Reconfigurable Multi-Processor Network-on-Chip on FPGA

Akash Kumar[1], Ido Ovadia[1], Jos Huisken[2], Henk Corporaal[1],
Jef van Meerbergen[1,3] and Yajun Ha[4]

[1]Technical University of Eindhoven, [2]Silicon Hive,
[3]Philips Research, [4]National University of Singapore.
Email: a.kumar@tue.nl

Keywords—MPSoC, NoC, FPGA, reconfigurable network.

***Abstract*—Dealing with real-time constraints is always a problem in a typical System-on-chip design. It is worsened in a multi-processor system connected via a network. FPGA prototyping is a quick way to do a real-time simulation of the system and identify the potential problems. In this paper we propose a reconfigurable MPNoC architecture in which both the network and the processing nodes are configured. The flow allows for each component to be tested separately prior to testing the entire design. This allows for quick design iterations of the system. An example design of such an architecture that has been mapped onto an FPGA is presented.**

I. Introduction

ONE of the major problems when mapping applications to processing platforms like NoCs is dealing with real-time constraints; e.g., how to deal with them on an architecture that includes non-predictable elements like caches and shared buses. This problem is becoming even worse due to the increasing dynamism inside applications and due to the dynamically changing set of running applications (on a single platform); this especially holds for the video domain. Guaranteeing real-time behavior therefore requires dynamic adaptation of the video quality, without being disruptive, and still satisfying non-functional constraints, like latency and throughput constraints. As a result many design iterations are needed. FPGA prototyping is one of the ways to explore the design space and to identify potential bottlenecks in the system, since it allows one to run cycle accurate models on real hardware, and is much faster than simulation.

Here we present a design flow that can be used to generate network-based MPSoC quickly. The application determines the architecture and the communication requirements of the system. The design of computation and communication infrastructure is decoupled. IP blocks (processing nodes) and the network are generated separately. The two are customized to the application requirements and tested at a higher level which allows for quick iterations.

The system was designed with configurable cores from Silicon Hive [3] and connected via the Æthereal network developed by ESAS (Embedded Systems Architecture on Silicon) group at Philips [4]. The Æthereal network is a TDMA based network-on-chip (NoC) that can provide guarantees in communication. The system was simulated and tested and mapped on to an FPGA.

The rest of the paper is organized as follows. We start with summarizing the design flow for Silicon Hive cores and for Æthereal NoC generation. We then introduce the flow which uses both of these and allows one to design and test MPSoC architectures quickly. This is followed by a section on the actual implementation work carried out together with results. The relevant work that has been done in MPSoC and NoC is overviewed, before presenting the conclusions of this research and a direction for the future work that will be carried out to further this research.

II. Silicon Hive Cores

Silicon Hive has an entire tool chain of rapidly designing custom VLIW cores, a library of function units for designers to choose from and adaptive software-development tools [3]. One of the main strengths of Silicon Hive cores lies in the ease with which the cores can be generated with design time configurability. The cores are generated from a flexible architecture template that can vary the number of processing units, function units, register files, interconnects, and local memories. New instructions, function-units and registers can also be added. Even the lengths of operations within the instruction words are configurable.

Figure 1 shows a flowchart of Silicon Hive system design flow. The flow starts with a TIM (The Incredible Machine) description file. In this file one can specify all information relevant for the generation, programming, and simulation of a processor, e.g. register file sizes and widths, interconnect, issue slots, operation sets, custom operations, memory and I/O subsystem of the

processor. Thus, using the TIM language the entire processor can be described in relatively few code lines. TIM also drives the development-tool generator that creates a matching assembler, linker, C compiler, instruction-set simulator, and cycle-accurate simulator. These boxes are shown in grey in the figure. Once a TIM file is created, it is tested with representative programs from the application domain. It provides important feedback to the designer, such as the scheduling of instructions to processor resources (i.e. register files, issue slots, interconnect), which reflects resource utilization.

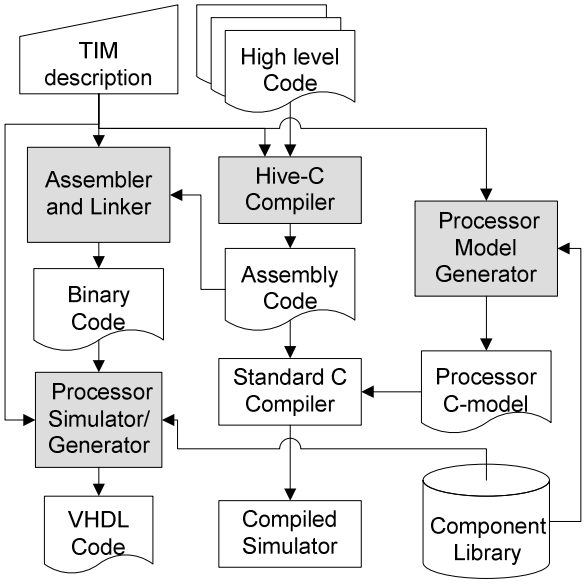

Figure 1: Silicon Hive design flow for cores

Once the design has been verified, a complete synthesizable RTL hardware description of the processors is generated. Pre-written blocks of VHDL or Verilog (stored in the *component library* depicted in the flow) are invoked from TIM description and the processor is generated. This flow has several properties that are useful for processor designers:

- It allows for quick generation of a processor, including VHDL generation.
- It allows for fast design-space exploration of a processor.
- The resulting processors are tuned to specific application domains in terms of area, performance and power trade-offs.

Some of the cores that have been designed by Silicon Hive are *avispa_im2*, *moustique_ic1* and *avispa_ch1* that have specifically been customized for image processing algorithms, camera based applications and wireless OFDM respectively. *Avispa_ch1* has 60 issue slots per word, i.e. it can do up to 60 DSP operations in parallel, the instruction memory is of size 48K, has 103 function units, 130 register files, and 4 dual-port mini-caches. The core area is about 4 mm^2 and it dissipates 150mW when running at 150MHz.

III. Æthereal Network-on-Chip

In this section we briefly describe the design flow of network generation and configuration. A detailed description can be found in [4]. Figure 2 shows the Æthereal design flow. The user provides the architecture around the network together with the communication requirements of the application. Communication, in Æthereal, is expressed by means of connections. A connection specifies a communication between a master port and a slave port, the required (minimum) bandwidth, the (maximum) allowed latency, and burst size for read and/or write data, and the traffic class (best-effort or guaranteed). The user also provides the topology to be used for the underlying network e.g. a mesh or a ring. With these details, a network is generated and the architecture entities mapped to it.

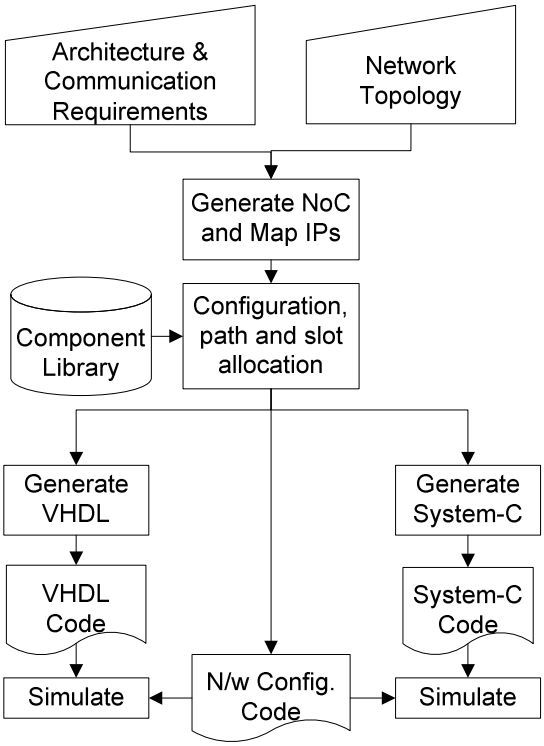

Figure 2: Æthereal design flow for network

For the network itself, many parameters are specified which can be either customized by hand or left to the tool. Some of the things that can be configured are flit duration, number of slots in the TDMA table; arity and BE buffer sizes for the routers; and number of ports, connection per port, and buffer size per connection for each instance of the network interface (NI).

This is followed by the configuration step in which the tool computes the *network configuration code* that contains the information to program the

hardware and setup the connections. Configuration code essentially contains the values to be written to NI registers, such as connection identifiers, and for each connection, the path and other relevant information. An API is available to the programmer to manage the communication between ports at run-time as well. The API supports the functions like *open_config_conn(), close_config_conn(), create_path()*, etc. It should be emphasized that if the network is to be reconfigured for a different communication pattern, it is possible to do so, provided the required hardware is already existent. A *slot* is also allocated for connections with guaranteed latency and throughput.

Once the entire NoC specification is ready, the user may generate a SystemC and/or a VHDL code. In either of these cases, TCL scripts used to simulate traffic are also generated. The TCL scripts are parameterized based on the specified communication requirements. It is also possible to analytically compute results for verification that GT traffic meets the previously specified requirements.

IV. Overall Design Flow

The overall design flow is presented in Figure 3. The application is taken and partitioned by the user. It is split into different sections of the codes to be run on Silicon Hive processors. The communication requirements between the cores are also determined by this partition for the network generation.

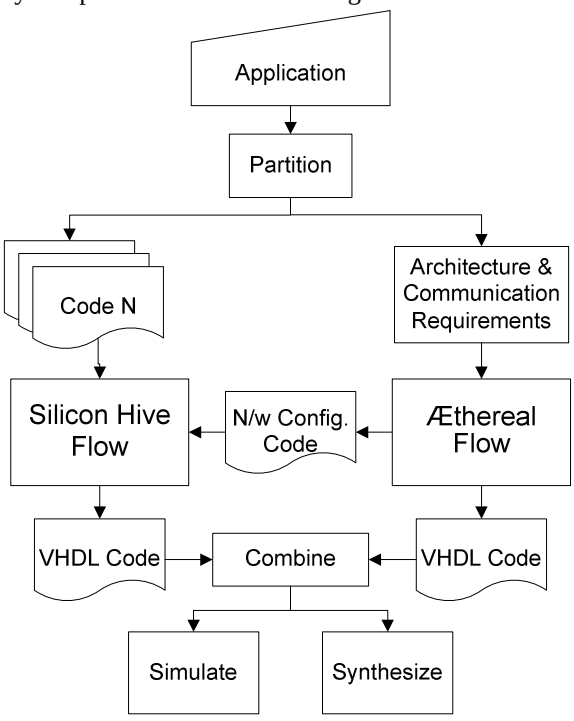

Figure 3: The new flow for overall design

These together with the overall architecture are fed to the Æthereal flow. The flow generates the network and provides the VHDL code of the whole network. It also provides the code which is needed for the configuration of the network. This code is fed to the Silicon Hive flow together with the application code. This generates the processor cores using the flow as explained above and tests them with the application programs.

Once the processor core is verified, the VHDL from the flow is generated and combined with the same from Æthereal flow. The two are simulated using all the code segments to confirm that the overall design is correct. The code can be then synthesized for the required technology accordingly – whether ASIC or FPGA.

The flow was tested with an example and mapped onto FPGA in our case. The same is explained in the next section.

V. Implementation and Results

Figure 4 shows the top level architecture that we decided to implement and test our design flow with. In the application we have two processing nodes and one configuration node. The network itself is rather simple – it has only one router and two network interfaces, one for each processing node – but sufficient to demonstrate the flexibility in the flow. Two connections are needed – in the first one, Node 1 is the master and Node 2 is the slave, while in the second, roles are reversed. The bandwidth for each of these connections is set to 100 MB/s, the data width is 32 bits, and both have "GT" (Guaranteed Throughput) traffic.

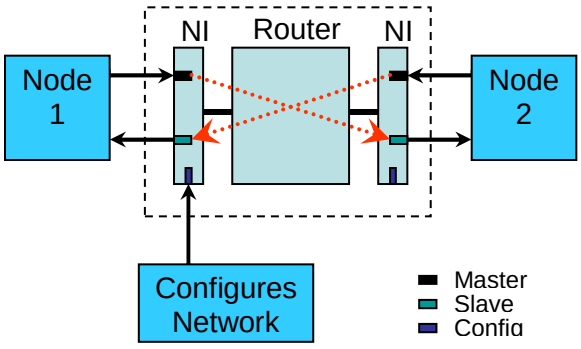

Figure 4: The top-level architecture with connections

The total number of cores needed for the application is 3. For sake of simplicity we decided to use the same core both for configuration of the network and for processing. The core used was customized for 4 issue slots, 32-bit data path, and one master and slave port for communication. The size of memory for data and program was set to 16KB and 32KB respectively. Both Æthereal and

Silicon Hive cores were configured to use DTL (Device Transaction Level) protocol for communication.

From both the flows, VHDL for each was generated and combined as described in the previous section. It was simulated to test if there were any errors in integration. Once the simulation was verified, the source code was synthesized for Xilinx Virtex II [1] series. The generated design was debugged and verified by using both logic analyzer and ChipScope. For both of them, a wrapper had to be written around the actual design to export the signals to be examined on a separate output port. *GoLogic* analyzers from NCI were used for the same [2].

The target platform for the design was Xilinx Virtex II 6000FF1152-C4. The chip itself has about 72,000 logic cells (LCs) and 144 block RAMs of 18 kbits each. The entire design uses about 65% of the entire chip area in terms of LCs. Each processor core takes about 20% while the network takes about 5% including network interfaces and the router. A total of 29 block RAMs were used for memory.

It should be mentioned that ChipScope also uses the block RAMs on-chip for storing the samples. In our example, we used a total of 61 block RAMs for ChipScope. This allowed us to sample 60 signals, each of depth 16,384.

The design was optimized for area and runs at about 12.5 MHz. The maximum frequency at which it can be operated is 18 MHz. The bandwidth achieved is 2.5 MB/s per connection, which is as expected. 100 MB/s is the bandwidth assuming the network runs at 500 MHz, while in the prototype it is only run at 12.5 MHz.

VI. Related Work

For MPSoC design a systematic design-flow has been proposed for hardware/software prototype generation from bus-functional models of various IPs [5]. This is a higher level of abstraction that allows the integration of heterogeneous hardware, software components and sophisticated communication interconnects to adapt different description models. A two-layer hardware-dependent software (HdS) has also been proposed for SoC design [6]. The HdS consists of hardware abstraction layer to abstract the sub-system architecture and SoC abstraction layer to abstract the global MPSoC architecture.

In order to meet the communication requirements of the future MPSoC designs, networks-on-chip are being developed. They are a promising alternative to traditional buses in terms of scalability and wiring. A host of networks are available in the literature. Pande et al [7] reviews the state of the art in this technology in terms of design, automatic synthesis and testing. One of the examples of NoC is Technion's QNoC (QoS NoC) that is based on a 2-D mesh [8]. Pre-emptive priority scheduling provides timing predictability between four service classes, and round robin scheduling is used within a class. An iterative simulation-based approach is used to determine the best network resources.

Another example is Nostrum mesh network [9][10]. In the Nostrum network, hot-potato routing is used for best effort traffic, meaning that packets are always routed and latency is deterministic. GT traffic is facilitated by zero-payload best-effort packets moving back and fourth between the source and destination. When necessary, payload can be added to these empty packets. Thus, these empty best-effort packets essentially reserve bandwidth for payload through a single path in each direction, which is called a *virtual circuit*. As a result, bandwidth is always reserved symmetrically between source and destination, even when no payload arrives.

VII. Conclusions and Future Work

In this paper, we have a presented a novel design flow that can be used to quickly generate network-based MPSoC. The IPs in the design are fully configurable and the network is designed to match the application requirements. As an example a simple architecture is implemented and mapped onto the FPGA.

We are already in the process of generating a more complex network with 4 cores and map a real application on the architecture. Our next course of action is to make the configuration of the application dynamic, i.e. to configure the memories of the processor nodes through a host. Through this dynamic configuration we hope to be able to demonstrate re-configurability of the system to support task dynamism in the system.

Further, we would like to emphasize on integration of the two flows at a higher level. In the current setup, the integration was achieved at the RTL level. Integration at a higher level would allow us to test different variations of the complete system in a shorter time.

Acknowledgment

The authors would like to thank Æthereal and Silicon Hive development teams for their assistance.

References

[1] [Online]. Available: http://www.xilinx.com/.
[2] [Online]. Available: http://www.nci-usa.com/.
[3] Tom R. Halfhill, "Silicon Hive Breaks Out", Microprocessor Report, Dec 1, 2003.

[4] Andrei R˘adulescu et al, “An Efficient On-Chip Network Interface Offering Guaranteed Services, Shared-Memory Abstraction, and Flexible Network Configuration”, IEEE Transactions on CAD of Integrated Circuits and Systems, 24(1), January 2005.

[5] Petkov, I et al., "Systematic design flow for fast hardware/software prototype generation from bus functional model for MPSoC," The 16th IEEE International Workshop on Rapid System Prototyping, 2005. (RSP 2005), vol., no.pp. 218- 224, 8-10 June 2005.

[6] Sungjoo Yoo et al., "Multi-processor SoC design methodology using a concept of two-layer hardware-dependent software," Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings , vol.2, no.pp. 1382- 1383 Vol.2, 16-20 Feb. 2004.

[7] Pande, P.P.; Grecu, C.; Ivanov, A.; Saleh, R.; De Micheli, G., "Design, synthesis, and test of networks on chips," Design & Test of Computers, IEEE , vol.22, no.5pp. 404-413, Sept.-Oct. 2005.

[8] Bolotin, et al, “Automatic Hardware-Efficient SoC Integration by QoS Network on Chip”. Proc. ICECS, 2004

[9] Millberg et al, “Guaranteed bandwidth using looped containers in temporally disjoint networks within the Nostrum network on chip”, Proc. DATE, February 2004

[10] Andreasson and Kumar,: ‘On improving Best-effort throughput by better utilization of Guaranteed-Throughput channels in an on-chip communication system’, Proc. IEEE Norchip, November 2004.