John von Neumann Institute for Computing


# Versatile FPGA-Based Functional Validation Framework for Networks-on-Chip Interconnections Designs

J.B. Perez Ramas, D. Atienza, M. Peón, I. Magán,
J.M. Mendías, R. Hermida

published in

*Parallel Computing:*
*Current & Future Issues of High-End Computing*,
Proceedings of the International Conference ParCo 2005,
G.R. Joubert, W.E. Nagel, F.J. Peters, O. Plata, P. Tirado, E. Zapata
( Editors),
John von Neumann Institute for Computing, Jülich,
NIC Series, Vol. **33**, ISBN 3-00-017352-8, pp. 769-776, 2006.



http://www.fz-juelich.de/nic-series/volume33

# Versatile FPGA-Based Functional Validation Framework for Networks-on-Chip Interconnections Designs

Javier B. Perez Ramas[⋆†], David Atienza[⋆‡], Miguel Peon[⋆], Ivan Magan[⋆], Jose M. Mendias[⋆], Roman Hermida[⋆]
[⋆]DACYA/UCM, C/ Prof. Jose Garcia Santesmases s/n, 28040 Madrid, Spain. {datienza, mendias, rhermida}@dacya.ucm.es ; {mikepeon}@fdi.ucm.es
[†]Working for INDRA Sistemas, S.A. Avda. de Bruselas 35, 28108 Alcobendas, Madrid, Spain. jbpramas@indra.es,
[‡]LSI/EPFL, EPFL-IC-ISIM-LSI Station 14, 1015 Lausanne, Switzerland. david.atienza@epfl.ch

Forthcoming System-On-Chip (SoC) devices will be made of hundreds of cores to be able to handle the complex functionality required in new consumer devices. In the last years, Network-On-Chip (NoC) has been proposed as a promising replacement for buses and dedicated interconnections to solve the scalability and complexity problems. As a result, NoCs imply a complex design process to define suitable network interfaces to access the on-chip network, the selection of convenient protocols and also topologies of switches to transport the data. Presently, several options for NoC topologies and interfaces have already been proposed at different levels of abstraction, but extensive testing is still required to functionally validate them at the physical level in environments with a variable number of processing cores and real applications. In this paper we illustrate the use of reconfigurable hardware (FPGA) to help bridging the gap between NoC design and validation. The defined reconfigurable system has enough memory capabilities to run complex SoC applications and can use any number of processing cores. We assess its effectiveness for NoC-based exploration with two case studies: two different applications running in the same SoC, which is connected using a standard NoC and a commercial bus solution.

## 1. Introduction

Recent research claims that the interconnection model inside a chip, traditionally based on a bus, can become a bottleneck in the near future due to hundreds of IPs with many simultaneous communications between them [1,6] causing permanent conflicts in the shared medium. More complex arbitration protocols, aggressive hierarchization and segmentation of the buses could be necessary to minimize those effects. Physical implementation is difficult because of the huge size of the shared structures connected to an equally high number of end-points. At system level, buses have been displaced by packet-based interconnection networks (standards as RapidIO, VME-X or PCI Express replace the traditional buses like VME64 or PCI gradually). The advantages of this option have already been studied: speed, scalability, reliability, resistance to the congestion and many others outlined by its defenders [12]. As an extrapolation of the previous idea within the scope of Systems-on-Chip (SoCs), a technology baptized as Network-on-Chip (NoC) has been recently proposed. A NoC implements a commutation network inside the chip in order to communicate the different Intellectual Property blocks (IPs from now on). The objectives are multiple, such as, increasing the efficiency of the IP communications, allowing multiple communications at the same time and, in general, solving the bus bottleneck [1,12].

Diverse NoC architectures have appeared and numerous theoretical studies indicate the advantages of this option. Although attempts of real tests for these systems exist (see Section 2), there have not been so far complete real-life tests that compare designs of real and customizable SoCs with standard

applications, and where different interconnection alternatives are utilized. Obtaining exact measures of real life applications with the different alternatives must be possible. In addition, the applications should be as much as possible interconnection-network independent, so that to prove the advantages of a new option it is not necessary to rewrite the function calls or mechanisms to interact with the intercommunication layer. Nevertheless, the eventual software to tune to exploit all the benefits of such intercommunication (e.g. message passing vs shared memory). Our platform tries to offer all of these characteristics.

The rest of this paper is organised as follows. First of all, in Section 2 a summary of relevant research in NoC and bus-based analysis and prototyping is presented. Then, in Section 3 we detail the architecture of our proposed Multi-Processor SoC (MPSoC) platform. Next, in Section 4 we describe two study cases, one for buses and another one for a NoC-based design and indicate the synthesis results obtained. Later, in Section 5 we describe two different real-life parallel applications running in our MPSoC platform and the empirical results obtained. Finally, in Section 6 we present our conclusions and possible future research lines.

## 2. Related Work

Recent studies propose that the bottleneck present in the traditional bus interconnection system can be solved for the SoC domain using the Network-on-Chip approach so a significant effort has been made to provide functional NoC architectures. [14] presents Xpipes, a NoC development framework highly parametrizable and flexible enough for investigatation due to the possibility of experimenting with customizable network topologies. Also, conversely to other precedents like Aethereal [15] or Proteo [10], Xpipes is the first technology presented with a compiler of NoC descriptions that allows to adapt the topologies and characteristics of the NoC for each application. Xpipes provides a complete and flexible development system of NoC with a library of personalizable components, such as, Open Core Protocol (OCP) [9] Network Interfaces (NIs), switches and links, and a compiler: XpipesCompiler [5]. The input is a file with the topological and parametric description of the desired NoC and the output is a SystemC file containing the implementation. Also, in [8] the interesting Hermes framework is explained. It is a NoC with a simple switch, which is economic in area, and with implicit routing in the destination address.

In addition, previous work using FPGA for NoC-based research exists. In [7] and [2], NoCs with a mesh-based topology and packet-switching as communication mechanism have shown the effectiveness of NoC. Also, in [4] a Xilinx Virtex 2 Pro FPGA is used to explore the performance of NoC solutions using Xpipes. However, it is limited to synthetic applications through traffic generators or previously generated traces. Finally, several NoC architectures (e.g. torus) and designs of switches/routers have been ported to FPGAs to validate their features (e.g. packet sizes, switching-mode) decided by additional HDL simulations [7,11,13,3]. None of this works enables a complete flexibility for testing interconnection mechanisms with real-applications as we propose in this paper with our FPGA-based MPSoC platform.

## 3. Proposed MPSoC Platform

The main characteristics of our generic MPSoC platform include a variable number of general purpose CPUs, standard connection that allow attaching any IP, the possibility to define or use any interconnection topology and the visibility of all the architectonic issues of the system in order to evaluate the interconnection, either in simulation or during real-time execution. The platform has been developed using a Virtex 2 PRO 20 FPGA from Xilinx [16], a suitable environment because it

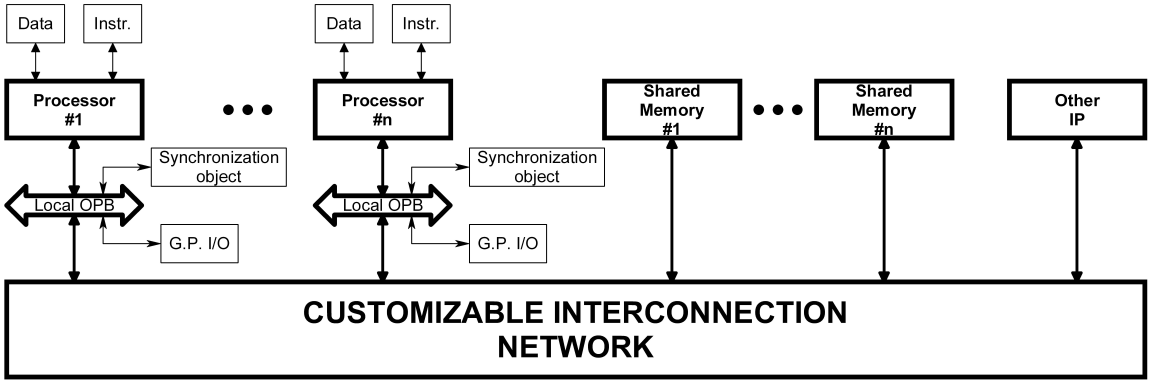



Figure 1. Generic architecture

provides a very high-density reconfigurable hardware platform, different processing cores (i.e. hard-wired PowerPCs and soft-core MicroBlazes), SRAM memories and various interconnection buses, such as, On-chip Pheripheral Bus (OPB) or Processor Local Bus (PLB).

The generic architecture of the system mapped onto the FPGA is shown in Fig 1. As it shows, its main components are the following ones:

- Processors: The general-purpose processors used are Xilinx MicroBlaze CPUs [17], a synthesizable VHDL 32-bit CPU based on a Harvard architecture. MicroBlaze uses local memory buses for instruction and data, and standard IBM OPB-bus external connection. Each CPU has its own private data and instruction memory.

- Interconnection network: The interconnection is the element under test and enables data transmission between the different IPs. Its choice is configurable to NoC or buses and affects the system behaviour. Moreover, the same NoC technology can be configured for different topologies, packet routing, etc. In all cases the network must provide an OPB interface to be connected to the platform.

- Shared memories: Internal FPGA memory is currently used, although external memories (e.g. SDRAM, DDR, etc.) can be easily added via specific controllers.

- Other IP cores: Any module (generic I/O, synchronization device, etc.) that uses the standard OPB interconnection or the OCP interface can be directly added to the SoC and connected to the interconnection used.

### 3.1. HW/SW MPSoC Toolflow

One of the most important elements in our MPSoC platform is the inclusion of the complete HW and SW flows in one overall toolflow. An overview of this working flow is presented in Figure 2. On the one hand, regarding the HW part, the general architecture of the system is given in the form of a Xilinx Embedded Development Kit (EDK) description. We provide the necessary IP cores in EDK format and the templates with the system interconnection and external interfaces. EDK is also used to instantiate the desired number of MicroBlaze processors. On the other hand, related to the SW part, the standard GNU GCC is used to compile the sources for the processing cores. At this point, all the elements of the SoC, except for the interconnection network, are integrated. The external ports of the generated EDIF contain the nets necessary to attach the interconnection network.

As interconnection network we have used the Xpipes NoC [14]. In our case, XpipesCompiler takes as input a file with the topological and parametric description of the desired NoC and produces

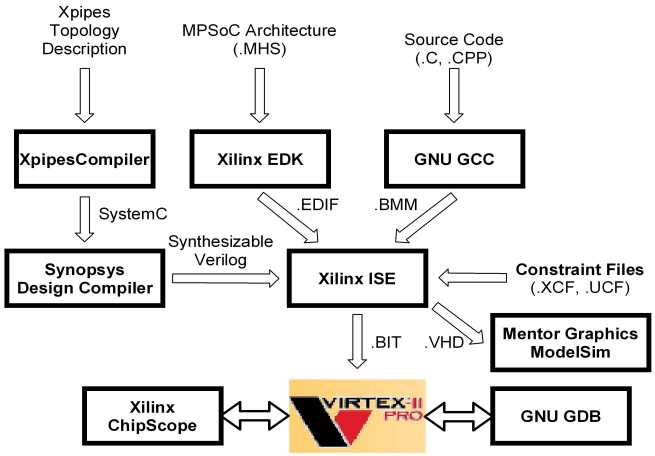



Figure 2. Working flow for the NoC case

as output a SystemC file containing the implementation. Therefore, Xpipes can be used and adapted for a wide range of applications. Using the Synopsys Design Compiler [18] synthesizable Verilog files are obtained from the SystemC ones. Xpipes uses the OCP protocol for interconnection of the network interfaces with the external cores; Thus, we have developed an OPB-to-OCP bridge in order to connect the MicroBlaze cores to it.

Then, we use Xilinx ISE to mix the obtained EDIF with the interconnection network Verilog files and the platform description VHDL ones. Finally, Xilinx tools map the whole design onto the final FPGA and the architecture with the application sources is ready to be downloaded onto the chip to start the testing process.

In addition, our proposed flow enables the use of additional debugging or simulation tools. Extensive testing and architecture analysis can be done at design level with Mentor Graphics ModelSim. Furthermore, at run-time, Xilinx ChipScope and GNU GDB can be used to debug and trace the framework. Statistics of the network can be obtained either via simulation or real execution on the platform.

## 4. Case Studies

To demonstrate the use of the MPSoC framework we present two case studies. First, a general purpose MPSoC system connected by a hierarchy of buses and, second, we have replaced the buses by a Xpipes-based NoC.

### 4.1. Baseline MPSoC Configuration

The test system fits suitably in a Virtex 2 PRO 20 FPGA with both interconnection options. We use four MicroBlaze processors with 4 Kb of private memory and two 64-Kb shared memories implemented in the FPGA. All memory accesses out of the shared range are immediately dispatched in the private ambit of the CPU and do not use the interconnection network. Interconnection bridges, depending on the chosen implementation, route the shared memory accesses and locks the local OPB until they are resolved.

### 4.2. Case study 1: bus based

First we show an interconnection based upon a simple bus hierarchy, using the standard OPB: a synchronous, master-slave, standard IP bus that can be efficiently implemented on a FPGA and is directly compatible with MicroBlaze. The bus hierarchy allows the CPUs to access the shared memories. There are as many OPB private buses as processors, and an additional public bus. Each

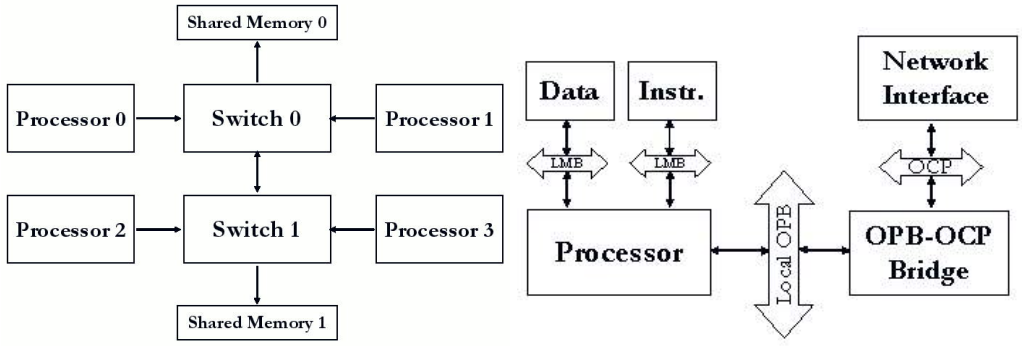
Table 1
Use of resources with both interconnection alternatives

| Platform version | Slices |
|---|---|
| Bus based | 3,050 of 13,700 (22%) |
| Xpipes NoC based | 9,497 of 13,700 (69%) |

private OPB has an OPB-OPB bridge connected to the shared medium that has also the shared RAM. Each OPB bus has its own arbiter, which has a fundamental role in the behaviour of the system. In all cases, we use the Xilinx dynamic arbiter [19] with a LRU priority system.

### 4.3. Case study 2: Xpipes NoC based

For this case study we have instantiated a custom 2 switches NoC-based interconnection topology using XpipesCompiler. The overall system architecture is shown in Figure 3.




(a) Global view (b) Detail of processor connection

Figure 3. Implemented NoC topology

In order to be able to connect the MicroBlaze using the structure explained in the previous point, it is precise to establish a connection between the OPB of the CPUs and the corresponding Network Interface (NI). As the NI implements a standard OCP interface, a bridge between OPB and OCP can serve to this purpose. In the present work a specialized specific module has been created for this purpose, written in VHDL. This module works as slave in OPB and as master in the OCP side. Optionally, the peripheral can implement a prefetch buffer of customizable size. This buffer anticipates the reading of contiguous data and does not have effect in the writings. Writes and prefetched reads have minimum latency. This has been done to minimize the impact of the NoC intrinsic latency and to take advantage of burst transfers functionality of Xpipes.

### 4.4. Synthesis Results

The use of FPGA resources for each version is shown in Table 1. Note that the NoC version is more than triple the size of the bus one. One first conclusion of this study is that the NoC generated by XpipesCompiler is optimized for silicon, but it is not the most suitable one for the internal structure of the FPGA architectures. Similar results can be envisaged for other NoCs with complex internal switches and NIs.

Table 2
Overall results obtained in Case Study 1

| | Bus | NoC, prefetching (2 words) | NoC, no prefetching |
|---|---|---|---|
| T (cycles) | $1.759\times10^{6}$ | $1.716\times10^{6}$ | $1{,}925\times10^{6}$ |
| BW (bytes/cycle) | 0.149 | 0.153 | 0.136 |
| BW/slices | $4.88\times10^{-5}$ | $1.61\times10^{-5}$ | $1.43\times10^{-5}$ |

## 5. Applications and Empirical Results

In order to test the platform and to show its versatility to compare different interconnection strategies we show two test applications that run on both interconnection options (bus and Xpipes NoC). In the case of the NoC case study, tests have been done with both prefechting enabled and disabled. Both applications are based on image processing with graphic frames stored in the shared memories. For each case, we measure the total execution time (T in Table 2 and Table 3). Because our experiments pursue architectural results, all data collected is expressed in clock cycles and not in real time.

Also, it is interesting to provide a measurement of the performance of each option and relate it to its architectural complexity. Bus is likely to have less performance than the NoC option but NoC uses more area, as some authors outline in their studies [5,7,8,11]. To approximate this relation, we calculate an efficiency coefficient as the quotient between the effective bandwidth of the application (BW in Table 2 and Table 3) and the number of slices that the implementation occupies.

### 5.1. Parallel greyscale dither

In this application, a 256x256 pixels grayscale image is stored in each shared memory. Each pair of processors work in one image to obtain the monochrome dithered image and store it in the same memory.

The experimental results (see Table 2) show that the application performance is not deteriorated by the use of a shared bus. The reason is that regular algorithms are executed across all the processors. Therefore, when an algorithm accesses the memory in a regular way, and the processing time gives enough margin for it, the accesses tend to align themselves after the initial conflicts. Hence, some processors access the memory while others process the last fetched data. As processing and access times are almost constant, once balance is reached, this situation is preserved for the rest of execution.

In the case of NoC without prefetching, the application is significantly slower. This is due to the high number of readings, as many as writings, which suffer from the extremely high latency of the NoC for single read-requests. Latency is almost insignificant for writings, but it is still not enough to accelerate the execution of the algorithm. A moderate improvement occurs with prefecthing. In this case, the execution takes around 11% less time compared to the NoC version without prefetching, and it is even slightly faster than the bus one. This is because each pair of CPUs can access its shared memories without colliding with the other pair. Particular reasoning deserve the results of the efficiency coefficient, as it is more than three times better for the bus than for the NoC: in order to obtain a performance improvement of around 2.5% with respect to the bus version, the NoC requires more than three times of space in the FPGA. From these results we can conclude that the bus, in this case, is the option that presents the best trade-off between area and performance for the current mapping of the NoC and buses.

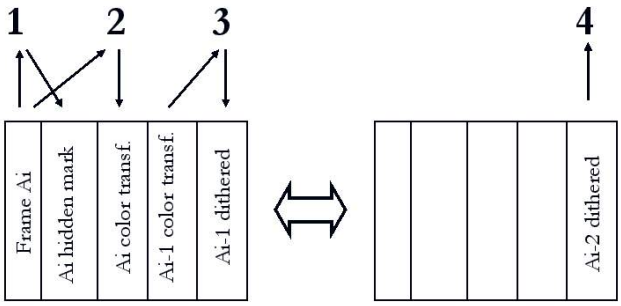



Figure 4. Test application 2: Multiprocessor image pipeline

Table 3
Overall results obtained in Case Study 2

| | Bus | NoC, prefetching (8 words) | NoC, no prefetching |
|---|---|---|---|
| T (cycles) | $2.927\times10^{6}$ | $1.708\times10^{6}$ | $3.381\times10^{6}$ |
| BW (bytes/cycle) | 0.113 | 0.193 | 0.097 |
| BW/slices | $3.7\times10^{-5}$ | $2.03\times10^{-5}$ | $1.02\times10^{-5}$ |

### 5.2. Multiprocessor image pipeline

In the second test a parallel application implements an image pipeline in four processors. Three processing elements process image data in one memory and the remaining CPU extracts in parallel results from the other memory and loads the new data. Then, the roles are exchanged for the next frame chaining the process (see Figure 4). Hardware semaphores are used to synchronise the processes.

A significant improvement of more than 41% in speed is attained using the NoC with prefetching (see Table 3) and, as a result, the efficiency coefficient of the implementation using the NoC with buffer is increased. However, it still continues being lower than for the bus ($3.7\times10^{\cdot5}$ vs $2.03\times10^{-5}$). Again, the main conclusion is that the application requirements determine the best option for the developer. However, with this topology the bus clearly achieves a better use of the FPGA resources as far as transmission efficiency is concerned.

## 6. Conclussions and Future Work

In this paper we have presented a new MPSoC platform that enables the execution of complex real-life applications and to study different interconnection alternatives. It is implemented in state-of-the-art FPGA technology. The system is easily scalable and the proposed architecture can support any number of IPs with the only limit of HW availability. Even systems that cannot be implemented in real-life can be used for architectural experiments through RTL simulation. Then, we have illustrated its versatility with an example of customization of the architecture with four general-purpose processing cores, and implemented with a high-performance bus and a state-of-the-art NoC. We have also provided typical parallel graphic applications and analyzed the performance of each type of interconnection. Finally, as an example of the utility of the proposed platform, we have studied the possible trade-offs between performance and area for both bus and NoC-based designs.

Our results suggest a number of possible future research lines. First, further work needs to be carried out to scale the platform to a larger variety of IPs. More specifically, it would be interesting

to add specific DSP-like cores and more types of existing memories, and we plan to incorpore the hard-wired PowerPC cores available on the used Xilinx FPGA for our platform. Second, because software and system nature determines the most convenient interconnection, we want to investigate this relationship more in deep, namely, we would like to know where a bus can be used without complexity or performance problems and, conversely, when a NoC is more appropriate. Third, our experiments indicate that the behaviour of the NoC can be significantly improved if simple tailor-made optimizations are added according to the eventual type of running applications. For instance, NoC performance has been remarkably enhanced by a simple prefetch buffer. Moreover, instead of using this simple mechanism, it can be considered the use of real data caches to obtain additional performance gains. Finally, due to the synthesis results obtained with the NoC, we consider that there is a big room for possible improvements by developing a network without abstracting the physical device's architecture. In this context, the design of switches according to the specific architecture of FPGAs could clearly achieve extensive benefits.

## Acknowledgements

This work is partially supported by the Spanish Government Research Grant TIC2002/0750 and a Mobility Post-doc Grant from UCM for David Atienza.

## References

[1] Luca Benini and Giovanni De Micheli. Networks on chip: a new soc paradigm. *IEEE Computer*, 35(1):70–78, January 2002.
[2] Gordon Brebner et al. Networking on chip with platform fpgas. In *Proc. of FPT*, 2003.
[3] Doris Ching et al. Integrated modeling and generation of a reconfigurable network-on-chip. In *Proc. of IPDPS*, 2004.
[4] Nicolas Genko, et al. A complete network-on-chip emulation framework. In *Proc. of DATE)*, 2005.
[5] Antoine Jalabert et al. xpipescompiler: A tool for instantiating application specific networks on chip. In *Proc. of DATE*, 2004.
[6] S. Kolson et al. A network on chip architecture and design methodology. In *Proc. of ISVLSI*, 2002.
[7] Theodore Marescaux et al. Networks on chip as hardware components of an os for reconfigurable systems. In *Proc. of FPL*, 2003.
[8] Fernando Moraes et al. Hermes: an infrastructure for low area overhead packet-switching networks on chip. *Integration, the VLSI Journal*, April 2004.
[9] OCP International Partnership (OCP-IP). Open core protocol standard, 2003. http://www.ocpip.org/home.
[10] David Siguenza-Tortosa et al. Vhdl-based simulation environment for proteo noc. In *Proc. of HLDVT Workshop*, 2002.
[11] C.A. Zeferino, et al. Rasoc: a router soft-core for networks-on-chip. In *Proc. of DATE*, 2004.
[12] A. Jerraya and W. Wolf. *Multiprocessor Systems-on-Chips*. Morgan Kaufmann, Elsevier, 2005.
[13] C.A. Zeferino et al. Socin: a parametric and scalable network-on-chip. In *Proc. of SBCCI*, 2003.
[14] Davide Bertozzi and Luca Benini. Xpipes: A Network-on-Chip Architecture for Gigascale Systems-on-Chip. In *IEEE Circuits and Systems Magazine*, 2004.
[15] Kees Goossens, et al. The Aethereal network on chip: Concepts, architectures and implementations. In *IEEE Design and Test of Computers*, Vol 22(5):21–31, Sept-Oct 2005.
[16] Xilinx Inc. The Virtex2 Pro HandBook.
[17] Xilinx Inc. MicroBlaze Processor User-Guide.
[18] Synopsys Inc. The Synopsys Design Compiler User Guide.
[19] Xilinx Inc. On-Chip Peripheral Bus v2.0 with OPB Arbiter.