Design-for-Test of Asynchronous Networks-on-Chip

Xuan-Tu TRAN*†, Vincent BEROULLE†, Jean DURUPT*, Chantal ROBACH†, and François BERTRAND*
*CEA/LETI — 17, rue des Martyrs, 38054 Grenoble, France.
†INPG/LCIS — 50, rue Barthlmy de Laffemas, 26902 Valence, France.
Email: Xuan-Tu.Tran@cea.fr; Telephone: +33-4-38 78 29 56

***Abstract*— Thanks to many advantages, asynchronous circuits have been used to solve the interconnect problems faced by system-on-chip (SoC) designers. Some asynchronous Networks-on-Chip (NoCs) architectures are proposed for the communication within SoCs, but lack methodology and support for manufacturing test to ensure these communication architectures work correctly. In this paper, we present an innovative asynchronous DfT architecture that allows to test the asynchronous communication network architectures, as well as the synchronous computing resources and the asynchronous/synchronous network interfaces on the asynchronous NoC-based SoCs. This asynchronous DfT architecture is implemented in Quasi Delay Insensitive (QDI) asynchronous circuits and uses an area of about $20 * 8$ Kgates in an asynchronous NoC-based SoC of 4.5 Mgates without memories.**

I. Introduction

As clock distribution on a large die becomes more and more difficult in System-on-Chip (SoC) design, the SoC designers have paid attention to asynchronous circuits. The ITRS road map predicts that the global asynchronous – local synchronous (GALS) platform will become the mainstream in the near future. In a GALS system, a number of synchronous islands communicate asynchronously with each other using an asynchronous communication networks. As a result, the asynchronous Network-on-Chip (NoC) paradigm has become an alternate solution for the communication of large SoC designs thanks to many advantages. Some asynchronous NoC are proposed such as CHAIN [1], NEXUS [2]. However, the testability of asynchronous circuits is one of the most challenging problems in designing a GALS system. In other words, one of the challenges of a GALS system design is how to ensure the asynchronous networks work correctly. Solving this problem will allow the SoC designers to make the GALS systems become reality.

Unfortunately, it is very difficult to test asynchronous circuits because of the many feedback loops. Only one defect may cause a suspension of the whole circuit. Some works on the test of asynchronous circuits are proposed in [3]. In these propositions, the test of the asynchronous circuits is done by inserting scan-latches to break the feedback loops. These works are less useful when the circuit under test has numerous Muller ports because of the increase of the test overhead.

The CEA-LETI has proposed an asynchronous NoC (ANoC) architecture [4], adapted to GALS systems, providing low latency for QoS and implemented in Quasi Delay Insensitive logic (QDI) style. To make this architecture practical for industrial applications, a test methodology is needed.

In this paper, we present an innovative Design-for-Test (DfT) methodology for the ANoC architecture. A generic, modular, scalable and configurable architecture is modeled in asynchronous logic and implemented in QDI circuit style. This architecture is suitable for the ANoC design and also reusable for the next designs.

The organization of the paper is as follows: Section II presents briefly the context and objectives of our work; Section III recalls the Asynchronous NoC architecture, for which we develop a DfT architecture; Section IV presents an innovative DfT architecture for the ANoC; The design and verification of this architecture is presented in Section V; And finally, some results and conclusions are given in Section VI.

II. Networks-on-Chip: Design and Test

In the coming billion-transistor era, the trend of SoC design is to integrate numerous IP blocks: processors, embedded memories, computing resources, etc. The bus-based architecture does not meet the increasing demands by the communication on the large SoCs. The NoC paradigm is emerging as a new design methodology for SoCs with numerous advantages compared to bus-based SoCs such as good efficiency, high scalability and versatility, as well as high bandwidth communication. Many NoC architectures with numerous topologies and design methodologies are proposed but there are few works on the test of the NoC-based Systems [5]–[8].

The test strategy for NoC-based systems addresses three problems: testing the IPs (i.e., the synchronous computation resources) and their corresponding network interfaces; testing the interconnect infrastructure itself (the nodes and the inter-node wire segments); and testing the whole integrated system.

To test the embedded IPs and their corresponding network interfaces, it requires test access mechanisms (TAMs) to transport the test vectors from the source to the core-under-test and the test results from the core-under-test to the sink. Some researches propose reusing the on-chip network as a high bandwidth TAM for the embedded IPs [9], [10]. The principal advantages of reusing NoCs as TAMs is that no extra TAM hardware cost is needed and the availability of multi-path core test.

To test the interconnect infrastructure, we must address two aspects: testing the node and testing the wire segments between the nodes. Because the nodes consist of FIFO buffers and routing logics, some works propose that we can

test these parts separately, using a BIST for FIFOs and a traditional method for the routing logics. However, the FIFO buffers are distributed all over the chip and it poses a big challenge for this approach. In our opinion, we prefer to test the whole node with an unique DfT architecture. This approach becomes more favorable when the network nodes are designed in asynchronous logic.

III. The Asynchronous NoC Architecture

A. Architecture description

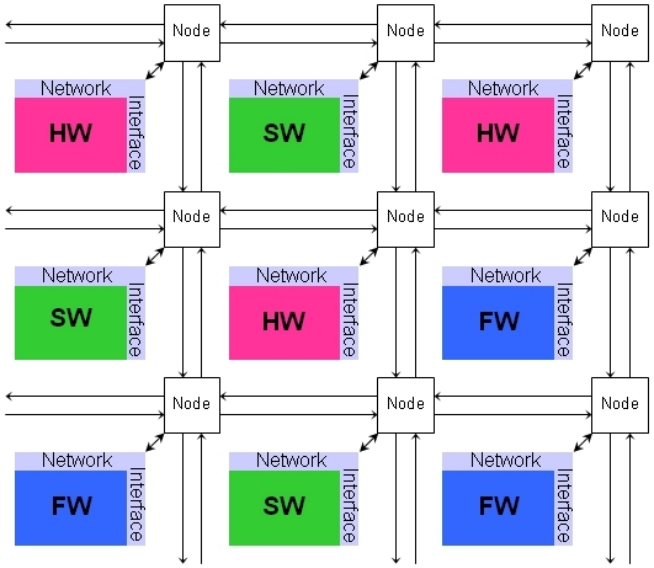

Fig. 1. Networks-on-Chip architecture.

The ANoC developed by the CEA-LETI [4] is composed of asynchronous nodes, links between nodes, synchronous computation resources (they may be hardware, software, or firmware resources), and network interfaces between the network and the resources, see Figure 1. The network nodes compute where to transmit an incoming data, arbitrate between potential concurrent data, and finally transmit the selected data to the selected link. The links between the nodes are bi-directional 32-bit data paths, implemented with a flit-level handshake signaling protocol.

The synchronous computing resources can either be generic blocks (CPU, DSP, memories, ...) or be configurable hardware IP blocks (FFT, MPEG, ...). Each resource is connected to the communication network by a network interface with the nearest node. The network interface is an asynchronous/synchronous interface block that contains a dedicated GALS interface in order to perform synchronization between the synchronous and asynchronous domains.

This asynchronous NoC is targeted to be used in the FAUST, a Flexible Architecture Unified System for Telecommunication, developed by the CEA-LETI to be demonstrated on 4-G telecommunication system [11].

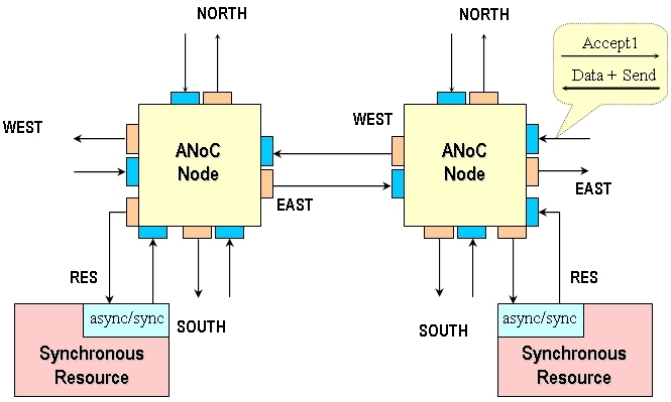

Fig. 2. Connections between two nodes.

B. Asynchronous Nodes

The nodes are the basic elements of the network and they usually have 5 bi-directional ports that connect to four neighboring nodes and the nearest synchronous computing resource via an asynchronous/synchronous network interface. A bi-directional port is driven by an input controller and an output controller, so that a node has 5 input controllers and 5 output controllers. Each input controller is connected to only 4 output controllers, going back and forth on the same network link is not allowed by the communication protocol. The design of these asynchronous network nodes and their operations are described in [4]. The interconnections between two nodes and their resources is described in Figure 2.

IV. An Innovative DfT Architecture for Asynchronous NoCs

A. Embedded core test and IEEE 1500 wrapper

To test the embedded cores in SoCs, a general architecture is first proposed in [12]. In this architecture, the embedded cores are covered by wrappers to improve the controllability and the observability of the embedded cores. Test vectors from the source are transported to the core-under-test and test results from the core-under-test are transported to the sink by the test access mechanisms (TAMs).

In order to make this architecture become reusable, a standard for embedded core test has been developed, the IEEE 1500 Standard for Embedded Core Test [13]. The objects of the IEEE 1500 Standard is to develop a core test wrapper architecture and a test language for the core-based SoCs. The TAMs are defined by the system chip integrators and then they may be serial, bus, or NoC architectures. The test stimuli and test results are shifted around the core by wrapper boundary register cells with a test clock. In our case, there is no global synchronization between networks elements so the IEEE 1500 wrapper is unsuitable. Therefore, we introduce in this paper an innovative DfT architecture for asynchronous NoCs, called ANoC-TEST, described in the following sections.

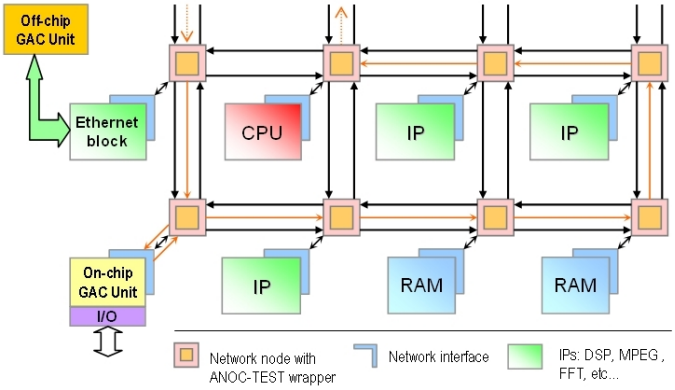

Fig. 3. ANoC-TEST general architecture.

B. ANoC-TEST architecture description

In this architecture, the network nodes are covered by the test wrappers, which are implemented in asynchronous circuits in order to better adapt to the GALS paradigms. To reduce TAM overhead we reused the links between the nodes to build a TAM with high throughput.

To generate test vectors, analyze the test results and control the test flows, a general-purpose unit is modeled, named GAC (Generator, Analyzer, and Controller) unit. This GAC unit may be implemented on-chip as an IP connected to a network node, or off-chip as a computer program that communicates with the ANoC via I/O ports or Ethernet ports of the ANoC. The ANoC-TEST general architecture is presented in Figure 3.

With this architecture, the test vectors generated by GAC block are transmitted to the node-under-test via the wire segments between nodes, loaded to the node-under-test by the test wrapper. On the other side, the test results are withdrawn from the node-under-test by the test wrapper, transmitted to the GAC block via the wire segments, and analyzed by an analyzer in the GAC block.

C. ANoC-TEST wrapper

Because the network node has 5 input/output ports, the test wrapper is composed of 5 input and 5 output wrapper boundary cells, see Figure 4. To control these input/output wrapper cells, a local test control module (TCM) is needed. The role of the test wrapper is to transport the test vectors to the node-under-test and get the test results from the node-under-test. To do this, the TCM has to control the input/output wrapper cells to do many operations such as updating test vectors (update new test vectors to the cell), shifting test vectors (shift the test vectors to the targeted cell corresponding to the targeted input port of the node-under-test), loading test vectors (load the test vectors to the node-under-test via the targeted input port), exporting test results (get the test results out of the node-under-test), shifting test results (shift the test results to the targeted cell corresponding to the target output of the wrapper), transmitting test results (transmit the test results to the TAM).

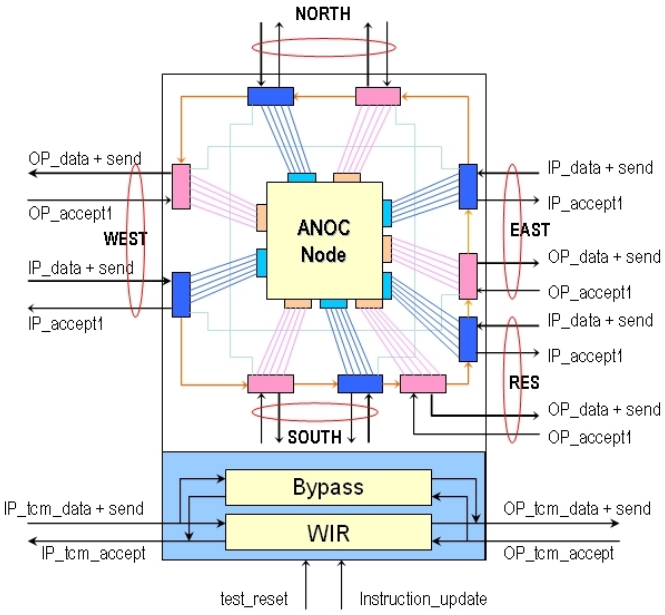

Fig. 4. ANoC-TEST wrapper.

To improve the quality of service (QoS), the CEA-LETI introduces in their asynchronous NoC k virtual channels with k levels of priority. All these levels of priority are arbitrated so that only one virtual channel is established at a time. Therefore, the test wrapper has to make no changes not only to data values, but also to their levels of priority. The design of the input/output ANoC-TEST wrapper boundary cells and the TCM module is described in [14].

D. GAC unit and test algorithm

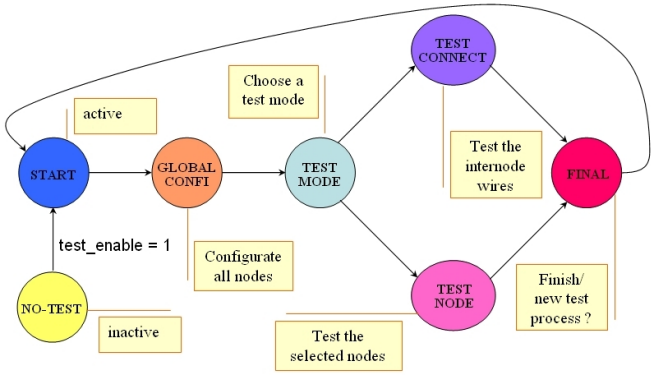

Fig. 5. The control flow of GAC unit.

The GAC (Generator, Analyzer and Controller) unit may be implemented on-chip as an IP, or may be implemented off-chip as a computer program. If the GAC unit is implemented on-chip, it may be modeled as a synchronous block and connected to the asynchronous communication network via an asynchronous/synchronous network interface. If the GAC unit is implemented as a computer program, this unit has to

be connected to the asynchronous network via the I/O ports or the Ethernet ports.

The role of the GAC unit is to generate test vectors for the asynchronous nodes and the computing resources, to configure the test wrappers via a serial asynchronous configuration channel, and to analyze the test results.

The control flow of the GAC unit is presented by a Finite State Machine (FSM) described in Figure 5.

V. Design and Verification

A. Flit-level Synchronization

The communication between the wrappers or between the wrappers and the nodes, as well as between the cells of the wrappers is established by a basic handshake protocol, the flit-level handshake protocol. This handshake protocol is defined as the "Send/Accept" protocol, in which the communication between two communication blocks is performed via the "Send" and "Accept" signals.

In the asynchronous NoC of the CEA, k virtual channels are implemented with k levels of priority in order to improve the QoS of the communication. To implement the flit-level handshake protocol for k virtual channels in the network, we need k "Send" and k "Accept" signals: $send < i >$ and $accept < i >$, where i gets values from 0 to $k-1$. The sender is allowed to send a new flit on virtual channel i with $send < i >= 1$, if and only if, the receiver indicated $accept < i >= 1$ at the previous cycle. With this "send/accept" protocol, flit transactions are realized in many virtual channels with an assurance of a free physical channel.

B. QDI asynchronous design

To design the ANoC-TEST wrappers, the Quasi Delay Insensitive (QDI) asynchronous design style [15] is used. We have used a 4-phase RTZ signaling protocol for asynchronous channels. In order to reduce power consumption, the 1-of-4 code signaling is used [16].

C. Asynchronous Design with SystemC/C++

All the asynchronous nodes as well as their test wrappers are implemented with the QDI style, which may be modeled in CHP language [15], Tangram [17], or Balsa [18]. In order to model the asynchronous logic with the same SystemC TLM framework we have used the basic SystemC "sc_fifo" primitives to model the asynchronous logic behavior. The sc_fifo primitive implements a blocking communication with associated $.read()$ and $.write()$ statements. Thanks to the class features of C++, we can model the different hardware types (such as multi-rail) easily. To model non-deterministic process like in CHP, a $.probe()$ function is added to the sc_fifo primitives.

Unfortunately, there are some drawbacks of designing in SystemC/C++. Firstly, in a communication channel the sc_fifo primitive has at least one place in the depth, where the depth of a communication channel in CHP may be zero. This makes the model more pipelined, but it is sufficient for behavioral simulations and validations. Secondly, we are able to model only the passive input & active output channels. Nevertheless, the active input & passive output channels are sometimes required. Finally, we can not see the visualization of data transfer with identical values in debug process. Refer to [4] for more detail about the modeling asynchronous logic in SystemC.

D. Verification platform

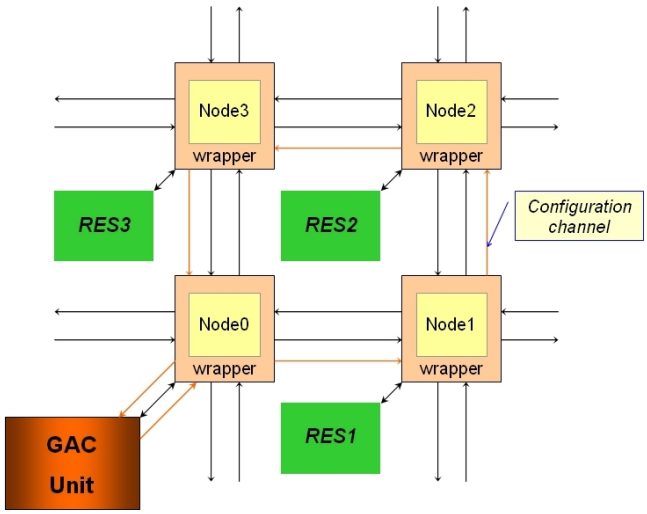

Fig. 6. A testbench with 4 asynchronous nodes and resources.

The proposed DfT architecture has been validated by a test-bench model with 4 nodes and resources as illustrated in Figure 6. All test processes of this model are controlled by a GAC unit that is modeled in SystemC/C++. The GAC unit also generates test vectors to the asynchronous nodes, the computing resources, and the internode wires. The test results are received and stored in a test result file and then are compared with the test vectors to analyze the defects of the circuits. In addition, the test architecture can test itself by comparison the configuration vectors that have been transmitted to and received from the configuration channel by the GAC unit.

VI. Results and Conclusion

The proposed DfT architecture is an innovative, generic, scalable, flexible, and configurable architecture so that it can be expanded to meet the extension of Asynchronous NoC-based SoCs. It can be modified to test the nodes in parallel. Because the nodes are identical, the test vectors can be reused and the test results can be compared to each other, it is especially useful in parallel test case.

This DfT architecture is used to test not only the network nodes and internode wires, but also the computing resources (IPs) and the asynchronous/synchronous network interfaces.

In this design, we reuse the internode wires as high bandwidth test access mechanisms (TAMs). This avoid the

congestion in the layout process. With a cycle time of 4ns and 32-bit width, a test path has a throughput of 1Gbytes/s.

The ANoC-TEST wrapper is modeled and validated in the SystemC/C++, described in Section V, that corresponds to the behaviors of asynchronous circuits. The surface cost of this wrapper is evaluated at about 8 Kgates, in report of 4.5 Mgates of our target asynchronous NoC with 20 nodes, 23 IPs (without memories).

REFERENCES

[1] J. Bainbridge and S. Furber. CHAIN: a Delay-Insensitive Chip Area Interconnect. *IEEE Micro*, Vol. 22(5):16–23, Sept.-Oct. 2002.

[2] A. Lines. Asynchronous Interconnect for Synchronous SoC Design. *IEEE Micro*, Vol. 24(1):32–41, Jan.-Feb. 2004.

[3] A. Efthymiou, J. Bainbridge, and D.A. Edwards. Adding Testability to an Asynchronous Interconnect for GALS SoC. In *Proc. of the 13th Asian Test Symposium*, Taiwan, Nov. 2004.

[4] E. Beigné, F. Clermidy, P. Vivet, A. Clouard, and M. Renaudin. An Asynchronous NoC Architecture Providing Low Latency Service and Its Multi-Level Design Framework. In *Proc. of the 11th International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pp. 54–63, New York, USA, Mars 2005.

[5] W. J. Dally and B. Towless. Route Packets, not Wires: On-Chip Interconnection Networks. In *Proc. of the Design Automation Conference (DAC)*, pp. 684–689, Las Vegas, NV, USA, June 2001.

[6] L. Benini and G. De Micheli. Networks on Chip: a New SoC Paradigm. *IEEE Computer*, Vol. 1, pp. 70–78, Jan. 2002.

[7] A. Jantsch and H. Tenhunen. *Networks on Chip*. Kluwer Academic Publisher, Feb. 2003.

[8] B. Vermeulent, J. Dielissen, K. Goossens, and C. Ciordas. Bringing Communication Networks on a Chip: Test and Verification Implications. *IEEE Communication Magazine*, pp. 74–81, Sep. 2003.

[9] É. Cota, L. Carro, F. Wagner, and M. Lubaszewski. Reusing an On-Chip Network for the Test of Core-Based Systems. *ACM Trans. on Design Automation of Electronic Systems*, Vol. 9(4):471–499, Otc. 2004.

[10] M. Nahvi and A. Ivanov. Indirect Test Architecture for SoC Testing. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 23(7):1128–1142, July 2004.

[11] Y. Durand, C. Bernard, and D. Lattard. FAUST: On-Chip Distributed Architecture for a 4G Baseband Modem SoC. In *Proc. of the IP based SoC Design Conference (IP-SoC)*, Grenoble, France, Dec. 2005.

[12] Y. Zorian. Testing Embedded-Core based System Chips. In *Proc. of the International Test Conference (ITC)*, pp. 130–140, Washington, DC, USA, Oct. 1998.

[13] IEEE Std. 1500, IEEE 1500 Standard for Embedded Core Test. *http://grouper.ieee.org/groups/1500/*.

[14] X-T. Tran, C. Robach, J. Durupt, V. Beroulle, and F. Bertrand. A DfT Architecture for Asynchronous Networks-on-Chip. In *the 11th European Test Symposium (ETS'06)*, Southampton, UK, May 2006 *(accepted)*.

[15] M. Renaudin, P. Vivet, and F. Robin. ASPRO-216: a Standard Cell QDI 16-bit RISC Asynchronous Microprocessor. In *Proc. of the 4th International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC)*, pp. 22–31, San Diego, USA, April 1998.

[16] J. Bainbridge, W. Toms, D. Edwards, and S. Furber. Delay-Insensitive, Point-to-Point Interconnect using m-of-n Codes. In *Proc. of the 9th International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pp. 132–140, Vancouver, BC, Canada, May 2003.

[17] K. van Berkel, J. Kessels, M. Ronken, R. Saeijs, and F. Chalij. The VLSI Programming Language Tangram and its Translation into Handshakes Circuits. In *Proc. of the European Design Automation Conference*, pp. 384–389, Amsterdam, Feb 1991.

[18] A. Bardsley, and D. Edwards. Compiling the Language Balsa to Delay-Insensitive Hardware. *Hardware Description Languages and their Applications (CHDL)*, pp. 89–91, April 1997.