

Via-Configurable Routing Architectures and Fast Design Mappability Estimation for Regular Fabrics

Yajun Ran

Department of Electrical and Computer Engineering
University of California, Santa Barbara, CA 93106
Email: ranj@ece.ucsb.edu

Malgorzata Marek-Sadowska

Department of Electrical and Computer Engineering
University of California, Santa Barbara, CA 93106
Email: mms@ece.ucsb.edu

Abstract—In this paper, we describe a new via-configurable routing architecture which shows much better throughput and performance than the previous structures. We demonstrate how to construct a single-via-mask fabric to reduce further the mask cost, and we analyze the penalties which it incurs. To solve the routability problem commonly existing in fabric-based designs, an efficient white-space allocation scheme is suggested, which provides a fast design convergence and early prediction of the circuit mappability to a given fabric.

I. INTRODUCTION

In recent years, rapid technological progress has made it possible to integrate more than one-billion transistors on a single chip. However, the manufacturability issues have presented a serious challenge for nanometer technologies. Pattern distortion in sub-wavelength lithography requires tremendous effort for post-layout optical proximity correction (OPC). Erosion and dishing in chemical-mechanical polishing (CMP) processes can cause significant interconnect/dielectric thickness variations. Tackling these problems in the design stage is not easy since they are related to neighborhood layout patterns. An accurate, efficient analysis of layout patterns is difficult to do during design optimization steps. Moreover, the extremely high cost of design and manufacturing has effectively dissuaded designers from using the most advanced technologies.

An alternative solution to these problems is to design circuits based on pre-fabricated fabrics, i.e. *structured-ASICs*. A fabric usually consists of repeated logic blocks. It has pre-fabricated transistors and partially fixed metal layers. Due to the high layout-pattern regularity, a fabric is much easier to manufacture. A base block in a fabric can be well characterized, and OPC effort for a block is small. Also, the fixed metal segments are uniformly distributed and thus help to reduce variations caused by the CMP. Moreover, a designer needs only to customize a few metal and/or via masks to complete a design, which can greatly reduce the mask cost as well as the design and manufacturing time. With the use of hard-wired switches, the performance can be significantly improved over that of FPGA-based designs.

There are many types of structured-ASICs. Several base logic blocks have been proposed in the literature, such as a via-configurable lookup table (LUT) [1], LUT/NAND3-mixed block [2], gain-based semi-universal-logic block [3]. There are also works on fixed routing structures such as direct-connected crossbar [1] and via-configurable routing [4]. However, the literature offers little discussion about the throughput of existing routing structures. For a fabric with fixed wire segments, routability is a more serious problem than in customized routing. The inflexibility of pre-designed wiring may significantly degrade circuit performance and require more chip area.

Given fixed logic and routing structures in a fabric, a densely-packed design might not be routable. To solve this problem, some blocks might be left only partially used to relieve congestion. This

is called white-space allocation in cell-based designs. White-space in a fabric-based design does not affect the regularity of the layout patterns since all the patterns are still there, though unused. Early white-space allocation can help to achieve faster design convergence by predicting and relieving congestion before routing. For a fabric with fixed capacity, both the logic and the available routing resources affect whether a design can be mapped to the fabric. In this case, a fast and tight routing-resource-aware white-space allocation scheme is essential for quick estimation of design mappability and design convergence. Moreover, the routing-resource-aware design methodology can help a fabric vendor to explore quickly the trade-off between the number of logic blocks and the number of routing tracks. When trading off routing resources, a vendor can balance the die cost by deciding whether to put more tracks on one layer or to use more metal layers. Consequently, a series of well-tailored fabric chips with different routing capabilities, not only logic capacities, can be provided such that designers can choose the best match between a design and a fabric. Given a good estimation on routing efficiency, this area estimator can be very useful for all the fabrics with fixed routing structures, such as FPGAs and via-programmable fabrics, as well as fabrics with metal-programmable masks.

Most traditional white-space problems in literature come from fixed-die cell placement. In a fixed-die cell placement the amount of white-space is given, and the problem is how to properly distribute it. In [5], white-space is evenly distributed. In [6]–[8], white-space is distributed to congested regions based on routing estimation. In [9], a congestion-minimizing framework allocates white-space based on Rent's rule with no explicit congestion prediction. With the guide of Rent's rule, efficient clustering and placement techniques were developed to reduce congestion in FPGA-based designs [10], [11]. All of these approaches try to reduce congestion using the given white-space but they cannot completely eliminate the routing congestion. Our white-space allocation problem is different because we want to quickly determine how much white-space is needed and where it should be put to obtain a routable design.

In this paper, we first study the via-configurable routing architectures. We propose a new basic routing structure which provides larger throughput than do the routing architectures in [1] and [4]. We show also how to construct a single-via-mask fabric and discuss the penalty it might impose on the fabric's throughput. We then propose a fast white-space allocation scheme to determine how much white-space we need to add, and where to add it, to completely eliminate, not just reduce, the routing congestion. In our design flow, a fast global routing is performed to obtain an accurate congestion estimation. We allocate white-space in-place by calculating routing demand-supply relations. We use our efficient incremental placement and global routing algorithms to move cells away from congested regions. Finally, we perform detail routing to find out whether a

design is indeed routable. These steps complete the flow and validate our approach.

The paper is organized as follows. In Section II we introduce our new base routing structure. In Section III we show the construction of a single-via-mask fabric. We introduce the test fabric architecture in Section IV and describe our routability-driven design flow in Section V. We report experimental results in Section VI. Section VII summarizes the contributions made by our research.

II. BASE ROUTING STRUCTURES

In this paper, we assume that metal-1 (M1) and metal-2 (M2) layers in a fabric are used for implementing cell functions. An input or an output of a gate is related to an M2 segment which can be accessed through vias to a metal-3 (M3) segment. We focus on the routing structures above the M2 layer.

In a via-configurable routing architecture, there is a potential via connection whenever segments from two adjacent metal layers intersect. As a result, switching a signal from vertical to horizontal direction, or vice versa, can be easily realized through a via connection between the two intersecting segments. To continue a signal in one direction in a fixed wiring scheme, some dedicated short segments in the adjacent layer, called *jumpers* (shown in Fig. 1), are commonly used [1].

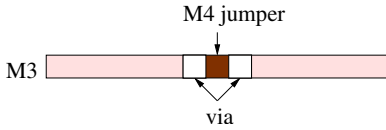


Fig. 1. A jumper connects two segments in one direction.

When planning wire segments, we have the choice to stagger or align the segments. Due to the use of short jumpers in a via-configurable routing structure, closely staggered wires are not favored. Figure 2 shows an example. The jumpers in a staggered wire segment scheme (Figure 2(a)) are distributed. As a result, the jumpers become the obstacles for vertical wires since they are in the same layer. Therefore, we adopt an aligned wire segment scheme as shown in Figure 2(b) where jumpers are aligned and form a dedicated jumper channel.

Aligned wire segments naturally form crossbars (as shown in Figure 3(a)) which provide the highest switching flexibility [1], [4]. In a crossbar structure, any horizontal segment can connect to any vertical segment, and vice versa. In a fabric, the width and height of a crossbar are usually determined by the size of a base block. A crossbar on top of a block provides pin access to the cells in the block, relays a signal in the same direction, or switches a signal to the other direction. In fixed wiring structures, there might be some dangling wires. Figure 3(b) shows an example. The signal only uses partial segments a and c . Partial segments b and d are redundant, called *dangling wires*. Dangling wires incur extra capacitance, degrade performance, and increase power consumption of a circuit. Using finer granularity of wire segments could reduce the impact of dangling wires.

Once the size of a base crossbar has been decided, the design of a routing structure in a fabric is reduced to determining the interconnections between two crossbars. In [1], two crossbars connect to each other using parallel jumpers. This is the *direct-jumper crossbar* structure which we depict in Fig. 4(a). Two types of wire segments are available in the crossbar [4], as shown in Fig. 4(b). The shorter segment is used only to switch a signal in the local crossbar or to access a local pin. The longer segment connects to a segment in

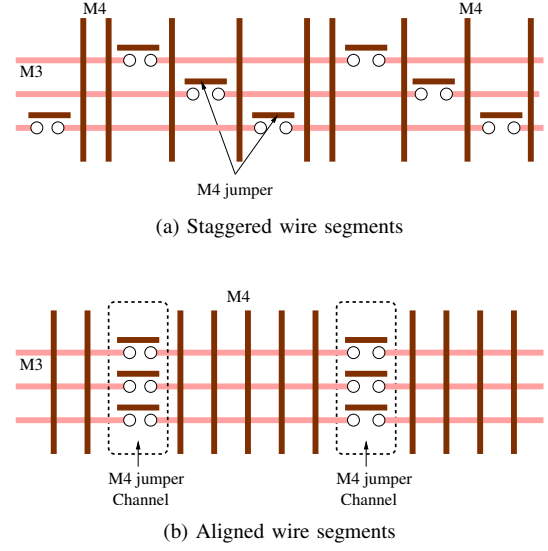


Fig. 2. Staggered versus aligned wire segments with jumpers.

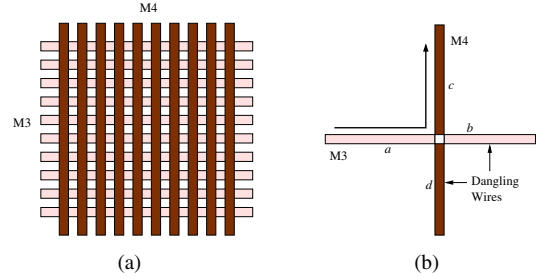


Fig. 3. A crossbar (a) and dangling wires (b).

an adjacent crossbar by a perpendicular jumper. We call this scheme a *jog-jumper crossbar* structure. Using a perpendicular jumper helps to pack two crossbars one wire-pitch closer, which increases the capacity of a crossbar by 2. From the figures, one can see that the jumpers occupy several tracks and reduce the number of available routing segments. Suppose that the width of a crossbar is n (in terms of wire pitches) and double vias are used for a connection. The number of available routing segments is $n - 4$ for the direct-jumper, and $n - 2$ for the jog-jumper crossbar. However, the number of segments which can connect to adjacent crossbars is only $(n - 2)/2$ for the jog-jumper crossbar because short segments can only be used locally.

In both direct-jumper and jog-jumper structures, all the wire segments in one metal layer are oriented in the same direction. As a result, a jumper from another layer is necessary to connect two wires in the same layer. In this paper, we propose a new crossbar structure, a *crossover crossbar*, shown in Fig. 4(c). In the crossover crossbar structure, a base crossbar is rotated by 90 degrees with respect to its neighbor. In this way, two crossbars can be stacked at the ends. The crossover portion provides potential via connections since two wire segments leading in the same direction are now residing in two different metal layers. A route will alternate between the two layers when it passes through the continuous crossbars. Compared to the jumper crossbars, the crossover crossbar provides $n - 2$ segments, and all of the segments can connect to adjacent crossbars.

The advantages of the crossover crossbar structure are summarized as follows.

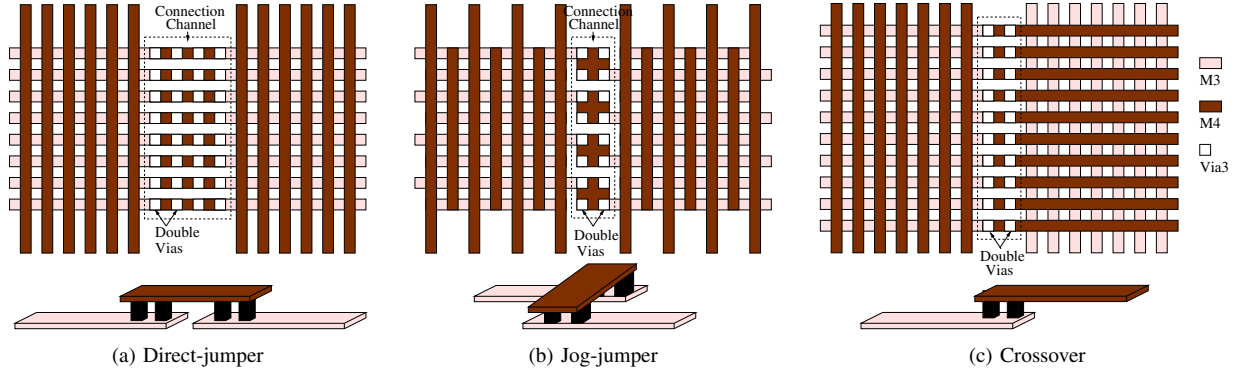


Fig. 4. Different crossbar structures

- *Higher capacity.* The crossover crossbar structure has $n - 2$ connecting segments compared to $n - 4$ in the direct-jumper crossbar and $(n - 2)/2$ in the jog-jumper crossbar.
- *Fewer vias and smaller via resistance.* The crossover connection needs only two vias, whereas both jumper connections need four vias (assuming that double vias are used for each connecting point to improve yield). As a result, the via resistance in the crossover connection is only $R/2$, but it is R for both jumper connections where R is the resistance of a single via.
- *Flexible connection to the upper layers.* In both jumper crossbar structures, a net route using an M3 segment needs a specific M4 segment to reach an M5 segment if we want to route it in the same direction using M5. However, since in the crossover crossbar structure, half of the net's route uses M4, a direct connection to an M5 segment is possible without wasting an M4 segment.

The upper layer M5 and M6 can use the same crossbar schemes. In general, the segments in M5 and M6 layer have larger granularity and are used to route long nets.

III. SINGLE-ROUTING-VIA-MASK FABRICS

In the routing structures discussed above, all the metal masks are fixed, and all the via masks are programmable. One way to reduce cost further is to use fewer via masks. In this section we shall describe how to construct a single-routing-via-mask fabric.

Suppose we use six metal layers for a fabric, let via3 (between M3 and M4) be programmable and other via masks be fixed. To make all the wire segments usable for routing, all the wires below M3 must be brought to M3, and the wires above M4 must be brought to M4 by fixed connections (fixed via4, via5 and wires) to access the potential vias between M3 and M4 segments.

We attach two dedicated M4 segments, called *accessing segments*, to both ends of each M5/M6 segment. Fig. 5(a) shows an example. The M5 segment is connected to the accessing M4 segment by fixed vias (via4). Because every accessing M4 segment has several potential connections to the M3 segments by programmable vias (via3), the M5 segment can connect to the M3 segments. Going up through one more via (from M3 to above M4), the M5 segment can also connect to other M4 segments. Moreover, through the accessing M4 segment of another M5/M6 segment, a signal can go back to the upper layer again. The red line in Fig. 5(a) shows a path of a signal, from M5 down to M4, M3, and then up to M4 and M5 layer. When an accessing M4 segment is parallel to an M5 segment (see Fig. 5(b)), a route may take a longer detour. Similarly, an M6 segment is connected to the

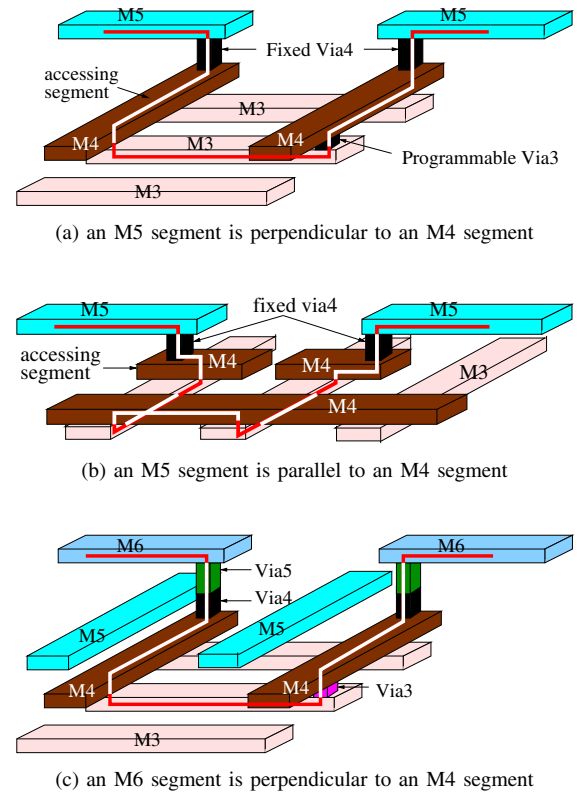


Fig. 5. M5/M6 segment connection in a single-via-mask fabric. Bold lines represent the potential signal path.

accessing M4 segment by stacked vias (via4 and via5) to access the potential via sites between the M3 and M4 (shown in Fig. 5(c)). In this way, a route can make use of fixed M5/M6 wire segments but will need to come down to M3/M4 each time it changes to another wire segment.

The penalty of a single-routing-via-mask fabric can be summarized below.

- Two dedicated accessing M4 segments have to be attached to each M5/M6 segment at two ends. As a result, the M4 segments available for local routing are reduced, which could worsen routability.
- An M3/M4 segment can only access an M5/M6 segment at

its terminal (through the accessing segment). However, it can connect to a M5/M6 segment in the middle by programmable vias. This causes some performance penalty but significantly improves routability.

IV. FABRIC ARCHITECTURE

The various routing structures are tested on a via-configurable gate array (VCGA) [12] shown in Fig. 6(a). A VCGA consists of an array of via-configurable blocks (VCBs). Each VCB contains four basic logic elements (BLEs). A BLE contains one via-configurable cell (VCC) and two via-configurable inverter arrays, which can implement both combinational and sequential functions customized by a via1 mask. There are two types of BLEs, odd-BLE (O-BLE) and even-BLE (E-BLE), which have different orientations. Fig. 6(b) and (c) show the M2 patterns for an E-BLE and O-BLE, respectively. All the inputs and outputs of a BLE are provided through the M2 segments. To reduce the routing stress on the top metal layers, the VCGA provides also a few inter-BLE M1/M2 wire segments for short connections between neighboring BLEs as shown in Fig. 6(d). A via-configurable switch box is used to switch a signal between horizontal M2 segments and vertical M1 segments.

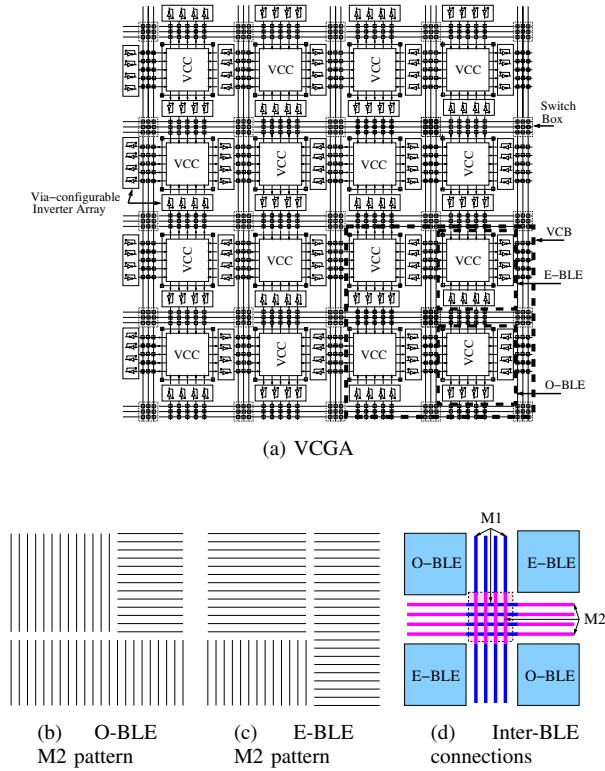


Fig. 6. Via-configurable gate array

V. FAST DESIGN MAPPABILITY ESTIMATION AND VALIDATION

Although the fixed routing structures can lead to much better manufacturability and significantly reduce mask cost, they might cause serious routability problems due to their inflexibility. When we map a design to a fabric with fixed routing structures, we want to know quickly whether the mapping is feasible. This not only depends on whether the fabric contains enough logic for the design, but also depends on whether the fabric can provide enough routing resources.

The mappability estimation should consider the required white-space to accommodate the routing needs.

In this section we describe our routability-driven design flow. Fig. 7 outlines our flow. Given a circuit netlist, we first pack it into circuit blocks such that each of them can be mapped to a master block. The main objective of the packing step is to achieve high design density and low inter-block connections. We adopt the algorithm in [13]. We then place circuit blocks. After we have an initial cell placement, we do a fast global routing by constructing Steiner trees for each net. We calculate the routing usage over each block and obtain an accurate estimation of congestion over each region. By combining the routing demands over each row and column with the routing supply provided by the given routing architecture, we determine whether and where extra master blocks must be inserted to provide enough white-space to eliminate congestion. After allocating white-space, we can immediately know whether the design is mappable at this stage by comparing the size of the fabric and the size of the design with required white-space. To validate our white-space allocation algorithm and complete our design flow, we perform an incremental cell movement to reduce congestion. With a routing tree already constructed, we can have an accurate evaluation on congestion change due to the cell movement. The evaluation is also very fast since it involves only a few partial routing tree segments which connect to the moved cell. The routing trees are incrementally updated when we move a cell. At the end, we rip-up nets passing through congested regions and re-route them using a more effective, yet more expensive method, e.g., maze search. The global routing results are then fed into a detailed router to complete the design. In the following section, we shall describe in detail the congestion-aware white-space allocation and incremental cell movement.

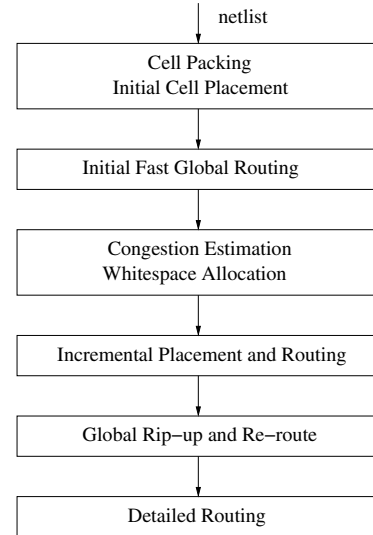


Fig. 7. Routability-driven design flow.

A. Congestion-Aware White-space Allocation

After initial global routing, we have an accurate estimation of routing usage over each region. When the routing demands exceed the routing supply, congestion occurs. A few congested spots could be relieved by re-routing some nets if there are extra routing resources nearby. When there are excessive overflows, e.g., very limited routing resources in some routing architectures, or highly complicated inter-connects in some designs, we have to allocate some white-space, i.e., add more master blocks, to accommodate the routing needs.

Conceptually, we need to answer two questions for white-space allocation: how much white-space should be allocated, and where they should be put. The first question is hard to answer without considering the capability of white-space reducing congestion. For the second question, ideally, white-space should go to congested regions. However, the manner of implementation is not trivial. If we force the white-space at congested regions and move away cells which were originally there, it's hard to guarantee that no new congestion will occur, particularly when the stresses of routing in regions around congested spots are also heavy. This method is equivalent to adding extra master blocks at the boundary of the original placement and then moving them to the congested regions as shown in Fig. 8(a).

To overcome the above difficulties, we propose a fast row/column based in-place white-space allocation scheme. We directly insert a row (column) of master blocks into the congested region and shift the other rows (columns) (Fig. 8(b)). As a result, the original cell placement is shifted. The existing routing is shifted or expanded across the allocated row/columns. Therefore, the original congestion estimation for each row/column does not change. The inserted empty blocks provide a space near the cells to be moved from congested regions. In this way, the white-space is quickly allotted to congested regions without generating new congestion. Moreover, the insertion of new rows (columns) is based on in-place routing demand-supply estimation. Hence the amount of white-space needed can be accurately determined.

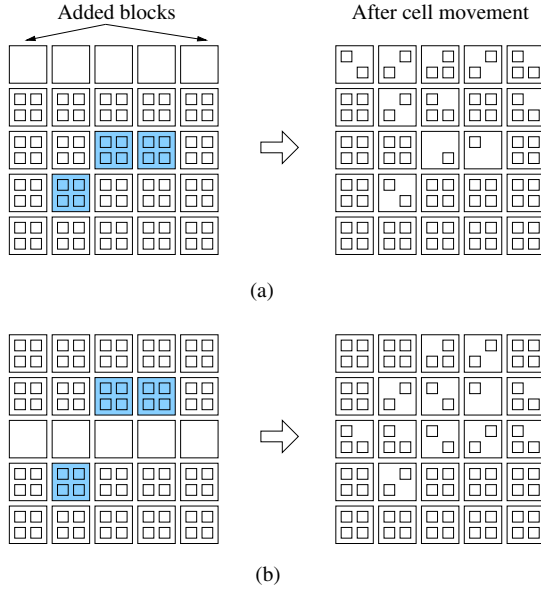


Fig. 8. White-space allocation and cell movement to eliminate congestion. Filled blocks represent congested blocks.

To determine where to insert a row, we scan rows from bottom to top, accumulating the routing usage over rows. An empty row is inserted when the following demand-supply condition is satisfied.

$$\sum_{r=i}^j U(r) > \alpha((j-i+1)+1)C_h \quad (1)$$

$$\begin{aligned} U(r) &= \sum_{i=1}^W u_h(i, r) \\ C_h &= Wc_h \end{aligned} \quad (2)$$

where $u_h(r, i)$ is the number of used horizontal track segments at

block $B(i, r)$, c_h is the number of total horizontal track segments at a block, and W is the number of blocks in a row. In other words, $U(r)$ is the total routing usage of r -th row, and C_h is the total routing supply of a row. The scan goes from row i to row j . α is the routing efficiency factor. Because not every net can be routed completely straight, and extra jogs consume routing resources, α is set around 0.7 for our experiments. The left-hand side of the inequality represents the total routing demands over the scanned rows, and the right-hand side represents the total routing supply of the scanned rows plus the empty row to be inserted. When the above condition is satisfied, we insert an empty row after the most congested row among the recently scanned rows. We then move the scanning point to the next unscanned row and continue the above procedure.

In practice, we observe that a large span of scanned rows may not be effective in reducing congestion. The reason is that cell movement is difficult due to other constraints if available empty sites are far distant from congested spots. Therefore, we choose a local scan window of width Δ . Only when the condition (1) is satisfied over a local scan window, we insert an empty row. If not, we shift the scan window one row forward. The width of a scan window is a function of circuit width W , i.e., $\Delta = \beta W$, where β is a constant around 0.1 in our experiments. Algorithm 1 shows the pseudo-code of our white-space allocation scheme.

Algorithm 1 Empty Row Allocation

```

1: calculate routing usage  $U(r)$  for each row  $r$ ;
2:  $\Delta = \beta W$ ;
3: sum_usage := 0;
4: nScannedRow := 0;
5: { $H$  is the circuit height}
6: for  $r := 1$  to  $H$  do
7:   sum_usage := sum_usage +  $U(r)$ ;
8:   nScannedRow++;
9:   if sum_usage >  $\alpha \times (\text{nScannedRow}+1) \times C_r$  then
10:     $H++$ ;
11:    idx := the most congested row between row ( $r$ -
      nScannedRow) and row  $r$ ;
12:    insert a row of blocks at row (idx+1);
13:    shift cells in row (idx+1) to row ( $H$ ) up by one row;
14:    sum_usage := 0;
15:    nScannedRow := 0;
16:   else if nScannedRow >  $\Delta$  then
17:     sum_usage := 0;
18:     nScannedRow := 0;
19:   end if
20: end for

```

Empty columns are inserted in a similar way. By inserting rows/columns of blocks based on the initial congestion estimation, we hope to provide enough tight routing resources for a circuit so that no design iterations are required to achieve routable designs.

The only problem of this white-space allocation scheme is that it might allocate more than enough white-space. But from experiments we see it provides a tight estimation. The simplicity and great efficiency make it very useful for fast design mappability estimation.

B. Increment Cell Placement and Global Routing

After congestion estimation, and possibly white-space allocation, we have some empty sites around the congested regions. If no extra blocks are allocated, it usually means the routing congestion can be eliminated by local cell swapping. In either case, we incrementally

move away a cell from a block which has hard constraint violations or congestion.

In our flow, the incremental cell movement algorithm starts with evaluating the costs for each block and inserting them in a priority queue where the block with the largest cost is at the top. Each time we choose a cell from the block with the highest cost it implies that moving away the cell can result in the largest cost reduction for that block. The movement stops when no gain can be achieved. We adopt the move approach from [14], where incremental placement is used for layout-driven optimization. We try the following cell movements:

- Move to a neighboring block.
- Move to a fanin block of the cell.
- Move to a fanout block of the cell.
- Move to a block where its topological sibling cell lies in.

We evaluate all the possible moves and each time choose the best one.

Unlike [14] which uses a probabilistic wire-length model, we utilize the existing global routes and hence have a deterministic congestion measure. When a cell is moved to another block, the pins belonging to the cell move accordingly. This invalidates the original routing trees. However, when a cell is moved only one block, e.g. moving to one of its four nearest neighbors, the routing trees are only locally changed and can be updated quickly. Fig. 9 shows an example where pin *A* is moved to a right block, and edge *a* moves to *a'* and *b* is discarded. In this example, we need only to examine the routing grids through which the edges *a*, *b* and *a'* pass.

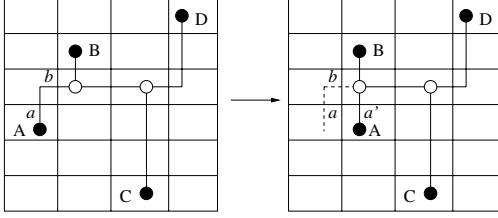


Fig. 9. Incremental Steiner-tree modification due to cell movement where solid dots represent pins and hollow dots represent Steiner points.

When a cell is moved to another block at a distance larger than one, we simply re-construct the Steiner trees for nets whose pins are moved. In this way, we always maintain a valid global routing and accurate congestion estimations.

Due to the nature of Steiner-tree construction used in initial global routing, routing congestion might not be completely eliminated with only the above incremental cell movement. Rip-up and re-route are usually required as a post-processing step. In our flow, we rip-up nets passing through the congested regions and re-route them using the *PathFinder* algorithm proposed in [15]. Because we have already allocated enough routing space, it is generally easy to fully eliminate all the congested areas. Finally, the global routing results are fed into a detailed router to complete the design.

VI. EXPERIMENTS AND DISCUSSIONS

A. Via-Configurable Routing Structures

We performed a set of experiments to examine particular routing structures and trade-offs. The M3/M4 crossbar on top of a BLE in our base architecture can accommodate 20×20 wire tracks, which along with the four M1/M2 inter-BLE wires results in a base unit of 24×24 wire pitches. To evaluate particular routing architectures without depending on white-space allocation algorithm, for each circuit we use a binary search to determine the minimum number of extra tracks

between blocks which need to be added to achieve routability. It is the same approach as used in most of the FPGA research based on VPR [16]. All the experimental results are based on 0.18 μm technology parameters. The resistance of a single via is 3Ω , and the wire capacitance is $180 fF/mm$.

We first investigated the capabilities of the three-base crossbar structures discussed in Section II. The experiments were run on a set of large MCNC benchmarks. We used only four metal layers. Table I lists the results. The column *#BLEs* gives the circuit sizes in terms of the number of packed BLEs. The columns under *Extra routing tracks* show the number of added tracks per row (column) required to successfully route a design using each crossbar structure. From the table, we can see that the crossover crossbar structure requires fewer routing resources than the jog-jumper (*J-jumper* in the table) and the direct-jumper (*D-jumper* in the table) crossbar structures. Consequently, due to the smaller total circuit area, the crossover crossbar structure also yields the best results in terms of the total wire-length and critical path delay. On average, a circuit implemented using the crossover crossbar structure has 29% and 10% less total wire-length compared to jog-jumper and direct-jumper crossbar structures, respectively. It is 21% faster than the jog-jumper implementation and 8% faster than the direct-jumper crossbar. The experiment demonstrates the larger throughput of the crossover crossbar structure.

We then examined the multi-layer routing structures with differing M5/M6 segment lengths. Both M3/M4 and M5/M6 use crossover crossbar structures. Fig. 10 shows the results for the circuit *alu4*. From the figure, we observe that the total wire length increases when the M5/M6 segment lengths increase. This is because, given the same routing area, there are fewer routing resources for longer M5/M6 segments. As a result some routes will use only parts of the M5/M6 segments, which increases the route wire-length. We observe that a circuit is not routable when the M5/M6 segment length is larger than 6, unless we add extra routing area. When the M5/M6 segment length increases, the number of used vias first decreases rapidly and then slowly increases. The decrease is predictable since routing a long net using shorter segments will take more segments than using the longer segments and thus will lead to more vias. But when the M5/M6 segments are too long, the scarcity of available segments causes more detours for a route and hence increases the number of vias. The critical timing increases with a longer M5/M6 segment length. We observed that the delay penalty caused by extra vias with length-1 segment is less than 3% of the critical path delay, which is much smaller than the delay increase (around 10~15%) due to larger total wire length when having longer M5/M6 segment lengths. This example shows that when routing resources are insufficient and M5/M6 segments are required to provide local routing, longer M5/M6 segments would cause a performance problem. For some cases (for example, *bigkey*) where M3/M4 segments are sufficient for local routing, longer M5/M6 segments can result in better performance due to reduced vias.

Next we examine the single-via-mask routing structure described in Section III (There are actually two via masks in a VCGA since the via1 mask is required for cell function implementation). Due to the fixed M5/M6 segment connections, a single-via-mask fabric is less flexible for routing. Table II shows the results. In this experiment, we chose length-4 M5/M6 segments and length-12 M4 accessing segments for a single-via-mask fabric. Column *area incr* gives the area increase (extra area required for a routable design) and *delay incr* gives the delay increase compared to the multi-via-mask implementation with length-1 M5/M6 segments. We observed that

TABLE I
TESTS ON MCNC BENCHMARKS FOR THREE CROSSBAR STRUCTURES WITH FOUR METAL LAYERS.

Circuit	#BLEs	Extra routing tracks			Total wire-length			Critical path delay (ns)		
		J-jumper	D-jumper	Crossover	J-jumper	D-jumper	Crossover	J-jumper	D-jumper	Crossover
alu4	36 × 35	13	8	5	2263485	1925622	1691041	6.5	5.9	5.4
apex2	40 × 39	13	8	4	2749598	2327341	1969921	7.4	6.5	5.8
bigkey	32 × 31	9	4	2	1419207	1155193	1065322	5.0	4.4	4.0
des	33 × 32	8	4	2	1225206	11058739	953461	7.1	6.6	6.3
elliptic	48 × 47	10	4	2	3352078	2727386	2474522	21.8	17.7	17.7
ex1010	45 × 42	18	10	7	4620563	3645988	3239009	10.8	9.1	8.3
frisc	46 × 44	8	3	1	2505372	1988937	1829493	19.3	17.3	16.3
misex3	35 × 34	13	8	5	2117885	1797122	1576889	6.7	5.9	5.5
pdc	37 × 36	15	9	6	2641750	2181373	1933786	12.2	10.7	9.9
seq	39 × 38	14	6	5	2701991	2149068	1959774	7.1	6.1	5.6

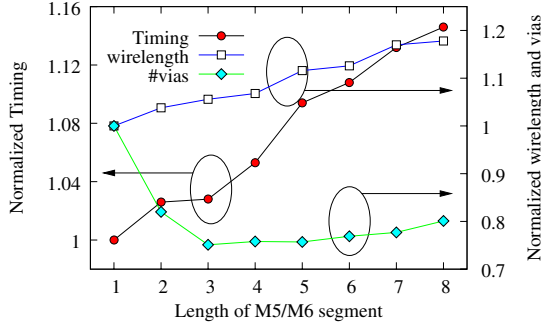


Fig. 10. The impact of different M5/M6 segment lengths for alu4.

TABLE II
SINGLE-VIA-MASK VS. MULTI-VIA-MASK FABRICS.

Circuit	area incr (%)	delay incr (%)	Used M5/M6 WL(%)	
			length-1	length-4
alu4	17.4	14.3	39.2	46.0
apex2	17.4	16.0	38.0	43.9
bigkey	0	2.5	44.2	49.0
des	0	5.3	47.2	51.1
elliptic	0	3.9	46.6	50.8
ex1010	46.0	25.4	42.9	51.4
frisc	0	2.0	39.7	45.0
misex3	17.4	15.2	39.5	46.6
pdc	26.6	16.7	40.2	46.1
seq	17.4	15.2	41.4	47.4

some designs with larger interconnect complexity are not routable without adding extra routing area, although all of them are routable using multi-via masks. Columns *Used M5/M6 WL* indicate the reason. Compared to a multi-via-mask with length-1 M5/M6 segments, a circuit implemented by a single-via-mask fabric uses on average only 42% of the M5/M6 wires that it would use in a multi-via-mask implementation (Column *length-1*). Even compared to a multi-via-mask fabric with length-4 M5/M6 segments (the same as in the single-via-mask fabric), the M5/M6 segments still account for on average 48% of those used in the multi-via-mask implementations (Column *length-4*). This is because in the single-via mask, a segment can connect only to two endpoints of an M5/M6 segment, but it can connect to some middle points of an M5/M6 segment in the multi-via-mask fabric. If extra routing area is needed, the delay will increase significantly.

Fixed wiring in a fabric can cause numerous dangling wires compared to the customized metal masks, particularly when a signal switches from one direction to another in a crossbar. By cutting dangling wires off to mimic the structured routing in [4], we have

observed that about 15~25% of delay improvement can be achieved compared to the implementation with fixed metal masks in the multi-via-mask fabric.

B. Routability-Driven Design Flow

We also experimented with our routability-driven design flow. In these experiments we assume that only four metal layers are used for the via-configurable fabric. With six metal layers there would be no routability issue for the available benchmark circuits which, compared to the modern designs, are relatively small.

We performed the experiments using two flows. The first flow was the same as the one used in the previous experiments, i.e., adding more inter-block tracks instead of allocating white-space to solve the routability problem. For fixed routing tracks, the second flow described in Section V added white-space to solve the routability problem. To compare our post-placement white space allocation scheme with the in-placement white-space allocation scheme, we also performed the experiments using Dragon from UCLA [7] which is the only publicly available placer capable of allocating white-space. Because Dragon cannot decide how much white-space is required, we determine, via a binary search method, the minimum white-space required to generate a congestion-free design.

We did experiments on the via-configurable gate array with crossover crossbar routing structure. Table III shows the results. The columns under *Area increase* give the percentage of area increase over the original densely packed design for various methods. Column *Adding tracks* lists the results of the first flow (varying routing tracks). Columns *WSA by Dragon* and *WSA by ours* give the results from the second flow (allocating white-space with fixed tracks) by Dragon and our algorithm. The columns labeled *diff* show the area differences between our method, the varying-track method, and Dragon. From the table, we observe that on average, our white-space allocation scheme requires 17% more area than the varying-track method, but 3% less than Dragon. Unlike in Dragon, we need several iterations to determine the amount of white-space. However, experimentally we observed that all the circuits were successfully routed in one shot using our white-space allocation and incremental cell movement algorithm. In some cases Dragon's results are worse than ours because of inaccurate congestion estimation. Fig. 11 shows as an example the circuit ex1010. From the figure, we can see that Dragon allocated too much white-space in the center of the placement area, but the congestion actually occurred close to the boundary. In contrast, our white-space allocation algorithm allocated white-space properly due to the accurate congestion estimation based on the actual global routing. Considering that the first flow is unrealistic for a fabric-based design, our white-space allocation provides a very tight design

VII. CONCLUSIONS

In this paper we have examined various aspects of via configurable routing structures. We have proposed a new crossover crossbar structure which achieves much higher throughput than the jog-jumper and the direct-jumper crossbar structures. As a result, the area and timing of a circuit implemented using the crossover crossbar structures can be greatly improved. We have shown how to construct a single-via-mask fabric. Compared to the multi-via-mask fabric, due to routing inflexibility, the single-via-mask fabric can incur up to 25% delay penalty and 46% area penalty. We further proposed a routability-driven design flow for fabric with fixed routing resources. The flow features a fast yet efficient white space allocation scheme and a congestion-aware incremental cell movement method for reducing congestion. Experimental results show that our flow can achieve results comparable to those of the variable-tracks-flow or in-placement white-space allocation, but it is much more practical and efficient in the context of fabric-based designs. Our flow could be used by a fabric vendor to quickly evaluate different architectures.

VIII. ACKNOWLEDGMENTS

This work was supported in part by NSF grant #CCR 0098069, and in part by the California MICRO program through IBM. The authors gratefully acknowledge equipment grant from Intel.

REFERENCES

- [1] C. Patel, A. Cozzie, H. Schmit, and L. Pileggi, "An architectural exploration of via patterned gate arrays," in *Proc. of ISPD*, 2003, pp. 184–189.
- [2] L. Pileggi, H. Schmit, A. J. Strojwas, P. Gopalakrishnan, V. Kheterpal, A. Koorapaty, C. Patel, V. Rovner, and K. Y. Tong, "Exploring regular fabrics to optimize the performance-cost trade-off," in *Proc. of DAC*, 2003, pp. 782–787.
- [3] B. Hu, H. Jiang, Q. Liu, and M. Marek-Sadowska, "Synthesis and placement flow for gain-based programmable regular fabrics," in *Proc. of ISPD*, 2003, pp. 197–203.
- [4] V. Kheterpal, A. J. Strojwas, and L. Pileggi, "Routing architecture exploration for regular fabrics," in *Proc. of DAC*, 2004, pp. 204–207.
- [5] A. E. Caldwell, A. B. Kahng, and I. L. Markov, "Can recursive bisection produce routable placements?" in *Proc. of DAC*, 2000, pp. 260–263.
- [6] U. Brenner and A. Rohe, "An effective congestion driven placement framework," *IEEE Trans. on CAD*, vol. 22, no. 4, pp. 387–394, April 2003.
- [7] X. Yang, B.-K. Choi, and M. Sarrafzadeh, "Routability-driven white space allocation for fixed-die standard-cell placement," *IEEE Trans. on CAD*, vol. 22, no. 4, pp. 410–419, April 2003.
- [8] C. Li, M. Xie, C.-K. Koh, J. Cong, and P. H. Madden, "Routability-driven placement and white space allocation," in *Proc. of ICCAD*, 2004, pp. 394–401.
- [9] B. Hu and M. Marek-Sadowska, "Congestion minimization during placement without estimation," in *Proc. of ICCAD*, 2002, pp. 739–745.
- [10] A. Singh, G. Parthasarathy, and M. Marek-Sadowska, "Interconnect resource-aware placement for hierarchical FPGAs," in *Proc. of ICCAD*, 2001, pp. 132–136.
- [11] A. Singh and M. Marek-Sadowska, "Efficient circuit clustering for area and power reduction in FPGAs," in *Proc. of FPGAs*, 2002, pp. 59–66.
- [12] Y. Ran and M. Marek-Sadowska, "The magic of a via-configurable regular fabric," in *Proc. of ICCD*, 2004, pp. 338–343.
- [13] B. Hu and M. Marek-Sadowska, "Wire length prediction based clustering and its application in placement," in *Proc. of DAC*, 2003, pp. 800–805.
- [14] D. P. Singh and S. D. Brown, "Incremental placement for layout-driven optimizations on FPGAs," in *Proc. of ICCAD*, 2002, pp. 752–759.
- [15] C. Ebeling, L. McMurchie, S. A. Hauck, and S. Burns, "Placement and routing tools for the Triptych FPGA," *IEEE Trans. on VLSI Systems*, vol. 3, no. 4, pp. 473–482, December 1995.
- [16] V. Betz and J. Rose, "VPR: A new packing, placement and routing tool for FPGA research," in *Proc. of FPGAs*, 1997, pp. 213–222.

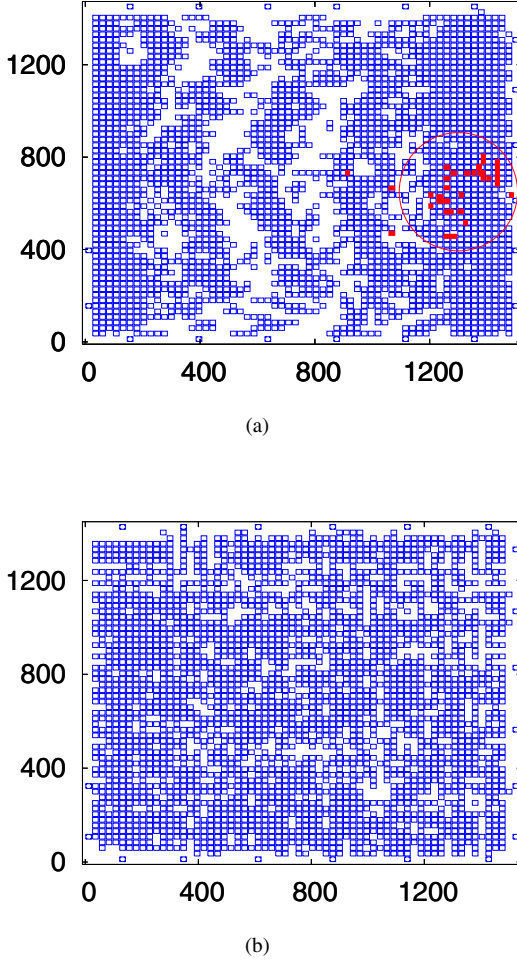


Fig. 11. Final placement after white-space allocation and incremental cell movement for ex1010 using different white-space allocation schemes: Dragon (a) and ours (b). There is no congestion using our white-space allocation algorithm, but some remaining congestion (shown as the bold lines in the circle (a)) occurs if using Dragon.

implementation in terms of final circuit area.

TABLE III

COMPARISON OF DIFFERENT CONGESTION ELIMINATION SCHEMES

Circuit	Area increase (%)			diff (%)	
	adding tracks	WSA by Dragon	WSA by ours	with tracks	with Dragon
alu4	46	64	61	+15	-3
apex2	36	77	60	+24	-17
bigkey	0	0	10	+10	+10
des	17	5	12	-5	+7
elliptic	17	32	47	+30	+15
ex1010	67	135	103	+36	-32
frisc	9	26	39	+30	+13
misex3	46	71	55	+9	-16
pdc	56	74	62	+6	-12
seq	46	64	59	+13	-5
AVG				+17	-3