Fishbone: A Block-Level Placement and Routing Scheme

Fan Mo
EECS, UC Berkeley
211 Cory Hall, UC Berkeley
Berkeley, CA94720
1-510-642-4641
fanmo@eecs.berkeley.edu

Robert K. Brayton
EECS, UC Berkeley
573 Cory Hall, UC Berkeley
Berkeley, CA94720
1-510-643-9801
brayton@eecs.berkeley.edu

ABSTRACT
A block-level placement and routing scheme called Fishbone is presented. The routing uses a two-layer spine topology. The pin locations are configurable and restricted to certain routing grids in order to ensure full routability and precise predictability. With this scheme, exact net topologies are determined by pin positions only; hence during block placement, net parameters such as wire length (and delay) can be derived directly. The construction of Fishbone nets is much faster than for Steiner trees; this enables the integration of block placement and routing; there is no separate routing stage.

Categories and Subject Descriptors
B.7.2 [**Integrated Circuits**]: Design Aids

General Terms
Algorithms, Design, Performance

Keywords
Placement, routing.

1. INTRODUCTION
As System-On-a-Chip (SOC) becomes more popular, block-level placement and routing will play an increasingly important role in design flows. This also facilitates hierarchical physical design, which narrows the gap between the capacity of existing algorithms and growing design size. In traditional design flow, placement and routing are separate stages. The goals of placement include minimizing area, total wire length, and maximizing routability. However, except for layout area, wire length and routability are based on estimations made during placement. The accuracy of this is a key factor in obtaining good final results; also bad estimation may lead to iterations of the two stages. The separation of these steps allows each step to focus on one or few targets trying for the best; such a sequential procedure saves run time. The cost is the possible non-convergence due to wrong early estimation. The problem of block-level routing is quite different from gate-level routing [4]. Usually the wiring density at the block level is not as high as the gate level. However routability problems do occur, even if the total routing density is low. One major cause is local congestion in the pin regions. The pins of the blocks are often placed on the block boundaries. When two blocks are placed side by side, the pins on the abutting edges are very close, causing routing congestion. Using more routing layers or leaving extra space between blocks are possible cures; however both are costly and it is hard to determine, especially at the placement stage, how much such extra resource is sufficient. Estimation of wire length during placement is usually based on certain net models. Various models are available [2,3,5,6], among which Minimum Steiner Tree (MST) and Half Perimeter (HP) models are the more popular. Since block-level routers commonly use the MST model, it is preferable to use the same model for estimation in placement. Unfortunately, its run-time cost is too high. Also due to congestion, nets planned individually using MST may require detours, causing deviation from the estimated wire length. Other estimation models, such as HP, have shorter run times, but are less accurate.

We propose using a fixed routing topology which is totally determined by the pin positions. Thus whenever a block placement is given, the routing of all the nets is determined precisely. Block placement and routing become an integrated task, and no estimation is necessary. The fixed net topology we use is called Fishbone; it is simply a spine with the output (source) pin of the net on a vertical trunk (e.g. on metal 4) and all input (sink) pins connected to the trunk by horizontal branches (e.g. on metal 3). It might seem naive to use a topology obviously inferior to the MST since previous research has shown that spine structures are not good net models for approximating Rectilinear Steiner Trees in terms of the "fidelity" of the estimation [7]. Spine topologies are only found in clock and power/ground routing. In general, Fishbone has longer wire lengths than MSTs. However, that is true only when the pin locations are fixed. A good block placement found with Fishbone routing should have located the pins such that the associated Fishbone nets have wire lengths comparable to those of the same placement but measured with the MST model. The goal of the Fishbone block-level placement and routing scheme is to find such placements. In other words, if the output of the Fishbone placer is post-processed by a MST router, the improvement will be small. An attractive point of the Fishbone is that the wire topology and thus the wire lengths are totally determined by the placement. Therefore a big difference between Fishbone-based placer and other block-level placers is that the Fishbone scheme can use precise wire lengths in the cost function of the placement algorithm. The routability of the Fishbone scheme can be tested quickly. This is enabled by requiring, in the Fishbone scheme, the pin locations on the blocks and the block locations on the layout to be constrained to certain grids. Pin locations on the blocks are made configurable by using a wide base pin (e.g. on metal 2) and using the concept of a configurable

virtual pin. Another advantage of Fishbone is that only two metal layers are necessary.

This paper is organized as follows. In Section 2, the Fishbone net topology and the routability conditions are described. In Section 3, the Fishbone block placement and routing flow is described. Section 4 gives experimental results, and Section 5 concludes and discusses future improvements and possible further advantages of this scheme.

2. THE FISHBONE ROUTING SCHEME

We assume that the layout uses metal layers up to m_B for internal routing of a block, and that the pins of the blocks and the I/O ports are on m_B. The layers m_{B+1} and m_{B+2} are used for global (Fishbone) routing. It is reasonable to assume that there is no obstruction in the global routing layers. Each global routing layer uses a uniform routing pitch and a fixed wiring direction (m_{B+1} horizontal and m_{B+2} vertical). The pitches of m_{B+1} and m_{B+2} may be different, but the same pitch is used for simplicity. The routing grids are given cyclic indices labeled 0,1,2,..,GR–1 in both the X and Y directions, where GR is the radix of the grid index. The notation $[x^{GX}, y^{GY}, z]$ defines the coordinate of a single point on the layout, and the superscripts denote the grid indices. $[x_1 \sim x_2, y, z]$ defines a horizontal stripe, and similarly $[x, y_1 \sim y_2, z]$ defines a vertical stripe. $[x, y, z_1 \sim z_2]$ defines a via between layers z_1 and z_2, if $z_2=z_1+1$, or stacked vias otherwise. A column channel (*or column for short*) is defined as $[x^{1\sim GR-1}, -\infty\sim\infty, m_{B+2}]$ and a row channel (*or row*) $[-\infty\sim\infty, y^{1\sim GR-1}, m_{B+1}]$.

Two routability conditions will be discussed for the Fishbone scheme; the basic condition and the dense condition. Then an interval packing algorithm is discussed, which assigns branches to horizontal metal stripes in the row channels and trunks to vertical metal stripes in the column channels. The flexibility of the branch and trunk assignments is made possible by the configurability of the "base/virtual" pin pairs. Finally the connections to the I/O-pads are discussed.

2.1 The Fishbone scheme

The Fishbone topology of a net is simply a vertical trunk connecting an output (source) pin of the net and horizontal branch(es) connecting this trunk to the input (sink) pin(s). The following definitions describe the base/virtual pin structure, a key element in the Fishbone scheme.

Definition 1 — A ***base input pin*** is a vertical stripe $[x^0, y^{1\sim GR-1}, m_B]$ with zero X-index and non-zero Y-index. A ***base output pin*** is a horizontal stripe $[x^{1\sim GR-1}, y^0, m_B]$ with zero Y-index and non-zero X-index. Base pins are part of the blocks and cannot be moved or modified.

Definition 2 — A ***virtual input pin***, corresponding to a base input pin defined above, is a point $[x^0, y^{G_{IY}}, m_{B+1}]$, connecting through a via down to the base input pin, where G_{IY} is the Y-index of the virtual input pin, ranging from 1 to GR–1. A ***virtual output pin*** is a point $[x^{G_{OX}}, y^0, m_{B+2}]$, connecting through two stacked vias down to the base output pin, where G_{OX} is the X-index of the virtual output pin, ranging from 1 to GR–1. The pair of base and virtual pin provides configurability for the connecting point to the net.

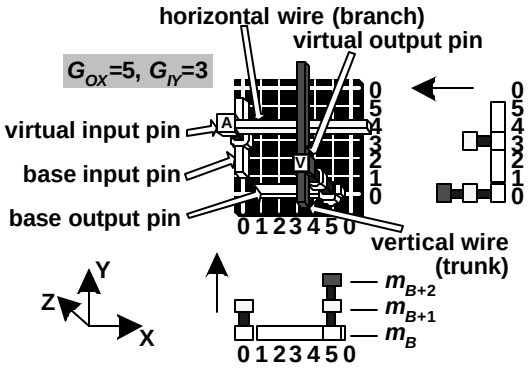

Figure 1. The base/virtual pin pair of the Fishbone scheme.

These definitions are illustrated in Figure 1. In the figure, GR=6, $G_{IY}(A)$=3 and $G_{OX}(V)$=5. The virtual input pin A, located at $[x^0, y^3, m_{B+1}]$, is made with a via connecting down to its base pin $[x^0, y^{1\sim5}, m_B]$. The virtual output pin V, located at $[x^5, y^0, m_{B+2}]$, is made with two stacked vias connecting down to its base pin $[x^{1\sim5}, y^0, m_B]$. The Fishbone routing deals with virtual pins only. The base-virtual pin mechanism provides flexibility in locating pin position in a small range.

The blocks are designed such that a commonly agreed GR is used (the selection of GR will be discussed later). All blocks used in the same design must use the same GR. For simplicity, the block size is assumed to be an integer number of column channels and row channels, given a GR. This extra requirement causes negligible increase in the size of the block since GR is small and the numbers of columns and rows are large. The Fishbone placer will always place the blocks such that their corners are at an X/Y-index of 0/0. Thus a location inside any block can be easily positioned by the global X/Y-coordinates and -indices. The following definition states how a Fishbone net is constructed with the output pin labeled 0 and the input pins labeled 1 to #*PI*. When routing is discussed, we simply use "pin" to refer to "virtual pin".

Definition 3 — Suppose that a net contains the output pin $[x_0^{G_{OX}(0)}, y_0^0, m_{B+2}]$ and input pins $[x_j^0, y_j^{G_{IY}(j)}, m_{B+1}]$. The ***vertical trunk*** of the net is a stripe $[x_0, \min(y_j)\sim\max(y_j), m_{B+2}]$. An input pin is connected to the trunk through a ***horizontal branch*** $[\min(x_0, x_j)\sim\max(x_0, x_j), y_j, m_{B+1}]$ and a via at $[x_0, y_j, m_{B+1}\sim m_{B+2}]$.

Since $G_{IY}\neq0$ for any virtual input pin, and all virtual output pins have their vias placed at a 0 Y-index by definition, a horizontal branch will never intersect a via of any virtual output pin. The vertical trunk is on layer m_{B+2}; thus it will never intersect any virtual input pin or horizontal branch on m_{B+1}.

Thus the virtual input pins and the horizontal branches will not interfere with virtual output pins, their vias to the base pins, and the vertical trunks. However, branches themselves may overlap; similarly for the trunks. Thus a problem arises on how to assign G_{OX} and G_{IY} for the virtual output pins and input pins such that no overlap occurs. The following is a sufficient but conservative condition for zero overlap, i.e. 100% Fishbone routability. We discuss this condition first, although it over-constrains the design significantly.

Condition 1: Basic Fishbone — A placement based on the basic Fishbone scheme is 100% Fishbone routable, if all the virtual output pins in the same column are assigned different

G_{OX} and all the virtual input pins in the same row are assigned different G_{IY}.

In other words, no two vertical trunks in the same column are placed on the same X-coordinate (X-index), and no two horizontal branches in the same row are placed on the same Y-coordinate (Y-index).

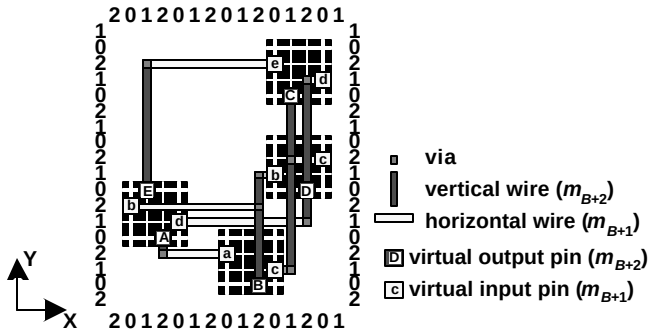

Figure 2. Basic Fishbone scheme (*GR*=3) connecting 4 blocks. Only virtual pins are shown; base pins are omitted.

This condition over-constrains the routing and possibly wastes routing resources. For instance, each m_{B+2} vertical track on the chip is either fully empty or only occupied by one vertical trunk. An example satisfying the basic Fishbone condition is illustrated in Figure 2. In the example, the input pins of a block are placed on the left and right sides of the block, and the output pins are placed on the top and bottom sides of a block (this is not a constraint; pins can be placed anywhere within the block). *Condition 1* implies that no two input pins of the same block can be placed in the same row, and no two output pins of the same block can be placed in the same column. Thus the numbers of input and output pins that can be accommodated in a block are equal to the numbers of rows and columns occupied by the height and width of the block. Although small *GR* allows more pins per block, it increases the possibility that the number of pins of different blocks appearing in the same row or column of the layout exceeds *GR*–1 (recall that to satisfy *Condition 1*, a row or column can maximally accommodate *GR*–1 pins).

A valid (but bad) placement exists even if *GR*=2, e.g. all the blocks, using G_{IY}=1 and G_{OX}=1 for any input and output pins respectively, are placed in a 45° line such that any column or row is occupied by only one block. In general, it gets harder to find a good placement in terms of area and wire length as *GR* becomes smaller, while using a large *GR* allows better placement, but either limits the number of pins of the blocks, or the block size needs to be expanded just to accommodate all its pins. Thus the choice of *GR* can significantly influence the quality of the result. Also note that *GR* should be defined early in the flow so that it can be used to determine block size and base pin locations and size. A reasonable *GR* can be chosen as $GR=\lceil \#B^{1/2} \rceil + r$, where #*B* is the total number of the blocks to be placed with the Fishbone scheme and *r* is a small integer e.g. 2 or 3.

An observation is that there is no vertical trunk at $[x^0, \text{any}\sim\text{any}, m_{B+2}]$, which means G_{OX} of a virtual output pin can safely be 0. In addition, X and Y directions (layer m_{B+1} and m_{B+2} respectively) can use different *GR*'s. However, for simplicity and symmetry, the requirement of $G_{IY}, G_{OX}=1\sim GR$ is used. The reason that virtual output pins and trunks are put on a higher metal layer is that the number of output pins in a design is not greater than the number of input pins. Such a layer assignment will use less vias than the opposite assignment. Also, since m_{B+2} may have a larger routing pitch than m_{B+1}, and there are more horizontal branches than vertical trunks in a design, such an assignment is helpful when a square layout is preferred.

The constraint that no two vertical trunks are placed at the same X-coordinate and no two horizontal branches are placed at the same Y-coordinate can be relaxed; this results in the so-called **dense** Fishbone scheme:

Condition 2: Dense Fishbone — A placement based on the dense Fishbone scheme is 100% routable if all the vertical trunks with the same X-coordinate (same G_{OX} in the same column channel) are non-overlapping, and all the horizontal branches with the same Y-coordinate (same G_{IY} in the same row channel) are non-overlapping.

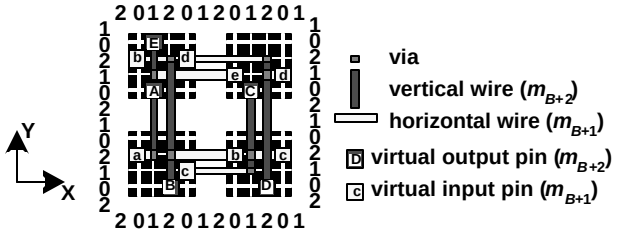

Figure 3. Dense Fishbone scheme (*GR*=3) connecting 4 blocks. Only virtual pins are shown; base pins are omitted.

The use of the dense scheme increases the routing resource utilization, while maintaining the Fishbone topology. Also, more pins can be accommodated in each block by using a smaller *GR*. The example in Figure 2 has been re-drawn in Figure 3 using the dense Fishbone scheme. The dense Fishbone scheme is the one adopted in our algorithm on which we report experimental results.

2.2 Interval packing algorithm for the arrangement of branches and trunks

The arrangement of the branches and trunks determines the Y/X-coordinates of the virtual input/output pins. Since a virtual input/output pin only has the freedom in choosing its Y/X-coordinate within a row/column, the problem is equivalent to determining G_{IY}/G_{OX}. With the *basic* Fishbone condition, the arrangement is trivial; whether the condition can be satisfied is just a matter of counting the number of input/output pins in the same row/column.

For the *dense* Fishbone condition, the sequence is to determine G_{IY} for input pins first and then determine G_{OX} for the output pins. First consider the G_{IY} determination for the virtual input pins. Given a placement of the blocks, the input pins are already assigned to row channels. The branch of input pin P_i is a horizontal stripe $[x_L(P_i)\sim x_R(P_i), y^{G_{IY}(Pi)}, m_{B+1}]$, in which $G_{IY}(P_i)$ is the Y-index of the virtual input pin, $x_L(P_i)$ and $x_R(P_i)$ are X-coordinates of the left and right terminals of the branch. Either $x_L(P_i)$ or $x_R(P_i)$ is the X-coordinate of the trunk of the net, while the other is the X-coordinate of the input pin (0 X-index). Since the G_{OX} determination for the virtual output pins is done after the determination of G_{IY} of the virtual input pins, the exact G_{OX} or the X-coordinate of the trunk is not known yet. The range of the X-coordinate of the trunk is $x^1(P_0)\sim x^{GR-1}(P_0)$. Hence, if the left

terminal of the branch connects the trunk, let $x_L(P_i)=x^1(P_0)$; and if the right terminal of the branch connects the trunk, let $x_R(P_i)=x^{GR-1}(P_0)$. An interval packing algorithm is employed to arrange all branches in the same row. The algorithm gives the minimum number of tracks needed to accommodate all the intervals [9].

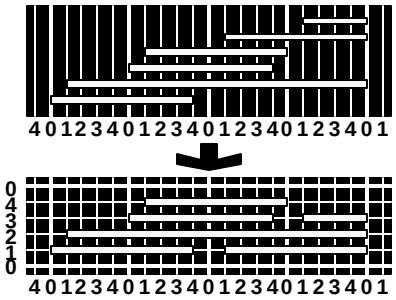

Figure 4. An interval packing example.

The algorithm is explained with the example shown in Figure 4. The interval packing approach was used in channel routing (also called left edge algorithm). First, order all branches in ascending order of their left X-coordinates. The algorithm greedily fills up a track with un-used branches in the ordered sequence. A new track is introduced when the current track is filled up. Here $GR-1$ tracks are available. Excessive tracks are considered violations.

After the branches are arranged and the Y-coordinates of the virtual input pins are determined, the placement of trunks or the determination of X-coordinates of the virtual output pins begins, also using interval packing; however since the branches are already determined, a trunk has its Y range known exactly.

The above algorithm returns an arrangement of branches and trunks for the given block placement and can report the number of violations. The goal of Fishbone block placement, besides seeking a minimal combination of area and wire length, is to find a solution with no violations.

2.3 I/O-pins

A remaining problem is how to handle the pins on the I/O ports of the chip. Assume that an I/O port carries exactly one pin, called an I/O-pin. Also assume that the I/O ports are evenly spread along the layout boundary. An obvious choice is to place all input ports (carrying output I/O-pins) on the top and bottom boundaries and all output ports (carrying input I/O-pins) on the left and right boundaries, as if the entire layout itself were a block. Then the Fishbone scheme can be applied directly to I/O-pins. However this simple approach has an adverse impact on the wire length. To allow I/O ports to be placed on any side of the boundary, we need to consider two special cases. One is the input I/O-pin being on the top or bottom side, and the other is the output I/O-pin being on the left or right side.

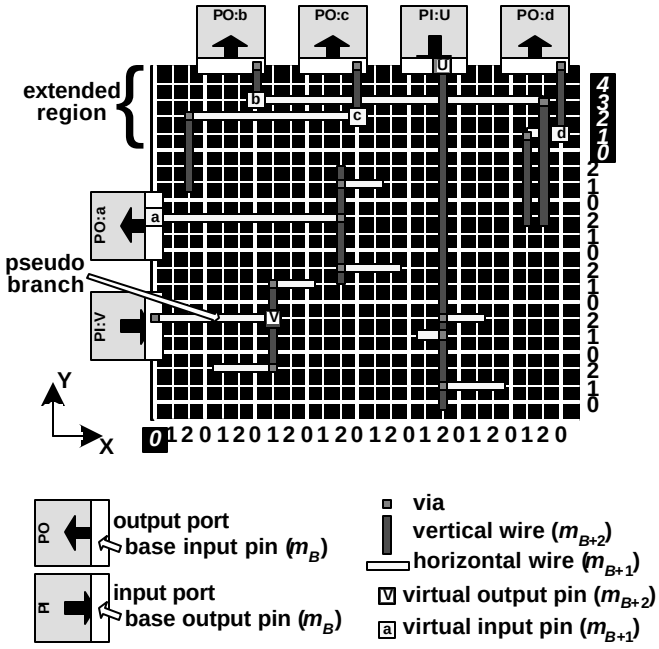

Figure 5. The I/O-pins and their connections.

The connections to the I/O-pins are illustrated in Figure 5. Only the top and left sides are shown; the bottom and right sides being symmetrical. All the base I/O-pins are still on m_B, covering at least the height/width of a row/column. The input I/O-pin on the left side (PO:a) and the output I/O-pin on the top side (PI:U) work as usual. The output pin on the left side (PI:V) connects to the vertical m_{B+2} trunk through a horizontal m_{B+1} wire, as if it is an input pin on the left side. This horizontal wire is called a pseudo-branch. The column of the virtual output pin or the trunk is chosen as the column into which the average X-coordinate of the input pins on this net falls. The exact X-coordinate of the trunk will be determined by the result of the interval packing of that column. The input I/O-pins on the top side (PO:b, PO:c or PO:d) use a so-called extended region to first create its virtual input pin, and then connect to its trunk through a horizontal m_{B+1} branch. The extended region does not have a GR in the Y direction. It extends until all the virtual input pins and their horizontal branches are accommodated via interval packing.

An extended region can be used for the output I/O-pins on the left side, but then the virtual output I/O-pin and its vertical trunk will appear in this extended region. Thus the input pin(s) on the corresponding net may all need to stretch their branches to the far left side of the layout. If there are multiple input pins, this is obviously bad for wire length, unless all blocks carrying these input pins happen to be placed on the left side of the layout.

3. DESIGN FLOW

Our block placement uses a simulated annealing framework with sequence pairs as the layout representation [1]. At each annealing step, a layout modification, such as block swapping or orientation change, is made. Note that a block can only have four orientations: normal, X-flipping, Y-flipping and XY-flipping, but 90 degree rotations are not allowed. Then the interval packing algorithm is run for each row and column to arrange branches and trunks, and the wire length is computed. The cost function is:

Table 2. Experimental results

	area (mm^2)			average wire length (mm) <1>									#routing violations				placement time			routing time			
				FB				HP			RST		FB	FB						FB	FB		
example	FB	HP	RST	pl	p-RST	ro-b	ro-v	pl	p-RST	ro	pl	ro	ro-b	ro-v	HP	RST	FB	HP	RST	ro-b	ro-v	HP	RST
ami33	5.09	5.32	5.10	0.70	0.76	0.76	0.78	0.89	0.90	0.89	0.85	0.83	2	0	6	7	1m36	0m30	19m	1m14	0m02	1m06	1m15
ami49	59.4	57.2	58.9	3.22	3.15	3.15	3.18	3.16	3.24	3.24	3.09	3.12	0	0	0	0	5m07	1m05	54m	0m22	0m02	0m22	0m24
playout	349	296	298	9.14	9.42	9.13	9.39	8.92	9.07	9.13	8.91	8.97	4	0	4	2	20m	3m15	1h39	1m55	0m13	1m55	2n33
ibm100	73.6	65.1	67.2	7.51	6.94	6.97	6.94	6.40	6.76	6.74	6.91	6.90	6	0	7	6	53m	8m	4h24	2m40	1m32	2m45	2m52
ibm101	102	83.9	81.0	8.58	7.92	7.93	7.91	7.28	7.68	7.64	7.90	7.87	5	0	11	7	72m	9m	6h02	3m40	2m26	4m27	4m37
ibm102	127	104	103	9.21	8.43	8.46	8.43	8.11	8.57	8.55	8.87	8.82	3	0	13	13	1h34	10m	7h07	6m58	4m18	6m06	7m15
ibm103	151	128	131	10.9	9.92	9.93	9.89	9.01	9.50	9.48	9.57	9.53	15	0	16	16	1h56	12m	8h09	9m13	6m26	9m07	10m
ibm104	187	152	156	11.7	10.5	10.5	10.4	10.6	11.2	11.2	10.2	10.2	23	0	20	23	2h17	13m	9h42	13m	9m13	15m	13m
compare	1.14	1.00	**1**	1.05	1.00	1.00	1.00	0.98	1.02	1.01	**1**	1.00	0.77	0	1.06	**1**	0.19	0.03	**1**	0.91	0.39	0.92	**1**

pl: placed. p-RST: post-placement RST estimation.
ro: routed. ro-b: routed with Fishbone placemen but with base pins only. ro-v: routed with Fishbone placement and virtual pins.
<1> The p-RST wire length estimates may be longer than others. This is possible, because RST only uses the center points of the base pins to measure wire length.

$$cost = w_A \cdot A + w_W \cdot W + w_G \# V_G$$

in which A is the chip area, W is the average wire length, and $\#V_G$ is the number of violations. Parameters w_A, w_W and w_G are the corresponding weights. The goal of the algorithm is to seek a Fishbone placement with $\#V_G$=0 while the weighted sum of area and wire length is minimized. The design flow is summarized in the following pseudo code.

```
1   randomly generate an initial layout
2   for each scheduled annealing step {
3       randomly do one of the following:
4           (1) swap a pair in one of the sequence pairs
5           (2) swap a pair of I/Os
6           (3) flip one of the blocks
7       update layout
8       interval packing and counting violation number
9       evaluate wire length
10      evaluate cost
11      accept or reject
12  }
```

Interval packing is very fast, because it usually handles only a very limited number of branches/trunks in a row/column. Therefore it can be run in the inner loop of the simulated annealing. The routing congestion is fully defined by the Fishbone routability condition and is reflected by the number of violations. In fact, wire delays can appear in the cost function as well; the Fishbone scheme enables explicit delay representations. The formulation of delay is quite flexible: average wire delay, maximum wire delay or delay slack. The design flow terminates when the placement is finished; *note* no separate routing is necessary, because the wires have been finalized using the Fishbone topology.

4. EXPERIMENTAL RESULTS

Three examples from the MCNC benchmark suite and a set of randomly generated examples were tested. Large fan-out nets are removed, because such nets are usually clocks or power/ground which will be routed on reserved metal layers by specific routers. The random examples are constructed as follows. The eighteen ISPD benchmark examples, ibm01 to ibm18, form a pool of blocks. Then blocks are randomly selected (with replacement) from the pool and their interconnections randomly generated. A routing pitch of 1-micron is assumed for the random examples on layers m_{B+1} and m_{B+2}, and a gate size of 16-micron^2 is used to compute the block size based on the gate count information given in the ISPD benchmark suite. The grid radix GR is set to be $\lceil \#B^{1/2} \rceil + 1$, where $\#B$ is the total number of the blocks to be placed in a rectangular layout. The block sizes in the examples are rounded up so that they contain an integer multiple of GR. For the MCNC examples, the pins defined in the benchmark set are shifted to the nearest legal Fishbone positions; the pins of the blocks in the random examples are randomly placed on the block boundaries at legal Fishbone positions. All the base pins are on layer m_B. Such block sizing and pin placement should not affect other placement algorithms in comparison with the Fishbone placement algorithm. The characteristics of all examples are listed in Table 1.

Table 1. The characteristics of the testing examples

example	#block	#I/O	#net	#pin	radix
ami33	33	42	117	522	7
ami49	49	22	407	953	8
playout	62	192	1609	4656	9
ibm100	30	200	3327	9983	7
ibm101	40	300	4340	13021	8
ibm102	50	400	5402	16208	9
ibm103	60	500	6481	19443	9
ibm104	70	600	7621	22864	10

The Fishbone placer is compared with two placement algorithms, one based on Half-Perimeter (HP) estimation and one based on Rectilinear-Steiner-Tree (RST) estimation. The RST placer uses an iterative heuristic [8]. All the placement algorithms use the same annealing schedule and sequence-pair formulation [1]; the only difference is just the different net models adopted. In the cost function, w_A=0.5, w_W=0.5, while w_G=0.03 for Fishbone and w_G=0.0 for RST and HP. Since the base pins are stripes instead of points, the center points of the base pins are used to estimate the wire length in RST and HP placement. After placement, the Fishbone layout is finished because the routing is already done. However, the other two need a real router to do the routing. The Cadence Warp Router is employed for this.

We made the following additional experiments. After the Fishbone and HP placements, we ran RST estimation for the layouts to compare the placements in terms of wire length but estimated with RST. In addition, we ran the Warp Router twice on the Fishbone placement, one with only base pins specified (ro-b), and the other with virtual pins determined by Fishbone specified (ro-v). This was done to see if the Fishbone placement is good for an unrelated router. Presumably these runs should not cause any

routability problems, because 100% routable solutions exist if Fishbone placement reports no violations. However, ro-b runs did cause problems. We discuss this below. All programs were run on a Sun Blade 1000 workstation.

The results are given in Table 2. All Fishbone placement tasks terminated with no violations. On average, the Fishbone scheme resulted in a 14% area overhead and a 5% increase in wire length. This can be considered as a price paid for 100% routability and predictability known during placement. The run time of the Fishbone scheme is the time taken only by the simulated annealing phase, which is on average 80% less than for RST placement; the RST and HP placements need extra time for routing. HP is comparable to RST in terms of the area and wire length, but the run time of HP is a small fraction of the RST time. The final routings for the RST and HP placements are not all violation-free, even though in all the examples, the Warp Router reports no over-capacity G-Cells (a coarse routing grid used in the global routing to estimate possible wiring congestion). All routing violations occur in the smaller pin regions. This means that such routability problems are hard to predict even in the early routing stages. Leaving more space between blocks and/or utilizing more routing layers may help, but it is unclear if, or how much, such extra resources should be used. It is not surprising that the Fishbone placement with virtual pins specified (FB ro-v) is 100% routable using the Warp Router. Also it runs much faster (because there are no violations to be repaired). The same Fishbone placement, but with only base pins given (FB ro-b), causes routing problems, although not as severe as those encountered in the HP and RST placements. This shows again that the routing in the pin regions is one of the major problems with block-level routing, especially when the pins cover several grids as do the base pins. The router may not be smart enough to find a good point (virtual pin) to build the connection to the net. We observed that sometimes the router builds two or more connection points to a base pin; such flexibility is not used or required in the Fishbone scheme. However, this does not seem to help in solving the routability problem in the non-Fishbone designs.

Comparing the wire length of the Fishbone (FB pl) with that of the post-placement RST estimation (FB p-RST), we find that the difference in wire length between Fishbone and RST for the same placement is 5%. This demonstrates that although in general RST routing is better than Fishbone routing, a good Fishbone placement can greatly reduce the gap.

5. CONCLUSION

In this paper the Fishbone block-level placement and routing scheme is presented. The fixed and simple net topology enables fast wire construction and precise wire length evaluation, which makes possible easy integration of placement and routing. The configurability of pin location via base/virtual pin pairs offers great flexibility in placing wires and making connections. Thus routability is very often achievable even with only two routing layers. The cost is some loss of optimality in area and wire length. For instance, block orientations are reduced from eight kinds to four (no 90^o rotations), which might cause the elimination of some better placements. Although Fishbone wire length is usually inferior to that constructed with RST or HP, if the placement made with RST or HP is eventually un-routable, as the experimental results show, then their superiority in area and wire length is diminished.

The following are some limitations of the Fishbone scheme. 1) It cannot handle obstructions in the Fishbone routing layers. 2) No 90^o rotations of the modules are allowed. This may account partly for the increased area. 3) The Fishbone topology is restricted to a vertical trunk. If the choice of a horizontal trunk were offered as well, the wire length could be decreased. 4) It needs a pre-defined radix. The radix should be known when the blocks are designed. Using the radix formula in this paper, this becomes a question of how many blocks are to be integrated on a chip. An approximate value is usually sufficient. However, if it is really hard to predict the number of the blocks, the block designer may offer several versions of the block layout, with different base pin positions according to different radixes. This is not hard, because the core layout of the block needs no modification.

Points 2 and 3 above can be addressed by making all base pins L-shaped. This allows for rotation and for choosing whether a net has a vertical or horizontal trunk. However, this choice makes the estimation of wiring violations a bit more complicated.

Extension of the Fishbone scheme to a timing-driven version is straightforward because the wire delays can be derived directly from the Fishbone nets. Note also, that in any row channel or column channel, an arbitrary permutation of the ordering of the indices is allowed. This could be used to alleviate cross-talk problems during a post-layout processing step. In addition, any output pins from the same block are separated by at least one (vertical) empty track (the 0-index track). Thus output tracks that are immediate neighbors must come from sources on different blocks, implying that their timing windows would rarely overlap, thus also alleviating cross-talk problems.

6. ACKNOWLEDGEMENTS

This work was supported by the GSRC (grant from MARCO/DAPPA 98DT-660, MDA972-99-1-0001). The authors are grateful to members of the Constructive Fabrics Group of the GSRC for many enlightening discussions.

7. REFERENCES

[1] H. Murata, K. Fujiyoshi, S. Nakatake and Y. Kajitani, "VLSI Module Placement Based on Rectangle-Packing by the Sequence-Pair", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 15, no. 12, Dec 1996, pages 1518-1524.

[2] A.B. Kahng, S. Mantik and D. Stroobandt, "Toward Accurate Models of Achievable Routing", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol 20, no. 5, May 2001, pages 648-659.

[3] S. Bodapati and F.N. Najm, "Prelayout Estimation of Individual Wire Lengths", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol 9, no. 6, Dec 2001, pages 943-958.

[4] H. Yoshimura, Y. Asahi and F. Matsuoka, "Scaling Scenario of Multi-level Interconnects for Future CMOS LSI", Digest of Technical Papers, Symposium on VLSI Technology, 2001, pages 143-144.

[5] F. Mo, A. Tabbara and R.K. Brayton, "A Force-Directed Macro-Cell Placer", International Conference on Computer Aided Design, Nov 2000, pages 177-180.

[6] "VLSI Physical Design Automation, Theory and Practice", edited by S.M. Sait and H. Youssef, World Scientific, 1999.

[7] J.L.Ganley, "Accuracy and Fidelity of Fast Net Length Estimates", ACM VLSI Integration, the VLSI Journal, vol.23, no.2, Nov, 1997, pages 151-155.

[8] A.B.Kahng and G.Robins, "A New Class of Iterative Steiner Tree Heuristics with Good Performance", IEEE Transactions on Computer-Aided-Design of Integrated-Circuits and Systems", vol.11, no.7, Jul 1992, pages 893-902.

[9] N.A.Sherwani, "Algorithms for Physical Design Automation", Kluwer Academic, 1993.