An Integrated Standard-Cell Physical Design Algorithm

Abstract — An integrated standard-cell physical design algorithm (ISCPD) is presented. A spine net topology is adopted, enabling quick construction of the placement and routing of a standard-cell design with guaranteed routability. Experiments show that ISCPD is comparable to commercial placement and routing tools in terms of area and wire length but several times faster, making it very suitable for physical synthesis flows, in which quick construction of standard-cell modules is a critical step at the floorplanning stage.

1. Introduction

A new trend in modern VLSI design, called physical synthesis, incorporates physical information in the logic synthesis by means of the floorplanning. The floorplanning deals with hard and soft modules. Each soft module contains a cell netlist that is pre-synthesized. The floorplanning requires that the cell size be given either through estimation or physical design. The shape (or aspect ratio) of the module and the pin locations are usually less important at this stage. The internal wiring of the modules uses metal layers, up to metal 3. Upper metal layers are reserved for inter-module connections, which are not considered here.

A common approach for estimating the area of a soft module is to divide the total cell area produced by the pre-synthesis by a given "utilization" number. Leaving alone all other sources of inaccuracy like the pre-synthesis, an inaccurate utilization number may affect final routability and timing closure. Unfortunately this parameter is quite empirical. On the other hand, doing physical design for the module is sound, but run time becomes a barrier. This is another critical issue, because floorplanning and pre-synthesis may be (incrementally) called many times in the flow. Calling a real standard-cell (SC) placer and router to get the module area is not affordable. Even then, most existing SC physical design algorithms require row utilization as an input, and even if used, several iterations might be needed to get a module size small enough but still fully routable. Ideally we want a quick estimate of the module area and make sure the area is small enough but within such area routability can be guaranteed.

We try to develop an algorithm that is tailored for this specific application:

(1) The size of the netlist is limited up to about 20K gates.
(2) Three metal layers can be used for routing.
(3) Layout aspect ratio is not fixed, but should be reasonable.
(4) No clock net or large fanout nets are present.

The algorithm need not be a general purpose SC physical design algorithm, but should fulfill:

(1) Area and wire length comparable to mainstream SC tools.
(2) No area utilization number should be required.
(3) Run time must be quick.

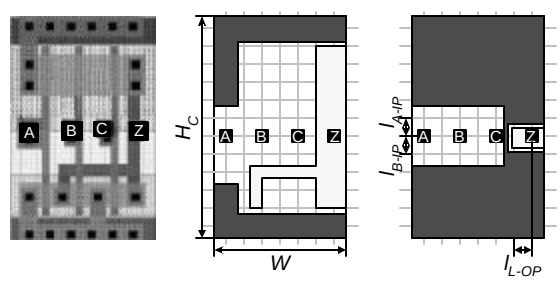
The major problem with conventional SC physical design tools is the separation of placement and routing. Utilization is a sort of routability estimation made in the placement. Various approaches have attempted to close this gap. One is to try to make the estimation in the placement more accurate [1-4, 14]. Another is to allow the router to locally modify the placement and/or reroute to fix the routing violations [5,6]. All such methods still separate the placement and routing.

The ISCPD algorithm, presented in this paper, takes a different approach. It uses a simple and predictable net topology which enables the integration of placement and routing. A two-layer (the two upper layers, metal 2 and 3) spine net topology is employed. The bottom layer (metal 1) is used to make connections between the SC cell pins and the spine nets. The spine of a net is a horizontal wire segment on metal 3 connecting the output pin driving the net. Input pins of the same net are connected to the spine using vertical segments on metal 2 called "ribs". Given sequences of SC cells in the rows, the cell placement and arrangement of the ribs of the nets are done from the left side, followed by the arrangement of the spines of the nets. The simple spine net topology makes the construction fast. The sequences of the cells in the rows uniquely determine the layout. Initial sequences are generated through recursive bi-partitioning. This partitioning benefits from the fixed spine net topology, making the term "cut" more meaningful. The algorithm is fast, offering a means of quick construction of a 100% valid (routable) standard-cell layout with acceptable quality.

The paper is organized as follows. Section 2 gives the preliminaries of the cell model and spine net topology. The ISCPD algorithm is described in Section 3. Experimental results are given in Section 4 and Section 5 concludes.

2. The standard-cell model and the spine-based routing

A cell is a rectangle with size of $W(c)$ by H_C. All the cells have the same height H_C. The pins of a cell are all on M1; some other parts of M1 are occupied by the internal connections of the cell. A pin can be reached either from its surrounding region on M1, if not blocked by existing M1 geometries, or from M2 by a via.



(a) layout (b) abstract view (c) adjusted view

Figure 1: The typical layout of a cell and its views.

Figure 1(a) is a typical layout of a NOR3 gate. It is abstracted in Figure 1(b), in which hatched regions represent the M1 areas already occupied by power, ground and other internal connections, and the dotted region is the M1 area occupied by the output pin Z. The white space is free for M1 routing. The ISCPD algorithm only uses part of the free M1 space, so the layout is further adjusted to Figure 1(c), in which the M1 blockages are expanded, and the output pin and input pins are isolated. It is also assumed that all the input pins have the same Y-grid, and parameters l_{B-IP} and l_{A-IP} denote the number of free M1 horizontal tracks below and above the pins. For the output pin, l_{L-OP} denotes the number of vertical M1 tracks to its left. The majority of the cells in a real library can be translated to their adjusted views without any modifications of their layouts. A few need slight modification with almost no area and

delay overhead. Only single-output cells are handled, but extension to cells with multiple output pins is easy.

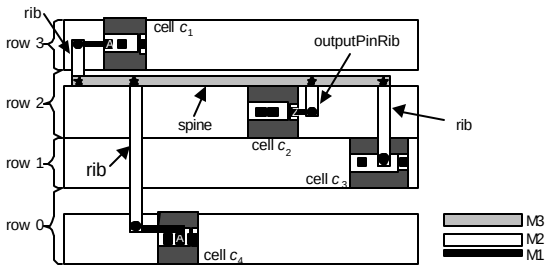


Figure 2: The spine topology.

The basic idea of the algorithm is to use a simple and predictable spine topology, which involves a horizontal spine on M3 in the row of the driver cell and vertical ribs on M2 connecting the load cells and the spine. An example is illustrated in Figure 2, in which cell c2 is the driver and all other cells are loads. Short M1 segments connect the input pins (on M1) of the loading cells to the ribs. A short vertical M2 segment called the "outputPinRib" leads the output pin (one M1) to the spine. The following properties form the basis of a quick way of constructing a 100% legal layout, owing to the simple and fixed net topology:

Property 1: One important property is that whenever the sequences of the cells in the rows are given, we know which rows the ribs vertically span.

Property 2: When all the ribs are fixed, we know how wide the spines horizontally span.

In the example, the spine of the net is always in the range of row2 (note that the gap, if inserted above a row, is viewed as part of that row). Therefore the rib connecting cell c4 spans the vertical range of row0~row2.

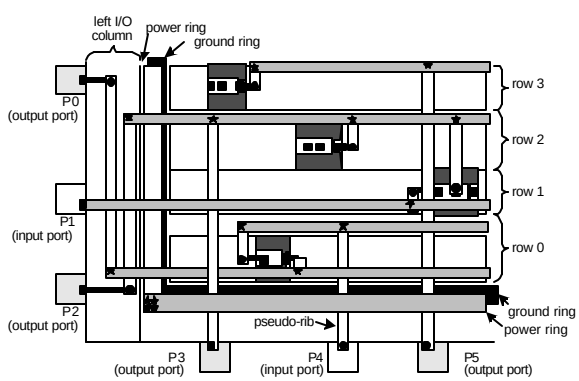


Figure 3: The I/O Connections.

I/O ports can be regarded as small cells placed along the boundary of the layout. An example of the left and bottom sides of the layout is illustrated in Figure 3. The cases in which an input port (with an output pin thus a spine, P1 in the figure) is on the left and an output port (with an input pin thus a rib, P3 or P5 in the figure) is on the bottom can be handled as usual. One special case is that an output port (with an input pin, P0 or P2 for example) is on the left. A so-called "left I/O column" is built to accommodate the ribs for such pins. Of course the spines of the related nets must stretch to the left I/O column to fulfill the connections. Another special case is that an input port (with an output pin, P4 for example) is on the bottom. It uses a "pseudo-rib" to route to its spine. The row where the spine stays is determined by the average of the rows of all the pins on the net, assuming this I/O-output-pin is in row −1. The I/O connections of the right and top sides are similar. Also shown in the figure are the power and ground rings.

3. The algorithm

3.1. Overview

A run of the ISCPD algorithm involves the following steps:

(1) Determine the number of the rows.

(2) Bi-partitioning to generate the initial sequences.

(3) Produce the layout from the sequences. Procedure AllCell() starts with an empty layout. It picks up a cell from one of the sequences (or row) at a time and push it towards left through procedure OneCell(*cell*) and have ribs of the cells arranged. After all cells are placed, all the ribs are arranged such that the horizontal ranges the spines span become known. Then the spines are routed row by row by using a simple left edge algorithm.

3.2. The determination of the row number

Given expected aspect ratio of the layout and utilization, denoted by a_0 and u_0 respectively, an estimate of the number of rows is:

$$N_R = \frac{\sqrt{\frac{a_0 A_C}{u_0}} - 2W_{PG}}{H_C},$$

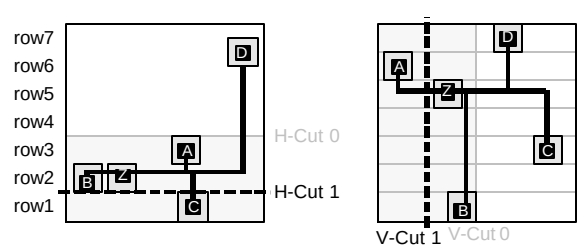
in which, W_{PG} is the width of the power/ground ring, H_C is the height of the cells and A_C is the total cell area. The selection of u_0 only affects the real aspect ratio. ISCPD can always produce a valid layout no matter u_0 is used, but a carefully chosen u_0 prevents creating layout with ill shape. The following empirical equation is used to estimate u_0:

$$u_0 = 1 - 3\times 10^{-5} \cdot N_C,$$

in which, N_C is the number of cells in the netlist. This basically predicts a linear drop of the utilization as cell number increases.

3.3. The generation of the initial sequences — partitioning

The initial sequences of the cells in the rows are generated through successive bi-partitioning. The goal is to minimize the number of wires crossing the cut lines. The process is to keep doing horizontal partitioning until a single row is reached and then doing vertical partitioning until a partition contains 5 or less cells (in a row).



(a) horizontal cut **(b) vertical cut**

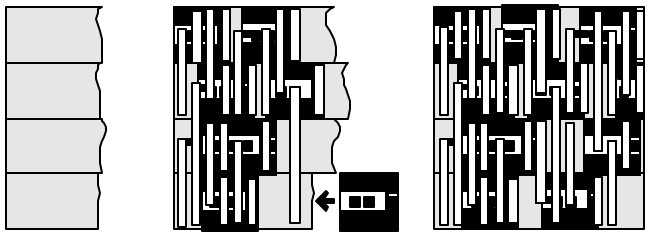
Figure 4: The bi-partitioning.

Fiduccia-Mattheyses (FM) partitioning algorithm is adopted [13] with modified definition of "cut". In vertical partitioning, a spine crossing the cutting line counts one cut. In horizontal partitioning, a rib crossing the cutting line counts one cut. External pins with connections to pins in the current partitioning region can

be considered if their relative positions to the current cutting line are known. In each cut, the pin distribution of a net across the cut is known. This feature is owing to the spine model. Since the positions of the cells (carrying the pins of the net under consideration) with reference to the cut line are known, we can tell how the cut number of this net will be affected if a cell is moved from one side of the cut line to the other. An example is shown in Figure 4. A horizontal cut example is shown in Figure 4(a), in which the net under consideration contains output pin Z and input pins A to D. Before the current cut, some previous horizontal cuts have refined the Y-positions of some cells. This cut takes place in row-1 and row-2&3, the dotted region in the figure; so only cells carrying pin A, B, C and Z can switch between these two partitions. Suppose the cell carrying Z moves to row-1, the spine also moves to that partition. The previous cut by the connection to pin C disappears, while the connections to A, B and D generate new cuts. So the total effect is an increase of 2 cuts. This shows the big difference between ISCPD partitioning and the traditional FM. In FM, a net can only have zero cut or one cut, independent of net topology. However in the partitioning of ISCPD, the cut number has more realistic meaning, owning to the known net topology. If cell carrying C moves to row-2&3, the cut of this net becomes zero. The move of the cells must obey some partition size balance; otherwise all cells may move to the same partition, which results in zero total cut. Vertical cut has similar behavior, as shown in Figure 4(b). I/O ports act as "cells" carrying external pins. The cells in each final partition are randomly ordered. The initial sequences of cells in the rows are thus generated.

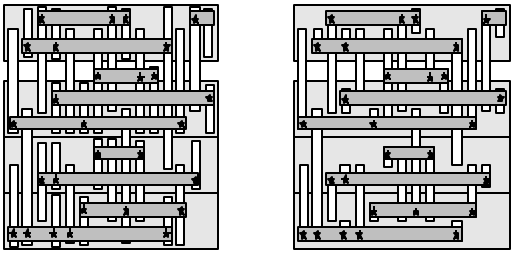
3.4. AllCells()

After the partitioning, the row a cell is in has been determined. Also determined are the sequences of the cells in the rows. Staring from the left edge of an empty layout, a cell is picked up at a time and placed into its row by OneCell(*c*), detailed later in this subsection. As a cell is placed, its ribs are constructed based on *Property 1.* When all the cells are placed, the vertical M2 ribs have been constructed as well. By *Property 2*, we know the left/right ends of the spines. Then the spines in each row are packed vertically by the left-edge algorithm [11]. The packing of spines in a row is different from channel routing, because there are no vertical constraints[1]. Therefore, the left-edge algorithm can show its power in getting an optimum packing quickly. If the required horizontal tracks in a row exceed $H_C/Pitch(3)$, where $Pitch(3)$ is the wiring pitch of M3, a gap between the row and the row immediately above is created to produce extra tracks. But in the gap, M1 is also available; so $Pitch(1)/(Pitch(1)+Picth(3))$ of the overflowing tracks are assigned to M3 in the gap and the remaining to M1. Finally the redundant M2 segments are chopped off. An example is shown in Figure 5, outlining these operations.



(a) empty rows (b) repeat OneCell(*cell*) (c) all cells and M2 ribs

[1] A vertical constraint in the channel routing is present, if pin A is on the top of the channel and pin B on the bottom and they are on the same vertical track. Obviously the horizontal segment A should be placed above segment B.



(d) packing of M3 spines in rows (e) chopping off redundant M2

Figure 5:The operations of All_Cells().

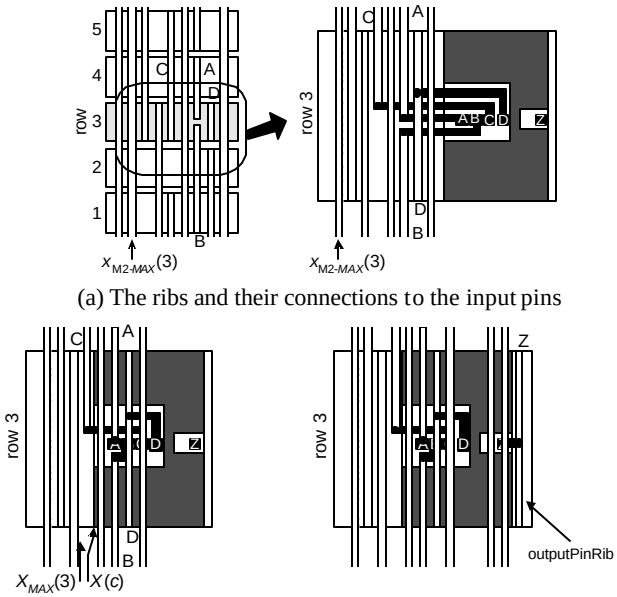
The rest of the problem is how to determine from which sequence (row) the next unplaced cell is picked up. The idea is to keep all rows marching with similar paces. An ordered list Q is set up, each entry being a row (a sequence). The rows are sorted in the ascending order of their current right end position $X_{MAX}(r)$. Each time the row at the head of the list is taken out and its next unplaced cell c is picked up. When c is placed and its ribs arranged, $X_{MAX}(r)$ is updated. If the row still contains unplaced cells, it is inserted back into Q at the appropriate position according to its new $X_{MAX}(r)$. Variable $X_{MAX\text{-}ALL}$ keeps track of the maximum $X_{MAX}(r)$ among all rows. Every time cell c of width $W(c)$ is filled into row r, the following steps are carried out:

$$X_{MAX}(r) = X_C(c) + W(c)$$
$$X_{NAX-ALL} = \max\left(X_{MAX-ALL}, X_{MAX}(r)\right)$$
$$X_{NAX}(r) = \max\left(X_{MAX}(r), X_{MAX-ALL} - x\right)$$

x in the third formula is a constant and its value is empirically chosen as $10Pitch(2)$.

<u>OneCell(*c*)</u>

The example in Figure 6 is used to describe the process of *OneCell*(*cell*). Before cell c is placed, some other cells may have already been placed and their ribs routed.



(a) The ribs and their connections to the input pins

(b) push to the left (c) the connection of the output pin

Figure 6: Connections of the input pins and the placement of a cell.

In the example, cell c is in row 3 and has input pins A, B, C and D and output pin Z. The following steps are carried out:

(a) Place the ribs and connect them to the input pins:

At this step, as shown in Figure 6(a), the cell is just placed far right in the row. To simplify the discussion, we say a rib being "up", "down" or "this", if the spine the rib connects to is in an upper, lower or this row. Since at this moment, the spines have not been arranged in the rows, the only information is in which rows the spines are located. Therefore, we assume conservatively that a rib occupies the full M2 track of the spine's row. Starting from X coordinate:

$$X_{S0} = \max\left[X(lastOutputPinRib), \left\lceil \frac{X_{MAX}(r)}{Pitch(2)} - r \right\rceil Pitch(2) \right],$$

where r=10 is a constant, each vertical M2 track is examined to see whether it can be used by any rib(s) of the input pin(s). If not, proceed with the next M2 track. If it is usable to any rib(s), first we choose the longest unassigned rib to fill in this track. Then we investigate if a remaining unassigned rib in an opposite direction can fit in the same track. A and B in Figure 6(a) are an example. Whether the sharing can happen also depends on the connectivity of the ribs and their input pins, which will be discussed shortly. If the rib is "this" as D in Figure 6(a), its spine happening to be in the current row, it occupies one whole vertical M2 track of the row. Obviously rib of kind "this" cannot share M2 track with any other rib. Note that when the spines are finally routed, the over-occupied M2 parts of the ribs will then be truncated as shown in Figure 5(e).

Now we discuss how a rib is connected to its input pin after it is put in a vertical track. Rib on the right side of its input pin is not allowed in ISCPD. An input pin routes on M1 to its rib. The leftmost pin just uses a horizontal M1 wire, and the rests may either go "above" or "below". If two pins both go "above" ("below"), the pin on the right must be above (below) the pin on the left. Recall that in the adjusted view of Figure 1(c), parameters $l_{B\text{-}IP}$ and $l_{A\text{-}IP}$ were defined that limit the horizontal M1 tracks below and above the input pins. Although for almost all library cells $l_{B\text{-}IP}+l_{U\text{-}IP}$ is greater than or equal to $N_{IP}(c)-1$, the number of input pins of cell c excluding the leftmost one, $l_{B\text{-}IP}$ or $l_{A\text{-}IP}$ alone may be smaller than $N_{IP}(c)-1$. This prevents us from arbitrarily selecting a horizontal M1 track for the connection of the rib and its input pin. A heuristic is that when a M1 track is assigned, we try to keep the numbers of the remaining "above" and "below" M1 tracks as balanced as possible. A special case is that attempt is made to put a rib in the same M2 track of a previous rib of the cell. Of course the sharing can only occur when one of them "goes down" while the other "goes up".

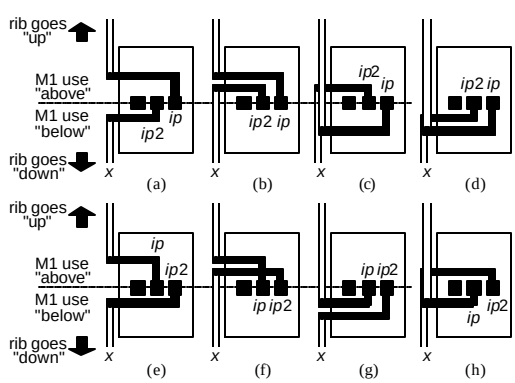


Figure 7: The sharing of the M2 track by two ribs.

In Figure 7, the rule for two ribs sharing the same M2 track is illustrated. In the figure, *ip* is already assigned for it has the longest rib that fits in the M2 track at *x*, and the rib of *ip* goes "up". Now we test if another rib *ip*2 going "down" can share the same M2 track. The first case, Figure 7(a)-(d), is that *ip*2 is to the left of *ip*. If *ip* already "uses above", we only need to find a M1 track for *ip*2, either below or above, that is available, as shown in Figure 7(a) and (b). When doing the track selection, the balancing mechanism as described before is employed. If *ip* already "uses below", there will be no way to connect to the *ip*2 rib without causing overlap, as illustrated in Figure 7(c) and (d). The second case, Figure 7(e)-(h), is that *ip*2 is to the right of *ip*. If *ip* already uses "above", *ip*2 can only use "below" as in Figure 7(e). Crossing may occur if *ip*2 uses "above" as shown in Figure 7(f). If *ip* already uses "below", *ip*2 must also use "below", as in Figure 7(g). If *ip*2 uses "above" as in Figure 7(h), overlap occurs. Symmetrical analysis can be derived, in which *ip* goes "down" and *ip*2 goes "up". The rule also works for the leftmost input pin, but without distinguishing "below" and "above". If the rule cannot be obeyed for reason like there are insufficient M1 tracks available, the sharing is rejected and the rib must be placed in another M2 track.

(b) Push the cell to the left:

The cell is then pushed to the left, as illustrated in Figure 6(b), until either it touches $X_{MAX}(3)$, or any of the input pin reaches its vertical M2 rib. Thus the cell position $X_C(c)$ is obtained, and $X_{MAX}(3)$ is updated.

(c) Connect the output pin:

Next, as shown in Figure 6(c), the output pin searches for its M2 track, making preparation for the later connection to the spine on M3 in this row. The M2 track of the output pin, called the outputPinRib, is just as the "this" rib, in the sense that it occupies one full M2 track in this row and the track can not be shared by others. Recall that the output pin has a region around it in the cell (the dotted region in Figure 1(b)) such that only the output pin itself can use the M1 resource in this region. This means that the M2 track of the output pin blocks all future routing in the row to take place on its left.

3.5. Post processing: simulated-annealing based improvement

The layout produced by a run of bi-partitioning and AllCell() can be further improved via simulated-annealing. At each annealing step, a pair of cells or I/Os are swapped and the layout is reconstructed with AllCell() and area and wire length re-evaluated. The main application of this algorithm is the generation of the soft standard-cell modules for the floorplanner. Hence simulated-annealing improvement which can be slow may not be so interesting to us. But we are curious in finding out how much layout area and wire length that can be further reduced via annealing.

4. Experimental results

In the experiment, fourteen examples were tested. The examples are from the LSynth91 synthesis benchmark set; the cell netlists were generated by *SIS* through technology independent optimization and technology mapping [8-10]. All clock nets were removed. The characteristics of the netlists are given in Table 1. A 0.35μm technology was used; the routing parameters are given in Table 2. The widths of the power/ground rings are 10 μm, and the I/O height is 5.0μm. We compare our algorithm with *Cadence Silicon Ensemble* version 5.3 (SE) running on the same machine. The standard-cell physical design in SE includes row generation, I/O placement, cell placement (*Qplace*) and routing (*Wroute*). The user has to define either the module size or the row utilization (ratio of the total cell area and the total row area) to enable the generation

of the layout outline and the rows. All programs were run on a Sun Blade 1000 workstation. The ISCPD program was written in Java.

Table 1: Testing examples

circuit	#gate	#eqi. gate	#I/O	#net	#pin
mm4a	181	277	11	188	493
C3540	1342	1641	72	1392	3700
s5378	1767	3030	81	1802	4700
des	3558	4522	501	3814	10606
dsip	3647	6172	425	3875	10242
C5315	4908	6307	301	5086	14294
C7552	5932	7162	313	5825	16121
i10	6778	8238	481	6554	18169
C7552L	8461	10793	313	8667	24464
C6288S	8493	10091	64	8525	23294
C6288	11921	14399	64	11889	33188
mult32	12419	13238	96	12451	32266
s38417	16633	29786	134	16661	43937
clma	24497	45455	464	24879	80242

Table 2: Design rules

	allowed direction	preferred direction	pitch
M1	both	horizontal	1.0μm
M2	vertical	vertical	1.2μm
M3	horizontal	horizontal	1.5μm

For each example, we tried to find the smallest but violation-free implementation by SE. The row utilization was gradually reduced until the layout was fully routable. The minimum resolution of the row utilization for this search was 5%. ISCPD was run only once for each example (ISCPD-part). Since we were interested in finding out how much improvement can be achieved by spending more time, simulated-annealing (ISCPD-SA) was tried for the ISCPD-part results. We put 90% weight on area and 10% on wire length in ISCPD-SA, because the goal of the algorithm is to find the smallest area without routing violation.

The areas and total wire lengths are reported in Table 3. Run times are given in Table 4. The time spent on the final run (minimum area) of SE is reported, and the total time for all iterations is also given. A layout example is shown in Figure 8.

Table 3: Area and total wire length

	area ($10^3 \mu m^2$) <note>			total wire length (mm)		
circuit	ISCPD part.	ISCPD SA	SE	ISCPD part.	ISCPD SA	SE
mm4a	24.4	23.3	24.9	6.4	6.1	7.1
C3540	209	200	178	88.4	74.8	92.8
s5378	400	360	327	127	149	124
des	1115	1095	937	677	612	629
dsip	827	810	778	659	625	587
C5315	1539	1467	1953	812	683	822
C7552	1866	1795	1777	898	899	753
i10	2359	2259	2039	1213	1144	979
C7552L	1902	1874	2081	733	751	565
C6288S	3941	3727	3681	1716	1721	1650
C6288	3496	3412	3538	1339	1255	882
mult32	2453	2324	4059	1124	972	1023
s38417	6869	6764	8512	3192	2785	2795
clma	24298	23391	24794	11193	9815	10489
compare	99%	96%	**100%**	112%	105%	**100%**

<note> the power/ground rings and the I/O height may noticeably affect the area of the small circuits.

Table 4: Run time (minutes)

			final run			all runs
circuit	ISCPD part.	ISCPD SA	SE place	SE route	SE P&R	SE P&R
mm4a	0	0.01	0.07	0.64	0.71	3.8
C3540	0.05	0.63	0.16	0.68	0.84	4.6
s5378	0.08	1.08	0.2	0.72	0.92	4.5
des	0.75	6.7	0.49	1.15	0.64	4.9
dsip	0.43	6.13	0.49	0.75	1.24	5.2
C5315	1.03	7.5	0.7	0.7	1.40	5.1
C7552	2.23	9.2	0.93	1.23	2.16	7.0
i10	1.7	7.7	1.18	1.27	2.45	8.3
C7552L	2.5	10.3	1.33	1.38	2.71	8.8
C6288S	5.19	13.6	1.72	1.47	3.19	12
C6288	5.73	15.6	2.1	1.79	3.89	16
mult32	5.53	16.5	2.35	2.64	4.99	22
s38417	11.2	23.3	18.7	97.6	116.3	377
clma	56.7	44.4	25.8	234	259.8	790
compare	63%	282%		**100%**		390%

<figure omitted>

Figure 8: Layout pictures of C7522. A five pin net is highlighted.

The results show that on average the partitioning-based and simulated-annealing improved ISCPDs are 1% and 4% respectively smaller in area than SE. In terms of total wire length, ISCPD-part and ISCPD-SA are on average 12% and 5% worse than SE. Comparing the results of ISCPD-part and ISCPD-SA, we find that the initial layout generated right after partitioning is already close to the final layout produced through simulated-annealing.

The run time of ISCPD-part is on average 37% faster than that of the **final** SE run with complete placement and routing (even though for experimental reasons ISCPD is written in Java). To obtain the final results of SE, several iterations might be involved with successively adjusted row utilization. If the total run time including all the SE runs is compared, SE can be five times slower than ISCPD-part. It is even slower than ISCPD-SA.

The run time of ISCPD-SA is longer; ISCPD-SA takes 182% more time. This experiment was done only to see how competitive ISCPD could be if used on the soft modules in the final step of full-chip Physical Synthesis flow. Using ISCPD in the final step would offer some additional benefits, that are hard to quantify, such as regularity and ability to rearrange the spine routing to account for cross-talk problems. Note that in ISCPD-SA, the user has the full freedom in setting a time-out for the simulated-annealing run, and whenever the annealing is stopped, the layout is always valid and at least about -3% area compared to the final one can be expected. (RKB: Fan, I did not quite understand this -3% comment)

5. Conclusion

In this paper, an integrated standard-cell placement and routing algorithm called ISCPD is presented. The core of the algorithm is a procedure that translates the sequences of cells in the rows into a placement of both the cells and the interconnections. The initial sequences are produced through a recursive bi-partitioning based on the spine model.

ISCPD (without simulated-annealing) is designed to be used in applications where quick construction of a 100% legal SC layout with good area and wire length results is sought. Full routability is always guaranteed. The algorithm needs no user input of either the

layout area or utilization. Instead the area is an output of the algorithm, preventing iterations to search for the smallest area that is still fully routable.

In the following we discuss several interesting problems and possible future extensions.

(1) Further improvement to the quality of the ISCPD-part results is not limited to applying ISCPD-SA. Any commercial router can work directly on the ISCPD-part placement to achieve better wire length and/or delay.

(2) The use of spine topology seems naïve at the first glance. However two factors justify its applicability. First, the fanout numbers of the nets in a standard-cell design are usually small. The superiority of the Steiner Tree topology is thus weakened. The other factor is that the cells, which carry the pins, are movable. So the application is different from that of building the net topology with short wire lengths for a fixed set of pins. In this sense, a placement in ISCPD should not create many nets which have obviously bad spine topologies (such as nets with more than one long rib on the same side of the spine).

(3) Wire widths can be adjusted during both the arrangement of the ribs and the packing of the spines to improve timing. Buffer insertion can also be incorporated easily. Another advantage is that the spine topology guarantees that the wire length is always the Manhattan distance between any output-input pin pair.

6. References

[1] M. Wang, X. Yang and M. Sarrafzadeh, "Dragon 2000: Standard-cell Placement Tool for Large Industry Circuits", ICCAD 2000, pages 260-263.

[2] X. Yang, B-K. Choi and M. Sarrafzadeh, "Routability Driven White Space Allocation for Fixed-Die Standard-Cell Placement", ISPD 2002, pages 42-50.

[3] M.C. Yildiz and P.H. Madden, "Improved Cut Sequences for Partitioning Based Placement", DAC 2001, pages 776-779.

[4] R. Kastner, E. Bozogzadeh and M. Sarrafzadeh, "Predictable Routing", ICCAD 2000, pages 110-113.

[5] H. Shirota, S. Shibatani, and M. Terai, "A New Rip-up and Rereoute Algorithm for Very Large Scale Gate Arays", CICC, 1996, pages 9.3.1-9.3.4

[6] P. Groeneveld, "Wire Ordering for Detailed Routing", IEEE Design & Test of Computers, vol 6, 1989, pages 6-17

[7] F. Mo and R.K. Brayton, "Fishbone: A Block-Level Placement and Routing Scheme", ISPD 2003, pages 204-209.

[8] E. Sentovich, K. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. Stephan, R. Brayton and A. Sangiovanni-Vincentelli, "SIS: A system for sequential circuit synthesis", Tech. Rep., UCB/ERL M92/41, Electronics Research Lab, University of California, Berkeley, May 1992

[9] R. Brayton, G. Hachtel and A. Sangiovanni-Vincentelli, "Multi-level logic synthesis', Proc. of IEEE, vol. 78, Feb. 1990

[10] http://www.cbl.ncsu.edu/pub/Benchmark_dirs/LGSynth91/

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

[12] B.S. Landman and R.L. Rosso, "On a Pin Versus Block Relationship for Partitions of Logic Graphs", IEEE Trans. Comp, G-20, 1971, pages 1469-1479.

[13] C.M. Fiduccia and R.M. Mattheyses, "A Linear-Time Heuristic for Improving Network Partitions", DAC 1982, pages 171-181.

[14] A.E. Caldwell, A.B. Kahng and I.L. Markov, "Can Recursive Bisection Alone Produce Routable Placement?", DAC 2000, pages 477-482.