Flexible Packed Stencil Design with Multiple Shaping Apertures for E-Beam Lithography*

Chris Chu
Department of Electrical and Computer Engineering
Iowa State University
Ames, Iowa 50011
e-mail: cnchu@iastate.edu

Wai-Kei Mak
Department of Computer Science
National Tsing Hua University
Hsinchu, Taiwan 300 R.O.C.
e-mail: wkmak@cs.nthu.edu.tw

Abstract— Electron-beam direct write (EBDW) lithography is a promising solution for chip production in the sub-22nm regime. To improve the throughput of EBDW lithography, character projection method is commonly employed and a critical problem is to pack as many characters as possible onto the stencil. In this paper, we consider two enhancements in packed stencil design over previous works. First, the use of multiple shaping apertures with different sizes is explored. Second, the fact that the pattern of a character can be located anywhere within its enclosing projection region is exploited to facilitate flexible blank space sharing. For this packed stencil design problem with multiple shaping apertures and flexible blank space sharing, a dynamic programming based algorithm is proposed. Experimental results show that the proposed enhancement and the associated algorithm can significantly reduce the total shot count and hence improve the throughput of EBDW lithography.

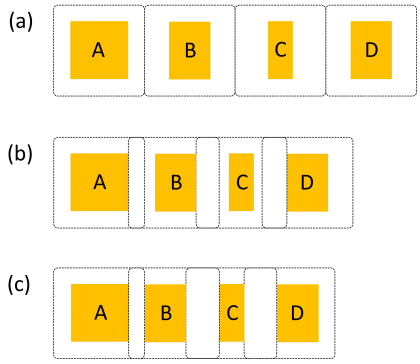

Fig. 1. (a) Four adjacent characters placed without overlapping their blank spaces. (b) The same characters are placed with their blank spaces overlapping to reduce the overall area as considered in [9, 10]. (c) Flexible blank space sharing by relocating the character patterns within their projection regions to further reduce the overall area as considered in this paper.

I. Introduction

With the continued delay of the introduction of extreme ultraviolet (EUV) lithography, the semiconductor industry is exploring other alternatives for manufacturing chips at ever smaller process nodes. E-beam direct write (EBDW)is a promising solution [1–3] and is being pursued by companies like TSMC. With the expected move to the next generation silicon wafers increasing the wafer size from 300 to 450mm, it will make massively parallel EBDW even more appealing [3].

E-beam direct write technology commonly employs the character projection method in which complex patterns, called characters, are printed onto a wafer [4]. An e-beam writing system has a stencil which can hold a set of characters. Patterns in a circuit that correspond to a character in the stencil can be projected in one shot. However, other patterns that do not match any character need to be fractured into constituent rectangles. Then each constituent rectangle requires its own shot and has to be printed in the variable shaped beam (VSB) mode [5].

Traditionally, a standard stencil adopts a grid-based layout with pre-designated spots for characters [6, 7]. Each character pattern can be of any size or shape up to the maximum allowed size, which is dictated by the shaping aperture. Since there is a fixed number of N pre-designated character spots on the grid, one just needs to pick the N most beneficial characters and put them into these spots. While this arrangement is simple, it is too restrictive. It was pointed out in [7, 8] that by carefully arranging and packing the different sized characters on a stencil, the final number of characters that can be put on a stencil can be greatly increased. It is because two adjacent characters can be placed with their blank spaces overlapping as illustrated in Fig. 1(b). This will allow more patterns on the wafer to be shot as characters leading to reduced write time and cost. Packed stencil design was studied in [9] and [10].

For the optimal use of the stencil, we consider two enhancements in packed stencil design over previous works [9, 10]. Firstly, we note that previous works are limited to the case that there is only a single shaping aperture, but the use of multiple shaping apertures with different sizes is possible [4, 11] as shown in Fig. 2. Since the enclosing projection region size of a character is determined by the shaping aperture size, using a smaller shaping aperture for smaller character patterns is helpful for packing more characters into the stencil. Secondly, the existing packing algorithms did not exploit the fact that the pattern of a character can be located anywhere within its enclosing projection region (as long as it is not too close to the region boundary) to maximize blank space sharing of adjacent characters. For example, consider the characters in Fig. 1(a), we can reduce the total area occupied by the three characters even further compared to Fig. 1(b) through *flexible blank space*

*This work was supported in part by the National Science Council, under Grant NSC 102-2220-E-007-013.

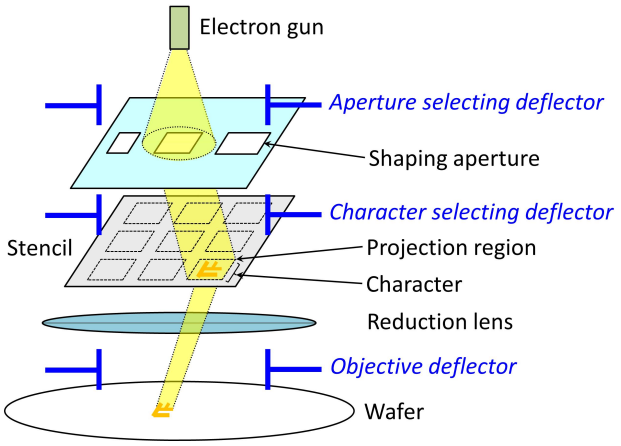

Fig. 2. A e-beam writing system with three shaping apertures.

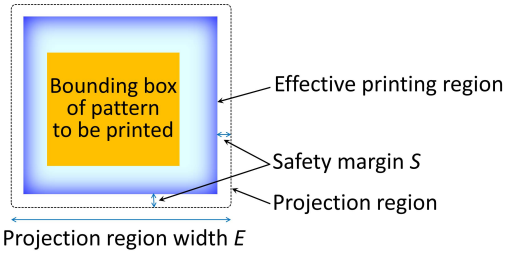

Fig. 3. The projection region and effective printing region for a character.

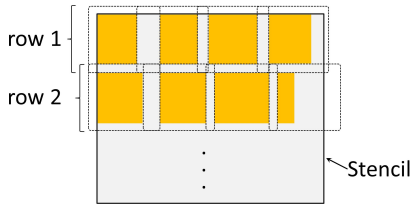

Fig. 4. Row-based stencil design.

sharing as illustrated in Fig. 1(c). Flexible blank space sharing can maximize total blank space sharing by relocating the character patterns within their projection regions.

In this paper, we investigate the packed stencil design problem with multiple shaping apertures and flexible blank space sharing. To the best of our knowledge, it has not been addressed by any existing work in the literature. We present the first algorithm targeting this problem and show that it can effectively increase the number of packed characters on a stencil and significantly reduce the total shot count.

The rest of the paper is organized as follows. In Section II, we introduce some preliminary information and the problem formulation. In Section III.C, we present a dynamic programming based approach for character selection and aperture size (or equivalently, projection region size) determination under flexible blank space sharing. Then we show how to actually pack the characters into the stencil while optimizing blank space sharing in Section III.D. Finally, we show some experimental comparisons with [9] and [10] in Section IV.

II. Problem Formulation

Refer to the general e-beam machine in Fig. 2, it has multiple differently sized shaping apertures. The aperture selecting deflector can control the direction of the e-beam to shoot it through one of the shaping apertures. Subsequently, the character selecting deflector can direct the e-beam to shoot through any desired character on the stencil. Note that the dimensions of the projection region on the stencil when an e-beam is shot through a particular shaping aperture are dependent on the dimensions of the corresponding shaping aperture.

Due to scattering of electrons, features that are too close to the boundary of projection region may not be printed accurately. Hence, we define an effective printing region as the part of projection region that is at least a safety margin S away from the boundary as shown in Fig. 3. In order to print characters on the wafer with no loss of accuracy, the pattern of each character must lie within the effective printing region. In addition, the pattern of any character A cannot overlap with the e-beam projection region of another character B. Otherwise, part of character A would be printed erroneously when printing character B. However, the blank space of two neighboring characters may overlap and is desirable to overlap in order to reduce the total area occupied by the characters.

We assume that standard cells are implemented by the characters. To minimize wastage of stencil area, the heights of different character projection regions should all be set to $2S$ plus the standard cell image height. In addition, characters on the stencil should be arranged in a row-based manner and each character pattern is always placed within its projection region such that the top blank space and the bottom blank space are equal to S. In this way, the bottom blank space of a row of characters can completely overlap with the top blank space of the row of characters below as in Fig. 4 and we can pack the maximum possible number of rows of characters into the stencil. Finally, we assume that the blank space of a character can lie outside of the available character area of the stencil [8] as in Fig. 4.

Next, we introduce the notations used in the rest of the paper.

- K denotes the number of choices for character projection region widths which is equal to the number of available shaping apertures.
- S denotes the safety margin from the effective printing region to the projection region boundary for an e-beam shot.
- W denotes the width of the available character area of the stencil.
- R denotes the number of character rows that can fit in the stencil.
- n denotes the number of different characters extracted from a design.
- w_c denotes the width of the pattern of character c. For simplicity, we will refer to the width of the pattern of a character as the character width in the rest of the paper.
- r_c denotes the number of occurrence of character c in a die.
- n_{VSB_c} denotes the number of e-beam shots required to print character c by the VSB method.

The problem of flexible packed stencil design with multiple shaping apertures is formally stated below.

Def 1. *Suppose the values of K, S, W, and R for an e-beam machine with multiple shaping apertures are given. A set of n characters extracted from a design and the values of w_c, r_c, and n_{VSB_c} for each character c are also given. (i) Determine K different character projection region widths, (ii) select the characters to be put on the stencil, and (iii) pack the characters on the stencil in a row-based manner with flexible blank space sharing to minimize the total shot count for the design under the following constraints.*
(C1) The pattern of each character must lie within the effective printing region of the character.
(C2) The pattern of each character cannot lie within the projection region of any other character.
(C3) The patterns of all characters must lie within the available character area of the stencil.

Lemma 1. *The flexible packed stencil design problem is NP-complete.*

It can be shown that the 0-1 knapsack problem, which is a well-known NP-complete problem [12], can be reduced to the subproblem of selecting and packing of n characters on a stencil in which n character projection region widths are allowed. So the flexible packed stencil design problem is NP-complete.

III. Algorithm

III.A. Algorithm Outline

As the problem is very complicated, we break it down into three steps. First, we propose a dynamic programming algorithm to determine K projection region widths and to select a set of most beneficial characters that roughly can be packed on the stencil. Then we distribute the selected characters to different rows and construct tight linear packing for each row. At last, we greedily refine the solution by checking if any of the unselected characters can be added to the end of the tight linear packing of some row. The outline of the algorithm is given in Algorithm 1.

Algorithm 1 Flexible Packed Stencil Design

1: Determine K projection region widths and select a set of characters to be put on the stencil by dynamic programming.
2: Assign the characters selected in Step 1 to rows on the stencil and construct tight linear packing for each row.
3: Greedily pack some of the unselected characters at the end of each row, if possible.

Our algorithm is based on the key concepts of tight linear packing and effective character width. So, we will first introduce them in Section III.B. Then, we will describe the details of Step 1 of Algorithm 1 in Section III.C and the details of Steps 2 and 3 in Section III.D.

III.B. Tight Linear Packing and Effective Character Width

We introduce some definitions related to our problem.

Def 2. *A linear packing of a group of characters is a packing of all the given characters in a row with flexible blank space sharing satisfying constraints (C1) and (C2).*

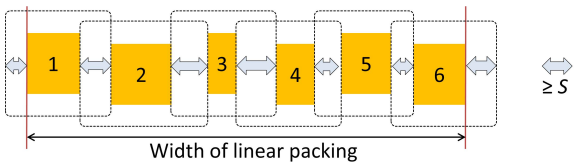

Fig. 5. A tight linear packing. Characters with even indexes are shifted down a bit to show the projection regions of all characters more clearly.

Def 3. *The width of a linear packing is the total span of the packing excluding the left blank space of the first character and the right blank space of the last character. (See Fig. 5.)*

In the following discussion, we assume that the characters in a linear packing are indexed from left to right by $1, 2, 3, \ldots$. And the width of the projection region for character i is denoted by E_i.

Def 4. *A tight linear packing is a linear packing such that for any two neighboring characters i and $i+1$, the right blank space of character i and the left blank space of character $i+1$ are exactly equal and completely overlap.*

For example, Fig. 5 shows a tight linear packing of six characters.

Def 5. *The effective width of a character c is defined as $(w_c + E_c)/2$.*

Below we show that the width of any tight linear packing is approximately equal to the total effective width of its characters.

Lemma 2. *For a linear packing P of k characters, let $W(P)$ denote the width of P, $w_i(P)$ denote the width of the i-th character ($i = 1, 2, \ldots, k$), $s_0(P)$ denote the width of the left blank space of the first character, and $s_k(P)$ denote width of the right blank space of the last character in P.*
For a tight linear packing P_t,

$$W(P_t) = \sum_{i=1}^{k} (w_i(P_t) + E_i)/2 - s_0(P_t)/2 - s_k(P_t)/2$$

For an arbitrary linear packing P_a,

$$W(P_a) \geq \sum_{i=1}^{k} (w_i(P_a) + E_i)/2 - s_0(P_a)/2 - s_k(P_a)/2$$

Proof. For a tight linear packing P_t, the right blank space of the i-th character and the left blank space of the $(i+1)$-th character are exactly equal and completely overlap for $i = 1, 2, \ldots, k-1$. So, its width can be expressed as $W(P_t) = \sum_{i=1}^{k} w_i(P_t) + \sum_{i=1}^{k-1} s_i(P_t)$ where $s_i(P_t)$ denote the width of the right blank space of the i-th character in P_t.

For each character i in P_t, we have $s_{i-1}(P_t) + w_i(P_t) + s_i(P_t) = E_i$ since its left and right blank spaces are equal to $s_{i-1}(P_t)$ and $s_i(P_t)$, respectively. Hence, $\sum_{i=1}^{k}(s_{i-1}(P_t) + w_i(P_t) + s_i(P_t)) = \sum_{i=1}^{k} E_i$. It implies that $2 \times \sum_{i=1}^{k-1} s_i(P_t) = \sum_{i=1}^{k} E_i - \sum_{i=1}^{k} w_i(P_t) - s_0(P_t) - s_k(P_t)$.

So, the width of a tight linear packing P_t can be re-written as

$$\begin{aligned} W(P_t) &= \sum_{i=1}^{k} w_i(P_t)/2 + (\sum_{i=1}^{k} E_i - s_0(P_t) - s_k(P_t))/2 \\ &= \sum_{i=1}^{k} (w_i(P_t) + E_i)/2 - s_0(P_t)/2 - s_k(P_t)/2. \end{aligned}$$

For an arbitrary linear packing P_a, we let $g_i(P_a)$ be the width of the gap between the i-th and $(i+1)$-th character patterns in P_a for $i = 1, 2, \ldots, k-1$, and let $g_0(P_a)$ be $s_0(P_a)$ and $g_k(P_a)$ be $s_k(P_a)$. Then P_a's width can be expressed as $W(P_a) = \sum_{i=1}^{k} w_i(P_a) + \sum_{i=1}^{k-1} g_i(P_a)$. Note that $g_{i-1}(P_a) + w_i(P_a) + g_i(P_a) \geq E_i$ for each character i. It implies that $2 \times \sum_{i=1}^{k-1} g_i(P_a) \geq \sum_{i=1}^{k} E_i - \sum_{i=1}^{k} w_i(P_a) - s_0(P_a) - s_k(P_a)$. So,

$$W(P_a) \geq \sum_{i=1}^{k} (w_i(P_a) + E_i)/2 - s_0(P_a)/2 - s_k(P_a)/2.$$

□

By applying Lemma 2, we can derive an upper bound of the difference between the width of a tight linear packing and the width of the minimum width linear packing and get Lemma 3.

Lemma 3. *Given a group of k characters, the width of any tight linear packing is less than $(E_1 + E_k)/2 - 2S$ away from the minimum width linear packing.*

It is apparent that if we can construct a tight linear packing, it will be a near optimal linear packing. As we will show in Section III.D, there is an efficient way to compute tight linear packing.

III.C. Projection Region Width and Character Selection by DP

In Step 1 of Algorithm 1, in order to determine the projection region widths and characters to be put into the stencil, we merge all rows of the stencil into a single row and use dynamic programming to maximize the overall shot saving subject to a total effective character width constraint.

Assume the characters are sorted in increasing order of width. We define $S[e, i, k, w]$ as the maximum shot saving using at most k different projection region widths for printing a subset of the first i characters such that the largest projection region width is e and the total effective width of the subset is at most w. The ranges of the parameters are $w_1 + 2S \leq e \leq w_n + 2S$, $0 \leq i \leq n$, $1 \leq k \leq K$, and $0 \leq w \leq RW$.

$S[e, i, k, w]$ can be expressed recursively as follow:

$$\begin{aligned} &S[e, 0, k, w] = 0 \quad \text{for all } e, k, w \\ &S[e, i, k, 0] = 0 \quad \text{for all } e, i, k \\ &S[e, i, k, w] \\ &\quad = \max \begin{cases} S[e, i-1, k, w] \\ \begin{cases} r_i(n_{VSB_i} - 1) + S[e, i-1, k, w - \frac{w_i + e}{2}] \\ \qquad \text{if } (w_i + 2S \leq e \text{ and } \frac{w_i + e}{2} \leq w) \\ 0 \qquad \text{otherwise} \end{cases} \\ \begin{cases} S[w_i + 2S, i, k-1, w] & \text{if } k > 1 \\ 0 & \text{otherwise} \end{cases} \end{cases} \\ &\qquad \text{for all } e, i \neq 0, k, w \neq 0 \end{aligned}$$

In the recursive expression above, $S[e, i, k, w]$ is the maximum over three cases. In the first case, character i is skipped. In the second case, character i is selected. It results in a shot saving of $r_i(n_{VSB_i} - 1)$ and a reduction of remaining effective width by $\frac{w_i + e}{2}$. In the third case, another projection region width of $w_i + 2S$ is used.

Then the maximum shot saving is given by $\max_i \{S[w_i + 2S, i, K, RW]\}$. It is clear that the above recursion to compute the values of $S[e, i, k, w]$ for all e, i, k, w can be implemented as a dynamic program. However, as we will see in Section 4, the memory requirement for typical problem instances can be up to tens of gigabytes. In other words, this dynamic programming formulation is not practical.

In order to reduce the memory requirement, we take advantage of the fact that many characters have the same width. Instead of considering each character separately, we group characters of the same width together. Let G_j be the group of characters of width w_j. Assume that the characters within each group are sorted in decreasing order of shot saving.

We define $S'[e, j, k, w]$ as the maximum shot saving using at most k different projection region widths for printing some subset of characters in each of the first j groups such that the largest projection region width is e and the total effective width of the subset is at most w.

$S'[e, j, k, w]$ can be expressed recursively as follow:

$$\begin{aligned} &S'[e, 0, k, w] = 0 \quad \text{for all } e, k, w \\ &S'[e, j, k, 0] = 0 \quad \text{for all } e, j, k \\ &S'[e, j, k, w] \\ &\quad = \max \begin{cases} S'[e, j-1, k, w] \\ \max_{i \in \{1,2,\ldots,|G_j|\}} \mathcal{R}(e, j, k, w, i) \\ \begin{cases} S'[w_j + 2S, j, k-1, w] & \text{if } k > 1 \\ 0 & \text{otherwise} \end{cases} \end{cases} \\ &\qquad \text{for all } e, j \neq 0, k, w \neq 0 \end{aligned}$$

where $\mathcal{R}(e, j, k, w, i)$ is the shot saving if the first i characters in G_j (i.e., the i highest shot saving characters in G_j) are included in the stencil. Let $G_j[i]$ be the set of first i characters in G_j. Then $\mathcal{R}(e, j, k, w, i)$ is given by the following expression:

$$\begin{aligned} &\mathcal{R}(e, j, k, w, i) \\ &\quad = \begin{cases} \sum_{c \in G_j[i]} r_c(n_{VSB_c} - 1) + S'[e, j-1, k, w - i \times \frac{w_j + e}{2}] \\ \qquad \text{if } w_j + 2S \leq e \text{ and } i \times \frac{w_j + e}{2} \leq w \\ 0 \qquad \text{otherwise} \end{cases} \end{aligned}$$

The three cases for $S'[e, j, k, w]$ are similar to those for $S[e, i, k, w]$ except that a set of i characters in G_j is selected instead of a single character in the second case. As the number of groups are typically at least tens of times less than the number of characters, the memory requirement to compute $S'[e, j, k, w]$ would be reduced to less than 1 gigabyte in practice as shown in Section 4. Note that this group-based dynamic program and the character-based dynamic program above generate identical solutions and have the same runtime complexity. The only difference is that the group-based approach uses much less memory.

III.D. Tight Packing Construction

Let $\mathcal{E}$ be the set of K projection region widths selected in Step 1 of Algorithm 1. The selected characters are tightly packed into the stencils in Step 2 of Algorithm 1 by Procedure 1 below:

Procedure 1 Character Packing

1: For each character i, set E_i to be the smallest value in $\mathcal{E}$ such that $E_i \geq w_i + 2S$.
2: Sort the selected characters in increasing order of $E_i - w_i$.
3: **for** each character i in sorted order **do**
4: Starting from the next row after the last packed character, find a row which i can be tightly packed at the end of it. Ignore character i if it cannot be packed to any row.
5: **end for**

In general, the projection region width E_i for each character i can be set to any value as long as it is at least $w_i + 2S$. However, E_i should be as small as possible to facilitate the packing of more characters into a row. Hence, E_i is set as in Step 1 above. According to Lemma 4 below, tight packings can be constructed by arranging the characters in increasing order of $E_i - w_i$. Hence, we can simply sort the characters in Step 2, then distribute the characters and tightly pack them to the rows in Steps 3-5.

Lemma 4. *Given a group of k characters, a tight linear packing can be constructed if the characters are ordered such that $E_i - w_i \leq E_{i+1} - w_{i+1}$ for $i = 1, \ldots, k-1$.*

Proof. We can prove the lemma by induction on k.
1. The base case that $k = 1$ is trivially true.
2. We show that if the statement is true for $k = k' - 1$, then it is also true for $k = k'$. Suppose the statement is true for $k = k' - 1$ and we have a group of k' characters. We can construct a tight linear packing for the k' characters as follows. First, we sort the k' characters such that $E_i - w_i \leq E_{i+1} - w_{i+1}$ for $i = 1, \ldots, k'-1$. By the assumption, a tight linear packing $P_{1..k'-1}$ can be formed for the first $k' - 1$ characters in the sorted order. We can append the k'-th character to the right end of $P_{1..k'-1}$ in such a way that the left blank space of the k'-th character completely overlap with the right blank space of the $(k'-1)$-th character as shown in Fig. 6. It follows that the left blank space of the k'-th character must be no smaller than the safety margin S since the right blank space of the $k'-1$ character is at least S. But it remains to show that the resultant right blank space of the k'-th character is also at least S for the constructed packing to be a feasible tight linear packing. Let δ denote the left blank space of the k'-th character (equivalently, the right blank space of the $(k'-1)$-th character). The resultant right blank space of the k'-th character is equal to $E_{k'} - w_{k'} - \delta$. Since $E_{k'} - w_{k'} \geq E_{k'-1} - w_{k'-1}$, so $E_{k'} - w_{k'} - \delta \geq E_{k'-1} - w_{k'-1} - \delta \geq S$. Hence, the constructed packing is a tight linear packing. □

After trying to pack all selected characters into the stencil, we further improve the shot saving in Step 3 of Algorithm 1 by trying to pack all unselected characters using the same packing procedure in Procedure 1.

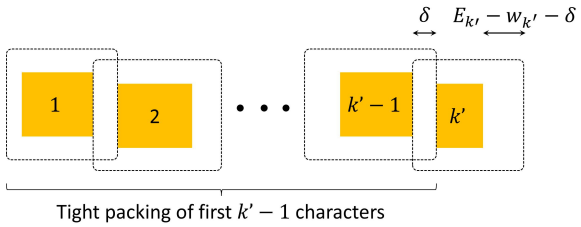

Fig. 6. Tight linear packing construction.

IV. Experimental Results and Conclusions

We implemented our approach in C and obtained the executable codes of [9] and [10] for comparison. All experiments were done on a Linux server powered by a 2.67 GHz Intel processor with 47 GB of memory.

In the first experiment, benchmarks 1D-1 to 1D-4 from [10] were used. The available character area of the stencil is $1000\mu m \times 1000\mu m$, and the number of character candidate in each benchmark is 1000. We set S to the minimum left/right blank space of the original characters in each benchmark. Recall that [9] and [10] assume that there is only a single shaping aperture and hence a single projection region width (i.e., $K = 1$). For our approach, we tried $K = 1$ and $K = 2$. Table I reports the comparison on total shot count, number of characters put on the stencil, and runtime. It also reports the shot count when using VSB only for reference.

Refer to columns 2, 3, 6, and 9 of Table I, it can be seen that even for $K = 1$ (i.e., all characters must use the same projection region width), our approach can reduce the shot count by $27.472\times$, $3.09\times$ and $1.65\times$ over VSB only, [9] and [10], respectively. The significant improvement over [9] and [10] is partially because they perform only simple blank space sharing while our approach performs flexible blank space sharing. Besides, their algorithms do not attempt to find the optimal projection region width while ours does.

Next, if we use two different sized shaping apertures (i.e., $K = 2$), a huge shot count reduction over using single shaping aperture can be obtained and we already can put all character candidates into the stencil for benchmarks 1D-1 and 1D-2. Our approach with $K = 2$ results in as much as $8.97\times$ and $4.28\times$ shot count reduction compared to [9] and [10], respectively.

For more testing, we generated some harder benchmarks (1D-1h to 1D-4h). We generated 200 extra character candidates into each of the original benchmarks while keeping the same stencil size, so it became impossible to put all characters on a stencil. Table II reports the results of our algorithm with $K = 1$, $K = 2$, and $K = 3$. It also reports the shot count when using VSB only for reference. As expected, the shot count reduction and the number of characters that can be put on the stencil increase with the value of K. And the greatest reduction occurs when K switches from 1 to 2.

We show in Table III the memory requirement of our program. The proposed character grouping technique in Section III.C can reduce the memory requirement of the dynamic program by roughly $40\times$. The resultant memory requirement after adopting the technique is less than 1 GB in each case.

Finally, we note that our algorithm also works for multi-beam direct write system [1], where multiple beam columns

TABLE I
COMPARISON WITH [9] AND [10].

	VSB only	[9]			[10]			Ours ($K = 1$)			Ours ($K = 2$)		
	#shots	#shots	#ch	CPU(s)	#shots	#ch	CPU(s)	#shots	#ch	CPU(s)	#shots	#ch	CPU(s)
1D-1	770543	50809	926	12.13	29536	934	1.91	12972	980	6.90	10418	1000	22.21
1D-2	770543	93465	854	10.24	44544	863	1.74	28594	895	6.41	10418	1000	20.73
1D-3	770543	152376	749	7.85	78704	758	2.35	55761	797	6.59	30785	902	21.13
1D-4	770543	193494	687	6.52	107460	699	2.96	79275	734	7.47	44468	837	29.07
Normalized	27.472	3.090	0.944	1.355	1.650	0.955	0.325	1.000	1.000	1.000	0.570	1.102	3.388

TABLE II
RESULTS ON HARDER BENCHMARKS BY OUR ALGORITHMS FOR DIFFERENT VALUES OF K.

	VSB only	Ours ($K = 1$)			Ours ($K = 2$)			Ours ($K = 3$)		
	#shots	#shots	#ch	CPU(s)	#shots	#ch	CPU(s)	#shots	#ch	CPU(s)
1D-1h	922770	58648	980	8.82	26467	1114	27.22	17534	1163	43.33
1D-2h	922770	86176	905	8.43	48891	1018	25.58	39630	1068	40.67
1D-3h	922770	135332	800	8.63	93109	916	25.69	75709	948	40.89
1D-4h	922770	169105	739	9.76	116219	855	27.02	98204	886	42.91
Normalized	9.680	1.000	1.000	1.000	0.598	1.141	2.966	0.475	1.188	4.718

TABLE III
MEMORY USAGE (GB) WITH AND WITHOUT USING THE MEMORY SAVING TECHNIQUE.

	With memory saving			Without memory saving		
	$K = 1$	$K = 2$	$K = 3$	$K = 1$	$K = 2$	$K = 3$
1D-1	0.136	0.271	0.407	5.236	10.436	15.670
1D-2	0.131	0.262	0.392	5.044	10.087	15.092
1D-3	0.141	0.282	0.422	5.041	10.082	15.087
1D-4	0.156	0.311	0.467	5.205	10.377	15.582
1D-1h	0.136	0.271	0.407	6.282	12.564	18.800
1D-2h	0.131	0.262	0.392	6.051	12.102	18.154
1D-3h	0.141	0.282	0.422	6.048	12.096	18.144
1D-4h	0.156	0.311	0.467	6.245	12.490	18.696

furnished with their own stencils write different regions of a wafer in parallel. As identical dies are typically manufactured on a wafer, the stencil design for all beam columns in a multi-beam system should be the same and is not different from a single beam system.

REFERENCES

[1] T. Maruyama, Y. Machida, and S. Sugatani. CP based EBDW throughput enhancement for 22nm high volume manufacturing. In *Proceedings of SPIE 7637*, February 2010. 7637-1S.

[2] T. Maruyama et al. CP element based design for 14nm node EBDW high volume manufacturing. In *Proceedings of SPIE 8323*, April 2012. 8323-14.

[3] B. J. Lin. Future of multiple-E-beam direct-write systems. In *Proceedings of SPIE 8323*, March 2012.

[4] R. Inanami et al. Maskless lithography: Estimation of the number of shots for each layer in a logic device with character projection-type low-energy electron-beam direct writing system. In *Proceedings of SPIE 5037*, pages 1043–1050, 2003.

[5] H. Pfeiffer. Variable spot shaping for electron-beam lithography. *Journal of Vaccum Sci. and Tech.*, 15(3):887–890, May 1978.

[6] M. Sugihara et al. A character size optimization technique for throughput enhancement of character projection lithography. In *Proc. of ISCAS*, pages 2561–2564, 2006.

[7] A. Fujimura. Design for E-beam: Design insights for direct-write maskless lithography. In *Proceedings of SPIE 7823*, pages 137–140, Sep. 2010.

[8] K. Yoshida et al. Stencil design and method for improving character density for cell projection charged particle beam lithography. US Patent Application No. 2009/0325085 A1, December 2009.

[9] K. Yuan, B. Yu, and D. Z. Pan. E-beam lithography stencil planning and optimization with overlapped characters. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 31(2):167–179, Feb 2012.

[10] B. Yu, K. Yuan, J.-R. Gao, and D.Z. Pan. E-BLOW: e-beam lithography overlapping aware stencil planning for MCC system. In *Proc. of DAC*, 2013.

[11] R. Inanami. Electron beam exposure apparatus, electron beam exposure method and method of manufacturing semiconductor device. US Patent No. 7449700, November 2008.

[12] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, NY, 1979.