Access Network Generation for Efficient Debugging of FPGAs

Habib ul Hasan Khan[1], Tomás Grimm[2], Michael Hübner[2], Diana Göhringer[3]

[1]Application-Specific Multi-Core Architectures Group, Ruhr-University Bochum, Germany

[2]Chair of Embedded Systems for Information Technology, Ruhr-University Bochum, Germany

[3]Chair for Adaptive Dynamic Systems, Technische Universität Dresden, Germany

Email: {habib.khan, tomas.grimm, michael.huebner}@rub.de, diana.goehringer@tu-dresden.de

ABSTRACT

The inclusion of access networks in modern FPGAs can provide a large number of use cases notably in debugging. Using access networks can eliminate the need for frequent synthesis during the debugging phase, which results in saving debugging time and reducing the time to market. Using supervisory control by a processor, required networks can be configured just by minor software modification. Furthermore, connecting thousands of nodes to the debugging system is also a complicated issue. Utilizing IP-XACT files for automatic network generation can solve such problems. A Tcl file can then be used which can perform automatic network generation. This paper demonstrates an access network design, which requires only small resources and hence is suitable for large designs along with a framework for automatic connectivity generation.

1. INTRODUCTION

The growing complexity of current embedded systems is making the debugging process more difficult and this affects the time-to-market badly. A study [1] shows that design complexity doubles every 18 months in contrast to the design productivity, which doubles every 39 months. This difference is because verification of complex designs requires extra time.

Normally system verification is carried out by virtual prototyping but when the design is complex, virtual prototyping may be affected due to speed issues. As claimed in the IBM report [2], HDL simulation reached only 10 Hz in contrast to implementations having target frequency of 1.6 GHz.

Because of speed limitations of verification by software based simulation methods, hardware simulators came into focus. However, due to hardware invisibility, monitoring of only those signals is possible which are available at FPGA pins. In order to enhance the visibility, Integrated Logic Analyzers (ILA) cores [3] were introduced. These cores are embedded into the design during the synthesis process and then can be used for logging of debugging data inside trace buffers [4]. However, since such solutions utilize scarce FPGA resources, only limited number of signals can be monitored. This limitation necessitates identifying the signal set required to be monitored during design time.

However during design verification phase, the portion of the circuit required to be debugged may change over time, and in any case, is not likely predictable during design time. If more signals are required to be monitored then re-synthesis is required. However, for large designs, the re-synthesis consumes a lot of time, which in turn affects the time to market.

A flexible network solution may allow the verifier to select the signals of interest, and a suitable debugging system can process these signals, which can then be processed as per debugging requirements. In order to connect a large number of signals without the need to re-synthesize, a class of flexible networks called access networks [5] were suggested. These networks allow re-configurability of the network without the need to resynthesize the design indefinitely just for the sake of debugging. However, these networks are dependent upon hardware resources. Hence, an efficient access network is required, which can provide efficient connectivity, with minor impact on the architecture overhead. Many large designs can have tens of thousands of debugging nodes. Connecting these nodes to such access networks is cumbersome. Hence, an efficient access network providing automatic connection generation utility is required. The ability to easily debug complex embedded system designs in this way provides a number of merits:

1. Reduced time-to-market because of an increased ability to isolate problems.
2. Less resource requirements for post-manufacturing verification because of enhanced functional coverage.
3. Fewer design revisions.
4. Increased customer satisfaction because of enhanced ability to access a higher number of debugging nodes.
5. The ability to connect automatically the DUT (design under test) to the access network reduces the human errors in connection making.

A DSAS approach was introduced in [6], which starts the Device under Test (DUT) and once the trace buffer is filled, it stops the DUT, saves the data to the external memory

through Ethernet and starts the DUT once again. Access network introduced in this paper can be used along with such system, which can increase the efficiency of the system greatly.
The rest of the paper is organized as follows. Section 2 presents related work and provides background information. Section 3 describes the FPGA debugging by a DSAS approach. In Section 4, the results are discussed. The paper is concluded in Section 5.

2. RELATED WORK

Commercial tools like Xilinx Chipscope and Altera SignalTap II use the trace buffer methodology. One draw-back in the commercially available tools is that the signal set for the trace buffer block has to be identified during design time. Therefore, any changes require re-synthesis of the trace buffer block. This approach results in an in-creased design time for the system before the task can be debugged [7]. In order to address this problem, a method of incremental synthesis has been proposed [8]. Nevertheless, recompiling would still be required, which is again time-consuming.
The idea of using a network for SoC communication is not new [9]. Such networks are mainly used for communication between a CPU and connected peripherals, using packet-based networks. Using a packet-based network provides a very good solution for SoC communication. However, the use of such networks is not suitable for synchronous and high bandwidth communications. Instead, indirect, non-blocking networks such as those used for traditional telephone networks, are more suitable for applications requiring synchronous high bandwidth communication [11]. Furthermore, such network does not require to be configured in real-time. It can be configured once for each application and does not need to address incremental connection requests. Hence, the network only needs to be re-arrangeably non-blocking [10].
Research has already been carried out on non-blocking multistage networks that permit connections from any in-put to any output of the network. Such networks are often called permutation networks because they offer all possible permutations of the inputs on the output side of the net-work. The Clos network can be used to construct a re-arrangeable non-blocking permutation network which has low cost (area) [10]. The Benes network can be used to reduce the network cost further by recursively replacing the middle stage of a Clos network with another Clos network [5]. However, these permutation networks may be more flexible than required. In some cases, it is not required to connect every network input to every network output. Instead, it is only required to have the ability to connect any inputs to any outputs. This flexibility can be used to lower the cost of the access network.
A great deal of research has been carried out on unordered networks [12, 13]. A class of unordered network called concentrators can be used to connect a number of inputs to a smaller number of outputs and hence provides a good mean as an interface. However, while few concentrator constructions have lower depth in comparison to permutation networks, they do not necessarily have a lower cost than permutation networks for all possible configurations. Hence the selection of suitable network architectures becomes difficult because the network size (numbers of inputs and outputs), and its impact on cost and depth must be deliberated before deciding an architecture [5].

The main purpose of the access network is to connect nodes to the debugging system efficiently. This can be done manually but requires a lot of effort and still the procedure is error prone. In [14] a script tool was used to extract the design nodes from HDL simulators. The IP-XACT standard [15] completely describes the interfaces for all elements from an architecture, where four schemata capture their main characteristics: design, component, bus and abstraction definitions. The design description captures all the components that are inside the architecture and the interconnections between them and the external pins. The component description captures the bus interfaces and each of their ports, channels, address maps, reset and clock signal configurations that are part of each component. The bus and the abstraction definitions are complimentary descriptions for a communication bus. The first one captures the high-level characterization of a bus, including connection method and addressing. The latter, on the other hand, describes the low-level characterization, including the name, the width and the direction of each port. The interface definitions from this standard can be used to generate the connectivity between the access network and the debugging system, which may help in reducing the human intervention required for manual connection between the access network and the debugging system.
In this paper, we will present two different access network methodologies i.e. gate-based and multiplexer-based. The access network can then be automatically connected to a DUT (or any design) utilizing a connectivity generation tool flow presented in this paper. The technique not only utilizes minimum resources but also provides access to all the available debugging nodes without the need to re-synthesize the design again.

3. METHOD DESCRIPTION

The research work has been divided into two portions, i.e. the access network and the connectivity generation.

3.1 Access Network

During the design phase, the SoC designer does not know how the design will behave during verification phase. Thus, it is impossible to identify a set of signals, which are error prone so that they can be connected to the debugging system. Instead, the designer may select a much larger set of signals from the DUT and connect them to the debugging system. However, connecting a very large set of nodes to the debugging system requires a large set of resources. Furthermore, the number of signals which can be handled simultaneously need to be limited for effective resource utilization. This task can be accomplished by a processor configurable access network as shown in Figure 1.

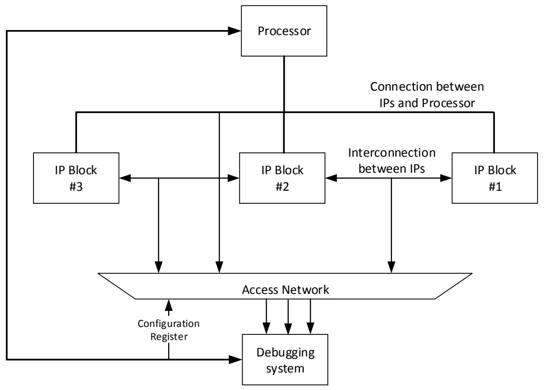

Fig. 1: Access network interconnection applied to a DUT

The access network can be created in several ways such as multiplexer network, permutation network or concentration network [5]. As claimed by [16], concentration network provides the best performance with the least resources. However, this has not been proven since the difference between the multiplexer and concentration network has been found to be unnoticeable [14]. In our research work, we have used two different approaches for access network generation namely the gate-based approach and the multiplexer-based approach. Figure 2 shows part of gate-based access network.

For this research work, the number of the extracted signal set (see Figure 2) is 16. However, it is worth mentioning that the number of extracted signals can be increased beyond 16. The upper bound on the number of signals has been defined because in the case of too many extracted signals the debugging may become cumbersome for the DSAS approach [6].

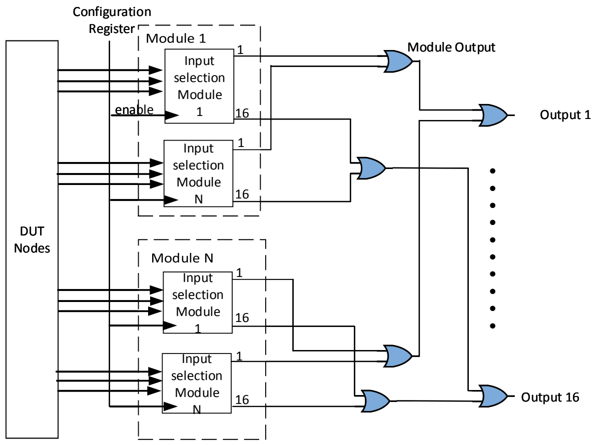

Fig. 2: Gate-based access network

Furthermore, it is extremely simple to exchange the signal set through the configuration register, which configures input selection and allows full visibility of all other network signals. The number of controllable signals can range from hundreds to thousands, depending on design complexity and hardware limitations. A special type of Input selection module (m x m) has been used and its enable port can be used to enable or disable the selector. When an input selection module is enabled, all the signals at its input are connected to the OR gates. Each OR gate has as many inputs as the number of selectors in the module. Only the output of the enabled selection module has a valid value. The output of all the other selection modules is zero at that specific time. A class of network called a hyper-concentrator can also be used instead of input selection module. A hyper-concentrator is closely related to a concentrator. It is a network with x inputs and x outputs such that any set of inputs can be mapped to any contiguous set of outputs without regard for the ordering of these outputs. Narasimha [17] has proposed a hyper-concentrator construction, which is claimed to be low cost in terms of switches and has moderately low network depth. The outputs of each hyper-concentrator are registered to ensure that the network can run at the speed of the integrated circuit. However, we have used the self-designed selection module to avoid undue complexity.

We also have developed an access network by using the multiplexer based approach. This type of network has n inputs and m outputs, where n is always greater than m. The proposed network has the ability that any subset of inputs can also be selected which is missing in the first approach. The n inputs are partitioned into k groups, where k is the number of input election modules. We further assume that the size of each of these k groups of inputs is ≤ m, the total number of outputs of the access network.

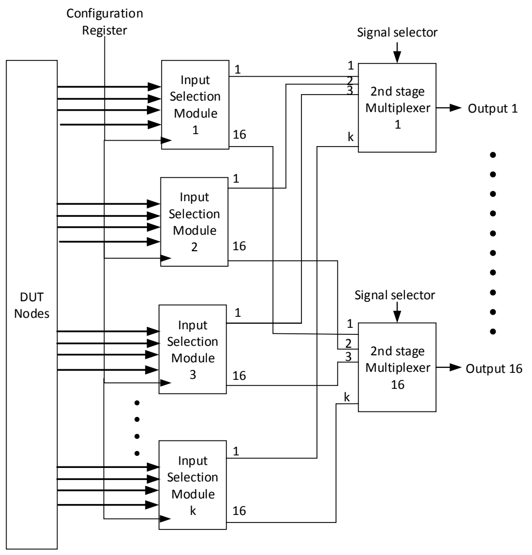

Fig. 3: Multiplexer-based access network

In the first stage of the network, each of the k groups is considered independently. A special type of input selection module (m × m) has been used and its control port can be used to select the required signals at the output. As already explained, a hyper-concentrator can also be used instead, but it does not retain the ordering of outputs. Hence, we have used a self-designed input selection module which retains the ordering of the output.
The second stage of the network combines the outputs of the k input selection modules with a simple multiplexed bus structure, as shown in Figure 3. The network can have multiple cascaded stages of multiplexers in order to expand the network as required. This implementation is advantageous as compared to the gate-based solution because it allows the selection of any of the input signal to the multiplexer stage. Configuration registers can be used to control the multistage multiplexers. As a result, any combination of the signals can be selected at the output.
The access network can then be controlled by a processor, which can issue appropriate control words for configuration registers. Hence, by only changing the software running on the processor, full visibility of the DUT can be achieved. This approach can eliminate the need to re-synthesize the design during the debugging process.
As an alternative to processor control, JTAG [18] can also be used for the configuration of the access network. However, for our current work, we have used the processor for configuring the access network.

3.2 Connectivity Generators

The connectivity-generation tool flow extends the work from [19], adding a step to automatically generate the connectivity between one or more DUTs to an access network. The tool benefits from the IP-XACT descriptions, as they capture the interfaces from all the components in a design, as well as the design itself and the connections between the components.
The IP-XACT [20] [21] standard is an XML format aimed at design integration and reuse in different tools and development flows. The Spirit Consortium created it to facilitate the exchange of IPs between different vendors and their integration in different electronic design automation (EDA) tools. Now Accellera 6 regulates it as the IEEE 1685-2014 standard.
This standard completely describes the interfaces for all elements from an architecture, where four schemata capture their main characteristics: design, component, bus and abstraction definitions.
The bus and the abstraction definitions are complimentary descriptions for a communication bus. The first one captures the high-level characterization of a bus, including connection method and addressing. The latter, on the other hand, describes the low-level characterization, including the name, the width and the direction of each port.

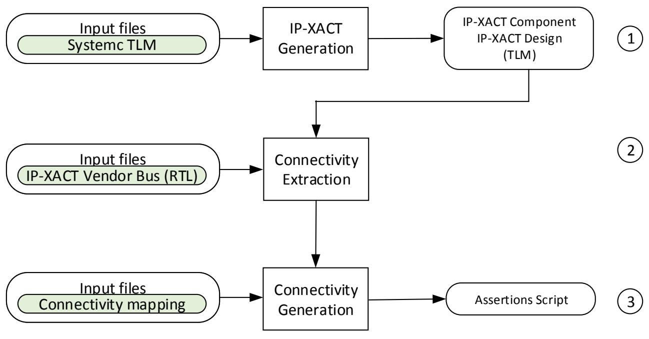

Fig. 4: Flow of the developed tool

An IP-XACT description captures parameter choices for configurable aspects of these IPs. It is then easier to add such IPs to different systems using different vendor's development tools. However, after generation, the description indicates which the selected choice is. Therefore, this tool generates checkers for the selected choice.
The starting point for the connectivity generation is the declaration of the interconnections and the ad-hoc connections in the design schema. Interconnections are protocol channels or signals that are part of the same set. Ad-hoc connections are either connections from external pins (input and output) to component ports, e.g. reset and clock signals, or a component port that feeds its signal to other components and is not part of a protocol, e.g. a clock divider.
The connectivity-generation tool flow has three steps, as illustrated in Figure 4. The generated files help to better automate the RTL development and to maintain the connectivity correct. The description of each step follows in the proceeding sections.

3.2.1 IP-XACT descriptions generation

The first step of the proposed method, illustrated in part 1 from Figure 4, generates the IP-XACT descriptions from SystemC models. This is the main starting point of the tool. The inputs are the model files and the outputs are the IPXACT descriptions for each component in the design and for the design itself.
For a description of the translation of SystemC to IP-XACT, please refer to [22], which describes the tool SCiPX.

3.2.2 Connectivity extraction

In the second step of the proposed method, illustrated in part 2 from Figure 4, the inputs are the generated files together with vendor provided IP-XACT descriptions for the communication protocols, which are divided into abstraction definitions and bus definitions. This lowers the complexity of the previous step.
This step can also be an entry point, when the internal components of the architecture are done and only the connectivity from the DUT to the access network is missing.

This step extracts the connectivity information present in the IP-XACT descriptions and passes it to the next step. The procedure analyzes the design description, which lists all connections. For each interconnection, it is necessary to find the correct abstraction and bus descriptions, which defines the signals. However, for each ad-hoc connection, there are two possibilities. It is necessary to find if there is an input pin feeding internal pins, or if there is a component's output pin feeding other pins.

3.2.3 Connectivity generation

In the third step of the proposed method, illustrated in part 3 from Figure 4, the tool receives as input a mapping of the DUT outputs to be connected to the access network.

Using all the information parsed from the IP-XACT descriptions in the second step together with the port mapping, the tool generates the connection command for each signal listed in the mapping.

The output of this step is a Tcl script containing the commands needed to automatically connect the selected DUT signals to the access network. The access network can then be controlled by a processor to provide desired connectivity for debugging or other requirements.

4. RESULTS

The proposed approach has been tested on a Zedboard. The resource utilization has been carried out for both access network approaches. The resources utilization for the gate based approach is shown in Figure 5. The percentage of resource utilization is plotted in Figure 6.

It is evident that the resources required for this approach are much less. Even with 4096 inputs, the utilization of flip-flops and LUTs is 4% and 2.5% respectively. The utilization for an access network with fewer inputs is even less. Hence, in cases where signal selection within the group is not required, this approach can be employed for its minor resource utilization. However, one drawback of this approach is that it does not allow input selection within the group.

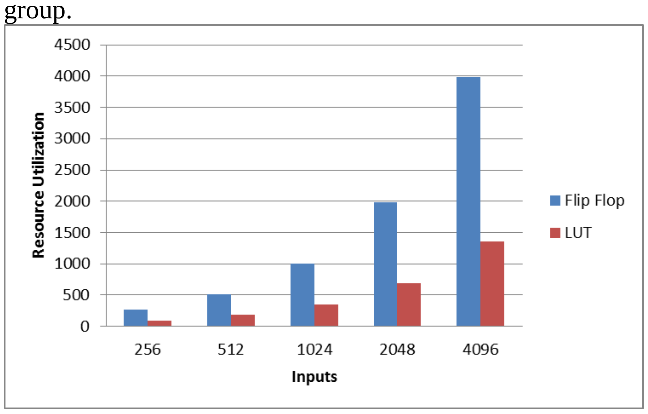

Fig. 5: Resource utilization for gate-based approach

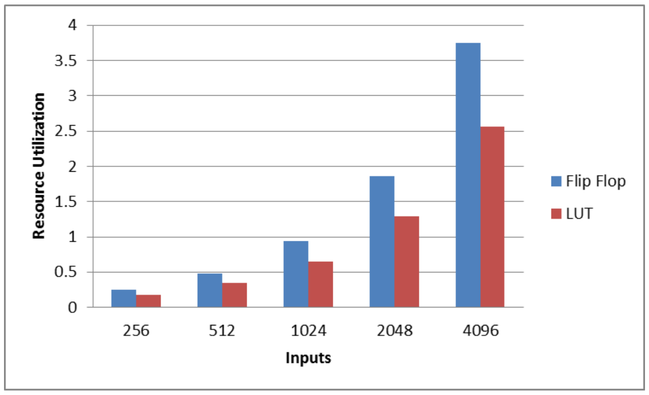

Fig. 6: Percentage of utilized resources on a Zedboard for the gate-based approach

Resource utilization has also been calculated for the Multiplexer-based access network as shown in Figure 7. Percentage of utilized resources for this approach is given in Figure 8. As can be seen in Figure 8, resource utilization is quite moderate. 12% of the resources have been consumed for an access network width of 4096. For less number of inputs, the resource utilization is even less. The main benefit of this approach is that any subset of signals can be selected. In cases where availability of resources is not an issue, this approach is recommended for its flexibility.

It is evident from the results that the two approaches i.e. the gate-based approach and the multiplexer-based approach have their merits and demerits. Based upon the requirement and the available resources, a suitable approach can be selected.

The presented connectivity generation tool flow can be used to generate the automatic connectivity between the DUT and access network. The resulting design can subsequently be controlled by a processor to generate the desired connectivity.

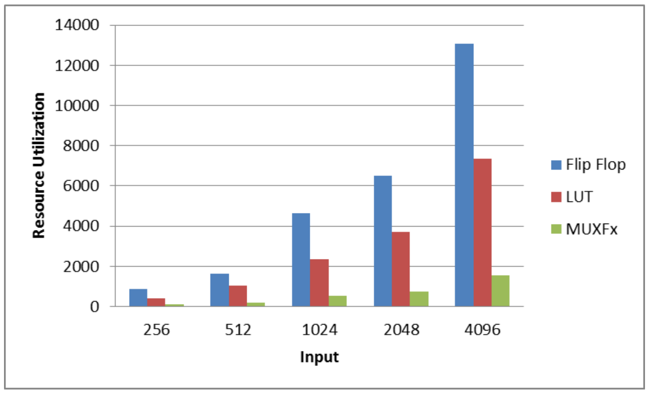

Fig. 7: Resource utilization for the multiplexer-based approach

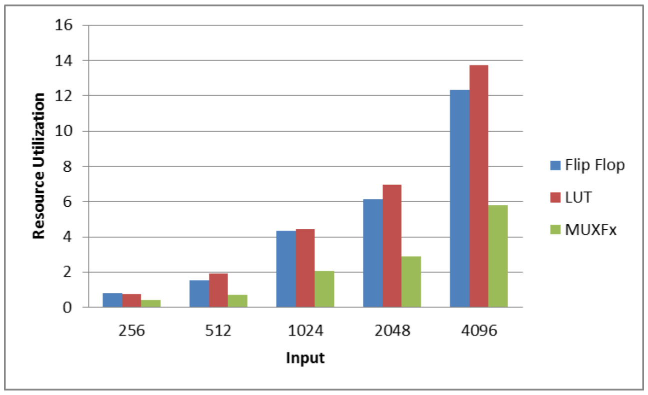

Fig. 8: Percentage of resource utilization on a Zedboard for the multiplexer-based approach

5. CONCLUSION

The paper presents two different techniques for access networks. The resource utilization for the two techniques has also been calculated. It was shown that the gate-based access network approach uses fewer resources but is less flexible and the multiplexer-based access network has a small resource overhead but offers additional flexibility.

Access networks can be used in a variety of applications including debugging. When used in debugging systems, the number of inputs to the debugging systems can be greatly increased. This will eliminate the need to re-synthesize the design and help in achieving faster debugging.

6. REFERENCES

[1] E. Hung and S.J.E. Wilton. "Towards simulator-like observability for FPGAs: a virtual overlay network for trace-buffers." In Proc of the ACM/SIGDA international symposium on Field programmable gate arrays. ACM, 2013.

[2] Asaad, S., Bellofatto, R., Brezzo, B., Haymes, C., Kapur, M., Parker, B., Roewer, T., Saha, P., Takken, T. and Tierno, J., 2012, February. A cycle-accurate, cycle-reproducible multi-FPGA system for accelerating multi-core processor simulation. In Proc. of the ACM/SIGDA international symposium on Field Programmable Gate Arrays. ACM, 2012.

[3] A. Herrmann and G. Nugent. "Embedded logic analyzer for a programmable logic device." U.S. Patent No. 6,389,558. 14 May 2002.

[4] H. Kuijsten. "Method and apparatus for a trace buffer in an emulation system." U.S. Patent No. 5,680,583. 21 Oct. 1997.

[5] Quinton, Bradley R., and Steven JE Wilton. "Concentrator access networks for programmable logic cores on SoCs." In Proc. of the Int. Symposium on Circuits and Systems (ISCAS), 2005.

[6] H.H. Khan and D. Göhringer. "FPGA Debugging by a Device Start and Stop Approach". In Proc. of the Int. Conference on ReConFigurable Computing and FPGAs (ReConFig). IEEE, 2016.

[7] T. Wheeler, P. Graham, B. Nelson, and B. Hutchings. "Using design-level scan to improve FPGA design bservability and controllability for functional verification." In Proc. of the Int. Conference on Field Programmable Logic and Applications (FPL). IEEE, 2001.

[8] E. Hung and S.J.E. Wilton. "Limitations of incremental signal-tracing for FPGA debug." In Proc. of 22nd Int. Conference on Field Programmable Logic and Applications (FPL). IEEE, 2012.

[9] Raghunathan, Vijay, Mani B. Srivastava, and Rajesh K. Gupta. "A survey of techniques for energy efficient on-chip communication." In Proc. of the 40th annual Design Automation Conference. ACM, 2003.

[10] Hwang, Frank. "The mathematical theory of nonblocking switching networks". Vol. 15. World Scientific, 2004.

[11] Beneš, Václav E., ed. Mathematical theory of connecting networks and telephone traffic. Vol. 17. Academic press, 1965.

[12] Pinsker, Mark S. "On the complexity of a concentrator." 7th International Telegraphic Conference. Vol. 4. 1973.

[13] Chung, F. R. K. "On concentrators, superconcentrators, generalizers, and nonblocking networks." The Bell System Technical Journal 58.8 (1979): 1765-1777.

[14] Panjkov, Z., Wasserbauer, A., Ostermann, T., & Hagelauer, R. "Hybrid FPGA debug approach". In Proc. of the Int. Conference on Field Programmable Logic and Applications (FPL). IEEE, 2015.

[15] IP-XACT Recommended Vendor Extensions to IEEE 1685-2014 (IP-XACT)

[16] B. R. Quinton, S. J. E. Wilton. "Post-Silicon Debug Using Programmable Logic Cores". Field-Programmable Technology, 45-48, 2005

[17] Narasimha, Madihally J. "A recursive concentrator structure with applications to self-routing switching networks." IEEE transactions on communications 42.234, (1994).

[18] Battaline, Robert P., et al. "Performance monitoring through JTAG 1149.1 interface." U.S. Patent No. 5,768,152. 16 Jun. 1998.

[19] Grimm, T., Lettnin, D., & Hübner, M. "Automatic generation of RTL connectivity checkers from SystemC TLM and IP-XACT descriptions". In Proc. of Nordic Circuits and Systems Conference (NORCAS). IEEE, 2016.

[20] V. Berman, "Standards: The P1685 IP-XACT IP Metadata Standard," IEEE Design & Test of Computers, vol. 23, no. 4, pp. 316–317, 2006.

[21] Standard Structure for Packaging, Integrating, and Reusing IP within Tool Flows, Std. IEEE Std. 1685-2009, 2010.

[22] J.-F. Le Tallec, et al., "Combining SystemC, IP-XACT and UML/MARTE in model-based SoC design". Workshop on Model Based Engineering for Embedded Systems Design (M-BED 2011), Grenoble, France, 2011, pp. 1–6.