Study of GALS based FPGA Architecture Using CAD Tool

Savitha Devaraj
Department of Electronics Engineering
Lokmanya Tilak College of Engineering, Navi Mumbai, Maharashtra, India

Neeta Gargote
Department of Electronics Engineering
Lokmanya Tilak College of Engineering, NaviMumbai, Maharashtra , India

Swati Choudhary
Department of Electronics Engineering
Lokmanya Tilak College of Engineering, NaviMumbai, Maharashtra , India

Abstract- Globally asynchronous and locally synchronous (GALS) paradigm is to partition a system design in decoupled clock-independent modules. GALS design eliminates the issue of using global clock and decrease area overhead when compared to such a globally asynchronous design methodology. Routing delays dominate other delays in current FPGA designs. We have proposed a novel Globally Asynchronous Locally Synchronous (GALS) FPGA architecture called the GAPLA to deal with this problem. In the GAPLA architecture, the FPGA area is divided into locally synchronous blocks and the communications between them arc through asynchronous I/O interfaces. Automatic design flow is developed for the GAPLA architecture. Starting from behavioral description, a design is partitioned into smaller modules and fit to GAPLA synchronous blocks. The asynchronous communications between modules are then synthesized. The CAD flow is parameterized in modeling the GAPLA architecture. By manipulating the parameters, we could study different factors of the designed GAPLA architecture. Experimental results demonstrate the efficacy of our architecture.

Keywords –Globally asynchronous and locally synchronous, Programmable Logic Array Architecture, Interconnect delay, CAD Tool, Asynchronous Circuits.

I. INTRODUCTION

Many of today's VLSI design are based on globally synchronous design. The new SOC design faces the challenge of of distributing a high-speed low skew local clock in a large die. Proper clock distribution needs numerous buffers and a carefully designed clock tree which introduces considerable area and power overhead. Asynchronous design methodologies can eliminate such overheads by removing the clock signal from design. The properly designed asynchronous systems consume less power and may be faster than the synchronous counterparts. The lack of reliable tools for asynchronous circuits makes it even more unacceptable for asynchronous designs to substitute the current synchronous VLSI design methods.

Routing delays have become a major roadblock for FPGA performance and the situation will only be worse when technology continues to scale and FPGA chips continue to grow large. Long routings not only increase the wire delay itself, but also need to go through more routing switch boxes, making the situation worse. For example, the Xilinx VirtexII xc2v8000 FPGA has a corner to corner interconnect delay of around 15ns [l]. Different approaches of solving this problem have been proposed. [2] and [3] pipelines the long interconnect delay and [1] proposes a synthesis flow to allow the long interconnect to run for several clock cycles. In those approaches, interconnects arc treated as circuit components instead of conventional wires. The interconnect retiming registers can be very expensive in area which make their FPGA size several times bigger than conventional FPGAs. GALS systems can be seen as synchronous logic blocks wrapped in asynchronous I/O interfaces. Interconnects inside each block is short and fast, which allows the synchronous logics to run at higher speed. Interconnects between synchronous logic blocks have longer delay, but they will not affect the clock speed of the logic blocks and only come into picture when there are communication between synchronous blocks. Therefore, performance improvement could be expected.

An automatic design flow for the GAPLA architecture has also been developed [5]. Starting from a behavioral circuit description, a design is first partitioned into smaller modules where each module can fit into one synchronous block (asynchronous Island). Then the control sequences for asynchronous communications between modules are generated and put into each module. After that a coarse-grained placer is used to place the modules to the GAPLA chip space and connections between islands are routed. Each module is then synthesized by calling existing FPGA tools. The CAD flow is designed as an auxiliary tool to study the GAPLA architecture. It is parameterized in modeling the architecture. Therefore, by changing the values of the architectural parameters, we could study the effect of these factors. In this paper, we report the results of our study of the GAPLA architecture using the above CAD flow.

II. THE GAPLA ARCHITECTURE

Figure 1 gives a basic building of the GAPLA architecture called an asynchronous island. The GAPLA architecture is a mesh of asynchronous islands. Each island contains a synchronous logic block and asynchronous wrapper. Each wrapper contains a local clock generator and I/O port controllers. The structure of the synchronous logic block can be any of the conventional FPGA structures. But the size of each synchronous logic block must be big enough to implement reasonable functions.

In our design, we adopt the Virtex II logic: array structure. The 4 dock signals generated by the clock generators are all distributed into the synchronous logic block. Logic tiles inside the synchronous block can freely choose to connect to one of these clock signals. The logics (controlled by the same clock signal are called a clock domain. Thus, the size and shape of each clock domain of the GAPLA architecture is programmable within the limit of a synchronous logic block. The routing resources between asynchronous islands contain horizontal and vertical routing channels for both data and handshaking control signals. Adjacent, asynchronous islands are also directly connected to enable fast communication. Please refer to [4] for details of the architecture design.

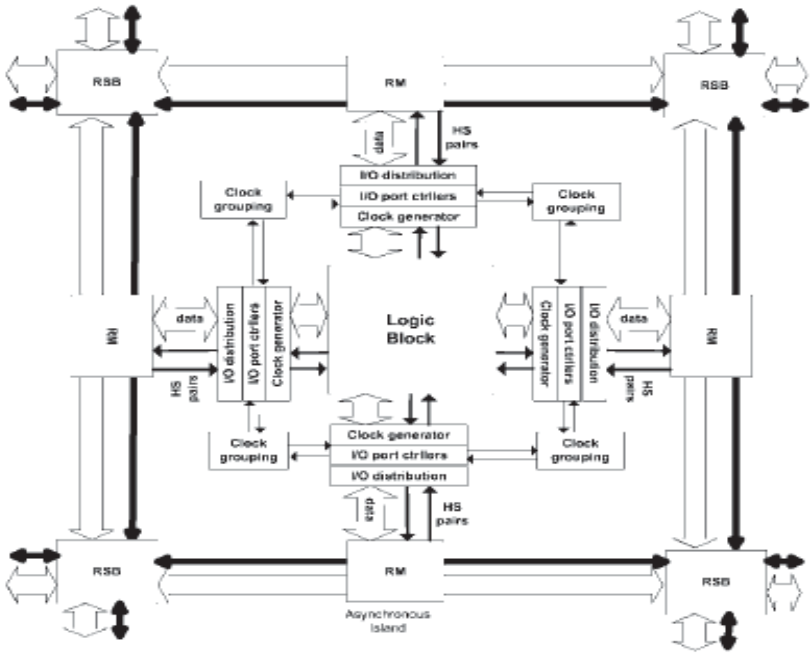

Figure 1. BLOCK DIAGRAM OF GAPLA ARCHITECTURE

The execution time of an application mapped on the GALS based FPGA architecture consists of two parts: computation time and communication time. Computation time is the time for synchronous logic blocks to finish the programmed computations. Communication time is the time consumed by the asynchronous communications between logic blocks. The best performance of the GAPLA architecture is the best trade off between communication time and computation time. The architectural parameters which affect these trade offs are as follows.

1. The size of a synchronous logic block. A large logic block means more operations can be put into one clock domain, which generally decreases the local clock speed and increases the overall computation time. But the

communication time will decrease since more communications will be done synchronous inside a clock domain.

2. The number of asynchronous I/O ports for each asynchronous island. Increasing the number of I/Os will increase the area overhead but will lessen the I/O constraints during the application partitioning process which could improve the logic usage of the logic block and performance. Since our design of the I/O port controllers are very simple and have small layout areas, we can afford to add more I/O ports as long as it will benefit the system performance.
3. The number of global routing channels. This factor not only affect the routability of GAPLA architecture and also affect the performance since the routing might be congested and need to detour if the routing resource is limited which increases communication time.

III. CAD FLOW

The CAD flow is developed to automatically implement designs in the GALS based FPGA. It is also used to investigate the architecture design. As mentioned above, performance of the GAPLA is affected by three groups of parameters. The CAD flow models the architecture based on them too. By given these parameters different values, we could compare the implementation results of a set of benchmarks and get to know the effect of these parameters.

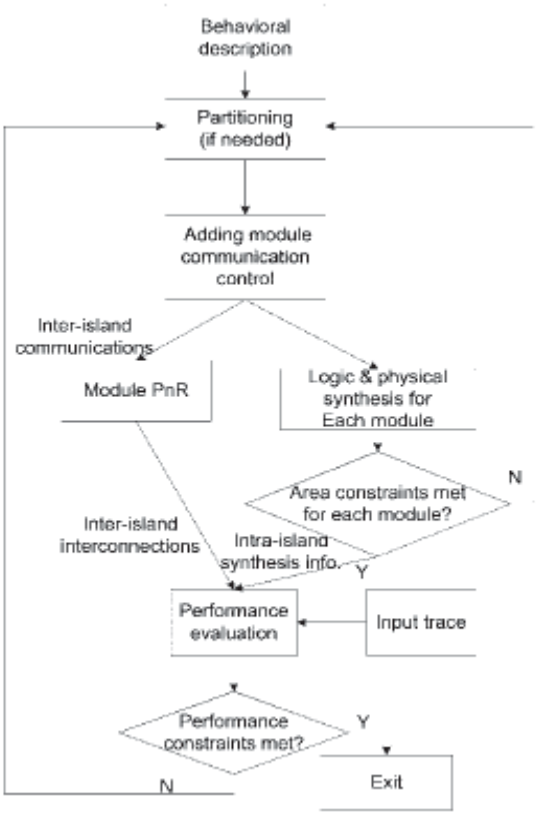

Figure 2. CAD Flow for GAPLA Architecture

Figure 2 shows **a** block diagram of the overall CAD design flow. The grey boxes are modules we proposed and white boxes arc modules using existing CAD tools. Partitioning is required first if the design is bigger than a given asynchronous island. Controls for asynchronous communications are then added to each module to ensure functional correctness. After that, each module is synthesized and place-and-routed using existing FPGA design tools. Also, all the modules are fed into a coarse-grained placer and router which place each module to the GAPLA chip space and finish the global routing between modules. Finally, a simulation model of the design implemented on the GAPLA

architecture is formed for performance evaluation. If all the performance constraints are met, the design is accomplished. In the following subsections, we briefly explain each functional module in the following.

3.1 Partitioning-

When partitioning a design into modules, we try to minimize the communication time between partitioned modules. Also, partitioning is conducted under two constraints: the area constraint, where the area of each module must be less than given area of synchronous block; the I/O constraint where the number of input/output ports must be less than given input/output ports per asynchronous island.

To calculate the asynchronous communication time, we first build a CDFG representation of the design and edges in the CDFG are given communication weights. The communication weight of an edge consists of two parts: its "communication frequency" and its "length". The communication frequency of an edge is defined as:

$$F(e)=\begin{cases} 1 \; initial\; value \\ \dfrac{f(E)}{m} \; if\; e\; is\; inside\; a\; branch, m \\ \quad is\; the\; number\; of\; branches \\ f(e)\times n \; if\; e\; is\; inside\; a\; loop, n\; is\; the \\ \quad loop\; count \end{cases}$$

The length of an edge l (e) is defined as follows: first, we do an As-Soon-As-Possible scheduling to the CDFG, and the control step assigned to node is denoted as cs (i). Then:

$$l(eij)= cs(i)- cs(j)$$

l(e) is used to localize all the interconnects. Therefore, it should be more likely to mapped to asynchronous communication channel. The final weight of an edge for the partitioning is a weighted combination of factors f(e) and l(e):

$$w(e)= \alpha \times \frac{f(e)}{max(f(e))}+\beta \times \frac{min[l(e)]}{l(e)}$$

α and β are user defined coefficient and α+β = 1, max $(f(e))$ is the maximum communication frequency of all edges, min(l(e)) is the minimum length of all edges. The GAPLA architecture allows multiple I/O ports of the same clock domain to be active at the same time. In this case, the time overhead for these asynchronous communications overlaps which leads to performance benefits. The partitioning algorithm should take advantage of this and partitions the system in a way that the overlap among asynchronous communications is maximized. As partitioning is done before the actual system timing information, a proper estimation is required. We use the factor N, the number of control steps where the data transmissions across partitions are required to represent this. Thus the overall cost function of the partitioning algorithms is formed as:

$$F= N \times \sum w(eij),$$

Where node i, j belong to different partitions. A simulated annealing algorithm is used as the partitioning algorithm. After one iteration of partitioning, each partition is synthesized separately. The partition that meets the area and I/O constraints are treated as an individual partition in the final result. The partitions that will violate the area and I/O constraints are further partitioned using above procedure.

3.2 Asynchronous Communication Control-

To add asynchronous communications, we need to provide a proper sequence of control values for the control signals of the corresponding asynchronous I/O port controllers. The asynchronous handshaking process is automatically managed by the built-in asynchronous FSMs inside the I/O port controllers based on these signals. Therefore, we need to know at what cycle an asynchronous communication should take place. To gather this information, operations inside each module are scheduled first.

We use an As-Soon-As-Possible (ASAP) algorithm to schedule each module in order to get the best performance. Because of the architecture design, the inter-module asynchronous communications blocks both the sender's and receiver's operations. Thus deadlock situation could occur after scheduling. Deadlock occurs when two or more communicating processes waiting for each others data in order to continue the execution. It can be solved by constructing Communication Dependency Graph (CDG). A CDG contains all the communication nodes of the system. And a directed arch between two communication nodes in a CDG iff there is a sequential dependency between the two nodes in any of them. The dependencies of communication nodes in the constructed CDG are enforced in all the processes of the system by adding dummy control edges to the processes. After adding these edges, an ASAP scheduling algorithm is used to schedule each process and deadlock is avoided.

The outputs of the scheduling are cycle-accurate descriptions for each module. After scheduling, we know exactly at what cycle data needs to be sent to or received from other modules. Therefore, the control signals for the asynchronous communications can be added accordingly.

3.3 Module Placement and Routing-

The placer will place each module to an asynchronous island. The optimization goal during placement is to minimize the total communication cost between modules. Since each communication edge carries a communication weight, as explained before, the goal of the placer is thus to

$$\sum_{min} f(eij) \times Dij$$

Where node i, j belongs to different modules. $D_{,}$, is the distance between the clock domains where node i , j are placed.

Since the effect of global routing on the system performance is greatly reduced a simple and fast line-search based router is used to route the asynchronous communications for the global routing channels of GAPLA FPGA. The inter-module routing resources are represented by two matrices Horizontal Routing Sources (HRS) and Vertical Routing Sources (VRS). The two matrices can be initialized at runtime to model different configurations of the GAPLA architecture. Nets are picked up once at **a** time from the netlist and routed. For multiple-terminal nets, the two terminals with the longest Manhattan distance are routed first.

3.4 Performance Simulation-

Simulation is used to estimate the performance of the design implemented on GAPLA architecture. From the synthesis results of each module, the information about the clock frequency for each module is obtained. From the module placer and router, the information about the placement position of each module and the interconnect delays between modules are obtained. This information together with the cycle-accurate VHDL descriptions of each module is fed into our VHDL simulation model of the GAPLA. The GAPLA simulation model contains the models for the pausible local clock generators, the I/O port controllers and the inter-island routing channels. The local clock generators are programmed to the corresponding clock frequencies of the modules. The modules are wrapped by the asynchronous interfaces and connected through the routing channels, composing a simulation model of the circuit implementation. Input traces are then read to the model and the performance can be observed.

IV. ARCHITECTURE PARAMETERS

4.1 STUDYING METHODOLOGY

To study these parameters, we need to implement a set of benchmarks on different configurations of them. Our benchmark set consists of 12 synthetic: benchmarks generated by TGFF [10]. TGFF generates Directed Acyclic Graphs (DAGS) with different number of nodes and connectivity intensities. We then assign each node an arithmetic operation and generate a VHDL description code from each graph as a benchmark. Thus, all the benchmarks are computation intensive ones with few or no control flow which are the cases for most FPGA applications. To exclude the factor of multipliers, only addition and subtraction operations are assigned. Table 1 gives the statistics for the benchmark set and their implementation results on a Virtcx II FPGA

	Nodes	Area (CLBs)	Clock (ns)	Worst wire dly(ns)	Exec. Time (ns)
ex1	150	280	7.625	4.537	99.1
ex2	400	760	9.682	6.617	203.32
ex3	500	972	11.7	8.473	386
ex4	500	856	6.51	4.023	266.9
ex5	800	1530	12.98	10.25	467.3
ex6	1000	2000	15.82	12.4	395.5
ex7	1500	2887	17.29	14.47	1072
ex8	1800	3264	14.78	12.12	1257
ex9	2000	3836	16.43	14.19	1610
ex10	2500	4492	17.02	14.20	1634
ex11	3000	5820	20.14	17.15	2296
ex12	4000	7537	19.39	17.05	3491

Table 1 Implementation results on Virtex II FPGA

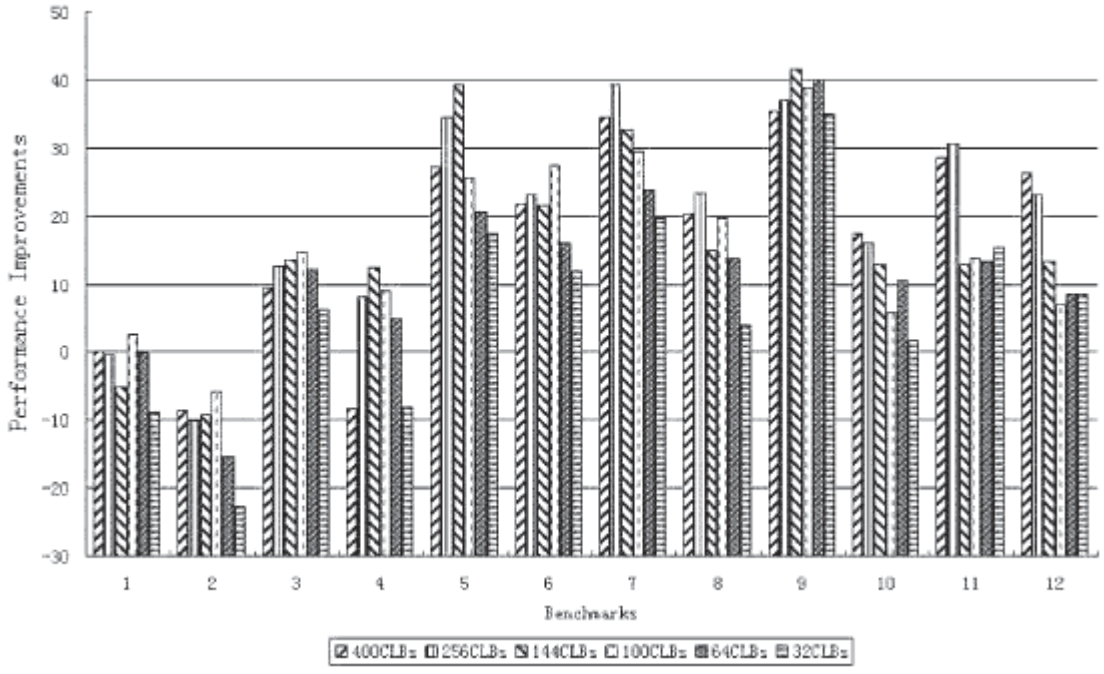

Figure 3 The effect of logic block size on performance of the GAPLA FPGA

It is time forbidden to study the three parameters at the same time. Therefore, the parameters are determined one by one. The size of ***a*** logic block is studied first since it is the single most important parameters for the GAPLA architecture. To do that the number of I/Os per clock domain and the routing capacities between modules are assumed to be infinite. Thus, the I/O constraints during partitioning process are lifted and the routing process is not necessary since the routing distance can be estimated as the Manhattan distance between terminals. After the size of a logic block is chosen, the number of I/Os per clock domain can be determined by looking into the partitioning results since, as explained before, the number of I/Os per clock domain could be fairly large without incurring huge area overhead. After that, the global routing capacities are determined by running the router on the after-placement benchmarks with

the first two parameters fixed on the GAPLA FPGA. The experimental results are explained in the following Subsections.

4.2 Size of a Synchronous Logic Block-

For every benchmark, 6 different sizes for a logic block are tried. They are 36, 64, 100, 144, 256, and 400 (in terms of CLBs). The performance improvements of all the 12 benchmarks are summarized in Figure 3. From the results, the GAPLA FPGA with logic block size 256 CLBs delivers the biggest performance improvement or1 average for all the 12 benchmarks. Thus, the size of a logic block is chosen to be 256 CLBs. If the last 8 benchmarks are considered, the average performance improvements could be more than 28%.

4.3 Routing Between Asynchronous Islands

Previous experiments assume that the global asynchronous routing channels are sufficient and therefore each net can use the shorted connection route. In this subsection, We study the impact of asynchronous routing channels on the performance of the GAPLA FPGA architecture. The first two sets of architectural parameters are fixed, namely 256CLBs per logic block and each asynchronous wrapper contains 8 input ports and 8 output ports and 256 data wires. After routing, the asynchronous communication time is more accurate based on the actual routing path. The experiments arc conducted for different routing configurations (in terms of number of asynchronous communication channels). The experimental results are given in Table 4. In the table,"P.I" represents the performance improvements compared to the synchronous implementation on a Virtex II FPGA. The entries marked with"-" mean that the routing is incomplete under the corresponding configuration.

	16	18	20	24	28	32
	P.I.	P.I.	P.I.	P.I.	P.I.	P.I.
ex1	-0.4	-0.4	-0.4	-0.4	-0.4	-0.4
ex2	-10	-10	-10	-10	-10	-10
ex3	12.6	12.6	12.6	12.6	12.6	12.6
ex4	8.1	8.1	8.1	8.1	8.1	8.1
ex5	34.4	34.4	34.4	34.4	34.4	34.4
ex6	21.5	21.5	21.5	21.5	21.5	21.5
ex7	39.2	39.2	39.2	39.2	39.2	39.2
ex8	21.0	21.0	21.0	21.0	21.0	21.0
ex9	-	-	33.5	33.8	37.1	37.1
ex10	-	11.3	11.6	12.6	13.8	13.8
ex11	26.6	27.5	29.3	29.3	29.3	29.3
ex12	-	-	13.0	15.6	16.5	16.5
Avg.	-	-	17.8	18.1	18.6	18.6

Table 2 Performance evaluation after routing

The results show that if the global asynchronous routing structure has less than 20 channels, some of the benchmarks cannot be successfully routed. And increasing the number of asynchronous tracks only has slight impact on the system performance (less than 1 percent on average). And increasing the number of global asynchronous routing channels will greatly increase the area overhead of the GAPLA architecture, therefore, we choose 20 channels for the global routing structure. We also assume that on average, each asynchronous channel has 16 bits of data wires. Therefore, the total number of global data wires is 320.

V.CONCLUSION

In this paper, We studied three critical architectural parameters of the GALS based FPGA system namely the size of each synchronous logic block, the number of I/Os per asynchronous island and the number of routing channels between islands, using the parameterized CAD tools. From the experimental results, the following values arc chosen for these parameters: 256 CLBs per logic block, 8 input ports, 8output ports, 256 data wires per asynchronous wrapper, and 20 global routing channels and 320 global data wires. The area overhead of the GAPLA architecture using this configuration is around 19.9%. The average performance improvement for all the benchmarks is 17.8%. If

only the large benchmarks, which arc suitable for the GAPLA FPGA; are considered, the average performance improvement on the last 8 benchmarks is 25.4%.

REFERENCES

[1] Jason Cong, Y. Fan, et al. Architecture and synthesis for multi-cycle communications. In proc, Int, Symp. Physical Design, Apr. 2003

[2] William Tsu, Andre Dehon, et al. High- Speed, hierarchial synchronous reconfigurable array. In Proc. Int. Symp. Field Programmable Gate Arrays 1999.

[3] Akshay Sharma, Katherine Compton, et al, Exploration of Pipelined FPGA interconnect structures. In Proc. Int. Symp. Field Programmable Logic and Applications, 2005.

[4] Xin Jia, Ranga Vemuri. **A** novel asynchronous FPGA architecture design and its performance evaluation. *.Proc. Iat. Workshop Field Programmable Logic and Applications,* 2005.

[5] Xin Jia, Ranga Vemuri. CAD tools for a globally asynchronous 1ocally synchronous FPGA architecture. *.Proc. 19th Int. conf, VLSI Design India,* 2006.

[6] *S* . Hauck, S. Burns, G.Borriello, C. Ebeling. A FPGA for irnplernenting asynchronous circuits. In *IEEE Design* & *Test of Cornyuters* Vol. 11, *No.* 3 Fall 1994.

[7] R.obert Payne. Self-tirned FPGA svstems. In Int .*Workshop Field Programmable Logic and Applications* 1995.

[8] T. Q. Ho, J. B. Rigaud, M. Renaudin, L. Fesquet et R. Rolland, "Implementing Asynchronous Circuits on LUT Based FPGAs", *Proc. of the Field-Programmable Logic and Applications, Reconfigurable Computing Is Going Mainstream, 12th Int.Conference*, FPL 2002, Montpellier, France, September 2-4, 2002.

[9] R. Ginosar, "Fourteen ways to fool your Synchronizer", *Proc. of the Ninth Int.Symp.on Research in Asynchronous Circuits and Systems, ASYNC'03*, Vancouver, Canada, 2003