Interconnection Generation for System-on-Chip Design

Markus Winter and Gerhard Fettweis
University of Technology Dresden, Vodafone Chair Mobile Communications Systems
winter@ifn.et.tu-dresden.de

***Abstract* – In recent years interconnection architectures for system-on-chip (SoC) design have been subject of intense research. The architectures range from clustered bus based systems to packet-switched network-on-chips. Thereby, the application of these architectures to systems should not be done by hand but automated. In this paper a system is described which automatically instantiates the network and links the modules to it. The underlying network architecture and the network generation flow are discussed.**

I. Introduction

At system-on-chip design (SoC) one fundamental problem is the interconnection between different modules integrated on the chip. Two issues must be addressed in this context. On the one hand the underlying architecture and topology must be considered. On the other hand the application of a chosen interconnection scheme and its verification need a significant amount of time. Communications accesses from modules in a complex system are hardly predictable. Therefore, network behavior must be tested for all assumable cases of access from modules. It is impossible to do this for every system-on-chip again.

One way to use a scalable and well tested interconnection scheme in a SoC is the automated attachment of the network to the modules. This is achieved via code generators. At most established network architectures like AMBA's AHB/APB [1] or IBM's CoreConnect [2] vendors provide tools where user-defined modules are instantiated and automatically connected to the generated network.

Most research in the area of interconnection and network-on-chip design is done in the fields of architectures, topologies or routing mechanisms. An overview about this can be found in section II. In this paper focus will be put on design methodology. A system is described for automatic generation of network architecture and linking to user modules.

In section III the underlying network architecture used in the generator is described. Section IV gives some more detailed descriptions on the generation process and in section V the problem of interfaces will be discussed. Finally, in section VI some results of generation process are considered.

II. Network Architectures Overview

The classical approach for connecting modules on one single chip is using a shared bus. These systems are easy to implement, well tested and understood. Some popular examples are AMBA's AHB [1] or Sonics SiliconBackplane [3]. Unfortunately, these systems have some important drawbacks. First of all, they have very limited bandwidth. Due to high capacity load, their maximum clock frequency is very limited. Next, only one communication at one time is possible. In [4] or [5] the system-on-chip is clustered and several buses are used in parallel. This is the way industry connects their SoCs currently (e.g. [6]). But these systems will not be able to provide necessary interconnection in future because of limited bandwidth and too much clustering.

Network-on-chips (NoC) are the focus of recent research [7]. They base on a number of routers which store and forward arriving packets in direction of the target module. Like in computer networks packet-switched systems lead to more overhead than circuit-switched systems. Packet latency may be high due to several hops on their way along the routers and buffering of packets in routers. Modules and routers are all independent from each other. Therefore, this is a globally asynchronous, locally synchronous (GALS) architecture where every module and router has its own clock ([8]).

However, in today's SoC design there is still the globally synchronous principle in use. But on the one hand performance of shared buses is not sufficient. On the other hand complete NoCs would be overkill for systems of about a dozen modules.

Thus, a way between shared buses and packet-switched NoCs exist. Communication is ensured via circuit-switched crossbar switches interposed between masters and slaves, shown in figure 1. Examples for this are Altera's Avalon ([10]) or specialized systems like in [11]. In such architectures parallel communications with higher bandwidth than provided by buses become possible. The arbitration and multiplexer logic depends very much on number of masters and slaves and some other parameters. Therefore, code for interconnection must be rewritten for every SoC or even for some changes during SoC design. This cannot be done by a designer but should be automated. Thus, automated interconnection tools are required.

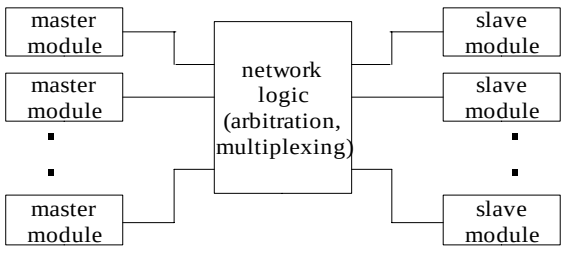

Fig. 1: System-on-chip structure with crossbar switch between modules

III. Basic Network Architecture

All tools for automated interconnection base on a special kind of network. The network architecture for the generation process described in this paper is explained in [11]. A brief introduction will be given in this section.

Network topology is a star based one using circuit-switching and providing parallel connections. Its basic topology can be found in figure 1. Transmission of data is done via requests initiated by masters and responses send by slaves. The master transmits an address to network logic and thus starts a transfer. Within the crossbar switch module address is decoded and associated with one slave. If there is more than one master which wants to get access to that slave, the one with highest priority will be chosen. Hence, fixed arbitration scheme is implemented.

In figure 2 the sequence of requests and responses at an example write transfer can be found. After a master has got access to a slave this is indicated to him via an address acknowledge. Now, master can start transmitting its data. Burst transfers are provided. After a transfer was initiated, several data can be send. Data transmission is completed via an acknowledge signal send by the slave. It indicates whether all data was received successfully.

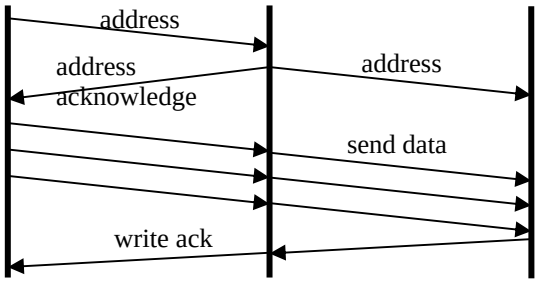

Fig. 2: Example timing chart for interaction between masters, network and slaves

IV. Network Generation Flow

In today's developments time-to-market is a very important aspect of design flow. As system complexity increases development and verification become more time consuming. Therefore, the application of networks on a chip should be automated as much as possible.

The network generation flow bases on XML. XML is a standard freely available for everybody. Parsers and functions to work with XML files exist as open source. Additionally, XML is an easy to understand and widely used way to store data or describe systems. Therefore, the proposed network generator can be used very easily. No special description languages are necessary. Furthermore, XML serves as an independent interface. The XML description can be done by human or by another tool which converts a higher level description into the needed XML-format. Thus, the proposed network generator is completely decoupled from higher levels in the SoC development and design flow.

In this XML-description the specification of required network is done. That means the parameters for network generation are defined. These parameters are the number of masters and slaves, how they are connected, the width of their data/address lines or the maximum burst length possible at one transfer. Additionally, an identification number is assigned to every module connected to the network. This ID specifies the priority of every master.

The generation flow can be seen in figure 3. The XML-specification is read by the network generator, which produces files on different levels of abstraction for desired interconnection.

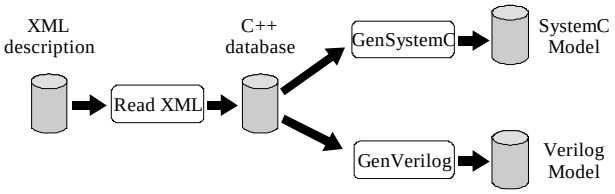

Fig. 3: Network generation flow

In today's system-on-chip design it is crucial to be able to test the whole system on a high abstraction level. By this, fundamental problems in design can be found much earlier and faster due to lower simulation times. Network generation flow must support interconnection on high simulation level. Therefore, a SystemC model as well as a Verilog description of network logic are generated based on the configuration given in XML.

In SystemC communication between modules is realized via C-function calls from one module in another one. For this interfaces must be defined. A module between masters and slaves implements the functions defined in the interfaces and simulates network logic. In figure 4 the interface configurations are shown. Every master and slave connects to the network interfaces and vice versa. Therefore, interface files and the network logic simulation model must be generated

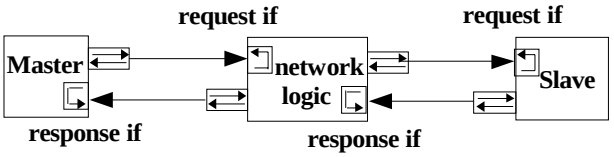

Fig. 4: Interface connections on SystemC level

Another problem is linking all modules together on top level. In SystemC modules are bound together via their ports in just one line of code. But in Verilog multiple signals from each master to network and to each slave must be instantiated and connected. As an example, the signals used in the architecture described in section III, are shown in figure 5.

However, instantiation and interconnection of modules and signals to a system-on-chip is automated and included in the network generator. When the generator will be extended to other network architectures, automated instantiation and interconnection will be an important feature. At packet-switched networks with multiple routers it is no useful work to interconnect and instantiate all routers and modules by hand.

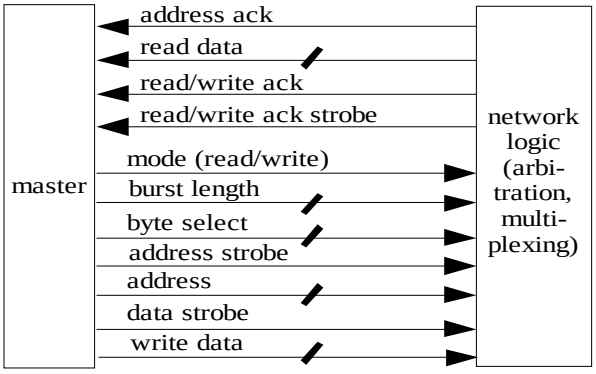

Fig. 5: Interface connections on RTL (Verilog)

The generation process can be combined with the GenCore tool described in [12] and [13]. This tool allows automated design of processor cores. By this approach not only processors but also complete SoCs including their interconnection scheme can be generated easily.

V. Interfaces

One of the major problems of automated interconnection is standardized interfaces. During SoC design or design space exploration it can be useful to test slightly or even completely different interconnection schemes.

Unfortunately, it is hardly possible to change the network without changing connected modules. These modules have to implement the interface associated with special kind of network. But interfaces of different networks are completely different.

Consequently, modules must be adapted according to the network used at the moment. This is a big amount of work which should not be done by engineers. Therefore, a way must be found how interconnection schemes can be substituted without the necessity to rewrite every module again and again.

Two ways seem to be possible in order to achieve this. On the one hand, interfaces and associated protocols can be standardized. Then it does not matter which kind of network will be used to connect the modules. For families of networks, e.g. the family of bus based systems, such standards already exist. An example for this is the Open Core Protocol [14]. But it is difficult to find a common way to describe interfaces from different families. A module is linked in a completely different way to a shared bus system than to packet-switched and mesh-based NoCs.

Another way to reuse already written modules in different networks lies in the generation process. Following section IV not only network logic but also linking between network and modules is automated. If the interfaces between network and module do not match, adapters must be introduced. There are different ways to build these adapters. They can be completely generated by the interconnection generator or only as a template. The user has to fill it and therefore translate the module interface into the network interface.

VI. Generation Results

In network logic slave side arbitration is used. For every slave arbitration and multiplexing of data from masters is done individually and independently from the other slaves. That means if a new slave connection shall be added to network, this will be done without any influence to the other slave connections. If an additional master is added, only the multiplexers must be widened. All this can be done very easily. Hence, network logic is fully scalable depending on number of masters and slaves. Network generation can be automated easily.

In order to test the generation process, networks for different configurations were generated. They differ in the number of masters, slaves, maximum allowed burst length or the width of data or address lines. Some interesting results could be obtained from this generation and following synthesis. Generated network logic was synthesized with UMC 130nm library and Synopsys Design Compiler.

The network switching logic needs time to compute arbitration and forward data via multiplexers. This time was estimated and is shown in figure 6 in dependency of numbers of masters and slaves. From this cycle time T maximum clock frequency f can be estimated via f = 1/T. Of course, cycle time increases with a growing number of masters and slaves because of more complex arbitration and cross coupling between masters and slaves. But it can be seen that with 20 attached modules – 10 masters and 10 slaves – logic latency does not exceed 1.5 ns. Therefore, a network clock frequency of 600 MHz and more is possible.

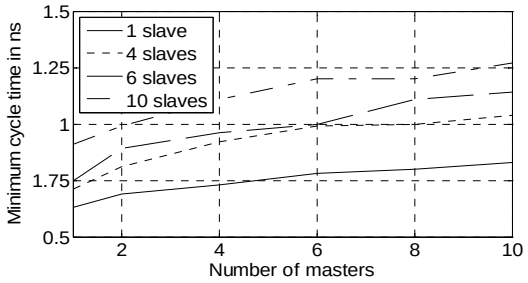

Fig. 6: Minimum cycle time versus number of attached modules

In figure 7 area consumption of network logic can be found. It is given in number of kilo NAND-Gates. The needed area grows linear with increasing number of masters or slaves because for every new module additional multiplexers and registers must be generated. It can be seen, that the overhead which is independent of the number of connected modules does nearly not exist.

It is important, too, whether there is about the same number of masters and slaves or if one kind of module occurs much more often than the other one. In the latter case much more than 20 modules are imaginable because coupling between every master and every slave will not be so costly and consume less area. In figure 8 the area consumption is visualized for 3 cases. 2 cases

are little number of masters and many slaves or vice versa. The 3rd case is the same number of masters and slaves. In the latter one, area consumption grows quadratically. If both the number of masters and slaves grows linear, the matrix for connecting every master to every slave will grow with 2nd order. If only one type of modules grows linear connecting via multiplexers is not that costly. This leads to conclusion that neither registers for control purposes per slave or master nor arbitration logic needs much area. Instead of this, area consumed by multiplexers for cross-coupling between masters and slaves is significant.

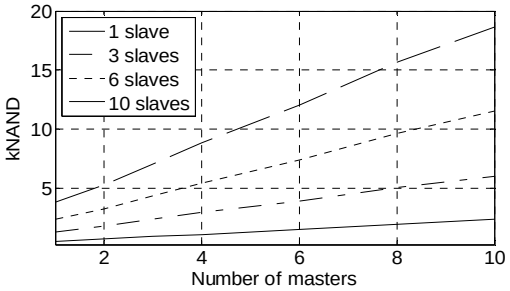

Fig. 7: Area consumption versus number of attached masters

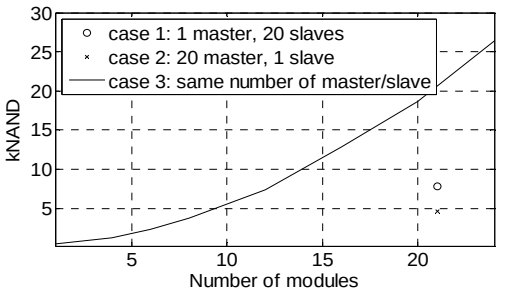

Fig. 8: Area consumption for square matrix connections (10 modules mean 5 masters and 5 slaves); special cases 1 and 2 in the lower right corner

Maximum possible throughput depends very much on the configuration of the generated network. Throughput is determined by maximum bus clock frequency, width of transmitted data and maximum possible parallel transmissions. The number of parallel connections arises from the minimum of number of masters or slaves which is variable thanks for generation. Therefore, it is impossible to give a precise number of transportable bits per second. It differs from network to network.

The general formula to compute maximum possible throughput for the generated network is:

$$TP = \min(N_M, N_S) * f * D$$

where N_M is the number of attached masters, N_S is the number of slaves, f is network clock frequency, D is data width and TP is the resulting throughput.

E.g., assume a SoC consisting of 4 master and 5 slave connections to the network. The bit width of data is 32 and bus clock frequency shall be 400MHz. Because there are 4 masters 4 connections can be established in parallel. In this case theoretical maximum throughput is 51.2 Gbit/s.

VII. Conclusions

In this paper a design methodology and network generator was introduced based on a circuit-switching interconnection architecture. It produces necessary network logic and links the modules to this network. Therefore, complete backbone connection in SoCs can be generated within minutes and does not need to be revised.

Due to generation of interconnection on high SystemC simulation level different system configurations can be tested without much effort. Therefore, the generator is very useful in an early design phase. But not only. Because of generation of synthesizable Verilog it is an important help during SoC design on lower design levels, too.

Following section V there shall be the possibility to easily replace one network architecture by another one. This feature shall be included in next steps. For this, other architectures must be explored and added to the generator.

References

[1] “AMBA Specification (Rev. 2.0)”, ARM Limited, 1999

[2] “The CoreConnect Bus Architecture”, IBM, white paper, 1999

[3] Silicon Backplane, http://www.sonicsinc.com

[4] N. Thepayasuwan, V. Damle, A. Doboli, “Bus Architecture Synthesis for Hardware-Software Co-Design of Deep Submicron Systems on Chip”, Proc. of the 21st International Conference on Computer Design (ICCD 2003), pp. 126-133

[5] C. Hsieh, M. Pedram, “Architectural Energy Optimization by Bus Splitting”, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 21, no. 4, April 2002

[6] O. Strom, “Verification and validation of Atmel’s new 32 Bit AVR microprocessor”, Fraunhofer 9th ITG/GI/GMM Workshop, February 2006, pp. 305-308

[7] Axel Jantsch, “Networks on Chip”, Laboratory for Electronics and Computer Systems, Royal Institute of Technology

[8] W.J. Dally, B. Towles,”Route packets, not wires: On-chip interconnection networks”, Proc. of DAC, June 2001, pp. 684-689

[9] SOPC Builder, Altera, http://www.altera.com/literature/lit-sop.jsp

[10] “Avalon Bus Specification – Reference Manual”, Altera, July 2003

[11] M. Winter, G. Fettweis, “Interconnection Architecture For System-on-Chip Design Providing Little Overhead, Low Latency and High Throughput”, 9th EUROMICRO Conf. on Digital System Design (DSD), Cavtat, Croatia, Aug. 2006

[12] G. Cichon, P. Robelly, H. Seidel, E. Matus, M. Bronzel, G. Fettweis, “Synchronous Transfer Architecture (STA)“, international workshop on systems, architectures, modeling and simulations (SAMOS), July 2004

[13] J.P. Robelly, G. Cichon, H. Seidel, G. Fettweis. "A HW/SW Design Methodology for Embedded SIMD Vector Signal Processors." In International Journal of Embedded Systems IJES, January 2005

[14] “Open Core Protocol”, www.ocpip.org