LESAR: A Dynamic Line-End Spacing Aware Detailed Router

Ying-Chi Wei, Radhamanjari Samanta, Yih-Lang Li

Department of Computer Science, National Chiao-Tung University, Hsin-Chu, 300, Taiwan

***Abstract*—As the VLSI technology scales down, 193nm optical lithography reaches the limit and one-dimensional (1D) unidirectional style lithography technique emerges as one of the most promising solutions for coming advanced technology nodes. The 1D process first generates unidirectional dense metal lines and then use line-end cutting to form the target patterns with cut masks. If cuts are too close, they will lead to conflicts. Line-end spacing rules become dynamic rather than static because of cut mask and also now need to be followed strictly. Line-end spacing check between two line-end pairs in the same mask has also been regarded as compulsory line-end spacing constraints that have not discussed in previous works yet. Complying with these rules during APR has become a new bottleneck. In this work, we propose to make the router aware of the dynamic line-end spacing rules, including end-end spacing and parity spacing constraints. Experimental results of our proposed router demonstrates that it can effectively expel all end-end spacing violations as well as 75% of parity spacing violations in a reasonable runtime increase of 14%.**

Keywords—Nanowire, routing, cut mask, manufacturability, line-end spacing

I. Introduction

Optical lithography using 193 nm has empowered the industry to grow for the last few years. Double patterning lithography (DPL) or multiple patterning lithography (MPL) is used to extend 193 nm even further. 1D unidirectional style design is going to replace the 193 nm optical lithography techniques for coming advanced technology nodes and beyond. 1D style design can be divided into "lines" and "cuts" [2]. There are many technologies that can manufacture 1D unidirectional parallel dense metal lines (Fig. 1(a)) like directed self-assembly (DSA), DPL or MPL. Fig. 1(b) shows how a cut mask will trim off the metal lines. The target pattern is formed with dummy wires and line-end extension as shown in Fig. 1(c).

It is necessary to take cut spacing into consideration. More masks with higher cost or e-beam lithography (EBL) is used as complementary lithography techniques to resolve cut spacing violations [5,6]. The works in [1,3] use integer linear programming (ILP) to reduce cut spacing violations in the post cut optimization stage. In [4], line-end extension is employed to reallocate cuts in 1-D layout. A nanowire aware router is described in [5] to effectively mitigate cut numbers, cut spacing violations and line-end extension length.

In this paper, we propose a detailed routing kernel tackling dynamic line-end spacing constraints related to cut spacing design rules for the cut mask. Moreover, consider the 1D unidirectional layout design using multiple patterning

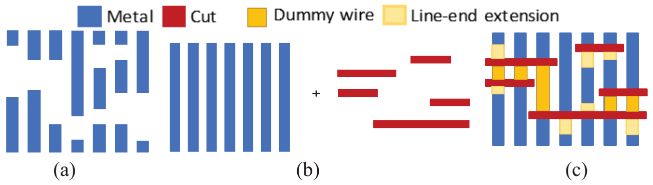


Fig. 1. The 1D target layout in (a) can be decomposed of (b) lines and cuts to form (c) 1D regular line-end pattern.

lithography technique, the dynamic line-end spacing design rules is composed of end-end spacing and parity spacing rules. Experimental results display that LESAR can generate the routing results that are free of end-end spacing violations and have less parity spacing violations by up to 75% as compared to those without considering dynamic line-end spacing constraints. The contributions of LESAR are listed in the following.

A. Line-end map is proposed for fast identification of parity spacing violations at the detailed routing kernel.

B. Instead of doing intuitive search and check during routing, pseudo obstacles are inserted on Line-end Map to guide the router to prevent parity spacing violations.

C. New routing cost formulation allowing the line-end constraints relax gradually to complete 100% routing and the new net ordering decision scheme is adopted.

The rest of the paper is organized as follows. In section II, two kinds of line-end spacing rules and problem formulation is introduced. Section III describes the detailed routing kernel for the proposed work. Section IV presents the experimental results and section V concludes the work.

II. Preliminaries

A. A. Dynamic End –End Spacing Rules

The line-end spacing is now a new bottleneck for back-end processing. For clarity, we use DPL for illustration in this paper. For a 1D routing design, all vertical wires of the same routing layer in odd columns are assigned to a mask while all wires in even columns are in the other mask. The router needs to consider the end-end spacing of two opposite wire line ends and the spacing of line-ends at parity neighboring tracks during APR. As shown in Fig. 2(a) and 2(b) respectively, the end-end spacing between line ends must absolutely be one of default end-end spacing, x and nx, or larger than nx, where x is the minimum spacing between two adjacent metal wires and n is a positive integer. For parity spacing, if there are two line end pairs with x spacing on the same parity (odd or even) neighboring tracks, either they should be aligned as illustrated in Fig. 2(c), or the

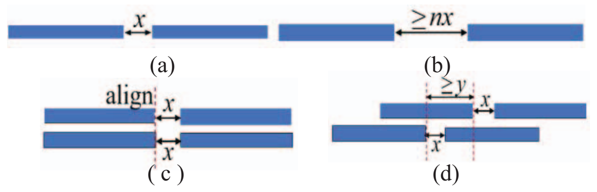
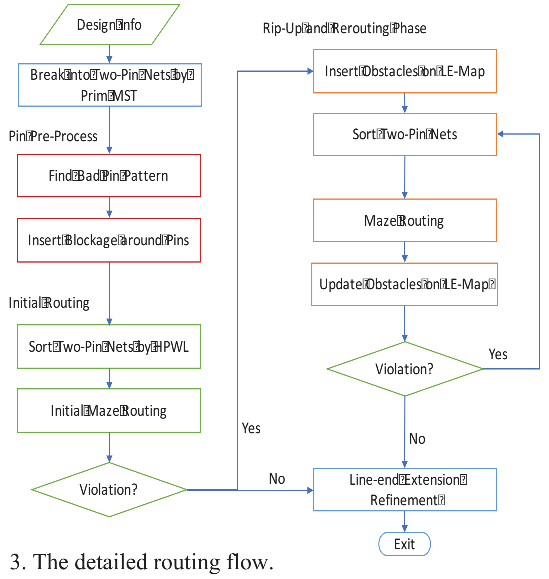


Fig. 2. Line-end spacing rules and wire patterns. (a) end-end spacing with normal x spacing; (b) end-end spacing with $\geq nx$ spacing; (c) two line-end pairs with x spacing must be aligned; or (d) with the parity offset $\geq y$.



Design Info
Break into Two-Pin Nets by Prim MST
Pin Pre-Process
Find Bad Pin Pattern
Insert Blockage around Pins
Initial Routing
Sort Two-Pin Nets by HPWL
Initial Maze Routing
Violation?
Yes
No
Rip-Up and Rerouting Phase
Insert Obstacles on LE-Map
Sort Two-Pin Nets
Maze Routing
Update Obstacles on LE-Map
Violation?
Yes
No
Line-end Extension Refinement
Exit

Fig. 3. The detailed routing flow.

offset of them should be larger than the parity spacing y that equals to mx as shown in Fig. 2(d), where m is a positive integer. Otherwise it will cause a parity violation. In this paper, the tracks in all Figures (e.g. Fig 2(c) and Fig. 2(d)) associated with parity spacing are of the same parity.

B. Problem Formulation

A netlist, a multiple layer design with a set of design rules including wire width, wire spacing, end-end spacing, parity spacing, available tracks and preferred direction (vertical or horizontal direction) of each routing layer are given. Besides, initial routing obstacles are also given. The objective is to develop a grid-based detailed routing kernel to determine the routing paths of all two-pin nets without overlapping violations and open violations and to minimize the dynamic line-end spacing violations of the design.

III. Routing Kernel

In this section, we will introduce the detailed routing algorithm considering dynamic end-end spacing rules including pin pre-processing to prevent potential violations, identification of line-end spacing violations, routing cost formulation design and the routing order of nets in our router.

A. Overview of the Routing Algorithm

Fig. 3 displays the proposed routing flow. At the very beginning, Prim's minimum spanning tree (MST) algorithm is conducted to decompose every multi-pin net into several two-pin nets. The pin pre-processing is next proposed to prevent potential parity violations around net pins. Then, the initial routing phase identifies the preliminary solutions. Next, the two-pin nets with violations undergo rip-up and rerouting iteratively until there is no violation found in the design. In the rip-up and rerouting phase, the line-end status will be inserted to the line-end map according to the line-ends of each net and the locations where the parity violations may occur are protected by inserting pseudo obstacles. Next, all two-pin nets having violations are sorted by net score, wire-length and parity violations. Rerouting is conducted by applying maze routing algorithm and adopting a routing cost function. After a new path of a two-pin net is found, the line end map is updated based on the line-ends of the new path. Finally, the line-end extension is performed to fit the line-end spacing rules and reduce end-end spacing violations.

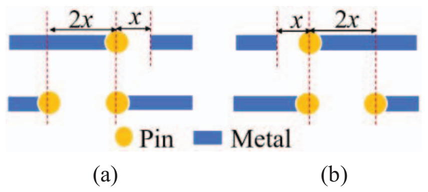


Fig. 4. Bad pin patterns cause parity violations easily. (a) bad pattern 1 with parity violation; (b) bad pattern 2.

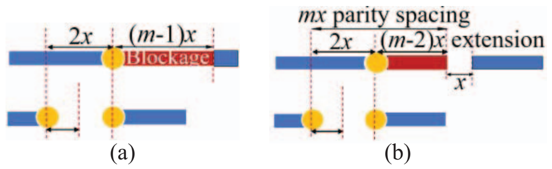


Fig. 5. Add blockage around pins to prevent parity violations.

B. Pin Pre-processing

Bad pin patterns tend to cause parity violations since pin placement algorithms do not consider the line-end spacing issues. We add one stage, pin pre-processing, to avoid parity violation. Assume parity spacing mx is larger than 2, the routing in Fig. 4 has parity spacing violation. However, inserting blockages at the possible locations will discourage wires entering into these locations, thereby avoiding potential parity violations. (Fig. 5). Blockage of length $(m\text{-}1)x$ is added as shown in Fig. 5(a). Therefore, the next line segment avoids the blockage. Line-end extension also can avoid parity spacing violation. In Fig. 5(b), the line segment is extended by $(m\text{-}2)x$ length to make the line end spacing x, then also the parity spacing in the neighboring parity tracks becomes mx to satisfy the parity spacing constraint.

C. Identification of Parity Spacing Violation

In this section, we introduce the kinds of wire patterns to be recognized as potential parity spacing violation candidates. As Fig. 6 shows, there is a line-end pair separated with the distance $d = x_{e2} - x_{e1}$. If $d < nx$ (end-end spacing), then we check whether the adjacent parity tracks have parity violations or not. In Fig. 6, $c = mx - d$. The checking intervals are from x_{el} to x_{c1}, where $x_{c1} = x_{e1} - c$, and from x_{e2} to x_{c2}, where $x_{c2} = x_{e2} + c$. If there is a line-end pair in one of the checking intervals, then parity violation will happen.

D. Line-end Map

It is time-consuming and not realistic to intuitively search for checking the line-end spacing violations during routing. Thus,

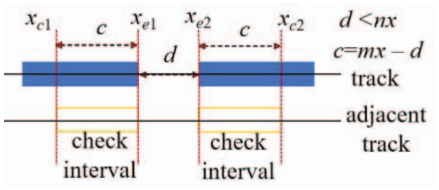


Fig. 6. Parity spacing check.

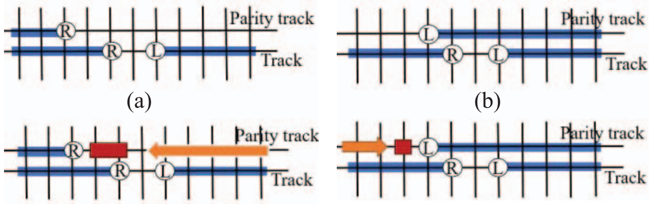


Fig. 7. Line-end pattern. (a,b) pattern leading to parity violation easily. (c,d) Add obstacles to prevent from parity violation.

we propose the Line end map which can guide the router during routing to avoid the region that may cause line-end spacing violation. At each iteration, the line-end status is updated after maze routing finds a path for a net. The line-ends of a net are added into the line-end map and obstacles are also inserted to avoid potential parity violation. Fig. 7(a) and Fig. 7(b) show the line-end patterns that tend to cause parity violation. To prevent from parity violation, obstacles are added to the map as shown in Fig. 7(c) and Fig. 7(d), where a wire is routed to the top track at the grid next to the obstacle. It is easy to resolve the parity and end-end violations by extending the right end at the top track to the right by two grids and the left end at the bottom track to the left by one grid such that two line end pairs are aligned (Fig. 7). Also, the end-end and parity violations can be expelled by extending the left end at the top track to the left by one grid and the right end at the bottom track to the right by one grid such that the end-end spacing is x for each line-end pair and parity spacing is $4x$ (assume m=4 for parity spacing rule). If a wire steps on the obstacles during maze routing exploring, the high penalty of inducing parity violation is imposed on the path. As a wire approaches any obstacle, the wire of aligned pattern or shifted pattern with offset $\geq mx$ by line-end extension is preferred as the examples are displayed in Fig. 7(c) and Fig. 7(d) respectively.

E. Routing Cost Formulations

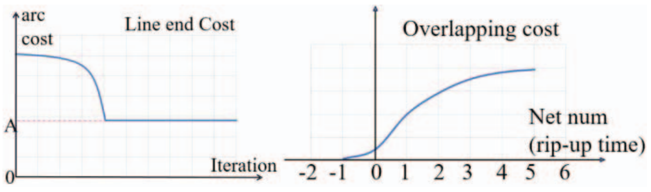
The router adopts the routing cost function defined as follows:

$$C_e = L_e + B_e + (O_e + V_e) \times H_e \qquad (1)$$

where C_e is the cost of edge e, L_e is the line-end cost, B_e is the basic unit wirelength cost, O_e is the overlapping cost, V_e is the via usage cost, and H_e is the history cost [7].

$$L_e = \begin{cases} arc\cot(\alpha \times iteration - \beta), & if\ N_r(n) < \delta \\ A, & otherwise \end{cases} \qquad (2)$$

where α, β and A are user-defined parameters. $N_r(n)$ is the number of times of rip-up for the currently routed net n and δ is the threshold value for the times of rip-up. The function curve is shown in Fig. 8(a). Initially, the cost is high and then it gradually decreases and relaxes as the number of iteration increases. The overlapping cost [8] is defined as follows:



(a) (b)

Fig. 8. (a) The function curve of line-end cost; (b) the function curve of overlapping cost.

$$\begin{cases} \frac{B}{1+e^{-\gamma(N_O - \varepsilon)}}, & if\ N_O(e) \geq 2 \\ \frac{B}{1+e^{-\gamma(N_r(n) - \varepsilon)}}, & if\ N_O(e) = 1\ and\ net(e) \neq n \\ \frac{C}{1+e^{-\gamma(0)}}, & if\ N_O(e) = 0 \\ 0, & if\ N_O(e) = 1\ and\ net(e) = n \end{cases} \qquad (3)$$

where γ, ε, B and C are user-defined parameters. $N_O(e)$ is the number of overlapping nets on edge e, and $net(e)$ is the net of wire on edge e. The function curve is shown in Fig. 8(b). A logistic function is used to evaluate the overlapping cost. As maze routing explores next grid through edge e, the router uses different sets to estimate overlapping cost, e.g., if 2 or more nets overlap on a grid, we set a cost proportional to its congestion value. If a net has been ripped-up several times, the cost increases to prevent from other nets passing through it.

F. Net Ordering

Net ordering is used to determine the routing priority of all nets. The routing net ordering of all two-pin nets impacts the routing quality and runtime. If the net ordering is bad for routing, it often requires much more iterations of rip-up and rerouting to achieve a high-quality routing. The two-pin nets are sorted by net score, wire-length and parity violations. The net score is composed of two factors, i.e., overlapping violation and rip-up times. These two factors are normalized to 0 ~ 1 among ripped-up two-pin nets. The net score of a two-pin net n is defined as follows:

$$Score(n_i) = p \times Norm(N_O(n_i)) + q \times Norm(N_r(n_i)) \quad (4)$$

where p and q are user-defined parameters for balancing the weightage of two factors. For our experiments, we have given twice the weightage to overlapping violations as compared to rip up times, i.e. $p = 2q$. $N_O(n_i)$ is the number of overlapping violations on net n_i, and *Norm* is a normalization function to normalize the input value into a value between 0 and 1. Net ordering is determined by performing sorting 2 or 3 times by different sorting keys. First, all the two-pin nets are sorted in increasing order of net score. Then they are sorted stably in increasing order of wire-length. Finally, if there is no overlapping violations or open nets, they are sorted with stable sorting in increasing order of parity violations.

IV. Experimental Results

This work is implemented in C++ with Si2 OpenAccess [9]. All the experiments are performed on Intel Xeon CPU E5620 2.40 GHz machine with 112 GB memory. We modify and scale the Nangate 15nm open cell library [10] and use it to synthesize the modules from OpenSPARC T1 [11] and ISCAS99 with Design Vision [12]. The placement and the routing results are generated by SOC Encounter (not the newest version [13]). All

TABLE 1 . Description of Modules

Design	Core Utilization	Standard Cells	Nets	Layers
b15	0.7	3570	3716	4
efc	0.94	1309	1379	4
sparc_exu_alu	0.78	1175	1576	4
sparc_exu_div	0.70	2910	3253	4
sparc_exu_ecc	0.75	1365	1595	4
ctu	0.7	13396	13656	5
fpu_add	0.78	8913	9270	4
fpu_mul	0.76	14386	14676	4
fpu_div	0.985	3605	3834	3

modules are illustrated in Table 1.

Table 2 compares the SOC Encounter routing results with ours. The table includes via number (Via#), wire-length (WL), end-end violation number (E. Vio), parity violation number (P. Vio), and the results after doing line-end extension (WL2, E. Vio2 and P. Vio2). SOC Encounter does not consider dynamic line-end spacing rules but LESAR does. Two routing results are listed in the columns "Before Line-End Extension". Next, for two routing results, each line-end is extended such that two adjacent line-end pairs are separated by minimum distance. Both the two routing results are free of end-end violations. LESAR is well aware of the occurrence of parity spacing violation during routing while SOC Encounter is not. Two routing results with line-end extension are listed in the columns "After Line-End Extension". The experimental results illustrate that the total number of parity spacing violation is reduced with the increasing of via number and the wire-length is quite close to the wire-length in original case. It also reveals the router will generate more via since it will change layer more often to comply with line-end spacing rules and to prevent from line-end spacing violation. Table 2 demonstrates that the proposed router LESAR can effectively reduce the parity violations by 75% with average 12.17% increase in via numbers compared to SOC Encounter. For runtime comparison, we turned off the feature of considering dynamic line-end spacing violation of LESAR and conducted routing on the same designs. As compared to the routing of LESAR that has turned off the feature of handling dynamic line-end spacing violation, the average runtime increase of LESAR is about 14%.

V. CONCLUSIONS

We propose a dynamic line-end spacing violation aware detailed routing that is the first work to consider parity spacing violation. The Line-end Map concept is proposed to fast identify end-end spacing and parity spacing violations during routing. Instead of intuitive search during routing, Line-end Map is used to guide the router to avoid parity spacing violations. The dynamic line-end spacing violation cost is designed as an arctangent form and well integrated into the routing cost function. The experimental results show that our router can effectively reduce the dynamic line-end spacing violations with less wirelength at the cost of 14% increasing runtime and 12% increasing via number.

REFERENCES

[1] Y. Du, H. Zhang, M. Wong, and K.-Y. Chao, "Hybrid lithography optimization with E-beam and immersion processes for 16nm 1-D gridded design," in Proc. ASPDAC, 2012, pp. 707–712.

[2] M.C. Smayling. 1d design style implications for mask making and cebl. In Proc. of SPIE 8880, pages 888012–1–888012–8, 2013.

[3] J. Ou et al., "Directed self-assembly cut mask assignment for unidirectional design," J. Micro/Nanolith. MEMS MOEMS., Vol. 14(3), 2015.

[4] Z. Xiao, Y. Du, M. D. F. Wong, and H. Zhang, "DSA template mask determination and cut redistribution for advanced 1D gridded design," in *Proc. SPIE*, vol. 888017. Monterey, CA, USA, 2013, Art. no. 888017.

[5] Y.-H. Su and Y.-W. Chang, "Nanowire-aware routing considering high cut mask complexity," in ACM/IEEE Design Automation Conference (DAC), 2015, pp. 138:1–138:6,

[6] J. Kuang, E. F. Young, and B. Yu, "Incorporating cut redistribution with mask assignment to enable 1d gridded design," in Proc. (ICCAD), Nov. 2016, pp. 48–55.

[7] W.-H. Liu, W.-C. Kao, Y.-L. Li, and K.-Y. Chao, "NCTU-GR 2.0: Multithreaded collision-aware global routing with bounded-length maze routing," IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol. 32, no. 5, pp. 709–722, May 2013.

[8] M. Pan and C. Chu, "FastRoute: A step to integrate global routing into placement," in Proc. ICCAD, 2006, pp. 464–471.

[9] "Si2 OpenAccess" https://projects.si2.org/oac_index.php

[10] "NanGate Free PDK15 Open Cell Library" http://www.nangate.com

[11] "OpenSPARC T1" http://www.oracle.com/technetwork/systems/opensparc/index.html

[12] "Synopsys Design Vision" https://www.synopsys.com

[13] "Cadence SOCEncouter" https://www.cadence.com

TABLE 2 COMPARISON BETWEEN THE PROPOSED LINE END SPACING AWARE ROUTER (LESAR) & SOC ENCOUNTER RESULTS

Benchmark	SOC Encounter							LESAR						
		Before Line-end Extension			*After Line-end Extension*				*Before Line-end Extension*			*After Line-end Extension*		
	Via#	*WL*	*E-E Vio*	*Parity Vio*	*WL2*	*E-E Vio2*	*Parity Vio2*	*Via#*	*WL*	*E-E Vio*	*Parity Vio*	*WL2*	*E-E Vio2*	*Parity Vio2*
b15	12783	23795.89	2768	0	23873.67	0	77	14696	22996.02	2234	0	23082.14	0	13
efc	3712	6010.52	854	0	6036.72	0	24	4370	5799.09	761	0	5829.17	0	7
sparc_exu_alu	4430	10141.23	969	0	10169.95	0	22	5247	9818.18	770	0	9849.28	0	4
sparc_exu_div	10151	24976.48	1903	0	25033.09	0	20	11602	24310.10	1490	0	24369.77	0	3
sparc_exu_ecc	4045	11407.42	778	0	11430.28	0	11	4644	11127.92	612	0	11151.89	0	1
fpu_add	30154	71238.47	6200	2	71426.88	0	125	34355	69195.38	5272	0	69403.93	0	39
fpu_mul	44965	97221.60	8244	1	97459.19	0	95	50065	94104.05	6329	0	94339.74	0	19
fpu_div	10602	16615.67	2831	0	16714.76	0	85	12897	15457.26	2537	0	15560.36	0	34
ctu	43219	110064.60	6050	0	110244.42	0	63	46155	106926.09	4351	0	107092.27	0	13
Ratio	1.0				1.0	1.0	1.0	1.12				0.97	1.0	0.25