NoCs: A new Contract between Hardware and Software

Axel Jantsch

Royal Institute of Technology, Stockholm, Sweden
E-mail: axel@imit.kth.se

Abstract

Future single chip systems will resemble more traditional computer networks than traditional central processors. The main reasons for this trend are the infeasibility of global synchrony on a single chip, the necessity of reuse of existing hardware and software components as much as possible, and the heterogeneity and irregularity of system functions and features. The consequences of this trend are far reaching and imply the shift in concern from computation and sequential algorithms to concurrency, communication and interaction in every aspect of design and development of hardware and software.

Based on an analsysis of current trends we suggest that there is an opportunity for defining an interface between applications and Netork-on-Chip (NoC) platform implementations with significant benefits for both worlds. We analyse the desirable properties of such an interface by means of studying a particular NoC platform, the Nostrum. We draw the general conclusion that such an interface, which we also call contract, has to include (1) description of functionality, (2) description of communication semantics and performance, and (3) mapping of task to resources.

1 Introduction

IC manufacturing technology will provide us with a a few billion transistors on a single chip within a few years. Assuming that these predictions hold and that the market will continue to absorb ever higher volumes of ICs, the key questions are: how will the future chips be organized and how will future systems, which include these chips, be designed? One possible answer is that single CPUs will still occupy entire chips and will exhibit correspondingly higher performance. The instruction set will essentially remain unchanged to provide backwards compatibility and the systems will still be implemented in C or C++. However, there are a few factors that make this scenario unlikely:

1. Physical effects of deep sub-micron technology make it increasingly difficult to maintain global synchrony among all parts of the chip. The clock signal will soon need several clock cycles to travel across the chip and the clock distribution tree is already today a major source of power consumption and cost. The trends of scaling to smaller geometric dimension and higher clock frequency make these problems more significant every year.

2. Synthesis and compiler technology development do not keep pace with IC manufacturing technology development. As a consequence, which is called the *design productivity gap*, we need either exponentially growing design teams or design time to design and implement systems which fit onto a single IC. Since both alternatives are unrealistic we have in the past escaped from the problem by using ever more complex components as primitive design units. These primitive design units have evolved from individual transistors to logic gates to entire ALUs, multipliers and finite state machines. This trend will likely continue with CPU and DSP cores and blocks for compression, encryption and similar functions being the primitive design units. These design units have however asynchronous interfaces to the outside and vastly different internal clocking regimes. As a result a *globally asynchronous and locally synchronous (GALS)* design style emerges already today.

3. Obviously, systems that can be implemented on a single chip become increasingly more complex. As a result different functions and features with vastly different characteristics and history reside on the same chip. Signal processing algorithms which recover and generate radio signals will coexist with global control, maintenance and accounting functions as well as with natural language comprehension and generation functions. These functions are developed in different contexts, by different teams, with different design languages and tools. However, they need to be integrated into a single chip.

These factors are not equally important for all application areas. Single CPU systems will still dominate applications where backward compatibility is of primary importance and most problems can reasonably expressed and implemented as a single, sequential algorithm. However, the rapidly growing areas of embedded systems and application domain specific platforms will explore and eventually adopt alternatives.

Taking these current trends and facts together it is natural to contemplate a design paradigm where a set of interacting functions and features are implemented on a set of asynchronously communicating resources, such as CPU cores and specialized hardware blocks. In this scenario the mapping and implementation of system functions onto resources is covered by traditional design and synthesis methods. It may even be an integral part of the reuse of a given system function and a resource. For instance, the purchasing of a bluetooth protocol stack may include its implementation on a combination of a custom hardware block and an ARM processor core. However, providing a chip level communication infra-structure and mapping of system level interactions onto the communication infrastructure is not covered by any traditional design methodology and is becoming the focus of research and tool development. In fact, in the last three years we have seen several concrete proposals for on-chip network architectures.

In 2000 Hemani et al. [6] have proposed a packet switched architecture with switches surrounded by six resources and connected to 6 neighboring switches. The architecture is called a Honeycomb due to the hexagon based pattern of switches and resources. The concept of packet switching re-appears in other consecutive approaches but the topology simplifies in most proposals and today two dimensional meshes, tori and trees are most common. In 2001 Dally and Towles [3] proposed a torus based packet switched network with very simple switches, which require less than 10% area overhead. MicroNetworks proposed by Drew Wingard [21] is another packet switched on-chip interconnection mechanism proposed recently. F. Karim et al. [9] describe an octagon topology, where each node is connected to three other nodes. Octagons can be connected together to form a hierarchical network. The Spin [5, 1] approach is based on a fat tree topology that can be slimmed if the locality of traffic does not require a high bandwidth at the root of the tree. Philips' Æthereal network [16, 4] emphasises the need for guaranteeing well defined quality of service levels for the communication infrastructure. Kumar et al. [11] have put forward a detailed packet switched, mesh based on-chip communication infra-structure together with a design methodology. The proposed concept of a region breaks the strict mesh-based geometry. A region can cover an arbitrary number of switches and resources and allows to accommodate larger resources such as FPGA areas and memory banks in a flexible way. This architecture we will also outline in the consecutive sections of this paper. Valtonen et al. [19, 18] propose an on-chip interconnected network of resources or cells but put the main emphasis on fault tolerance and unlimited scalability. Several other approaches to on-chip packet switched networks have been proposed recently and a good overview of the state of the art can be found in [7].

Keutzer et al. [10] and Sgroi et al. [17] provide general discussions and motivations for communication centric on-chip architectures and platforms and for the strict conceptual separation of computation from communication. The impact of well designed regular communication centered platforms on design productivity have been elaborated in [8].

There is another trend worth noticing that changes the balance between customization and standardization. Tsugio Makimoto observed first in 1987 that the electronics industry experiences cylic changes between customization and standardization with a cycle time of around 10 years. These cycles, also known as Makimoto waves [13], started in 1957 with a standardization phase around discrete standard devices such as transistors and diodes. According to these waves the period 1987 to 1997 saw a customization phase with ASICs being the main instrument for developing dedicated and optimized components. Today we are in a standardization phase with FPGAs fully benefiting from this trend.

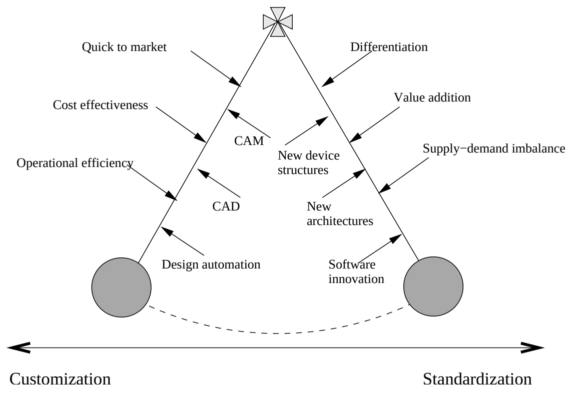

Figure 1. Makimoto's semiconductor pendulum [13]

Figure 1 shows Matimoto's semiconductor pendulum and the forces driving it. Consider the forces pushing it towards standardization. Cost effectiveness is a major concern today due to quickly increasing manufacturing and nonrecurring engineering (NRE) costs [10]. Similarly, there is a strong demand to increase operational efficiency and decrease maintenance costs for increasingly complex electronic equipment. Moreover, the emergance of deep submicron and nantechnology and new architectures such as networks on chips, application specific instruction set processors (ASIP) and new FPGA architectures, coincide with forces pushing towards standardization (new device structures, new architectures). The current trend towards platforms [2, 20] is a natural reaction to these forces and it indicates that the pendulum has not yet reached the culmination of the standardization wave.

Adding the apparent insufficiency of traditional CPUs and the desire for standardization together, it is not remote to expect the emergence of standardized platforms that provide a stable base for the quick and efficient development of new applications and products. It is likely that different application areas require different platforms. For instance mobile terminals will need a different platform than infrastructure applications due to their vastly different requirements on performance, cost and power consumption. Thus, it is open how many specialized platforms will appear on the scene but a small number below ten could possibly dominate most major application areas such as consumer electronics, communication, computing, robotics and automotive.

If this analysis is correct we will see new interfaces between the platforms and the application developers. The interface between processors and software, the instruction set, has been very successful and can serve as a model to understand the major components of the interfaces between next generation platforms and software. The major difference between them and traditional instruction sets stems from concurrency and communication, which is of prime importance in the former but absent from the latter. In order to discuss this interface in concrete terms, we introduce a concrete architecture in the next section. Most details of this architecture are less important for our discussion and will be different in future platforms. However, the strong emphasis on the communication between independent resources is significant and will lead us to important implications.

2 Network on Chip Architecture

The Nostrum architecture [11], as outlined in figure 2, provides the communication infrastructure for the resources. In this way it is possible to develop the hardware of resources independently as stand-alone blocks and create the NOC by connecting the blocks as elements in the network. Moreover, the scalable and configurable network is a flexible platform that can be adapted to the needs of different workloads, while maintaining the generality of application development methods and practices.

2.1 Physical and link layer

A mesh interconnection topology is simplest from a layout perspective and the local interconnections between resources and switches are independent of the size of the network. Moreover, routing in a two-dimensional mesh is easy resulting in potentially small switches, high bandwidth, short clock cycle, and overall scalability. A NoC consists of resources and switches that are connected using channels as a mesh (Manhattan- like structure) so that they are able to communicate with each other by sending messages. A resource is a computation or storage unit. Switches route and buffer messages between resources. Each switch is connected to four other neighboring switches through input and output channels. A channel consists of two one-directional point-to-point buses between two switches or a resource and a switch. Switches may have internal queues to handle congestion. The precise layout and geometry depends on the technology generation. We expect that the area of a resource is the maximal synchronous region in a given technology. It is expected to shrink with every new technology generation. Consequently the number of resources will grow, the switch-to-switch and the switch-to-resource bandwidth will grow, but the network wide communication protocols will be unaffected. Figure 3 illustrates the principles of the physical floor plan within the NOC. Consider a 60nm CMOS technology expected in 2008, a $22mm \times 22mm$ chip size, a resource size of $2mm \times 2mm$ and a minimum wire pitch of $300nm$. A NoC would accommodate 10×10 resources, each switch would occupy $30\mu m \times 30\mu m$ and the channels would use 3 metal layers, hence we have space for 300 wires. Since we need control, handshaking and signaling, this scenario would yield an effective data bus width of 256 wires.

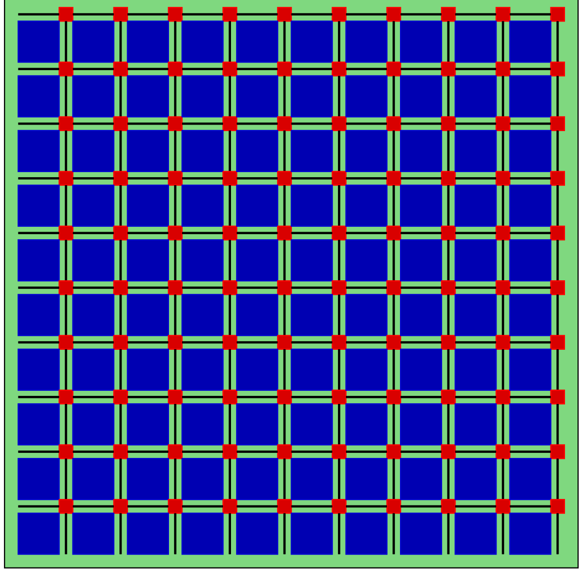
Figure 2. Each node in the mesh contains a switch (small rectangles) and a resource (large rectangles).

Thus, there is a bandwidth of 128 bits per cycle in each direction between two switches. 128 bits is the unit of the data link layer and is called a packet in the rest of this paper.

2.2 Network layer

The network layer is responsible for communicating messages from a resource to any other resource in the network.

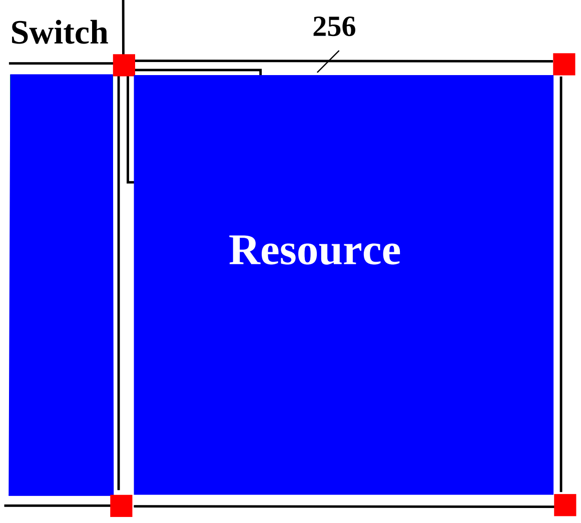

Figure 3. The expected footprints of resource ($2\text{mm} \times 2\text{mm}$), switch ($30\mu\text{m} \times 30\mu\text{m}$) and channels ($2\text{mm} \times 30\mu\text{m}$) in 60nm CMOS technology.

Nostrum offers best effort (BE) and guaranteed latency (GL) communication services. BE communication is based on individual packets that are communicated and routed independently. Hence, each of these packets contain a relatively long header with the full address information. Since BE packets can deviate from the shortest path between source and destination when traffic load is high [14, 15], no maximum delay can be guaranteed. However, packets that have been longer in the network increase their priority and thus their chance to reach the destination. In contrast, GL communication is based on a *virtual circuit* that is a direct stable connection between a source and a destination. A virtual circuit has to be opened and closed. A virtual circuit is opened by reserving particular time slots in every switch from source to destination. These time slots cannot be used by other packets. Consequently, packets traversing an open virtual circuit cannot be disturbed by other BE or GL packets, thus they experience a deterministic latency from source to destination. The GL packets have a much shorter header without address information, because the switches know where GL packets are to be sent.

2.3 Session layer

The session layer manages task level communication. Tasks are concurrent processes that reside in resources. When two communicating tasks are located in the same resource, local communication mechanisms can be used, as for instance provided by the operating system. When they are in different resources network communication services have to be employed.

In Nostrum shared memory and message passing communication is defined and both variants can be parameterized with respect to connectivity, sycnhronization, performance and reliability considerations [12]. For instance a meassage passing channel can be configured with paramters describing the *direction*, *burstiness*, *latency*, *bandwidth*, *quality class* and *reliability level*.

Without going into more details it becomes obvious that applications have a wide range of choices. Indeed, we need to provide all these choices because different applications have different needs and legacy code requires communication primitives popular in the past even if they are less than optimal for future NoC platforms.

While the network layer communication facilities are specific for a particular NoC, is the session layer communication a medium for inter-task communication and NoC independent. Consequently, if we want to be independent of a particular NoC we need to focus on the session layer communication primitives. In fact, they become part of the NoC Assembler Language that forms the interface, or contract, between application and NoC platform implementation, as we will see in the following sections.

3 NoC Design Flow

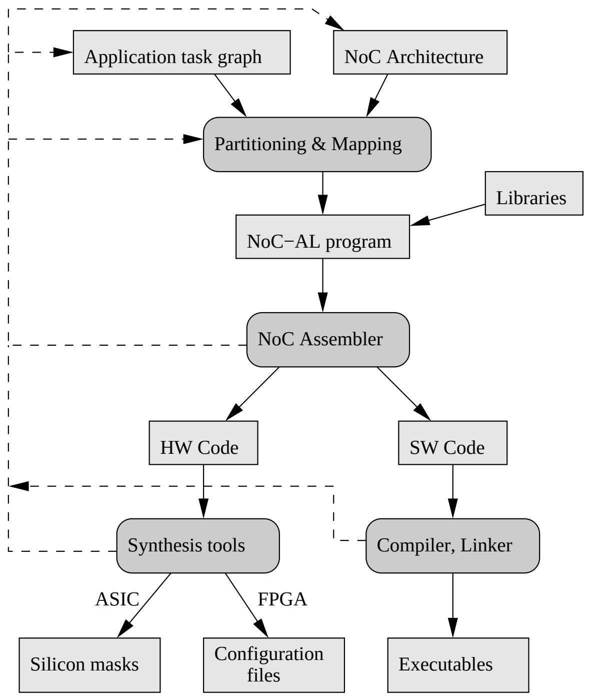

Figure 4. A NoC Platform based design flow [12].

Figure 4 sketches an example design flow for a NoC platform. The primary inputs are the application task graph and a description of the NoC configuration. The task graph captures the functionality of the application in terms of communicating tasks. The NoC architecture defines the

```
NoC Architecture
{ Topology: mesh 1 x 2
  Resource List: Row1: R1=SHARC DSP,
                       R2=ARM CPU}
NoC Application
{ R1: {#include <NoC-AL-SHARC.h>
       #include <f1.h>
       #include <f2.h>
       double in1,in2,out,x,y;
       int ch1=0, ch2=0, ach2=0;
    Process P11
    { while (1)
      { x=f1(in1,in2);
        while (ch1<=0) {ch1=channel(P11,P21);} //Open channel ch1 until success
        while (send(ch1,x)!=1) {continue;}}    //Wait for send to ch1 success
      while (close(ch1)!=1) {continue;}}       //Close channel ch1
    Process P12
    { while (1)
      { while (ach2<=0) {ach2=accept(&ch2);}   //Wait for channel ch2 accepted
        while (receive(ch2,y)!=1) {continue;}  //Wait for receive from ch2 success
        out=f2(y);}}}

  R2: {#include <NoC-AL-ARM.h>
       #include <f3.h>
       double a,b;
       int ch1=0, ch2=0, ach1=0;
    Process P21
    { while (1)
      { while (ach1<=0) {ach1=accept(&ch1);} //Wait for channel ch1 accepted
      while (receive(ch1,a)!=1) {continue;}  //Wait for receive from ch1 success
      b=f3(a);
      while (ch2<=0) {ch2=channel(P21,P12);} //Open channel ch2 until success
      while (send(ch2,b)!=1) {continue;}}    //Wait for send to ch2 success
      while (close(ch2)!=1) {continue;}}}}   //Close channel ch2
```

Figure 5. An example NoC-AL program [12].

size of the network, number, type and location of resources and other global configuration options. The first step maps all tasks onto resources and allocates bandwidth for the communication between the tasks. Apparently this is a most demanding and important step resulting in many key design decisions. After this step we know which tasks reside in which resources and how the communication channels between the resources are configured to provide the required bandwidth and reliability for the inter task communication. The result of this step is described in a *NoC Assembler Language (NoC-AL)* program. The name has been chosen due to the analogy to traditional assembler languages, which also form an architecture specific interface between the higher level compilation process and processor architecture. In the NoC-AL program all major decisions about the NoC size, configuration, resources and mapping has been taken. A NoC-AL program can be implemented on a NoC with a relatively simple and mechanical mapping step as far as the network is concerned. The implementation of the individual resources may still be very complex and require important design decisions because they are implemented with traditional flows and tools. For a processor resource a traditional compilation process has to be performed and perhaps a resource specific operating system has to be intergated. For an FPGA or a custom block a corresponding synthesis flow has to be executed. On the other hand the resource implementation may also be very simple in case a resource is reused together with the functionality in question. For instance, a filter algorithm that has been implemented and optimized for a DSP, can be reused together with the DSP to form a ready-made resource.

Figure 5 shows an example NoC-AL program to illustrate the different parts. First, the topology and the kind of resources are described. In this example we have only two resources, a Sharc DSP and an ARM processor. Then for each resource all the tasks, that are mapped to this resource, are described. Resource R1, the Sharc, contains two processes, P11 and P12. Resource R2, the ARM, contains only one process, P21. The individual processes are ordinary C programs. Note, that no scheduling policy for resource R1 is described here. It is up to the consecutive resource implementation phase to provide a static or dynamic scheduling mechanism.

For other kinds of resources, the processes have to be described in other formalisms. For example custom HW or FPGA resources must be described in VHDL or Verilog. In principle, NoC-AL does not define which languages to use for the individual resources, as long as there exists an implementation flow.

The processes can communicate with each other via predefined communication primitives. For instance process P11 opens a channel to process P21 with the library function channel(). All communication between resources must be expressed with predefined library functions. The same is true for any other language, such as VHDL or Verilog, used for resource description. These libraries implement a well defined communication semantics, which itself is language independent. This allows processes to communicate with each other in a well defined way even if they reside in different resources.

4 HW-SW Interface

NoC-AL is one example of a possible interface between applications and NoC based platforms. It can be used to study the desirable properties of such an interface.

First, it configures the NoC and allocates resources. In NoC-AL the sysntax of this part is specific for Nostrum but it can easily be generalized to any other topology such as tori or trees. There is nothing in later parts of the NoC-AL program which is dependent on precisely how the NoC is configured. Thus, when porting an application to another kind of NoC, e.g. from mesh based Nostrum to tree based SPIN, only this NoC configuration part needs to be rewritten. It may or may not be necessary to change the binding of processes to resources.

Second, the communication between resources is defined in an implementation independent way. Thus, an application can use the communication primitives such as channel(), send(), close(), etc. in any of a number of design languages such as C or VHDL, without committing to a specific NoC. In fact, the communication primitives can be implemented also in bus based communication networks. The implementation of these communication primitives is provided in libraries. These libraries are both NoC and resource specific. Consequently, when a process is mapped to a pariticular processor, say ARM, in a particular NoC, say Nostrum, it will need a library specific for ARM and Nostrum.

Third, NoC-AL and the communication libraries cleanly separate the higher level design tasks from the back-end implementation steps. High level optimizations such as resource allocation and mapping can be performed disconnected from the tedious compilation steps. Since the communication primitives, as described in [12], define also performance and reliability levels, the application can express its non-functional requirements. During implementation (by the NoC Assembler in figure 4) these requirements have to be met. If this is not possible, for instance because the bandwidth requirements of the application are too high for a particular NoC instance, no implementation can be generated and the designer has to reconsider earlier design decisions. For this reason we have the dotted lines in figure 4, which indicate iterations in the flow. Since performance requirements are also placed on the implementation of individual resources, we have to backtrack if these requirements cannot be met.

To put it in a nutshell: a future interface between applications and NoC based platforms must specify

1. the way the functionality of processes is described;
2. the functionality and performance of communcation primitives;
3. how a NoC platform is configured and how process are mapped onto resources.

Item 1 is already provided today by means of standard design languages (C, C++, VHDL, Verilog, SystemC, etc.) and associated design flows. As soon as we also have items 2 and 3, we have a new contract between the application and the implementation world. This would open tremendous potential to quickly map applications onto a NoC platform and even retarget them to other NoC platforms similarly to the way we compile a C program to different CPUs and operating systems today. In the implementation world continuous optimization of platform implementations would be of immediate benefit for a large number of applications.

Figure 6 draws the analogy between a traditional instruction set and NoC-AL with the main difference being that we now have to fully integrate communication and task-resource mapping into the contract.

5 Conclusion

The instruction set has been a powerful contract between processor hardware implementations and software applications that has decoupled these two worlds allowing for independent development and optimizations while making very simple but sufficient assumptions about the other world.

The emergance of NoC platforms provides an opportunity to develop and establish a similarly fruitful and long term contract provided we include communication and task-resource mapping into the agreement.

References

[1] A. Adriahantenaina, H. Charlery, A. Greiner, L. Mortiez, and C. A. Zeferino. SPIN: a scalable packet switched onchip micronetwork. In *Proceedings of the IEEE International Symposium on Circuits and Systems - Designer's Forum*, pages 70–79, March 2003.

[2] H. Chang, L. Cooke, M. Hunt, G. Martin, A. McNelly, and L. Todd. *Surviving the SOC Revolution - A Guide to Platform-Based Design*. Kluwer Academic Publishers, 1999.

[3] W. J. Dally and B. Towles. Route packets, not wires: Onchip interconnection networks. In *Proceedings of the 38th Design Automation Conference*, June 2001.

[4] K. Goossens, J. van Meerbergen, A. Peeters, and P. Wielage. Networks on silicon: Combining best-effort and guaranteed services. In *Proceedings of the Design Automation and Test Conference*, March 2002.

[5] P. Guerrier and A. Greiner. A generic architecture for onchip packet-switched interconnections. In *Proceedings of Design, Automation and test in Europe*, pages 250–256, 2000.

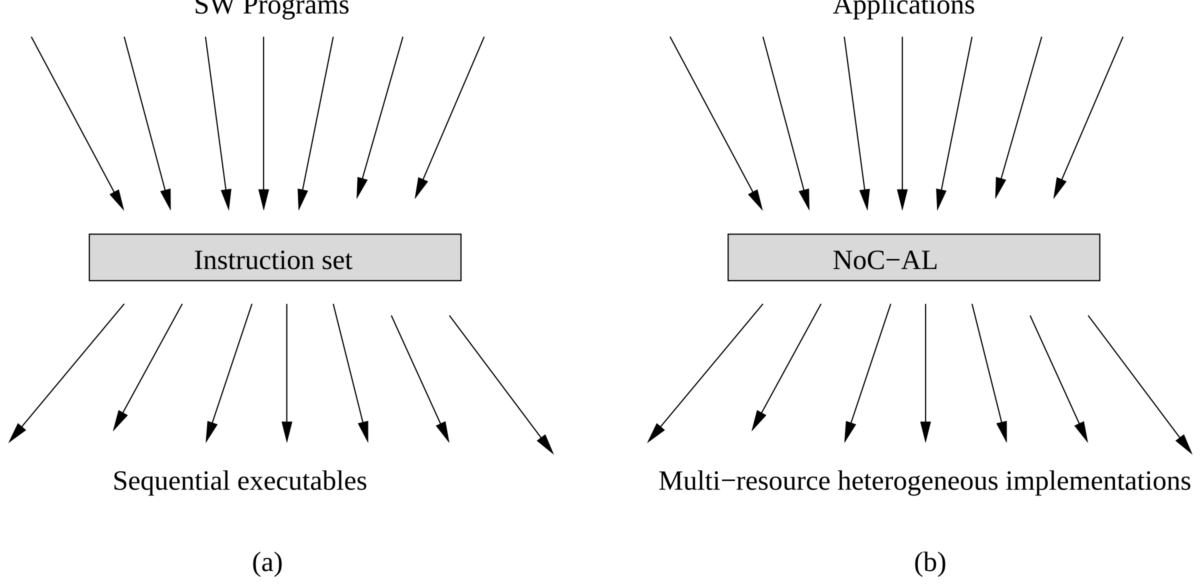

Figure 6. A NoC Assembler Language as interface between application and implementation (b) is similar to an instruction set serving as interface between SW programs and executables (a).

[6] A. Hemani, A. Jantsch, S. Kumar, A. Postula, J. Öberg, M. Millberg, and D. Lindqvist. Network on chip: An architecture for billion transistor era. In *Proceeding of the IEEE NorChip Conference*, November 2000.

[7] A. Jantsch and H. Tenhunen, editors. *Networks on Chip*. Kluwer Academic Publishers, February 2003.

[8] A. Jantsch and H. Tenhunen. Will networks on chip close the productivity gap? In A. Jantsch and H. Tenhunen, editors, *Networks on Chip*, chapter 1, pages 3–18. Kluwer Academic Publishers, February 2003.

[9] F. Karim, A. Nguyen, and S. Dey. An interconnect architecture for networking systems on chip. *IEEE Micro*, 22(5):36–45, September/October 2002.

[10] K. Keutzer, S. Malik, R. Newton, J. Rabaey, and A. Sangiovanni-Vincentelli. System-level design: Orthogonalization of concerns and platform-based design. *IEEE Trasnactions on Computer-Aided Design of Integrated Circuits and Systems*, 19(12):1523–1543, Decmber 2000.

[11] S. Kumar, A. Jantsch, J.-P. Soininen, M. Forsell, M. Millberg, J. Öberg, K. Tiensyrjä, and A. Hemani. A network on chip architecture and design methodology. In *Proceedings of IEEE Computer Society Annual Symposium on VLSI*, April 2002.

[12] Z. Lu and A. Jantsch. Network-on-chip assembler language. Technical Report TRITA-IMIT-LECS R 03:02, version 1.0, Institute of Microelectronics and Information Technology, Royal Institute of Technology (KTH), Stockholm, Sweden, June 2003.

[13] T. Makimoto. The rising wave of field programmability. In R. W. Hartenstein and H. Grnbacher, editors, *Field-Programmable Logic and Applications, 10th International Workshop, FPL 2000*, volume 1896 of *Lecture Notes in Computer Science*, Villach, Austria, August 2000.

[14] E. Nilsson. Design and implementation of a hot-potato switch in a network on chip. Master's thesis, Department of Microelectronics and Information Technology, Royal Institute of Technology, IMIT/LECS 2002-11, Stockholm, Sweden, June 2002.

[15] E. Nilsson, M. Millberg, J. Öberg, and A. Jantsch. Load distribution with the proximity congestion awareness in a network on chip. In *Proceedings of the Design Automation and Test Europe (DATE)*, pages 1126–1127, March 2003.

[16] E. Rijpkema, K. Goossens, A. Radulescu, J. Dielissen, J. van Meerbergen, P. Wielage, and E. Waterlander. Trade offs in the design of a router with both guaranteed and best effort services for networks on chip. In *Proceedings of the Design Automation and Test Conference*, pages 350–355, March 2003.

[17] M. Sgroi, M. Sheets, A. Mihal, K. Keutzer, S. Malik, J. Rabaey, and A. Sangiovanni-Vincentelli. Addressing the system-on-a-chip interconnect woes through communication-based design. In *Proceedings of the 38th Design Automation Conference*, June 2001.

[18] T. Valtonen, J. Isoaho, and H. Tenhunen. An atonomous error-tolerant cell for scalable network-on-chip architectures. In *Proceedings of the 19th IEEE NorChip Conference*, Kista, Sweden, November 2001.

[19] T. Valtonen, T. Nurmi, J. Isoaho, and H. Tenhunen. Interconnection of autonomous error-tolerant cells. In *Proceedings of the International Symposium on Circuits and Systems*, Scottsdale, AZ, USA, 2002.

[20] A. S. Vincentelli. Defining platform-based design. *EEDesign of EETimes*, February 2002.

[21] D. Wingard. MicroNetwork-based integration of SOCs. In *Proceedings of the 38th Design Automation Conference*, June 2001.