

# Post-Silicon Debug Using Programmable Logic Cores

Bradley R. Quinton  
*Electrical and Computer Engineering*  
*University of British Columbia*  
*Vancouver, Canada*  
bradq@ece.ubc.ca

Steven J.E. Wilton  
*Electrical and Computer Engineering*  
*University of British Columbia*  
*Vancouver, Canada*  
steve@ece.ubc.ca

## Abstract

*Producing a functionally correct integrated circuit is becoming increasingly difficult. No matter how careful a designer is, there will always be integrated circuits that are fabricated, but do not operate as expected. Providing a means to effectively debug these integrated circuits is vital to help pin-point problems and reduce the number of re-spins required to create a correctly-functioning chip. In this paper, we show that programmable logic cores (PLCs) and flexible networks can provide this debugging capability. We present an architecture and example implementation. We show that the area overhead of this proposed architecture would be well below 10% for many target ICs.*

## 1. Introduction

Advances in integrated circuit (IC) technology have made possible the integration of a large number of functional blocks on a single chip. There are many challenges associated with this high level of integration. One of these challenges is ensuring that the IC design is functionally correct. Although pre-fabrication simulation is used to help ensure that the IC performs as desired, the complexity of modern ICs prevents exhaustive verification. Design errors (bugs) are often found post-fabrication. For complex ICs, the process of verifying and debugging new devices is a significant cost and time investment [1]. As the level of IC integration continues to rise, this problem will be exasperated as more functionality is combined into a single “black-box” that must be tested using only the external chip interfaces.

In this paper, we show that *programmable logic cores* (PLCs) embedded on fixed-function ICs [2, 3] offer a unique opportunity to address this problem. During the post-fabrication verification of ICs, a flexible network is used to provide relevant signals from within the fixed-function IC to the programmable logic core; the programmable logic core is then be used to monitor these signals, and return information about these signals to the verifier (possibly in real-time). In more complicated cases

the verifier can override the internal signal to observe the effect on the output of the IC.

Embedding programmable logic cores on fixed-function SoCs is not new [4, 5, 6]. In these previous designs, however, the purpose of the programmable logic core is to provide post-fabrication flexibility for the design itself. Using this approach the parts of the design that are expected to change are mapped to the programmable logic core. Then, if requirements change post-fabrication, these changes can hopefully be incorporated into the programmable logic portion of the chip. Our approach is different in that our programmable logic core is *not* used to implement part of the design. Instead, we use the programmable logic core solely to implement debugging circuitry that will be used to debug the fixed-function chip. This unique use of a programmable logic core requires a unique flexible network, as well as unique interface circuitry. Our debugging architecture, which contains the programmable logic core as well as the network and interface circuitry, is the focus of this paper.

The post manufacturing re-configurability of the network and programmable logic core is a key aspect of the technique. During post-fabrication verification, the region of the circuit being debugged may change over time, and in any case, is not likely predictable during design time. The flexible network allows the verifier to select the internal signals that are of interest for any given test, and the programmable logic core provides a means to process these signals in a manner that depends on the debugging task being performed.

The ability to easily verify ICs in this way provides a number of benefits: 1) Reduced time-to-market because of an increased ability to isolate and understand unexpected behaviours, 2) Decreased resources required for post-manufacturing verification because of an increase in the functional coverage of a given test, 3) Elimination of design revisions caused when one design error hides a second error, 4) Increased customer satisfaction because of enhanced ability to provide “work-arounds” to known bugs.

This paper is organized as follows. Previous work in this area will be outlined in Section 2. Section 3 outlines the framework that we have assumed. Our debugging flow and architecture is described in Sections 4 and 5. Finally, Section 6 shows that the overhead required by our architecture is modest.

## 2. Previous work

The observation or control of intermediate signals/variables is an important tool that is commonly used when debugging systems. Implementation of this technique in the software portion of a system is straight-forward using standard debug software. In the hardware portion of a system, internal signals can be observed or controlled using test points. However, these test points can only be inserted at the circuit board level, or, if an FPGA implementation is used, within the FPGA [7, 8]. It is difficult to observe or control signals within fixed-function ICs. After fabrication, the chip cannot be easily modified to provide these signals to output pins where they can be observed. Even if the signals were identified before fabrication, providing external access to these signals at design-time is problematic since I/O resources are often limited and do not operate at the high speeds that would be required.

A number of methods have been used to facilitate functional debug of post-silicon ICs. Some of these methods are ad-hoc, such as controlled un-loading of DFT scan chains. Other methods are more formalized such as in-circuit emulation (ICE) for stand-alone processors [9]. Modern processors have made use of the JTAG boundary-scan mechanism to enable similar functionality. Recently there have been proposals to extend the JTAG based access to support ICs with multiple CPUs on a single die [10]. These targeted mechanisms make use of key characteristics of processors that are not present in general digital logic designs. They rely on the ability to halt the processor to examine the signals. This is a requirement because the test interface of these mechanisms is much slower than the internals of the processor. In addition, the interface for a processor is well defined and it is possible to create specialized debug logic to address a number of possible debugging situations. With the trend towards system-on-chip (SoC) designs, application-specific logic and processors are being combined on a single die. This application specific logic is rarely designed to enable controlled halting of the normal operation. IP blocks such as high-speed serial receivers/transmitters and clock synthesis units will not tolerate clocks that are not free-running. These blocks often require clocks to be maintained in a specific frequency range. In addition, application-specific digital logic often communicates with ad-hoc interfaces, making it difficult or impossible to

design fixed debug logic that will be useful once a problem is discovered. In order to effectively debug this logic, flexible, full speed, real-time debugging is required. Programmable logic cores (PLCs) have the potential to provide this functionality.

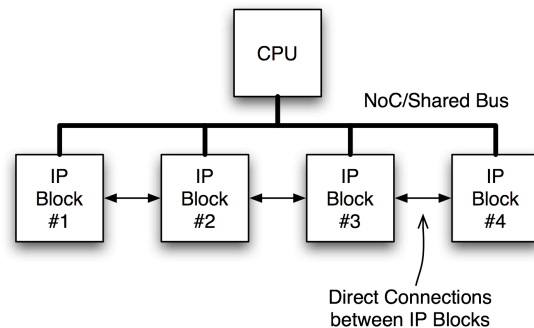


Figure 1: Baseline Architecture

## 3. Framework

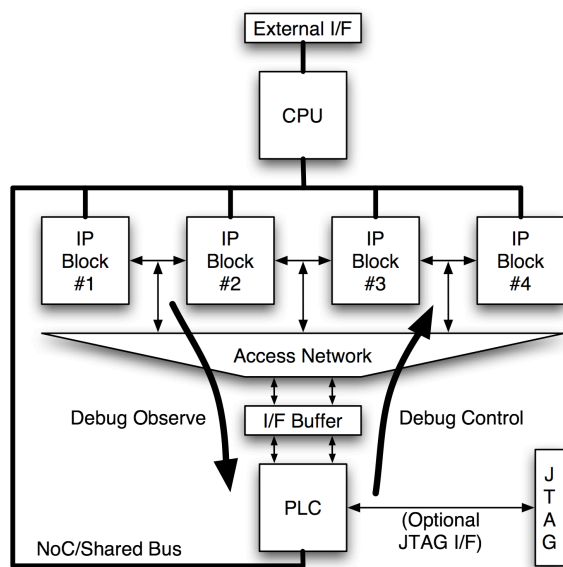
We assume a SoC design containing multiple IP cores, as shown in Figure 1. Typically, at least one of these cores will be a processor. These cores can be connected either by fixed wires, or via a Network-on-Chip (NoC) architecture [11]. Although it has been suggested that, in future integrated circuits, all inter-block communication will be done using dynamic packet-based networks, today's integrated circuits often contain a mix of networks and fixed wires to connect IP blocks. Our technique works with both styles of interconnect; in the following discussion we assume that most inter-block communication is done using fixed direct wires, while a NoC is used only for communication between the processor and the IP-blocks.

## 4. Debugging flow

Figure 2 shows how the baseline architecture of Figure 1 can be enhanced to improve the ease with which the system can be debugged. The heart of the debugging architecture is the Programmable Logic Core (PLC) which is used to observe and control internal signals in the rest of the integrated circuit. The PLC is connected to the rest of the chip using an interface buffer and access network. In addition, the PLC can communicate with the on-chip CPU using a shared bus or NoC (which probably already exists in the SoC). This subsection describes how a SoC designer (or verifier) can use the architecture of Figure 2 to ease the debugging process.

At design time, the SoC designer does not yet know whether the chip will fail, and if so, how it will fail. Thus, it is impossible to select a set of signals within the integrated circuit that will be connected to the PLC. Instead, the designer chooses a much larger set of signals (which we refer to as the

*observable/controllable signals*) from the integrated circuit, and connects them to the input of the access network, as shown in Figure 2. In the next section, we consider access networks with up to 7200 inputs. Although this is a large number of observable/controllable signals, it is still not sufficient to provide access to every internal signal in an integrated circuit. Although the selection of the observable signals is somewhat instance-dependent, we assume that all signals that are used to connect IP blocks to other IP blocks are selected (we do not consider signals internal to IP blocks as observable/controllable signals). However, our technique will work equally well if the designer wishes to provide the ability to control/observe selected intra-IP block signals as well.



**Figure 2: Enhanced Architecture**

After the chip has been fabricated, and is deemed to require debugging, the designer (or verifier) can program debugging circuitry into the programmable logic core (PLC). The PLC is used to pre-process observed data before passing it to the supervisory processor (or test interface), and to provide high-speed control of the IC's internal signals based on directions from the supervisory processor (or test interface). By allowing debug signals to be manipulated, the use of the PLC significantly reduces the off-chip communication requirements. For example, a counter may record the number of transitions on a signal over given period, and only return the total number of transitions. As another example, key bytes in the data stream could

be recorded while other unimportant signals could be discarded.

In addition to programming the PLC, the designer or verifier can select signals from the set of *observable/controllable signals* to connect to the PLC. Since the number of pins on a PLC is typically smaller than the number of observable/controllable signals, an access network is used to select the observable/controllable signals that will drive and be driven by the PLC. The configuration of this network (i.e. which observable/controllable signals are connected to the PLC) can be programmed at the same time as the PLC by programming memory bits that control the routing paths in the network.

## 5. Architecture

This section presents more details regarding each component in the architecture of Figure 2.

### 5.1. Programmable logic core

As described earlier, the programmable logic core (PLC) is used to pre-process observed data and to provide high-speed control of the IC's internal signals.

Programmable Logic Cores have been well studied [2] and commercial PLC's are available from several vendors [3,15,16]. For the application described in this paper any one of these cores would provide the desired functionality. However, the logic density and performance of the core will have a direct impact on the efficiency of the overall scheme. In our architecture, we need only a small amount of programmable logic (since we don't intend to replace entire IP blocks); we estimate that a few thousand ASIC gates would be sufficient for most debugging tasks. However, for our current area estimates we use a larger (10,000 gates) commercial PLC. This ensures conservative overhead estimates.

In addition to flexible resources, most modern FPGAs, and many commercial PLCs contain embedded RAM. This RAM is important in our application since it allows test data to be stored in the PLC itself. Since the RAM takes up area that could otherwise implement logic, there is a trade-off between the amount of logic and the amount of storage provided in the PLC. The best choice for this trade-off is again very application-dependent.

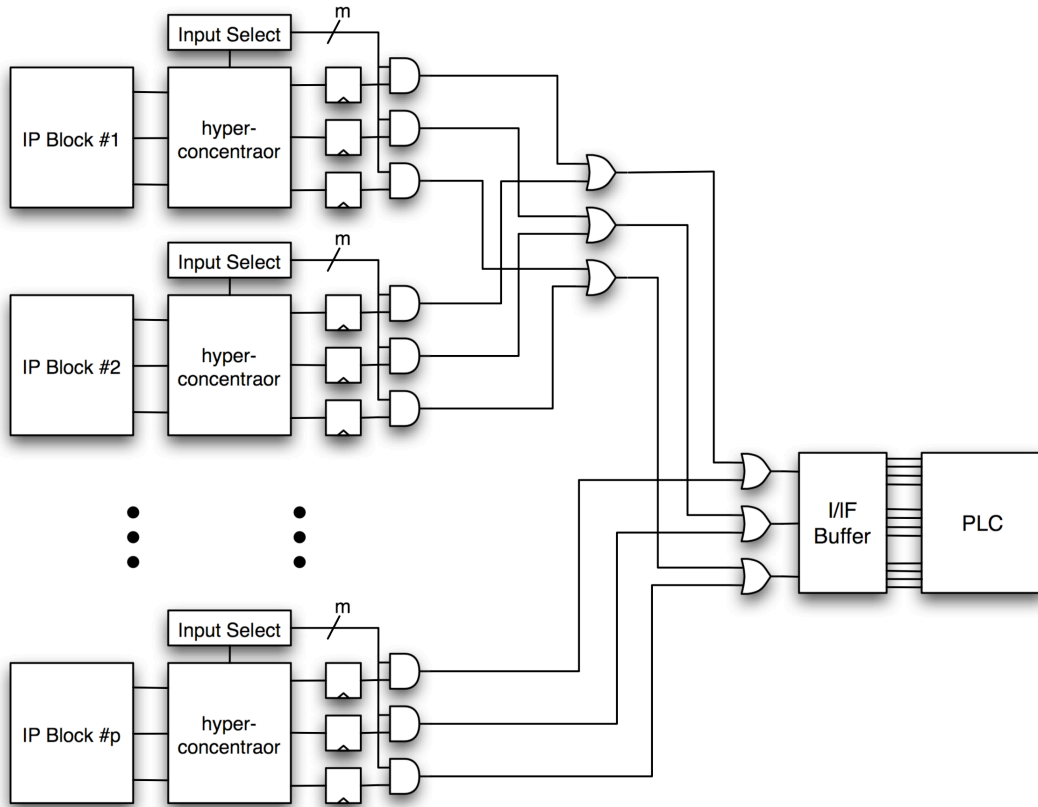


Figure 1 'Observe' Access Network Architecture

## 5.2. Access network

As described above, the access network is used to programmably connect any subset of the *observable/controllable signals* to the PLC.

Although there are several possible ways of designing this network, we employ a concentrator network [12]. Such a network has  $n$  inputs and  $m$  outputs (the inputs and outputs can be reversed to construct a bi-directional network) where  $n > m$ . The network has the property that any subset of size  $\leq m$  of the  $n$  input signals can be connected to the outputs, but without regard for the order of the output signals. This relaxation of the output ordering constraint takes advantage of the fact that all pins in a PLC are equivalent. Previous work has shown practical constructions of these networks with low area cost and low network depth [13].

Figure 3 shows the structure of the 'observe' portion of our network. The network structure for the 'control' portion of the network is very similar since a mirror of the network can be used in the reverse direction.

We use a two-stage network implementation. The  $n$  inputs are partitioned into  $k$  groups [18], where

$k$  is the number of IP blocks. We assume that the size of each of these  $k$  groups of inputs is  $\leq m$ , the number of network outputs. In the first stage of the network each of the  $k$  groups is considered independently. A class of network called a *hyper-concentrator* is used for each of the  $k$  groups. 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. We use the hyper-concentrator construction proposed by Narasimha [19], which is 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.

The above assumes that each IP block has the same number of pins,  $k$ . If this is not the case, the same hyper-concentrator network is constructed for each IP block based on the IP block with the largest number of pins. Standard logic optimization tools are used to remove unneeded logic.

The second stage of the network combines the outputs of the  $k$  hyper-concentrators with a simple multiplexed bus structure. We use an enabled OR-tree implementation, as shown in Figure 3. This implementation is advantageous as it avoids the



enhanced architecture. The model is parameterized based on the total number of gates in all IP cores and the number of observable/controllable signals. The area is calculated by summing the size of the IP cores themselves, a published area estimate of a specific programmable logic core and the area of standard cell implementations of the interface circuitry and access network. A 90nm technology and the STMicroelectronics Core90GPSVT standard cell library were assumed throughout [20].

The PLC we assumed was the 90nm PLC from IBM/Xilinx and we used the area estimate published for this core [3]. This PLC is equivalent to approximately 10,000 ASIC gates. The PLC has 384 available I/O ports. We assigned half of these I/O ports to be inputs and the other half were assumed to be outputs. Taking into account the interface buffering, the number of simultaneously observable and controllable internal signals is therefore 23. The interface to the NoC/shared network is implemented in the PLC logic and therefore required no additional area overhead.

The access network area and interface buffer area were calculated by synthesizing the architecture of Figures 3 and 4 for various numbers of inputs. For the hyper-concentrator portion of the network we used 2:1 multiplexers for switching and a flip-flop (with enable) per 2:1 multiplexer for routing control. The multiplexed bus was constructed using 2-input 'OR' gates, 'AND' gates and flip-flops to maintain the state of the bus. For the interface buffer between the PLC and the access network, an optional serial-to-parallel buffer was created for each PLC pin. Each buffer contained enough storage to accommodate a 4:1 ratio between the clock frequency of the fixed logic and the PLC. Flip-flops (with enable) were used as the storage mechanism in the buffer.

In order to better delineate the contributions to the area overhead of the proposed architecture, the network and PLC overhead are presented separately. Figure 5 shows the overhead of the access network. Integrated circuit sizes of between 5 million and 80 million gates were assumed (this is the sum of the gate count of all IP blocks before the debugging enhancements were added). For each integrated circuit size, the number of observable/controllable signals was varied from 100 to 7200, and the area overhead of the network relative to the original integrated circuit was plotted. As shown in Figure 5, as the number of observable/controllable signals increases, the area overhead of the network increases linearly. The figure shows that the overhead of the access network for a 20 million gate integrated circuit in which 7200 signals are to be controlled or observed is roughly 2%.

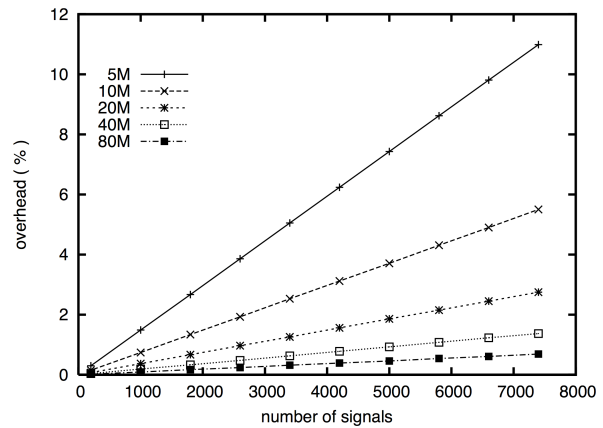


Figure 5 Access Network Overhead

The area overhead due to the PLC core does not depend on the number of controllable/observable signals. The PLC overhead is simply the area of the PLC core divided by the size of the original integrated circuit. Using the 10,000 ASIC gate core from IBM/Xilinx, the overhead of the PLC core ranges from less than 1% to about 9% for our assumed integrated circuits. These overhead results are shown in Figure 6.

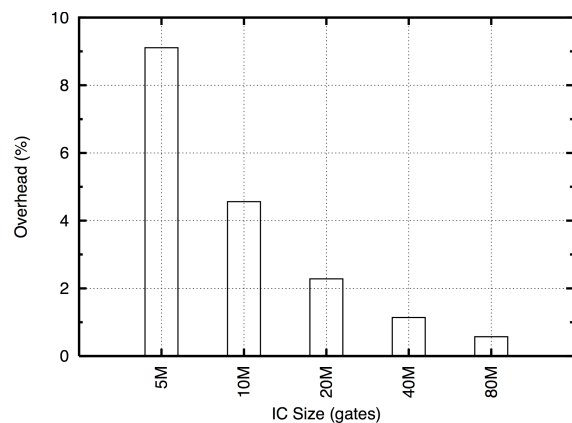


Figure 6 PLC Overhead

Finally, Figure 7 shows the combined overhead results. The graph shows that for large IC designs, the cost of implementing the extra logic to facilitate post-silicon debug is modest. For example, on a 20 million gate ASIC it would be possible to observe and control 7200 internal signals for an overhead of 5%. Note that as the IC design becomes larger, the area of the PLC is amortized and its overhead becomes less significant. For these large ICs the cost of the access networks will be the important issue.

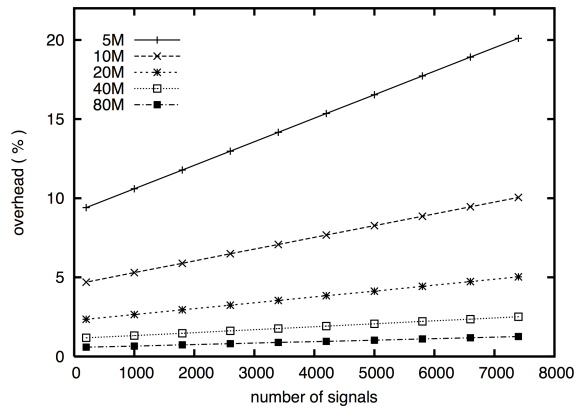


Figure 7 Architecture Overhead

## 7. Conclusion

In conclusion, we have shown that it is feasible to integrate a PLC in such a way that it could be used to assist post-silicon debug of complex modern ICs. This will reduce time-to-market and will enhance verification effectiveness. We have outlined the integration of the architecture into ICs and highlighted key design decisions that must be addressed during this integration. Finally we have shown the overhead required for a specific example implementation is quite modest for large ICs.

## 11. References

- [1] S. Sandler, "Need for debug doesn't stop at first silicon", *E.E. Times*, Feb. 21, 2005.
- [2] S.J.E. Wilton and R. Saleh, "Programmable Logic IP Cores in SoC Design: Opportunities and Challenges", *Proc. IEEE Custom IC Conf.*, San Diego, CA, pp. 63-66, May 2001.
- [3] P. Zuchowski, et al., "A Hybrid ASIC and FPGA Architecture", *Proc. IEEE/ACM Inter. Conf. on Computer-Aided Design*, pp. 187-194, 2002.
- [4] M. Borgatti, et al., "A Reconfigurable System featuring Dynamically Extensible Embedded Microprocessor, FPGA, and Customisable I/O", *IEEE Journal of Solid-State Circuits*, vol. 38, no. 3, pp. 521-529, March 2003.
- [5] T. Vaida, "PLC Advanced Technology Demonstrator TestChipB", *Proc. IEEE Custom IC Conf.*, pp. 67-70, May 2001.
- [6] L. Cali, et al., M. Borgatti, "Platform IC with Embedded Via Programmable Logic for Fast Customization", *Procs IEEE Custom ICs Conf.*, pp. 419-422, Oct. 2004.
- [7] Altera Corp., "SignalTap Embedded Logic Analyzer Megafunction", Data Sheet, v2.0, [www.altera.com](http://www.altera.com).
- [8] Xilinx, "ChipScope Pro Software and Cores User Guide", Data Sheet, v6.3.1, [www.xilinx.com](http://www.xilinx.com).
- [9] H-M Chen, et al., "Analysis of Hardware and Software Approaches to Embedded In-Circuit Emulation of Microprocessors", *Proc. of the Asia-Pacific Computer Systems Architecture Conference*, Melbourne, Australia, 2002.
- [10] R. Leatherman and N. Stollon, "An embedded debugging architecture for SoCs", *IEEE Potentials*, Feb/Mar. 2005.
- [11] W.J Dally and B. Towles, "Route Packets, Not Wires: On-Chip Interconnection Networks", *Proc. of the Design Automation Conference*, Las Vegas, NV, pp. 684-689, June 2001.
- [12] F. K. Hwang, *The Mathematical Theory of Nonblocking Switching Networks*, World Scientific, New Jersey, 1998.
- [13] B.R. Quinton and S.J.E. Wilton, "Concentrator Access Networks for Programmable Logic Cores on SoCs", *Proc. of the IEEE International Symposium on Circuits and Systems*, Kobe, Japan, May 2005.
- [14] T. Chelcea and S. M. Nowick, "Robust Interfaces for Mixed-Timing Systems", *IEEE Trans. on VLSI*, vol. 12, no. 8, Aug. 2004.
- [15] Actel Corp., "VariCore Embedded Programmable Gate Array Core 0.18 $\mu$ m Family", *Data Sheet*, v2.2, [www.actel.com](http://www.actel.com).
- [16] M2000, "90nm FlexEOS Product Description", *Product Description*, [www.m2000.fr](http://www.m2000.fr).
- [17] IEEE Std. 1149.1-1990, IEEE 1149.1 Standard Test Access Port and Boundary-Scan Architecture, *IEEE Computer Society*, 1990
- [18] J. Y. Hui, *Switching and Traffic Theory for Integrated Broadband Networks*, Kluwer Academic Publishers, Boston, 1990.
- [19] M. J. Narasimha, "A Recursive Concentrator Structure with Applications to Self-Routing Switching Networks", *IEEE Trans. on Communication*, vol. 42, no. 2/3/4, pp. 896-897, April 1994.
- [20] STMicroelectronics, "Core90GPSVT\_Databook", *Data Sheet*, Oct. 2004.