Increasing Observability in Post-Silicon Debug Using Asymmetric Omega Networks

André B. M. Gomes[1,2], Fredy A. M. Alves[2],
Ricardo S. Ferreira[1], José Augusto M. Nacif[1,2]

[1]Departamento de Informática, UFV
[2]Instituto de Ciências Exatas e Tecnológicas, Campus UFV-Florestal
{andre.maciel,fredy.alves,ricardo,jnacif}@ufv.br

ABSTRACT

Current pre-silicon verification techniques can not guarantee error free designs for complex integrated circuits during their first fabrication. Some errors are only uncovered when the device is running at full clock speed. Using post-silicon debug techniques, the designer can monitor the device capturing errors that occur only after millions of clock cycles. However, in order to identify the cause of the error, the signals must be stored in a trace buffer memory, dumped, and then analyzed. Thus, the trace buffer size limits the number of analyzed signals, forcing the designer to select a signal subset to monitor from all tapped signals. In this paper, we propose a novel asymmetric network, based on the traditional Omega Network. We propose to use this network as an interconnection fabric to connect the monitored signals to the trace buffer. We compare the Asymmetric Omega Network performance to the Mux Tree Network, which is adopted by industry as the standard solution to interconnect signals for post-silicon debug. We show that our Asymmetric Omega Network is ≈ 4.6 times more effective reducing the blocking rate at the cost of $\approx 21\%$ area overhead compared to Mux Trees.

Categories and Subject Descriptors

B.7.3 [**Integrated Circuits**]: Reliability and Testing

General Terms

Verification, Design

Keywords

Verification; Trace-Based Debug; Interconnection Network

SBCCI '15 August 31-September 04, 2015, Salvador, Brazil
 ISBN 978-1-4503-3763-2/15/08...$15.00
DOI: http://dx.doi.org/10.1145/2800986.2801011

1. INTRODUCTION

Verification is the most challenging stage in an integrated circuit development cycle. Due to the systems increasing complexity, the first tape-out normally includes design errors [24]. In post-silicon debug, the correctness of the manufactured device is evaluated by using real applications and environments, at full system speed. The four major steps in post-silicon validation involve: *1*) detecting a problem; *2*) localizing the problem to a reduced region; *3*) identifying the root cause; *4*) fixing or bypassing [16]. This process has become essential and on average consumes 35% of the integrated circuit development cycle [4].

Observability limitation is a challenging problem in post-silicon debug. In industry, a popular post-silicon debug technique uses a memory buffer to trace signal values enabling the analysis of real time data. The data is captured and stored over a period enhancing the circuit observability. Unfortunately, few signals can be monitored each time because the buffer size is constrained by design overhead costs. As a consequence, the debugging process still suffers from lack of observability, making the validation process difficult and time-consuming even for a circuit with few or no errors [2].

To achieve better results, the designer should choose a good set of signals to trace prior to the identification of any design errors. Usually, the designer selects the signal set based on the circuit complexity and *ad hoc* error localization probability assumptions [2]. In a complex design, a good signal subset requires interaction among many experts [12]. Recent works [3], [11] have proposed algorithms for automatic signal selection. These algorithms are mostly based on the likelihood of restoring data from other signals [14]. Even with the help of a tool to automatically select the signals, some unavailable combinations can be crucial for understanding the cause and correcting the error. In a worst-case scenario, the designers are forced to include missing signals and perform a respin on the integrated circuit under debug. This situation increases the time-to-market and reduces revenue [25].

In industry the standard solution to interconnect million-gate designs with thousands of signals is the Pipelined Multiplexer Tree (Mux Tree) [1]. This interconnection network allows the signals to be dynamically selected resulting in different subsets. The Mux Tree interconnection network imposes low area overhead because it is composed by few stages

and it provides a unique path for each input/output pair. Unfortunately, the Mux Tree architecture does not allow any subset selection because signals that are connected to the same multiplexer inputs can not be traced together. In Mux Tree interconnection networks the blocking rate reaches on average, around 36% of all signal combinations. The impossibility to trace certain signal subsets can further delay the post-silicon debug process.

In a deterministic environment, the full trace observability is achieved by running the device repeatedly, selecting the signals that have been blocked to trace. But in real environments we can observe low levels of determinism, repeatability, and controllability. Asynchronous clock domains, electrical conditions and environment conditions contribute to an inconsistent outcome each time a test runs [7, 16]. In post-silicon validation, identifying the root cause of a bug in these conditions is substantially harder. By reducing the blocking rate in the interconnection network, higher observability rates can be achieved.

In this paper we propose a novel Multistage Interconnection Network (MIN) for post-silicon debug. This interconnection network is an asymmetric version of the traditional Omega network [15]. Compared to Mux Tree, our Asymmetric Omega Network presents lower blocking rates at the cost of a slightly higher area overhead. The contributions of this work are twofold. First, we propose the Asymmetric Omega Network describing how to construct this MIN starting from the symmetric version. Second, we discuss different area vs. blocking rate tradeoffs showing the feasibility of the Asymmetric Omega Network for post-silicon debug.

2. POST-SILICON VERIFICATION

In post-silicon debug the behavior of the circuit is observed in a real environment. When an integrated circuit fails, designers extract the traced information to analyze and find the root cause of the problem. Unfortunately, there are limitations that prevent the validation engineers to observe the behavior of every signal in detail for long periods of time. So, they are restricted by temporal or spatial scopes. The temporal scope limits the time window in which the signal set is observed. The spatial scope limits the number of signals in the signal set to trace [25]. Current debug process is restricted by costs, and often, it contains a better spatial scope or a better temporal scope, but not both together. These techniques are named traced-based debug and scan-based debug, respectively.

In a scan-based debug, the System-on-Chip (SoC) is stopped during the execution, then the value of all internal registers are dumped through the I/O ports. To retrieve the values of every register, this technique uses chained test registers. Therefore, full spatial scope is available. In a deterministic environment, a scan-based debug can be restarted at any time. However, when the system is not deterministic, it is hard to identify a state to restart, since there is few or no knowledge about the failure [17]. The start/stop and dump processes make this technique impractical to capture consecutive clock cycles [19].

Trace-based techniques overcomes the scan-based limitation allowing a better temporal scope. Figure 1 presents a block diagram of the trace buffer post-silicon debug technique. The **trace buffer** memory is used to store a set of internal signal values. If we consider a 16 Kbit buffer, it is possible to explore different temporal vs. spatial trade-offs. For example, we can use this buffer to store the values of 16 signals over 1,024 periods of time or 64 signals over 256 periods of time. The nature of the error being debugged will guide this selection. The **trigger logic** monitors the circuit behavior starting the signal trace when previous defined conditions are met. The **interconnection network** is responsible for selecting which signals are going to be stored in the trace buffer.

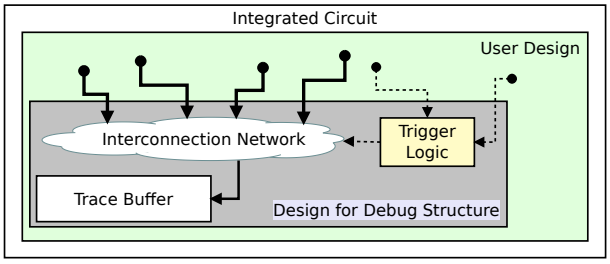
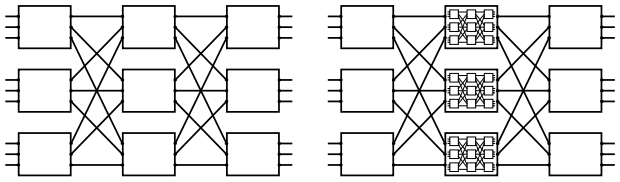


Figure 1: Trace buffer block diagram.

Before starting the manufacturing process, the designer chooses which signals will be tapped, and during debug runtime, which signals will be stored for further analysis. To allow this flexibility at runtime, the designer instantiates an interconnection network to connect the signals to the trace buffer memory. Due to cost restrictions, in post-silicon debug the area occupied by the interconnection network is a key factor. There is a huge number of interconnection network architectures for different applications [6].

A crossbar network, which is a non-blocking network, presents $O(N^2)$ cost, N being the number inputs/outputs. The non-blocking characteristic is possible because there is a dedicated path between each input and output. However, a crossbar often presents prohibitive area overhead to be embedded in a Design for Debug (DfD) infrastructure. The Clos network [5] is a non-blocking MIN with permutation capabilities, its recursive construction can reduce the number of crosspoints switches (see Figure 2). However, even when using a rearrangeable Clos [8, 13, 22], a non-blocking MIN still presents a prohibitive cost for post-silicon debug.



(a) 3 Stage Clos. (b) 5 Stage Clos.

Figure 2: Clos networks.

In industry, Mux Trees are used as the standard solution for signal interconnection in post-silicon validation due to low area overhead [1]. Often, this network is built by same size multiplexers, usually 2x1 multiplexers, that are connected in stages. The number of stages for a Mux Tree network with only 2x1 multiplexer is $\log_2(input) - \log_2(output)$. Mux

Trees are blocking networks, and there is always a unique path between an input and an output. These characteristics minimizes the number of stages and switches in the network. Figure 3 shows a 16x4 Mux Tree connected to a trace buffer. The input port number 7 is connected to output port number 1. The selection of port 7 makes impossible to trace concurrently ports {4, 5, 6}.

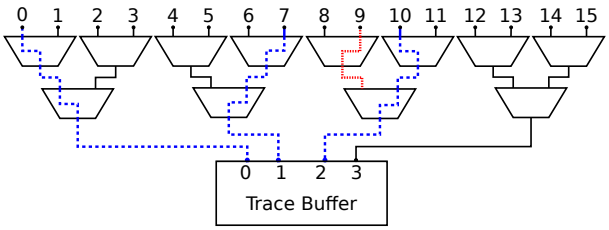
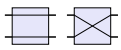


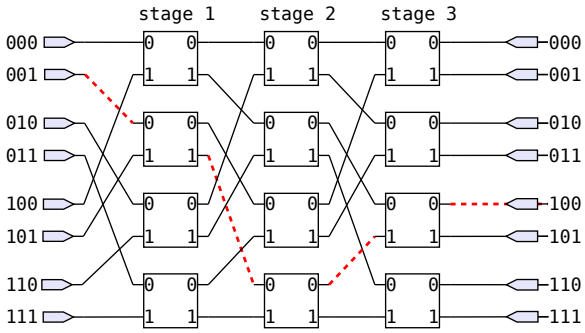
Figure 3: 16x4 Mux Tree network.

When a route can not be established between an input and an output, a blockage occurs. In this work, we use the blocking rate metric to measure the number of routed connections over the number of available outputs for a given set of signals. For example, consider the 16x4 Mux Tree network in Figure 3. For the input set {0, 7, 9, 10}, the blocking rate is 25%. The main characteristic of Mux Trees is low area overhead. On the other hand, the blocking rate for this interconnection network is, on average, 36%.

An Omega network [15] is a blocking network classified as a Banyan [10] network and Bidelta network [18, 20]. Thus, it has only one path between an input/output pair, and it has permutation capabilities. Omega network uses the *Perfect Shuffle* pattern [23] and $log_2(N)$ stages, N being the number of inputs/outputs. Extra stages can be added, doubling the path availability [9, 21]. Often, 2x2 switches (radix-2) are used, but other sizes are also possible.



(a) Straight and Crossover Connections.



(b) 8x8 Omega network.

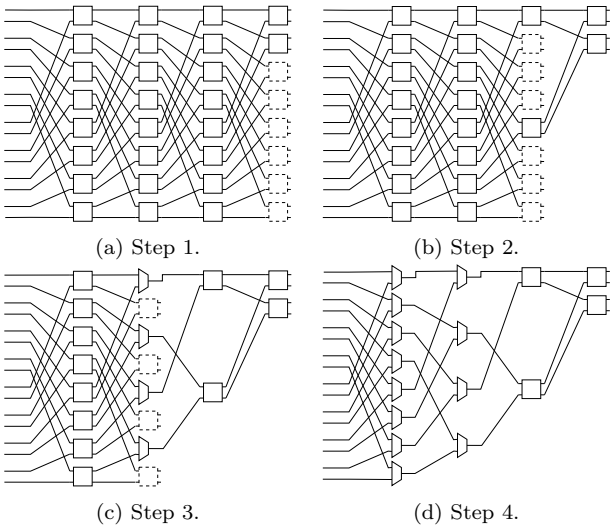
Figure 4: Omega Network. In (a), the available switch connections in the Omega switch: straight and crossover. In (b), an example of an 8x8 Omega network: input port 0 is routed to output port 4.

Figure 4a presents the possible Omega switches connections: straight and crossover. The former is selected by forcing control input to 1, the latter by forcing control input to 0. In Figure 4b we show a symmetric 8x8 Omega network. The input port 001_b is connected to the output port 100_b. To establish this connection, the Omega network uses the destination tag routing. In this method, routing is defined by the binary representation of the selected output number. The first bit is 1, so in the first stage, the switch connection is configured as crossover. The second and third bits are 0. So, the respective switches connections are configured as straight. A second routing method, called XOR-tag, uses this connection pattern and an exclusive or operation between the input and output to determine the connection path. In Figure 4b, $001_b \oplus 100_b = 101_b$, thus, the routing is performed by configuring the Omega switches to crossover, straight, and crossover, respectively.

Omega and others Banyan networks are originally **symmetric** networks [6]. In this paper we introduce a new network, which is an **asymmetric** version of Omega. In post-silicon context, designers want to tap many signals, but only few of them can be monitored at the same time. In section 3 we show how our asymmetric Omega is built, and how designers can explore many possible configurations achieving different area vs blocking rate trade-offs.

3. ASYMMETRIC OMEGA NETWORK

In post-silicon, an asymmetric network with N inputs and M outputs, with $N > M$, connects the tapped signals to the trace buffer. We propose an Omega network to be used as this asymmetric interconnection fabric. Figure 5 presents a small example of how to remove switches in order to transform a 16x16 Symmetric Omega Network in a 16x4 Asymmetric Omega Network. The resulting network is composed by 16 tapped signals (N) and 4 signals that can be traced to the buffer (M).



(a) Step 1. (b) Step 2.

(c) Step 3. (d) Step 4.

Figure 5: Process to build a 16x4 Omega network. The process starts in (a) to remove muxes from the last stage. Then, it removes muxes from the third, second, and first stage, respectively in (b), (c) and (d).

Figure 5a illustrates our starting point, a 16x16 symmetric Omega network. This network is originally composed by 32

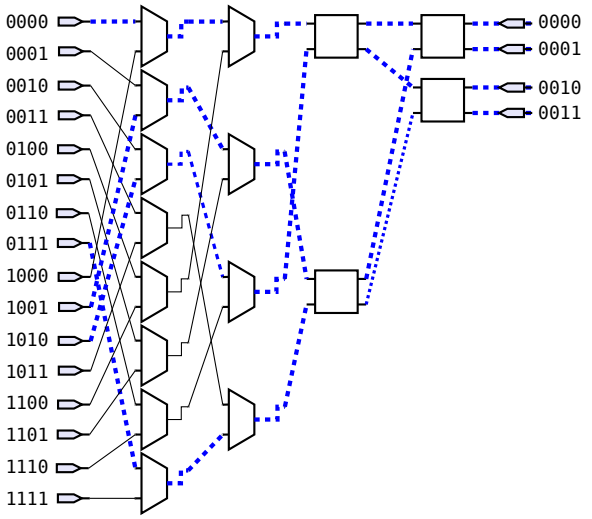


Figure 6: Routing the input set {0, 7, 9, 10} in a 16x4 Omega network.

switches. In the resulting asymmetric Omega network, 12 output signals are not used, thus we remove the switches that drive these signals. In our example, we choose to keep the first 4 output signals on the top. By removing the last 6 switches, we can backward propagate the switch deletion to the other stages. Hence, the second step is to remove the other 6 switches that are connected to the previous deleted switches, as illustrated by Figure 5b. In Figure 5c, another 4 switches are removed and 4 switches are replaced by 2x1 multiplexers. The last step, that we present in Figure 5d, is to replace 8 switches by 8 2x1 multiplexers, resulting in the final interconnection network. Considering that 1 switch is composed by 2 2x1 multiplexers, our asymmetric Omega network is built by 20 multiplexers. Using the same input set presented in Figure 3, {0, 7, 9, 10}, we show the routing for a 16x4 Omega network in Figure 6. For this input set the blocking rate is 0%.

In Algorithm 1 we show how we propagate the deletion of unused muxes. The matrix *mmx* is used to hold the available muxes. Before the process begins all muxes are available, so in line 1 the matrix is initialized with ones. We check which switches are connected to a removed switch by using the vector *pttrn* with inverse Shuffle-Pattern (line 2). N is the variable used to hold the number of the network inputs. The variable *outputs* holds the list of available network outputs. The loop in line 6 removes all outputs, and in line 9, another for loop uses the *outputs* variable to make the network outputs available. In line 13–16, we iterate from the last to the first stage checking if an entire 2x2 switch was removed. To verify if a switch was removed, we just test if both outputs of the switch (indexed by j) are equal to 0. Finally, in line 17–18, one output from each switch that is connected to the current switch is removed.

In this work, we assume that in post-silicon debug it is not important to drive a signal to a specific output port. Hence we can measure the blocking rate considering that it is possible to drive a signal to any available output port. The number of possible configurations in Omega networks for this

Algorithm 1 Remove unused muxes in an asymmetric Omega network

```
 1: mmx ← muxes matrix initialized with 1
 2: pttrn ← inverse pattern of connection
 3: N ← number of inputs
 4: outputs ← list of available outputs
 5: ls ← last stage
 6: for i ← 0 to N − 1 do
 7:     mmx[ls][i] ← 0
 8: end for
 9: for i ← 0 to SIZEOF(outputs) − 1 do
10:     out ← outputs[i]
11:     mmx[ls][out] ← 1
12: end for
13: for i ← ls to 1 do
14:     for j ← 0 to N/2 do
15:         if mmx[i][2 ∗ j] == 0 then
16:             if mmx[i][2 ∗ j + 1] == 0 then
17:                 mmx[i − 1][pttrn[2 ∗ j]] = 0
18:                 mmx[i − 1][pttrn[2 ∗ j + 1]] = 0
19:             end if
20:         end if
21:     end for
22: end for
```

scenario is $\binom{in}{out}$. For instance, there are $\approx 10^{80}$ combinations for a 4096x32 Omega network. Hence, we empirically calculate the blocking rate using the heuristic presented in Algorithm 2. We use two queues: *in_q* and *out_q*. The former holds the input signals to be routed and the latter the available network outputs. Variables *routed* and *failed* hold, respectively, the number of routed and not routed connections. Variable *miss* is used to count the number of times a different network output had been tried before a connection is established or to determine that a blockage has occurred.

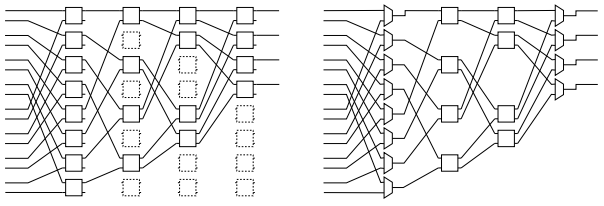
In line 8, a while loop is used to iterate until all outputs had been routed, or at least had been tried to route. In line 9, we try to route the first input and output signals in the queues. If the signals are successfully routed (line 10–14), they are removed from the queues, the routed counter increments and miss variable is restored. Otherwise (line 15–24), if the routing fails, the output port is reallocated to the end of the queue (line 16–17) and we try to route the same input to the next available output. This process repeats until all outputs have been tested (line 19). If a connection can not be established to any output, a blockage is determined (line 21). In line 26 we can calculate the blocking rate. To achieve more reliable results, we execute this algorithm several times with different input/output combinations in order to calculate the average blocking rate. In Mux Trees there is only one available output for a range of inputs, so the blocking rate is easily determined by routing a determined signal of an input set. We have designed a program to calculate the Mux Trees blocking rate. For these small networks instances, the computed blocking rate for both are the same: $\approx 28\%$.

In order to reduce the blocking rate in asymmetric Omega networks, we can disperse the available network outputs. However, less multiplexers will be removed, since the process to remove the multiplexers needs that an entire switch

Algorithm 2 Determine the blocking rate in an asymmetric Omega network

```
1: function omega_blkng_rate
2:     in_q ← queue of inputs signals
3:     out_q ← queue of outputs available
4:     routed ← 0
5:     failed ← 0
6:     miss ← 0
7:     blkng_rate ← 0
8:     while routed + failed < number of outputs do
9:         r ← ROUTE_PATH(FRONT(in_q), FRONT(out_q))
10:        if r == TRUE then
11:            POP_FRONT(in_q)
12:            POP_FRONT(out_q)
13:            routed ← routed + 1
14:            miss ← 0
15:        else
16:            PUSH_BACK(FRONT(out_q))
17:            POP_FRONT(out_q)
18:            miss ← miss + 1
19:            if miss ≥ SIZE(out_q) then
20:                POP_FRONT(in_q)
21:                failed ← failed + 1
22:                miss ← 0
23:            end if
24:        end if
25:    end while
26:    blkng_rate ← routed/number of outputs
27:    return blkng_rate
28: end function
```

is removed to propagate the deletion. The process to remove the unused multiplexers presented in Algorithm 1 works for any network output configuration. In Figures 7 and 8, we show two possible output arrangements, grouping two by two and grouping four by four. In Figure 7 we have chosen the outputs {0, 2, 4, 6}, and 28 2x1 multiplexers are used. In Figure 8, we have chosen the outputs {0, 4, 8, 12}, which increased the number of multiplexers to 44. In order to facilitate the comparison among the different Omega topologies we have added a suffix Gi, where i is the grouping arrangement number for a given configuration. So, the configuration presented in Figure 7 is Omega G2, and the configuration in Figure 8 is Omega G4. Algorithm 2 is able to handle different output configurations. So, using the same process, the computed blocking rate to the arrangement in Figure 7 is $\approx 10\%$. In Figure 8, the blocking rate is reduced to $\approx 0\%$.



(a) (b)

Figure 7: Omega (G2): 16x4 asymmetric Omega network with 28 2x1 multiplexers. In (a), the dotted switches are removed, and (b) presents the final network.

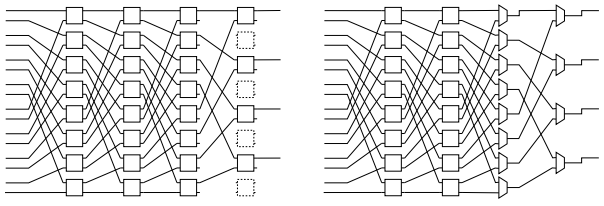
To illustrate the multiplexer dispersion impact in the blocking rate, consider the networks shown in Figures 5d, 7b, and 8b. We assume that the input list is composed by {0, 1, 5, 9}. The network in Figure 5d is able to route $0 \rightarrow 0$ and $1 \rightarrow 1$. But it can not connect $5 \rightarrow 2$ or $5 \rightarrow 3$; it can not neither connect $9 \rightarrow 2$ or $9 \rightarrow 3$. The network in Figure 7b connects $0 \rightarrow 0$, $1 \rightarrow 2$, $5 \rightarrow 4$, but it can not connect $9 \rightarrow 6$. The network shown in Figure 8b is able to connect $0 \rightarrow 0$, $1 \rightarrow 4$, $5 \rightarrow 8$ and $9 \rightarrow 12$. In this example, the blocking rate for each of these networks are, respectively 50%, 25%, and 0%.

4. RESULTS

In our experiments we compare the blocking rates and areas of Mux Tree, Omega, and Clos networks considering several sizes. For each network topology, we have generated the Verilog HDL and synthesized these codes using a commercial tool. The presented area values are in terms of NAND equivalent gate counts. In Figure 9 we present our blocking rate and area results. Each graph contains the comparison among a Mux Tree, a 3-stage rearrangeable Clos network (Clos *R3*), and the Omega network with different output arrangements. Omega *G1* means that the network output is grouped in the top, as shown in Figure 5d. For the other Omega network results, the number identifies how the outputs are indexed. For example, the outputs for Omega *G2* are {0, 2, 4, 6, ...} (Figure 7b), and the outputs for Omega *G4* are {0, 4, 8, 12, ...} (Figure 8b).

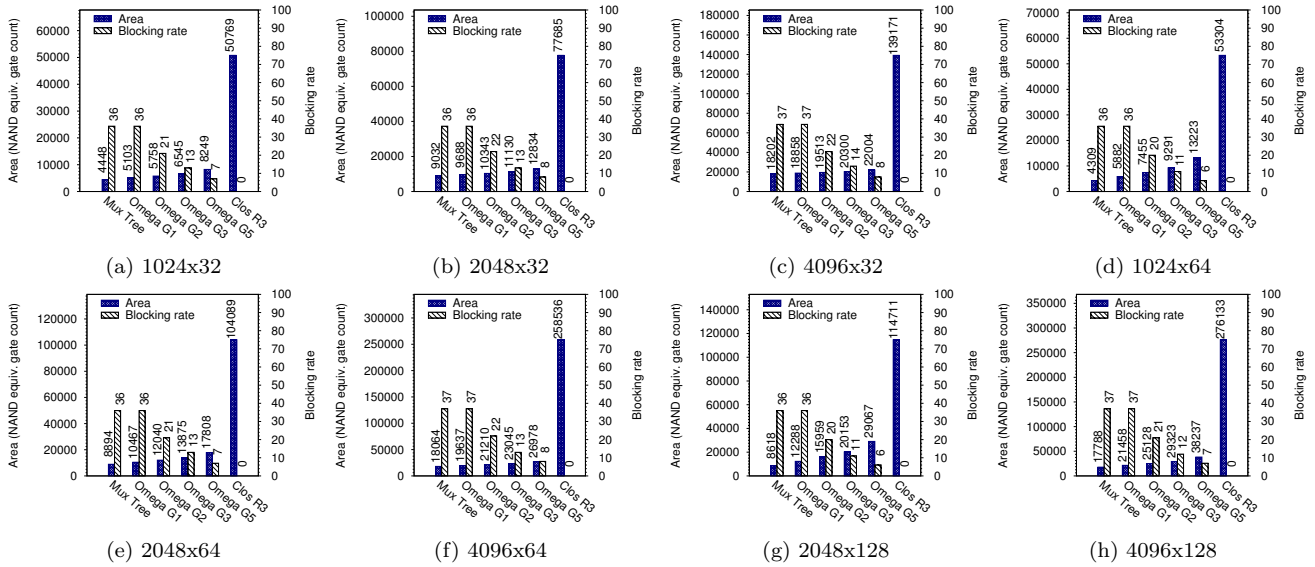
Each result graph contains 2 y-axis. The left y-axis is the area computed by synthesis in terms of *NAND equivalent gate count*. The y-axis on the right side is the blocking rate computed using Algorithm 2. The blocking rate for Clos is 0, once it is a non blocking network. The number of samples has been determined by visualizing a graph of the blocking rate vs the number of samples. Near to 100,000 samples, the blocking rate ranges up to 0.001, and this range is ≈ 0 near 1 million samples. So, in this experiment we have executed our algorithm 1 million times for each network size to determine the blocking rate.

Our results show Omega network in 4 different output arrangements: *G1*, *G2*, *G3*, *G5*. We do not show the results of Omega *G4* because their blocking rate is practically the same of Omega *G3*. In Figure 9c we show that Omega network achieves ≈ 4.6 times less blocking in 4,096x32 network, at the price of increasing the area overhead in $\approx 21\%$, compared to a same size Mux Tree.



(a) (b)

Figure 8: Omega (G4): 16x4 Omega Network with 44 2x1 multiplexers. In (a), the dotted switches are removed, and (b) presents the final network.



(a) 1024x32 (b) 2048x32 (c) 4096x32 (d) 1024x64

(e) 2048x64 (f) 4096x64 (g) 2048x128 (h) 4096x128

Figure 9: Comparison of Mux Tree, Omega and Clos networks for several input and output sizes.

The designer can also choose to have less overhead but, still reducing the Mux Tree blocking rate. For example, 4,096x32 Omega *G3* reduces the blocking rate in ≈ 2.6 times compared to Mux Tree. It increases the area overhead in $\approx 12\%$, which is less than the Omega *G5* overhead. Another network with the same size, Omega *G2*, shows just a small $\approx 7\%$ overhead, and also reduces the blocking rate from $\approx 36\%$ to $\approx 22\%$.

In Figure 9b, the area overhead of the Omega *G5* is $\approx 42\%$ compared to the Mux Tree, and in Figure 9a, the area overhead of the Omega *G5* is $\approx 85\%$. In Figures 9b and 9a, the blocking rate is reduced by ≈ 4.5 and ≈ 5.1 times respectively. Hence, the greater the number of inputs compared to the number of outputs, the lower is the overhead in Omega, due the number of stages in each network.

5. CONCLUSION AND FUTURE WORK

In this work we have proposed the use of Omega asymmetric networks as interconnection fabrics for post-silicon debug. We have implemented the algorithm to remove multiplexers from the original Omega network to build the asymmetric Omega network. We have successfully demonstrated that asymmetric Omega networks can be used in post-silicon debug to decrease the signals blocking rate, increasing observability. Moreover, we show that the higher is the difference between input and output sizes, more we can benefit from decreasing area overhead and blocking rates.

As future work, we will implement tools to automate the asymmetric Omega network generation in Verilog HDL. We also intent to explore Omega network topologies with different radix sizes.

Acknowledgments

We would like to thank CAPES, FAPEMIG, FUNARBE and UFV for the financial support.

6. REFERENCES

[1] M. Abramovici. In-system silicon validation and debug. *Design Test of Computers, IEEE*, 25(3):216–223, May 2008.

[2] M. Abramovici, P. Bradley, K. Dwarakanath, P. Levin, G. Memmi, and D. Miller. A reconfigurable design-for-debug infrastructure for socs. In *Design Automation Conference, 2006 43rd ACM/IEEE*, pages 7–12, 2006.

[3] K. Basu and P. Mishra. Rats: Restoration-aware trace signal selection for post-silicon validation. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 21(4):605–613, April 2013.

[4] K.-H. Chang, I. Markov, and V. Bertacco. Automating post-silicon debugging and repair. In *Computer-Aided Design, 2007. ICCAD 2007. IEEE/ACM International Conference on*, pages 91–98, Nov 2007.

[5] C. Clos. A study of non-blocking switching networks. *Bell System Technical Journal*, 32(2):406–424, 1953.

[6] W. Dally and B. Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[7] A. DeOrio, D. S. Khudia, and V. Bertacco. Post-silicon bug diagnosis with inconsistent executions. In *Proceedings of the International Conference on Computer-Aided Design*, ICCAD '11, pages 755–761, Piscataway, NJ, USA, 2011. IEEE Press.

[8] A. Duguid. Structural properties of switching networks. Technical Report Progress Report BTL-7, 1959.

[9] R. Ferreira, J. Vendramini, and M. Nacif. Dynamic reconfigurable multicast interconnections by using radix-4 multistage networks in fpga. In *Industrial Informatics (INDIN), 2011 9th IEEE International Conference on*, pages 810–815, July 2011.

[10] L. R. Goke and G. J. Lipovski. Banyan networks for partitioning multiprocessor systems. In *Proceedings of the 1st Annual Symposium on Computer Architecture*,

ISCA '73, pages 21–28, New York, NY, USA, 1973. ACM.

[11] E. Hung and S. Wilton. Scalable signal selection for post-silicon debug. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 21(6):1103–1115, June 2013.

[12] E. Hung and S. J. E. Wilton. On evaluating signal selection algorithms for post-silicon debug. In *Quality Electronic Design (ISQED), 2011 12th International Symposium on*, pages 1–7, March 2011.

[13] F. K. Hwang. *The Mathematical Theory of Nonblocking Switching Networks (Series on Applied Mathematics)*. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 2004.

[14] H. Ko and N. Nicolici. Algorithms for state restoration and trace-signal selection for data acquisition in silicon debug. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 28(2):285–297, Feb 2009.

[15] D. Lawrie. Access and alignment of data in an array processor. *Computers, IEEE Transactions on*, C-24(12):1145–1155, Dec 1975.

[16] S. Mitra, S. Seshia, and N. Nicolici. Post-silicon validation opportunities, challenges and recent advances. In *Design Automation Conference (DAC), 2010 47th ACM/IEEE*, pages 12–17, June 2010.

[17] N. Nicolici and H. Ko. Design-for-debug for post-silicon validation: Can high-level descriptions help? In *High Level Design Validation and Test Workshop, 2009. HLDVT 2009. IEEE International*, pages 172–175, Nov 2009.

[18] J. H. Patel. Processor-memory interconnections for multiprocessors. In *Proceedings of the 6th Annual Symposium on Computer Architecture*, ISCA '79, pages 168–177, New York, NY, USA, 1979. ACM.

[19] P. Rosinger, B. Al-Hashimi, and N. Nicolici. Scan architecture with mutually exclusive scan segment activation for shift- and capture-power reduction. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 23(7):1142–1153, July 2004.

[20] S. Sibal and J. Zhang. On a class of banyan networks and tandem banyan switching fabrics. *Communications, IEEE Transactions on*, 43(7):2231–2240, Jul 1995.

[21] H. Siegel. Interconnection networks for simd machines. *Computer*, 12(6):57–65, June 1979.

[22] D. Slepian. Two theorems on a particular crossbar switching network. unpublished manuscript, 1952.

[23] H. S. Stone. Parallel processing with the perfect shuffle. *Computers, IEEE Transactions on*, C-20(2):153–161, Feb 1971.

[24] B. Vermeulen and S. Goel. Design for debug: catching design errors in digital chips. *Design Test of Computers, IEEE*, 19(3):35–43, May 2002.

[25] B. Vermeulen and K. Goossens. *Debugging Systems-on-Chip: Communication-centric and Abstraction-based Techniques*. Embedded Systems. Springer, 2014.