Journal of
Micro/Nanolithography,
MEMS, and MOEMS

Nanolithography.SPIEDigitalLibrary.org

Parallel compression/decompression-based datapath architecture for multibeam mask writers

Narendra Chaudhary
Serap A. Savari

SPIE.

Narendra Chaudhary, Serap A. Savari, "Parallel compression/decompression-based datapath architecture for multibeam mask writers," *J. Micro/Nanolith. MEMS MOEMS* **16**(4), 043503 (2017), doi: 10.1117/1.JMM.16.4.043503.

Parallel compression/decompression-based datapath architecture for multibeam mask writers

Narendra Chaudhary* **and Serap A. Savari**
Texas A&M University, College Station, Texas, United States

Abstract. Multibeam electron beam systems will be used in the future for mask writing and for complementary lithography. The major challenges of the multibeam systems are in meeting throughput requirements and in handling the large data volumes associated with writing grayscale data on the wafer. In terms of future communications and computational requirements, Amdahl's law suggests that a simple increase of computation power and parallelism may not be a sustainable solution. We propose a parallel data compression algorithm to exploit the sparsity of mask data and a grayscale video-like representation of data. To improve the communication and computational efficiency of these systems at the write time, we propose an alternate datapath architecture partly motivated by multibeam direct-write lithography and partly motivated by the circuit testing literature, where parallel decompression reduces clock cycles. We explain a deflection plate architecture inspired by NuFlare Technology's multibeam mask writing system and how our datapath architecture can be easily added to it to improve performance. [DOI: 10.1117/1.JMM.16.4.043503]

Keywords: multibeam mask writer; data compression; datapath, architecture; algorithms.

Paper 17112P received Aug. 2, 2017; accepted for publication Oct. 17, 2017; published online Nov. 7, 2017.

1 Introduction

The trend of doubling the number of transistors per unit area on integrated circuits according to Moore's law is slowing down. IEEE and AIP recently published a special issue of a journal devoted to the end of Moore's law.[1] The high cost of fabricating a mask set is a contributing factor to the slowdown, and the high cost is partly due to the increase in the number of mask layers and the decreasing throughput of mask writers. The increase in the number of mask layers is mainly due to the multiple patterning requirement at new nodes, and variable-shaped beam mask writers are unable to fulfill the throughput requirements of sub-10 nm nodes. The aperture array-based multibeam mask writers introduced by IMS Nanofabrication[2] and NuFlare Technology[3,4] have been proposed to improve mask write times. These multibeam systems use a large number of smaller beams to write patterns for the mask layers. Multibeam systems use grayscale pixel-based writing where each beam provides a dose according to digital data. This dose is transferred to a single pixel or number of pixels.[5,6] The multibeam systems also face a throughput bottleneck described by Tennant's law[7] (1999). Let areal throughput be the area of a wafer that can be printed per unit time using direct-write-like lithography technologies. Then

$$\text{Areal throughput} \propto \text{resolution}^5.$$

Besides the pixel throughput problem, multibeam systems also have to process large amounts of data and communicate grayscale data at high data rates. A recent paper[4] by NuFlare discusses the difficulties associated with data processing and communication. These data processing and communication requirements will continue increasing in the future in proportion to the square of the resolution. The current datapath architecture of multibeam systems shown in Fig. 1(a) is ill equipped to handle these requirements. In the current datapath architecture, the GDSII or OASIS formatted data are converted into a tool-specific format and transferred to the mask writing tool. Data processing is next performed online to do rasterization, proximity effect corrections, and other corrections that result in pixel data and beam deflection data. The data processing capability and datapath communication capacity affect the efficiency of this kind of datapath. IMS and NuFlare have had some success in addressing throughput requirements by parallel processing[3,4] using graphical processing units and by increasing the communication capacity[2,4] of the datapath. However, these approaches have limitations. In computer architecture, Amdahl's law[8] (1967) states that the speedup or throughput gain for the execution of a task with increased resources is always limited by the fraction of the task that cannot benefit from the improvement. In the case of parallel processing, the throughput gain is limited by the part that cannot be parallelized. Let Speedup denote the throughput gain, $\text{Fraction}_{\text{enhanced}}$ be the proportion of the task benefiting from improvements, and $\text{Speedup}_{\text{enhanced}}$ be the speedup in $\text{Fraction}_{\text{enhanced}}$ by parallelism or other improvements. Then

$$\text{Speedup} = \frac{1}{(1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}.$$

Observe from Table 1 that an increase in datapath speed by a factor of 10 improved the mask writing time only by a factor of 1.5. Part of the throughput bottleneck can be attributed to the time needed to write the pixel data. Reference 4 also discusses a data transfer bottleneck arising from the communication to the blanking aperture array. Our objective

*Address all correspondence to: Narendra Chaudhary, E-mail: narendra5@tamu.edu

1932-5150/2017/$25.00 © 2017 SPIE

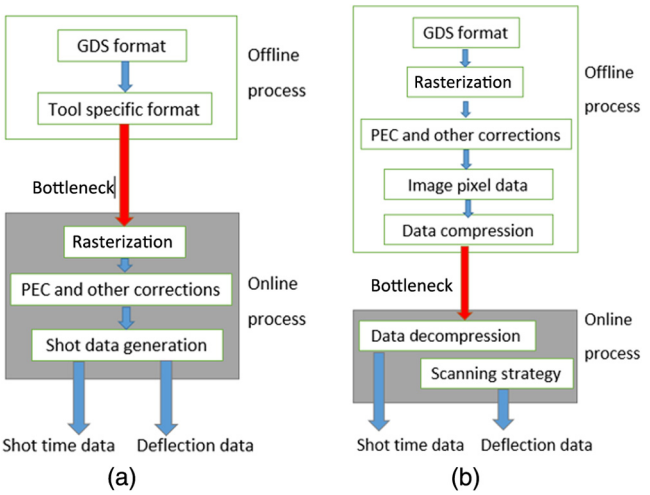

Fig. 1 (a) Current datapath architecture. (b) Proposed datapath architecture.

Table 1 The effect of an increase in datapath resources in recent IMS mask writing tools.[2]

Tools	Datapath	Mask write time 100 mm × 130 mm
IMS MBMW-101 Beta	12G	15 h/mask
IMS MBMW-101 HVM	120G	10 h/mask

is to improve the communication and computational efficiency of the aperture array-based multibeam mask writers. For direct-write lithography systems, there have been multiple proposals of datapaths in which a major component of the data processing and the generation of pixel-based data is performed offline (see, e.g., Refs. 6 and 9–14). Dai and Zakhor[9,10] initiated the study of the communication and computation efficiency within datapaths by considering the hardware constraints of some architectures, and they proposed the Block C4 layout image compression algorithm. Yang and Savari[11,12] proposed the Corner2 algorithm and custom run-length encoding schemes to reduce the hardware complexity and improve the decoding speeds, and Chaudhary et al.[13] modified this algorithm for proximity-corrected grayscale data. Carroll et al.[14] implemented a variation of a Lempel-Ziv[15,16] data compression algorithm in the reflective electron beam lithography system.[17] We began to consider the impact of parallelism on aperture array multibeam mask writers in Refs. 5 and 6. Our inspiration in those papers was the IMS mask writer. While those papers proposed the use of multiple simple run-length decoders, they did not examine how to coordinate the operation of the decoders to meet existing device constraints. Furthermore, the data transfer to and the data fan-out at the control circuitry of the blanking aperture array use metal transmission lines operating at low clock frequencies,[4] and earlier papers did not consider the number of clock cycles in the communication of data. In this work, we introduce a datapath architecture [see Fig. 1(b)] where the parallel decompression portion is inspired by the very large scale integrated circuit testing literature;[18] this architecture addresses the synchronization requirement among the decoders and shows how to reduce the number of clock cycles needed to transfer typical mask data. We also recommend that the data be compressed by custom-designed parallel compression algorithms to decrease the processing time, memory, and file sizes. In addition to data decompression, our proposed online processing includes the generation of deflection data from a predefined scanning strategy [see Fig. 1(b)].

The remainder of the paper is organized as follows. In Sec. 2, we present the details of the existing multibeam blanker system. In Sec. 3, we describe the scanning strategy and the compression constraints. In Sec. 4, we describe the compression scheme and the decompression architecture. In Sec. 5, we discuss the experimental results, and we conclude the paper in Sec. 6.

2 Multibeam Blanker System

The major component in both the NuFlare Technology[3,4,19] and the IMS Nanofabrication[2,20–22] multibeam mask writing systems is the blanker. The multibeam blanker system we consider is motivated by the 2016 NuFlare system.[4,19] We focus on the NuFlare system in this paper because NuFlare disclosed more information than IMS about their circuit design and data communication system and because we discussed certain aspects of the IMS system in Refs. 5 and 6. We follow NuFlare's choice of 10-bit representations as opposed to IMS' 4-bit representation with beam overlaps and point out that it is not straightforward to compare these representations since IMS uses 5 nm × 5 nm pixels and NuFlare uses 10 nm × 10 nm pixels. Furthermore, since NuFlare Technology has not disclosed all the details about its multibeam mask writing systems, we make various assumptions to explain our understanding of the multibeam blanker system; for example, we created Figs. 2–5. The multibeam system consists of two plates: namely, an aperture plate and a deflection plate.[2,3,19–22] The aperture plate consists of an array of apertures that converts a broad beam into multiple beams.

The deflection plate consists of an equal number of apertures at the same coordinates as the aperture plate. The deflection plate also contains a pair of electrodes at each aperture, providing enough deflection voltage to deflect the beam passing through the aperture.[19] A grayscale dose control for each beam is provided by controlling the deflection voltage for a discrete amount of time by means of a control circuit for each beam.[19] The deflection control circuit of each beam is connected by a bus to the other control circuits to enable the communication of data within a row.[19] Each row's data is provided by a "side pad" digital circuit, which could be a field programmable gate array (FPGA) or an application-specific integrated circuit (ASIC). The deflection plate is a stand-alone device or a chip packaged separately from the side pad. Figure 2 shows the deflection plate design for an array with 16 beam apertures with four rows and four columns.

Figure 2 shows a control circuit associated with each individual beam. The control circuitry of each beam consists of two parts, namely, a digital beam logic for the communication of 10-bit dose data with the ability to generate 1024 discrete time intervals and an analog circuit (mainly a differential amplifier) to drive the electrodes for the amount of time specified by the digital beam logic.[19] We do not

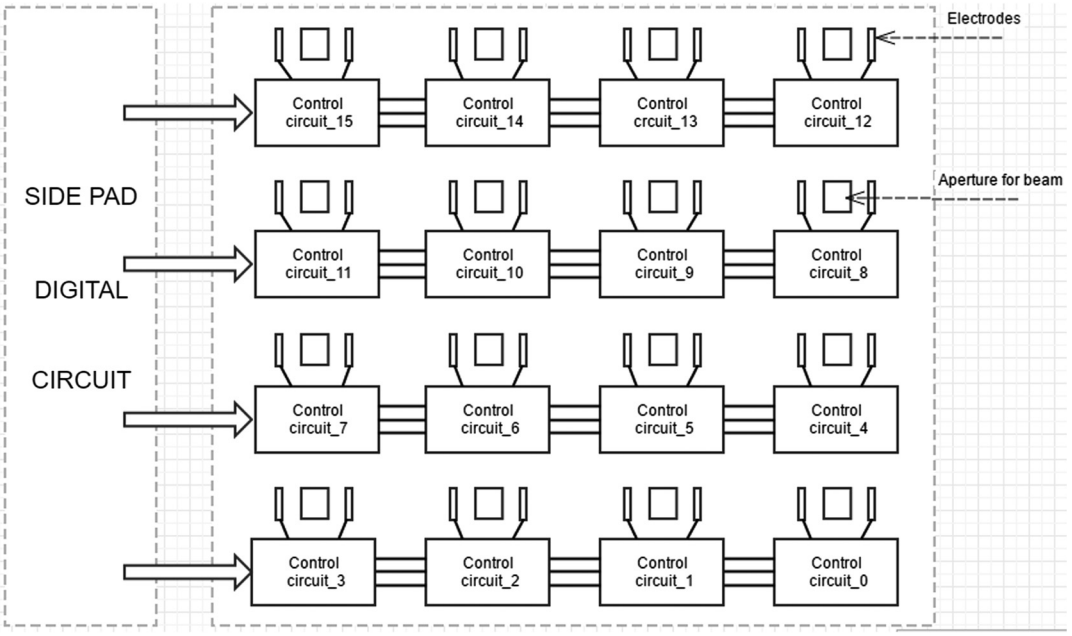

Fig. 2 Deflection plate of a 4×4 beam array with side pad circuitry. Each beam has a control circuit connected to electrodes. The electrodes deflect the beams.

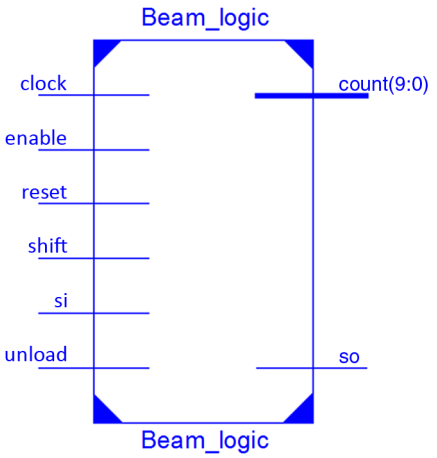

Fig. 3 Digital beam logic associated with the control circuit of each beam.

consider the analog design of the amplifier circuit in this paper and focus instead on the digital logic needed for each beam. Figures 3 and 4 show one implementation of the beam logic. The digital beam logic consists of a shift register with a one-bit serial input to communicate 10-bit data and a 10-bit countdown and "stop at 0" counter to provide 1024 time intervals. Registers[23] are digital logic circuits that store data in the form of binary numbers. Shift registers[23] are the registers that load the input bit into the most significant bit or least significant bit of the register value and "shift" the stored binary number by one position. The counter starts from the 10-bit input dose value as the initial value, and the value decreases by 1 at every clock cycle until the counter value reaches zero, at which point the counter stops.

The beam logics require the supply voltage (VDD) and the ground (GND) as inputs. Besides the supply input (VDD and GND), a beam logic requires a clock signal input, a reset signal input, an unload signal input to load the counter with the shift register value as the initial value, and an enable (trigger) input signal to start the counter. These input signals should be synchronous and broadcast to all the control circuits of the beams, as simultaneous control of all the beams requires that all the counters start at the same clock cycle. Here, a synchronous signal means the input signal (reset, unload, and enable) values are sampled only at the positive edge of the clock signal. Let *si* and *so* denote the serial input and the serial output to the shift register, respectively; they can be alternately implemented with a 10-bit parallel input and a 10-bit parallel output.[19,24] An extra digital input shift signal is also provided to control the shift register, i.e., the data are moved in/out of the shift register only when the shift signal is high (1). The shift signal is common to all the control circuits in a row of the array.

As shown in Fig. 2, the beams are organized in the shape of a grid with rows and columns. We connect the control circuits of a row of beams in a beam array for the communication of the 10-bit dose data. The serial/parallel output of one control circuit is connected to the serial/parallel input of the next control circuit.[19,24] This allows the data to be shifted within a chain of beam logics. Figure 5 illustrates the connection of four beam logics in a row. This type of communication system removes the need for addressing each 10-bit dose value to a control logic. Let N be the number of beams in a row. When no compression is used, $N * 10$ clock cycles are required to transfer the 10-bit data to each beam logic in a row. The clock, shift, unload, reset, and enable are all common input signals to each beam logic block in a row. Each beam logic also has a separate 10-bit output count, which is the counter value and serves as the timer for the beam deflection.

The beam array consists of multiple rows of beams in parallel.[19,24] The clock, enable, unload, and reset signals are common to all the rows in the beam array. Here, all the beam logic blocks receive the same set of signals to start all the counters at a given time instance.[19] The shift signals and

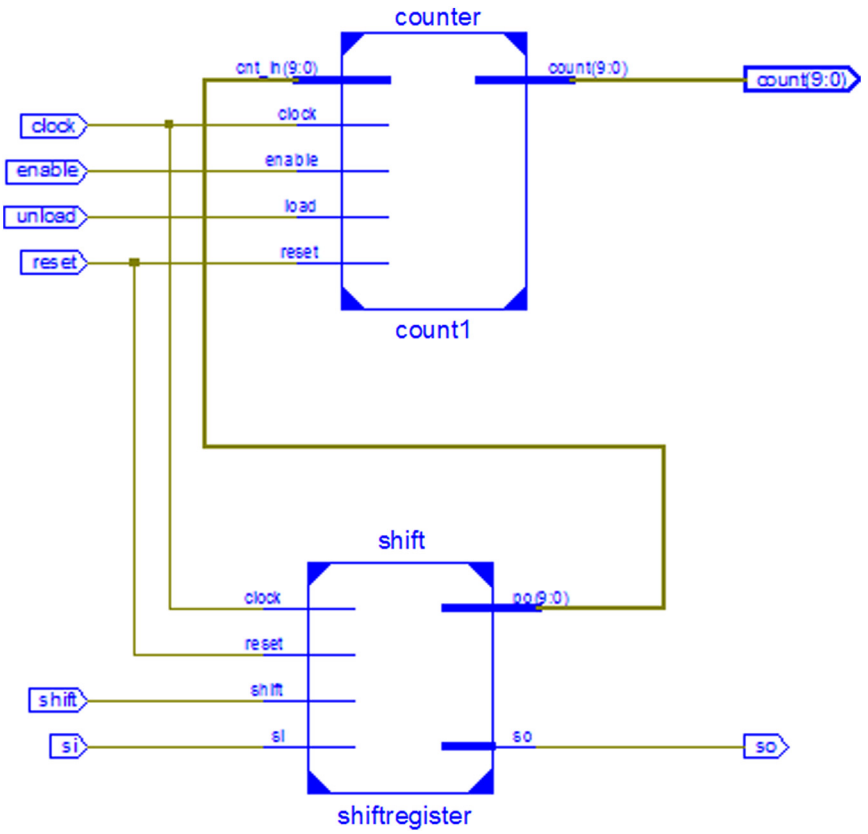

Fig. 4 Expanded view of the digital beam logic associated with the control circuit of each beam.

the serial inputs for the data are different for each row of beams. Separate serial inputs and shift signals for each row of beams are required for a data transfer protocol without addressing.

3 Scanning Strategy and Compression Constraints

In earlier work, we proposed a zigzag scanning strategy and a writing in stripes motivated by the IMS multibeam mask writer.[5] We believe the zigzag scanning strategy requires less deflection of the multibeam array and covers the entire scanning area in fewer steps. A reduction in the deflection of the beam array could be desirable to reduce the settling time and increase the frequency of the beam array deflection. Furthermore, the use of a scanning strategy with predefined steps would help in reducing the beam array deflection data because it could be calculated in real time similar to raster scanning. It differs from those for the vector scanning strategies used in variable-shaped beam systems, where the coordinates of positions must be specified.

The IMS multibeam system uses a beam size of 20 or 10 nm with a 4-bit dose control for each beam. It also has 5 nm × 5 nm pixels with beam overlaps to provide 241 or 61 gray levels for the 5 nm × 5 nm pixels depending on the beam size. The NuFlare multibeam system has a 10-nm beam size and 10 nm × 10 nm pixels. Each beam has a 10-bit dose control. In contrast with the IMS system, the beams do not overlap on the pixel grid to create the higher dose levels. Therefore, each 10 nm × 10 nm pixel has 1024 gray levels assuming single pass writing. For both systems, the distance between the individual beams in the X- and Y-directions is 160 nm, i.e., 16 pixels in the NuFlare multibeam system and 32 pixels in the IMS multibeam system. Since the NuFlare system has no beam overlaps, its horizontal and vertical movement step sizes will differ from the zigzag scanning strategy of the IMS system. In Table 2, we propose a zigzag strategy for our family of multibeam arrays inspired by the NuFlare multibeam system.

For any choice of scanning strategy, after writing one set of beam shots the entire beam array moves to a new position. This imposes a synchronization constraint on the beam array as the data for all the beams should be available before the beam array moves to a new position. If we have a 512 × 512 beam array, then the entire beam array would require 2,621,440 bits at one position as each beam needs 10-bit data and there are 262,144 beams. The next set of 2,621,440 bits of 262,144 beams will be needed when the beam array moves to a new position. We can consider the data of each set of 262,144 beams as a frame of a grayscale "video" with dimension 512 × 512. Any compression scheme and decompression architecture should operate with these parameters. The decompression scheme should be able to decode the frames in their given sequence. The data inside the frames may be compressed in a variety of ways as long as the synchronization constraint is satisfied. The "video" analogy and constraints also allow the separate and parallel compression of each frame as long as the sequence of compressed frames does not change. Thousands of frames can be potentially compressed in parallel to reduce the compression time. However, there are some differences between this communication problem and traditional "video" communication; the latter application can take advantage of inter/intraframe dependencies. For our problem, there are weaker

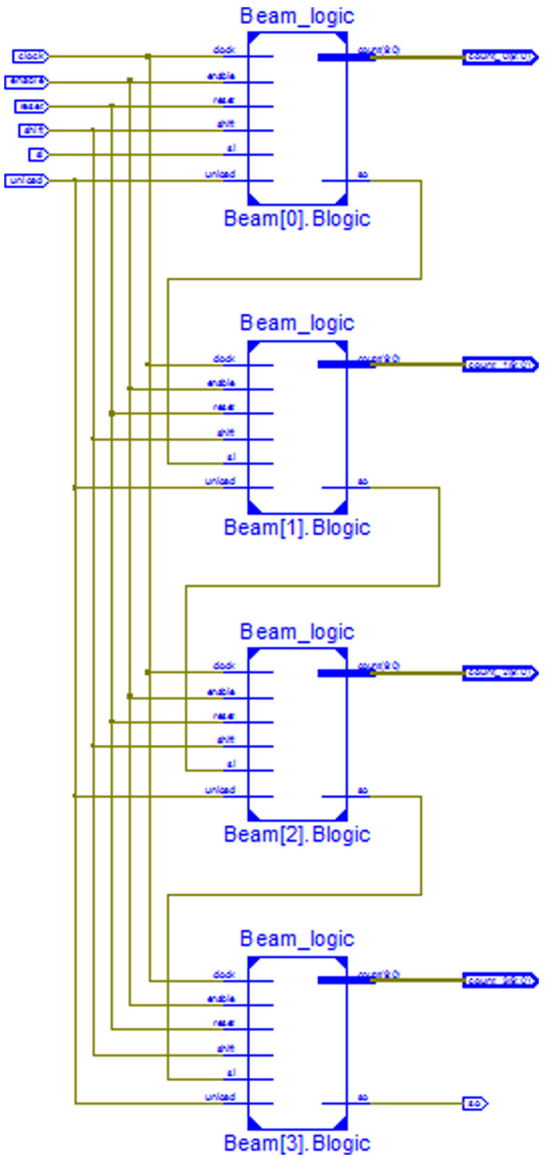

Fig. 5 Row of beam logics connected in a chain by the si inputs and the so outputs.

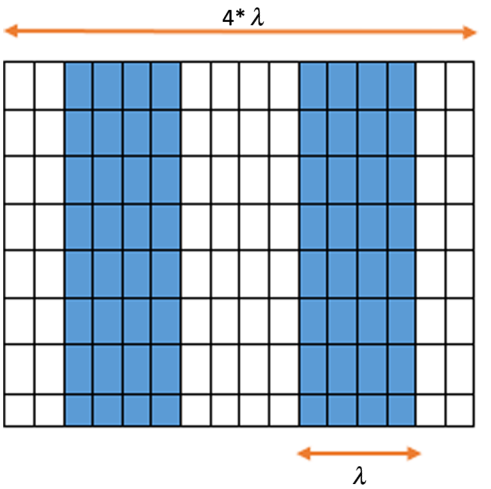

Fig. 6 50% sparsity with simple design rules. λ is equal to the half pitch.

intraframe dependencies in the data because of the large distance between the beams in a beam array; for example, in a 512×511 beam array, the distance between successive beams in either the X- or Y-direction is 16 pixels. The interframe dependencies are mainly governed by the choice of scanning strategy. Our scanning strategy is described in Table 2. Observe that the distance a beam moves from one frame to the next in the X-direction is large, so there does not appear to be any appreciable interframe dependencies for our scanning strategy. However, a scanning strategy that more closely resembles raster scanning might provide interframe dependencies.

Simple design rules[25] require a large fraction of the pixels in a typical mask layer to have pixel values of zero; i.e., the data are sparse. Figure 6 shows a pattern without correction in which 50% of the pixel values are zeros. Complementary lithography and multiple patterning further contribute to the prevalence of sparse data, and our data compression scheme is designed to take advantage of the existing sparsity within mask data.

Table 2 Overview of scanning strategy for a family of beam arrays.

Array	Number of beams	Distance in pixels between the centers of two neighboring beams in the same row or column	Horizontal movement of array from one writing to the next within a "stripe"	Vertical movement of array from one writing to the next within a "stripe"
$2^N \times (2^N - 1)$ N odd	$2^N(2^N - 1)$	$d = \sqrt{2^{N-1}}$	$2d$ or $(2d - 1)$ pixels depending on the iteration	$+1$ or -1 pixels depending on position of array within zigzag
8×7	56	2	$+3$ every second iteration, $+4$ pixels for all other iterations	Progresses in the sequence $n, n+1, n+2, n+1, n, n+1, \ldots$ for some n
32×31	992	4	$+7$ every fourth iteration, $+8$ pixels for all other iterations	Progresses in the sequence $n, n+1, \ldots, n+4, n+3, \ldots, n+1, n, n+1, n+2, \ldots$ for some n
128×127	16,256	8	$+15$ every eighth iteration, $+16$ pixels for all other iterations	Progresses in the sequence $n, n+1, \ldots, n+8, n+7, \ldots, n+1, n, n+1, n+2, \ldots$ for some n
512×511	261,632	16	$+31$ every 16th iteration, $+32$ pixels for all other iterations	Progresses in the sequence $n, n+1, \ldots, n+16, n+15, \ldots, n+1, n, n+1, n+2, \ldots$ for some n

4 Compression Scheme and Decompression Architecture

We first discuss a simple run-length code for data compression and explain the decompression architecture for it. We will later propose a more effective scheme. As we mentioned earlier, the majority of 10-bit symbols are zeros. In our problem, we choose to represent a 10-bit zero symbol by a single-bit "1" and to encode the 10-bit nonzero symbols by the 11-bit string which concatenates the prefix "0" to the original 10-bit string in order to reduce data volumes and overall clock cycles. For example, two encodings of 10-bit symbols are illustrated below.

$$1101011010 \text{------} > \underline{0}1101011010,$$

$$0000000000 \text{------} > \underline{1}.$$

Observe that the maximum possible compression ratio of this simple scheme is 10, which occurs when all symbols are zero. The worst possible compression ratio is $10/11 = 0.91$, which occurs when all symbols are nonzero.

The data for multiple beams are arranged in a column by column fashion. If we take Fig. 2 as a reference in the numbering convention of a beam array, then the data are arranged in a "frame" in the following sequence. Let U_i be the symbol or beam dose associated with beam number i. For a beam array with 16 beams organized in four rows and four columns, U_0 is the first symbol in a "frame" and U_{15} is the last symbol. After this data are written and the beam array moves, a new "frame" starts. A similar strategy is applied to any size of beam array. The sequence of uncompressed symbols is given below.

$$U_0, U_4, U_8, U_{12}, U_1, U_5, U_9, U_{13}, U_2, U_6, U_{10}, U_{14}, U_3, U_7, U_{11}, U_{15}.$$

To compare different communication schemes, we compare the number of clock cycles they use on the hardware. We first look at the number of clock cycles for uncompressed data transmission. The serial communication of n 10-bit symbols $U_0, \cdots, U_{n-1}$ to n beams with a single bit wire will use $n * 10$ clock cycles. One could alternatively use $n * 10$ bit parallel communication wires to communicate data to all the beams, which should ideally take 1 clock cycle. The parallel communication will not always be helpful as $n * 10$ bits must be collected from a capacity constrained channel and stored before they are transmitted to the n beams. The process of collection and storage of data from a serial source may offset the advantage of parallel communication, and there will be a significant and undesirable increase in the hardware on the deflection plate. There can be many degrees of parallelism between these two extremes, but for uncompressed data transfer all of them have implementation trade-offs.

We know from the previous discussion that the compressed symbol C_i corresponding to an uncompressed symbol U_i will consist of either a single bit string or an 11-bit string.

$$C_i = \begin{cases} 1, & \text{if } U_i = 0000000000 \\ \mathbf{0}U_i & \text{if } U_i \neq 0000000000 \end{cases}. \quad (1)$$

The compressed string is communicated to the side pad digital circuit; after that, it needs to be decompressed and transferred to the individual beams. An effective communication strategy should not only compress the data but should also reduce clock cycles by means of data decompression.

The compressed string $C_0, \cdots, C_{n-1}$ can be decoded in multiple ways. We first look at the case when a single decoder is used to decompress the data as shown in Fig. 7(a). The function of the decoder is to collect the C_i symbols and generate the U_i symbols. When the input C_i symbol is 11 bits long, the transmission through a one-bit wire to the decoder will take 11 clock cycles and the U_i symbol can be generated in the last 10 of those 11 clock cycles. When the input C_i symbol is one-bit long, the transmission takes only one cycle and the generation of the U_i takes the next 10 cycles. Hence, a single decoder always takes 11 clock cycles to generate each U_i from any C_i symbol. This shows that a single decoder decreases throughput by taking 11 clock cycles instead of the 10 needed in the uncompressed case.

In the second case, we have two decoders as shown in Fig. 7(b). We pass every other C_i to each decoder. Since the C_i symbols are of variable lengths, we need some additional hardware to detect the symbol length. The detector in Fig. 7(b) detects the length of C_i by checking the prefix and directs the entire symbol to the corresponding decoder. Consider the special case of two symbols $C_1 = 1$ and $C_2 = 1$; they can be decompressed in parallel as shown in Fig. 7(c). The parallel decompression would only take 13 cycles, i.e., 2 cycles for the communication of the compressed symbols to the decoders and 11 cycles for parallel decompression. Two decoders provide the opportunity for the parallel decompression of $C_i = 1$ symbols. Any other combination of symbols will not reduce the number of clock cycles. We also assume that the number of decoders is known in advance.

We can extend and generalize this approach to multiple decoders. Multiple decoders provide increased opportunity of parallel decompression of $C_i = 1$ symbols. This approach should increase the parallelism and communication efficiency for the uncompressed sparse data at the cost of increased hardware complexity. We can use the set of k decoders to decompress and transfer data to k nodes as shown in Fig. 7(d). The compressed symbols C_i of length one bit can again be decompressed in parallel, while the C_i of length 11 bit are decompressed serially. The C_i of 11 bit length take 11 cycles, and the C_i of one bit take one cycle plus 10 cycles every column because of parallel decompression. The simple compression scheme is summarized in Table 3.

Figure 8 shows the logical architecture of the parallel decompression combined with the existing deflection plate architecture. We assume that the deflection plate in Fig. 8 consists of four columns and k rows. The set of k decoders are connected to the head column of the beam logic, and they transmit the $U_1, \cdots, U_k$ symbols to the deflection plate. The decompressed symbols are shifted further to their desired beam logic by the chain of shift registers.

If the sole goal of the data compression scheme was the effective communication of data from the storage device to the side pad, then a more complex compression scheme would better serve the purpose. Data compression is

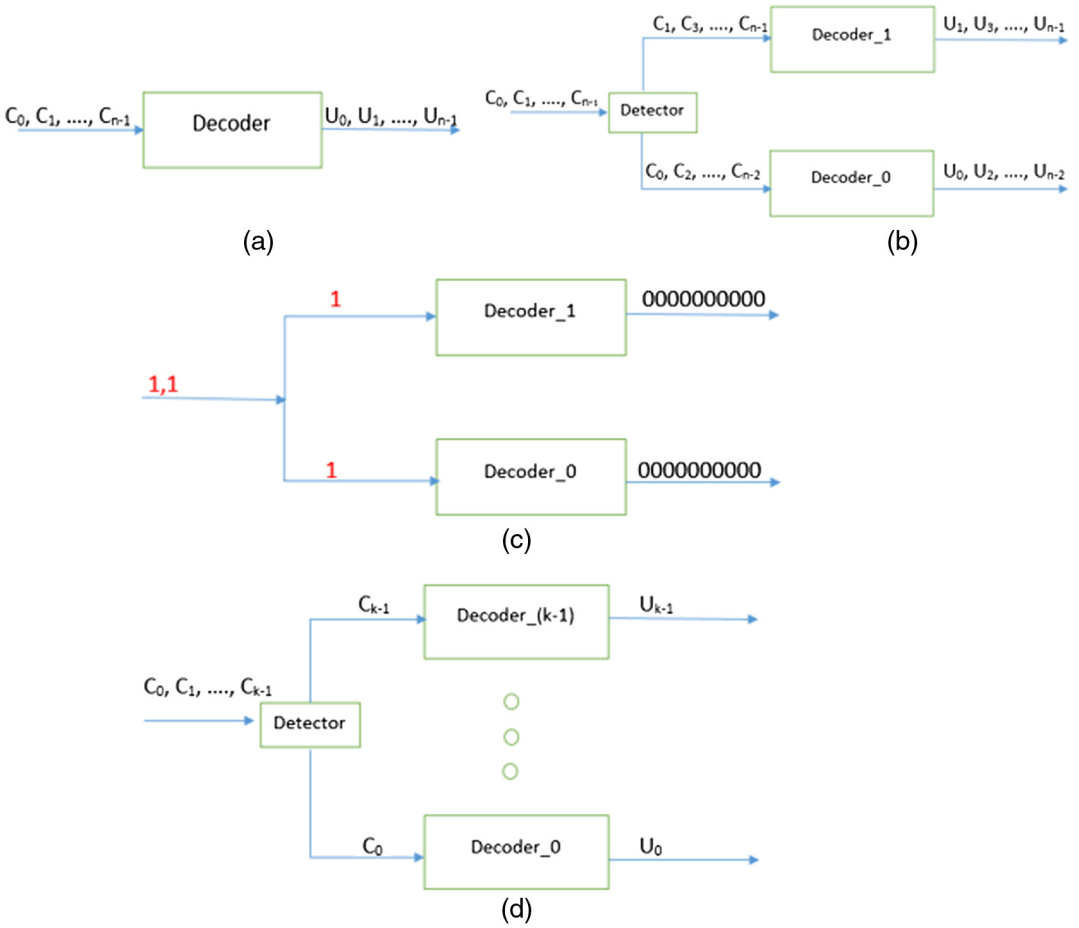

Fig. 7 Multiple ways of decoding. (a) single decoder, (b) two decoders, (c) two decoders working in parallel when both compressed symbols are "1," and (d) *k* decoders for data decompression.

Table 3 Simple compression scheme and clock cycles for decompression.

Symbol	Explanation	Prefix	Suffix	Cycles to decode	Cycles for uncompressed data transfer
Z	0000000000	1	—	1+ (10 every column)	10
NZ	10-bit value	0	10-bit value	11	10

typically used to reduce the data transmission bottleneck from capacity constrained channels. The data communication from a large storage device to the side pad digital circuit of an FPGA or ASIC is also capacity constrained. Our simple compression scheme will mitigate the data communication problem. We next extend the compression scheme as shown in Table 4 to improve the compression ratio and decrease clock cycles. We introduce two extra symbols, namely, an all zero frame (AZF) with prefix 001 and an all zero column (AZC) with prefix 000. The AZF signifies that all the pixel values in the video frame have a zero value, which occurs when the $M * N$ beam shots all have a zero dose value. In this case, the shot data does not need to be communicated to the deflection plate as the array should simply move to the next position. In uncompressed data transfer, $M * N * 10$ clock cycles are needed, but the AZF symbol instructs the beam array to move to the next position. The AZF symbol can be detected in three clock cycles. The AZC symbol signifies that all the pixels in a column of a frame have a zero value. In this case, all the zero pixels can be generated in parallel according to our previous discussion of the parallel architecture. It takes three clock cycles to decode the AZC symbol and $M + 10$ clock cycles for the parallel generation of zero symbols for a column using the parallel architecture. The zero symbol (Z) has the same representation and processing as in the simple compression scheme. The nonzero symbol (NZ) now uses 12 bits with a prefix of 01 and 12 cycles to decode. A similar parallel architecture with minor changes can be applied to the extended compression scheme.

The compression of data should happen offline as the compression algorithm for the extended scheme can run on thousands of processors. According to our "video" analogy, each frame can be compressed on a separate thread of execution. Since there is no dependency between the frames in this compression algorithm, the parallel compression

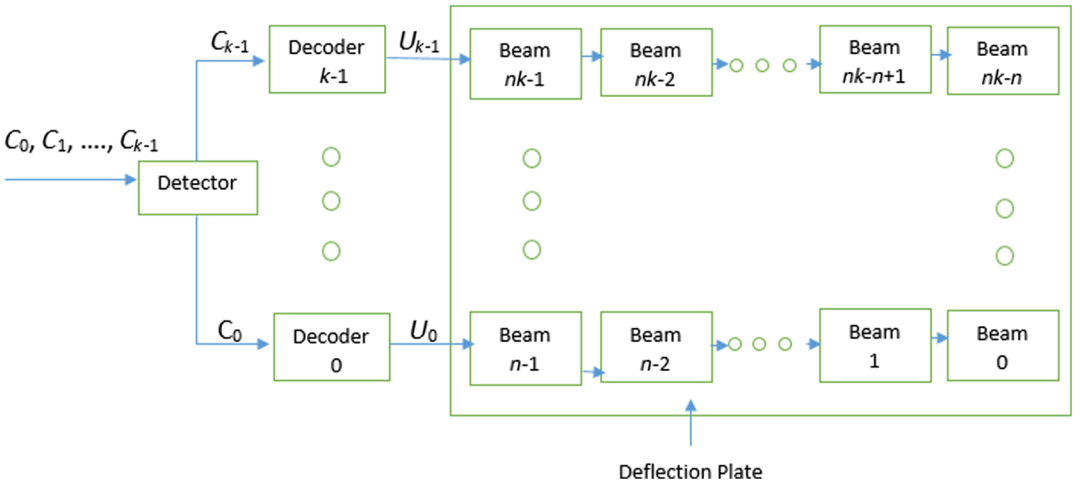

Fig. 8 Logical decompression architecture with the deflection plate.

Table 4 Extended compression scheme for M (rows) $\times$ N (columns) array.

Symbol	Explanation	Prefix	Suffix	Cycles to decode	Cycles for uncompressed data transfer
AZF	all zero frame	001	—	3	$M * N * 10$
AZC	all zero column	000	—	$M + 10 + 3$	$M * 10$
Z	0000000000	1	—	1+ (10 every column)	10
NZ	10-bit value	01	10-bit value	12	10

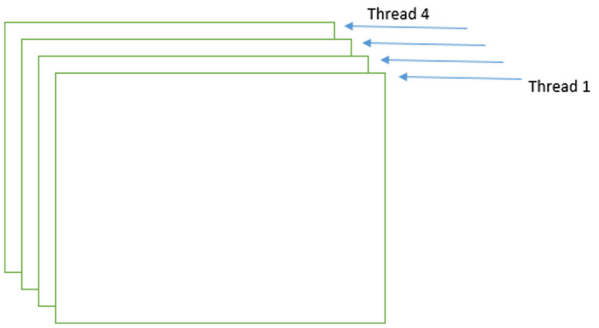

Fig. 9 Compression of four frames in parallel on four different threads of execution.

of data should provide nearly linear speedup (execution time inversely proportional to the number of processors). Figure 9 shows the compression of four frames in parallel on four different threads of execution.

5 Experiments and Results

We performed our experiments on two sets of data using the extended compression scheme. One image is an inverse lithography technology (ILT) mask motif pattern of contact holes with a smallest element of 80 nm. This pattern was enlarged by repeating it to generate a layout image of dimension 30,017 × 33,300 pixels with each pixel having dimensions of 10 nm × 10 nm. We also produce results for 23 layers of an image compression block (ICB) based on the FREEPDK45 45 nm library with a smallest element of 60 nm. The image size for the ICB data is 91,209 × 90,970 pixels with each pixel having dimensions of 10 nm × 10 nm. We used the electron beam proximity effect correction algorithm of GenISys, Inc., BEAMER_v4.6.2_x64, to generate proximity-corrected images with 256 shot dose levels. Each pixel shot dose level was multiplied by four to create 1024 dose levels. This does not change the nature of the data for our compression algorithm as the nonzero/zero doses remain nonzero/zero.

The images were divided into stripes and frames according to the scanning strategy of Table 2. Each frame was subsequently compressed according to the extended compression scheme. Two separate implementations of algorithms were done. In one implementation, the frame generation and the data compression were done serially. In the other implementation, the frame generation and the data compression were done in parallel. The implementations of the frame creation and compression algorithms are in C++, and the parallelization was done by the OpenMP library. The computation of the number of cycles to decode was implemented in C++. The experiments were performed on Intel i7-2600 CPU processors at 3.40 GHz with 8 GB of RAM on a Windows7 Enterprise operating system. The processor contains four cores.

Table 5 shows the results for the compression ratios and the speedups. The compression speedup reported is the ratio of the execution times of the serial and parallel compression algorithms without write to memory. The decompression speedup reported is the ratio of the computed uncompressed transfer cycles to the computed decode cycles. The

Table 5 Data compression ratios and speedups with parallel decompression architecture and parallel compression speedup.

Data type and beam array ($M \times N$)	Uncompressed (MB)	Compressed (MB)	Compression ratio	Decompression speedup	Compression speedup (8 threads)
23 layers ICB (128×127)	230,471.5	3141.4	73.4	24.9	3.2
23 layers ICB (512×511)	267,885.6	4372.3	61.3	22.4	2.4
2 ILT layers (128×127)	2500.4	262.6	9.5	6.6	3.1
2 ILT layers (512×511)	3730.2	309.1	12.1	7.6	2.0

uncompressed data consist of the original image data and the extra zero padding for the frames at the edge of the image. The definition of the compression ratio is as follows:

$$\text{Uncompressed data} = \text{Original data} + \text{Zero padding data for edge frames}, \tag{2}$$

$$\text{Compression ratio} = \frac{\text{Uncompressed data size}}{\text{Compressed data size}}. \tag{3}$$

Figure 10 shows the plot of the compression speedup with respect to the number of threads. We can see that the compression speedup increases with more threads. It is not a linear speedup in the number of threads probably due to the frame generation part of the algorithm and the memory accesses. The speedup of a single thread is not exactly equal to one as the serial and parallel codes were different. The compression ratio and decompression throughput are higher for the ICB data as that data are more sparse. In the ILT case, the throughput and compression ratios are higher for the 512×511 beam array compared with the 128×127 beam array while the reverse occurs for the ICB data. Observe that the 512×511 beam array has a larger frame size and column size than the 128×127 beam array, and the probabilities of the AZF and AZC symbols are lower for a larger beam array when the sparsity is lower. The nonuniformity in decompression clock cycles per frame could be a concern for some datapath systems. We report the clock cycles per frame results in Figs. 11 and 12. Figure 11 shows the histogram of clock cycles that every frame takes for layer 1 and layer 2 of the ILT data with a 128×127 beam array. Figure 12 shows the plot of clock cycles taken by each frame in one stripe of layer 2 of the ILT data with a 128×127 beam array. Layer 1 has 99.6% sparsity and layer 2 has 92.2% sparsity. We can see that even for layer 2 the maximum number of clock cycles taken by any frame stays well below the maximum possible cycles in the uncompressed data communication case ($128 \times 127 \times 10$

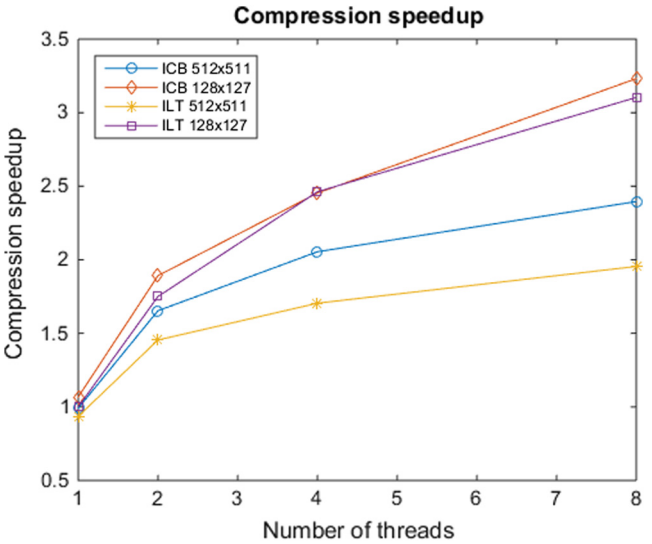

Fig. 10 Compression speedup with respect to the number of threads in the quadcore machine.

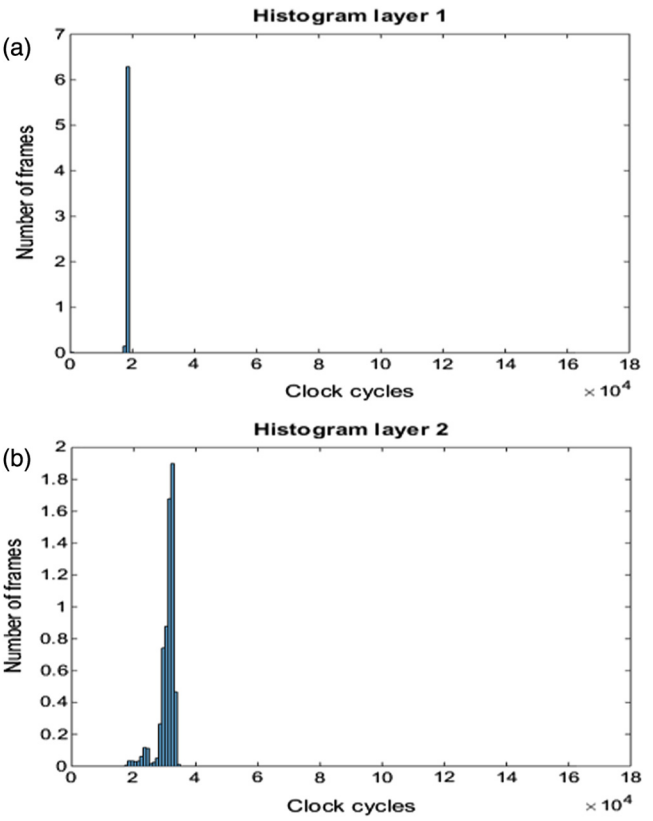

Fig. 11 Histograms of clock cycles in the ILT data with a 128×127 beam array. (a) Layer 1. (b) Layer 2.

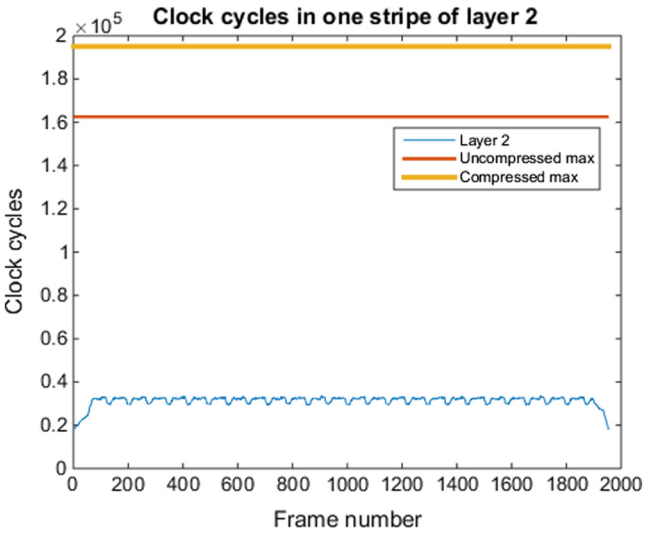

Fig. 12 Clock cycles taken by the frames in one stripe of layer 2 of the ILT data with a 128×127 beam array.

cycles) or the compressed data communication case ($128 \times 127 \times 12$ cycles).

6 Conclusion and Future Work

To try to improve the communication and computational efficiency of multibeam mask writing systems, we propose a datapath architecture that uses parallel data compression and decompression and is inspired by multibeam direct-write tools and circuit testing. Our parallel compression algorithm will help to address two important problems in mask writing—namely, data volume and data preparation time. Our decompression architecture can be attached to the existing deflection plate architecture. We also show that video- and pixel-based representations are effective file formats for multibeam systems. In the future, we would like to implement the parallel architecture in hardware. We would also like to evaluate a recent scheme[24] proposed by NuFlare Technology offering a new circuit design and other architecture changes.

Acknowledgments

This work was supported in part by the National Science Foundation grant ECCS-1201994. This work would not have been possible without the support of GenISys, Inc., which was provided through an Agreement of Cooperation between GenISys, Inc. and Texas A&M Engineering Experiment Station at College Station. The authors also thank N. Hayashi of Dai Nippon Printing for providing an ILT test layout image and D. Lie and S. Mukhopadhyay for providing the ICB. Finally, the authors are grateful to S. Yoshitake, T. Groves, J. Yang, and S. Khatri for helpful conversations and correspondences. The ICB data and decompression demonstration video can be found at http://people.tamu.edu/~narendra5/

References

1. *Comput. Sci. Eng.* **19**(2), 1 (2017).
2. C. Klein and E. Platzgummer, "MBMW-101: world's 1st high-throughput multi-beam mask writer," *Proc. SPIE* **9985**, 998505 (2016).
3. H. Matsumoto et al., "Multi-beam mask writer MBM-1000 and its application field," *Proc. SPIE* **9984**, 998405 (2016).
4. H. Matsumoto et al., "Multi-beam mask writer MBM-1000," *Proc. SPIE* **10454**, 104540E (2017).
5. N. Chaudhary, Y. Luo, and S. A. Savari, "Impact of parallelism on data volumes for a multibeam mask writer," *J. Vac. Sci. Technol. B* **34**(6), 06KF01 (2016).
6. N. Chaudhary, Y. Luo, and S. Savari, "A parallel multibeam mask writing method and its impact on data volumes," *Proc. SPIE* **10032**, 1003206 (2016).
7. D. Tennant, "Limits of conventional lithography," in *Nanotechnology*, G. Timp, Ed., pp. 161–205, Springer, New York (1999).
8. J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, 5th ed., Morgan Kaufmann Publishers Inc., San Francisco, California (2011).
9. V. Dai and A. Zakhor, "Lossless layout compression for maskless lithography systems," *Proc. SPIE* **3997**, 467–477 (2000).
10. V. Dai, "Data compression for maskless lithography systems: architecture, algorithms and implementation," PhD Thesis, University of California, Berkeley (2008).
11. J. Yang and S. A. Savari, "Lossless circuit layout image compression algorithm for maskless direct write lithography systems," *J. Micro/Nanolithogr. MEMS MOEMS* **10**(4), 043007 (2011).
12. J. Yang, "Lossless circuit layout image compression algorithms for multiple electron beam direct write lithography systems," PhD Thesis, University of Michigan, Ann Arbor (2012).
13. N. Chaudhary et al., "Lossless layout image compression algorithms for electron-beam direct-write lithography," *J. Vac. Sci. Technol. B* **33**(6), 06FD01 (2015).
14. A. Carroll et al., "The REBL DPG: recent innovations and remaining challenges," *Proc. SPIE* **9049**, 904917 (2014).
15. J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Trans. Inf. Theory* **23**(3), 337–343 (1977).
16. J. Ziv and A. Lempel, "Compression of individual sequences via variable-rate coding," *IEEE Trans. Inf. Theory* **24**(5), 530–536 (1978).
17. P. Petric et al., "REBL nanowriter: reflective electron beam lithography," *Proc. SPIE* **7271**, 727107 (2009).
18. A. Chandra and K. Chakrabarty, "System-on-a-chip test-data compression and decompression architectures based on Golomb codes," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **20**(3), 355–368 (2001).
19. H. Matsumoto, "Blanking system for multi charged particle beams, and multi charged particle beam writing apparatus," U.S. Patent 20,160,141,142 (2016).
20. E. Platzgummer, C. Klein, and H. Loeschner, "Electron multibeam technology for mask and wafer writing at 0.1 nm address grid," *J. Micro/Nanolithogr. MEMS MOEMS* **12**(3), 031108 (2013).
21. E. Platzgummer, C. Klein, and H. Loeschner, "eMET POC: realization of a proof-of-concept 50 keV electron multibeam mask exposure tool," *Proc. SPIE* **8166**, 816622 (2011).
22. E. Platzgummer, "Maskless lithography and nanopatterning with electron and ion multibeam projection," *Proc. SPIE* **7637**, 763703 (2010).
23. M. M. Mano, *Digital Design*, EBSCO Publishing, Inc., Ipswich, Massachusetts (2002).
24. H. Matsumoto et al., "Multi charged particle beam writing method and multi charged particle beam writing apparatus," U.S. Patent App. 15/384,021 (2017).
25. L. Zhao, Y. Wei, and T. Ye, "Analysis of multi-e-beam lithography for cutting layers at 7-nm node," *J. Micro/Nanolithogr. MEMS MOEMS* **15**(4), 043501 (2016).

Narendra Chaudhary is currently pursuing his PhD in electrical engineering at Texas A&M University. He received his BTech degree in electrical engineering from the Indian Institute of Technology, Jodhpur, India, in 2012. From 2012 to 2014, he worked at the National Instruments R&D. Since then, he attends Texas A&M University and has been working under the supervision of Dr. Serap Savari. His current research interests include signal processing, data compression, and computer architecture with applications in VLSI fabrication.

Serap A. Savari is on the faculty of Texas A&M University. She has served on the program committee of the annual Data Compression Conference since 2000. From 2002 to 2005, she was an associate editor for the *IEEE Transactions on Information Theory*. She has served on the program committees for several conferences and workshops in information theory, and she joined the program committee of the European Mask and Lithography Conference in January 2017.