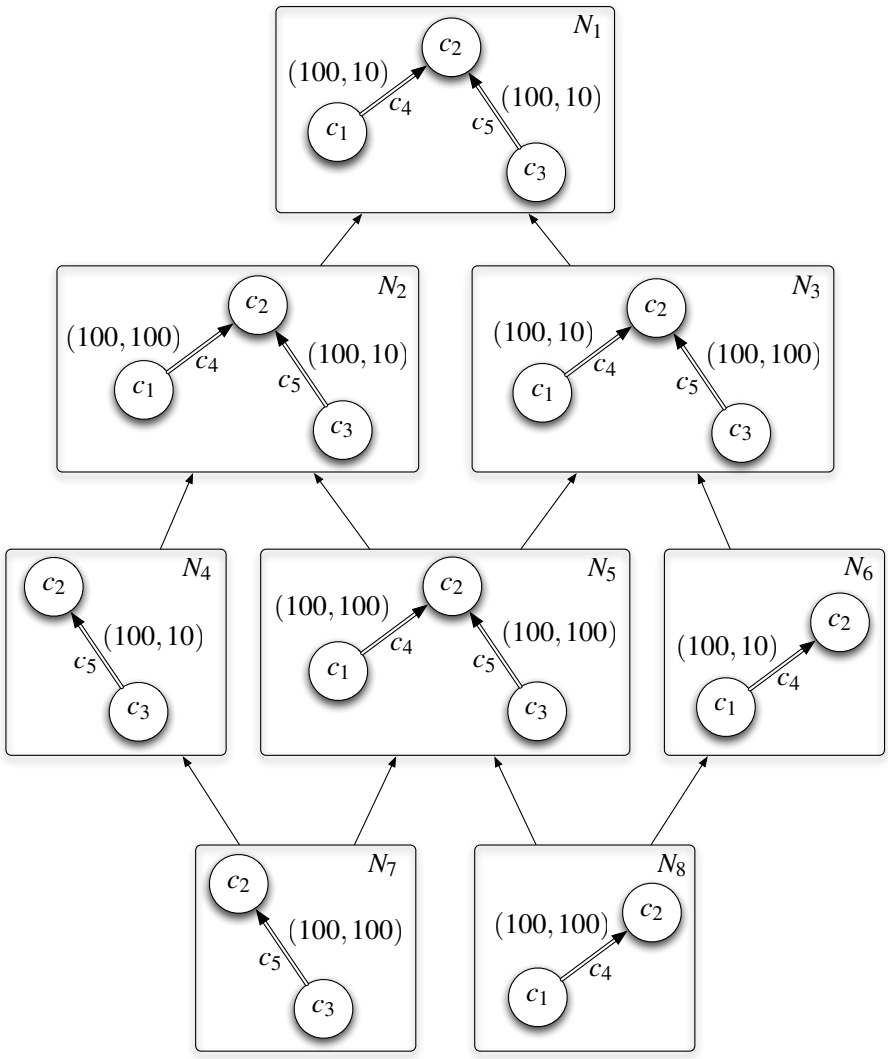


Figure 4.2: Hasse diagram of the partial order $\leq_{(b,l)}$ for subset of communication structures.

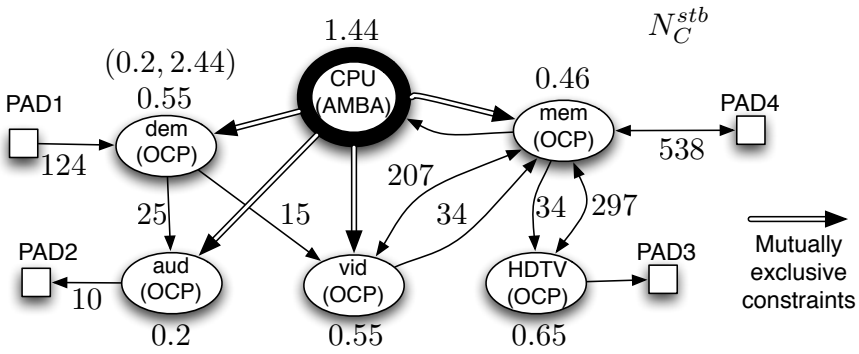


Figure 4.3: The system-level specification of a simplified Set-Top Box. Each core in the specification is annotated with and area in mm^2 and each arrow is annotated with a bandwidth constraint in MB/s.

We close this section with two examples of how communication structures are used to capture the end-to-end communication requirements of a communication synthesis problem. The first example is about on-chip communication while the second is about building automation networks.

Example 4. (Set-top-box communication requirements): Consider the simplified *Set-Top Box System* shown in Figure 4.3. The SoC *specification* contains six IP cores that exchange messages through a dozen of point-to-point channels and interact with the external environment through four major I/O connections (pads). The data input stream is processed by the demux core (`dem`) that sends an audio stream to the audio decoder and a video stream to the video decoder. The video decoder accesses the external memory through a memory controller. The memory is used both as an intermediate storage and to send the decoded stream to the display controller and HDTV encoder. Finally, a master CPU controls the operation of all the blocks and handles the interaction with the environment. Additional non-functional constraints are often part of the specification: e.g, the `dem` core must occupy position $(0.2, 2.44)$ (in millimeter); the `cpu` communicates with the other cores,

one at the time.

The *communication specification* is captured as communication structure $N_C^{stb} \in \mathscr{G}_{\mathbf{q}_C}$ where $\mathbf{q}_C = (x, y, a, \tau, b, h)$; nodes represent IP cores (that can be sources and/or destination of a communication) and have an associated position (x, y) in the Euclidean plane, an area a, and a type τ denoting the supported interface protocol; links represent distinct inter-core communications. Each links is associated with two quantities: a minimum average bandwidth b and a maximum latency h (not shown in the figure to avoid cluttering).

The position of the *dem* core is fixed at coordinates $(0.2, 1.44)$. Hence, each configuration $l \in L_C^{stb}$ must be such that $l(dem) = (0.2, 1.44, 0.55, OCP, \perp, \perp)$. Since there are no other floorplanning constraints, the position of the other IP cores can be determined during the synthesis process. Moreover, since the constraints between the CPU and the IP cores are mutually exclusive, for all $l \in L_C^{stb}[(b)]$, only one among $l((CPU, dem))$, $l((CPU, aud))$, $l((CPU, vid))$, $l((CPU, mem))$ can be different from zero.

Example 5. (Centralized control in buildings): Control algorithms are usually described in a centralized manner. In control theory jargon, the *plant* is the object to control. It governs the evolution of some physical variables, like temperature, that we want to steer to a desired value. To achieve this goal, a *controller* observes the physical variables and reacts with control variables that are sent back to the plant and that influence its evolution.

When the plant occupies a large area, the physical quantities must be observed in different points through *sensors*, and the control actions must be applied in different points by *actuators*. For instance, in large buildings, temperature must be sensed in each room and control actions must be sent to dampers that are located in different part of the Heat, Ventilation and Air Conditioning

(HVAC) system. Therefore, a network is used to connect sensors and actuators to the controller. Since the performance of the control algorithm are assessed assuming ideal communication, the network must provide reliable end-to-end communication with tight precise latency bounds (that depends on the time scale of the dynamics of the physical variables).

The communication specification is captured as a communication structure $N_C \in \mathscr{G}_{\mathbf{q}_C}$, $\mathbf{q}_C = (x, y, z, \tau, l, t, b, p)$, where $D_{(x,y,z)} = \mathbb{R}^3$ is the position in the Euclidean space, $D_\tau = \{sensor, actuator, controller\}$ is the type associated with a node, $D_l = D_t = D_b = \mathbb{R}$ are the maximum latency, message period and message length, respectively, associated with a communication requirement, and $D_p = [0, 1]$ is the maximum message error rate. In some case, the specification is restricted to centralized control, i.e. links in N_C can only connect sensors to controllers, and controllers to actuators.

Chapter 5

Building Complex Communication Architectures from Components

In this chapter we define how communication structures can be composed to build more complex communication structures. We define the composition of communication structures and introduce the concept of communication library and communication platform. The library elements need first to be instantiated. Instantiation consists in changing the identifiers of the nodes of a communication structure such that nodes belonging to different communication structures can be interconnected (simply by having the same identifier). Instances of library elements can be composed according to a composition rule. The composition rule is defined by: 1) an operator that given the quantities attached to the components of two communication structures define the quantities that should attached to the result of the composition, and 2) a relation that defines when a configuration is valid for a set of components.

5.1 Composition

To allow the incremental design of complex on-chip communication, we introduce two operations: *renaming* and *parallel composition*. The identifiers of two nodes in different sub-nets can be renamed to be the same to indicate either that the same node is used to implement both or the presence of an implicit connection between the two sub-nets at these nodes. A *renaming function* $r : U_V \rightarrow U_V$ is a bijection on the vertex universe. With R we denote the set of all renaming functions. Given a communication structure N and a renaming function r, with abuse of notation we use $r(N)$ to denote a new communication structure where the components have been renamed according to r.

The *composition* of two communication structures N_1 and N_2, denoted by $N_1 \| N_2$, results in a new communication structure N that contains the set of components $\mathscr{C}_1 \cup \mathscr{C}_2$. We define the operator $\|$ by two rules. The first rule is needed to establish how the configurations of the components being merged contribute to the formation of the ones of the combined entity. This rule is captured by a binary operator on the configuration universe, denoted $\oplus_{\mathbf{q}}$, which must satisfy the following properties. It must be commutative and associative such that the composition of communication structures will also satisfy these properties. This is important since we want the result of the composition to be independent of the order in which communication structures are instantiated and composed. Further, if $l_1 : \mathscr{C}_1 \rightarrow D_{\mathbf{q}}$ and $l_2 : \mathscr{C}_2 \rightarrow D_{\mathbf{q}}$, then $l = l_1 \oplus_{\mathbf{q}} l_2$ must be such that $l : \mathscr{C}_1 \cup \mathscr{C}_2 \rightarrow D_{\mathbf{q}}$. This operator is defined on sets of configurations as follows. Let $L_1 \subseteq [\mathscr{C}_1 \rightarrow D_{\mathbf{q}}]$ and $L_2 \subseteq [\mathscr{C}_2 \rightarrow D_{\mathbf{q}}]$, then $L_1 \oplus_{\mathbf{q}} L_2 = \{l_1 \oplus_{\mathbf{q}} l_2 | l_1 \in L_1 \wedge l_2 \in L_2\}$. A second rule restricts the legal compositions by forcing the composed structure to satisfy certain properties. This rule, that defines a class of communication structures the result of the composition must belong to, is given by a relation between the components and the configurations and it is denoted by $\mathscr{R} \subseteq 2^{U_{\mathscr{C}}} \times U_{\mathbf{q}}$.

Example 6. (Composition of communication specifications):

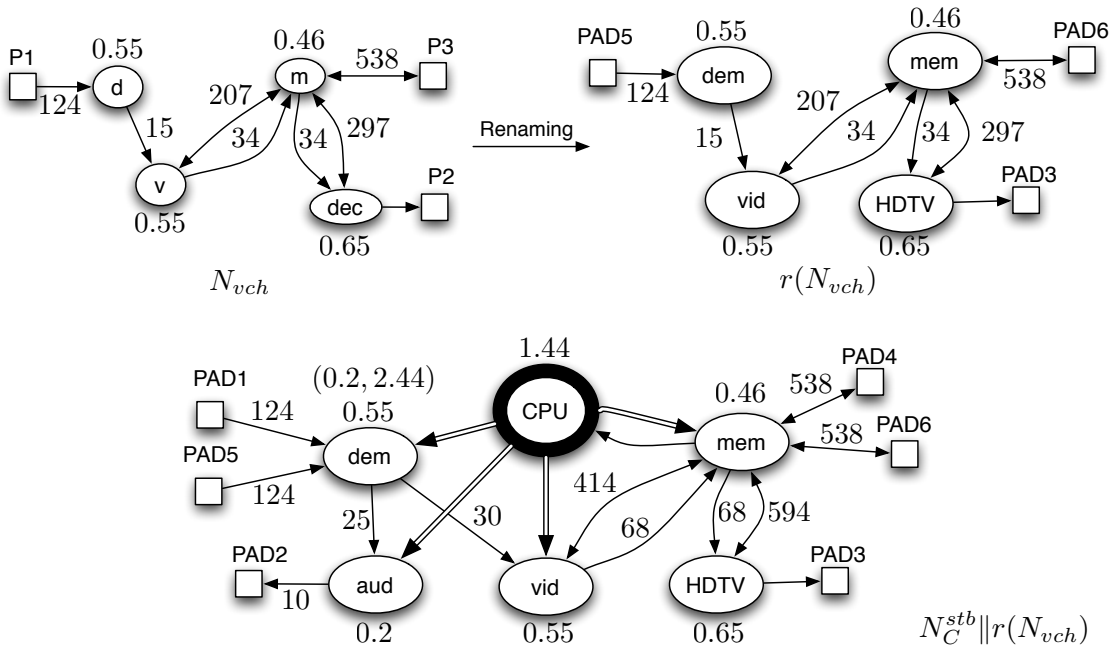


Figure 5.1: Example of parallel composition of networks: the set-top box is expanded by adding a video channel and an extra off-chip memory bank.

Definition 10. *Given a binary operator* $\oplus_{\mathbf{q}}$ *and a composition rule* $\mathscr{R}$*, and two communication structures* N_1 *and* N_2 *belonging to* $\mathscr{G}_{\mathbf{q}}$*, their composition is* $N_1 \|^{\mathscr{R}}_{\mathbf{q}} N_2 = N \in \mathscr{G}_{\mathbf{q}}$*, where* $\mathscr{C} = \mathscr{C}_1 \cup \mathscr{C}_2$*,* $L = \{l \in L_1 \oplus_{\mathbf{q}} L_2 | (\mathscr{C}, l) \in \mathscr{R}\} \neq \emptyset$*; the composition is not defined if* $L = \emptyset$*.*

We want to add an extra video channel to our set-top box chip by *reusing* the already instantiated IP cores. In Fig. 5.1, N_{vch} is a communication structure capturing the communication requirements of a set-top-box video channel. To reuse the same IP cores, we rename the nodes according to a renaming function r such that $r(d) = dem$, $r(m) = mem$, $r(v) = vid$ and $r(dec) = HDTV$. Since the new video channel must be displayed on the same device, $r(P2) = PAD3$ forces the same output pad to be reused. For the demodulator input, though, we need an additional pad. We also add a new pad to connect a second memory bank to the memory controller. Fig. 5.1 shows

the result of the composition $N_C^{stb} \|_{\mathbf{q}_C}^{\mathscr{R}_C} r(N_{vch})$. Intuitively, we have added the bandwidths of common requirements and we have restricted the position of the `dem` core. More precisely, we need to define the operator $\oplus_{\mathbf{q}_C}$. Given two communication structures structures $N_1,\ N_2 \in \mathscr{G}_{\mathbf{q}_C}$, let $l_1 \in L_1$ and $l_2 \in L_2$ be two configurations. The configuration $l = l_1 \oplus_{\mathbf{q}_C} l_2$ is defined as follows:

- there is no "interference" between components not shared by N_1 and N_2, i.e $l(c) = l_1(c)$ for all $c \in \mathscr{C}_1 \setminus \mathscr{C}_2$, and $l(c) = l_2(c)$ for all $c \in \mathscr{C}_2 \setminus \mathscr{C}_1$;
- common nodes must be "compatible", meaning that they must agree on the positions and interfaces:
$$\forall c \in V_1 \cap V_2,\ l(c) = \begin{cases} l_1(c) & if\ l_1(c) = l_2(c) \\ \perp^6 & if\ l_1(c) \neq l_2(c) \end{cases}$$
(notice that it is sufficient to have some compatible configurations for the composition to be defined);
- for all $c \in E_1 \cap E_2$, $l[(b)](c) = l_1[(b)](c) + l_2[(b)](c)$ and $l[(h)](c) = \min\{l_1[(h)](c), l_2[(h)](c)\}$.

We now define the composition rules. We want each node to have an assigned position and interface protocol, therefore we define the following rule: $\mathscr{R}_C^v = \{(\mathscr{C}, l) \in 2^{U_\mathscr{C}} \times U_{\mathbf{q}_C} | \forall v \in \mathscr{C},\ \forall q \in \{x, y, a, \tau\},\ l[(q)](v) \neq \perp\}$. Also assume, for instance, a designer is given an area budget ν_a for the IP cores on a chip then another composition rule can be stated as follows:

$$\mathscr{R}_C^a = \left\{ (\mathscr{C}, l) \in 2^{U_\mathscr{C}} \times U_{\mathbf{q}_C} \,\middle|\, \sum_{c \in V} l[(a)](c) \leq \nu_a \right\}$$

Therefore, the composition rule is $\mathscr{R}_C = \mathscr{R}_C^v \cap \mathscr{R}_C^a$. We give examples of other rules in Section 5.2.

5.2 Platforms

In Section 2.2.1, we defined a platform as the closure of a set of library elements under the operation of parallel composition. These elements either have a corresponding implementation that is ready to be used or can be synthesized by tools operating at a lower level of abstraction. A *communication library* $\mathscr{L}$ is a collection of communication structures, i.e. $\mathscr{L} \subset \mathscr{G}_{\mathbf{q}}$. Even though the platform could be defined in a recursive way as in Definition 5, here we give an iterative definition that is easier to understand.

Definition 11. *The* communication platform *generated by the communication library* $\mathscr{L}$ *under composition* $\|_{\mathbf{q}}^{\mathscr{R}}$ *is*

$$\langle \mathscr{L} \rangle = \{ N(\mathscr{C}, \mathbf{q}, L) = r_1(N_1) \|_{\mathbf{q}}^{\mathscr{R}} \dots \|_{\mathbf{q}}^{\mathscr{R}} r_m(N_m) \mid r_i \in R',\ N_i \in \mathscr{L}, L \neq \emptyset, m \geq 1 \}$$

where $R' \subseteq R$ *is a set of valid renaming functions. An element* $N \in \langle \mathscr{L} \rangle$ *is called a* communication platform instance.

Notice that the set of admissible renaming functions is restricted to a subset R' of all renaming functions. For instance, we may forbid to rename a router node to a source node. The definition of a platform is very general. We have already seen a platform in Examples 4 and 5 that is the specification platform. The library in this case contains one simple communication structure that is a point-to-point connection between a source and a destination node. The specification is a platform instance of the specification platform.

The set of alternative implementations are also captured by a library of components and composition rules, i.e. by a platform. In this case we are interested in all those platform instances that are possible candidates for implementing the specification. Let $\mathbf{q}_P$ be the vector of quantities

associated with the communication structures in the platform. While the vector of quantities $\mathbf{q}_C$ associated with the specification represents the required services, the vector $\mathbf{q}_P$ represents the *capabilities* of the components in the platform, i.e what the components can do. For instance, a pair of quantities (x,y) associated with a component in the specification denote the coordinates where a component *must* be located, whereas the same quantities associated to a component of a platform instance denote the coordinates where a component *can* be located. Similarly, while quantity b represents the required minimum bandwidth between a source and a destination, we will use γ to denote the maximum bandwidth that a link can support, i.e. the link *capacity*.

Example 7. Communication Platform for On-Chip Communication: The vector of quantities that characterizes the on-chip communication platform is $\mathbf{q}_P = (x, y, \tau, in, out, \gamma)$. Each node has an associate position (x,y), a type τ, two multisets *in* and *out* of input and output port interfaces, respectively. Each link is associated with a capacity γ, i.e. the maximum bandwidth that it can sustain, a singleton set *in* denoting the input interface of the target not and a singleton *out* denoting the output interfaces of the origin node.

The definition of composition $\|_{\mathbf{q}_P}^{\mathscr{R}_P}$ captures the set of valid communication architectures (i.e. communication platform instances) that can be obtained out of the communication library. The rest of this example shows the flexibility that our framework provides in defining the set of communication structures that can be obtained by composition of library elements.

Consider a communication library whose elements are nodes and links. Fig. 5.2 shows a communication library $\mathscr{L}$ and two possible platform instances N_P^1 and N_P^2. Library $\mathscr{L}$ contains the following set of components: a bus node and a bidirectional bus-segment connecting two bus nodes; a mesh node and two mesh links for East-West connection and North-South connection,

respectively. It contains also a set of interface communication structures to connect IP cores to bus nodes and mesh nodes. Each node has an associated multi-set of input interfaces *in* and output interfaces *out* (depicted as filled and non-filled shapes attached to nodes in Fig. 5.2). A link connects an output interface of a node to an input interface of another node. Mesh links have an associated maximum capacity γ^M_{max} while bus-segments (including the link between an IP core and a bus node) have an associated interval of capacities $[0, \gamma^B_{max}]$ corresponding to different configurations. We introduce two more quantities i_x and i_y for mesh structures that are the row and column index of a node. Now, we state a set of composition rules such that the only platform instances that are valid in this platform are either busses or meshes:

1. The number of bus nodes can be at most the number of bus segments minus one. This ensures that the topology of a bus is a collection of trees. Also, since a bus node has only two bidirectional ports to connect to other bus nodes,each bus is a chain of IP cores (as shown by the platform instance N^1_P).

2. An East-West mesh link can connect two mesh nodes (u,v) only if $l[(i_x,i_y)](u) = (i,j)$ and $l[(i_x,i_y)](v) = (i,j+1)$; a North-South mesh link can connect two mesh nodes (u,v) only if $l[(i_x,i_y)](u) = (i,j)$ and $l[(i_x,i_y)](v) = (i+1,j)$ (as shown by the platform instance N^2_P).

3. Each bus configuration l must be such that the sum of the capacities of the links connecting the cores to the bus is less then γ^B_{max}. This rule restricts the possible configuration of a bus, and it models the fact that the total bus capacity is shared among all IP cores connected to it.

These three rules define $\mathscr{R}_P$ for this specific platform.

Remark 1. (Communication Platform for Building Automation): In building automation sys-

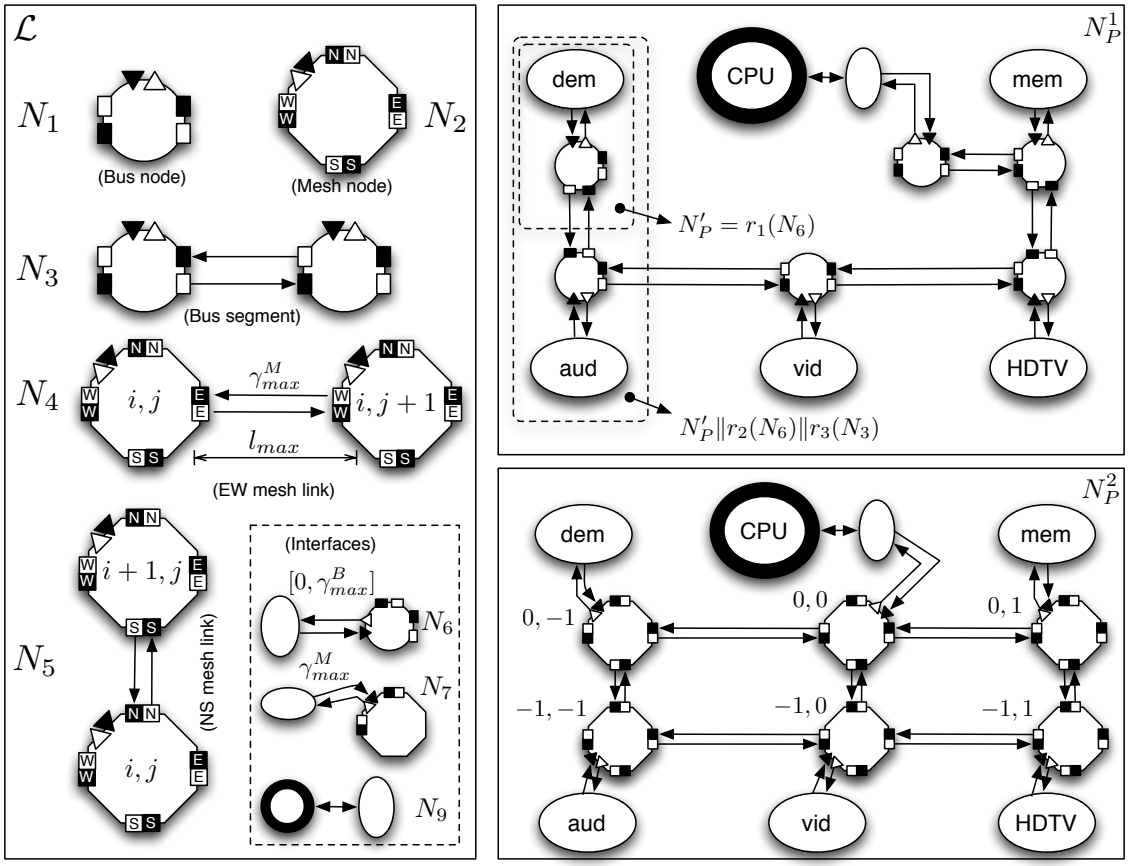


Figure 5.2: Example of a library $\mathscr{L}$ for on-chip communication and two alternative implementations for the set-top box example based on composing elements instanced from $\mathscr{L}$.

tems, networks are usually hierarchical. In Chapter 8 we give some detailed explanation of this interesting applications. Sensors and actuators are locally connected by buses. Each bus is connected to a switch that, in turn, is connected to the central controller. In other applications (see Section 8.4), the network is a tree where sensors and actuators are the leaves.

The vector of quantities that characterizes the platform is $\mathbf{q}_P = (x, y, z, \tau, in, out, \gamma, p)$ where (x, y, z) represent the position in the Euclidean space, τ is the type of a component, γ is the capacity of a link in maximum number of messages per seconds and p is the minimum mes-

sage error rate. The two quantities *in* and *out* have the same meaning as in the case of on-chip communication.

The composition rules that we consider for buses is the same as in the case of on-chip communication. Other rules are also reported in Chapter 8.

Chapter 6

Communication Synthesis for Networked Systems

In this chapter we present a general formulation of the communication synthesis problem where:

- the input specification consists of a set of end-to-end communication constraints captured by a communication structure N_C.
- The platform is implicitly described by a communication library $\mathscr{L}$ that contains the communication components and the composition rule $\|_{\mathbf{q}}^{\mathscr{R}}$.
- The output consists of a communication structure that satisfies all end-to-end constraints in N_C, uses only components in $\mathscr{L}$, and such that the total cost of the implementation is minimized.

To define the optimization problem we first relate an implementation to a communication structure that represents the possible communication scenarios (i.e. the specifications) that the implementation can support. Also, we relate an implementation with a platform instance. The relations are defined by abstraction functions. Finally, we define a general optimization problem to explore the entire design space.

6.1 Relations Among Communication Structures

The same specification can be implemented by many platform instances. On the other hand, the same platform instance can implement a variety of different specifications. For a given platform instance, deriving an implementation that implements a given specification is called *mapping* in the platform-based design terminology. The implementation is a refinement of both the specification and the platform instance. Being a refinement means that the implementation contains more details, that are captured not only by the number of components of the communication structure defining the implementation, but also by the vector of quantities. In our example, the implementation of a communication specification is a communication structure derived from a platform instance by adding the information regarding the routing of packets and the latency. Routing is captured by a quantity ρ called *transfer table*. To define ρ, we introduce another quantity λ with domain D_λ representing a name attached to each component. To simplify the notation, we assume that this quantity implicitly belongs to any vector of quantities. For a component c, we denote its name with $\lambda(c)$ (instead of the more cumbersome notation $l[\lambda](c)$, simply because each configuration assigns the same name to the same component). The name of a component c is different from its *identifier* which is denoted by the symbol c itself. In particular, the renaming function does not

change the name of a component but only its identifier (this is the main reason to distinguish them). The domain of ρ is $D_\rho = 2^{D_\lambda \times D_\lambda \times D_\lambda}$. Therefore, a transfer table is a set of triples $(\lambda(s), \lambda(d), \lambda(v))$ where $\lambda(s)$ and $\lambda(d)$ are the names associated with the source and destination of the packets, respectively, and $\lambda(v)$ is the name of a node in the communication structure. Each triplet specifies the name of the next hope for each packet that arrives at the node from a given source in its transit to a given destination. An implementation is a communication structure $N_I(\mathscr{C}_I, \mathbf{q}_I, L_I)$ where $\mathbf{q}_I$ contains all the quantities in $\mathbf{q}_P$ with the addition of quantity ρ and other implementation dependent quantities (like for instance latency h as defined in Section 4.1). The vector of quantities $\mathbf{q}_I$ also contains the quantities coming from the specification domain.

During the refinement from the specification and the platform toward the implementation, some quantities are added that are dependent on the value of other quantities. For instance, the latency information associated to the components of an implementation depends on the actual network traffic which is known only after mapping. This quantity is *derived* from the others. However, if it is measured in number of hops, then it is an independent quantity and each link has a latency equal to one while each node has a latency equal to zero. Another example of derived quantity is the bit error rate over wireless communication links that depends on the interference from other nodes in a communication structure. These quantities depend on the abstraction of the specific protocol that is used at the network level and at the lower level of abstraction (e.g., Layer 2 of the OSI protocol stack [8]). For instance, packets traveling on a BUS incur in different latencies if the protocol is AMBA rather then OCP. To compute derived quantities, that are often used to model specification dependent metrics, we formally introduce the notion of a *model*. Let q denote a derived quantity. Two cases can arise. If the configurations of a component c of a communication

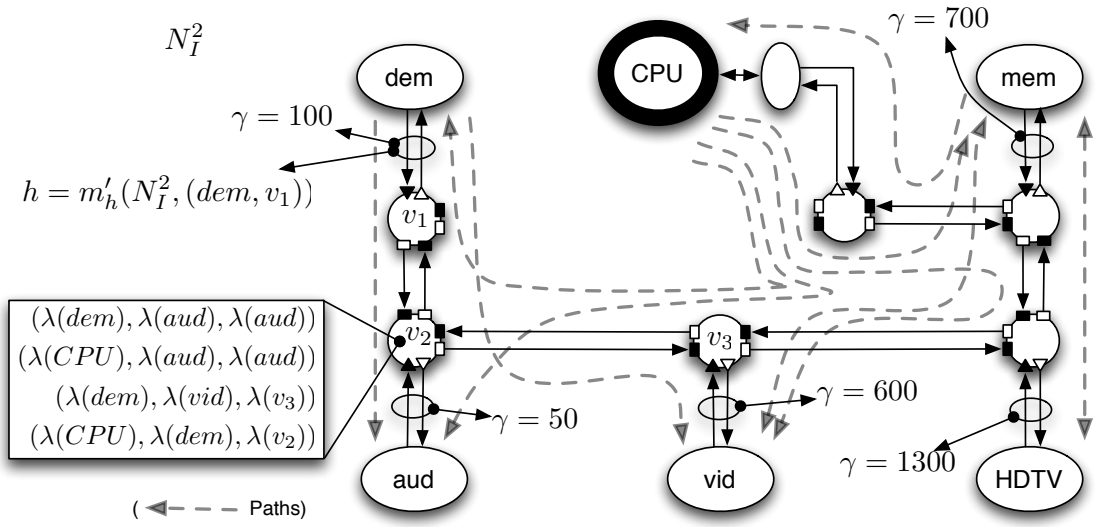


Figure 6.1: Example of communication implementation for the set-top box.

structure contain enough information to determine the value of q, then the quantity is *directly derived* from a function $m_q : D_{\mathbf{q}} \rightarrow D_q$, and we call m_q a *direct model* for q. For instance, the power dissipated on a link is directly derived from its communication bandwidth. If the computation of the value of q depends not only on the configuration but also on the other components and how they are configured in the communication structure, then the quantity is *indirectly derived* from a function $m'_q : \mathscr{G}_{\mathbf{q}} \times U_{\mathscr{C}} \rightarrow D_q$, and we call m'_q an *indirect model* for q. During the refinement process, some quantities can be determined by models (like latency, for instance) while independent quantities are computed by optimization algorithms (like transfer tables, for instance).

Example 8. Transfer tables and latency: Fig. 6.1 shows a bus-based implementation of the set-top box example of Fig. 4.3. The light-gray arrows represent paths in the communication structures. The paths are implicitly defined by the transfer tables of each bus-node. For instance, the transfer table of node v_2 contains an element $(\lambda(CPU), \lambda(dem), \lambda(v_1))$ meaning that a packet from the `CPU` core to the `dem` core must be sent to v_1. The transfer table information can be used at a lower

abstraction level to optimize the bus circuitry (e.g. decoders and multiplexers) or even to segment the bus and insert bus bridges.

The latency to access the bus for each IP core depends on the actual set of components and the bus configuration. When refining the platform instance N_P^1 shown in Fig. 5.2 into the implementation N_I^2, shown in Fig. 6.1, a range of latencies $[h_{min}, h_{max}]$ is first considered for the access link (dem, v_1). This range can be computed by a best and worst case analysis of a bus. An indirect model m'_h is used to restrict the range of latencies depending on the actual specification mapped on the implementation. Therefore, the indirect model becomes part of a composition rule that can be stated as follows:

$$\mathscr{R}_I^h = \{(\mathscr{C}, l) \in 2^{U_\mathscr{C}} \times U_{\mathbf{q}_I} \mid l[(h)](c) = m'_h((\mathscr{C}, \mathbf{q}_I, \{l\}), c),\ \forall c \in \mathscr{C}\}$$

The latency of an end-to-end communication is the sum of the latencies of all components in the path. Notice that in this example of bus model we lump the latencies on the access link to the bus and assign a latency equal to zero to each bus segment.

Assuming a 128 bit-wide bus and $200Mhz$ clock frequency, the maximum theoretical throughput is $1.6GB/s$. Hence, we can assign capacities to the links connecting the cores to the bus nodes as shown in Fig. 5.2. Given the capacity assignment, the communication implementation can support a larger set of specifications than the one in Fig. 4.3. For instance, the throughput of the `dem` core can be increased up to $100MB/s$. In the rest of this section we define precisely the set of specifications that an implementation can support.

Other examples of composition rules are the following. For each configuration l of a communication structure $N_I(\mathscr{C}, \mathbf{q}_I, L)$, the bandwidth on each link must be less than or equal to the capacity of the link, i.e. $l[(b)] \leq l[(\gamma)]$, A possible additional rule is *deadlock freedom*, which

requires the channel dependency graph of N_I to be acyclic [33].

Remark 2. (On transfer tables): One may think that transfer tables are only needed for routers or switches that have multiple input and multiple output ports. Their task is to observe the packet header and look up in a table to find the output port to forward the packet to. In the case of buses, a transfer table is not necessary because a data that is posted on the bus gets broadcast to all the other bus participants. Nevertheless, in a refinement step, transfer tables associated to bus nodes can be used to further optimize the bus by distributing simple muxes and demuxes on bus nodes and/or segmenting the bus.

To define a general synthesis problem for communication structures, besides defining the cost of a communication implementation, we also need to define the constraints that the implementation must satisfy. The constraints, that we will explain in details in Section 6.2, come from the specification and from the platform. In short, the implementation must satisfy the communication constraints captured by the specification N_C and must be a composition of library elements (i.e. must belong to the platform). Therefore, we need to relate an implementation to the set of specifications that it can correctly implement, and to a platform instance. We define these relations by abstraction functions as follows.

Given an implementation N_I, a path of length n is a sequence of n links $\pi = (e_1, \ldots, e_n)$ such that $e_i = (v_{i-1}, v_i)$. Even if the topology is such that a path can be found between two nodes in N_I, packets may not be able to flow through the path. A *real path* from a source node s to a destination node d according to a configuration $l \in L_I$ is such that $v_0 = s$, $v_n = d$ and $\forall e_i \in \pi$, $\exists(\lambda(s), \lambda(d), \lambda(v_i)) \in l[(\rho)](v_{i-1})$.

The communication structure N_C that characterizes the set communication specifications

that an implementation N_I can correctly implement is given by the a function $\Pi : \mathscr{G}_{\mathbf{q}_I} \rightarrow \mathscr{G}_{\mathbf{q}_C}$ called *path abstraction*. The path abstraction depends on the vector of quantities $\mathbf{q}_I$ and $\mathbf{q}_C$ and it is specified by defining the two following abstractions:

Component abstraction . The nodes $V_C \subseteq \mathscr{C}_C$ are a subset of the nodes $V_I \subseteq \mathscr{C}_I$. The set of links that belongs to $\mathscr{C}_C$ are defined as follows. There exists a link $(s,d) \in \mathscr{C}_C$ if and only if there exists a real path from s to d in N_I according to some configuration $l_I \in L_I$. Notice that the configuration bears the information regarding the transfer tables and therefore it determines the set of real paths in an implementation.

Configuration abstraction . The set of configuration L_C is determined by a function $N_I \rightarrow L_C$ that is not necessarily injective.

We give a specific example of path abstraction for the case of on-chip communication. A similar abstraction can also be defined for building automation systems.

Example 9.: Given an an on-chip communication implementation N_I, the communication structure N_C characterizing the set of specifications that N_I can correctly implement is given by the path abstraction $\Pi : \mathscr{G}_{\mathbf{q}_I} \rightarrow \mathscr{G}_{\mathbf{q}_C}$, which is defined by the following construction:

- the nodes of V_C of N_C are the nodes of V_I of N_I that are IP cores.
- a configuration l_C belongs to L_C if an only if there exists a configuration $l_I \in L_I$ such that the following conditions are satisfied:
  1. $l_C(c) = l_I(c)$ for all $c \in V_C$
  2. for all links $e \in \mathscr{C}_I$

$$\sum_{(s,d) \in \mathscr{C}_C : e \in \pi(s,d)} l_C[(b)](s,d) = l_I[(b)](e)$$

i.e. the sum of the bandwidth of all end-to-end communication that share a link e must be equal to the bandwidth specified in the implementation.

3.

$$\sum_{e \in \pi(s,d)} l_I[(h)](e) = l_C[(h)](s,d)$$

i.e the end-to-end latency from a source to a destination is equal to the sum of the latencies along the path.

To relate implementations and platform instances we introduce the abstraction relation $\Psi : \mathscr{G}_{\mathbf{q}_I} \rightarrow \mathscr{G}_{\mathbf{q}_P}$ that removes the transfer tables and the latency quantities, i.e. given an implementation N_I it returns $\Psi(N_I) = N_P(\mathscr{C}_I, \mathbf{q}_P, L_I[\mathbf{q}_P])$. Given a specification N_C and a platform $\langle \mathscr{L} \rangle$, implementation N_I must satisfy two constraints: $N_C \leq_{\mathbf{q}_C} \Pi(N_I)$ and $\Psi(N_I) \in \langle \mathscr{L} \rangle$. When the implementation is constrained to have a specific topology such as a mesh or a torus, an additional condition $\Psi(N_I) \leq_{\mathbf{q}_P} N_P$ must be satisfied where N_P is the platform instance capturing the specific topology.

Remark 3. (On the generality of the path abstraction): Notice that in defining the implementation we only need the the type of the nodes, their interfaces and their transfer tables. Based on these quantities, we can derive the other constraints. In particular, we can compute the real paths of an implementation and therefore we can check the end-to-end requirements and the local constraints on nodes and links.

6.2 A General Optimization Problem

Our objective is to find an implementation N_I that minimizes a given cost function $F : \mathscr{G}_{\mathbf{q}_I} \rightarrow R_+$. We assume that the cost function is monotonic, i.e. $N_1 \leq_{\mathbf{q}_I} N_2 \Rightarrow F(N_1) \leq F(N_2)$. This is a reasonable assumption since a less performing communication structure should also cost less. First, we formulate the problem of configuring a platform instance N_P to implement a specification. The communication synthesis problem can be stated as follows:

$$
\begin{aligned}
\texttt{PR1}(N_P): \quad & \min_{\mathscr{C}_I, L_I} && F(N_I) \\
& \textit{subject to} && N_C \leq_{\mathbf{q}_C} \Pi(N_I), && (6.1)\\
& && \Psi(N_I) \in \langle \mathscr{L} \rangle && (6.2)\\
& && \Psi(N_I) \leq_{\mathbf{q}_P} N_P && (6.3)\\
& && (C_I, l_I) \in \mathscr{R}_I,\ \forall l_I \in L_I && (6.4)
\end{aligned}
$$

Constraints 6.1 and 6.2 require N_I to implement the specification and to be a refinement of a platform instance. Constraint 6.3 requires the implementation to be contained in the performance envelope of the given platform instance N_P and Constraint 6.4 requires the implementation to satisfy the rules defined at the implementation level (e.g. deadlock freedom). Let Alg be a hypothetical algorithm that solves problem PR1 exactly. Given a library $\mathscr{L}$, platform $\langle \mathscr{L} \rangle$ can be explored by using Alg to solve problem PR1 for each $N_P \in \langle \mathscr{L} \rangle$. The following lemma relates the cost of the solution to problem PR1 for different platform instances.

Lemma 1. *Let N_C be a specification, $N_{P,1}$ and $N_{P,2}$ two platform instances such that $N_{P,1} \leq_{\mathbf{q}_P} N_{P,2}$. Let $N^*_{I,1}$ and $N^*_{I,2}$ be the implementations found by Alg for platform instances $N_{P,1}$ and $N_{P,2}$,*

respectively. Then $F(N^*_{I,2}) \leq F(N^*_{I,1})$.

By contradiction. . Since $N^*_{I,1}$ solves problem PR1, $N_C \leq_{\mathbf{q}_C} \Pi(N^*_{I,1})$. Also, $\Psi(N^*_{I,1}) \leq_{\mathbf{q}_P} N_{P,1} \leq_{\mathbf{q}_P} N_{P,2}$, therefore, $N^*_{I,1}$ is a solution of PR1 for $N_P = N_{P,2}$. Suppose that $F(N^*_{I,1}) < F(N^*_{I,2})$. This condition would contradict the optimality of Alg, since we found a solution that has a cost less than $F(N^*_{I,2})$. Therefore $F(N^*_{I,2}) \leq F(N^*_{I,1})$ must hold. □

According to Lemma 1, if we can find the greatest element $\overline{N_P}$ of $\langle \mathscr{L} \rangle$ with respect to the ordering relation $\leq_{\mathbf{q}_P}$, then the solution of problem PR1 with $N_P = \overline{N_P}$ is the best communication structure among all possible platform instances. Unfortunately, such greatest element is not guaranteed to exist in any given platform. Hence, instead of looking for it, we can look for an upper bound $N_P^{\langle \mathscr{L} \rangle}$ of $\langle \mathscr{L} \rangle$ (which is not required to belong to the platform). The existence of an upper bound is related to the platform being finite (i.e. containing a finite number of platform instances). A communication structure is finite if the set of its components is finite. A library is finite if it is a finite set of finite communication structures.

Proposition 1. *Given a vector of quantities* $\mathbf{q}$ *such that each quantity is either finite or bounded, a finite library* $\mathscr{L} \in \mathscr{G}_{\mathbf{q}}$ *and a finite set of valid renaming function* $R' \subset R$*, for any composition rule* $\mathscr{R}$ *and operator* $\oplus_{\mathbf{q}}$*, there exists an upper bound* $N^{\langle \mathscr{L} \rangle} \in \mathscr{G}_{\mathbf{q}}$ *of the communication platform* $\langle \mathscr{L} \rangle$ *with respect to the ordering relation* $\leq_{\mathbf{q}}$.

By construction. . Any platform instance is finite. A platform instance can be written as $N = r_1(N_1) \|^{\mathscr{R}}_{\mathbf{q}} \cdots \|^{\mathscr{R}}_{\mathbf{q}} r_k(N_k)$, for $r_i \in R'$ and $N_i \in \mathscr{L}$. Because R' is finite and the library is also finite, N is finite. Let $\bar{\mathscr{C}}$ be the largest set of components that any platform instance can contain. Let the vector of quantities $\mathbf{q}$ be partitioned in the vector of finite quantities $\mathbf{q}'$ and the set of bounded quantities $\mathbf{q}''$.

The domain $D_{\mathbf{q}''}$ has a greatest element v'' that is the vector of all greatest elements of the quantities in $\mathbf{q}''$. Now, let the set of values for $\mathbf{q}$ be:

$$\bar{D}_{\mathbf{q}} = \{v \in D_{\mathbf{q}} \mid v[\mathbf{q}'] \in D_{\mathbf{q}'} \wedge v[\mathbf{q}''] = v''\}$$

We define the communication structure

$$\bar{N}(\bar{\mathscr{C}}, \mathbf{q}, [\bar{\mathscr{C}} \rightarrow \bar{D}_{\mathbf{q}}])$$

and we prove that $N^{\langle\mathscr{L}\rangle} = \bar{N}$. Let $N(\mathscr{C}, \mathbf{q}, L)$ be a platform instance. By construction $\mathscr{C} \subseteq \bar{\mathscr{C}}$. We need to prove that for each $l \in L$ there exists a configuration $l' \in [\bar{\mathscr{C}} \rightarrow \bar{D}_{\mathbf{q}}]$ such that $l(c) \preceq_{\mathbf{q}} l'(c)$ for all components in $\mathscr{C}$. By construction, there exists a configuration $l' \in [\bar{\mathscr{C}} \rightarrow \bar{D}_{\mathbf{q}}]$ such that $l[\mathbf{q}'](c) = l'[\mathbf{q}'](c)$. Also for this configuration, $l[\mathbf{q}''](c) \preceq_{\mathbf{q}''} v'' = l'[\mathbf{q}''](c)$. Therefore, there exists a configuration $l' \in [\bar{\mathscr{C}} \rightarrow \bar{D}_{\mathbf{q}}]$ such that $l(c) \preceq_{\mathbf{q}} l'(c)$ for all $c \in \mathscr{C}$. From the generality of N it follows that $\bar{N}$ is an upper bound of $\langle\mathscr{L}\rangle$, i.e. $N^{\langle\mathscr{L}\rangle} = \bar{N}$. □

Assuming that the upper bound can be constructed, it follows from Lemma 1 and Proposition 1 that in order to solve the communication synthesis problem we need to solve the optimization problem $\texttt{PR2} \equiv \texttt{PR1}(N_P^{\langle\mathscr{L}\rangle})$. In general the upper bound $N_P^{\langle\mathscr{L}\rangle}$ does not satisfy the composition rules $\mathscr{R}_P$ (in fact, these rules are not taken into account by the constructive proof of the upper bound itself). Constraint 6.2 makes sure that the final implementation is a refinement of a platform instance. The solution N_I^* to problem PR2 is the best communication structure that implements the specification among all possible implementations that can be constructed from $\mathscr{L}$ through composition. Notice that, once the upper bound has been found, Constraint 6.2 is the only constraint that depends on the library $\mathscr{L}$. Thus, the properties of the optimization problem can potentially depend on the library and consequently, an algorithm that solves the communication synthesis problem efficiently

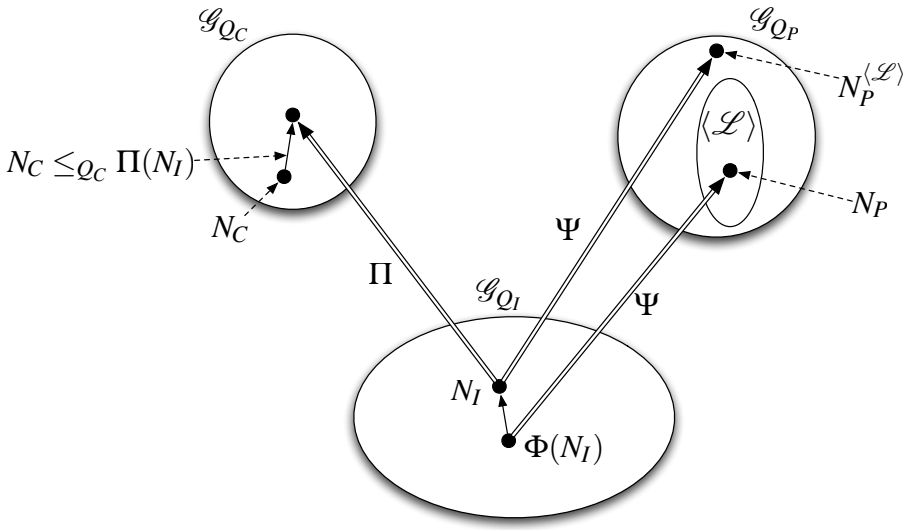


Figure 6.2: Summary of the procedure to define problem PR2.

could also depend on the library.

Figure 6.2 summarizes the main results of this section. Given the upper bound $N_P^{\langle\mathscr{L}\rangle}$, we define an implementation N_I with the same set of components and capacity configurations such that the transfer tables are still to be decided. Their values are decided such that the abstraction $\Pi(N_I)$ corresponds to a specification that at least satisfies the given specification. Notice that $\Pi(N_I) = \Pi(\Phi(N_I))$ since N_I and $\Phi(N_I)$ have the same set of real paths by definition. On the other hand, once the unused components are removed from N_I, the abstraction $\Psi(\Phi(N_I))$ must be a platform instance.

Part III

Applications

Chapter 7

On-Chip Communications

With the advances of IC technology, global interconnects have become the dominant factor in determining chip performance: they are not only becoming responsible for a larger fraction of the overall delay and power dissipation but exacerbate also design problems such as noise coupling, routing congestion, and, timing closure, thereby imposing severe limitations on design productivity [77]. Because of these characteristics, most VLSI circuits can be considered *distributed systems*, a fact that challenges traditional design methodologies and the electronic design automation tools that are based on them. Systems-on-Chip (SoCs) are typically designed by assembling intellectual property (IP) components from different vendors and/or different divisions of the same company in the attempt of reducing time-to-market by reusing pre-designed and pre-verified elements. However, since these components are designed independently, the assembly step is often a challenging problem that requires the design of communication interfaces to match different protocols and data parallelism and the routing of global interconnect wires to meet the constraints imposed by the target clock period.

The Open Core Protocol (OCP) [89] tackles this problem by defining a standard open-domain interface with which IP cores should comply to allow fast integration using appropriate interconnect architectures. While there is no intrinsic limitation on the interconnect architecture for OCP, most designers rely on traditional bus architectures so that pre-designed components can be used. In this domain, proprietary protocols such as the ARM AMBA BUS and the IBM CoreConnect are popular among SoC designers making the adoption of a universal standard difficult at best.

We argued that SoCs are distributed systems. For this reason, bus architectures may not be always ideal; in fact, scalable, multi-hop, packet-switched Networks-on-Chip (NoCs) have been proposed in a set of seminal papers [57, 34, 19] as a solution for the integration of IP components as an interesting alternative. Borrowing from the communication networks literature, the components that can be combined to build an NoC can be heterogeneous including elements such as interfaces, routers, and links. Because of the many degrees of freedom in NoCs (such as choice, topology, and positions of the communication components, core interfaces, protocols) and composition rules, and of the many objectives that are of interest (such as performance, power consumption, reliability, and occupied area) to find an optimal or just a good solution is a very challenging proposition. Hence, the NoC design problem had been simplified by limiting the number and types of components considered, by focusing on a subset of the relevant objectives, by constraining NoC topology and components positions, and by dividing the optimization process in successive stages. Limiting the degrees of freedom has also the important side effect of reducing implementation and layout complexity. In NETCHIP [20] an *application-specific* NoC is obtained by mapping the application cores on standard topologies (e.g torus, mesh, hypercube) in an optimal way (SUNMAP [83]).

In [62], Hu and Marculescu perform mapping and routing on the NoC with optimal energy and performance. Lahiri *et al.* use standard topologies consisting of collections of channels (point-to-point links or shared busses) interconnected by bridges [72]. Ogras *et al.* [91] proposed a perturbation method that starting from the mapping of an application on a standard topology, optimizes the initial performance and cost by inserting custom long links between routers. In [84] Murali *et al.* synthesize NoCs that, albeit being more general than the approaches that start from a regular topology, are still constrained to be "two-level structures", where star topologies are interconnected by links chosen to to satisfy inter-cluster communication requirements.

Srinivasan *et al.* proposed to synthesize an application-specific NoC without assuming any pre-existing interconnection fabric [116]. The design flow is based on floorplan, generation of admissible router positions, and optimal routing. The synthesis problem is solved via integer linear programming (ILP). Due to the complexity of ILP, only a few locations for the installation of routers are considered. This simplification still yields running time of the order of several hours even for relatively small instances. More recently, the same authors proposed an efficient approximation algorithm that guarantees the optimality of the solution within a certain bound from the global optimum [117]. This bound is strongly tied to the cost model. Specifically, the per-hop cost is lumped on the NoC links and is assumed to be linear in the bandwidth, while no constraints are imposed on the router size.

We showed that a rich set of interesting results for NoCs exist; however, few are the examples of practical applications of NoCs. In fact, the debate between those who favor standard bus architectures or variations thereof and those who advocate the adoption of NoC approaches ranging from constrained architectures to custom ones is vibrant. We do not take sides even though

the NoC approach has undisputable fundamental merits that may make it successful in the long run. Instead, we propose a general methodology for the design of on-chip communication that can explore a large number of alternatives including as special cases NoCs, bus architectures and hybrid ones. Given the generality of our approach, it can be used to build a framework where different constrained solutions can be compared using a number of evaluation factors.

Our approach is based on the synthesis of optimal heterogeneous network topologies by assembling components from a fine-grained library without enforcing any constraint on the topology other than the ones formally captured in the library. In particular, the network that we obtain need not be direct and not even connected if these constraints are not captured in the composition rules of the communication components. The possible choice by the synthesis algorithms for a multistage network rather than a direct network is a consequence of the number of concurrent accesses to shared resource (e.g. DRAM controllers) and of the constraints on the amount of communication resources provided by the routers (i.e. number of input and output ports) and by the links (i.e. bandwidth capacity).

7.1 Design Flow

Figure 7.1 shows the software organization of COSI-OCC. The input to COSI-OCC is a project file that contains pointers to the communication specification and to the library. All files are in XML format. The communication specification contains a list of IP cores and inter-core communication constraints. The specification is parsed to yield an internal communication structure where the nodes are the IP core and the links represent constraints. If there are unplaced IP cores, PARQUET is used to floor-plan the chip [11]. The result of the floor-plan is parsed to extract two

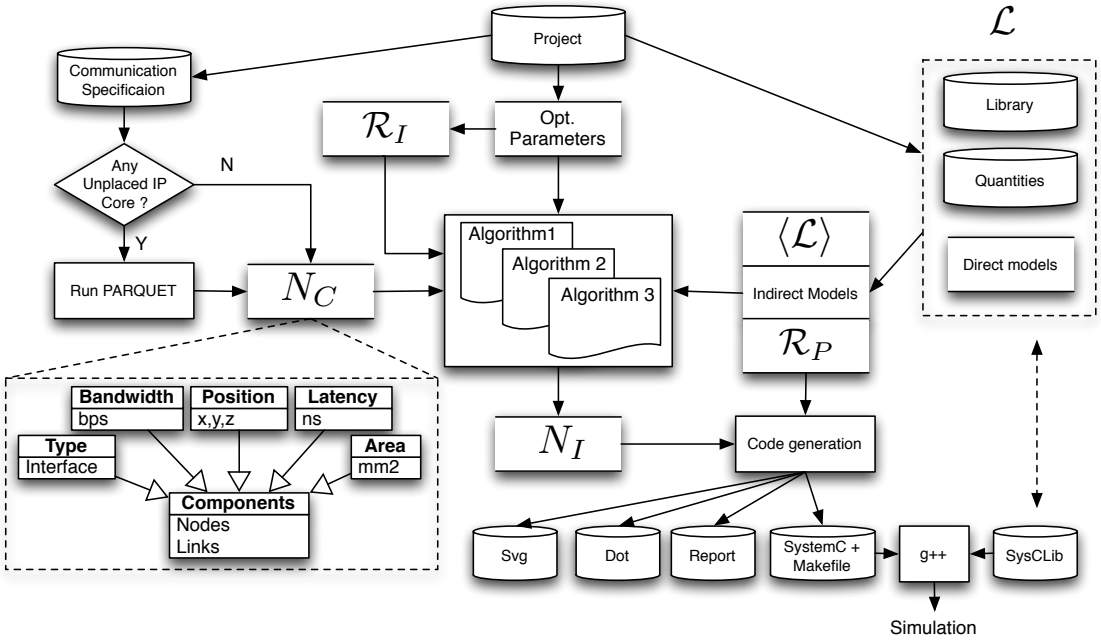


Figure 7.1: COSI-OCC open software infrastructure.

information: a bipartite graph where the nodes are the output (sources) and input (destinations) ports of the cores and the links are the communication constraints, and a set of polygons representing the area not occupied by IP cores. Additional area can be reserved by adding special IP cores to the input specification. The library file contains the description of each library element and the parameters of the performance and cost models attached to them. The library is used to construct the platform data structure that also contains a description of additional constraints like, restriction on the position of nodes, topological constraints or requirements on the implementation like deadlock freedom. The project file includes also the optimization parameters such as the relative weights of power and area costs. The communication specification and the platform are passed to the synthesis algorithm that derives the network implementation N_I.

COSI-OCC includes a set of code generators to produce an SVG graphical representation and a DOT logical representation of N_I. A SYSTEMC netlist can be generated from N_I by assembling the corresponding SYSTEMC-view of each element instantiated from the library that is contained in `SysCLib`, also part of the COSI-OCC distribution. The generation of the SYSTEMC netlist is a further refinement of N_I that requires the binding of each port of the nodes to links, the generation of the routing tables, and the computation of the weights for the *weighted fair queuing* algorithm, which is used by the routers to schedule flits. The COSI-OCC distribution includes a set of algorithms to solve some variants of the communication synthesis problem. For instance, we provide an algorithm that generates deadlock-free networks, e.g. for cases where support for virtual channels is not available in the routers. Our approach to solve this problem is different from the one proposed in [84] where a graph with all possible connections is constructed explicitly (i.e the Switch-Cost Graph) and the set of prohibited turns is precomputed before deciding the routing of packets. In COSI-OCC, we build the solution incrementally by instantiating links from the library. The optimization algorithm operates directly on the channel-dependency graph of the communication structure and at run-time checks that such graph is kept acyclic (i.e. it checks that the corresponding composition rule is satisfied). More information on COSI-OCC is available at the COSI project web site [31].

Comparison with Component Composition Frameworks

In this section, we compare COSI-OCC with component composition frameworks (CCFs) (Table 7.1).

The comparison of COSI-OCC with other CCFs will be done at the component-composition language (CCL) level, i.e. the language used by the CCF to define the structural aspect of a design. The following criteria are selected for comparison: the formal framework underlying the CCL, what

CCF Tool	Formal Model	Captures	Library	Composition Rules	Features
BALBOA [38]	N/A	Structure and types	IP in C++, RTL Level	Type compatibility	Type resolution
LSE [120]	N/A	Structure through modules and interfaces	Components written in C-like language (RTL Level)	Interface compatibility	Polymorphism, type inference, Use-based specialization
MCF [76]	UML metamodel	Entities and relationships, ports and interfaces, Transaction and RT level, transactors, busses and switches	IP in SystemC	Entity and relationship constraints in OCL	Consistency checking and type inference
SPARTACAS [80]	Rosetta language [12]	Interfaces and behavior	Rosetta models	Input-output values matching	Automated component adaptation
ABRIE [29]	Authors' defined	Components' type and ports, constraints on ports; connections' type and role, constraints on roles	Object or source code (software only)	Port matching (including function containment)	Automated component selection and matching
COSI-OCC	Domains of communication structures	Components, connections, types, ports, performance metrics	Set of communication structures	User-definable metrics compositions and relation on communication structures	Synthesis

Table 7.1: Tools for component based design.

the CCL captures, the supported library of components, the supported composition rules, and the features provided by the CCF. BALBOA and Liberty Simulation Environment (LSE) are not based on a rigorous formal model (as also noted in [76]). Both assume that the library of components is described in an imperative language such as C or C++ and that the components are exported using an interface description language. The only composition rule is that the types of connected components must be compatible. They can both perform type inference.

The Metamodeling-driven Component composition Framework (MCF) is based on a meta-model captured in UML. Compared to BALBOA and LSE, MCF supports components described at the RTL level or at the transition level. The two levels can communicate through transactors. Composition rules are captured using the Object Constraint Language (OCL) that is able to express constraints on entities (components) and relationships (connections). MCF is geared toward SystemC IP libraries and can perform consistency checking and type inference.

As opposed to the previous tools, SPARTACAS captures the architectural specification and the library components using the same language, Rosetta. The reason is that SPARTACAS defines the composition rules on the set of inputs and output values of components, and it features automated component adaptation. In this context, the specification of the problem is the desired input-output relation and the adaptation selects components from the library to match the problem specification.

ABRIE as been developed mainly for software reuse. A system is defined by components and connections. Each component has a type and a set of ports. Each component has a type and a role. There are constraints on the ports that can be connected to certain roles. ABRIE also provides automated component selection.

COSI-OCC has been developed for the synthesis of communication infrastructures. It is based on a formal model centered around a mathematical object called communication structure (Section 4.2) that captures components in terms of nodes and links. Different form the other CCFs, we attach also performance metrics to components and define ordering relations based on those. Because we target synthesis, rather than simulation, our approach to model libraries is similar to SPARTACAS. A library is a set of communication structures, i.e. it is captured in the same way as the specification of the problem. In contrast to the other CCFs, we allow the user to define the composition rules such that different systems can be obtained using the same library. Composition rules can be defined in a general way as relations between components and their properties (Section 5.1). Finally, COSI-OCC features the automatic synthesis of communication structures.

Remarkably, the output of COSI-OCC could very well be a description of a communication structure in one of the other CCFs. At the same time, once a system is simulated in a CCF, the communication requirement on each connection among components can be passed to COSI-OCC that uses synthesis to automatically refine the connections into a more sophisticated communication structure.

Comparison with NoC design frameworks

There is a *large* number of interesting contributions in the area of analysis and optimization methods for NoC design. In this section, we limit our attention to the ones that are more readily comparable to COSI-OCC. When comparing different approaches, we use our framework presented in Part II so that a direct comparison with COSI-OCC is immediate.

NetChip [20]. The NetChip design environment provides a complete solution to design, simulate and synthesize NoCs. The specification is captured by a *core graph* that is a communication struc-

ture N_C where $\mathbf{q}_C = (b)$. A platform instance is captured by a communication structure N_P where $\mathbf{q}_P = (\gamma)$. In NetChip, the library $\mathscr{L}$ contains a set of predefined topologies (like meshes, tori etc.). The NoC optimization is carried out by solving $\texttt{PR1}(N_P)$ for each topology in the library. Once and implementation N_I is found, other tools generate a SystemC executable simulation.

Gerstlauer *et al.* [51]. In this work, there are two abstraction levels: the network level and the link level. In this methodology, the specification is given as a set of processing elements and a set of channels between them. The first step is the assignment of channels to busses. The library that is available to the user contains busses, masters, slaves and transducers (that transfer data from one bus to another). The second step of the methodology consists in optimizing the link design. This environment is very interesting, but it lacks automatic synthesis and it only provides bus-based communications. On the other hand, the vector of quantities that are taken into account is very large and includes the protocol behavior.

SlicNet [6]. A variety of different analysis and optimization technique for NoCs have been developed within the SlicNet project. Each problem corresponds to a variant of $\texttt{PR1}(N_P)$ for a particular N_P. For instance, Ogras and Marculescu propose a technique to insert long links in a regular structure to improve the performance of the NoC [91]. In this case N_P is a mesh with some other extra links added. The cost function $F(N_I)$ is the value of network traffic at which the network enters a congested state. The objective is to maximize this value. Constraint 6.4 of problem PR1 includes an upper bound on the number of long links that can be inserted. Besides this work, other optimization techniques only consider regular topologies. Other contributions are in the development of direct and indirect statistical models for many quantities related to NoCs, as well as FPGA prototyping of NoCs.

Srinivasan *et al.* [116]. This work discusses a formulation of problem PR2 for a specific library of communication components that contains routers and links. The problem is formulated as an integer linear programming problem. Therefore, highly non-linear constraints, such as deadlock freedom, cannot be taken into account. Also, function F is considered to be linear in the value of quantity b.

QNoC [21]. The focus of the QNoC architecture is on quality of service. This NoC provides different service classes for the network traffic. This corresponds to consider the domain of quantity b as the set of sets of commodities, where each commodity is a source-destination flow with its own quality of service. Some tools are provided to map an application on a regular mesh and then remove nodes and links that do not belong to real paths (problem $\texttt{PR1}(N_P)$ with N_P being a mesh topology).

APSRA [82]. This work proposes a methodology for the computation of deadlock free routing functions such that the adaptiveness of the routing algorithm is not compromised. The model used to capture routing is very similar to the definition of the quantity ρ. Moreover, as in COSI-OCC, the definition of deadlock is also based on the channel dependency graph. The authors solve problem $\texttt{PR1}(N_P)$ for a given N_P. The decision variable of the problem is $l[(\rho)](v)$ for each node v. The cost function to maximize is a measure of the adaptiveness of the routing protocol. Constraint 6.4 of problem PR1 is that the network must be deadlock free.

GeNoC [23]. This is the only work that proposes a formal model for NoC that is used for verification. This work does not solve an optimization problem but a feasibility problem. Given the description of N_I in the Common Lisp language, the ACL2 theorem prover [68] is used to verify that Constraint 6.1 and Constraint 6.4 of problem PR1 are satisfied.

7.2 Specification

As described in Section 4.2, we capture the specification of the communication requirements of a SoC as a communication structures $N_C(\mathscr{C}_C, \mathbf{q}_C, L_C)$ with quantities $\mathbf{q}_C = (x, y, a, \tau, b, h)$. A node represents the port of a core and a link represent an end-to-end communication requirement between two cores. A port can be assigned to a specific position (x, y) on the chip or it can be unplaced, in which case the set of configurations L_C contains all possible assignment of the core port to positions on the die area.

Since the performance and cost of the on-chip communication depend on the position of the cores, the first step in our design flow is to restrict the possible configurations of a specification by fixing the position of the ports of each core. In COSI-OCC we rely on the capabilities of the PARQUET floor-planner [11] to obtain these positions. In particular, we generate an input to PARQUET as follows:

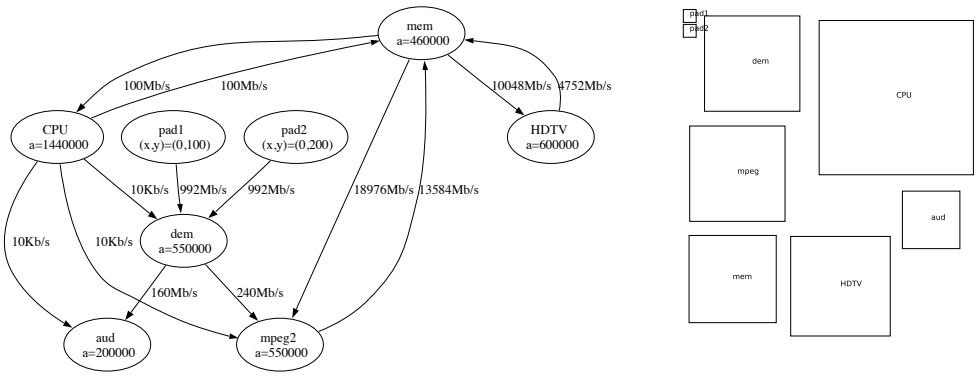
1. For each link in $e(u, v) \in \mathscr{C}_c$, we generate a net with a weight that is proportional to the maximum bandwidth requirement between from u to v.

2. For each placed core, i.e. for each core with an assigned (x, y) position on the chip, we give directions to PARQUET to fix its position in the floorplan.

3. Each block is considered as a hard rectangular block with aspect ration between 0.5 and 2.

The result of the floorplanning is then reinterpreted as a new communication structure $N'_C \leq_{\mathbf{q}_C} N_C$ where each component has a fixed assigned position on the chip. The communication structure N'_C is the input to the communication synthesis algorithm that is described in Section 7.4.

An example of specification and chip floorplan is shown in Figure 7.2 for the set-to-box



(a) (b)

Figure 7.2: Specification of the set-top-box example as given to COSI-OCC(a), and, Chip floorplan after elaboration from PARQUET(b).

example of Figure 4.3.

7.3 Library and Composition Rules

Figure 7.3(a) shows the internal architecture of an input-queued router. Data flits arriving at the inputs are buffered into queues and then sent to the right output by a cross-bar switch according to the information stored in a routing table. [1] Depending on the number of virtual channels supported by the router, there can be one or more queues for each input, called lanes. A router with i inputs and j outputs is characterized by an energy-per-flit metric $E(i, j)$ and an area metric $A(i, j)$. The table in Figure 7.3(b) reports $E(i, j)$ values across different router configurations and technology processes. Given a target technology process, the area and energy dissipation of a router depend on five parameters: number of inputs, flit-width, number of lanes, queue length inputs, and number of outputs. We obtained these values not through an analytical model, but by running a series of

[1] We assume that the bit-width of each port is equal to the flit-width. The routing table and scheduler are not shown in Figure 7.3.

simulations with ORION [124].

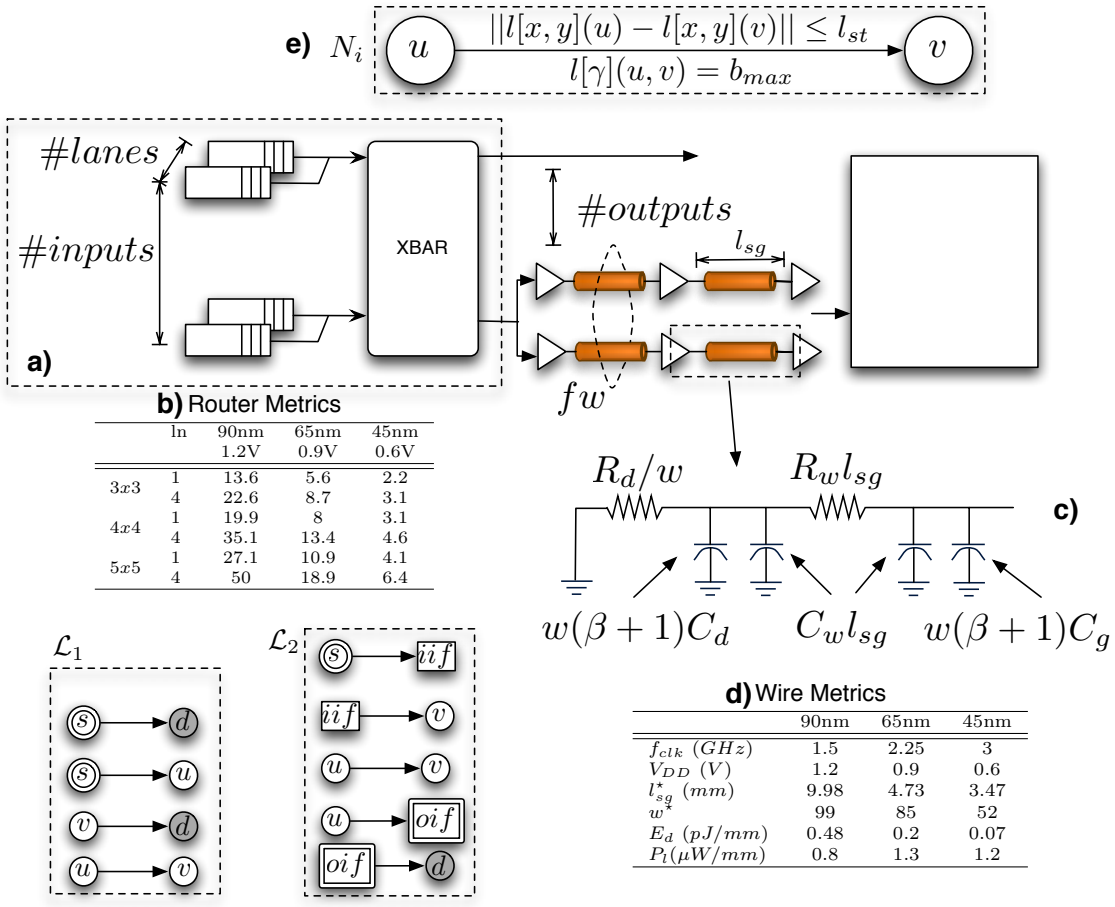
Network interfaces are directly connected to cores. Their characterization in terms of power and area is the same as for the routers. Differently from routers, interfaces need to provide extra services such as protocol conversion, flit-width adjustment, and packetization. Hence, their performance can be very different from the one of a router especially in terms of throughput and latency. We are aware of this difference and we plan to incorporate more detailed models for network interfaces in future release of COSI-OCC.

A link is a bundle of wires that connects the output port of a node with the input port of another node. Figure 7.3(c) shows the first-order RC model of a buffered wire where: R_d is the transistor driving resistance, w is the width of the buffer NMOS transistor normalized by the minimum technology width, β is the PMOS-to-NMOS sizing ratio, C_d and C_g are the diffusion and gate capacitance per unit width, R_w and C_w are the wire resistance and capacitance per unit length, and l_{sg} is the length of the buffered segment. The delay d of such segment is :

$$d = 0.7\left[\frac{R_d}{w}\cdot C_1 + (R_w + C_w)\frac{l_{sg}^2}{2} + l_{sg}\cdot R_w \cdot w(\beta+1)C_g\right]$$

where $C_1 = (w(\beta+1)(C_d+C_g) + l_{sg}\cdot C_w)$. To make the delay linear in the wire length, designers can insert an optimal number of buffer with optimal-size $w^\star$ spaced by a distance $l_{sg}^\star$ called *critical length* [14, 58, 59]. The delay $d^\star$ of a critical length is called *critical delay*. Assuming a synchronous design implementation with target clock frequency f, the maximum distance that a signal can travel on an optimally repeated wire within one clock period $T = 1/f$ is $l_{st} = \left\lfloor \frac{T}{d^\star}\cdot l_{sg}^\star \right\rfloor$ and it is called *critical sequential length*.

The power dissipated by an optimally-buffered line of length l, where $l_{sg} < l < l_{st}$, running at frequency $\frac{1}{T}$ with an activity factor α (the fraction of buffers that are switched during an average



	ln	90nm 1.2V	65nm 0.9V	45nm 0.6V
3x3	1	13.6	5.6	2.2
	4	22.6	8.7	3.1
4x4	1	19.9	8	3.1
	4	35.1	13.4	4.6
5x5	1	27.1	10.9	4.1
	4	50	18.9	6.4

	90nm	65nm	45nm
f_{clk} (GHz)	1.5	2.25	3
V_{DD} (V)	1.2	0.9	0.6
$l^{\star}_{sg}$ (mm)	9.98	4.73	3.47
$w^{\star}$	99	85	52
E_d (pJ/mm)	0.48	0.2	0.07
$P_l(\mu W/mm)$	0.8	1.3	1.2

Figure 7.3: Modeling the NoC components.

clock cycle) is:

$$P = \frac{l}{l^{\star}_{sg}} \cdot \left[\frac{\alpha}{T} V_{dd}^2 \cdot \left(C_1^{\star} + C_w\right) + \frac{V_{dd}}{2} \cdot \left(w^{\star}(I_n^{off} + 2I_p^{off}) w_n^{min}\right) \right]$$

where V_{dd} is the supply voltage and I_n^{off} (I_p^{off}) is the leakage currents per unit NMOS(PMOS), and w_n^{min} is the width of the NMOS transistor of a minimum-size inverter. The two terms of the sum are the switching power and the leakage power. [2]

[2] We simplified the formula omitting the contribution of the short-circuit current which is negligible with respect to the other two contributions.

The table in Figure 7.3(d) summarizes the metrics of interests for the purpose of NoC synthesis. In particular, each link is characterized by an energy dissipation per bit per unit length E_d/l and an area per bit per unit length, which includes the wiring and buffer areas.

Figure 7.3(e) shows the basic NoC component N_i, i.e. a link. The set of configurations of a component contains all assignments of positions to the two nodes such that their distance is not greater than the maximum distance l_{st}. The capacity of a link is equal to b_{max} and the latency is equal to one hop. The capacity b_{max} is different from the clock frequency. In fact, in order to avoid router congestion, the capacity of a link should be set in such a way that the routers' injection rate is far from saturation. Otherwise, the actual communication latency would grow exponentially.

In Figure 7.3, $\mathscr{L}_1$ and $\mathscr{L}_2$ are two possible communication libraries. There are many types of nodes: s is a source node (withou any input), d is a destination node (without any output), u and v are routers, iif is an input interface and oif is an output interface. Since each component in $\mathscr{L}_1$ has the same interface, this library allows establishing direct connections between a source and a destination. Instead, library $\mathscr{L}_2$, where the source interface iif is different from the destination interface oif, supports a design flow where there are dedicated sockets to connect the cores to the NoC.

Two important composition rules are implemented in COSI-OCC. At the platform level, rule $\mathscr{R}_P$ allows only communication structures where the number of input and output links of a node does not exceed the number of input and output ports, respectively. At the implementation level, rule $\mathscr{R}_I$ allows only deadlock-free communication structures by forcing the channel-dependency graph of an implementation to be acyclic. Moreover, the bandwidth on any channel cannot exceed its capacity.

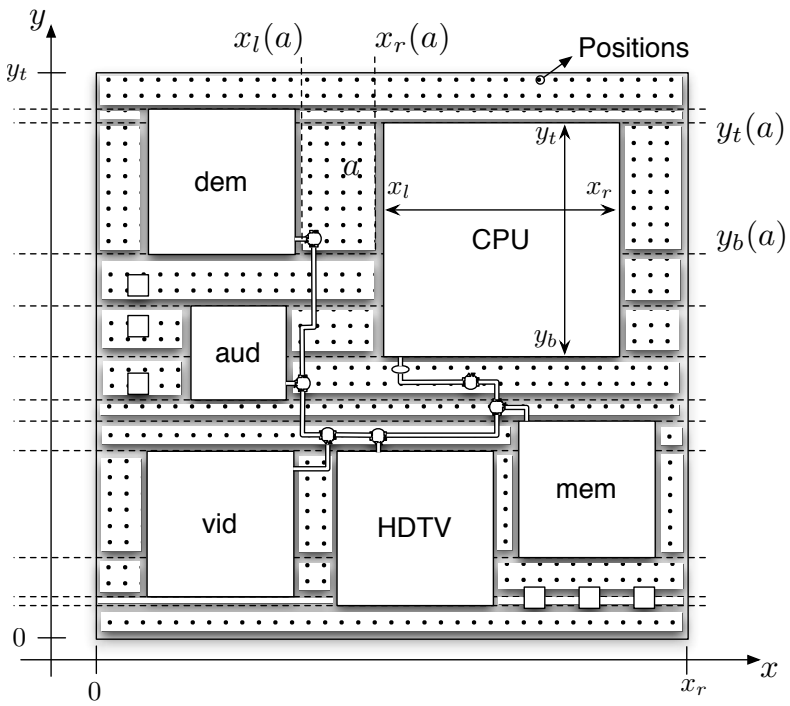


Figure 7.4: Slicing method to find the available area for NoC implementation.

7.4 Optimization Algorithm

We begin our optimization process by assigning a fixed position to each core with a floor-planner. To identify some of the degrees of freedom of the optimization variables, we define the area A_C available to lay out the network. We assume that placing the network over the cores is not allowed. Hence, A_C is the area of the chip that is not occupied by the IP cores. Fig. 7.4 shows the floor-plan of the set-top box example and a possible layout of the bus implementation of Fig. 6.1. A_C can be represented as the union of a finite set of rectangles $\mathscr{A}$ that can be automatically computed. Each rectangle $a \in \mathscr{A}$ is described by four real numbers $x_l(a)$, $x_r(a)$, $y_t(a)$ and $y_b(a)$, denoting its left, right, top and bottom boundaries, respectively.

Next, we discretize the rectangles to make quantities x and y finite and apply Proposition 1,

We assume that the available positions are uniformly distributed in a with a given density δ, Hence the number of positions available is equal to $\lceil \delta \cdot (x_r(a) - x_l(a))(y_t(a) - y_b(a)) \rceil$. The set $D_{(x,y)}$ is the union of the positions in all rectangles in $\mathscr{A}$. Quantity τ is also finite and quantity γ is bounded, moreover we assume to have a finite library. Because we want $l_I[(x,y,\tau)]$ to be injective (i.e. only one component of a specific type can be installed in a particular location), the maximum number of nodes in any platform instance is limited to $|D_{(x,y,\tau)}|$. Thus, let $U'_V \subset U_V$ be a set of nodes such that $|U'_V| = |D_{(x,y,\tau)}|$. The set of valid renaming functions R' is such that each node of the library elements is renamed to one of the nodes in U'_V. Hence, it is possible to find an upper bound $N_P^{\langle \mathscr{L} \rangle}$ following the construction of Proposition 1.

At this point, we may attempt at solving Problem PR2 with Integer Linear Programming (ILP). To illustrate this approach, consider library $\mathscr{L}_1$ of Fig. 7.3. First, we have to linearize the cost function by assuming that the cost of a router is the sum of the cost of each input port plus the cost of each output port. Then, we define the energy per flit as $\min_{i,j}[E(i+1,j) - E(i,j)]$ for an input port and as $\min_{i,j}[E(i,j+1) - E(i,j)]$ for an output port (and, similarly, the leakage power and area occupation). This linearized cost function is a lower bound of the real cost of the network. Hence solving the ILP with this cost function returns a solution that is optimistic. Using one binary variable for each installation site denoting whether a router is installed at that site, one binary variable for each link that can be installed between two sites, and one binary variable to denote that a constraint is routed through a link, the number of variables of the ILP problem becomes very large. It is equal to $|U'_V|^2 \cdot |E_C| + |U'_V|^2 + |U'_V|$ where the first term is the square of the number of installation sites times the number of constraints. For the simple example of Fig. 4.3 with 70 installation sites, the number of binary variables is $93,170$. This many variables causes an ILP solver to run very slow.

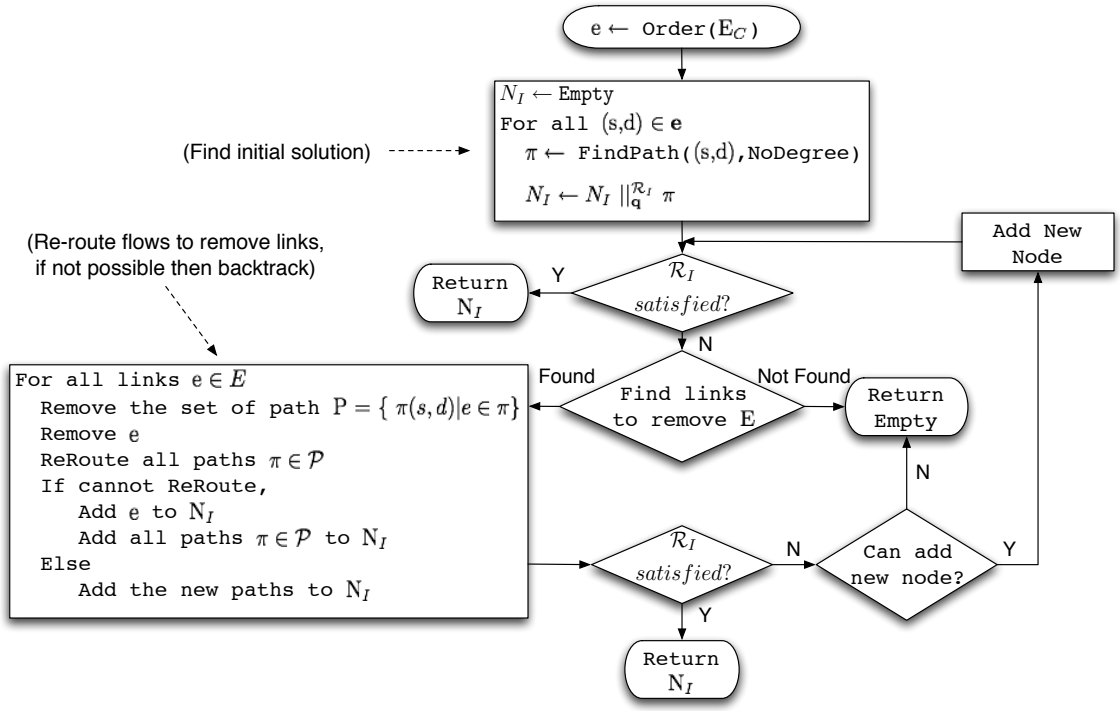


Figure 7.5: High-level description of the heuristic algorithm.

Moreover, some composition rules (e.g deadlock freedom) cannot be included in the ILP since they are highly non linear. Because of these difficulties, we devised a heuristic approach to solve problem PR2. In Section 7.5, we compare the results obtained by the heuristic with a lower bound provided with a further optimistic approximation of the ILP formulation.

Structure of the Algorithm

Figure 7.5 shows the high-level structure of the heuristic algorithm. In the first step, we find an initial solution not taking into consideration node degree constraints. In the second, we use an iterative procedure that removes degree violations by deleting links and/or adding routers. The initial solution is found with the same technique that is used in algorithms for global routing:

the end-to-end constraints in E_C are first ordered by decreasing bandwidth. One path in $N_P^{\langle \mathscr{L} \rangle}$ is then found for each constraint one at a time (the actual implementation of procedure `FindPath` depends on the composition rules). In this phase, the degree constraint rule is not taken into account; however, if we are lucky, N_I may still satisfy the degree rule, in which case the algorithm returns N_I and stops. Otherwise, we activate an iterative procedure to remove the degree constraint violations.

This procedure implements a rip-up and reroute approach one link at a time. The links connected to the output of nodes with output degree violations and links connected to the input of nodes with input degree violations are the ones that are considered for rip-up and re-route. For each link, all source-destination paths containing that link are re-routed by procedure `FindPath` that now takes into account the degree constraint rule. If a path cannot be removed, the algorithm back-tracks by reinserting the link and all the paths. Otherwise, the new paths are added to the communication implementation. If the re-routing procedure finds an implementation that satisfies the composition rules, the algorithm ends with success.

Otherwise we try adding a new node (router) to yield a feasible solution. The idea is that when a new node is added, multiple links entering/exiting a node can be merged/split into/from one link, thereby reducing the degree of the node (Figure 7.6). However, if no node can be added (e.g., because delay constraints would be violated) the algorithm ends with an empty implementation implying that no solution was found. Figure 7.6 illustrates how new nodes are added. First, among all nodes with input/output degree violations, the one with the highest number of input/output links is selected. All input/output links to/from the node are candidates for the merge operation. A subset of them is chosen with a criterion that depends on the optimization goal. The source and target nodes of the selected links are connected to the new router, which is instanced in a position such

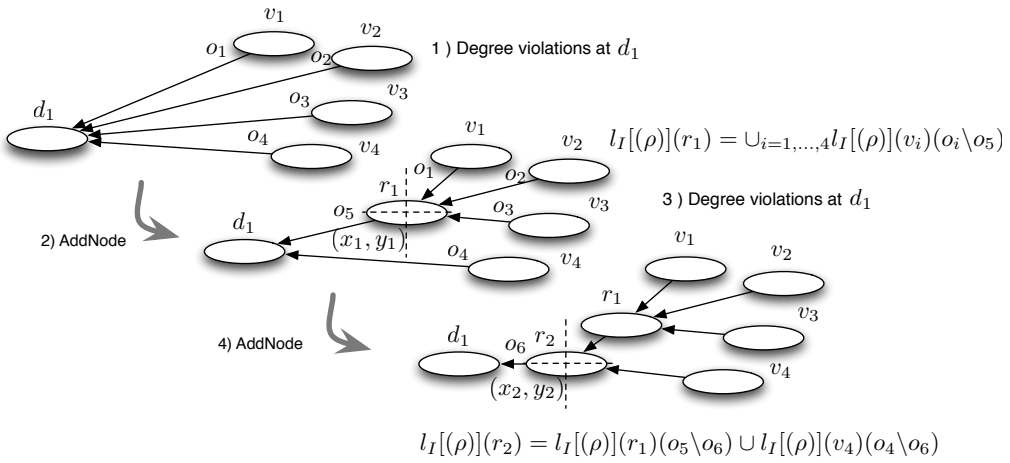


Figure 7.6: Procedure for adding a new router to the NoC implementation. For an expression *exp*, we denote by $exp(x \backslash y)$ the same expression where variable *x* has been replaced by *y*.

that the cost of the links is minimized. The transfer table of the router is set according to the new paths flowing through it. As it executes this local transformation the algorithm makes sure that link capacities and degree constraints are not violated. Note that the merge operation does not change the number of nodes with degree violations.

The `FindPath` procedure

This procedure is available in different forms to search for the "best" path between a source and a destination core depending on the particular composition rules that the user specified. If a delay model must be taken into account to check delay constraints (rule $\mathscr{R}_1$), the best path is discovered by a labeling algorithm (`SpLabeling`) that finds the minimum-cost constrained shortest path between two nodes; a modified version of the Dijkstra shortest path algorithm is used otherwise. If deadlock freedom (rule $\mathscr{R}_2$) is included in the set of rules $\mathscr{R}_I$, then `FindPath` runs on the channel-dependency graph of the communication implementation to make sure that this graph remains acyclic. The degree constraints of the nodes can be taken into account by adding rule $\mathscr{R}_3$).

Procedure `Reach`$(N_I, v, \mathscr{R}, \mathscr{L})$

$N_R \leftarrow$ empty communication structure ;
forall $N_i \in \mathscr{L}$ *with* $C_i = \{v_i, u_i, (u_i, v_i)\}$ **do**
  forall $l_i \in L_i$ **do**
1 **if** $l_I(v) = l_i(u_i)$ **then**
2 let $N_i'(\mathscr{C}_i, \mathbf{q}_P, \{l_i\})$;
3 $N_i'' \leftarrow r(N_i)$, with $r(u_i) = v, r(v_i) = id(l_i(u_i))$;
4 **if** $\Psi(N_I) \|^{\mathscr{R}}_{\mathbf{q}} N_i''$ *is defined* **then**
  $N_R \leftarrow N_R \|^{\mathscr{R}}_{\mathbf{q}} N_i''$
return N_R ;

`FindPath` explores the upper bound $N_P^{\langle\mathscr{L}\rangle}$ without building an explicit representation. In fact $N_P^{\langle\mathscr{L}\rangle}$ is explored locally at run-time by procedure `Reach`. This procedure takes as input parameters the current communication implementation N_I, a node $v \in \mathscr{C}_I$, the composition rules $\mathscr{R}$, and the platform library $\mathscr{L}$. `Reach` checks which links can be instantiated with the source node v (Line 1). For each link, an instance is generated by renaming the nodes appropriately (Lines 2 and 3). Function *id* associates a unique identifier to a node depending on its type and position. If the new link can be composed with the communication implementation without violating the composition rules (Line 4), then it is added to the reachable communication structure N_R (note that $N_R \in \mathscr{G}_{Q_P}$. Therefore the set of rules $\mathscr{R}$ must contain $\mathscr{R}_P$).

Procedure `SpLabelling` is one particular implementation of `FindPath`. It solves the constrained shortest-path problem [55] using a labeling algorithm [37]. We use the number of hops as a model for latency. A distance label is a tuple $D = (H, C)$ associated to a node v where H is the number of hops of the path from the source s to v with minimum cost C. A distance label D is dominated by D', written $D < D'$ if $D.H \leq D'.H$, $D.C \leq D'.C$, and $D.H \neq D'.H \vee D.C \neq D.C'$. A set of distance labels $D[v]$ is associated to each node v. The queue Q contains pairs (v, D) where v is a node and $D \in D[v]$ is a distance label of v. Distance labels in the queue are ordered by number of

Procedure `SpLabelling`$(s,d,N_I,\mathscr{L},\mathscr{R},\mathscr{R}_I)$

$D[s] \leftarrow \{(0,0)\}$, $D[v] \leftarrow \emptyset, \forall\, v \in U'_V \setminus \{s\}$;
$Q \leftarrow (s,(0,0))$;
while $Q \neq \emptyset$ **do**
  $(v,D_v) \leftarrow$ `ExtractMin(`Q`)` ;
  $N_\pi \leftarrow$ path from $(s,(0,0))$ to (v,D_v) ;
  $N_R \leftarrow$ `Reach(`$N_I \| _{\mathbf{q}_I} N_\pi, v, \mathscr{R}_P, \mathscr{L}$`)` ;
  forall $(v,u) \in \mathscr{C}_R$ **do**
    $l \leftarrow$ `Configure(`L_I, (v,u)`)` ;
    define $N'(\{(u,v)\}, \mathbf{q}_I, \{l\})$;
    if $N_I \|^{\mathscr{R}_I}_{\mathbf{q}_I} N'$ *is defined* **then**
      $f \leftarrow$ Compute incremental cost ΔF ;
      $D_u = (D_v.H + 1, D_v.C + f)$;
      if $\nexists D \in D[u]$ *s.t.* $D < D_u$ **then**
        $D[u] \leftarrow D[u] \cup \{D_u\}$;
        `Insert(`$Q,(u,D_u)$`)` ;
        $\pi[(u,D_u)] \xleftarrow{N'} (v,D_v)$;
if $D[d] = \emptyset$ **then**
  return $\emptyset$;
else
  return `ToGraph(`π`)` ;

hops and for the same number of hops, by cost. The procedure starts with an empty set of distance labels for all nodes but the source, which has the distance label $\{0,0\}$. The pair $(s,\{0,0\})$ is the only element in the queue (Line 2). The minimum distance label node is extracted from the queue (Line 3) and the set of possible links departing from the node (computed by procedure `Reach`) is processed (Lines 4 and 5). Each link is first configured by selecting one possible configuration, then composition rules are checked (Line 6). If this distance label is not dominated by any other already present at u, then it is added to the set of distance labels of u (Line 9), the new pair is added to the queue (Line 10), and the predecessor tree is updated (Line 11). A path from s to d that satisfies the hop constraint exists if the set of distance labels at d is not empty (Line 12). If this is the case, the path with minimum cost C is selected and returned. During the construction of the initial solution, the composition rules $\mathscr{R}$ and $\mathscr{R}_I$ do not contain rule $\mathscr{R}_3$, which is added during the re-routing procedure instead.

7.5 Results

Table 7.2 lists the SoCs that we used in our experiments. We selected the test cases based on several criteria:

- the number of IP cores $|V_C|$, ranging from 12 to 42, and the size of the chip, as large as $48mm^2$;
- the total bandwidth requirement, defined as the sum of the bandwidth requirements over all end-to-end constraints E_C, and ranging from 9 to 99 $Gbps$;
- the maximum input degree of a destination core and the maximum output degree of a source core, ranging from 2 to 25 depending on the SoC application.

The goal of our experiments is to study the impact of these application features on the synthesized NoC. Specifically we are interested in the following metrics: the power and area breakdown, the maximum and average input and output degree of the nodes, the maximum and average number of hops among all source-destination paths, and the maximum and average latency. The latency measurements are obtained by simulating the SYSTEMC implementation of the NoC generated by COSI-OCC. The maximum latency is the largest end-to-end delay experienced by any packet over the entire simulation, i.e. the time that elapses between the generation of the head flit to the delivery of the tail flit to the destination. The average latency is computed by dividing the sum of the latencies of each packet over the simulation run by the total number of flits sent. The SYSTEMC model of the NoC implements wormhole routing and weighted round robin packet scheduling. Moreover, each packet has one header flit, one tail flit, and four payload flits.

7.5.1 Impact of the Application Characteristics

The SoC applications used in this experiment were: a Multi-Window Displayer (MWD), an MPEG4 decoder (MPEG4), a Video Object Plane Decoder (VOPD) as well as two applications, called dVOPD and tVOPD obtained by instantiating two and three VOPDs, respectively sharing a common memory. We assumed a $100nm$ technology and a clock frequency of $1.5GHz$. The link capacity b_{max} was set to $1.12GBps$ We used six libraries of communication components differing for the flit-width of the data path (32 bits and 128 bits corresponding to $280 \cdot 10^6$ and $70 \cdot 10^6$ flits per second, respectively) and the size of the largest switch available in the library (2×2, 5×5 and 8×8).

The results are reported in Figure 7.7. Each histogram is divided into five zones, one for each application. Each zone contains six bars, one for each library.

Name	$\|V_C\|$	$\|E_C\|$	Area (mm^2)	Total Bw. ($Gbps$)	Ref.
MWD	12	13	3×4	8.96	
MPEG4	12	27	3×2.35	27.8	[20]
VOPD	12	15	1.53×1.18	27.9	
dVOPD	26	34	2×2.23	66.6	[104]
tVOPD	38	51	2.78×2.37	98.84	
VProc	42	69	8×6	78.2	

Table 7.2: Characteristics of the selected SoCs applications.

The power consumption and the area occupied by the network are increasing functions of the total bandwidth requirement. Most test cases do not need the instantiation of large routers. For instance, the number of input and output ports on each router in the NoCs supporting the MWD and the VOPD applications is no greater than two since each core is a source and/or destination of few communication constraints. These NoCs are basically a set of dedicated point-to-point links with very little sharing. Hence, the difference between the maximum and the average latency is small.

The dVOPD and tVOPD applications show the effect of merging different communications into a common link. In these applications, a central memory is shared among a few cores. Since the memory has only one input and one output port, one or more routers are needed to merge concurrent accesses to memory using time multiplexing. Allowing the installation of larger routers provides two advantages: (1) the total power consumption is reduced (14% and 12% for dVOPD and tVOPD, respectively) as a consequence of a reduced hop count, and (2) the end-to-end latency (both

the maximum and average value) is reduced. The latency decrease is modest compared to the reduced number of hops because the time spent for contention among the input FIFOs grows with the router size. Generally, however, for these applications the reduced number of hops counterbalances the negative effect due to contention.

In the MPEG4 application, the SDRAM is shared among many more cores than in the case of the dVOPD and tVOPD applications. Hence, the use of larger routers does not give the same benefits. Despite the significant differences between the maximum number of hops in the 2×2 and 8×8 cases, there are no gains in terms of maximum latency, which in fact is even worse for larger routers (a 73% increase with respect to the 2×2 case). Here, port contentions cannot be counterbalanced by the reduced hop count, as opposed to the dVOPD and tVOPD cases where routers have no more than five inputs.

This set of test cases shows that the power consumption and the area occupied by the NoC implemented with 32-bit links is much smaller than in the 128-bit implementation. The latter case gives smaller link utilization (i.e. flit rate), which reduces the latency due to contention. This, however, is a minor gain and does not justify the use of wider data parallelism, which should be limited to cases when the required bandwidth cannot be achieved with narrower links.

7.5.2 Effect of Technology Scaling

For the second set of experiments we selected the VProc SoC as a representative embedded system application and we studied the impact of scaling the technology on the performance and cost of the synthesized NoC. VProc features a central memory that serves 25 different cores. Each core requires a write and read bandwidth of $960 Mbps$ (in each direction). Technology scaling generally enables higher transistor densities and clock frequencies. Hence, as we scale the technology

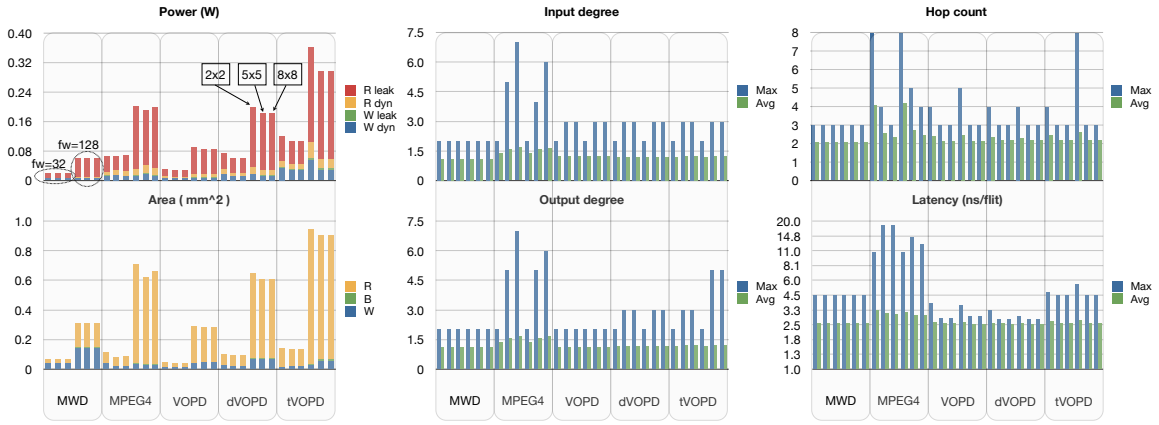


Figure 7.7: Properties of the synthesized NoCs for the MWD, MPEG4, VOPD, dVOPD and tVOPD applications. Power is expressed in $Watts$, area in mm^2 and latency in $ns/flit$. We used the following notation: `R` for routers, `W` for wires, `B` for sequential buffers. Latency is reported on a logarithmic scale.

we double the bandwidth requirements from each core to the central memory while keeping the core size fixed. This choice mimics the fact that two cores can fit in the area of one, as the transistor density doubles with the new process generation.

We used a total of 9 libraries obtained as the combination of three different technology processes with three different router designs: specifically, we used $100nm$, $70nm$, and $50nm$ technologies while the routers' maximum size was set equal to 2×2, 5×5 and 8×8, respectively. The clock frequency was set to $1.5Ghz$ at $100nm$. Since the total memory bandwidth is $3GBps$, we set the flit width to 128 bits to achieve a link capacity of $3.2GBps$ with a maximum flit rate of $200 \cdot 10^6$ per input port of the routers. We increased the clock frequency to $2.25GHz$ and $3GHz$ for the 70 and $50nm$, respectively. We also increased the link capacities to 6.4 and $12.8GBps$ for the two technologies, respectively. The synthesis constraints were set as follows: the density of the installation sites was fixed to 20, which gives a total of 364 different possible locations to install the routers; the synthesis goal is minimum latency with a constraint that each path be no longer than 10 hops. The

results are reported in Figure 7.8.

The critical sequential length drops from 9.98mm at 100nm down to 3.47mm at 50nm due to the different electrical parameters of the wires and also to the increased clock frequency (from 1.5 to 3 GHz). Since the chip size is $8 \times 6mm^2$, the entire chip can be spanned in one clock cycle at 100nm while 3 cycles are needed at 50nm. Hence, it is not surprising that the maximum number of hops at 50nm does not change much across the various router configurations because intermediate FIFOs are needed to segment long interconnection links. The use of larger routers does not help to reduce the number of hops while it increases the chance of contention. Therefore, contrary to the 100nm case, the reduced number of hops does not balance the higher contention probability and, ultimately, the average latency obtained by simulation actually increases.

In terms of power consumption, while there is an advantage in using larger routers with the 100nm technology, the picture changes at 50nm. In fact, the higher clock frequency and the increase in the transistor leakage power discourage the use of larger routers. When stateful repeaters are needed to span large distances among the cores, it is more efficient to spatially distribute small routers on the chip. Finally, the result highlights the need for more accurate timing models for synthesis rather than the simple measure based on hop count.

7.5.3 Quality of the Solution

The cost of the solution obtained by our algorithm can be compared to the solution to an equivalent Mixed Integer Linear Programming (MILP) formulation of the the same problem. Consider an SoC specification as in Figure 4.3 that is comprised of a set of n cores and m end-to-end communication constraints. After floor-planning, the chip area A is partitioned in two disjoint regions: $A_{IP} = A_1 \cup ... \cup A_n$ is the region occupied by the cores and $A_N = A \setminus A_{IP}$ is the (non-convex)

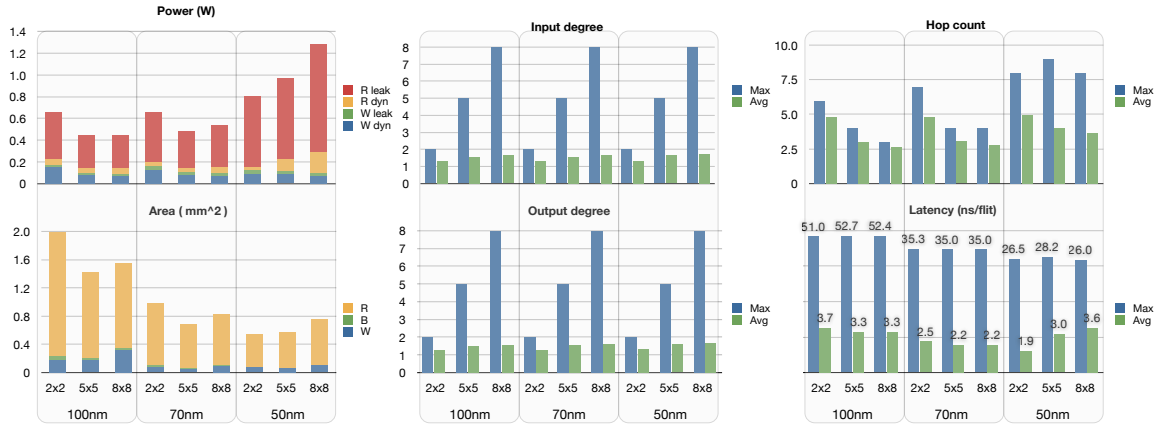


Figure 7.8: Properties of the synthesized NoCs for the VProc applications.

region available for the NoC. A_N can be sampled such that a finite number of points are considered as installation sites for routers. We use variables like u and v to refer to these points. The set of all valid network topologies can be implicitly captured by a graph $G(V,E)$ where V represent the set of installation sites and E the links among them. Notice that E does not have to be $V \times V$ necessarily, but some constraints (for instance the maximum distance that a link can span) can be used to prune the set E. We want to install the necessary amount of routers and links to implement the specification at minimum expense.

Let $\mathscr{C} = \{1, ..., m\}$ be an index set such that each end-to-end constraint is associated to one index $q \in \mathscr{C}$. With $s(q)$,$d(q)$,$b(q)$ and $h(q)$ we denote the source, destination, minimum bandwidth requirement and maximum number of hops of constraint q. Let x_{uv} be an integer variable that is equal to 1 if the link (u,v) belongs to the network implementation and 0 otherwise. Let $\mathbf{x^e}$ be a vector of all x_{uv} for some ordering of E. Also, let y^q_{uv} be an integer variable that is equal to 1 if the path from $s(q)$ to $d(q)$ contains edge (u,v), and $\mathbf{y}^q$ be the vector of all y^q_{uv}. With $\mathbf{Y}$ we denote the matrix whose columns are the vectors $\mathbf{y}^q$. For each installation site u, let $x_u = 1$ if a router is

installed at u and $x_u = 0$ otherwise. With $\mathbf{x}$ we denote the vector of all x_u for some ordering of V.

For the same ordering of V, we define the vector $\mathbf{b}^q$ as follows: $b^q(v) = 1$ if $v = s(q)$, $b^q(v) = -1$ if $v = d(q)$ and $b^q(v) = 0$ otherwise. Finally, with M we denote the node-edge incidence matrix of G. The NoC optimization problem can be written as follows:

PRMILP :

$$\min F = \sum_{(u,v)\in E} f_{uv} + \sum_{u\in V} f_u$$

subject to

$$1)\ x_{uv} \le x_u \wedge x_{uv} \le x_v \qquad \forall (u,v) \in E$$

$$2)\ x_{uv} \ge y^q_{uv} \qquad \forall (u,v) \in E,\ \forall q \in \mathscr{C}$$

$$3)\ M\mathbf{y}^q = \mathbf{b}^q \qquad \forall q \in \mathscr{C}$$

$$4)\ \sum_{q\in\mathscr{C}} y^q_{uv} b(q) \le b_{max} \qquad \forall (u,v) \in E$$

$$5)\ \sum_{(u,v)\in E} y^q_{uv} \le h(q) \qquad \forall q \in \mathscr{C}$$

$$6)\ \sum_{u\in V:(u,v)\in E} x_{uv} \le in_{max}(v) \qquad \forall v \in V$$

$$7)\ \sum_{v\in V:(u,v)\in E} x_{uv} \le out_{max}(v) \qquad \forall u \in V$$

$$8)\ x_u \in \{0,1\} \qquad \forall u \in V$$

$$9)\ x_{uv} \in \{0,1\} \qquad \forall (u,v) \in E$$

$$10)\ y^q_{uv} \in \{0,1\} \qquad \forall (u,v) \in P, \forall q \in \mathscr{C}$$

Constraints 1 ensure that a link in installed between two nodes only if such nodes are present in the network. Constraint 2 ensures that a link is used by a path only if it belongs to the implementation. Constraint 3 is the flow-conservation constraint. Constraint 4 is the capacity constraints saying that for each edge the total bandwidth cannot exceed the edge capacity. Constraint

5 says that the number of edges in a path cannot be greater than the hop constraint. Constrain 6 and 7 are the input and output degree constraints, respectively.

The cost function F is the sum of the cost of the links plus the cost of the nodes that can be written as: $f_{uv} = \alpha_{uv}(\mathbf{x^e}) + \beta_{uv}(\mathbf{x^e}, \mathbf{Y})$ and $f_u = \alpha_u(\mathbf{x^e}) + \beta_u(\mathbf{x^e}, \mathbf{Y})$. The installation cost of a component (i.e. area and leackage power), denoted by $\alpha_u(\mathbf{x^e})$ for nodes and $\alpha_{uv}(\mathbf{x^e})$ for edges, depends only on the way in which components are interconnected. The utilization cost (i.e. the dynamic power consumption), denoted by $\beta_u(\mathbf{x^e}, \mathbf{Y})$ for the nodes and $\beta_{uv}(\mathbf{x^e}, \mathbf{Y})$ for the edges, depends on the topology and on the the bandwidth flowing in each component (i.e. the routing decisions).

The cost function F is, in general, neither convex nor convex. In fact, the area, leakage power and energy per flit metrics of a router depend on the product of the number of input and output ports (due to the complexity of the crossbar switch). If F can be approximated by a convex or a piece-wise concave function, specialized algorithms exists to solve this problem ([85, 93]) when constraints 6 and 7 are not taken into account.

Ideally, we would like to compare the *exact* solution of problem PR2 with the one found by our heuristic algorithm. Therefore, we linearize the cost function as sketched in Section 7.4. However, the best we can do is to compare our results with a lower bound since even the relaxed ILP formulation has prohibitive running times for our test cases. We relaxed the problem further by converting the ILP into a Linear Program (LP). We used CPLEX [32] to solve the linear program. We then computed the ratio of the power consumption of the solution found by CPLEX over the one of our heuristic algorithm across various benchmarks. Table 7.3 report these results together with the number of positions $|D_{\{xy\}}|$, the computation time 't_{cpu} LP' of CPLEX and the computation

time 't_{cpu} H' of our heuristic.

In most cases our heuristic algorithm is 2-3 orders of magnitude faster than solving the LP (a remarkable fact since the LP DOES not find a feasible solution). The power of the NoC found by the heuristic is within 2x from the power found by CPLEX that is very optimistic for the change in the cost function and for the relaxation of the integer constraints.

Name	$\lvert D_{(x,y)} \rvert$	t_{cpu} LP	t_{cpu} H	Ratio
MWD 2x2	94	4.76	0.11	1
MWD 5x5	94	4.83	0.11	1
MWD 8x8	94	4.78	0.11	1
MPEG4 2x2	117	434	5.29	0.49
MPEG4 5x5	117	479	1.46	0.55
MPEG4 8x8	117	394	1.32	0.48
VOPD 2x2	63	1.98	0.13	0.73
VOPD 5x5	63	0.87	0.13	0.78
VOPD 8x8	63	0.85	0.13	0.78
dVOPD 2x2	147	130	1.8	0.69
dVOPD 5x5	147	60	1.66	0.66
dVOPD 8x8	147	60	1.65	0.66
tVOPD 2x2	150	438	4.54	0.71
tVOPD 5x5	150	423	3.32	0.66
tVOPD 8x8	150	426	3.34	0.66

Table 7.3: Evaluating the heuristic algorithm of Figure 7.5.

Chapter 8

Building Automation Networks

Electronics controllers for a large number of applications such as public infrastructure management, industrial plant control, automotive networks, avionics, and building automation are *networked* because of the distributed nature of the plant that they control. Figure 8.1 illustrates the design process of mapping an embedded control specification onto a networked execution platform. At the specification level, an abstract model of the plant is used to derive the desired properties of the feedback controllers such as stability and robustness [90]. Each controller C_i is derived assuming a continuous-time model and then discretized with the choice of a suitable sampling period Δt_i that preserves its properties [46]. Complex plants with multiple physical quantities to be controlled typically require multiple controllers that may be discretized with different sampling periods. At each sampling period, a certain amount of data is transferred from the sensors to the input of the controllers and from the controllers' outputs to the actuators. Therefore, each logical connection between the controllers and the plant implicitly defines a *message frequency*. For instance, controller C_1 in Figure 8.1(a) receives b_{p1} messages per second from the sensors and sends b_{1p} messages per

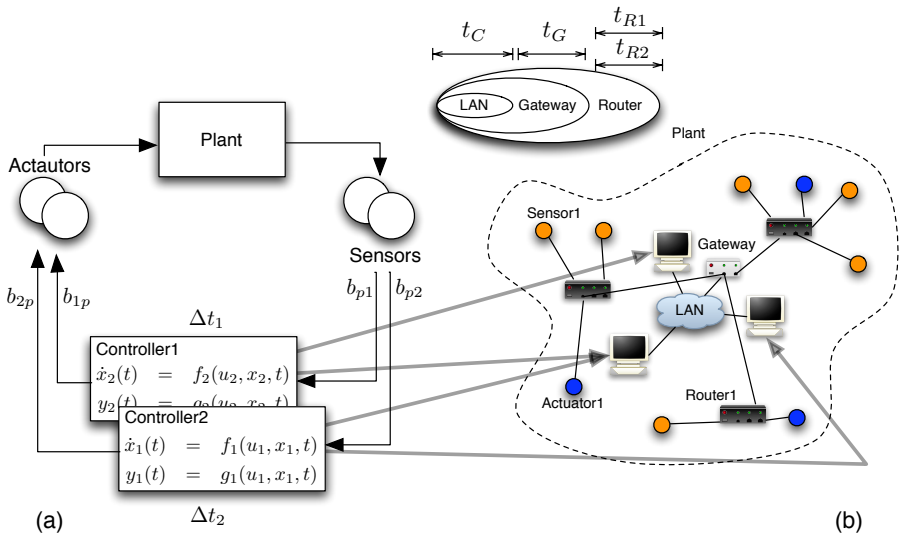


Figure 8.1: A distributed embedded control system: (a) controller specification and (b) networked execution platform.

second to the actuators. Moreover, distributed computation requires multiple computers that need to exchange data via an interconnect network[1].

The network shown in Figure 8.1(b)) is heterogeneous and hierarchical; it contains a high-performance local area network (LAN), also known as the *backbone network*, that connects the various computers and is attached via a *gateway* to a *control network island*, or *zone*. In general, the plant is partitioned into multiple zones according to its physical characteristics. The various zones are also connected via gateways. Each zone contains a subset of the sensors and actuators that are linked to its gateway by a network of links and *routers*. For simplicity, Figure 8.1(b)) shows only a single zone: here the control network is made of six sensors, three actuators, and three routers that are linked via the gateway to the backbone network.

[1]These computers are given different names in different application areas, like *direct digital controls* in building automation, *programmable logic controllers* in industrial automation, and *electronic control units* (ECU) in automotive electronics.

This two-tier architecture is not the only possible network organization, but it is becoming increasingly popular for many important applications including *heating, ventilation and air-conditioning (HVAC)* control systems [67]. Generally, the goal of feedback control in a HVAC system is to regulate physical quantities such as temperature, humidity, and pressure to optimize an indoor environment for human comfort (comfort HVAC) or for machine operations (industrial HVAC) while minimizing operation, installation, and maintenance costs. The design of the communication network plays an increasingly important role in reaching these goals.

Once distributed over a set of computers interconnected by a backbone LAN, the overall control system requires that the worst-case computation time t_C be bounded. The messages from(to) the sensors(actuators) need to cross the gateway that accounts for a worst-case communication delay equal to t_G. Since a controller C_i can tolerate a loop delay not greater than its sampling period Δt_i, the design of the control networks must satisfy a set of real-time constraints like $(t_{Ri} \leq \Delta t_i - t_C - t_G)$ while guaranteeing that all required messages are gathered from the sensors and delivered to the actuators. In addition, the network cost (given by the sum of the costs of its components and of the installation costs) should be minimized.

Today it is standard practice to deploy the networked embedded system first on a predefined distributed architecture chosen on the basis of experience and heuristic considerations and then tweak the software implementation of the control algorithm to meet latency, bandwidth, and reliability requirements. This tuning phase can become a nightmare for engineers and technicians because the behavior of the control algorithm is affected by the loop delay that is the result of computation and *communication* delay. To detangle the behavior of the control algorithm from the communication delay, the network is often over-designed. This is far from ideal, since many systems are highly

cost sensitive and using a network that is not tailored to the application and not optimized is clearly expensive. Moreover, the complexity of large networked embedded systems continues to increase, thus making heuristic and experience-based design practices inadequate. Two factors contribute to the complexity of these systems:

- The scale of control networks for the automation of large buildings is of the order of thousands of sensors distributed on a surface of hundreds of thousands of square meters.
- A rich variety of alternative protocols and technologies are available to build such networks [67]. The number of alternative implementations that can be constructed by assembling heterogenous components represents a great opportunity to match the network and the communication requirements with the intent to reduce cost. Unfortunately, the difficulties encountered in handling heterogeneous networks represent an obstacle that cannot be easily overcome by experience. This is the reason why systems tend to be homogenous and often sub-optimal.

We apply our methodology with the intent of assisting engineers in the design process so that the final implementation satisfies specifications while taking into consideration the overall cost of deployment including development cost and time.

8.1 Specification

The specification of a communication synthesis problem for building automation systems is divided in two parts: the specification of the communication constraints and the definition of the environment in which the network is embedded. The environment consists of the geometry of the

building in terms of walls and cable ladders[2] and a way of capturing them is explained in details in Section 8.2. In this section we focus on capturing the communication constraints.

The specification is comprised of nodes that are sensors, actuators and gateways, and end-to-end communication constraints. Each constraint is characterized by a message period, a message length and a maximum latency. Therefore a specification is a communication structure $N_C(\mathscr{C}_C, \mathbf{q}_C, L_C)$, where vector of quantities $\mathbf{q}_C = (\tau, x, y, z, t, b, m)$ contains the type of the nodes τ, the variables x,y and z that represent the position of the nodes in the Euclidean space, the maximum latency t in seconds, the message frequency b in number of bits and the message length m in bits.

As discussed in the introduction of this chapter, a building automation system (BAS) is usually partitioned into multiple gateway zones according to the physical characteristics of the building. Typically a gateway zone coincides with a floor and the gateways can only be installed in specific closets [67]. The computers processing the control algorithms are also typically installed in pre-determined locations. The control network for the entire building is then obtained as the composition of the control network of each gateway zone and the high-speed backbone LAN (that can also be synthesized using the same techniques that we show in the rest of this chapter). Figure 8.2 illustrates a gateway zone for a simplified version of a building automation system. The floor of the building measures $30 \times 20\ m^2$ and the ceiling height is $3m$. In Figure 8.2, $A = \{a_1, ..., a_{10}\}$ is the set of actuators that are placed at the ceiling level, and $S = \{s_1, ..., s_9\}$ is the set of sensors that are placed $1.3m$ above the floor. The gateway is placed on the north wall and up to four routers may be installed on the other walls. Each potential router in the set $R = \{i_1, \ldots, i_4\}$ has an associated fixed position $p(i_j)$.

[2]Other details can be captured. We capture walls and cable ladders only since these two elements are the ones that most affect the performance and cost of a network.

A gateway zone contains a gateway g, a set S of sensors and a set A of actuators. Therefore, for this simple example, the nodes contained in the specifications are $V_C = \{g\} \cup S \cup A$, and the links are $E_C = (S \times g) \cup (g \times A)$. Sensors and actuators are connected to the gateway through switches and routers. The number of these intermediate nodes within the control network may vary as well as their positions. The number of possible nodes positions, however, is typically limited since they must be easy to access and kept away from possible hazards. In fact, the choice of how many nodes to install and where to install them is part of the design of the control network and does affect its cost. For wired networks, sensors and actuators are connected to routers on simple networks implemented with twisted-pair wire technology, while the links between the routers and the gateway offer relatively high-bandwidth and low-latency. Typically, a bus connects a subset of the sensors and the actuators and one router such that nodes on different busses can communicate trough the routers that are connected by high bandwidth links. The choice of a bus standard and the length of the wires implementing the link affects directly the cost of the control network. Various protocol standards at different layers of the OSI model are available to control these busses like BacNet[87, 24], LonWorks [41], and ARCNET [1]. Wireless technology is a promising alternative option for future implementations since the installation cost of a network can be potentially reduced. However, wireless communication is less reliable than wired communication Therefore extra attention has to be paid at design time to make sure that the location of wireless nodes, transmission power, routing strategy and access control mechanisms are configured to guarantee the end-to-end specification constraints.

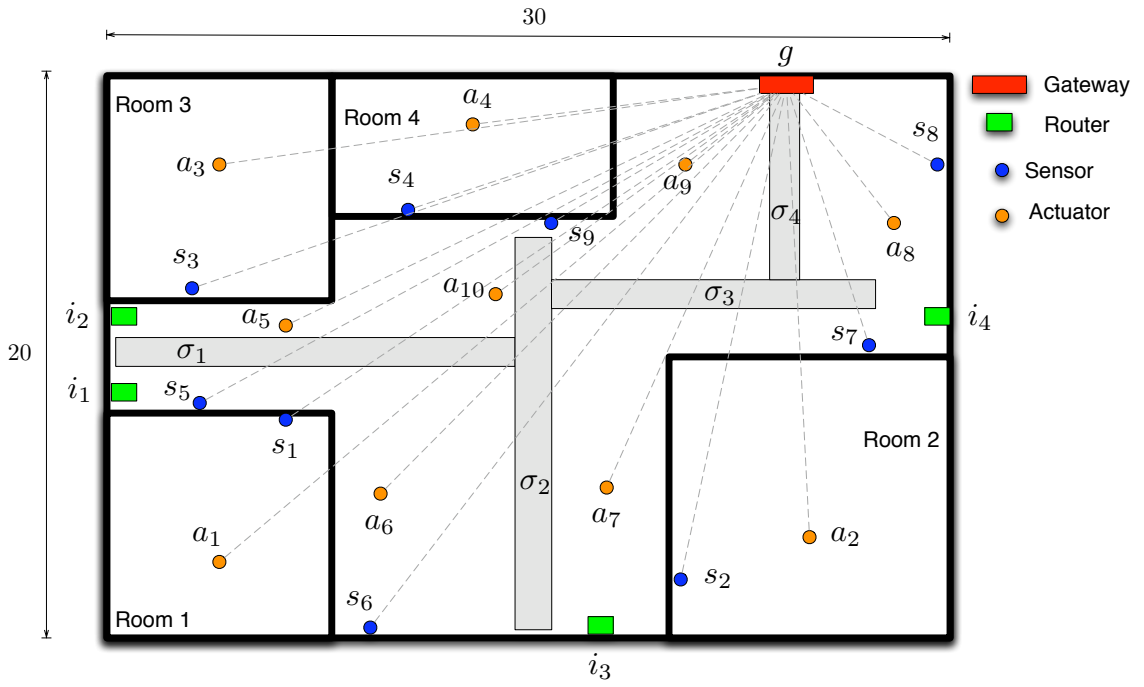


Figure 8.2: Example of gateway zone associated to a building floor.

8.2 Capturing the Building Geometry

The cost of a building automation network is affected by the building geometry. The cost of a wired network depends on the wire length of each link that can be computed only if the wire layout is known. The cost of a wireless network depends on the number of nodes and their power consumption. The number of nodes, as well as their transmitting power, depends on the path loss experienced by the wireless signal. Obstacles, like walls, contribute to the path loss and may require the use of intermediate wireless router to function as repeater such that connectivity is guaranteed. Therefore, we need a way of capturing the geometry of a building with a model that is amenable to fast computation of wire length and path loss. We capture a building as a set

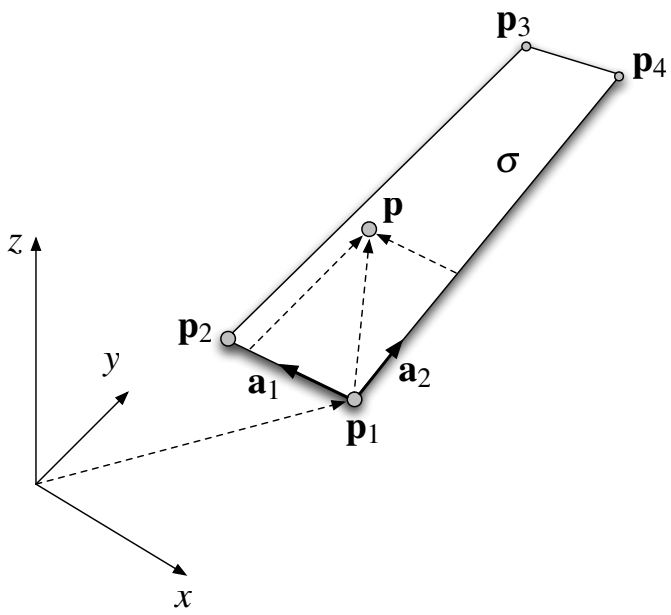


Figure 8.3: Representation of a two-dimensional face with four vertexes in the Euclidean space.

of two-dimensional faces with four vertexes (in a three-dimensional space). First, we introduce a parametric representation of this type of faces. Then, we show some geometric relations that will be useful in later sections. Figure 8.3 shows a reference system in the Euclidean space with a face σ. The four vertexes of the face are named $\mathbf{p}_1$, $\mathbf{p}_2$, $\mathbf{p}_3$ and $\mathbf{p}_4$. The convention that we use is the following: with $\mathbf{p}_1$ we denote the vertex that is closest to the origin of the reference system; $\mathbf{p}_2$ and $\mathbf{p}_4$ are the vertexes adjacent to $\mathbf{p}_1$; $\mathbf{p}_3$ is the remaining vertex. The two unit vectors $\mathbf{a}_1$ and $\mathbf{a}_2$ have the same directions as the two edges of the face that are incident to $\mathbf{p}_1$. In particular, these two unit vectors have the following expressions:

$$\mathbf{a}_1 = \frac{\mathbf{p}_2 - \mathbf{p}_1}{||\mathbf{p}_2 - \mathbf{p}_1||} \ , \ \mathbf{a}_2 = \frac{\mathbf{p}_4 - \mathbf{p}_1}{||\mathbf{p}_4 - \mathbf{p}_1||}$$

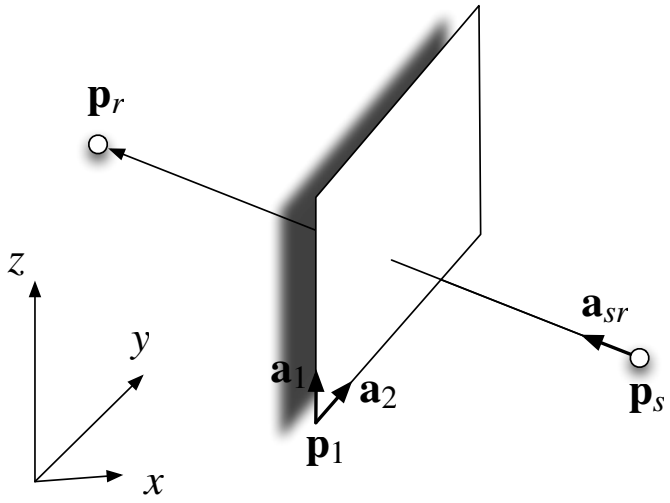


Figure 8.4: Intersection of a ray with a face.

Any point $\mathbf{p}$ on the two-dimensional face can be written as linear combination of the the two unit vectors as follows:

$$\mathbf{p} = \mathbf{p_1} + t_1\mathbf{a}_1 + t_2\mathbf{a}_2,\ 0 \leq t_1 \leq ||\mathbf{p}_2 - \mathbf{p}_1||,\ 0 \leq t_2 \leq ||\mathbf{p}_4 - \mathbf{p}_1||$$

When a face is used to capture the presence of a wall, we also add the thickness information. Similarly, we can write the parametric equation of a segment in a three dimensional space as follows. Let $\mathbf{p}_s$ and $\mathbf{p}_r$ be the two extreme points of the segment. The set of points belonging to the segment is defined by the following parametric equation:

$$\mathbf{p} = \mathbf{p}_s + t\mathbf{a}, \quad \mathbf{a} = \frac{\mathbf{p}_s - \mathbf{p}_r}{||\mathbf{p}_s - \mathbf{p}_r||}, \quad 0 \leq t \leq ||\mathbf{p}_s - \mathbf{p}_r||$$

These representation provides an inexpensive way of computing the intersection of two segments and the intersection of a segment with a face. Consider a segment from a point $\mathbf{p}_s$ to a point $\mathbf{p}_r$ and a face σ. Figure 8.4 depicts this setting. A point is common to the line and to the face

if and only if we can find three parameters t_1, t_2 and t such that:

$$\mathbf{p}_s + t\mathbf{a}_{sr} = \mathbf{p_1} + t_1\mathbf{a}_1 + t_2\mathbf{a}_2, \;\; 0 \leq t_1 \leq ||\mathbf{p}_2 - \mathbf{p}_1||, \; 0 \leq t_2 \leq ||\mathbf{p}_4 - \mathbf{p}_1||, 0 \leq t \leq ||\mathbf{p}_r - \mathbf{p}_s||$$

This is a system of three equations in three unknowns. If the system can be solved and the three parameters satisfy the inequality constraints, then an intersection point exists and can, indeed, be computed.

Consider another segment from $\mathbf{p}_u$ to $\mathbf{p}_v$. The following system defines the intersection of two segments:

$$\mathbf{p}_s + t_1\mathbf{a}_{sr} = \mathbf{p_u} + t_2\mathbf{a}_{uv}, \;\; 0 \leq t_1 \leq ||\mathbf{p}_s - \mathbf{p}_r||, \; 0 \leq t_2 \leq ||\mathbf{p}_u - \mathbf{p}_v||$$

This is a system of three equations in the two unknowns t_1 and t_2. To find a solution, we project the system on the x-y plane. If the 2 by 2 system does not have a solution or if the two parameters do not satisfy their inequalities, then the two segments do not intersect. Otherwise, we check if the two parameters t_1 and t_2, found after the projection of the system, also satisfy the equation along the z axis. If this is the case, then we found an intersection point, otherwise the two segments do not intersect.

We rely on the computation of the intersection point between two segments to compute the actual wire layout between two points for a given set of cable ladders (captured by horizontal faces located at the ceiling level). Figure 8.5 shows the four cable ladders from the example of Figure 8.2. First, we compute the intersection points for each pair of surface ($\mathbf{x}_1$, $\mathbf{x}_2$ and $\mathbf{x}_3$ in this example). Then, we define a *connectivity graph* that has two types of vertexes: *Intersection points* (shown as large nodes in Figure 8.5) that have a fixed position, and *ancillary nodes* representing any point on a surface (shown as small circles in Figure 8.5). An ancillary node of a surface σ_i is connected to an intersection point $\mathbf{x}_j$ if the point belongs to the surface.

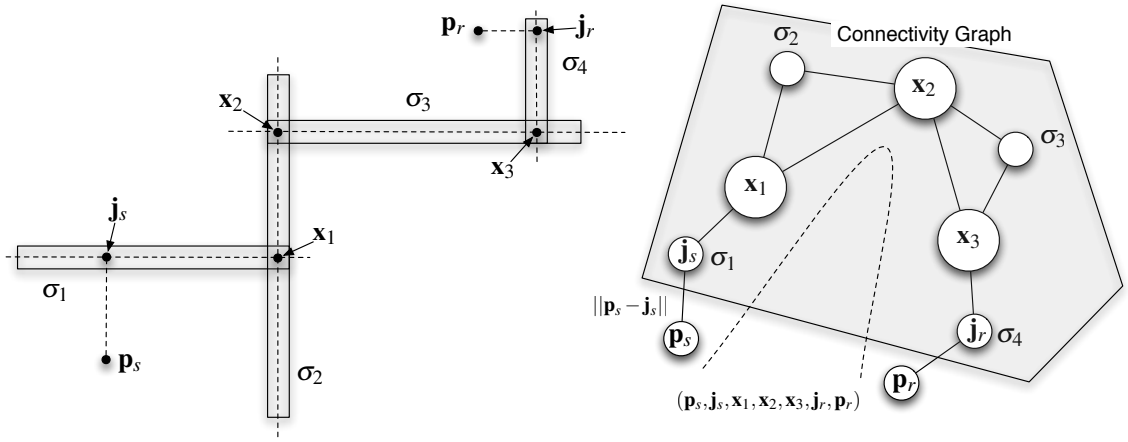


Figure 8.5: Construction of the connectivity graph and example of computation of wire layout for the example of Figure 8.2.

Consider two points $\mathbf{p}_s$ and $\mathbf{p}_r$. To compute the wire layout that connects them, we first compute the closest points $\mathbf{j}_s$ and $\mathbf{j}_r$ to $\mathbf{p}_s$ and $\mathbf{p}_r$, respectively. These points must belong to cable ladders. Let σ_s and σ_r be the two surfaces representing the cable ladders, respectively (these two surfaces are σ_1 and σ_4 in the example of Figure 8.5). To compute the actual path followed by the wire from $\mathbf{p}_s$ to $\mathbf{p}_r$, we set the position of the ancillary node σ_s to $\mathbf{j}_s$ and of the ancillary node σ_r to $\mathbf{j}_r$. Finally, we compute the shortest path between $\mathbf{p}_s$ and $\mathbf{p}_r$ on the connectivity graph where the length of the edges is simply the distance between the positions associated to its extreme nodes. In the example of Figure 8.5, the wiring layout follows the list of points $(\mathbf{p}_s, \mathbf{j}_s, \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{j}_r, \mathbf{p}_r)$.

8.3 Wired Networks

In this section we explain the sequence of steps that should be followed to define a synthesis flow for wired networks for building automation systems. We first define the platform, which entails defining the library of communication components and the composition rules (Section 8.3.1).

Then, we formulate an optimization problem that aims at finding an optimal implementation of the communication architecture supporting the control application (Section 8.3.3).

The cost of a wired network is mainly attributed to the installation cost. This cost is the cost of installing the components plus the cost of installing the wired connections. Because the installation of sensors and actuators is mandatory (and it is given as part of the specification), their cost is a fixed constant in the optimization problem. The variables of the optimization are the position and number of routers to install, and for each router, the subset of sensors and actuators that connect to it. Therefore, the problem is to select the minimum number of sub-networks such that each node in the specification is covered. Notice that there is a limit on the number of nodes and total bandwidth for a sub-network. Moreover, depending on these two quantities, the delay incurred by messages changes.

Further restrictions are imposed on the topology of the sub-networks. In practice, independently of the protocol of choice, the suggested topology for the physical implementation of the network is the *daisy-chain bus*. The main reason behind this choice is the impedance matching that can be performed by installing simple devices at the end of the chain. Due to their ubiquity, we focus our attention on daisy-chain bus topologies. Nevertheless, the extension to other kinds of topologies is supported by the framework and impact the formulation and efficiency of the algorithm. In fact, in Section 8.4.2, we synthesize wireless networks with tree topologies.

Figure 8.6 shows an example of wire layout for a daisy-chain bus that connects a sensor and two actuators to a router. The layout is constrained by the network topology and the building structure. The standard way of laying out wires relies on raceways or cable ladders that are installed along the building aisles. Special conduits are used to bring wires from the nodes to the cable

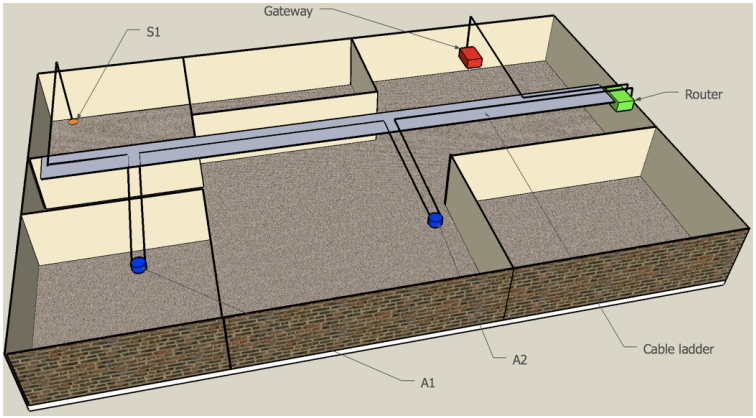


Figure 8.6: Example of daisy-chain wiring of a few components in a building automation system.

ladders. We capture these constraints with a set Σ of rectangular surfaces in the Euclidean space. We constraint wires to only travel on these surfaces. Wires from nodes are first laid out to the closest raceway and then towards their destination. For the example of Figure 8.2, the set of surfaces is $\Sigma = \{\sigma_1, .., \sigma_4\}$. They are about one meter wide and disposed at the ceiling level.

8.3.1 Library of Communication Components

There are many networking technologies that are appealing solutions for building automation networks. The two most common wired protocols are BACNET [24] and LONWORKS [41]. These two protocol define layer 3 and above[3] of the OSI protocol stack [8] and can use many LAN technologies for low level communication (i.a medium access control and physical layer). For a good review of control busses, we refer the interested reader to [112, 119].

We are interested in optimizing the network cost while guaranteeing performance constraints. The cost of the network depends mainly on the wiring, while the performance of a bus

[3]Level 1 is the physical layer and level 2 is the data link layer.

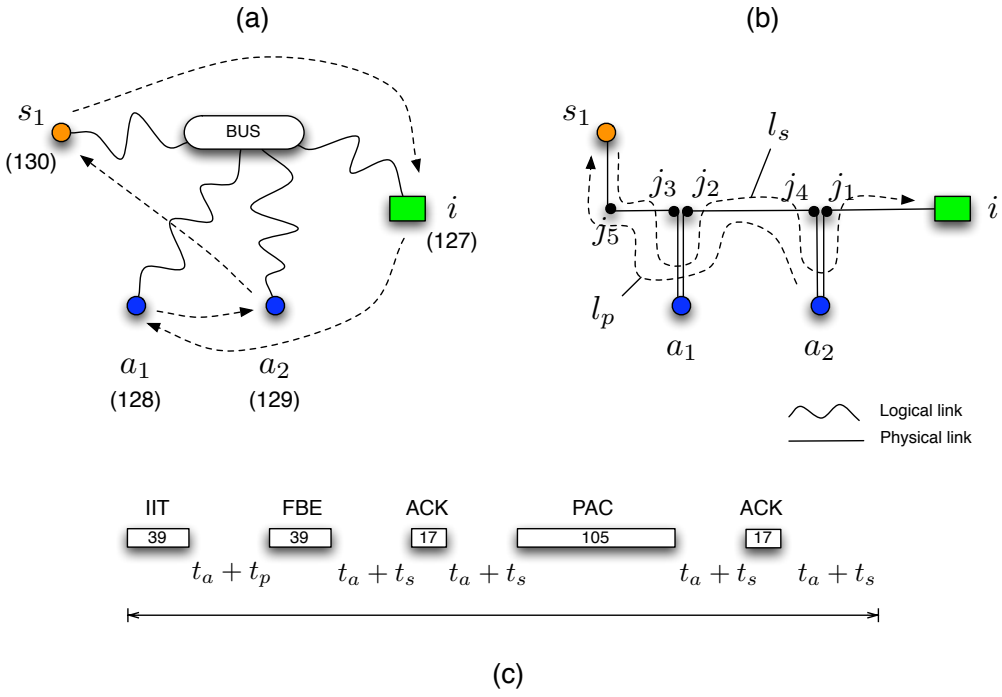


Figure 8.7: Graphical representation of a daisy-chain bus: (a) logical network, (b) physical network, (c) sequences of messages generated by the token passing protocol for a short packet transmission.

depends mainly on the characteristics of the medium access control protocol. Therefore, we need accurate models for the wire length (i.e. the distance between two nodes) and the timing behavior of the bus protocol. A simple way to guarantee deterministic timing properties is to use a token passing scheme among the bus participants. One such protocol is ARCNET [1]. ARCNET is a *token passing bus* with deterministic performance that can operate at different speeds ranging from 19*Kbps* up to 10*Mbps* (but optimized for 2.5*Mbps*). A token passing bus (Figure 8.7(a)) is a centralized communication system where nodes are logically organized in a ring. A node can send messages only when it holds the token. The token is passed from one node to its logical neighbor which is the one with the next highest address.

To model the performance of a set of LonWorks components connected in a daisy-chain

on an ARCNET bus, we need to analyze the token passing bus protocol. Consider the case where sensor s_1 sends a short packet to router i. A successful transmission of a message requires a sequence of protocol messages that includes: a token pass (Invitation to Transmit IIT), a Free Buffer Enquiry (FBE), an Acknowledge (ACK), a Packet (PAC) and a final ACK. Figure 8.7(c) shows each message in the sequence annotated with its length in number of bits (in this example we assumed that the payload of the message contains one byte only).

Between one protocol message and the next one, two other delays contribute to the total communication delay: the response time t_a of the chip that implements the protocol interface, and the propagation delays t_p and t_s relative to the signal traveling distances l_p and l_s, respectively (Figure 8.7(b)). The real distance between them depends on the physical path traveled by the twisted-pair wires. The distance does not only affect performance, but also cost. In fact, the wire installation cost is a major concern in the design of building automation networks especially in retrofitting of installations. Given two points in the Euclidean space $p_1 = (x_1, y_1, z_1)$ and $p_2 = (x_2, y_2, z_2)$, the length of a link connecting them can be defined at different abstraction levels. For instance, for a given ceiling level h, we could define the distance as $d(p_1, p_2) = |z_1 - h| + |z_2 - h| + ||(x_1, y_1) - (x_2, y_2)||_1$ if $p_1 \neq p_2$, and zero otherwise. This definition captures the fact that each vertex must be wired to the ceiling first. The use of the L^1-norm captures the fact that wires follow straight lines, but it does not capture the real layout of the wires. In fact, the effective distance between any pair of elements of the control network is neither the L^1- norm, i.e. the Manhattan distance, nor the L^2-norm, i.e. the Euclidean distance. We adopt the more refined model presented in Section 8.2. We first compute the distance from p_1 to the closest point q_1 in space that belongs to a raceway. Then, we compute the distance from p_2 to the closest point q_2 that belongs to a raceway. Finally, we derive the actual

layout of the wire between q_1 and q_2 and we compute its length. The final distance between p_1 and p_2 is obtained by adding up these three contributions.

We define a *wiring path* as a sequence of points in space and we assume that wires follow a straight line between two points. For instance, in Figure 8.7(c) the wiring path between a_1 and s_1 is $\pi_w(a_1, s_1) = (j_3, j_5)$. The length of this path is

$$|\pi_w(a_1, s_1)| = ||p(a_1) - j_3||_1 + ||j_3 - j_5||_1 + ||j_5 - p(s_1)||_1$$

The physical lengths in the example of Figure 8.7(b) are $l_s = |\pi_w(s_1, a_1)| + |\pi_w(a_1, a_2)| + |\pi_w(a_2, i)|$ and $l_p = |\pi_w(a_2, a_1))| + |\pi_w(a_1, s_1)|$. Therefore, when we instantiate an ARCNET bus, we can attach a quantity to each wire that represent its wiring path and compute the latency and cost of each daisy-chain bus. The set of parameters defined in Table 8.1 are used to compute cost and performance of a wired network based on LonkWorks technology. (where the delay of $12.6\mu s$ refers to t_a). From this table, we observe that a bus must satisfy many constraints like the maximum number of nodes that can connect to it (i.e. the *degree*) and the maximum length of the daisy chain. These two parameters change depending on the speed of the bus. The cost of a daisy chain bus can be computed by adding together the cost of each node plus the wiring cost. Notice that, for each component, the installation cost must also be taken into account.

The delay between two nodes is the sum of two contributions: the time required to acquire ownership of the bus, and the delay incurred by a message on the path from its source to its destination. Notice that the source and the destination of a packet may be attached to different busses which makes the computation of the delay not trivial. For toke ring busses, the time to access the bus is called *token rotation time*. Figure 8.8 shows a typical configuration of a bus with k participants. The token rotation time can be computed as follows. The total number of bits sent on the

Component	Performance	Cost
BUS (twisted-pair)	Degree : 8 Length: 120m Delay: 5.5ns/m Bandwidth: 2.5$Mpbs$	Price: \$0.6$/m$ Inst.: \$7$/m$
Router	Delay: 320ns	Price: \$500 Inst.: \$240
Sensor	Delay: 12.6μs	Price: \$110 Inst.: \$50
Actuator	Delay: 12.6μs	Price: \$200 Inst.: \$50

Table 8.1: Characterization of the intrinsic performance and cost of a realistic library of components for building automation systems.

bus during a token rotation (i.e. from the token being passes to v_1 to the token returning to v_1) is the sum of the bits sent by all nodes on the bus. To send an m bit message on the bus, a software task running on a node sends the the message to the protocol interface which adds several protocol related information. Figure 8.7 shows the protocol phases and their length in bits that are IIT (39 bits), FBE (39 bits), ACK (17 bits) and PAC, whose length depends on the length of the payload. Moreover, each byte of the payload is protected with 3 additional bits for error checking purposes. Thus, the number of bits injected on the bus by a sender can be computed as follows:

$$IIT + FBE + ACK + PAC + ACK = 2 \cdot 39 + 2 \cdot 17 + PAC + 11 \cdot \left\lceil \frac{m}{8} \right\rceil$$

where $PAC = 94$ if the message is shorter than 256 bytes and $PAC = 105$ if the message between 256 and 507 bytes. If a node does not send messages but only receives them, it will just pass on the token and contribute with a total number of bits equal to 39. Let M be the total number of bits sent on the bus for a complete rotation of the token and let $speed$ represent the bus speed. The token

rotation time can be approximated by $M/speed$. It is possible to refine this model adding also the physical delays like t_a and the signal propagation time.

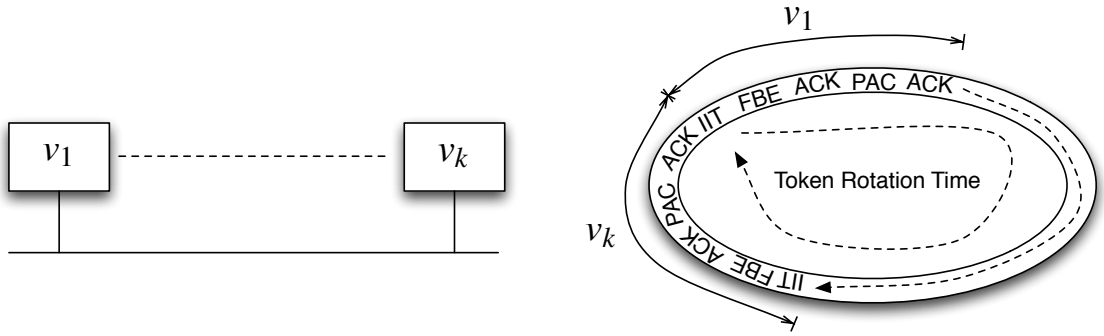


Figure 8.8: An example of a token ring bus with k participant v_1,...,v_k and the graphical representation of the token rotation time.

Consider the case where each node on the bus sends one byte to a central node also on the same bus. The number of bits required for each message is 217, therefore if we use ARCNET at $2.5Mbps$, the time required to sent the message is $217/2.5 = 86.8\mu s$ plus five times t_a, plus the propagation delays. Multiplying this value by the number of nodes on the bus gives the token rotation time. To compute the maximum number of messages per second that a bus can support, we proceed as follows. Consider an ARCNET bus configured at $2.5Mbps$. The number of bits necessary to send a message of one byte is 217, therefore we obtain that at most $2.5 \cdot 10^6/217 = 11520$ packets per second can be sent on the bus.

The delay between two nodes can be computed as follows. If two nodes v_i and v_j are on the same bus, the delay between them is the token rotation time of that bus. If the two nodes are on two different busses, BUS_i and BUS_j, respectively, then the delay between them is the token rotation time of BUS_i plus the token rotation time of BUS_j plus the delay of the higher level network that connects the two busses. As discussed in the introduction to this chapter, the higher level network

is usually much faster than the busses and its delay can often be neglected.

The token rotation time can be interpreted as the worst case access time of a node to a bus. This time is computed by an indirect model that, given the configuration of the bus, computes the token rotation time. We denote this model by trt.

8.3.2 Communication Platform and Implementation

The platform is characterized by the vector of quantities $\mathbf{q}_P = (\tau, x, y, z, \pi_w, \gamma)$ where the triple (x, y, z) represents the position of nodes, τ represents the type of nodes, γ is the number of bits per second sent by a node, and π_w represents the wiring path associated to each link.

The composition rules $\mathscr{R}_P$ that must be satisfied by any composition of platform instances are the follows. Given a pair $(\mathscr{C}_P, L_P)$ of components and configurations of a platform instance the following topological constraints must be satisfied:

- The result of a composition must be a set of daisy-chains. These constraints are described formally as follows. The degree of each vertex $deg(v)$ (i.e. the number of links incident to the vertex) is at most 2. Since we don't want isolated vertexes to be instantiated, the degree must be at least one. These two constraints are called *degree constraints*. Self loops are not allowed. The degree constraints are not enough since they might leave space for ring structures. The additional constraint is that the number of links must be less than or equal to the number of vertexes minus one. This constraint is called *topology constraint*. Summarizing:

  Degree constraint : $1 \leq deg(v) \leq 2, \quad \forall v \in V_P$

  Topology constraint : $|E_P| \leq |V_P| - 1$

- If $|E_P| = |V_P| - k$, then the number of connected components of a platform instance is equal to

k. Since the maximum number of nodes connected to a bus is limited by a maximum degree n_{max}, the following must hold:

$$\left\lceil \frac{|V_P|}{|V_P| - |E_P|} \right\rceil \leq n_{max}$$

- The total length of the wires in each connected component must be less than or equal to l_{max}. Let N_P^i be the i-th connected component in N_P, corresponding to a bus. Then, the total length of the bus is the sum of the length of the wiring paths on all the edges. The following must hold for any connected component $N_P^i \leq_{Q_P} N_P$:

$$\sum_{e \in E_I^i} |l[\pi_w](e)| \leq l_{max}, \quad \forall l \in L_P$$

- Let $V_B \subset \mathscr{C}_P$ be the set of vertexes on a connected component of N_P representing a daisy chain bus. Then the following must hold:

$$\sum_{v \in V_B} l_P[\gamma](v) \leq speed, \ \forall l_P \in L_P$$

The implementation $N_I(\mathscr{C}_I, \mathbf{q}_I, L_I)$ is characterized by the vector of quantities $\mathbf{q}_I = (\tau, x, y, z, t, b, m, \pi_w, \gamma, d)$ where d is the access time to a bus. For the node $v \in V_I$ and a configuration $l_I \in L_I$, the quantity $l_I[d](v)$ is indirectly derived and must satisfy the constraint $l_I[d](v) = trt(\mathscr{C}_I, l_I, v)$.

8.3.3 Optimization Algorithm

In this section we present our approach to solve the communication synthesis problem for wired networks in building automation systems. First, we define the conditions under which an implementation satisfies a specification. Let N_C denote the communication specification. Also, let N_I denote the communication implementation that we assume to be the composition of a set of

daisy chain busses and a higher level network. Therefore, we assume that $N_I = N_I^1||...||N_I^n||N_I^H$. To ensure connectivity, each component N_I^i must contain a router among the ones given as input to the problem. Let $r_i \in V_I^i$ be the router associated with bus N_I^i.

The implementation satisfies the specification if the following constraints are satisfied:

- For all constraints $e(u,v) \in E_C$ and for all configurations $l_C \in L_C$

$$l_I[\gamma](u) \geq \left(2 \cdot 39 + 2 \cdot 17 + PAC + 11 \cdot \left\lceil \frac{l_C[m](e)}{8} \right\rceil\right) l_C[b](e)$$

  This constraint ensures that the node is able to send at least as many bits as required by the specification. To make this rule more general, we introduce a model for the bandwidth (in bits per second) that, given the message length and the message frequency, returns the total bandwidth taking into account the protocol overhead. We denote the model by a function $bandwidth(\mu,\beta)$ of the message length μ and the message frequency β, and write $l_I[\gamma](u) \geq bandwidth(l_C[m](e), l_C[b](e))$.

- For all constraints $e(u,v) \in E_C$ and for all configurations $l_C \in L_C$, if u belongs to chain N_I^i and v belongs to chain N_I^j, then the following must hold:

$$l_I[d](u) + l_I[d](r_j) \leq l_C[t](e)$$

  If u and v are on the same bus then:

$$l_I[d](u) \leq l_C[t](e)$$

Clearly, the router is essential to allow communication among different busses. Each router contributes to the bus traffic by injecting messages sent to nodes that belong to the bus from nodes that belong to other busses.The message frequency at which the router injects messages on a bus can be

computed as follows. For a given specification configuration l_C, the aggregate message frequency of router r_i is:

$$\beta_i = \sum_{(u,v) \in E_C : v \in \mathscr{C}_I^i \wedge u \notin \mathscr{C}_I^i} l_C[b](e)$$

Also let the message length be defined as:

$$\mu_i = \max\{l_C[m](e) : e(u,v) \in E_C \wedge v \in \mathscr{C}_I^i \wedge u \notin \mathscr{C}_I^i\}$$

Therefore the following constraint must hold:

$$l_I[\gamma](r_i) \geq bandwidth(\mu_i, \beta_i)$$

In deploying a bus, the algorithm needs to take into account the composition rules defined in Section 8.3.2 and needs to make sure that bandwidth and latency requirements are satisfied. We say that a daisy-chain, or simply a chain, is valid if it satisfies all the aforementioned constraints. Given a specification N_C, a valid network implementation is *a set of valid daisy chains, each containing exactly one router, such that each node in $\mathscr{C}_C$ is contained in exactly one chain*.

We solve the communication synthesis problem with a two-step approach: (1) chain generation and (2) chain selection. The chain generation algorithm is in charge of generating a set of valid chains while the chain selection algorithm picks a subset of these chains such that each node in the specification is contained in exactly one bus.

Algorithm 3 is a greedy algorithm that generates minimum-length valid chains. We use the example in Figure 8.9 to explain how the algorithm proceeds.

For each possible router position r, the algorithm starts by creating a chain N_I that initially contains only a router vertex (line 1) in position p_r. Then, it tries to expand this chain by attaching other vertices, each representing a node in the specification (i.e. either a sensor, an actuator or a

Algorithm 3: Find all minimum-length valid chains

Input: Available routers $R = \{r_1, ..., r_m\}$ with associated positions p_{r_i}; Specification N_C
Output: Set of valid daisy chain busses C
forall $r \in R$ **do**
  $A[v] \leftarrow false, \forall v \in V_C$
1 Initialize N_I with r in position p_r
  $Extended \leftarrow true$
2 **while** $Extended$ **do**
    $Extended \leftarrow false$
3 $(u^*, v^*) \leftarrow \arg\min_{v \in V_C : A[v] = false \wedge u \in V_I \wedge deg(u) \leq 1} d(u,v)$
    Instantiate $N_I'(\mathscr{C}_I', \mathbf{q}_I, \{l_I'\}$ such that:
    $\mathscr{C}_I' = \{(u^*, v^*), u^*, v^*\}$
    $l_I'(u^*) = l_I(u^*)$ and $l_I'[(\tau, x, y, z)](v^*) = l_c[(\tau, x, y, z)](v^*)$
    $\gamma_v \leftarrow \sum_{(v^*, z) \in E_C} l_C[b](v^*, z)$
    $\mu_v \leftarrow \max\{l_C[m](v^*, z) : (v^*, z) \in E_C\}$
    $l_I'[\gamma](v^*) =$ `Bandwidth(`μ_v, γ_v`)`
    $l_I'[\pi_w](u^*, v^*) =$ `WiringPath(`u^*, v^*`)`
    $N_I \leftarrow N_I || _{Q_I} N_I'$
    `UpdateRouter(`N_I,r`)`
    `UpdateDelay(`N_I,trt`)`
    $A[v^*] \leftarrow true$
    $C' \leftarrow \emptyset$
4 **forall** $N_I^i \in C(i)$ *s.t.* $u \in V_I^i$ *and* $deg(u) \leq 1$ **do**
5 $N_I'' \leftarrow N_I^i || _{Q_I} N_I'$
      `UpdateDelay(`N_I'',trt`)`
6 **if** N_I'' *is valid* **then**
        $C' \leftarrow C' \cup \{N_I''\}$
        $Extended \leftarrow true$
    $C(i) \leftarrow C(i) \cup C'$
return C

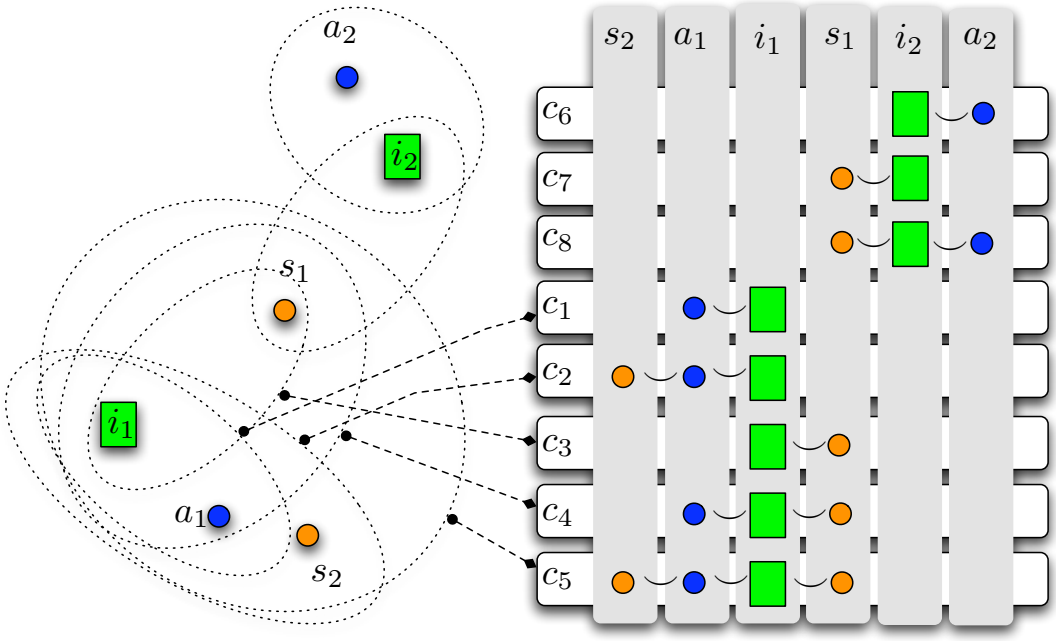


Figure 8.9: The chains generated by Algorithm 3 and the resulting covering matrix.

controller), that may end up being "covered" by router r. If this attempt fails then the chain is discarded as an invalid chain. Otherwise, it is included in the set of valid chains that are passed to chain selection algorithm. An array A is used to track those vertices that have been already considered to extend chains. At each iteration of the main loop, the algorithm attempts to select a vertex v^* that is closest to one of the extreme vertexes of the chain and that has not been considered yet (lines 3). The domain of arg min contains those vertexes with degree less then or equal to 1, which ensures that new vertexes are only attached to the extreme of a chain. For instance, consider router i_1 in Figure 8.9. During the first iteration actuator a_1 is selected as the the closest vertex to extend the chain of i_1. Chain c_1 that only covers vertex a_1 is generated.

On line 4, the algorithm starts analyzing all the previously generated chains that should be extended. If node v^* is to be added to the chain, and u^* is the extreme node of the chain that is closest to v^*, then among the previously generated chains only the ones containing u^* as one extreme vertex are considered for extension.

On line 6, the algorithm checks if the chain can be extended with the new vertex v^*. Checking if the extension is possible means checking that it satisfies the composition rules and the constraints of bandwidth, length, degree and delay. If this is not the case the chain is not extended. All newly generated chains are saved in the set C' and eventually added to $C(r)$.

In the example of Figure 8.9, chain c contains a_1 and i_1. The closest vertex to a_1 is s_2, while the closest to i_1 is s_1. Since s_2 is closer to a_1 than s_1 is to i_1, the algorithm extends the chain by attaching s_2 to a_1 and generating the new chain c_2. Chain c now contains s_2, a_1 and i_1. The next vertex is s_1 which is closer to i_1. Therefore, the algorithm extends all chains in $C(i_1)$ with i_1 as extreme vertex. The newly-generated chains are c_3, c_4 and c_5.

If an extension violates the constraints, then the extended chain is not generated and, therefore, not added to the set of chains of a router r. Algorithm 3 returns a set of valid chains each covering a subset of the sensors and actuators and having a cost associated with them. This can be directly translated into a covering matrix for the chain selection algorithm.

In the example of Figure 8.9, chains c_6, c_7 and c_8 are generated for router i_2. Vertexes s_2 and a_1 are only covered by router i_1, while vertex a_2 is only covered by router i_2. Hence, in this case, both routers are essential and must be installed. Which of the two routers will cover sensor s_1 depends on the cost of the chains. For this example, the only two possible solutions are $C' = \{c_2, c_8\}$ and $C'' = \{c_5, c_6\}$. The cost of C' is $f_2 + f_8$ while the cost of C'' is $f_5 + f_6$. The chain

selection algorithm will select the least cost solution.

The complexity of the chain generation algorithm depends on the maximum degree of the chains. Let D denote the maximum degree. The main loop starting at line 2 is executed at most $2 \cdot D$ times corresponding to D left and D right extensions. Each loop iteration removes one vertex from the set of sensors and actuators to be covered. Also, at most $|C(i)|$ new chains are generated at each iteration. Therefore, the number of basic operations in the main loop is at most

$$\sum_{i=1}^{2 \cdot D} [(|S| + |A| - i) + (i+1)]$$

where S and A are the sets of sensors and actuators, respectively. For $|S| + |A| \gg D$ the complexity is $O(D(|S| + |A|))$. The maximum number of chains that are generated by the algorithm is $(D+1)(D+2)/2 - 1$.

The second step of our algorithm consists in selecting a set of chains to cover all the nodes of a specification N_C. Let $C = \{N_I^1, ..., N_I^n\}$ be the set of all valid chains and f_i be the cost of chain N_I^i. Let $z_j \in \{0,1\}$ be a binary variable that evaluates to one if chain N_I^j is taken in the final implementation. Also, we define two other binary variables (that are not decision variables in the optimization problem, but are computed statically from the set of chians). Let x_{vj} be 1 if $v \in V_I^j$ and 0 otherwise, and let y_{rj} be 1 if router $r \in R$ belongs to V_I^j and 0 otherwise. The optimization problem that we want to solve can be stated as follows:

$$\begin{aligned}
\min \quad & \sum_{j=1}^{n} f_j z_j \\
s.t. \quad & \sum_{j=1}^{n} x_{vj} z_j = 1, \forall v \in V_C \quad (1) \\
& \sum_{j=1}^{n} x_{rj} z_j = 1, \forall r \in R \quad (2) \\
& z_i z_j = 0, \forall (u,v) \in E_C, \forall i \neq j \; s.t. \; x_{ui} = x_{vj} = 1, \; l_I^i[d](u) + l_I^j[d](r_j) \geq l_C[t](u,v) \quad (3)
\end{aligned}$$

Constraint (1) forces a node to be covered by only one chain[4]. Constraint (2) forces a router to serve only one chain[5]. Constraint (3) is a set of delay constraints. If the source u and the destination v of a communication constraint belong to two different chains N_I^i and N_I^j, respectively, then the sum of the delay from u to router r_i plus the delay from router r_j to destination v must be less than the latency constraint from u to v. This problem is an instance of the Binate Covering Problem (BCP), which is *NP*-complete. Various algorithms for the exact or heuristic solution of BCP are known [122].

8.3.4 Results

We first show the results provided by our algorithm for the the example of Figure 8.2 and for two different ARCNET configurations: $2.5Mbps$ and $78Kbps$. We generate three different outputs to analyze the synthesis result: a textual report of the performances and cost of the network, a `dot` [2] file that shows the logical structure, and an `svg` [7] file that shows the physical structure of the network.

Figure 8.10 shows the logical structure of the LonWorks network on ARCNET @$2.5Mbps$. The solution has three daisy-chain busses. The daisy-chains are limited both by the maximum number of nodes (8) and by the maximum wire length ($120m$). Given the high speed of the bus, there is a large bandwidth and delay slack. Figure 8.11 shows the physical implementation of the network.

Table 8.2 shows the estimation of cost and performance for each sub-network. The cost is represented by a pair of values: the first value is the cost of the components (sensors, actuators, and routers) and the second value is the cost of the wires. Observe that the delay is considerably

[4]This constraint can be modified to force a node to be covered by more that one chain in the case of fault tolerance requirements.

[5]This constraint can be modified if a router has multiple ports and can actually serve more that one daisy chain bus.

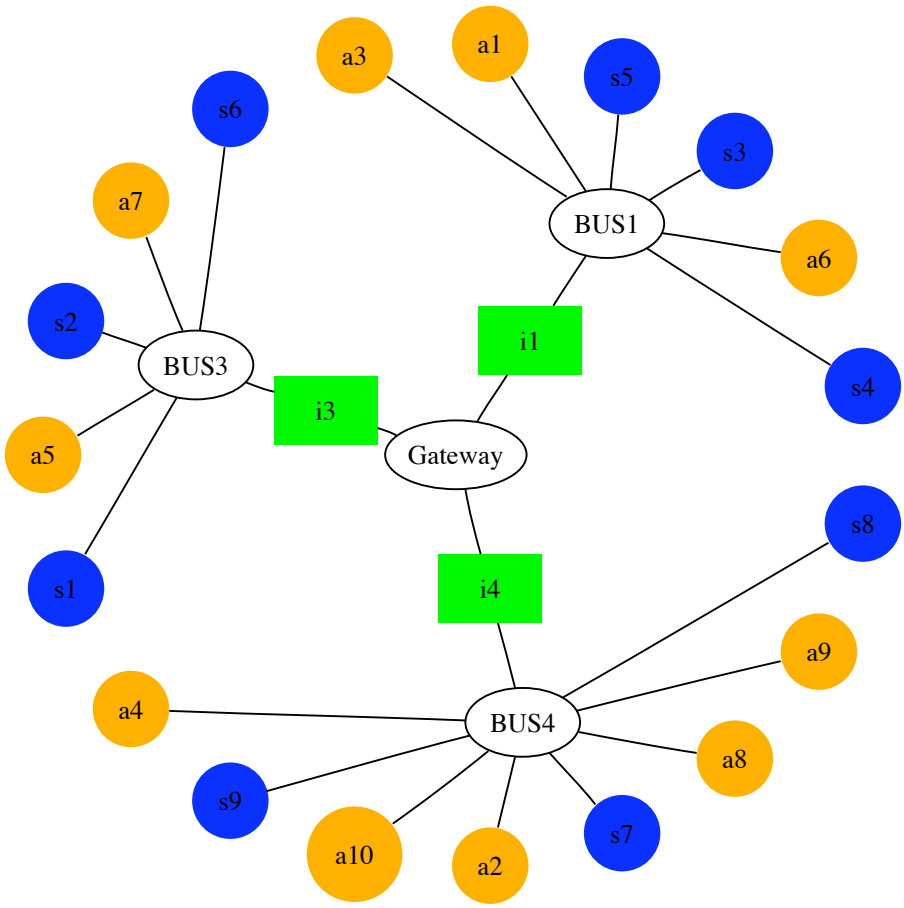


Figure 8.10: Logical components of the synthesized network for the example of Figure 8.2.

smaller than the required delay and that the bandwidth utilization is fairly low. This suggests that for this network we could consider a different implementation with lower speed and lower cost. For instance, with a slower signaling, wires can be longer and, moreover, the degree can be higher.

For instance, at $78Kbps$ ARCNET allows to connect up to 64 nodes on a bus segment that can be as long as $1200m$. Using this kind of protocol, we obtain a considerably cheaper solution ($5760 compared to $7160) in exchange for a longer delay. The delay is longer not only because the number of devices connected on the bus is higher but also because its signaling speed is much

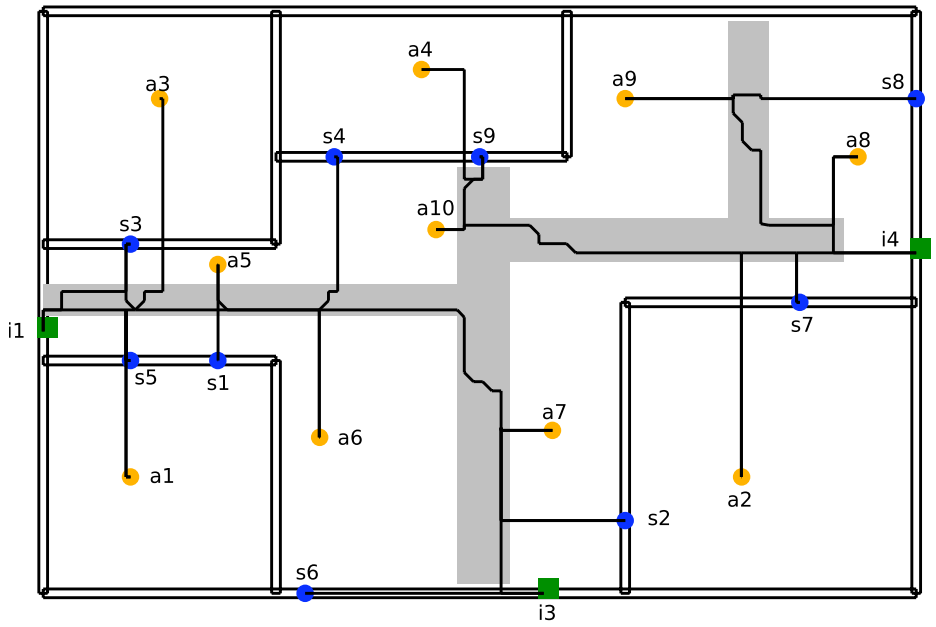


Figure 8.11: Physical deployment of the synthesized network implementation for the example of Figure 8.2.

lower. The bandwidth utilization is close to 50%.

On the other hand, while the cheaper solution is sufficient to support the application under design, once it is deployed it may prevent the future extension of the building automation system to support other applications. Since the deployment of a wired network in a building has considerable installation costs, this is another trade-off that must be considered carefully. In this regard, our tool can be useful to quickly analyze alternative solution hypothesis during the design-exploration phase.

We selected two other examples to test our design flow. The floor-plan of the first example, together with the communication constraints, are shown in Figure 8.12. It refers to one floor of the L2 building that is part of the United Technologies Research Center situated in Hartford, Connecticut. The second example is a typical case of big-box store, but the detailed floor-plan is a sensitive information that we cannot show in this thesis.

Router	**Deg**	**Length** $[m]$	**Delay** $[\mu s]$	**Bandwidth** $[Kbps]$	**Cost** $[dollars]$
			ARCNET @ $2.5Mbps$		
i_2	8	53	1367	17.3	$(2380, 404)$
i_3	3	20	607	6.5	$(1400, 152)$
i_4	8	58	1377	17.3	$(2380, 442)$
			ARCNET @ $78kbps$		
i_4	19	142	54000	41.1	$(4680, 1080)$

Table 8.2: Performance and cost of the synthesis result. The cost is a pair (n, w) where n is the cost of the nodes including the routers and w is the wiring cost.

Building	Bw (Kb/s)	l_{max} (m)	Degree	d_{max} (ms)	U_{max} $(\%)$	Router ($)	Wires ($)	Total ($)
L2	78	1000	32	91	89	3700	5020	18960
	250	400	20	22	20	5180	4939	20359
BB	78	1000	32	91	89	2220	4317	16777
	250	400	20	19	20	4440	4131	18811

Table 8.3: Library parameters and synthesis results for the two L2 building and big box store (BB) examples for a new installation. In this table, Bw is the bus speed, l_{max} is the maximum allowed bus length, d_{max} is the maximum latency experienced by any sensor, and U_{max} is the maximum bus utilization among all instantiated bus.

The floor-plans of these two examples occupy approximately the same area with a slightly different form factor ($72 \times 32\ m^2$ and $60 \times 56\ m^2$, respectively). The number of sensors is 64 in both examples and the communication constraints are the same as in the example of Figure 8.2. We considered six possible locations for the routers (represented by yellow dots in Figure 8.12).

Table 8.3 reports the synthesis result in the case of a new installation while Table 8.4 shows the result in the case of retrofitting. We report the maximum delay among all sensors d_{max} and the maximum bus utilization U_{max} among all installed daisy-chain buses together with the cost breakdown among routers and wires. It is interesting to highlight the difference in cost between

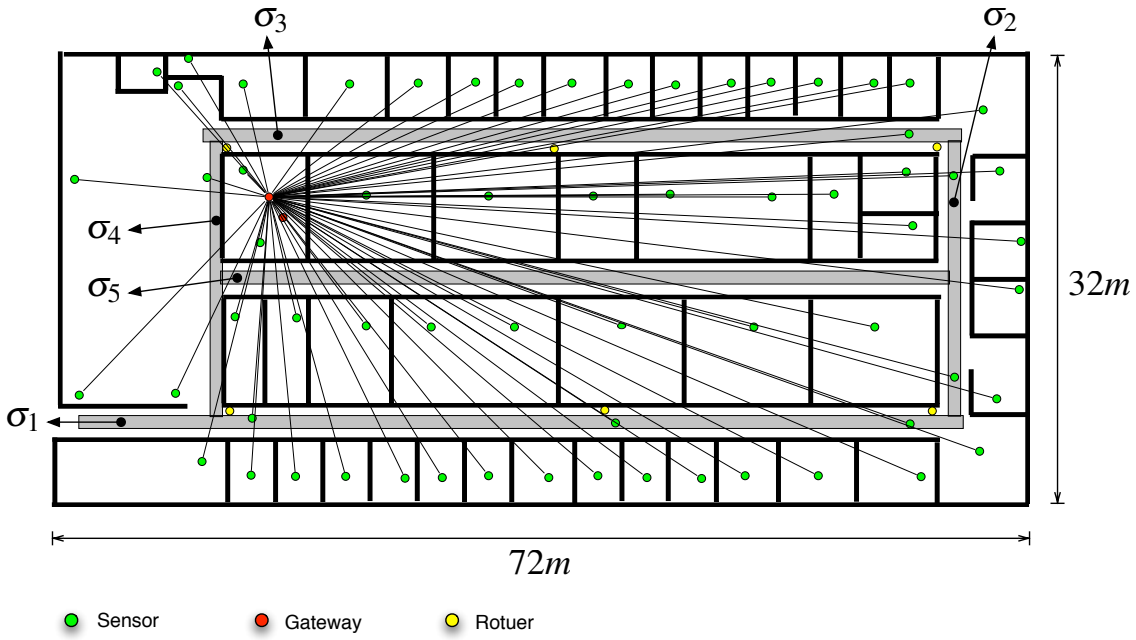


Figure 8.12: Communication specification and building floorplan of a UTRC premise in Hartford, CT.

Building	Bw (Kb/s)	l_{max} (m)	Degree	d_{max} (ms)	U_{max} (%)	Router ($)	Wires ($)	Total ($)
L2	78	1000	32	91	89	5920	12680	28844
	250	400	20	22	20	5180	13744	29198
BB	78	1000	32	91	89	3700	12044	25984
	250	400	20	19	20	4440	11855	26535

Table 8.4: Library parameters and synthesis results for the two L2 building and big box store (BB) examples for a retrofitting installation. In this table, Bw is the bus speed, l_{max} is the maximum allowed bus length, d_{max} is the maximum latency experienced by any sensor, and U_{max} is the maximum bus utilization among all instantiated bus.

the L2 building and the big-box store. Despite the similarities in the square footage and number of sensors, the network cost is different. Also, the difference in cost between a low speed network and a high speed network becomes negligible in the case of retrofitting since the cost of wiring dominates the cost of the routers.

8.4 Wireless Networks

With the introduction and improvement of wireless technology, new ways of implementing distributed applications in buildings have become feasible. The number of applications that can benefit from this are numerous, such as fire detection systems, temperature control, as well as more advanced applications such as distributed control and estimation of air flow in buildings. Wireless networks are particularly interesting because they eliminate the cost of wiring. This cost becomes the main concern in retrofitting old buildings, or in instrumenting part of a building that cannot be reached by wires.

Because wireless links do not rely on a mechanical connection, they are resilient to hazards like fire, and to unchecked operation. In fact, when building a wired network, multiple independent paths should be provided for each connection such that, even if one wire is cut or interrupted, the network can still provide the required quality of service. Wireless system are robust from this point of view. However, communication over wireless media is not as reliable especially in an indoor environment. Moreover, even if installing a wireless network is potentially cheaper than a wired one, the maintenance cost due battery replacement may be high. The lifetime of nodes in a wireless network is a critical parameter to judge its cost effectiveness and it is affected by transmission power and data rates.

The application domain that we target for wireless network is building automation systems. The specification of the communication problem is the same as in the case of wired networks with the addition of the end-to-end maximum packet error rate. The rest of this section will cover the modeling of library components and the formulation of the optimization problem. We will close this chapter with experimental results.

8.4.1 Library of Communication Components: Modeling ZigBee Networks

ZigBee [127] is currently considered the most appealing technology for wireless sensor networks. It offers a set of operational modes that are well-suited for a large set of low-rate and power-constrained network applications. The protocol stack of a ZigBee node is composed of the physical layer and Medium Access Control (MAC) Layer described in the IEEE802.15.4 standard [9], and a network layer and an application framework defined by the ZigBee Alliance [127].

A ZigBee network, called Personal Area Network (PAN), is organized in a hierarchical manner. Nodes are divided in three categories depending on the services that they offer to other nodes. Each ZigBee network has a unique *coordinator* called the PAN coordinator. This device is the one that started the network and that decides the network identifier, *PANid*, and the superframe parameters like the beacon order and the superframe order that are going to be defined later in this section. The second category is the *routers*. A router maintain a routing table and can relay messages. The last category is the *end-devices*. An end-device can only transmit messages to its parent. Coordinators and routers are the only devices that can allow other devices to join a network. When a node i joins a network thorough a node j, j is called the parent of i and i the child of j.

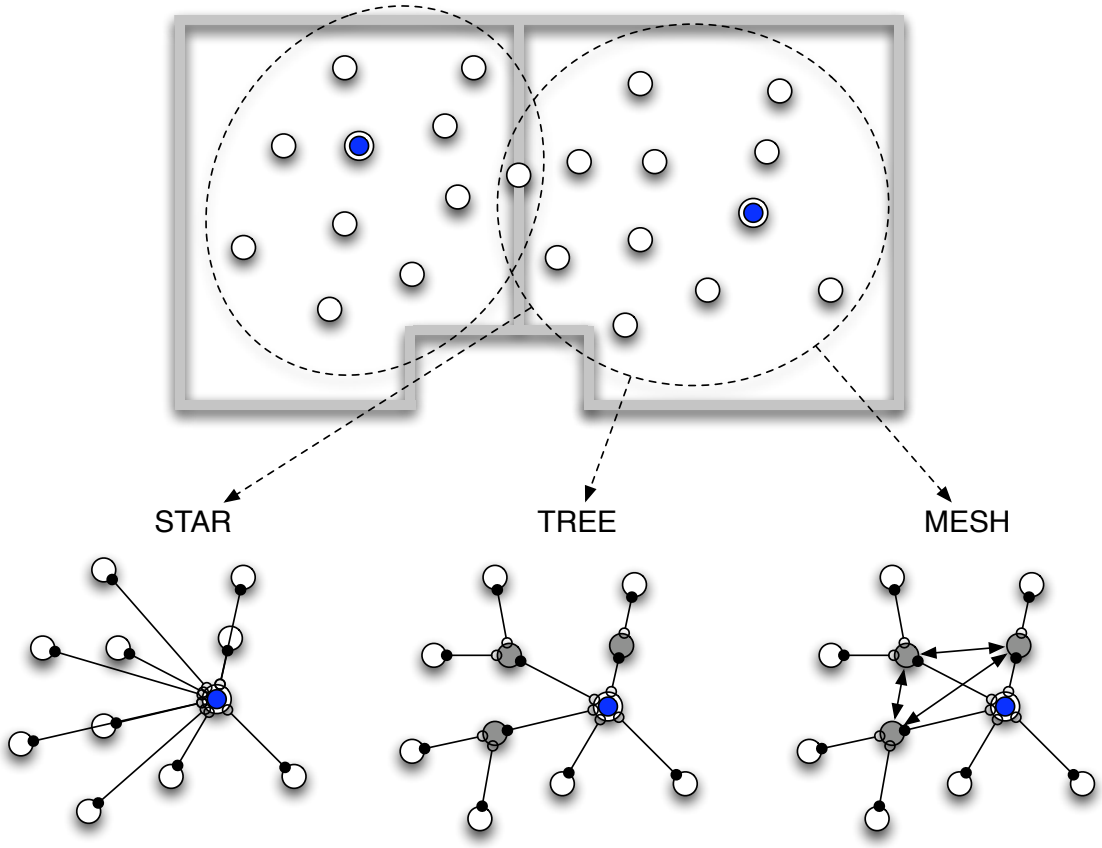


Figure 8.13: An area covered by two ZigBee networks (top) and the possible network topologies for one of them (bottom).

Topology and Routing Figure 8.13 shows a field covered by two ZigBee networks operating on two different logical channels. Each network has its own coordinator, a set of routers and a set of end-devices. In identifying the two networks, we purposely omitted the explicit representation of any link (i.e. the network topology). Rather, we only show the membership of the nodes to the networks. The actual network topology can be one of the following:

- A star topology, in which the Zigbee coordinator controls all the nodes in the network; devices directly communicate with the ZigBee coordinator; this is basically the case of a PAN with a

PAN coordinator and a set of devices directly associated with it;

- A mesh topology, in which peer-to-peer communications between devices are allowed; The ZigBee coordinator is still responsible for starting the network, but it can be extended by the use of ZigBee routers. The ZigBee routers are devices that act as coordinators in the IEEE802.15.4 PAN. Coordinators shall not send beacons in the mesh topology.

- A tree topology, in which ZigBee routers move data and control messages through the network using a hierarchical routing strategy. Beacon-enabled communications in the PAN are allowed.

The routing protocol defined by the ZigBee standard is tree routing for star and tree topologies. This is the topology that we assume for our modeling and experiments since performance assessment is easier.

Medium Access Control At the MAC layer, nodes are grouped in PANs (Personal Area Networks). A PAN is started by a node that assumes the role of PAN Coordinator, which establishes the values of a set of configuration parameters. These parameters have to be adopted by all the nodes that want to be associated with such a PAN. Basically, the coordinator fixes the physical channel, the Beacon Order (*BO*) and the Superframe Order (*SO*) of the *superframe structure*.

While simultaneous transmissions in different PANs can not collide, because they take place in different channels, intra-PAN transmissions need to be coordinated. The superframe structure (see Fig.8.14) is a flexible way to manage medium access control inside a PAN.

The PAN coordinator can periodically transmit a beacon frame (beacon-enabled mode). The time interval between two consecutive beacons is called Beacon Interval (*BI*) and it is defined as

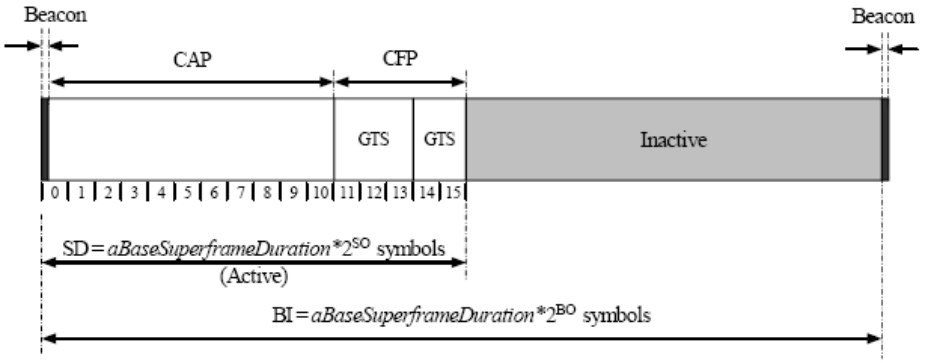


Figure 8.14: Structure of a superframe as defined by the ZigBee protocol standard.

$BI = aBaseSuperframeDuration \times 2^{BO}$ *symbols*. The *aBaseSuperframeDuration* has a constant duration of 960 symbols. BO can range from 0 to 14. BO = 15 means that no beacon has to be transmitted (nonbeacon-enabled mode). The BI is composed of an active part and an (optional) inactive part. The duration of the active part is determined by the Superframe Duration (SD), which is defined as $SD = aBaseSuperframeDuration \times 2^{SO}$ *symbols*. SO can range from 0 to BO (no inactive period).

The active period can be further divided in two parts:

- A Contention Access Period, in which transmissions are ruled by a slotted CSMA/CA algorithm and therefore collisions can occur. A mininum length (440 symbols) of the contention access period has to be guaranteed for the transmission of management frames.

- An (optional) Contention Free Period, which is composed of up to $in_{max} = 7$ Guaranteed Time Slots (GTSs). A GTS can be of transmit or receive type, meaning that the GTS can be used by the device for transmission (reception) of data to (from) the coordinator. Transmissions in the GTS do not need to be ruled by the CSMA/CA algorithm, since a GTS is uniquely allocated to a device, and concurrent transmissions by other devices in the same PAN are forbidden.

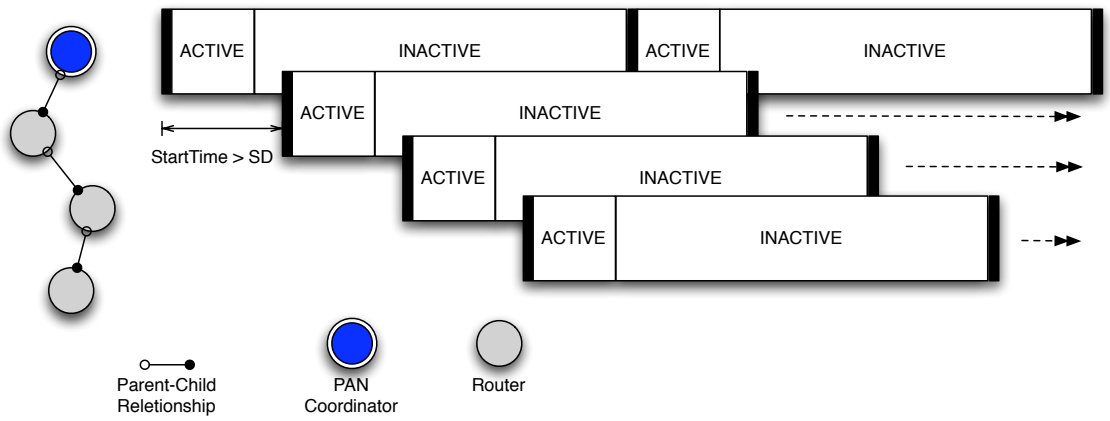


Figure 8.15: Relative timing of the superframes in a beacon-enabled ZigBee network.

In the inactive part, nodes can put the transceiver into a sleep state to save energy. Optionally, a device can assume the role of coordinator, which has to adopt the same *BO* and *SO* as the PAN coordinator. In a beacon-enabled PAN ($BO \neq 15$), such a device will start transmitting its own beacon. Its active part must not overlap with the active part of other coordinators in the network. Figure 8.15 shows how the active parts are scheduled within a superframe. Consider a network organized according in a tree topology that establishes a parent-child relationships among all the coordinators, with the PAN coordinator being the root of the tree. The beacon transmitted by a parent is the incoming beacon for the children. Each children emits the outgoing beacon after a certain time called *StartTime*. The start time is determined by a device when it joins a network. The maximum number of coordinators in a beacon-enabled PAN, including the PAN coordinator itself, can be no more than $n_{max} = BI/SD$.

When a device starts acting as a coordinator, the physical size of the network can increase; devices that are not able to associate with the PAN coordinator (e.g. they are too far from it) can join the network through a coordinator. From the perspective of contention-free communications,

an additional coordinator makes a new set of GTSs available; if a proper number of coordinators is available in the network, such that no more than 7 devices are associated with each one of them, communications among nodes can be completely performed in a contention-free manner.

Physical Layer At the physical layer, the IEEE802.15.4 standard offers a total of 27 channels, with a peak rate of 250Kbit/s. The distance at which packets are successfully received with some desired probability depends on the modulation format, encoding, output radio power, packet size, noise floor, interference and channel conditions. The relation among the physical layer parameters and the successful packet reception probability is shortly described below.

Let us consider a node i of the network transmitting packets with a radio power level P_i toward node j. The distance between node i and j is denoted with $d_{i,j}$. We denote with $\mathrm{PL}(d_{i,j})_{\,dB}$ the path loss attenuation between the transmitter and the receiver. For example, for the Telos Sky [81] wireless sensors, the following generic yet representative model of the path-loss can be used [118]:

$$\mathrm{PL}(d_{i,j})_{\,dB} = \mathrm{PL}(d_0)_{\,dB} + 10\,\beta\,\log_{10}\left(\frac{d_{i,j}}{d_0}\right) + \Omega_{i,j}$$

where $\mathrm{PL}(d_0)$ is the path loss computed at a reference distance d_0, β is the path loss exponent, and $\Omega_{i,j}$ is the shadowing attenuation, which is modelled as a Gaussian random variable having zero average and variance $\sigma^2_{i,j}$. The model of the path loss in the previous equation can also be used when there are walls between the transmitter and the receiver by adding an extra attenuation for each wall. The path loss is used to compute the received radio power $P_{i,j}(d)$ at the node j from the node i:

$$10\log_{10} P_{i,j} = 10\log_{10} P_i - \mathrm{PL}(d_{i,j})_{\,dB}$$

Given the received power, it is not difficult to compute the Signal to Interference plus Noise Ratio

(SINR) in dB for which we can use the following model:

$$10\log_{10}\mathrm{SINR}_{i,j} = 10\log_{10}P_{i,j} - P_{n\,dB} \tag{8.1}$$

In Equation (8.1), we assume that nodes are not simultaneously transmitting, so that the collision probability can be neglected and P_n summarizes the thermal noise and the power of the interference coming from co-channel radio systems (as, e.g., WIFI networks). Hence, we assume that the power spectral density of the noise is a constant term N_0. A typical value for the power of the thermal noise for the Telos Sky receivers is $N_0 = -170$dBm.

Given the SINR, the bit error probability of the link from node i to node j can be expressed as follows:

$$p_b(\mathrm{SINR}_{ij}) \triangleq f_1(\mathrm{SINR}_{ij}) \tag{8.2}$$

where $f_1(\cdot)$ is a function that accounts for the relation among the modulation format, the statistical distribution of the SINR, and the bit error rate. For example, let us consider a wireless propagation scenario where sensors do not move, so that the shadow fading can be considered constant over short period of times (less than 0.5s). Furthermore, we make the assumption that there are slowly moving obstacles causing slow fading. The Telos Sky nodes use O-QPSK (offset quadrature phase shift keying) modulation with DSSS. The bit error probability for O-QPSK with coherent demodulation in a slow Rayleigh fading environment, which exhibits non-selective behavior both in frequency and in time, can be expressed by [118]

$$f_1(\mathrm{SINR}_{i,j}) \approx \frac{1}{2}\left(1 - \sqrt{\frac{\mathrm{SINR}_{i,j}}{1+\mathrm{SINR}_{i,j}}}\right)$$

Using (8.2), it is possible to express the packet loss probability. Assume that a packet at the data-link layer is composed of an overhead of O bits and a payload of b_i bits, which incorporates

data that node of level i wish to transmit. Notice that the payload may have variable size according to the source coding technique employed, and the amount of data available. Under the assumption that the CRC code is always able to detect erroneous packets (see [65] for an experimental support), the packet loss probability, without any retransmission mechanism, can be expressed as:

$$p'(i,j) \triangleq f_2(\text{SINR}_{ij}) = 1 - [1 - p_b(\text{SINR}_{ij})]^{O+b_i} \tag{8.3}$$

Notice that in previous expression we did not include any forward error correction mechanism (FEC). However, (8.3) can be easily extended to include FEC. For example, convolutional or block codes may be used.

Cost of a Wireless Network The cost of a wireless network is the sum of the cost of the nodes. Each node has a retail price and an installation cost that may be dependent on its position. Moreover, it has a maintenance cost that is the cost of replacing the batteries over 20 years of operation. If E_{tx} and E_{rx} are the energy to transmit and receive one bit (respectively) on a wireless channel, and E is the battery capacity (expressed in Joule), then the total number of batteries to be replaced is $B = (b_{rx}E_{rx} + b_{tx}E_{tx})/E$ where b_{rx} and b_{tx} are the total number of bits received and transmitted over 20 years. The maintenance cost can be calculated as $B \cdot c_B$ where c_B is the sum of the retail price of the batteries plus the cost of replacing them.

The parameters that we use in our experiments are the following:

- the energy per bit used in transmission mode depends on the transmitting power. Therefore, in our synthesis flow this is an input parameter. We use the data available from the Xbow [125] data-sheet. For instance, for a transmission power of $-10dbm$, the transmission energy is approximately $132nJ$. The energy spend in receiving mode is $236nJ$.

- The total energy stored in a battery depends on the battery capacity. The unit of measure for the battery capacity is Ampere-Hour (Ah). We assume the availability of two $2Ah$, $1.5V$ batteries on a node, therefore the total energy stored in the batteries is $2 \cdot 2 \cdot 1.5 \cdot 3600 = 21600J$

- The price of a node is $ 40, the installation cost is $ 60 and the cost of changing the batteries is $ 20.

The energy used by a node depends on the time spent in an active mode (either transmit or receive). It is therefore convenient to have node in a sleep mode as long as possible. However, this is not always possible especially for those applications with fast dynamics. The experimental results in Section 8.4.3 will show the trade-offs involved in the design of wireless networks.

8.4.2 Formulation of the Optimization Problem

We formulate the synthesis problem for wireless network as an integer linear program (ILP). The inputs to the problem are the same as the ones described in Section 8.1. The communication constraints are captured by a communication structure $N_C(\mathscr{C}_C, \mathbf{q}_C, L_C)$ where $\mathbf{q}_C$ now also contains a variable PER representing the packet error rate. The set of possible routers that can be installed is denoted by R, and for each router $r \in R$ a position p_r is also given. Since R and V_C are finite, it is possible to define a node-edge incidence matrix A of the induced graph on $R \cup V_C$. An entry of the matrix $A(v,e)$, with $v \in R \cup V_C$ and $e \in (R \cup V_C)^2$ is equal to 1 (respectively -1) if a wireless link e can be installed with its source(destination) being v, and 0 otherwise. The matrix A is used to express the flow conservation constraints as described in Section 3.1. We assume that nodes in V_C are simple sensors and actuators (e.g. reduced function devices in ZigBee terminology) that don't provide routing services, and that the network is a tree where the leaves are the nodes in V_C.

The network relies on a synchronization service such that each router transmits/receives to/from its children at regular intervals called *beacon intervals*.

Let x_u be a binary variable that is equal to 1 if node u is installed and 0 otherwise. Let l_{uv} be a binary variable that is equal to 1 if a wireless link is instantiated between node u and node v and v is the parent of u. Moreover, for a constraint $e = (s,d) \in E_C$, let y_{uve} be a binary variable that is equal to 1 if the path from s to d uses the wireless link available between node u and node v, and $\mathbf{y}_e$ be the column vector of such variables[6] for all the edges. A path from s to d is an assignment to the components of $\mathbf{y}_e$ that satisfies $A_e\mathbf{y}_e = \mathbf{b}_e$, where $\mathbf{b}_e$ is a vector such that $\mathbf{b}_e(v) = 1$ for $v = s$, $\mathbf{b}_e(v) = -1$ for $v = d$ and $\mathbf{b}_e(v) = 0$ otherwise. The components of $\mathbf{y}_v$ that are equal to 1 are the wireless links belonging to the path, thus we simply refer to a solution of $A_e\mathbf{y}_e = \mathbf{b}_e$ as a *path*.

Given a path and given an additive quantity $w(u,v)$ defined on each link, we can compute the end-to-end value of the quantity with a linear expression $w(s,d) = \sum_{u,v} y_{uve} w(u,v)$. Because we assume synchronization, each wireless link is characterized by an upper bound on the delay that is equal to the beacon interval. Instead of using the packet error rate *PER* as a measure of the quality of a path, we consider the probability of success $PSR = 1 - PER$ that can be expressed as:

$$PSR(s,d) = \prod_{(u,v)\in(R\cup V_C)^2 : y_{uve}=1} PSR(u,v)$$

which is not a linear expression. Therefore we use the logarithm of *PSR* in the formulation of the problem:

$$\log PSR(s,d) = \sum_{(u,v)\in(R\cup V_C)^2} y_{uve} \log PSR(u,v)$$

[6] We assume a fixed ordering of the edges.

The following ILP models the synthesis problem:

$\min_{\mathbf{x},\mathbf{y}_q} F$

s.t.

1. $x_u + x_v - 2l_{uv} \geq 0$	$\forall u,v \in R \cup V_C$
2. $l_{uv} + l_{vu} \leq 1$	$\forall u,v \in R \cup V_C$
3. $l_{uv} = 0$	$\forall v \in V_C$
4. $\sum_{(u,v)\in(R\cup V_C)^2} l_{uv} - \sum_{z\in R\cup V_C} x_z = -1$	
5. $\sum_{u\in R\cup V_C} l_{uv} \leq in_{max}$	$\forall v \in R$
6. $l_{uv} + l_{vu} - y_{uve} \geq 0$	$\forall u,v \in R \cup V_C, \forall e \in E_C$
7. $l_{uv} + l_{vu} - y_{vue} \geq 0$	$\forall u,v \in R \cup V_C, \forall e \in E_C$
8. $A_e \mathbf{y}_e = \mathbf{b}_e$	$\forall e \in E_C$
9. $\sum_{e\in E_C} y_{uve} bandwidth(l_C[m](e), l_C[b](e)) \leq b_{max}$	$\forall u,v \in R \cup V_C$
10. $\sum_{r\in R} x_r \leq n_{max}$	
11. $\sum_{(u,v)\in(R\cup V_C)^2} y_{uve} delay(u,v) \leq l_C[t](e)$	$\forall e \in E_C$
12. $\sum_{(u,v)\in(R\cup V_C)^2} y_{uve} \log PSR(u,v) \leq \log(1 - l_C[PER](e))$	$\forall e \in E_C$
13. $x_u,\ l_{uv},\ y_{uve} \in \{0,1\}$	$\forall u,v \in R \cup V_C, \forall e \in E_C$

where the cost function (defined in details in Section 8.4.1) can be expressed as follows:

$$F = \sum_i c_i x_i + \sum_{ij} c_{ij} \sum_q y_{ijq}$$

The cost c_i includes the cost to install a node while c_{ij} contribute to the maintenance cost of nodes i and j (that obviously depends on the number of bits transmitted by i and received by j).

Constraints 1 through 5 define the tree topology of the network. Constraint 1 says that a link can only be installed between two installed nodes. Constraint 2 imposes a unidirectional parent-children relationship while constraint 3 says that sensors cannot be parent nodes. Constraint 4 forces the topology to be a tree by forcing the number of edges to be one less than then number of nodes. Constraints 5 limits the maximum number of children per node. Constraints 6 and 7 say that a path can be routed on an link only if that link is installed. Constraint 8 is the classical balance equation. Constraint 9 is a constraint on the maximum utilization of a link. We use a notation

similar to the one introduced in Section 8.3.3 where *bandwidth* is a model that takes into account the message length, message frequency and protocol overhead to estimate bandwidth. Constraint 10 limits the maximum number of routers that can be installed. Constraint 11 and 12 are the end-to-end constraint on the delay and packet error rate. Finally, constraint 13 requires all variables to be binary. Notice that this problem is still very general and can be directly written by interpreting the constraints given by the platform in terms of the available components, the building structure and the performance and cost models of each component. In fact, the ILP formulation represents a class of problems. One instance of the ILP corresponds to one particular building and one particular network configuration.

8.4.3 Results

As a case study, we consider the centralized and distributed estimation of a physical quantity in a building. Figure 8.16 shows the input to the network synthesis problem. The building that we model is one floor of a business premises that is comprised of 25 rooms for a total dimension of $32 \times 70\ m^2$. Green dots represent sensors, the red dot corresponds to a central gateway and yellow dots are the positions where routers can be installed. A connector between two dots corresponds to a communication constraint with associated latency, period, number of bits and packet error rate requirements. Figure 8.16-b shows the communication constraints in the case of centralized estimation where the samples coming from all the sensors must be delivered to the central gateway. Figure 8.16-a shows the communication constraints in the case of distributed estimation. The number of sensors is less than in the centralized case, but there is mutual interaction between sensors corresponding to neighboring states. The mutual interaction is considered bidirectional and takes place at a higher rate than communication with the central gateway.

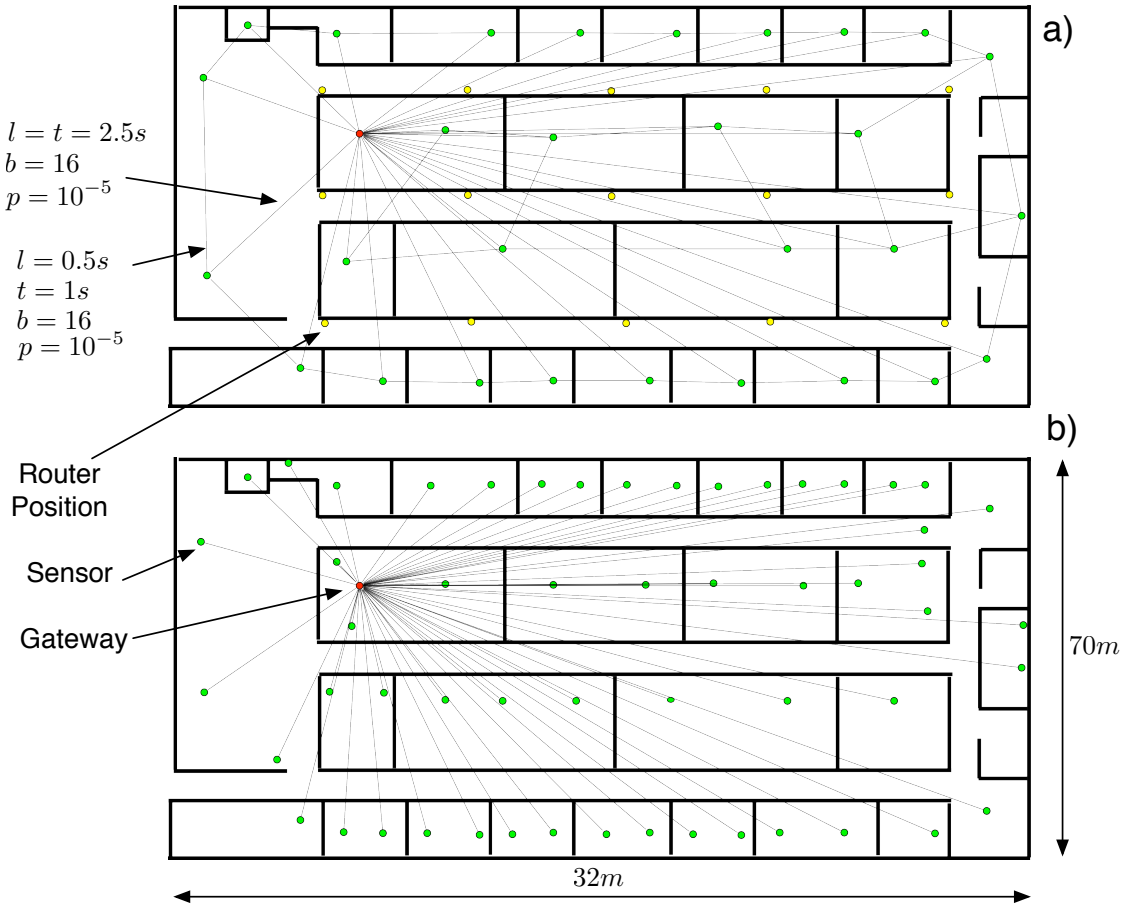


Figure 8.16: The two test cases used in our experiments: a) Distributed estimation, b) Centralized estimation.

BO	SO	Cost (\$ in 20 yr)	T_{cpu} (s)	Gap
7	3	2376	1400	5%
8	3	15150	590	4.9%
8	4	22900	1400	6%

Table 8.5: Results for the centralized estimation case.

Table 8.5 shows the results of the optimization for the centralized case. We report the beacon order BO, the superframe order SO, the total network cost over 20 years, the computation time T_{cpu} in seconds and the optimality gap. The problem is unfeasible for $SO < 3$ due to the bandwidth requirements and for $BO > 8$ due to the latency requirements. The minimum difference between BO and SO must be 4 because the minimum number of routers needed for this network is greater than 8. This is due not only to the number of sensors, but also to the packet error rate constraint and the building structure that limit the maximum length of a hop intersecting multiple walls. We also mention that the wired network implementation for this application has a total cost of $\$18,000$.

In the distributed estimation case, the number of sensors to connect is reduced by half. However, the stringent latency requirements between neighboring nodes limit the maximum beacon order to 5. The reduced bandwidth requirement at the gateway, allows a superframe order of 1 making the duty-cycle equal to 6.25%. This instance of the problem has been solved to the global optimum for a total cost of $\$23,400$.

Chapter 9

Conclusions and Future Work

The automatic synthesis of communication networks for distributed embedded systems is an essential part of a system-level design flow. The ever increasing complexity of electronic systems, together with time-to-market constraints, is causing a major design paradigm shift from components to their interaction. In fact, to increase productivity, systems are designed by assembling pre-designed and pre-verified components using a suitable interaction scheme. To support designers in the selection of the communication infrastructure and to guarantee the quality of the solution without incurring in long verification cycles, a correct-by-construction design flow for communication systems is an essential enabling technology.

In this thesis we proposed a platform-based design methodology to design cost effective communication systems. We formally defined the specification of the problem as a set of end-to-end communication constraints among the agents of a system. These constraints include the quality of service requested for each communication, interface compatibility, as well geometric constraints like distance and obstacles from each source-destination pair. These are the properties that the com-

munication system must preserve. The set of valid implementations is captured by the notion of a platform that is the set of communication systems that can be obtained by composing communication components from a library (i.e. platform instances). Each component is characterized by performance and cost. We formally capture these metrics as quantities and we provide composition operators that derive the performance and cost of the composite from the same metrics defined on the components. The final implementation is defined as the mapping of the specification onto a platform instance and it is computed by synthesis algorithms.

Many optimization algorithms are available in the literature, mainly developed over the years by the Computer Science and Operation Research communities that can find a direct application to the communication synthesis problems for networked embedded systems. However, taking into account real life end-to-end constraints like delay and reliability, as well as component constraints, such as the limited number of ports, makes the problem difficult to abstract and hard to solve. Moreover, the possibility of exploring different solutions by simply selecting different platforms is of great importance to allow designers to explore large design spaces and choose the implementation technology that best fits their needs. These desirable features can be only achieved by disentangling specification, libraries of components and platforms, models of performance and cost, and optimization algorithms. Implementing a software infrastructure that provides all these features requires a major engineering effort.

COSI, the software infrastructure that is an integral part of this thesis, was built keeping the designer's needs in mind. Obviously, the input functions to parse the specification of the problem (which also include the description of the environment the network is embedded in) and the output functions to present the results, as well as the quantities that characterize a communication structure

change depending on the application domain. COSI provides the basic data structures that can be adapted to implement design flows for many application domains. We have used COSI to solve the communication synthesis problem in two application domains: on-chip communications and building automation systems, and for different network implementation technologies.

This is the beginning of a long term project thato touches the research needs of many different areas. What follows is a list of research directions aimed at providing a complete solution for the communication synthesis problems that arise in networked embedded systems.

Theoretical investigation. COSI provides a prototype of an environment for the automatic design of embedded networks. To improve its generality and usability the following theoretical aspects should be investigated:

Extension of the algorithms. The optimization algorithms should be extended to take into account fault tolerance. This can be achieved by adding redundancy constraints and component reliability: the general theoretical framework should be already capable of capturing both. The second extension deals with on-line optimization algorithms. In many situations, the environment can undergo abrupt changes that are difficult to predict. In these cases, a distributed network control algorithm should be able to dynamically reconfigure the network such that some level of quality of service can still be guaranteed. This task requires to study distributed versions of the optimization algorithms. The balance between deployment cost and on-line capabilities should be investigated.

Assessment of the sensitivity of the solution. The solution of the synthesis problem depends on the cost and performance of the components of in the library. These models are analytical abstractions of the real metrics that allow the synthesis problem to be solved efficiently. It is desirable that the solution does not depends too much on modeling errors (or uncertainty on the cost functions). The

sensitivity of the solution should be assessed by analyzing the overall design flow. From the user perspective, a design environment should provide a set of solutions rather than the prescription of one implementation.

Interface refinement and synthesis. Since networked systems are heterogeneous, the interfaces between different components are often incompatible. In general, it is be desirable to have an automatic way of synthesizing interfaces between incompatible protocols.

Experimental setting. Even if we advocate the correct-by-construction deployment of networked embedded systems, to limit the complexity of the design problem we use models that can only approximate reality. The validation of the results produced by a design flow consists in verifying that the embedded network design satisfies the specification. This goal can be achieved by simulation or by implementation on a real testbed.

Virtual Prototyping. As networked systems become complex, building a real testbed for academic purposes is not a viable solution. Thus, a virtual prototyping environment is very desirable. This environment must have two important features. It must be able to simulate heterogeneous networks with different components, protocols and transmission media. At the same time, it must be able to incorporate models for physical phenomena like wireless propagation, air-flow propagation and human interaction. We plan to reuse existing tools for heterogeneous design, but a major effort is needed to develop accurate simulation models.

Real testbeds. The actual implementation of a networked embedded system will require the collaboration with industrial partners.

Automatic distribution of control algorithms on networked architectures. The cost of the communication infrastructure supporting a networked control application depends on the requested level

of quality of service. Therefore, it is clear that the selection of the control algorithm and its distributed implementation affect the cost/performance trade-off of the networked embedded system. The quality of a centralized control algorithm as a function of the communication characteristics should be quantified. The control strategy should be selected in such a way that the network requirements are minimized. Then, an automatic procedure should be developed to derive a distributed implementation of the centralized control. The distribution should be driven by the trade-off between the network cost and the computation cost.

Stochastic optimization for networked embedded system design. Stochastic optimization is a promising tool in automated system design to deal with uncertainties of all kinds. There are at least three reasons to use stochastic optimization methods. First, one way of making the design robust is to abstract modeling errors as random variables. Second, many environmental effects can only be characterized as stochastic processes. Third, network components themselves are unreliable and their behavior can only be statistically characterized.

Bibliography

[1] Ata. arcnet (http://www.arcnet.com/lit.htm).

[2] Graphviz (http://www.graphviz.org/).

[3] http://www.autosar.org.

[4] http://www.itrs.net/.

[5] Private communication with prof. alberto l. sangiovanni vincentelli.

[6] SLICNET.

[7] Svg (http://www.w3.org/graphics/svg/).

[8] *ISO/IEC 7498-1, Information Technology – Open Systems Interconnection – Basic Reference Model: The Basic Model*. 1994.

[9] Part 15.4: Wireless medium access control (mac) and physical layer (phy) specifications for low-rate wireless personal area networks (wpans). IEEE Std 802.15.4-2996, September 2006.

[10] Trevor Meyerowitz-Alessandro Pinto Alberto Sangiovanni-Vincentelli Guang Yang Haibo Zeng Qi Zhu Abhijit Davare, Douglas Densmore. A next-generation design framework for

platform-based design. In *Conference on Using Hardware Design and Verification Languages (DVCon)*, February 2007.

[11] S. N. Adya and I. L. Markov. Fixed-outline floorplanning : Enabling hierarchical design. *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, 11(6):1120–1135, December 2003.

[12] P. Alexander, D. Barton, and C. Kong. Rosetta usage guide. Technical report, University of Kansas, 2000.

[13] F. Allen, G. Almasi, W. Andreoni, D. Beece, B. J. Berne, A. Bright, J. Brunheroto, C. Cascaval, J. Castanos, P. Coteus, P. Crumley, A. Curioni, M. Denneau, W. Donath, M. Eleftheriou, B. Fitch, B. Fleischer, C. J. Georgiou, R. Germain, M. Giampapa, D. Gresh, M. Gupta, R. Haring, H. Ho, P. Hochschild, S. Hummel, T. Jonas, D. Lieber, G. Martyna, K. Maturu, J. Moreira, D. Newns, M. Newton, R. Philhower, T. Picunko, J. Pitera, M. Pitman, R. Rand, A. Royyuru, V. Salapura, A. Sanomiya, R. Shah, Y. Sham, S. Singh, M. Snir, F. Suits, R. Swetz, W. C. Swope, N. Vishnumurthy, T. J. C. Ward, H. Warren, and R. Zhou. Blue gene: a vision for protein science using a petaflop supercomputer. *IBM Syst. J.*, 40(2):310–327, 2001.

[14] H. B. Bakoglu. *Circuits, Interconnections, and Packaging for VLSI*. Addison-Wesley, Reading,MA, 1990.

[15] Felice Balarin, Massimiliano Chiodo, Paolo Giusto, Harry Hsieh, Attila Jurecska, Luciano Lavagno, Claudio Passerone, Alberto L. Sangiovanni-Vincentelli, Ellen Sentovich, Kei

Suzuki, and Bassam Tabbara. *Hardware Software Co-Design of Embedded Systems: The Polis Approach*. Kluwer Academic Publishers, 1997.

[16] Felice Balarin, Luciano Lavagno, Claudio Passerone, Alberto Sangiovanni-Vincentelli, Yosinori Watanabe, and Guang Yang. Concurrent execution semantics and sequential simulation algorithms for the metropolis meta-model. In *CODES '02: Proceedings of the tenth international symposium on Hardware/software codesign*, pages 13–18, New York, NY, USA, 2002. ACM.

[17] Felice Balarin, Yosinori Watanabe, Harry Hsieh, Luciano Lavagno, Claudio Passerone, and Alberto Sangiovanni-Vincentelli. Metropolis: An integrated electronic system design environment. *Computer*, 36(4):45–52, 2003.

[18] G. H. Barnes, R. M. Brown, M. Kato, D. Kuck, D. Slotnick, and R. Stokes. The ILLIAC IV computer. *IEEE Trans. on Computers*, 8(17):746–757, August 1968.

[19] L. Benini and G. De Micheli. Networks on chip: A new SoC paradigm. *IEEE Computer*, 2002.

[20] D. Bertozzi, A. Jalabert, S. Murali, R. Tamhankar, S. Stergiou, L. Benini, and G. De Micheli. NoC synthesis flow for customized domain specific multiprocessor systems-on-chip. *IEEE Trans. on Parallel and Distributed Systems*, 16(2):113–129, February 2005.

[21] Evgeny Bolotin, Israel Cidon, Ran Ginosar, and Avinoam Kolodny. Qnoc: Qos architecture and design process for network on chip. *J. Syst. Archit.*, 50(2-3):105–128, 2004.

[22] S. Borkar, P. Dubey, K. Kahn, D. Kuck, H. Mulder, S. Pawlowski, and J. Rattner. Platform 2015: Intel processor and platform evolution for the next decade, 2005.

[23] Dominique Borrione, Amr Helmy, Laurence V. Pierre, and Julien Schmaltz. A generic model for formally verifying noc communication architectures: A case study. In *NOCS*, pages 127–136, 2007.

[24] Steven T. Bushby. Bacnet^{TM} - a standard communication infrastructure for intelligent buildings. *Automation in Construction*, 6(5–6):529–540, 1997.

[25] S. Chaki, S.K. Rajamani, and J. Rehof. Types as models: Model checking message-passing programs. In *Proc. 29th ACM Symp. Princ. of Prog. Lang.*, 2002.

[26] Arindam Chakrabarti, Luca de Alfaro, Thomas A. Henzinger, Marcin Jurdzinski, and Freddy Y. C. Mang. Interface compatibility checking for software modules. In *Proceedings of the 14th International Conference on Computer-Aided Verification (CAV)*, volume 2404 of *Lecture Notes in Computer Science*, pages 428–441. Springer-Verlag, 2002.

[27] Arindam Chakrabarti, Luca de Alfaro, Thomas A. Henzinger, and Marielle Stoelinga. Resource interfaces. In *Proceedings of the Third International Conference on Embedded Software (EMSOFT)*, volume 2855 of *Lecture Notes in Computer Science*. Springer-Verlag, 2003.

[28] Henry Chang, Larry Cooke, Merrill Hunt, Grant Martin, Andrew J. McNelly, and Lee Todd. *Surviving the SOC Revolution. A Guide to Platform-Based Design*. Kluwer Academic Publishers, Norwell, MA, 1999.

[29] Yonghao Chen and Betty H. C. Cheng. Facilitating an automated approach to architecture-based software reuse. In *Automated Software Engineering*, pages 238–245, 1997.

[30] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, second edition, 2001.

[31] The Communication Synthesis Infrastructure (COSI). http://embedded.eecs.berkeley.edu/cosi/.

[32] CPLEX. http://www.ilog.com/products/cplex/.

[33] W. J. Dally and C. L. Seitz. Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Trans. Comput.*, 36(5):547–553, 1987.

[34] W. J. Dally and B. Towles. Route packets, not wires: On-chip interconnection networks. In *Proc. of the Design Automation Conf.*, June 2001.

[35] William J. Dally and B. Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers, San Mateo, CA, 2004.

[36] Luca de Alfaro and Thomas A. Henzinger. Interface theories for component-based design. In *Proceedings of the First International Workshop on Embedded Software*, pages pp. 148–165. Lecture Notes in Computer Science 2211, Springer-Verlag, 2001.

[37] M. Desrochers and F. Soumis. A generalized permanent labelling algorithm for the shortest path problem with time windows. *Information Systems and Operations Research*, 26(3):191–212, 1988.

[38] Frederic Doucet, Sandeep K. Shukla, Masato Otsuka, and Rajesh K. Gupta. Balboa: a component-based design environment for system models. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 22(12):1597–1612, 2003.

[39] Alberto Sangiovanni-Vincentelli Douglas Densmore, Roberto Passerone. A platform-based

taxonomy for esl design. *IEEE Design and Test of Computers*, 23(5):359– 374, September 2006.

[40] I. Dumitrescu and N. Boland. Algorithms for the weight constrained shortest path problem. *International Transactions in Operational Research*, 8(1):15–29, January 2001.

[41] Echelon. Lonworks core technology (http://www.echelon.com/developers/lonworks/default.htm).

[42] Jack Edmonds and Richard M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM*, 19(2):248 – 264, April 1972.

[43] EDN. http://jima4media.wiki.zoho.com/iphone-contents.html.

[44] Rolf Ernst, Jorg Henkel, and Thomas Benner. Hardware-software cosynthesis for microcontrollers. *IEEE Des. Test*, 10(4):64–75, 1993.

[45] Alberto Ferrari and Alberto L. Sangiovanni-Vincentelli. System design: Traditional concepts and new paradigms. In *Proceedings of the International Conference on Computer Design*, pages 1–12, October 1999.

[46] Gene F. Franklin, J. David Powell, and Abbas Emami-Naeini. *Feedback Control of Dynamic Systems*. Prentice Hall, 4th edition, 2002.

[47] D Gajski and R Kuhn. New vlsi tools. *Computer*, 16(12):11 – 14, Dec 1983.

[48] D. Gajski, F. Vahid, S. Narayan, and J. Gong. Specsyn: An environment supporting the specify-explorerefine paradigm for hardware/software system design, 1998.

[49] M. R. Garey and D. S. Johnson. *Computers and Intractability : A Guide to the Theory of*

NP-Completeness (Series of Books in the Mathematical Sciences). W. H. Freeman, January 1979.

[50] M. Gasteier, M. Münch, and M. Glesner. Generation of interconnect topologies for communication synthesis. In *DATE '98: Proceedings of the conference on Design, automation and test in Europe*, pages 36–43, Washington, DC, USA, 1998. IEEE Computer Society.

[51] Andreas Gerstlauer, Dongwan Shin, Junyu Peng, Rainer Dmer, and Daniel Gajski. Automatic layer-based generation of system-on-chip bus communication models. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 26(9):1676–1687, 2007.

[52] Greg Gibeling, Andrew Schultz, and Krste Asanovic. The ramp architecture & description language. Technical report, U.C. Berkeley, February 2006.

[53] A V Goldberg and R E Tarjan. A new approach to the maximum flow problem. In *STOC '86: Proceedings of the eighteenth annual ACM symposium on Theory of computing*, pages 136–146, 1986.

[54] R. K. Gupta and G. De Michelli. Hardware-software cosynthesis for digital systems. *IEEE Design and Test of Computers*, 1993.

[55] G. Handler and I. Zang. A dual algorithm for the constrained shortest path problem. *Networks*, 1980.

[56] David Harel. Statecharts: A visual formalism for complex systems. *Sci. Comput. Program.*, 8(3):231–274, 1987.

[57] A. Hemani, A. Jantsch, S. Kumar, A. Postula, J. Vberg, M. Millberg, and D. Lindqvist.

Network on chip: An architecture for billion transistor era. In *Proc. of the IEEE NorChip Conference*, November 2000.

[58] S. Heo and K. Asanovic. Replacing global wires with an on-chip network: a power analysis. In *Proc. of the Intl. Symp. on Low Power Electronics and Design*, pages 369–374, 2005.

[59] R. Ho, K. W. Mai, and M. A. Horowitz. The future of wires. *Proceedings of the IEEE*, pages 490–504, April 2001.

[60] C. A. R. Hoare. Communicating sequential processes. *Commun. ACM*, 21(8):666–677, 1978.

[61] R. Michael Hord. *Parallel Supercomputing in Mimd Architectures*. CRC Press, 1993.

[62] J. Hu and R. Marculescu. Energy- and performance-aware mapping for regular NoC architectures. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 24(4):551–562, November 2005.

[63] F K. Hwang, D S. Richards, and P Winter. The steiner tree problem. Jan 1992.

[64] Axel Jantsch. *Modeling Embedded Systems and SoC's: Concurrency and Time in Models of Computation*. Morgan Kaufmann, 2004.

[65] Jaein Jeong and Cheng Tien Ee. Forward error correction in sensor networks. Technical report, EECS Department, University of California, Berkeley, 2003.

[66] H.C. Jokcsh. The shortest route problem with constraints. *Journal of Mathematical Analysis and Applications*, 14:191–197, 1966.

[67] Wolfgang Kastner, Georg Neugschwandtner, Stefan Soucek, and H. Michael Newman.

Communication systems for building automation and control. *Proceedings of the IEEE*, 93(6):1178–1203, June 2005.

[68] Matt Kaufmann, J. Strother Moore, and Panagiotis Manolios. *Computer-Aided Reasoning: An Approach*. Kluwer Academic Publishers, Norwell, MA, USA, 2000.

[69] Aaron Kershenbaum. *Telecommunications network design algorithms*. McGraw-Hill, Inc., New York, NY, USA, 1993.

[70] B Kienhuis, E Deprettere, K Vissers, and P Van Der Wolf. An approach for quantitative analysis of application-specific dataflow architectures. *Application-Specific Systems, Architectures and Processors, 1997. Proceedings., IEEE International Conference on*, pages 338 – 349, Jun 1997.

[71] J. G. Klincewicz. Hub location in backbone/tributary network design: a review. *Location Science*, 6(1):307–335, May 1998.

[72] K. Lahiri, A. Raghunathan, and S. Dey. Design space exploration for optimizing on-chip communication architectures. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 23(6):952–961, December 2004.

[73] Luciano Lavagno, Alberto Sangiovanni-Vincentelli, and Ellen Sentovich. Models of computation for embedded system design. pages 45–102, 1999.

[74] Yanbing Li and Wayne Wolf. Hardware/software co-synthesis with memory hierarchies. In *ICCAD '98: Proceedings of the 1998 IEEE/ACM international conference on Computer-aided design*, pages 430–436, New York, NY, USA, 1998. ACM Press.

[75] Madhav V. Marathe, R. Ravi, Ravi Sundaram, S. S. Ravi, Daniel J. Rosenkrantz, and III Harry B. Hunt. Bicriteria network design problems. *J. Algorithms*, 28(1):142–171, 1998.

[76] Deepak A. Mathaikutty and S. K. Shukla. MCF: A metamodeling based component composition framework – composing SystemC IPs for executable system models. *To Appear*, 2008.

[77] J. D. Meindl. Interconnect opportunites for gigascale integration. *IEEE Micro*, 2003.

[78] Michael Minoux. Network synthesis and optimum network design problems: Models, solution methods and applications. *Networks*, (19):313–360, 1989.

[79] Gordon E. Moore. Network synthesis and optimum network design problems: Models, solution methods and applications. *Electronics*, 38(8), April 1965.

[80] Brandon Morel. Spartacas automating component reuse and adaptation. *IEEE Trans. Softw. Eng.*, 30(9):587–600, 2004. Senior Member-Perry Alexander.

[81] Moteiv. Tmote sky data sheet (http://www.moteiv.com/products/docs/tmote-sky-datasheet.pdf).

[82] S.Kumar V.Catania M.Palesi, R.Holsmark. Apsra: A methodology for design of application specific routing algorithms for noc systems. Technical Report DIIT-TR-01-060406, Universit di Catania, Italy, April 2006.

[83] S. Murali and G. De Micheli. SUNMAP: A tool for automatic topology selection and generation for NOCs. In *Proc. of the Design Automation Conf.*, pages 914–919, June 2004.

[84] Srinivasan Murali, Paolo Meloni, Federico Angiolini, David Atienza, Salvatore Carta, Luca Benini, Giovanni De Micheli, and Luigi Raffo. Designing application-specific networks on chips with floorplan information. In *Proc. Intl. Conf. on Computer-Aided Design*, pages 355–362, November 2006.

[85] Ana Muriel and Farhad N. Munshi. Capacitated multicommodity network flow problems with piecewise linear concave costs. *IIE Transactions*, 36:683–696, 2004.

[86] John Von Neumann. First draft of a report on the edvac. *Res. Div. Rep. 50–9, Contract W-36-O34-ORD-7593 with Ordnance Dept., Dept. of the Army*, November 1945.

[87] H. Michael Newman. *Direct Digital Control of Building Systems: Theory and Practice*. Wiley, 1994.

[88] L. M. Ni and P. K. McKinley. A survey of wormhole routing techniques in direct networks. pages 492–506, 2000.

[89] OCP-IP.

[90] Katsuhiko Ogata. *Modern Control Engineering*. Prentice Hall, 4th edition, 2001.

[91] U. Ogras and R. Marculescu. Application-specific network-on-chip architecture customization via long-range link insertion. In *Proc. Intl. Conf. on Computer-Aided Design*, November 2005.

[92] Ross B. Ortega and Gaetano Borriello. Communication synthesis for distributed embedded systems. In *ICCAD '98: Proceedings of the 1998 IEEE/ACM international conference on Computer-aided design*, pages 437–444, New York, NY, USA, 1998. ACM Press.

[93] A. Ouorou, P. Mahey, and J.-Ph. Vial. A survey of algorithms for convex multicommodity flow problems. *Manage. Sci.*, 47(1):126–147, 2000.

[94] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Dover Publications, Inc., 1998.

[95] Roberto Passerone. *Semantic foundations for heterogeneous systems*. PhD thesis, University of California at Berkeley, 2004.

[96] Roberto Passerone, Luca de Alfaro, Thomas A. Henzinger, and Alberto L. Sangiovanni-Vincentelli. Convertibility verification and converter synthesis: Two faces of the same coin. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD'02)*, November 2002.

[97] Roberto Passerone, James A. Rowson, and Alberto L. Sangiovanni-Vincentelli. Automatic synthesis of interfaces between incompatible protocols. In *DAC*, San Francisco, CA, June 1998.

[98] M. Pedram, N. Bhat, and E. Kuh. Combining technology mapping with layout, 1997.

[99] S.; Bolliger M.; Day M.N.; Hofstee H.P.; Johns C.; Kahle J.; Kameyama A.; Keaty J.; Masubuchi Y.; Riley M.; Shippy D.; Stasiak D.; Suzuoki M.; Wang M.; Warnock J.; Weitzel S.; Wendel D.; Yamazaki T.; Yazawa K. Pham, D.; Asano. The design and implementation of a first-generation cell processor. *Solid-State Circuits Conference, 2005. Digest of Technical Papers. ISSCC. 2005 IEEE International*, pages 184–592 Vol. 1, 6-10 Feb. 2005.

[100] A. Pinto. Metropolis design guidelines. Technical Report UCB/ERL M04/40, EECS Department, University of California, Berkeley, 2004.

[101] Alessandro Pinto, Alvise Bonivento, Alberto L. Sangiovanni-Vincentelli, Roberto Passerone, and Marco Sgroi. System level design paradigms: Platform-based design and communication synthesis. *ACM Trans. Des. Autom. Electron. Syst.*, 11(3):537–563, 2006.

[102] Shiv Prakash and Alice C. Parker. Synthesis of application-specific heterogeneous multiprocessor systems (abstract). In *ISCA '92: Proceedings of the 19th annual international symposium on Computer architecture*, page 434, New York, NY, USA, 1992. ACM Press.

[103] National Building Controls Information Program. Building energy use and control problems: Defining the connection, May 2002.

[104] Antonio Pullini, Federico Angiolini, Paolo Meloni, David Atienza, Srinivasan Murali, Luigi Raffo, Giovanni De Micheli, and Luca Benini. 65 nm NoC design: Opportunities and challenges. *Proc. of the 1st Intl. Symp. on Networks-on-Chips*, 2007.

[105] R. Ravi, Madhav V. Marathe, S. S. Ravi, Daniel J. Rosenkrantz, and Harry B. Hunt III. Approximation algorithms for degree-constrained minimum-cost network-design problems. *Algorithmica*, 31(1):58–78, 2001.

[106] David L. Rhodes and Wayne Wolf. Co-synthesis of heterogeneous multiprocessor systems using arbitrated communication. In *ICCAD '99: Proceedings of the 1999 IEEE/ACM international conference on Computer-aided design*, pages 339–342, Piscataway, NJ, USA, 1999. IEEE Press.

[107] Fabio Romeo. Embedded Systems: The Real Story, Magneti Marelli Electronic Division. *Invited Talk, Design Automation Conference*, 2001.

[108] James A. Rowson and Alberto Sangiovanni-Vincentelli. Interface-based design. *dac*, 00:178, 1997.

[109] A. Salek, J. Lou, and M. Pedram. An integrated logical and physical design flow for deep submicron circuits, 1999.

[110] Alberto Sangiovanni-Vincentelli. Defining platform-based design. *EEDesign of EETimes*, February 2002.

[111] Alberto L. Sangiovanni-Vincentelli. Quo vadis sld: Reasoning about trends and challenges of system-level design. *Proceedings of the IEEE*, 95(3):467–506, March 2007.

[112] G. Schickhuber and O. McCarthy. Distributed fieldbus and control network systems. *Computing and Control Engineering Journal*, 8(1):21–32, Feb 1997.

[113] Roshan Lal Sharma. *Network Topology Optimization*. John Wiley & Sons, Inc., New York, NY, USA, 1990.

[114] Kanna Shimizu and David L. Dill. Deriving a simulation input generator and a coverage metric from a formal specification. In *DAC*, New Orleans, LA, June 10-14 2002.

[115] Mohit Singh and Lap Chi Lau. Approximating minimum bounded degree spanning trees to within one of optimal. In *STOC '07: Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 661–670, New York, NY, USA, 2007. ACM.

[116] K. Srinivasan, K. S. Chatha, and G. Konjevod. Linear-programming-based techniques for synthesis of network-on-chip architectures. *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, 14(4):407–420, April 2006.

[117] Krishnan Srinivasan, Karam S. Chatha, and Goran Konjevod. Application specific network-on-chip design with guaranteed quality approximation algorithms. In *ASPDAC*, January 2006.

[118] G. L. Stüber. *Priciples of Mobile Communication*. Kluwer Academic Publishers, 1996.

[119] J.-P. Thomesse. Fieldbus technology in industrial automation. *Proceedings of the IEEE*, 93(6):1073–1101, June 2005.

[120] Manish Vachharajani, Neil Vachharajani, and David August. The liberty structural specification language: A high-level modeling language for component reuse. In *Proc. of the Conf. on Programming Language Design and Implementation*, pages 195–206, June 2004.

[121] J.; Ruhl G.; Dighe S.; Wilson H.; Tschanz J.; Finan D.; Iyer P.; Singh A.; Jacob T.; Jain S.; Venkataraman S.; Hoskote Y.; Borkar N. Vangal, S.; Howard. An 80-tile 1.28tflops network-on-chip in 65nm cmos. *Solid-State Circuits Conference, 2007. ISSCC 2007. Digest of Technical Papers. IEEE International*, pages 98–589, 11-15 Feb. 2007.

[122] Tiziano Villa, Timothy Kam, Robert Brayton, and Alberto L. Sangiovanni-Vincentelli. Explicit and implicit algorithms for binate covering problems. *IEEE Transactions on Computer-Aided Design*, 16(7):677–691, July 1997.

[123] Alberto Sangiovanni Vincentelli. Defining platform-based design. *EEDesign of EETimes*, February 2002.

[124] H. S. Wang, X. Zhu, L. S. Peh, and S. Malik. Orion: A power-performance simulator for interconnection networks. In *Proc. of the 35th Intl. Symp. on Microarchitecture*, pages 294–305, November 2002.

[125] Xbow. http://www.xbow.com/.

[126] Ti-Yen Yen and Wayne Wolf. Communication synthesis for distributed embedded systems. In *ICCAD '95: Proceedings of the 1995 IEEE/ACM international conference on Computer-aided design*, pages 288–294, Washington, DC, USA, 1995. IEEE Computer Society.

[127] ZigBee. Zigbee alliance (http://www.zigbee.org).

[128] Hans Zima and Barbara Chapman. *Supercompilers for parallel and vector computers*. ACM Press, New York, NY, USA, 1991.