NoC Interface for a Protocol Processor

Seppo Virtanen, Jani Paakkulainen, Tero Nurmi, Jouni Isoaho

University of Turku, Dept. of Information Technology
and Turku Centre for Computer Science (TUCS)
Lemminkäisenkatu 14 A, FIN-20520 Turku, Finland
Email: seppo.virtanen@utu.fi, jani.paakkulainen@utu.fi, tero.nurmi@utu.fi, jouni.isoaho@utu.fi

***Abstract*— In this paper we present our design and implementation of Network-on-Chip (NoC) support into our TACO protocol processor architecture. Our signaling scheme is Virtual Component Interface standard (VCI) compliant. Due to the data- and I/O-intensive nature of protocol processing, memory access from I/O logic plays a key role in NoC interface design. We have addressed this problem by using dual-port memory for protocol data units (PDUs) and a separate single port memory for other user data. We evaluated our NoC interface with VHDL synthesis of a case study in IPv6 processing. Based on our simulations and synthesis, the logic part is able to operate at 200 MHz in 0.18 μm CMOS technology. Adding a NoC interface into our architecture did not considerably increase area and power costs.**

I. Introduction

With tightening design schedules and more complex verification requirements, the decision to include an IP block in a SoC or NoC design often depends on the on-chip communication mechanisms supported by the IP. If several IP blocks are available for the same on-chip functionality or processing task, the ones without support for typical on-chip communication schemes are among the first to be discarded. The Virtual Component Interface Standard (VCI) [13] defines on-chip interfaces for interconnecting different kinds of complex IP blocks. VCI makes it possible for compliant IPs to communicate with each other easily and without errors. By adhering to the interface definitions given in the VCI standard, IPs developed in different organizations can be integrated into a single network-on-chip with less effort and thus in less time.

In our research project TACO (Tools for Application-specific Hardware/Software Codesign) we are developing a methodology for obtaining an optimized protocol processor architecture, its instruction set and application code from a high abstraction level protocol processing application description. We have developed a transport triggered base protocol processor architecture [10], [12] and system-level simulation, physical characteristics estimation and synthesis models for it [4], [9], [11].

The external communication in our TACO protocol processor architecture has until now been conducted using custom solutions. For some time we have seen it necessary to enhance the architecture by adding support for a standard on-chip communication scheme. Recently we were invited to participate in a NoC design co-operation project within the national COMPLAIN research project in Finland. We were asked to provide an IPv6 client implementation of our TACO protocol processor architecture for inclusion in a multimedia processing NoC platform. The IPv6 client core was required to be BVCI (Basic VCI) [13] compliant. Thus, we proceeded to specify, design and implement a BVCI interface module into our protocol processor architecture. The module acts as a wrapper that maps signals from our existing input/output blocks to VCI compliant signals towards an on-chip bus.

In the following pages, we first give an overview of our TACO protocol processor design framework and the base processor architecture. Then we present our design and implementation of a VCI-compliant Network-on-Chip interface. We conclude the paper with results from a case study, in which we were especially interested in determining the amount of additional area and power required by adding a NoC interface into our architecture.

II. The TACO protocol processor platform

The challenge in protocol processor design is to find an architecture that is a good compromise between a general purpose processor and a custom, protocol-specific processor (ASIC): ideally the architecture should be programmable, and at the same time optimized for a family of protocols and tasks required in the processing of these protocols. The TACO protocol processor design framework [9] addresses this design problem by providing tools and methods for helping the designer in specifying, simulating, evaluating and synthesizing programmable protocol processors. These protocol processors are called TACO processors [10], [12].

The starting point in the TACO design flow is a high level application description or specification. The development of the application software guides the processor design work so that the processing requirements of the application determine the hardware architecture of a protocol processor [3], [8]. The approach is quite different from what is found in most commercial protocol processors available today. They are most often multiprocessors with high performance general purpose computing elements.

The TACO protocol processor architecture, as seen in Fig. 1, is based on transport triggered architectures (TTA) [1], [7]. The most important difference between TTAs and traditional processors is that in TTAs operations occur as side effects of programmed data transports. In traditional processors the operations are programmed, and data transports occur as side effects of the programmed operations. A TTA based processor

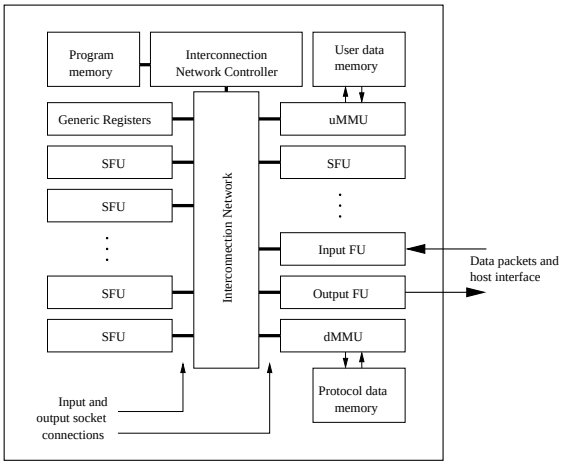

Fig. 1. A functional view of the TACO architecture.

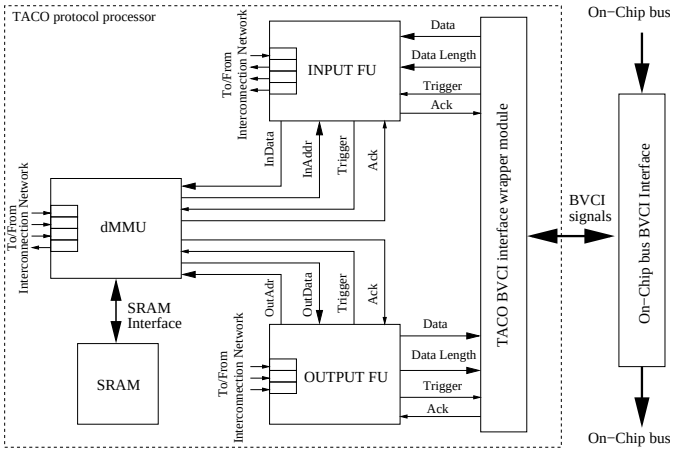

Fig. 2. Connections between the interconnection network, internal datagram memory, input and output FUs and the BVCI interface in a TACO processor.

is composed of functional units (FUs) that communicate via an interconnection network. This network of data buses is controlled by a special-purpose controller unit. The FUs are connected to the buses through modules called sockets. It was observed in the TACO project that this kind of modularity facilitates both component reuse and hardware design automation.

Although TTA based, the TACO architecture differs from basic TTA in many ways. The reader is referred to [12] for a detailed discussion on the differences. Within the scope of this paper we limit ourselves to stating one important difference between the two architectures: all TACO functional units are Special Functional Units (SFUs, as seen in Fig. 1). The generic TTA FUs as described in [1] are missing from the TACO architecture. SFUs are FUs with a distinct application-domain specific task to execute, and usually they can not efficiently be used in some other application domain. The application domain for TACO SFUs is protocol processing.

The SFUs in TACO processors have a varying number of input and output registers with programmable addresses. SFU operations are executed every time data is moved to a specific kind of input register, the trigger register. Each SFU has one trigger register, and each SFU performs a specific protocol processing task or operation (for example, CRC calculation or logical comparisons).

The benefit of TTA-based platforms is their modularity and scalability. Functional units can be added to the architecture or they can be refined and changed as long as they provide the same interface to the sockets connecting them to the interconnection network. The TACO architecture (see Fig. 1) is therefore more of a template for protocol processors, instantiated for a specific protocol, or family of protocols. Extensive component and module reuse is typical for TACO design projects.

III. TACO NoC Interface

As seen in Fig.1, data and control I/O in TACO processors is normally managed by two SFUs, the input and output FUs. These two units take care of all external connections. Until now we have used a proprietary signalling scheme in the input and output FUs. To implement BVCI support into our architecture, we had two major design alternatives:

1. Remove the input and output FUs and replace them with BVCI compliant FUs.
2. Design a wrapper module that maps signals from the existing FUs to BVCI signals.

We decided to proceed with alternative **2.** This way we would not have to redesign the existing FUs and we would be loyal to one of the key principles within the TACO design framework: modularity. With a wrapper module we could include BVCI support when necessary, and leave it out in projects that do not require NoC bus compliancy.

The I/O path in TACO processors is constructed of the input and output FUs, the protocol data MMU (dMMU) and the protocol data memory. Fig. 2 shows the connections between the modules in the path and how the new BVCI wrapper module connects to the path. The dMMU manages access to the dual-port protocol data memory. If dMMU notifies the Input FU that incoming data can be written into the memory, the data is passed through the Input FU and dMMU to the memory. Some information, like the base memory address of the incoming protocol data unit (PDU), is stored into the Input FU. While the PDU is being written into the memory, its processing can be started by the other functional units through the interconnection network. Since the memory is dual-port, it can simultaneously be read from and written to as long as the operations do not concern the same address.

To write modified data back into the protocol data memory, the functional units accessing the memory through the interconnection network need to either wait for the incoming PDU write to finish, or to use the user data memory (see Fig. 1) for temporary storage. This of course depends on the application and its program code implementation.

Once the PDU has been processed, the Output FU is informed of an outbound PDU. The Output FU accesses the protocol data memory from a given base address and sends out

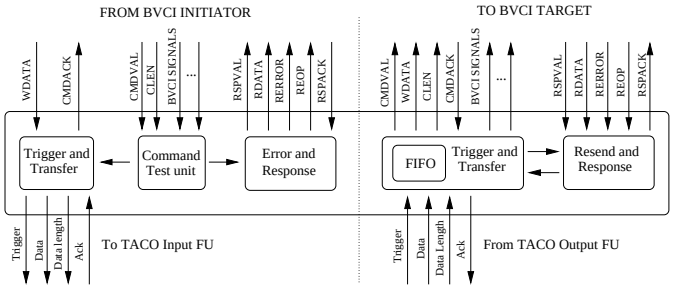

Fig. 3. Functional block diagram of the TACO NoC wrapper module.

a given number of data words. In our current implementation the Output FU blocks other read accesses to the protocol data memory while it sends a processed PDU. While the Output FU is sending a PDU, a new PDU can simultaneously be written into the protocol data memory through the Input FU and its processing can start immediately.

So far the above memory access scheme has not caused problems for us in terms of performance. Due to the basic nature of communication protocols, PDUs are always of reasonable size (no matter which protocol is used). So, transferring PDUs does not block the rest of the processing for too long. Also, PDU processing can usually be carried out at a much higher speed than the speed at which the PDUs appear at the inputs of the processor. The reason we have implemented the memory access this way is to be able to use dual-port memory; if the Input and Output FUs and the interconnection network should all have simultaneous access to the protocol data memory, a four-port memory implementation would be needed. Another solution that we intend to explore in the near future is to design and implement an equal-priority arbitration scheme into the dMMU. Naturally, a four-port access scheme would still be the fastest alternative, if it could be implemented at the same speed as two-port access.

Fig. 3 shows the internal structure of the TACO VCI wrapper module. On the input side (left side of Fig. 3), the TACO protocol processor acts as a VCI target and the VCI interface on the on-chip network side acts as a VCI initiator. The communication is started by the initiator raising the CMDVAL signal and asserting certain other VCI signals (e.g. WDATA, CMD, CLEN). On CMDVAL, the TACO interface wrapper checks that the correct VCI command is issued. If the command is incorrect, the wrapper responds by generating a VCI error packet to the on-chip originator of the erroneous data. If the command is correct, the wrapper raises the trigger signal towards the Input FU. If the Input FU responds with an Ack, the CMDACK signal is raised, and data transfer from the on-chip network to the TACO dMMU begins.

On the output side (right side of Fig. 3), the TACO wrapper acts as the VCI initiator. When the Output FU is ready to send data to an on-chip IP, it raises the trigger signal. The wrapper responds with an Ack towards the Output FU, if there is space left on the output FIFO of the wrapper. If the FIFO is full, the Ack is not asserted. The FIFO size is parameterizable, and is typically set to match the packet size of the on-chip network. With at least one item in the FIFO, the wrapper raises the CMDVAL signal towards the VCI interface on the on-chip network side. As soon as a CMDACK signal is detected by the wrapper, data from the FIFO is transferred over the VCI interface to the target.

IV. Implementation

As mentioned in the introduction, we were invited to participate in a NoC design co-operation project within the national COMPLAIN research project in Finland. The objective in the co-operation was to combine existing IPs developed in the participating universities into a multimedia processing Network-on-Chip platform. The target application for the platform was specified to be receiving, decrypting and decompressing an encrypted MPEG video stream transmitted over a WLAN-IPv6 connection. The main building blocks for the platform were the PROTEO packet switching on-chip network [6], a TACO processor for IPv6 processing, an RSA decryption unit [5] and a RISC processor core for decompressing the video stream. The PROTEO network required VCI compliancy from all included IPs. The platform was required to be able to process incoming data in a 100 Mbps network environment.

We had already previously designed and implemented a TACO processor for IPv6 routing [12]. For the NoC platform we needed to provide an IPv6 client processor. The processor should receive and verify incoming IPv6 datagrams and remove the IPv6 header information from them. The datagram payloads should then be transported to the decryption block through the PROTEO on-chip network. After examining the IPv6 client application we decided to modify the existing router processor architecture and its application code to obtain an IPv6 client processor model. Using the available modules in our TACO libraries we constructed a processor architecture instance with one 32-bit internal data bus, and the functional units listed in table I. After specifying the architecture for IPv6 client operation we specified and implemented a BVCI wrapper unit between the NoC interface and the TACO input and output FUs as described in the previous section.

SystemC simulations of the TACO IPv6 client and its application code revealed that the IPv6 client operation requires 1920 clock cycles per datagram. In a 100 Mbps network environment (with a peak transmission rate of about 8300 datagrams per second) this means that the minimum clock speed for a TACO IPv6 client with the chosen set of functional units and interconnection buses is 17 MHz. We proceeded to synthesize the TACO IPv6 client using 0.18 μm CMOS technology and a standard cell library with the target clock speed of 200 MHz. Targeting the synthesis to this clock speed should make it possible for the IPv6 client to operate at network speeds of up to 1 Gbps. The module areas obtained from synthesis are shown in Table II.

Power consumption of different TACO modules and the wrapper module has been estimated with an early estimation model based on Rent's rule [2]. From Rent's rule and the

FU type	description
Matcher	Find a bitstring inside another data word
Shifter	Shift data logically left or right
Comparator	Make Boolean evaluations ($<, >, =, ...$)
Masker	Replace sequence of bits with another sequence
IP Checksum	Calculate IP checksum for given words
Counter	Count up or down
Input FU	Data/host input
Output FU	Data/host output
dMMU	Packet (datagram) memory management
uMMU	User data memory management

TABLE I

IPV6 CLIENT FUNCTIONAL UNITS AND THEIR DESCRIPTIONS.

	Estimated average power [mW]	Synthesized area [μm^2]
Matcher	14.9	12 851
Shifter	26.9	19 302
Comparator	34.1	17 197
Masker	13.0	11 335
Checksum	34.2	15 595
Counter	32.9	11 782
Input FU	30.5	26 064
Output FU	28.7	17 002
dMMU	15.6	11 721
uMMU	8.4	7342
Sockets (35)	79.1	31 720
Network Controller	33.4	21 169
Total (IPv6 client part)	351.7	203 080
BVCI Interface wrapper	33.7	23 137
Total (IPv6 client with BVCI wrapper)	385.4	226 217

TABLE II

ESTIMATED POWER CONSUMPTION AND SYNTHESIZED AREAS FOR ALL MODULES IN THE TACO IPV6 CLIENT PROTOCOL PROCESSOR.

information on standard cell library (0.18 μm) we can estimate the number of gates and the power consumption in all modules.

Designer experience helps in estimating Rent's constant (i.e. the average number of I/Os in one logic gate) and corresponding Rent's exponent. In the estimations we used NAND2 gate and D flip flop as our basic gates (both with driving strength of 2) representing combinatorial and sequential logic, respectively. In power estimation the frequency of 200 MHz has been used.

The results presented in Table II revealed that the BVCI wrapper module did not considerably increase the size or power use of the IPv6 client processor. In fact, the increase in both size and power use is about the same as the cost of adding one functional unit into the architecture. For the IPv6 client processor discussed above, the area is increased by 10 % and the estimated power consumption by 9 %. The relative increase in area and power consumption is naturally reduced as more functional units and interconnection network buses are added in the architecture. The number of functional units and buses needed in a TACO architecture increases with the complexity of the target application.

V. CONCLUSIONS

We presented the latest improvement to our protocol processor architecture, a VCI compliant NoC interface. By resorting to the VCI standard, we see ourselves now able to provide TACO processors as IPs to a variety of NoC projects. Our NoC interface acts as a wrapper that maps signals from the input/output functional units of our protocol processors to VCI compliant signals towards an on-chip NoC bus.

Our implementation showed that the amount of additional area and power required by adding a NoC interface into our architecture was tolerable: our case study of an IPv6 client processor required 10 % more area and 9 % more power with VCI support included. As a generalization, we concluded that the area and power consumed by our VCI interface wrapper module equals the area and power consumed by an average functional unit. The IPv6 client was synthesized using 0.18 μm CMOS technology and a standard cell library with the target clock speed of 200 MHz. Targeting the synthesis to this clock speed makes it possible for the IPv6 client to operate at network speeds of up to 1 Gbps.

REFERENCES

[1] H. Corporaal. *Microprocessor Architectures - from VLIW to TTA*. John Wiley and Sons Ltd., Chichester, West Sussex, England, 1998.

[2] B. S. Landman and R. L. Russo. On a pin versus block relationship for partitions of logic graphs. *IEEE Transactions on Computers*, C-20(12):1469–1479, December 1971.

[3] J. Lilius and D. Truscan. UML-driven TTA-based protocol processor design. In *Proceedings of the 2002 Forum for Design and Specification Languages (FDL'02)*, Marseille, France, September 2002.

[4] T. Nurmi, S. Virtanen, J. Isoaho, and H. Tenhunen. Physical modeling and system level performance characterization of a protocol processor architecture. In *Proceedings of the 18th IEEE NORCHIP Conference*, pages 294–301, Turku, Finland, November 2000.

[5] T. Ristimäki and J. Nurmi. Implementation of a fast 1024-bit RSA encryption on platform FPGA. In *Proceedings of the 6th IEEE International Workshop on Design and Diagnostics of Electronics Circuits and Systems (DDECS'03)*, Poznan, Poland, April 2003.

[6] D. Sigüenza-Tortosa and J. Nurmi. Proteo: A new approach to network-on-chip. In *Proceedings of the IASTED International Conference on Communication Systems and Networks (CSN'02)*, Malaga, Spain, September 2002.

[7] D. Tabak and G. J. Lipovski. MOVE architecture in digital controllers. *IEEE Transactions on Computers*, 29(2):180–190, February 1980.

[8] S. Virtanen and J. Lilius. The TACO protocol processor simulation environment. In *Proceedings of the 9th International Symposium on Hardware/Software Codesign (CODES'01)*, pages 201–206, Copenhagen, Denmark, April 2001.

[9] S. Virtanen, J. Lilius, T. Nurmi, and T. Westerlund. TACO: Rapid design space exploration for protocol processors. In *the Ninth IEEE/DATC Electronic Design Processes Workshop Notes*, Monterey, CA, USA, April 2002.

[10] S. Virtanen, J. Lilius, and T. Westerlund. A processor architecture for the TACO protocol processor development framework. In *Proceedings of the 18th IEEE NORCHIP Conference*, pages 204–211, Turku, Finland, November 2000.

[11] S. Virtanen, D. Truscan, and J. Lilius. SystemC based object oriented system design. In *Proceedings of the 2001 Forum on Design Languages (FDL'01)*, Lyon, France, September 2001.

[12] S. Virtanen, D. Truscan, and J. Lilius. TACO IPv6 router - a case study in protocol processor design. Technical Report 528, Turku Centre for Computer Science, Turku, Finland, April 2003.

[13] VSI Alliance. *Virtual Component Interface Standard*. VSIA, April 2001.