Design Methodology for IC Manufacturability Based on Regular Logic-Bricks

V. Kheterpal, V. Rovner, T.G. Hersan, D. Motiani, Y. Takegawa, A.J. Strojwas, L. Pileggi
{vkheterp, vrovner, tgh, dmotiani, yoichi, ajs, pileggi}@ece.cmu.edu
Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15232

ABSTRACT

Implementing logic blocks in an integrated circuit in terms of repeating or regular geometry patterns [6,7] can provide significant advantages in terms of manufacturability and design cost [2]. Various forms of gate and logic arrays have been recently proposed that can offer such pattern regularity to reduce design risk and costs [2,4,9,11,12]. In this paper, we propose a full-mask-set design methodology which provides the same physical design coherence as a configurable array, but with area and other design benefits comparable to standard cell ASICs. This methodology is based on a set of simple logic primitives that are mapped to a set of *logic bricks* that are defined by a restrictive set of RET(Resolution Enhancement Technique)-friendly geometry patterns. We propose a design methodology to explore trade-offs between the number of bricks and associated level of configurability versus the required silicon area. Results are shown to compare a design implemented with a small number of regular bricks to an implementation based on a full standard cell library in a 90nm CMOS technology.

Categories and Subject Descriptors

B.7.1 Types and Design Styles

General Terms

Algorithms, Performance, Design, Reliability, Experimentation.

Keywords

Integrated Circuits, Regularity, Manufacturability, RET

1. INTRODUCTION

As CMOS technology continues to scale, systematic variations begin to dictate integrated circuit (IC) yield and performance. In order to achieve acceptable design quality, these variations must be reduced, or at least taken into account during the circuit and layout design flow [2].

DAC 2005, June 13–17, 2005, Anaheim, California, USA.

Currently, existing lithography tools are being forced to operate at their resolution limit. As a result, printability becomes greatly hampered and neighborhood-pattern dependent. To ensure an accurate transfer of sub-wavelength design features, and thus reduce the variability, a number of Resolution Enhancement Techniques (RETs) are currently being utilized. These consist of Optical Proximity Corrections (OPC), including Sub-Resolution Assist Features (SRAFs), Phase Shift Mask (PSM) techniques, and Off-Axis Illumination (OAI) schemes. Most importantly, however, existing design rules cannot guarantee a design adherence to RETs, such as PSM, and thus the post-layout processing steps cannot fully exploit the benefits offered by these techniques. In addition, as the lithography interaction distance continues to increase with each process generation, rule-based OPC has been replaced with a computationally intensive model-based OPC.

By introducing a regular fabric that is designed based on the restrictions placed by RETs and the lithography system, a RET-correct by construction design can be achieved. With this *micro-regularity*, we should note that the number of unique geometry patterns in a neighborhood is also decreased. These two benefits, offered by the fabric, ultimately reduce the systematic variations.

Field Programmable Gate Array (FPGA) and memory designs have relied on their *micro-* and *macro-regularity* (e.g. bit-cell repeatability) to address the manufacturability challenges posed by new technology nodes. Due to the limited number of unique shapes present, one can afford to perform RETs, simulation-based modeling and silicon verification for the small structures as they will ultimately appear, surrounded by the regular geometry neighborhood of other cells. It is for this reason that memories and FPGAs are often the first products to utilize a new manufacturing process.

Recently, new methodologies have been proposed for design of regular logic structures that can bridge the performance-cost gap between programmable devices and full-custom designs [11,12,13]. These designs provide an underlying regular fabric for the logic, which is generally constructed from homogeneous configurable logic blocks, and, like FPGAs and memories, allow them to be fine-tuned for manufacturability and performance based on the shared layers which are used for multiple designs and applications. Recent work on via-configurable regular fabrics [1,9] in particular has shown that fabrics built from a fully programmable universal block, capable of being configured to implement many different functions, can simplify the synthesis process while providing performance comparable to that of ASIC implementations. The simplified synthesis process, however, can result in poor silicon utilization, which quickly translates to larger designs and possibly increased yield loss due to random defects. Furthermore, these arrays of logic with fixed sets of building

blocks are unable to exploit the *macro-regularity* which might be inherent to a particular design.

In this paper, we propose a methodology for regular logic design that addresses design cost, printability, and manufacturability by constructing logic bricks from logic primitives in a manner by which they are regular by construction. We demonstrate that this methodology can offer efficient area utilization while providing the regularity comparable to fixed arrays.

2. GEOMETRIC REGULARITY

Requiring circuit fabrics to be constructed from a regular set of physical geometries is intended to provide more robustness with respect to process variations, and, therefore, offer a RET friendly layout. Of particular concern are those layers of the manufacturing process that are most susceptible to systematic variations, such as the front end of line (FEOL) transistors; particularly the predictability of the active and polysilicon layers. A number of micro-regularity constraints can be imposed for RET compatibility, such as: single orientation of Critical Dimension (CD) lines, constant poly pitch, and disallowing minimum width devices.

It is often claimed that the macro-regularity found in SRAM cores makes it is easier to ensure correctness over the range of process variations. To illustrate this regularity, consider the 2D Fast Fourier Transform (FFT) for the polysilicon layer of a commercial 90nm memory layout shown in Figure 1 (left). Note the single sharp peak at a spatial frequency of 2.5 (shapes per micron). The second peak represents a harmonic due to the 0-1 nature of the mask layout, but there is clearly a single fundamental frequency present. A similar experiment was conducted using the polysilicon layer for a standard cell layout portion of a chip in the same technology. As shown in Figure 1 (right), there is not a single spatial frequency component that can be attributed to the ASIC layout. Furthermore, it is interesting to note that the general shape of the FFT remained the same no matter what section of the ASIC was examined. The multitude of spatial frequencies suggests a lack of geometric regularity, which corresponds to difficulties with RETs, and significant systematic variations of CD parameters, such as effective gate length.

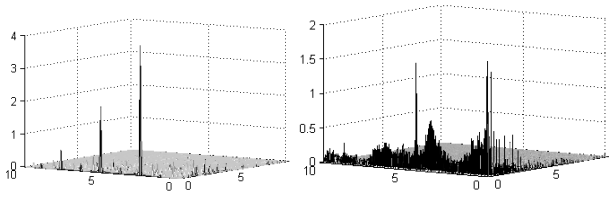


Figure 1: 2D FFT of the layouts for the polysilicon layers of SRAM core (left) and Standard cell ASIC (right).

The polysilicon gate forming the channel of the transistor is the smallest feature, CD, which needs to be printed on the wafer. In addition, the channel length of the transistor is directly related to the performance of the device, and exponentially related to the leakage of the device. Thus any variations of the CD lines need to be minimized. By requiring a single orientation for all of the devices, variations due to lens aberrations can be eliminated, and alternating PSM can be applied. Such orientation restrictions are becoming more prominent at the 65nm node, and we apply this constraint for our regular brick designs that are described in this paper.

We further constraint the polysilicon gates of our regular bricks to be placed onto a grid with a fixed pitch, or its multiple. By utilizing this constraint, the lithography process can be optimized to minimize the CD variation for a particular pitch. Moreover, a single pitch simplifies the application of SRAFs by eliminating forbidden pitches. The pitch selection is a crucial step in designing the fabric. The device pitch utilized in our regular fabric is based on the minimum metal-1 pitch available by the process. The rationale behind picking a polysilicon pitch based on the metal-1 pitch is that the source, drain, and gate region of the device will have to be inter-connected using metal layers..

The previous restrictions have been applied to enable the printability of the polysilicon layer. However, to reduce transistor performance variations, active layers would also have to be addressed. Our regular fabrics will not allow the use of minimum width devices, so as to reduce the number of corners present in the active layer. Specifically, this is done to reduce STI stress effects acting in the channel, and thus reduces the mobility variations.

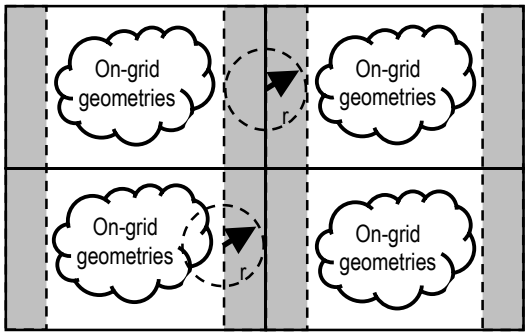


Figure 2: Regular Bricks. Micro-regularity constraints within bricks and compatible borders (shown in grey) provide macro-regularity compatible over radius of influence r.

We define a 'brick' to comprise a logic function created from a small set of logic primitives that are mapped onto a micro-regular fabric, as shown in Fig. 2. While the set of logic primitives has to be flexible to provide efficient implementation for the logic required, it should be small enough to significantly restrict the number of geometric shapes allowed within a brick. Constraining the number of geometric-shape primitives within a well-defined footprint provides more regular FEOL circuits and, ultimately, back end of line (BEOL) metal patterns [10]. Although the fabric is built from heterogeneous elements, as long as they share regularity by construction and offer well defined boundaries, as shown in Fig. 2, accurate models for the behavior of the bricks can be obtained before synthesis and placement.

3. BRICK LIBRARY

3.1 Logic Primitives

The selection of logic primitives for the bricks is crucial to the overall area and performance. In [1] it was shown that NAND and 2:1 MUXs can be used in different configurations to efficiently implement all of the three-input functions. For this reason, the logic primitive set that we have chosen for the experiments

described herein consists of a NAND, a 2:1 MUX, inverters and buffers. To improve performance and power efficiency, the MUX is implemented using pass transistor logic (PTL) where the input and output buffers can be omitted for signals that are local to the brick. The NAND is implemented using CMOS logic which provides high noise immunity and level restoring capabilities. Therefore, the combination of CMOS NAND and PTL MUX forms an efficient set of primitives [8].

3.2 Deriving a Set of Bricks

In general, the set of bricks required to efficiently implement a design is dependent on the inherent nature and structure of the design. We propose a technique for deriving the brick library that is optimized for a particular design, or application domain, as follows. Our brick derivation algorithm involves reducing the number of logic groups required to implement a design into a smaller set of via-configurable logic bricks. For example, the brick in Figure (3d) implements the logic groups in Figures (3a) and (3b) with only slight changes due to via-configurability. In general we would like to obtain a small set of bricks such that the

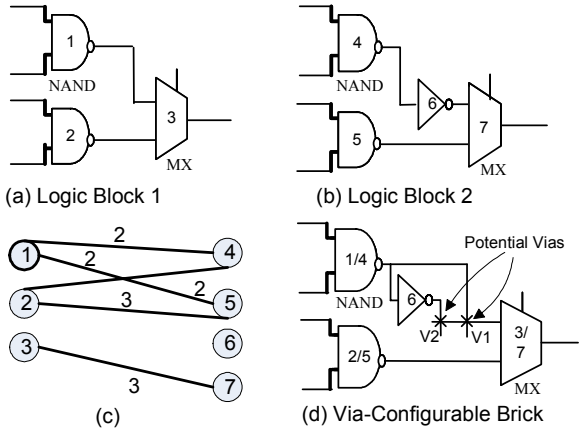


Figure 3 : Constructing via-configurable logic bricks

reduced heterogeneity of shapes produces a RET friendly physical layout. However, a very small set of bricks would imply more configurability per brick, and therefore, a larger footprint (more area). We explore this tradeoff as our algorithm produces a library with any desired number of bricks by:

- Identifying structural similarity between logic groups.
- Using this similarity metric to derive a set of bricks that cover the given logic groupings.

In order to identify similarity between two logic functions, we first merge them into a brick using weighted maximum matching in a bipartite graph. The similarity metric is then the average area efficiency of implementing the logic groups using the merged brick. The nodes in each partition of the bipartite graph correspond to the gates in each logic group, as shown in Figure (3c). Edges are inserted between nodes that perform the same function (e.g. in Figure (3c), nodes 1, 4 and 5 are NAND gates and nodes 3 and 7 are MUXs).

The weight assigned to each edge is computed using the similarity between the neighborhoods of the nodes adjacent to this edge. Once the graph is constructed, weighted maximum matching is applied to the graph. The mapping of gates obtained from this provides us with a brick that optimally implements both of the logic groups. For example, the maximum matching solution for the graphs of logic groups in Figs. 3a and 3b is shown in 3c, which corresponds to the brick in Fig. 3d. Brick 3d can be configured using via V1 to implement logic grouping 3a, and via V2 can be utilized to achieve the functionality of 3b.

To derive a set of bricks covering all of the given logic groups, we first construct a complete graph for which each node corresponds to a logic group. The cost on an edge is the similarity of the nodes adjacent to the edge, derived using the algorithm described above. The most dissimilar node in this graph is defined to be the node for which the sum of incident edge weights is the maximum. The following steps are applied to merge nodes in the graph until the number of nodes in the graph is equal to the required number of bricks:

- The most dissimilar node is removed from the graph. This process is repeated until there are only two nodes left in the graph.
- A brick covering these two nodes is derived using the algorithm described above.
- The logic blocks corresponding to these two nodes are removed from the initial set of logic blocks and the merged brick is inserted into the set.
- The complete graph is reconstructed.

We applied this brick derivation algorithm to study various tradeoffs when designs are implemented using varying number of bricks. Each design was first synthesized to a set of logic blocks, where a logic block performs up to a 3-input function. These logic blocks were reduced into a library of 3, 4, and N bricks. The

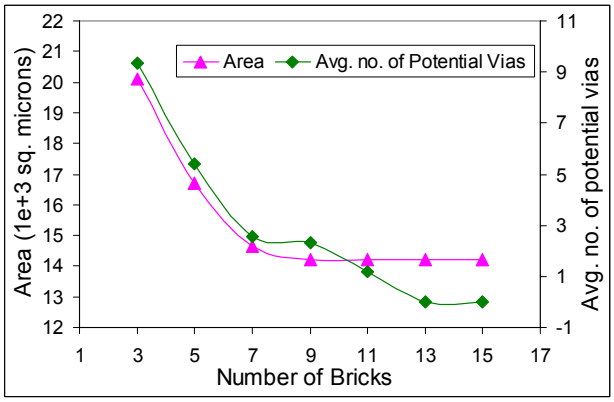


Figure 4: Area tradeoffs for a Firewire Controller

bricks were physically constructed to follow the micro-regularity fabric constraints. Figure 4 shows the die-area and average number of potential vias per brick in a small block design (Firewire controller) as the number of unique bricks is varied. Since the number of logic primitives is limited and each logic block performs up to a 3-input function, the number of unique logic blocks produced after synthesis is small. Hence, the algorithm scales well even for large designs.

We observed a similar trend to the one in Figure 4 for a number of other benchmarks. In general, we observed that there is little benefit in terms of area for using more than 10 unique bricks.

4. GENERIC BRICK LIBRARY EXAMPLE

While efficient design-specific bricks can be built to explore inherent regularity in designs, complete flows do not yet exist to accurately quantify the full benefit of such a design methodology. For this reason, we will first attempt to partially quantify the benefit of a regular brick methodology using a fixed generic brick library. For this experiment we have chosen to design the physical footprint of the bricks to be the same as a D flip-flop with scan. Given this constraint, we then determined the small set of bricks which could be used to implement a variety of block designs effectively, while still offering sufficient regularity.

Although the building block presented in [1] to implement all 3-input functions is too generic for our purposes here, we used a similar set of primitives to build more specific, efficient implementations for each of the 80 unique 3-input functions that are invariant under input permutations (these sets of functions are called p-equivalent by Harrison in [5]). This set of logic primitives consists of 2:1 MUXs, 2-input NAND gates, as well as buffers and inverters.

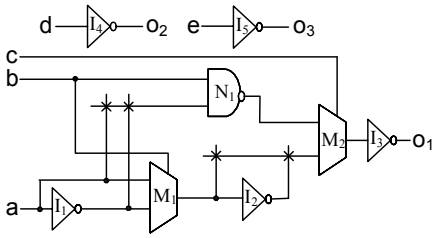


Figure 5: Example brick

These 80 implementations are reduced to a set of 5 unique via-configurable bricks using the algorithm described in Section 2. Figure 5 shows the schematic for one of the derived bricks. MUX M_1 is via-configurably connected to MUX M_2 while the NAND gate drives the other input of M_2. MUX M_2 drives the primary output of the brick through inverter I_3. Extra inverters (I_4 & I_5) are added to each brick to make the footprint of each brick identical to that of a D flip-flop with scan. These inverters are used for local and global signal buffering. Transistor sizing is performed in order to improve the performance of each individual brick for the given footprint. For example, in Figure 5, gate N_1 (NAND) is sized to optimally drive M_2 (MUX). Similarly, gate M_1 (MUX) is sized to drive either I_2 or M_2. This load predictability enables efficient sizing of components within the brick.

The other 4 bricks are similar to this brick in terms of components used, but differ in the subset of 3-input functions they can implement. Sizing on each brick can be performed manually since the number of bricks is small and the number of unique transistor sizes should be small to maintain micro-regularity between bricks.

The set of five bricks along with the D flip-flop forms a generic brick library that is comprised of only six bricks. In addition to the 80 single output functions, we can also use these primitives to efficiently build a full-adder with the same footprint as the other bricks. The footprint of each brick is the same, and their height is comparable to a standard cell row. Furthermore, the physical design for each brick is dictated by the fabric micro-regularity. Figure 6 (left) shows the layout for the brick in Figure 5, and the corresponding 2D FFT of the overall regularity for a design based on the 6 generic bricks (right).

5. EXPERIMENTAL REGULAR BRICK FLOW

In order to characterize the area and performance of the designs utilizing the brick library derived in section 4, we employed the flow shown in Figure 7, which maps an RTL description of a design to a DRC clean implementation using an array of bricks.

We start with a restricted library composed of the primitive cells described in the previous section (NAND, 2:1 MUX, buffers and inverters). Essentially, these are primitive cells which ultimately implement the logic functions within the bricks. The library is further restricted such that each cell has a fixed drive strength, which is representative of its behavior within a brick. We use Design Compiler from Synopsys to perform logic optimization and technology-mapping to this restricted library in order to obtain a gate level description of the design.

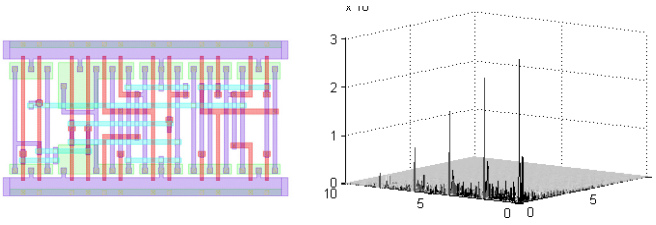


Figure 6: Layout for the brick in Figure 5 (left) and 2D FFT of polysilicon layout constructed from bricks (right).

Given the gate-level netlist, we obtain an ASIC-style global placement of the design using a physical synthesis tool [3], including in-place resynthesis, buffer insertion and wirelength optimization. The library for physical synthesis includes six buffers of different drive strengths in addition to the fixed drive strength primitive cells. The result of this stage is an ASIC-style global placement that has been optimized for performance, area, and routability based on physical information.

Next, we pack the primitive cells into the predefined set of bricks using the information from the global placement. The objective of this step is to cluster the primitive cells into legal bricks such that a global placement of bricks can be derived by minimally perturbing the placement of primitives. We begin by generating an illegal packing solution where each primitive might get duplicated into multiple bricks. We define a duplication cost for each primitive cell to be the cost involved in duplicating it into multiple bricks. The cost of each brick is the sum of the duplication costs of the primitives that it packs together. The cost function also takes into consideration the timing criticality of the primitive cells and tries to minimize perturbation of the ASIC-style global placement.

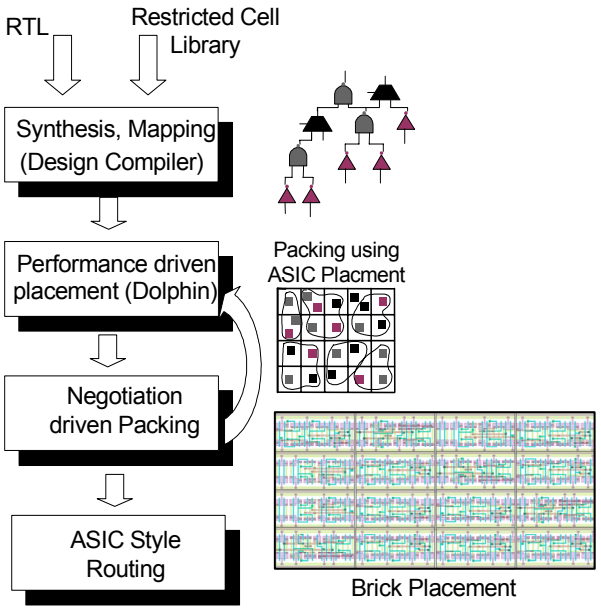


Figure 7 : Brick based RTL to GDS-II synthesis flow

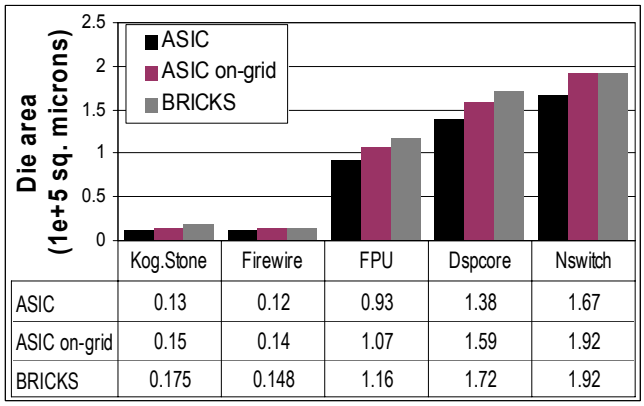
We resolve this illegal packing into a legal one in an iterative fashion, where all of the iterations involve the following steps:

1. Computation of new duplication costs for each primitive cell. This duplication cost is the weighted sum of *N* (number of times this primitive cell was duplicated in the previous iteration) and the iteration number *I*.
2. Generation of a new packing solution based on the computed duplication costs. This is done by greedily creating lowest cost bricks by packing timing critical cells first and then proceeding to the non-critical cells.

As iterations proceed (*I* increases), the average duplication cost of each primitive cell increases and thereby guarantees convergence to a final solution with very little duplication and minimal perturbation to the global placement. To further minimize the loss in performance due to the motion of cells, we perform packing in an iterative loop with physical synthesis. For each iteration, the packing restricts the locations of a few bricks to specific chip regions. Physical synthesis is repeated with these restrictions to provide new locations for the remaining cells, and to rebuffer and restructure logic to meet the performance and area constraints. This iteration loop is repeated until all the cells have been packed into bricks and each brick has a legal placement. ASIC-style global and detailed routing is then performed on the regular array of bricks. It should be noted that this partially modified ASIC-standard-cell flow is clearly suboptimal for implementation based on regular bricks, but is the best physical design option available to us at this time. Physical synthesis flows for regular logic fabrics are part of our on-going and future work.

6. RESULTS

Following the flow described in section 5, we benchmarked a set of circuits implemented using our regular bricks from section 4. Figure 8 shows the area comparison for the benchmarks implemented using our six regular bricks vs. a commercial 90nm standard-cell library implementation. In contrast to our brick cells, the standard cells in the library used for the ASIC flow *do not restrict* the polysilicon, metal-1 and contacts on a strict grid. It should be noted that such restrictions on standard cells would incur a 15-25% area penalty. For example, the D flip flop with scan used for our regular brick flow is 20% larger than the identical flip flop in the standard cell library due to our micro-regularity constraints. While such constraints are not required at 90nm, they will be necessary for standard cells at 65nm and below, which are the target technologies for our regular bricks. For this reason, we compare our gridded brick results with both a non-gridded ASIC implementation, and with the ASIC area results scaled by 15%.



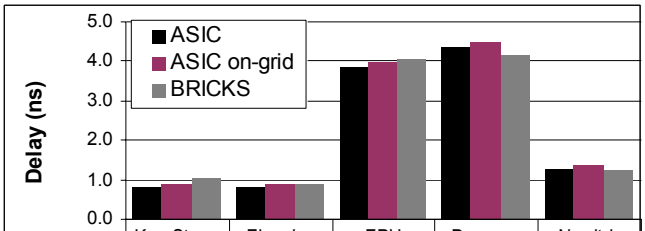
	Kog.Stone	Firewire	FPU	Dspcore	Nswitch
ASIC	0.13	0.12	0.93	1.38	1.67
ASIC on-grid	0.15	0.14	1.07	1.59	1.92
BRICKS	0.175	0.148	1.16	1.72	1.92

Figure 8 : Generic Bricks - Block area. The ASIC on-grid results are approximates based on a 15% scaling factor.

There is an average of 26% increase in the die-area of designs implemented using the brick methodology as compared to the standard cell design without grid constraints. This is primarily due to the 15-25% area penalty incurred by utilizing fixed polysilicon and metal-1 pitches. In addition, since the bricks are comprised of a very limited set of primitives, there can be some area inefficiencies when a design requires a high amount of specificity and simplicity in terms of cell functionality. An example of such a design is a Kogge-Stone adder, where the area penalty is as high as 35%. For such cases, our brick derivation algorithm (Section 3.2) can be applied for a specific design to generate a set of specific bricks which would provide a more efficient implementation. This is part of our future work, and some early results are shown in Fig. 10. It should also be noted, however, that the bricks are *more* area efficient than standard cells in terms of performing complex three-input functions. This provides a significant advantage in the network switch (Nswitch) benchmark, where the area is the same as the ASIC (on-grid) implementation.

Figure 9 shows the performance comparison for the block designs in Figure 8. The Y-axis is the average delay of the 10 most critical paths in nanoseconds. The delay is measured after parasitic extraction of detailed routed GDS clean implementation of each block. Despite the fact that the bricks use a very limited subset of cells from a typical standard cell library, both in terms of sizing as well as functionality, the performance of the brick-based designs is competitive to that of the ASIC implementations, and even slightly better in some cases. The average delay increase over all the benchmarks is about 7%. The maximum increase in delay is

about 26% for the Kogge-Stone adder. As with the area comparison, this is primarily due to the datapath design requirements for a very specific set of cells to provide the necessary functionality. The larger die-area of the Kogge-Stone also contributes to increase in parasitics which further cause a loss in performance. In contrast to this, the brick cells perform better (5%) at designs like Dspcore where complex three or more input functions dominate. The primary reason for this improvement is because the brick cells perform complex 3-input functions using un-buffered pass-transistor logic, having to buffer the signals only while routing them to other bricks.



	Kog.Stone	Firewire	FPU	Dspcore	Nswitch
ASIC	0.830	0.820	3.830	4.350	1.270
ASIC on-grid	0.89	0.88	3.96	4.50	1.37
BRICKS	1.040	0.890	4.081	4.144	1.249

Figure 9 : Generic Bricks - Performance

7. CONCLUSIONS & FUTURE WORK

We proposed a new regular logic brick design methodology which attempts to relieve the increasing stress on resolution enhancement techniques for sub-wavelength lithography. We have shown that a small set of distinct configurable logic blocks (bricks) is sufficient for an efficient implementation of a design. This set of bricks is optimized for manufacturability through a set of RET-friendly design rules that we propose. We have characterized the area and performance of designs implemented using the brick library and compared them with ASIC implementations in 90nm technology. Even with our simple prototype design flow, results indicate that our method is efficient in terms of both, die-area and performance. Our future work explores the application of the brick derivation flow (Section 3.2) to derive optimal bricks for particular designs. Figure 10 shows an example of the Kogge-Stone adder implemented with design-specific bricks. The x-axis corresponds to the number of bricks used in each implementation, and the y-axes show die-area as well as the average delay of the 10 most critical paths. We also show the area and delay of the ASIC and generic brick library (Section 4) implementations. We see that an implementation with more than just 8 unique "design specific" bricks is extremely competitive with the ASIC flow. Furthermore, we produce designs with extreme physical regularity (far fewer distinct geometry shapes in the physical layout) compared to an ASIC, thereby improving printability and manufacturing yield.

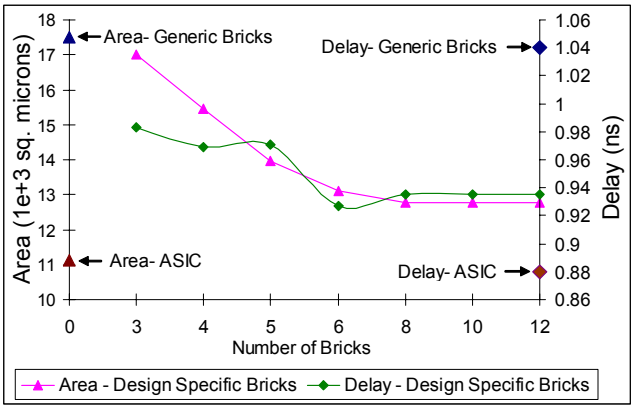


Figure 10: Kogge-Stone example area and performance with design-specific bricks.

8. REFERENCES

[1] A. Koorapaty et al. "Exploring Logic Block Granularity for Regular Fabrics", Proceedings of the Design, Automation and Test in Europe Conference (DATE), Feb. 2004.

[2] L. Pileggi, H. Schmit, A. J. Strojwas et al., "Exploring regular fabrics to optimize the performance-cost trade-off". Proceedings of the ACM/IEEE DAC, June 2003.

[3] http://www.monterey.com/products/dolphin.html.

[4] Fan Mo, Robert K. Brayton, "Whirlpool PLAs: a regular logic structure and their synthesis", Proceedings of the 2002 IEEE/ACM ICCAD

[5] M. A. Harrison, Introduction to Switching and Automata Theory, McGraw-Hill, 1965.

[6] M. Palusinski, A. J. Strojwas and W. Maly, "Regularity in Physical Design", GSRC Workshop, Las Vegas, NV, June 17-18, 2001

[7] A. J. Strojwas, "Process-Design Interaction Modeling Based Design for Manufacturability", Tutorial, Design Automation Conference, June 2003.

[8] Yamashita, S. et al., "Pass-transistor/CMOS Collaborated Logic: The Best of Both Worlds", VLSI Circuits, 1997. Digest of Technical Papers.

[9] Yajun Ran and Malgorzata Marek-Sadowska, "On Designing Via-Configurable Cell Blocks for Regular Fabrics", Design Automation Conference (DAC) 2004.

[10] V. Kheterpal, A. J. Strojwas, L. Pileggi, "Routing Architecture Exploration for Regular Fabrics", Proceedings of the ACM/IEEE DAC, June 2004.

[11] T. Okamoto, T.Kimoto, N. Maeda, "Design Methodology and Tools for NEC Electronics' Structured ASIC ISSP", Proceedings of the ISPD, April 2004.

[12] D. Sherlekar, "Design Considerations for Regular Fabrics", Proceedings of the International Symposium on Physical Design (ISPD), April 2004.

[13] K.-C. Wu, Y.-W. Tsai, "Structured ASIC, Evolution or Revolution?", Proceedings of the International Symposium on Physical Design (ISPD), April 2004.