Automated Conversion from a LUT-based FPGA to a LUT-based MPGA with fast Turnaround Time

Francisco-Javier Veredas[†] [‡], Michael Scheppler[†], Hans-Joerg Pfleiderer[‡]
[†]Infineon Technologies AG, D-81699,Munich, Germany
[‡]Microelectronics Department, University of Ulm, D-89081, Ulm, Germany

Abstract

Mask Programmable Gate Arrays (MPGAs) see a growing importance because of the increase of design cost and turnaround times in ultra-deep submicron technologies which mostly impact ASICs. Several design methodologies have been proposed in recent years for converting an evaluated Field-Programmable Gate-Array (FPGA) prototype-design into an MPGA. An automatic conversion flow is essential to success. In this paper, we present a conversion flow for a Look-up Table-based (LUT-based) MPGA without applying re-synthesis but preserving the gate-level netlist and reusing the placement. The resulting flow has a special routing tool and buffer insertion algorithm for timing integrity. The experimental investigations use a commercial FPGA and industrial benchmarks.

1. Introduction

FPGAs have the advantage of allowing short development cycles. This is mostly due to the device configuration at customer site rather than device production in a factory. With this benefit they threat ASIC business which traditionally meant the implementation of an individual customer chip by widely reusing IP.

On the other hand, FPGAs are cost prohibitive in mid-volume applications (100k to 3 million units in a 90nm process technology) and impracticable for large-volume production (greater than 3 millions) [13]. For reasons of cost reduction and low power dissipation MPGAs have high potential for mid-volumes. In large-volumes (ASSP market), embedded MPGAs may become important because they offer increased flexibility at moderate cost of non-recurring engineering (e.g. mask cost) and fast turnaround time.
In ultra-deep sub-micron, the parasitic technologies aspects (e.g. leakage current) strongly affect physical design and substantially increase the effort for verification and yield optimisation. On the other hand system level design is absorbed with the task of increasing system complexity. A method of bridging this growing gap is to hide the technological aspects of a device with predefined structures, so the silicon design complexity is low. While FPGAs require total predefinition comprising both logic and interconnect, it is possible to differentiate the MPGAs into three classes. Depending on the complexity of the basic component-cell you can distinguish: CMOS-based MPGAs with elementary gate structures [7, 11], Cell-based MPGAs (also called structured ASICs) [15, 12, 2] and LUT-based MPGAs [3, 10]. There are CMOS-based MPGAs products since the 80's. Cell-based MPGAs are on their way into products today, while LUT-based MPGAs still are under investigation.
Our proposed methodology to design MPGAs consist in prototyping the design with a FPGA framework and then convert the design to a MPGA. Existing conversions use a target architecture with elementary cell structures because this offers high density. The downside of this approach is the need of re-synthesis and complete place and route. Due to the physical difficulties a significant risk of corrupted timing integrity is taken. In this paper a target architecture is taken which preserves the gate-level structure of the FPGA and in addition has predefined routing resources. It will be shown that the conversion flow is rather simple and high design security can be obtained.
The paper is organised as follows: Section 2 gives an overview of the LUT-based MPGA architecture. Section 3 explains the conversion design flow. Section 4 describes the experimental methodology and shows the results. Section 5 concludes the paper.

2. LUT-based Programmable Architectures

There are two major players in the FPGA market: Xilinx Inc. and Altera Inc. Both use architectures which are array based and have logic clusters at the lower hierarchy level. Both companies have two families one for high performance and high complexity another for low cost and

3-9810801-0-6/DATE06 © 2006 EDAA

higher volume. Both companies follow the trend of embedding more and more optimised macros into the array (e.g. DSP, transceivers, ...).
We used the Xilinx Virtex-II Pro [1] device for our studies. The conversion design flow presented in this paper can be easily adapted to other Xilinx family devices as e.g. the Xilinx Spartan-III. The use of Altera FPGA devices would require extra modifications in the conversion flow (e.g. dedicated placement).
We named the LUT-based MPGA presented in this paper Zelix. The Zelix MPGA has the same gate-level logic elements as the Xilinx Virtex-II Pro. For the sake of simplification the embedded hardware blocks (as 18 Kb RAM memory modules) are not yet implemented in the Zelix, i.e. the studies focus on the logic.

2.1. Xilinx Virtex-II Pro FPGA Architecture

The Xilinx Virtex-II Pro is a 130nm CMOS nine-layer (copper) FPGA device. Virtex-II Pro has an island style architecture. It consists of a two-dimensional array of Configurable Logic Blocks (CLBs) and programmable interconnect resources. Each CLB is a cluster of four identical sub-blocks called Slice. The Slice consist of two four-input LUTs, two FFs, gates (two AND2, one OR2, two XOR2), multiplexors, inverters and buffers. The LUTs are used to map four-input boolean logic. The gates are used to implement special functions as carry chains. Four multiplexors are used as mapped multiplexors (after synthesis) and the other multiplexors are used as configurable routing switches (i.e. to do programmable routing inside the Slice). The inverters and the buffers are needed to implement the different clock types allowed in the FPGA. The four slices of one CLB share a common programmable crossbar for doing the input/ output signal communication with other CLBs in the array. This communication is realised with a switch matrix. The switch matrix routes a signal to north, south, east or west direction. The topology of the switch matrix in the Virtex-II Pro is a special disjoint type. A CLB together with its associated switch matrix is called a Tile. The input/output programmable crossbar is also used for doing local signal communication between slices.
The programmable interconnect is responsible for connecting the inputs of the logic cells and the outputs together. The Virtex-II Pro has two hierarchical levels: local interconnect and global interconnect. The local interconnect is the programmable crossbar described before. The global interconnect consist of vertical and horizontal routing channels. These are classes of segment lengths in the global interconnect. A wire in a channel can expand two Tiles (double lines), six Tiles (hex lines) or all the row/column (long lines). The Virtex-II Pro has 40 double lines, 120 hex lines and 24 long lines in a channel. All these lines are routed within the switch matrix. There is a special type of wire that can connect points of two CLB without passing through the switch matrix, so called direct neighbour lines. The direct neighbour lines connect adjacent CLBs. The lines go directly from an output crossbar to an input crossbar.
The programmable resources are controlled by latches.
The routing resources in a FPGA consume about 88% of the total area [9]. To reduce the area penalty of the programmability, FPGA vendors use a high number of metal layers and the latest process technology in their device design.

2.2. Zelix MPGA architecture

The use of MPGAs is addressed to reduce the high penalty area of FPGAs, but losing user-programmability for fab-programmability. In our MPGA, all the mask layers that define the circuit devices are pre-defined, except those that specify the final metal layers. These metal layers are customised to connect the logic gates, thereby implementing the desired circuit. The programmable interconnect resources (i.e. SRAM cells) of an FPGA are changed with metal layers and vias. Also, the configuration of logic (e.g. LUTs) is done with metal layers (connecting the logic to VDD or VSS). The Zelix MPGA consists of a two-dimensional array of Mask-Configurable Logic Blocks (Mask-CLBs) and mask-programmable interconnect resources (see Fig. 1). The logic of a Mask-CLB is the same as in the Virtex CLB. The Mask-CLB has also four slices. As the programmable resources of the Virtex CLB has been removed, the floorplan of the logic inside the Mask-CLB is different.

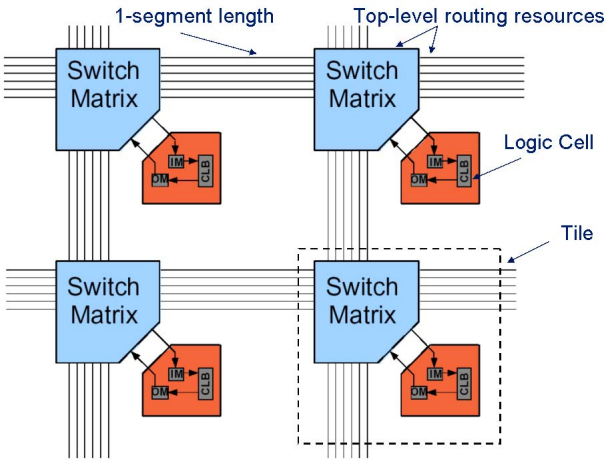

Figure 1. Zelix MPGA array

Current LUT-based MPGAs use a random interconnect structure (e.g. Altera HardCopy-II [3]). This is because final interconnect is customised by metal masks and therefore it is possible to route with a semi-custom tool. The

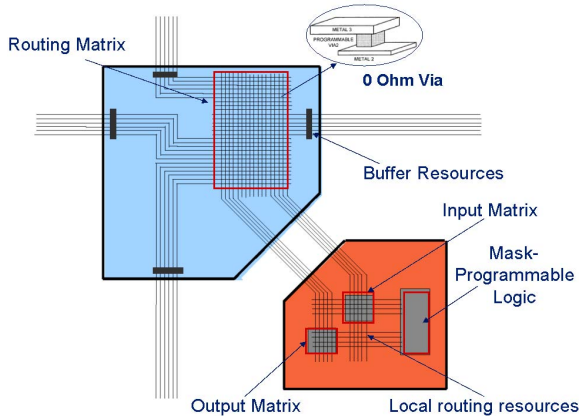

Figure 2. Zelix MPGA tile

Zelix mask-programmable interconnect is planned in the same fashion as the FPGA, i.e. we have a predefined regular interconnect structure. The benefits of this regular interconnect structure are the possibility to use an FPGA routing tool, on one hand, and easy-predictability of physical phenomena (e.g. crosstalk), on the other hand [8]. The Zelix MPGA has two hierarchical levels for the interconnect: top-level interconnect and local interconnect. The local interconnect is a mask-configurable input/output crossbar (Fig. 2). As the configuration is done with vias, we have a full-populated crossbar (in constrast to an FPGA which uses a sparse-crossbar).
The top-level interconnect is used for signal communication between CLBs. This communication is done through a switch matrix. The topology of the switch matrix is a full-populated crossbar. The Zelix MPGA has only one type of top-level wire: 1-segment line, i.e. one wire can only expand one Tile. This segmentation simplifies the routing tool and the MPGA device design. Although the segmentation is 1-segment line, there is not difference between a two 1-segment lines and a 2-segment line (with the same direction) in the final mask layout. A 1-segment line topology is also useful for buffer insertion planning. For instance, in a wire of length two, a 2-segment line has only two points for inserting a buffer (input, output), but two 1-segment lines have three points (input, mid point, output).
Although it is possible to use the top-level interconnect for routing the clock or the control signals in the Zelix MPGA a global clock and global control network are assumed.

3. Conversion Design Flow

The goal of the conversion flow is to have the same (or better) timing performance for an MPGA compared to an FPGA. The previous section has shown that the Zelix MPGA logic is the same as the Xilinx Virtex-II FPGA, so the logic delays are assumed equal. The delay difference is in the interconnect. To verify that we reach this goal, we use Static Timing Analysis (STA). In STA, the timing analysis is carried out in an input-independent manner and looks for the worst-case delay of the circuit over all possible input combinations [14]. Commonly STA is used as a part of the sign-off methodology in a digital silicon device. The conversion flow checks the delay of all nets with STA. The same check with circuit-level simulations is rather complex and time expensive. Therefore prohibitive.
The Xilinx Virtex-II Pro design enviroment is called ISE. After our designed circuit is placed and routed, the Xilinx ISE creates two files for STA. One is a verilog gate-level netlist (in terms of LUTs, AND-gates, ...) and the other is a Standard Delay Format (SDF) file. The SDF file has annotated the static delays of the cells and the interconnect. The STA analysis is done with Synopsys PrimeTime tool. This tool displays a report with the delays of all nets. To manage this amount of reported data a Perl-script has been created.

3.1. Zelix MPGA Front-end Design Flow

The design flow without buffer insertion of the MPGA architecture is shown in Fig. 3. The Zelix MPGA flow starts from the Xilinx STA gate-level netlist. This netlist models gates for configurable signal routing by means of buffers. The buffers are used to annotate programmable interconnect delays. A Perl-script removes all buffers (including the hardwired buffers). In a later step of the flow the necessary buffers for the interconnect are added.

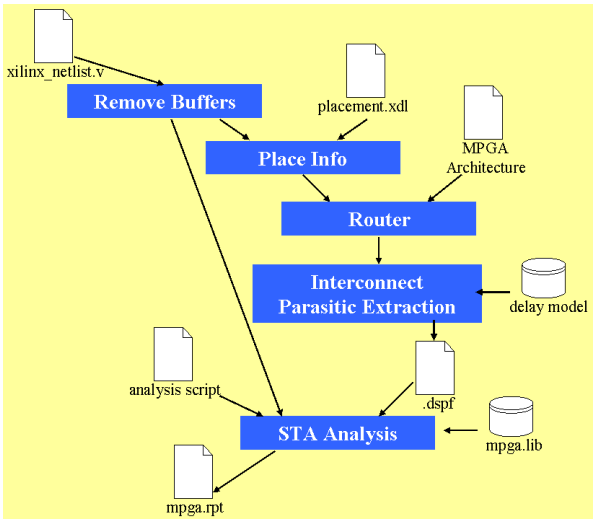

Figure 3. Zelix MPGA Design Flow

For our studies we import the same placement from the Xilinx ISE flow. It would be possible to do the MPGA placement from another placement tool as Cadence Encounter.

The reason for using the Xilinx placement is that we want to compare our MPGA routing with the Virtex-II Pro routing. Further studies can look for a better MPGA placement. The Xilinx placement information is extracted from the Xilinx Description Language (XDL) file. For matching the names of the Xilinx gate-level netlist and the XDL file a deep analysis of the Virtex-II Pro architecture has been done. The obtained placement information is Tile level and not Slice level. We use Tile level placement because in the MPGA we expect no major differences for STA within the placement inside a Tile.

The router tool has been programmed in C using software fragments from the VPR FPGA tool [5]. For routing the FPGA PathFinder algorithm has been programmed [6]. Our variant of the Pathinfinder maze router algorithm implements a routability-driven router instead of a timing-driven routing. In order to use a timing-driven routing a delay calculator is needed, as the buffer insertion is done at a later stage in our flow, a delay model for the interconnect is not accurate. Moreover, because the regular interconnect architecture has a penalty in terms of area, we want to optimise the number of wires over the critical path. Notice that the conversion flow must guarantee the same timing as in a FPGA, not a better performance. It is possible to set the number of tracks per channel in the MPGA router. The number of tracks per vertical or horizontal channels can be different (in the VPR FPGA tool vertical and horizontal channels are the same). This feature is important because the MPGA layout of a Tile usually is rectangular, so in one direction there are more tracks per channels than in the other. The target MPGA architecture is described in a Perl-based language. Then the parser and abstraction steps transform the architecture into an internal graph representation.

A Perl-script creates an interconnect RC tree with the routing information (a DSPF file). Fig. 4 shows an example of a net with our interconnect model. The interconnect parasitic model has five different elements: wire, input crossbar, output crossbar, cell input/output pins, input/output pads. All the cases are modelled as worst case scenario. The length of the wire is the distance from one Tile to the closest Tile. For the input crossbar the worst case is assumed when one wire comes to the switch matrix and goes to the input cell: the same parameters for the wire are assumed. Also we assume that the output crossbar has no effect on the STA, because it is modelled with the worst-case scenario of the input crossbar. The capacity effect of the cell input/output pins is depreciable in comparison with the wire or crossbar capacity. Values for the input/output pad capacity has been set. Infineon 130nm CMOS six-layer (copper) is used for setting the values of the interconnect elements. Infineon 130nm process technology is comparable with the IBM 130nm process technology used by Xilinx in the Virtex-II Pro.

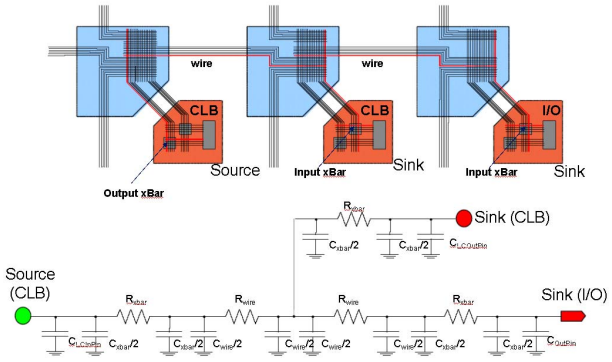

Figure 4. Example of the interconnect model

Once we have the interconnect information of the Zelix MPGA, we can proceed to do the STA analysis. The STA tool used is Synopsis PrimeTime. The flow guarantees that all the net delays are reported with a PrimeTime TCL-script. The STA analysis of the Xilinx Virtex-II Pro and the Zelix MPGA are compared. The delays in the Zelix MPGA are calculated from the output of one gate to another gate. In Virtex-II the same path as in the Zelix MPGA is used, i.e. if there are buffers in the path, we include them. When a MPGA-net has more delay than the corresponding FPGA-net, there are two possible solutions: first look if the mapped circuit still satisfy the specifications (slack, critical path, ...) and second we can try to reduce the delay. In our conversion flow we assume that we are in the second case (to be in a worst-case scenario). Due to a reduced area of the MPGA array we also expect in general shorter interconnect delays.

3.2. Buffer Insertion

The delay of a wire is proportional to its intrinsic capacitance and resistance. Moreover the wire delay depends quadratically on the wire length. If we reduce the capacitance or the resistance of a net, we reduce the delay. A big delay on a net can come because a wire is too long or because the net has a larger fanout. The solution for both cases is the same: buffer insertion. A buffer (also called repeater) breaks the net into two smallest nets, and because the length quadratically dependence, the delay is reduced. The buffer introduces a small delay to the total path, but isolates two nets. We apply the van Ginneken's algorithm [16] for buffer insertion. The van Ginneken's algorithm uses the Elmore model for delay analysis. The algorithm finds the buffers positions in a RC-tree such that the Elmore delay is minimal. The topology of the RC-tree is a Steiner tree. Our programmed algorithm is a general extension of the published van Ginneken's algorithm. For instance, in the van Ginneken's paper the nodes of the Steiner tree can have only up-to two leafs, in our case there are no leaf limitations.

Circuit	#gates	#gates without buffers	#slices	#nets	average fanout	maximum fanout
IFX_PP	764	275	65	381	1.92	29
IFX_ACE	1526	654	215	717	2.68	40
IFX_FIR	3057	1391	333	1401	2.36	38
IFX_IIR	4347	1995	506	2090	1.92	68

Table 1. Circuit characteristics after synthesis

Van Ginneken's algorithm assumes that a set of buffer insertion candidate locations are predetermined for the given topology. The later is a van Ginneken's algorithm quality problem for ASIC semi-custom methodologies, where the buffer positions are unknown a priori [4]. In a MPGA the buffer positions are predefined, but must be planned [17]. The Zelix MPGA architecture has buffers at the end point of the interconnect one-segment wire.
After the buffer insertion a verilog and parasitic file is created. After STA analysis, we look if the delays reaches the requested time imposed by the FPGA. If the delay is still larger in the MPGA than in the FPGA, the solution is careful analysis of the net and manual buffering and routing.

4. Experimental results

In this section we present the experimental results of four benchmarks circuits using the conversion flow described in Section 3. The benchmark circuits used in these experiments are four Infineon Technologies proprietary circuits: an 8-bit FIR Filter (IFX_FIR), a 16-bit Biquad IIR Filter (IFX_IIR), a state machine of a Protocol Processor (IFX_PP) and a matching unit of an Associative Search Processor (IFX_ACE). The first two benchmarks are data processing oriented and the other two benchmarks are control flow oriented. The purpose of these benchmarks is a proof-of-concept of the conversion flow. Although it is possible with our conversion flow, an architectural design exploration is out of the scope of this paper.
Each circuit has been synthesised with the commercial Synplicity tool: Synplify Pro. The Virtex-II Pro device used is a 2vpfg256-7. Table 1 shows the circuit characteristics after synthesis. The second column is the number of gates in the Virtex-II netlist. The third column shows the number of cells that we have after removing the buffers from the Xilinx netlist. The fanout report does not includes clock or FF control nets. The average fanout is around two. The biggest fanout (20-70) is concentrated in a small number of nets. In the conversion of the data to the MPGA router, a critical point is the Virtex-II placement information. The current Perl-script version cannot match (0.5% of total) all the names of the verilog netlist and the Xilinx XDL file. Therefore this information must be introduced by hand in the data structure for the MPGA router. We assume a dedicated clock network. Therefore, the net of the clock is not routed. Although it is planned to use a predefined network for the flip-flops control signals, in our experiments we treated the control signals as a normal signal. We observed no differences in the track per channel utilisation when the clock is routed with the other signals. This is because in the worst case the number of tracks per channel can be augmented by one (as we have one clock). The use of a dedicated balanced clock is because of the delay and the skew of the clock. We can see in Table 2 the number of tiles that we have after logic packing. In this experiments the placement is not stressed, so the logic-cells are sparsely placed in the 22 columns x 17 rows array. The benefit of a sparse placement is a relaxed routing, i.e. a low utilisation of tracks per channel. The number of tracks per channels is shown in Table 2. In the Zelix MPGA we have the different utilisation by channel (vertical or horizontal). The number of tracks is 4 to 7 times greater for the Virtex-II than for the Zelix MPGA. This is because the hierarchical interconnect segmentation (double lines, hex lines, ...) of the Virtex-II needs more routing resources compared to a one-wire model (our case). We have to say that a segmentation in a FPGA is necessary to be efficient in terms of area and delay [5]. Further studies with more benchmarks must be realised to see and understand the real difference between the routing efficiency in the MPGA vs FPGA.

Circuit	#tiles	Zelix MPGA		Virtex-II FPGA		averg. tracks ratio
		X	Y	X	Y	
IFX_PP	39	8	7	39	33	x4.8
IFX_ACE	82	6	9	69	36	x7
IFX_FIR	98	15	15	78	69	x4.9
IFX_IIR	169	14	14	93	50	x5.1

Table 2. Tile packing and routing result (tracks per channel). X: horizontal, Y: vertical

The STA analysis has been realised after the routing. The second column of the Table 3 shows the number of MPGA nets with more delay than the correspondent Virtex-II net. These 'larger-delay' nets are about 1%-2% of the total nets. The third columns shows the number of buffers inserted using the van Ginneken's algorithm. The average number of

buffer per Tile is around one. The circuits in [17] have an average number between 2 - 14 buffer per Tile. We note that the number of buffer insertion is better in our case because the technology used in their experiments is 90nm (ours is 130nm) and the experiments are with a Tile with 8 LUTs (ours is 16 LUTs). More LUTs in a Tile means more local interconnect and less global routing. Although an automatic

Circuit	nets with larger-delay	#inserted buffer	avrg.buffer per tile
IFX_PP	6	32	0.8
IFX_ACE	19	112	1.4
IFX_FIR	20	93	0.9
IFX_IIR	22	126	0.7

Table 3. Results after STA

Perl-script for generating a DSPF file is under development, some nets have been mapped by hand to show the delays after buffer insertion. Table 4 shows the delays of two nets with the three cases: Virtex-II, Zelix MPGA without buffer insertion and Zelix MPGA with buffer insertion. The first net has a fanout of 21 and 21 buffers were inserted. The second net has a fanout 23 and 20 buffers were inserted. The results shows the benefits in the delay reduction with buffer insertion. The four designs have been mapped into

net	Virtex-II (ns)	Zelix w/o buff. (ns)	Zelix w/ buff. (ns)
1	0.389	0.419	0.088
2	0.740	0.827	0.089

Table 4. Delay in two nets of the IFX_FIR circuit

the conversion flow within a week. It is possible to follow all the steps of the conversion flow in about two hours for a 2000-gates design. The same design using a Cell-based MPGA requires 1-2 weeks (because of the timing integrity problems caused by re-synthesis).

5. Conclusions

A conversion flow from a LUT-based FPGA to a LUT-based MPGA has been presented which preserves the gate-level netlist and the placement of the FPGA. The predefined routing resources of the MPGA apply a uniform segment length of one and 100% usability of the switch matrix. The flow includes a routing step and buffer insertion. Preliminary experimental investigations indicate that the new routing resources require substantially less routing tracks than in the FPGA. Timing integrity seems to be obtainable with high security.

For production the metallisation stack of a ASIC technology is sufficient. For customisation only few mask steps are required. So we conclude, that although we sacrificed area with the target architecture the benefits of design security, total conversion cost and reachable yield optimisation justifies the solution presented.

References

[1] *Virtex-II Pro^{TM} Platform FPGA Handbook*. Xilinx Inc., San Jose, CA, 2002.

[2] Rapidchip technology fast custom through platform-based design. *White Paper*, 2004.

[3] *HardCopy Series Handbook*. Altera Inc., San Jose, CA, 2005.

[4] C. J. Alpert and A. Devgan. Wire segmenting for improved buffer insertion. *Proceedings of the IEEE/ACM Design Automation Conference*, pages 649–654, June 1997.

[5] V. Betz, J. Rose, and A. Marquardt. *Architecture and CAD for Deep-Submicron FPGAs*. Kluwer Academic Publishers, Norwell, MA, 1999.

[6] C. Ebeling, L. McMurchie, S. A. Hauck, and S. Burns. Placement and routing tools for the Triptych FPGA. *IEEE Tansactions on VLSI*, pages 473–482, December 1995.

[7] J. M. Green and H. Klar. CMOS gate array architecture for digital signal processing applications. *IEEE Journal of Solid State Circuits*, 31(3):410–418, March 1996.

[8] S. P. Khatri, A. Mehrotra, R. K. Bryton, R. H. J. M. Otten, and A. Sangiovanni-Vincentelli. A novel VLSI layout fabric for deep sub-micron application. *Proceedings of the 36th ACM/IEEE Design Automation Conference*, pages 491–496, June 1999.

[9] G. Lemieux and D. Lewis. *Design of Interconnection Networks for Programmable Logic*. Kluwer Academic Publishers, Norwell, MA, 2004.

[10] A. Levinthal and R. Herveille. FlexASIC structured array: A solution to the DSM challenge. *DesignCon 2005*, February 2005.

[11] M. Okabe and et al. A 400k-transistor cmos sea-of-gate array with continuos track allocation. *IEEE Journal of Solid State Circuits*, pages 1280–1286, October 1989.

[12] T. Okamoto, T. Kimoto, and N. Maeda. Design methodology tools for NEC elsectronics' structured ASIC ISSP. *Proceedings of the ISPD'04*, pages 90–96, April 2004.

[13] M. Santarini. Structured ASIC deserve serius attention at 90nm. *EDN Magazine*, pages 69–74, July 2005.

[14] S. Sapatnekar. *Timing*. Kluwer Academic Publishers, Norwell, MA, 2004.

[15] D. D. Sherlekar, O. Siguenza, and H. Yang. Maximize design flexibility with fast turnaround time while minimizing design cost with metal programmable libraries. *IP Based Design 2003 Conference*, November 2003.

[16] L. P. P. P. van Ginneken. Buffer placement in distributed RC-tree networks for minimal Elmore delay. *International Symposium on Circuits and Systems*, pages 865–868, May 1990.

[17] T. Zhang and S. Sapatnekar. Buffering global interconnects in structured ASIC design. *Proceedings of the Asia/South Pacific Design Automation Conference*, January 2005.