Design Automation for Mask Programmable Fabrics

Narendra V. Shenoy
nshenoy@synopsys.com

Jamil Kawa
jamil@synopsys.com

Raul Camposano
raul@synopsys.com

Advanced Technology Group, Synopsys Inc.,
700, East Middlefield Road, Mountain View, CA 94043

ABSTRACT

Programmable circuit design has played an important role in improving design productivity over the last few decades. By imposing structure on the design, efficient automation of synthesis, placement and routing is possible. We focus on a class of programmable circuits known as *mask programmable* circuits. In this paper, we describe key issues in design and tool methodology that need to be addressed in creating a programmable fabric. We construct an efficient design flow that can explore different logic and routing architectures. The main advantage of our work is that we tailor tools designed for standard cell design, that are readily available in the market, to work on a programmable fabric. Our flow requires some additional software capability. A special router that understands programmable routing constructs to complete connections is described. In addition, a tool that packs logic efficiently after synthesis is also presented.

Categories and Subject Descriptors: B.7.1 [Hardware]: INTEGRATED CIRCUITS - Types and Design Styles. B.7.2 [Hardware] INTEGRATED CIRCUITS - Design Aids.

General Terms: Design, Performance

Keywords: Integrated circuits, Mask programmable fabrics

1. INTRODUCTION

A common strategy to improve circuit design productivity has been to impose structure on the design methodology. This enables automation of tasks and scaling of problem size that would be impossible to achieve in a manual environment. Design styles, such as standard cell, gate array, field programmable gate array(FPGA), offer varying trade-offs in productivity for performance (area, delay, power of the circuit), predictability and flexibility.

We focus on a class of structured design that limits the number of metal and via masks that need to be configured. This can be considered as a restricted form of standard cells. The design is implemented as an array of identical cells. A cell has mask programmable logic and mask programmable interconnect. An array of such cells is termed as a "Mask Programmable Fabric" (MPF). Our work is motivated by the following observations:

- Mask costs have dramatically increased with shrinking feature size. Re-spins are often required due to specification changes, logic bugs after first silicon, *etc.* and add significantly to the design expense. Implementing circuitry that will evolve over the lifetime of a product as a mask programmable fabric will amortize the mask costs effectively.
- Each design has its own peculiar characteristics that drive the decision of what constitutes the best programmable fabric for its implementation. Hence any solution must encourage efficient experimentation to choose the best programmable fabric. This poses a restriction that the implementation flow be automated to the maximum.
- Variability during manufacturing is an increasing factor with decreasing feature size. Since the structure is repetitive, we can characterize properties such as capacitance, resistance of the devices and interconnect easily. This eases analysis during the implementation flow. A regular structure helps reduce problems due to manufacturing defects. We can focus yield optimization techniques on the single programmable fabric cell.

The case for MPFs has been eloquently made by Pileggi *et al.* [20].

1.1 Problem Statement and Contributions

Our approach is to create a design flow that relies on constructing a programmable fabric from repeated instantiations of a single programmable cell. A key distinction of our approach is to let the user define the architecture of a programmable cell. Our main contributions in this work are:

- An automated flow to evaluate different programmable cells. It is undesirable to construct a set of new tools to deal with MPFs due to the costs and risks involved. We construct a design flow that is minimally perturbed from the traditional standard cell flow.
- Design of a generic programmable interconnect architecture that considers accessibility to pins and routing capacity.
- Software support for the design flow. We describe two specific tools, namely a packing optimizer and a router.
- A detailed analysis of various trade-offs of area, delay, via-density and power compared to the standard cell approach. We believe this is the first approach to provide results for MPFs.

1.2 Previous Work

A lot of the key ideas in mask programmable structures

DAC'04, June 7–11, 2004, San Diego, California, USA.

have their roots in the work done in the context of FPGAs. Since this is a rich field, we refer to the work by Brown *et al.* [4] as a general reference. Rose *et al.* [22] present a discussion on architectural trade-offs of FPGAs. An optimum lookup-table size in an FPGA is investigated by Rose *et al.* in [23]. Unlike FPGAs, which seek an architecture that fits all possible designs, we are only constrained to find a good architecture for a given design.

Determining the nature of the combinational logic for a programmable cell is a well researched area. Xilinx Inc. and Altera Corp. rely on look-up tables, while Actel Corp. relies on multiplexor based architectures. Works by Lin *et al.* [12], Thakur and Wong [25], and Zilic and Vranesic [26] focus on the design of universal logic modules. The need for embedding a flip-flop in the programmable cell is pointed out in [23]. The interconnect architecture in FPGAs depends on the underlying technology used to achieve programmability. The concept of a switch box and its flexibility is studied by many researchers, *e.g.* Rose and Brown [21], Chang *et al.* [6], Schmit and Chandra [24] to name a few. Zuchowski *et al.* [27] and Lien *et al.* [11] explore embedding FPGAs in a larger system in silicon. We believe embedded MPFs offer better trade-offs in performance and cost compared to embedded FPGAs. They also do not suffer from the need to down load a bit-stream to initialize the programmable component.

Work in mask programmable structures can trace its origin to the gate array based design flow. Recent work by Patel *et al.* [19] highlights the emergence of via programmable gate arrays. The routing model they use is a crude approximation to a true mask constrained router. *Our work seeks to correct this by providing results with a true mask programmable router.* In a recent effort [20], they make an eloquent case for via programmable gate arrays. Once again, they can only partially assess the trade-offs by using a standard cell router. Regular structures for interconnect are proposed by Khatri *et al.* [9] and by Mo and Brayton [15] as a means of achieving timing convergence. Mask programmability is exploited by eAsic Corp [18] for interconnect realization, while the logic is configured by using field programmability.

The literature is rich in efforts related to special tools for FPGAs. The work by Murgai *et al.* [16] and Francis *et al.* [7] describes specialized synthesis techniques for FPGAs. The first work also uses integer programming to pack gates into a logic cell. We shy away from specialized synthesis techniques as we wish to re-use the standard cell flow as much as possible. Specialized placement and routing algorithms have also been designed for FPGAs. Work by Brown *et al.* [5], McMurchie and Ebeling [13] provide some of the early insights on routing for FPGAs. Lemieux *et al.* [10] show that two step routing (global routing and detailed routing) can be competitive to a combined routing process. We design a three-phase detail router that includes an intermediate step of track assignment (motived by the work of Batterywala *et al.* [1]). Betz and Rose describe a packing, placement and routing system for FPGAs in [2]. The book by Betz *et al.* [3] describes interactions between architecture, placement and routing and is an excellent reference on this subject.

1.3 Paper Organization and Definitions

The rest of the paper is organized as follows. In Section 2 we investigate the construction of a flow for programmable fabrics. Concepts in design of a programmable cell are described in Section 3. New tools needed to support the flow form the content of Section 4. Section 5 presents the experimental results on our cells, flow and tools and comparisons with a standard cell based implementation. We present some conclusions from this work in Section 6.

In the rest of the paper, we use the term programmable to imply mask programmability. We number the metal layers in increasing integers; metal 1 is closest to the substrate. We refer to a via layer by using "via" followed by the lower metal layer, *e.g.* via1 connects metal 1 and metal 2. We assume that metal 1 has a horizontal preferred direction. Successive metal layers have a preferred direction alternating between horizontal and vertical directions. We define an active pin in an MPF cell as a pin that is not tied to power or ground.

2. MPF DESIGN FLOW

A typical design flow is presented in Figure 1. The standard cell design flow (in a very simplistic view) follows the bold arrows. It starts with creating a library of standard cells and defining their power, delay, physical and electrical characteristics. Given a design, the flow consists of synthesizing the design to a net-list of gates consisting of elements from the library. The physical design step consist of floor-planning, physical synthesis and routing. We finally do an extraction for interconnect networks and back-annotate them in a static timing analyzer for analysis. We use industry standard tools for the standard cell design flow.

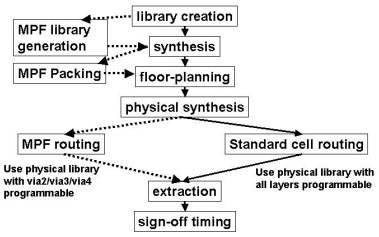

Figure 1: Design flow for standard cell and programmable fabrics

In Figure 1, the MPF flow consists of following the dashed arrow where possible. Once again we use industry standard tools for the design flow, except for the MPF portions (where we use an in-house tool). We use a custom language to describe a MPF cell called the MPF language (see Section 4.1). It contains information on the nature of the logic and routing programmability. Our flow with mask programmable fabrics mimics the standard cell flow with the following exceptions:

- The MPF library consists of a single cell. Using the MPF library and a description of the nature of the MPF cell, we generate a derived library for synthesis. The cells in the derived library are generated by enumerating logic functions obtained by tying a sub-set of inputs to power or ground.
- After synthesis, the net-list consists of instances from the derived library. We go through a reverse transformation of converting each cell in the derived library to an instance of the MPF cell with its inputs appropriately tied. We also pack the logic to fully utilize unused resources in a cell. The

need for this will be clear in Section 3 and the approach will be outlined in Section 4.

• Instead of a standard cell router, which assumes that all metal and via layers are available for realizing physical connections, we propose a special router. The MPF router relies on symbolic constraints called connections (between a pair of pins, a pin and metal segment or between a pair of segments) to realize the physical connectivity. This information is provided in the MPF language.

We create a baseline implementation flow for a set of designs using standard cells. This defines the performance metrics (area, delay and power) for the standard cell implementation. Each instance of an MPF cell will result in performance that is inferior to standard cell due to the constraints of mask programmability. The ratio of performance metric in an MPF library to a standard cell library will be invariably greater than one. The rest of the paper focuses on methods to reduce this multiple.

3. MPF CELL DESIGN

We consider several aspects in the design of a fabric cell in Sections 3.1, 3.2, 3.3, 3.4, and 3.5. In Section 3.6, we outline a generic scheme that enables easy routing of the interconnect fabric. The reader will find a tight link between the choices made during cell design and the impact on tools.

3.1 Programmable Layers

Deciding on layers that provide programmability is key. We leave the mask layers in front end of line set untouched. From the back end of line set, if we pick layers close to the top, the number of foundry steps that remain to be completed upon re-programming is small. However from a routing perspective, all connections to pins will need to brought to the higher programmable layers. This eats scarce vertical routing resources. Moreover, it renders layers above the MPF implementation unavailable for the rest of the chip. Hence we choose to restrict programmability to the lower layers of metal and vias (metal 2 to metal 5) [1]. We pick via2, via3 and via4 to be the programmable layers for our cell.

3.2 Combinational Logic

The choice of programmable combinational logic is important to a specific design. If the design is "rich" in arithmetic operations, it is important to have gates such as 3-input XORs. In such a case, a look-up table serves well. On the other hand if the design is dominated by control logic, then a multiplexer tree structure that enables complex and-or-invert gates fares better. We use a multiplexer tree for our cell.

3.3 Local Inversions

In a typical design, the ratio of inverters to the total number of gates is at least 10%. Implementing an inverter using a large programmable gate is an extremely inefficient use of resource. In a look-up table architecture, a majority of the inversions can be absorbed into the table functions. It is not possible to do so for other architectures. A solution is to create small inverters in the programmable cell that can only drive input pins within the cell. Thus inversion of a signal is generated locally where it is needed. In this case, we also need to determine the number of such inverters in a cell. From a tools perspective, we need to convert individual inverters (after synthesis) into local inverters as much as possible (thereby reducing programmable cell count). This is addressed in the packing step (Section 4.3). For our cell, we assign two local inverters for each combinational logic function in a cell.

[1]Metal 1 is one of the hardest layers to generate in terms of mask complexity due to closely packed features. Keeping it fixed is cost advantageous

3.4 Sequential Logic

The nature of the sequential element is to a large extent dictated by the choice of sequential behavior required on a per design basis. The need for set/reset/multiplexer enable lines and support for scan test are to be considered. We provide only the non-inverted next-state output Q (*i.e.* we do not provide $\overline{Q}$). Each output in a cell has to be designed with sufficient strength (large transitor size) to drive the programmable interconnect fabric. By not providing $\overline{Q}$, we avoid having a large output stage in the cell and thus save area. We can generate $\overline{Q}$ using local inversion.

3.5 Ratio of Combinational to Sequential Logic

Sequential content in a design can vary significantly. If the number of sequential elements in a design (compared to the number of combinational gates) is small, then having a combinational to sequential ratio of 1:1 in the MPF cell is unfavorable. Depending on the design, the user can decide on an MPF cell in which the ratio of programmable combinational logic to programmable sequential logic varies from one to two. Software support to take advantage of this is provided in the packing step (Section 4.3).

3.6 Programmable Interconnect

It is important to design a generic interconnect scheme that enables the router to operate efficiently independent of the decisions made earlier in this section (Section 3). We provide a set of guidelines that enable such a design.

All pins are designed to be available on metal 2 as vertical segments that span a few metal 3 tracks on the north and south edges of the cell. This enables access from pins to the interconnect fabric by simply using a via. Logic programmability requires tying pins to power and ground. Unused pins are tied to ground. We solve this by dedicating horizontal metal 3 tracks that span all input pins to power and ground. Thus tying a pin to power or ground is achieved by a via2 connection. Figure 2 shows six pins. Each pin spans six

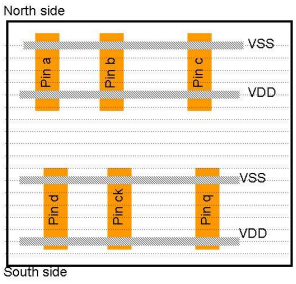

Figure 2: Pin Access

metal 3 tracks (in dashed lines). Two of these are used to

enable tying pins to power and ground (VDD and VSS). There must be as many free tracks as active pins on each side. Failure to do so can leave pins orphaned during routing, *i.e.* there is no way to connect to them. During routing, we use an analysis step called pin reservation (Section 4.4) to detect this.

The MPF cell uses a cross-bar architecture on metal 3, metal 4 and metal 5. Each metal segment spans the cell (almost) in the preferred direction. A segment can connect to metal segments on neighboring layers by means of an appropriate via. An MPF cell can have up to four MPF cells as its neighbors. Since an interconnect fabric is defined within a cell, connectivity with the interconnect in the neighboring cell is achieved by means of vias. Figure 3 shows an example

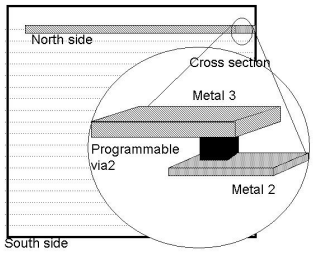

Figure 3: Programmable Connection to Neighboring Cell

of how a metal 3 segment can be extended to the cell on the right. We provide metal 2 segments that connect to neighbors by abutment. Hence a signal that needs to propagate to the right requires a programmable via2 in the current cell and a similar via in the neighbor cell. Within a cell, we reserve a fixed number of metal 3 and metal 4 segments for within-the-cell routing. These are utilized to connect signals reaching the cell to the appropriate pin. For a cross-bar architecture, a safe number to use is the number of active pins. The remaining number of segments are available for routing signals through a cell. Thus managing the number of active pins in a cell is an effective means of controlling local routability. We use this as a constraint during the packing step (Section 4.3).

4. MPF DESIGN TOOLS

As can be seen from Figure 1, we have restricted the extra tool support required to enable an MPF flow to be very minimal.

4.1 Language for Programmable Fabric Description

We have designed a textual format to describe an MPF cell. During library generation the logic function information in the description is used to create a set of functions for use in the derived library for synthesis. The language also contains constraints on how multiple instances of elements from the derived library can be packed onto a single MPF cell. This is used in the packing step. Finally it contains constructs to describe MPF routing constraints. The MPF router understands only three objects: pins and segments and connections. The interconnect fabric defines the segments. A connection is a mechanism to realize an electrical equivalence between either a pair of pins, a pair of segments or a pin and a segment.

4.2 Library Generation

Library generation consists of enumerating all distinct (equivalent up to permutation of inputs) functions of the programmable logic function. The enumerated functions are added as logic gates to the derived library. In addition, a sequential cell that models the sequential circuit in the MPF cell is also added.

4.3 Packing

We first map the logic gates in the derived library to appropriately programmed instances of the MPF cell. We then define a set of optimization steps intended to reduce the number of MPF cell instances in the design. Each step during packing ensures that the active pin count in an MPF cell does not exceed an user specified thresh-hold. This is implicit in the graph constructions described below.

We first merge each sequential cell with either a cell driving it or driven by it. We formulate this as a graph matching problem on a graph derived from the connectivity of the net-list. Each sequential cell and its immediate combinational fanin and combinational fanout cells (in the net-list) are nodes in the graph. There is an edge between a sequential cell and each of its combinational fanin and fanout cells. Packing a sequential and a combinational cell can be realized by solving a maximum matching problem on the graph. Since this graph is often sparse and not connected, an optimum graph matching algorithm [8, 14] can be used efficiently.

The second step consists of packing multiple combinational cells if we have a combinational to sequential ratio exceeding one. The graph is constructed in similar manner to the previous step, except that we consider only combinational cells with a fanin-fanout relationship. Unfortunately, this results in large connected graphs. Although the optimum matcher finds excellent solutions, it requires long computation times. Instead we resort to a heuristic which yields an approximate solution (with an observed quality nearly 10% away from the optimum) in a fraction of the runtime.

Finally, we seek to push inversions to be local to a cell. Each independent inverter competes for an unused local inverter in its fanout cell with other inverters in the fanins of the fanout cell. We create a graph with a node for each inverter in the design (assuming all fanouts of the inverter can accommodate it as a local inverter). An edge exists between two inverters if they have a common immediate fanout cell or if one drives the other. We now solve an instance of a Maximum Independent Set on this graph. The decision problem to check if there is a Maximum Independent Set of size k in a graph is NP-Complete. However, it is trivial to devise a heuristic to compute the Maximal Independent Set using node degrees.

4.4 Routing

From the placement information, we build a routing graph for the whole design. The nodes in the graph represent pins in the MPF cells or segments in the interconnect fabric. For each top level port in the design, we pick a set of "closest" metal segments in the interconnect fabric. The router will connect to one of these segments when it attempts to complete the routing of the net associated with the port.

The connections between a pair of segments and a segment and a pin are available from the MPF description. The connections between neighboring cells are inferred from the placement and the MPF description. Each connection forms an edge in the routing graph between the pair of objects it connects.

The routing problem is broken into multiple steps to manage the complexity much like in standard cells. In the first step we reserve metal segments that can connect to an active pin in the design. This step, called pin reservation, merely seeks to detect any pins that are orphaned due to a poor cell design. Also the mechanism of reserving a segment for a pin prevents nets other than the one connected to the pin from using the segment. This ensures that the pin is not orphaned due to poor detailed routing (due to its sequential nature in processing nets). Global routing is carried out in a similar manner to standard cells. We use each MPF cell as the granularity of coarseness during global routing. Thus the detail router is restricted to only routing within an MPF cell. During global routing we discount the horizontal and vertical capacities of each MPF cell by the maximum number of active pins. At the end of global routing, we check if an embedding of the global routing into the fabric is possible without any capacity violations. If we find a violation, the solution is to augment the interconnect fabric in the MPF cell by one more routing resource and redo the steps. Once we succeed at this step, the interconnect architecture and the routing algorithm will ensure a solution exists after detail routing. It is easy to see in a cross-bar routing architecture, if the global routing can be successfully embedded and each MPF cell has at least as many horizontal and vertical tracks free as the number of active pins, the detail router cannot fail. We embed the horizontal and vertical segments that span more than one MPF cell using an approach based on track assignment. Finally detailed routing is carried out the design, focusing on one MPF cell at a time.

5. EXPERIMENTAL RESULTS

We use an eight layer 0.13μ process technology for all our experiments. We first gather a set of circuits ([17]) that are good candidates for small cores and a customer testcase (ckt1). Note that for the MPF flow, we use only five layers of metal. We map all designs to a standard cell library and complete the standard cell flow in Figure 1. The designs are placed and routed with near 90% utilization. The data is presented in Table 1. We provide the number of cells (column 2), area in μ^2 (column 3), via count (column 4), worst path delay amongst different clocks in a design (column 5) and power in μW (column 6). Note that "fpu" is a pipelined floating point unit that needs to be retimed. Since we have not included retiming in the synthesis step, a large delay is seen for a path in both the standard cell and MPF implementations.

5.1 Programmable Fabric C1S1

For the first fabric, we use a combinational function based on the ACT2 architecture from Actel[2]. We add a D flip-flop with set/reset circuitry and only a Q output. We provide two inverters in each cell for local inversion; two was experimentally observed to be sufficient. We call this cell C1S1 as it has one combinational and one sequential element. Figure 4 shows the C1S1 cell with only the metal 1 and metal 2 layers (the interconnect is omitted). The area is 132 μ^2. The interconnect fabric consists of 21 horizontal segments on metal 3, 28 vertical segments on metal 4 and 27 horizontal segments on metal 5. The performance multiples for

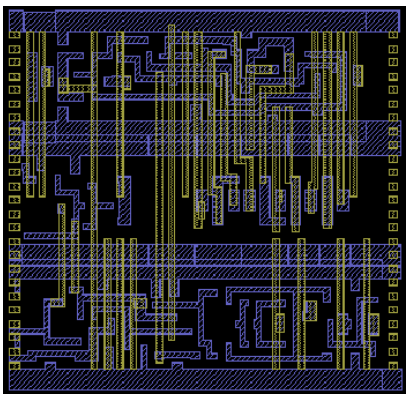

Figure 4: C1S1 cell layout

this circuit are provided in Table 1 (columns 7-10). We note that the area ratio[3] in MPF to standard cell lies in 4.2-10.2. The packing step reduces the cell count by typically 20%-30% from the post-synthesis net-list. The utilization of the MPF placement and routing is managed at approximately 90%. In column 8, we see that the via count increases significantly. However the via density (via count divided by area) stays close to or less than the via density for standard cells. This leads us to believe that the yield (if it is via-critical) will be similar to a standard cell design at the area multiple of the MPF implementation [4]. 20-30% of the vias are used for logic programming (*i.e.* programming the inputs of the MPF cell). The rest of the vias are used for interconnect. Column 9 provides the worst delay multiple (amongst the different clocks for each design) for C1S1 and is in the range of 2.3 to 7.2. The power multiple presented in column 10 is between 1.3 and 2.3.

5.2 Programmable Fabric C2S1

The second fabric we design is called C2S1. It uses the same combinational logic as C1S1. It has two identical combinational functions and four local inverters. The area of the cell is 180 μ^2. The interconnect fabric consists of 21 horizontal segments on metal 3, 38 vertical segments on metal 4 and 27 horizontal segments on metal 5. The data for this circuit is provided in Table 1 (columns 11-14). Note that the area multiple for C2S1 is in the 3.5 to 7.8 range. The via counts are less by about 10% compared to C1S1. In the case of C2S1, 20-33% of the vias are used for logic programming. The worst delay multiple compared to standard cell is in the 2.4 to 3.9 range. The power multiple lies between 1.3 to 2.3.

[2] $Z = \overline{(A_1A_2)(S_1 + S_2)D_3 + (A_1A_2)}(S_1 + S_2)D_2 + (A_1A_2)\overline{(S_1 + S_2)}D_1 + \overline{(A_1A_2)}\ \overline{(S_1 + S_2)}D_0$.

[3] Testcase fpu is particularly bad as it has less than 10% of the cells as sequential elements (in the standard cell implementation).

[4] Note however that the critical areas for the standard cell and the MPF implementation can be very different. Under such a model, yield could be an issue. However, of late critical area has less influence on yield compared to printability and via related issues

Table 1: Experimental results

Design	Standard cell (actual)					C1S1 (multiple)				C2S1 (multiple)			
	#cells	Area	#Via	Delay	Power	Area	Via	Delay	Power	Area	Via	Delay	Power
risc16f84	1390	32918	16.2k	1.83	0.02	4.15	3.20	2.97	1.44	3.46	2.92	3.24	1.45
cordic	2513	67557	27.8k	3.69	0.07	6.54	5.15	2.36	1.32	5.13	4.48	2.40	1.30
ckt1	4711	87568	55.7k	2.76	0.06	5.05	3.15	7.19	1.73	4.33	2.94	3.89	1.72
usb	7193	132641	88.5k	1.36	0.11	5.33	3.71	6.36	1.65	4.35	3.58	3.54	1.70
fpu	7486	125145	81.3k	129.30	0.12	10.22	6.51	3.67	2.30	7.77	5.95	3.71	2.30
des	57440	1182525	560.7k	6.14	0.59	4.53	3.70	1.80	1.97	3.76	3.43	1.77	1.94

6. CONCLUSIONS

We have presented a discussion on the key issues for a MPF cell design. We have discussed the minimal set of changes in standard cell tools to enable a design flow for a generic MPF cell. We described a packing optimization step and a true mask programmable router. Specialized tools for specific mask programmable cells may improve upon the results we present. From the work by Zuchowski *et al.* [27] we know that embedding FPGAs leads to an area ratio of 50, a delay ratio of 10 and power ratio of 100 compared to a standard cell implementation. MPFs on the other hand are much effective with an area ratio between 3-7, a delay ratio between 2-6.4 and a power ratio between 1.3-2.3. We advocate using the MPF approach to design portions of circuitry that are subject to change in a system on a chip context. Such portions will be typically less that 1% of the total silicon area; thus an area multiple of even 5 is affordable for reduced costs and accelerated time to market.

7. ACKNOWLEDGMENTS

We acknowledge the help provided by A. Ghosh, M. A. Ali, E. C. Channakeshav, T. J. Sun, S. Tantry, in putting together the design flow.

8. REFERENCES

[1] S. Batterywala, W. Nicholls, N. Shenoy, and H. Zhou. Track Assignment: A Desirable Intermediate Step Between Global Routing and Detailed Routing. In *Proceedings of the International Conference on Computer-Aided Design*, 2002.

[2] V. Betz and J. Rose. VPR: A New Packing, Placement and Routing Tool for FPGA Research. In *International Workshop on Field Programmable Logic and Applications*, pages 213–222, 1997.

[3] V. Betz, J. Rose, and A. Marquardt. *Architecture and CAD for Deep Sub-Micron FPGAs*. Kluwer Academic Publishers, 1999.

[4] S. Brown, R. Francis, J. Rose, and Z. Vranesic. *Field Programmable Gate Arrays*. Kulwer Academic Publishers, 1992.

[5] S. Brown, J. Rose, and Z. G. Vranesic. A Detailed Router for Field-Programmable Gate Arrays. In *IEEE Transactions on Computer-Aided Design*, pages 620–628. IEEE, May 1992.

[6] Y.-W. Chang, D. F. Wong, and C. K. Wong. Universal Switch Modules for FPGA Design. In *ACM Trans. on Design Automation of Electronic Systems*, pages 80–101, 1996.

[7] R. J. Francis, J. Rose, and K. Chung. Chortle: A Technology Mapping Program for Lookup Table-Based Field Programmable Gate Arrays. In *Proceedings of the Design Automation Conference*, pages 613–619, 1990.

[8] H. N. Gabow. An Efficient Implementation of Edmonds' Algorithm for Maximum Matching on Graphs. *Journal of the ACM*, 23:221–234, 1976.

[9] S. Khatri, A. Mehrotra, R. Brayton, A. Sangiovanni-Vincentelli, and R. Otten. A Novel VLSI Layout Fabric For Deep Sub-Micron Applications. In *Proceedings of the Design Automation Conference*, pages 491–496. IEEE/ACM, 1999.

[10] G. G. F. Lemieux, S. D. Brown, and D. Vranesic. On Two-Step Routing for FPGAs. In *Proceedings of the International Symposium on Physical Design*, pages 60–66, 1997.

[11] F. Lien, J. Feng, E. Huang, C. Sun, T. Liu, N. Liao, and D. Hightower. A Hardware/Software Solution for Embeddable FPGA. In *Proceedings of the Custom Integrated Circuits Conference*, pages 5.3.1–5.3.4, 2001.

[12] C.-C. Lin, M. Marek-Sadowska, and D. Gatlin. On Designing Universal Logic Blocks and Their Application to FPGA Design. In *IEEE Transactions on Computer-Aided Design*, pages 519–527. IEEE, May 1997.

[13] L. McMurchie and C. Ebeling. PathFinder: A Negotiation-Based Performance-Driven Router for FPGAs. In *Proceedings of ACM/SIGDA Int. Symp. on Field Programmable Gate Arrays*, pages 111–117, 1995.

[14] K. Mehlhorn and G. Schafer. Implementation of O (nm log n) Weighted Matchings in General Graphs. The Power of Data Structures. In *Algorithm Engineering*, pages 23–38, 2000.

[15] F. Mo and R. K. Brayton. Fishbone: A Block-Level Placement and Routing Scheme. In *Proceedings of the International Symposium on Physical Design*, pages 204–209, 2003.

[16] R. Murgai, N. Nishizaki, N. Shenoy, R. K. Brayton, and A. Sangiovanni-Vincentelli. Logic Synthesis for Programmable Gate Arrays. In *Proceedings of the Design Automation Conference*, pages 620–625, 1990.

[17] Opencores. *IP Cores Repsoitory*. World Wide Web, http://www.opencore.com.

[18] Z. Or-Bach, Z. Wurman, R. Zeman, and L. Cooke. Customizable and programmable cell array. *U.S. Patent 6,331,790*, Issued 18 Dec. 2001.

[19] C. Patel, A. Cozzi, H. Schmit, and L. Pileggi. An Architectural Exploration of Via Patterned Gate Arrays. In *Proceedings of the International Symposium on Physical Design*, pages 184–189, 2003.

[20] L. Pileggi, H. Schmit, A. J. Strojwas, V. Kheterpal, A. Koorapaty, C. Patel, V. Rovner, and K. Y. Tong. Exploring Regular Fabrics to Optimize the Performance-Cost Trade-Off. In *Proceedings of the Design Automation Conference*, pages 782–787. IEEE/ACM, 2003.

[21] J. Rose and S. Brown. The Effect of Switch Box Flexibility on Routability of Field Programmable Gate Arrays. In *Proceedings of the Custom Integrated Circuits Conference*, pages 27.5.1–27.5.4, 1990.

[22] J. Rose, A. El Gamal, and A. Sangiovanni-Vincentelli. Architecture of Field Programmable Gate Arrays. In *Proceedings of the IEEE*, pages 1013–1029, 1993.

[23] J. Rose, R. Francis, D. Lewis, and P. Chow. Architecture of Field Programmable Gate Arrays: The Effect of Logic Block Functionality on Area Efficiency. In *IEEE Journal of Solid-State Circuits*, pages 1217–1225, 1990.

[24] H. Schmit and V. Chandra. FPGA Switch Block Layout and Evaluation. In *Proceedings of IEEE/ACM Int. Symp. on Field Programmable Gate Arrays*, pages 11–18, 2002.

[25] S. Thakur and D. F. Wong. On Designing ULM-based FPGA Logic Modules. In *Proceedings of ACM/SIGDA Int. Symp. on Field Programmable Gate Arrays*, pages 3–9, 1995.

[26] Z. Zilic and Z. G. Vranesic. Using BDDs to Design ULMs for FPGAs. In *Proceedings of ACM/SIGDA Int. Symp. on Field Programmable Gate Arrays*, pages 24–30, 1996.

[27] P. S. Zuchowski, C. B. Reynolds, R. J. Grupp, S. G. Davis, B. Cremen, and B. Troxel. A Hybrid ASIC and FPGA Architecture. In *Proceedings of the International Conference on Computer-Aided Design*, pages 187–194, 2002.