Physical Synthesis of DNA Circuits with Spatially Localized Gates

Jinwook Jung, Daijoon Hyun, and Youngsoo Shin
Department of Electrical Engineering, KAIST
Daejeon 305-701, Korea
Email: jinwookjung@kaist.ac.kr

***Abstract*—With the current DNA nanotechnology, we are now able to arrange DNA molecules on a DNA origami to compose a logic gate. This in turn realizes a spatially localized DNA circuit, on which the logic gates are placed on the specific locations as in electronic circuits. In this paper, we address three key problems in designing large-scale spatially localized DNA circuits. An AND gate, made of four hairpins, functions in stochastic manner and sometimes outputs a wrong result. Given tolerable error probability at each circuit output, we address how the probability that each AND gate functions correctly can be determined, which in turn determines the location of constituent hairpins. In the second problem, we study how hairpins are arranged on a DNA origami to minimize the area of a whole circuit, which determines the area of the origami board. The third problem regards the DNA domain assignment so that connected gates can communicate without interference.**

I. Introduction

Solving various computational problems by the massive parallelism intrinsic to DNA hybridization was earlier research interest of DNA computing researchers, since the first demonstration of DNA computer by Adleman [1]. A few difficulties in practical implementation such as DNA sequence design and interpretation of computing results, however, have limited its application. A recent application is more toward designing a circuit, which can directly interact with living things and molecules, e.g. through drugs. In this regard, research has been performed to develop logic gates, i.e. AND and OR gates, with DNA and their application to digital circuit design. Such devices can serve as the foundation for a variety of future biological applications, including targeted drug delivery, smart therapeutics and autonomous DNA nanorobots [2]–[4].

Various DNA logic gates have been reported [5]–[8], which can be categorized into global and local. A global DNA gate floats in a solution relying on diffusion to interact with other gates [7]. Signal transmission among gates is realized through matching DNA sequences, therefore every gate in a circuit must have distinct DNA sequences. A localized DNA gate, which we take into account in this paper, has its own location on a special molecular board [8]. This makes a DNA circuit more reliable since the spatial localization minimizes undesirable interference among gates, and even faster due to the larger effective concentration.

A synthetic DNA molecule called DNA origami is used to build a DNA board; about 200 DNA molecules can be integrated on a typical DNA board [8]. Even though multiple

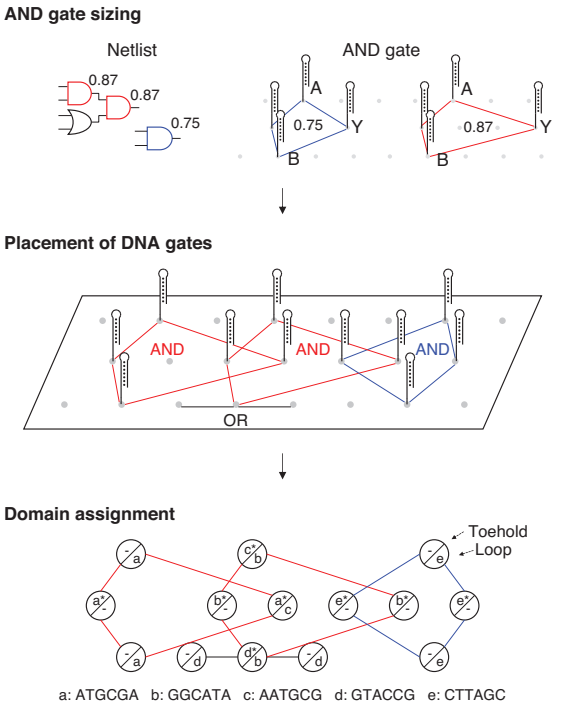

Fig. 1. Physical synthesis of localized DNA circuits.

DNA origamis can be connected to create a large DNA board, it is desirable to minimize the number of origamis because of manufacturing cost; a special DNA molecule, M13mp18, whose cost reaches to more than $15,000 per 1 nmol is necessary to build an origami [9]. Thus, it is important to integrate DNA molecules on a DNA board as many as possible.

A. Contribution

Our focus in this paper is on the physical synthesis of DNA circuits to realize large-scale integration, which is outlined in Fig. 1. A gate-level netlist is synthesized as a DNA circuit with all locations of hairpins on a DNA board determined, and the sequences of the hairpins are appropriately assigned to guarantee the proper circuit operation. We identify three key problems in this paper that have to be addressed to establish the efficient physical synthesis for DNA circuits:

- Sizing of AND gates in a netlist to guarantee proper circuit operation (Section III).

978-1-4673-7166-7/15/$31.00 ©2015 IEEE

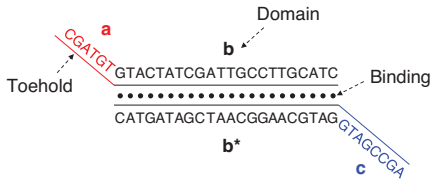

Fig. 2. Double-stranded DNA structure with toeholds.

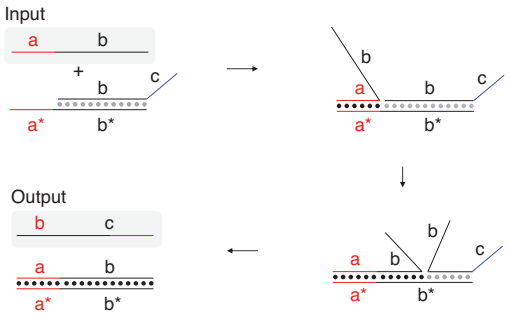

Fig. 3. DNA strand displacement.

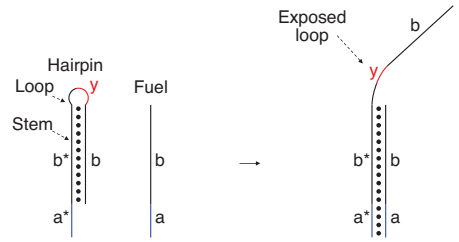

Fig. 4. Strand displacement in a hairpin.

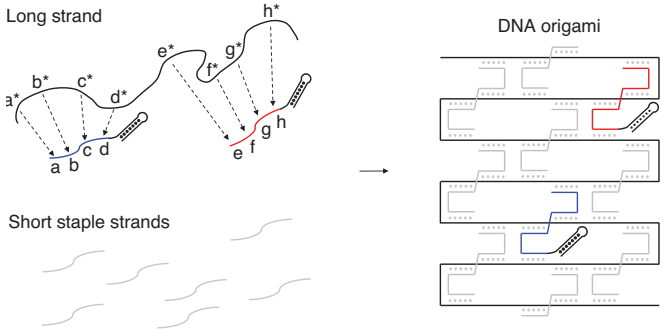

Fig. 5. DNA origami.

- Placement of DNA circuit to reduce the circuit footprint (Section IV).
- Domain assignment for each hairpin after DNA gate placement (Section V).

II. Preliminaries

A. DNA Structure

A DNA strand is made of four types of base nucleotides, adenine (A), thymine (T), cytosine (C), and guanine (G). It is well known that binding (or hybridization) arises only between A and C, and between T and G. A basic DNA structure useful for designing a DNA circuit is shown in Fig. 2. A sequence of bases is called a domain, e.g. a, b, and c. A complementary domain whose bases bind with those of other domain is denoted by $*$, e.g. b^*. A single strand attached at binded double strand is named a toehold.

B. DNA Strand Displacement

A basic mechanism of DNA circuit operation is through strand displacement [10]. Suppose that a single strand made of domains a and b is given as an input to a DNA structure with a^* as a toehold, as illustrated in Fig. 3. Once a and a^* bind, it may occur that the input strand fully binds with the DNA structure, which causes a strand of b and c to be generated as an output. The output strand can initiate a subsequent strand displacement. The probability that a and a^* bind generally increases as they contain more bases [11]; experiments indicate that a domain of 6–10 bases has a reasonable binding probability.

C. Localized DNA Circuit

In a general DNA circuit, strand displacement arises in a solution. Since input strand and DNA structure freely move, reaction rate is very low and a circuit takes a substantial amount of time (e.g. several hours [6] to operate). In a localized DNA circuit, which is our interest in this paper, DNA structures are attached to a board (called origami) which allows higher reaction rate and faster computation (e.g. several minutes [7]).

1) Hairpin and Origami: A basic structure in a localized DNA circuit is a hairpin shown in Fig. 4. Its strand displacement is also shown, in which input strand is called a fuel. An output of displacement is a strand y–b, a part of the hairpin, which is a prime difference from general displacement process of Fig. 3.

A special DNA structure called origami is used as a circuit board to place hairpins. Fig. 5 illustrates how origami is constructed. It is a self-assembly process to produce large and complex DNA structures using a long single strand (called scaffold) with multiple short strands (called staples). Each staple has a specific domain that is complementary to a certain part of the scaffold; it makes the staples be attached to the designated locations in the scaffold thereby folding it into a desired pattern. We assume that a triangular tiling pattern is used for a DNA board as the pattern will give us the maximum integration denstiy with a pitch of 6nm [12]. Multiple origamis can be used to build large circuit; they interact each other with floating single DNA strands [8].

Hairpins can be arranged on the desired positions of a origami, by combining hairpins with staple strands of the origami. If a hairpin binds with an input fuel (given as a floating single strand), it exposes its loop. The exposed loop is not detached from the DNA origami and only can interact with neighboring hairpins. This feature can be used to isolate the

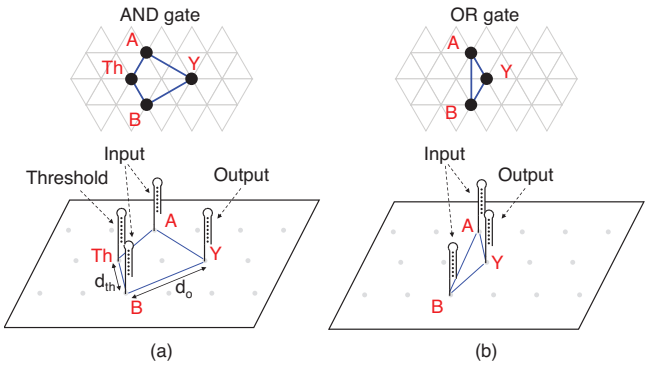

Fig. 6. (a) AND gate, (b) OR gate.

TABLE I
PROBABILITY THAT AND GATE PRODUCES CORRECT OUTPUT

A	B	*Prob*(*Y* is correct)
0	0	1.0
0	1	p
1	0	p
1	1	1.0

information, and thus the hairpin is used widely to compose DNA gates [7], [8].

2) DNA Gates: A few hairpins can be configured on a DNA origami to perform Boolean AND and OR functions.

AND Gate: Fig. 6(a) shows an AND gate. It consists of four hairpins: two for inputs, one for output, and one for a special function called "threshold". If two fuels arrive, they combine with respective input hairpins; one input (combined with fuel or loop) subsequently binds with the threshold while the other binds with the output, exposing the output strand. If a single fuel arrives, it combines with one input hairpin, and they are then likely to combine with the threshold, since it is located closer to inputs than the output; this, however, also means that the output may be asserted in non-zero probability when only one input receives a fuel. An AND gate is thus inherently stochastic with correct output probability listed in Table I.

Hybridization rate between two hairpins is inversely proportional to the square of their distance [7], [13]. Thus, the probability p that an input hairpin combines with the threshold over the output is given by:

$$p = \frac{1/d_{th}^2}{1/d_{th}^2 + 1/d_o^2} = \frac{d_o^2}{d_{th}^2 + d_o^2}, \quad (1)$$

where d_{th} and d_o are defined in Fig. 6(a). Input and threshold hairpins are located in the minimum pitch as shown in Fig. 6(b) for the sake of small footprint. Thus, p is determined by how far the output is located from the threshold; its value is shown for three different shapes of an AND gate in Fig. 7.

OR Gate: Fig. 6(b) shows an OR gate that is implemented with three hairpins; two for inputs and one for output. Two

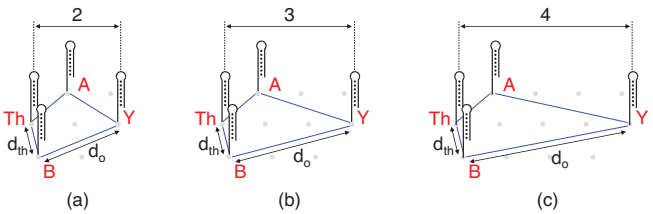

Fig. 7. AND gates of different shapes: (a) $p = 0.75$, (b) $p = 0.87$, and (c) $p = 0.92$

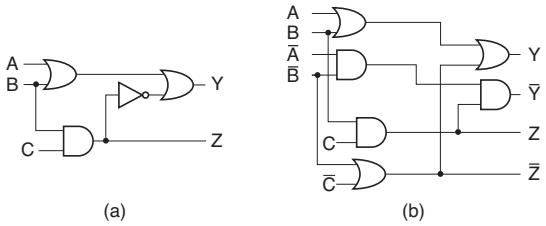

Fig. 8. (a) An example circuit, and (b) its dual-rail implementation.

input hairpins transfer DNA signal to an output hairpin through the similar process of the AND gate; the only difference is the absence of the threshold. If one of the inputs exposes its loop, the toehold of the output is combined with the loop. Then the loop of the output is exposed, which serves as the output signal of the gate.

3) Circuit Design and Operation: There is no NOT gate for a localized DNA circuit as the logic values 1 and 0 are represented by the existence of a DNA strand; it cannot be distinguished whether a strand exists or not due to the asynchronous operation. Thus a DNA circuit is needed to be synthesized in a dual-rail fashion to achieve signal inversion. Any gate-level netlist can be easily converted into a dual-rail circuit network as shown in Fig. 8. Along with the original netlist, its dual with the complement inputs and outputs are generated. NOT gate is now simply implemented by exchanging the fanout of a net with that of its complement.

The operation of a DNA circuit is basically carried out on a molar basis, typically on the order of several nanomoles. Billions of identical DNA circuits operate in parallel, and generate multiple identical output strands; the output hairpins of gates, which generate circuit outputs, are replaced with the double-stranded DNA shown in Fig. 2. Circuit outputs are then identified using the relative concentration of the output strands. If the concentration is low (high), then the corresponding logic value is interpreted as 0 (1).

III. AND GATE SIZING

The stochastic nature of an AND gate operation can result in a spurious circuit output; a large AND gate is desirable to restrict the spurious output. Small amount of the spurious output strands, however, can be eliminated by using a special global DNA gate, called thresholding, which determines the maximum tolerable error ratio of a circuit [5], [6]. Our

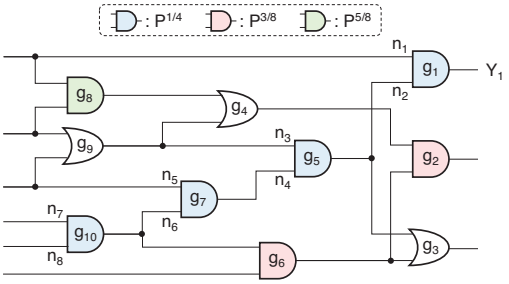

Fig. 9. Example circuit.

TABLE II
GATE SIZING OF AND GATES

Circuit	#Gates	Conservative	Prop.	Diff. (%)	Max size	Avg. size
s298	158	395	305	23	5	3.9
s386	192	576	431	25	6	4.5
s400	222	555	465	16	5	4.2
c432	280	1260	960	24	9	6.9
s510	424	1272	903	29	6	4.3
c880	610	2440	1581	35	8	5.2
c1908	726	3267	2480	24	9	6.8
c2670	1072	4815	2969	38	9	5.5

objective is therefore to control the probability, with which each circuit output releases an output strand spuriously, under the given maximum tolerable error ratio while making AND gates as small as possible.

A path from a circuit input to a circuit output will generate a spurious output strand with the largest probability when an input vector is given such that all the on-path signals are sensitized as logic value 0, and all the side-input signals for AND (OR) gates are sensitized to logic value 1 (0); we will designate such path sensitization as the *worst sensitization*. Erroneous outputs of any AND gates on a path always propagate toward the circuit output through the path under the worst sensitization, even if only one AND gate on that path operates spuriously. Therefore, a circuit output from a path under the worst sensitization is correct only when every AND gate in that path correctly operates; the correct output probability for the worst sensitization is given by $\prod_{g_i \in AND(P)} p_i$, where $AND(P)$ is the set of AND gates in a path P.

A. Algorithm

We devise an algorithm based on the assumption that every path to a circuit input to a circuit output is sensitizable to the worst sensitization. The algorithm finds iteratively a circuit path P that contains the largest number of AND gates, of which correct output probabilities are not yet determined. For each iteration, the correct output probability p of each AND gate in P is assigned with the objective of to

$$\text{minimize } \sum_{g_i \in AND(P)} p_i, \tag{2}$$

such that

$$\prod_{g_i \in AND(P)} p_i \ > \ 1-R, \tag{3}$$

where R is the given maximum tolerable error ratio. Eq. (2) is minimized when every p_i has the same probability $(1-R)^{1/n}$, where n is the number of AND gates in the path. It is simply proven using the inequality of arithmetic and geometric means.

With logic value correlations, such as fanout reconvergence by which some paths could not be sensitized as the worst sensitization, the algorithm would overestimate the error probability of a circuit output. However, it does not occur in a gate-level netlist after logic minimization by the following proposition:

Proposition 1: A path of a dual-rail circuit network after logic minimization is sensitizable such that all the on-path signals are 0, and the side-input signals for AND and OR gates are 1 and 0, respectively.

Proposition 1 holds regardless of logic value correlations in a circuit. Since a gate-level netlist is always obtained through logic synthesis, in which multiple steps of logic minimization are performed, we assume in this paper that every circuit network is after logic minimization. Proof of Proposition 1 is given in Appendix.

Let us now take an example circuit of Fig. 9 to show how the algorithm works with it. At first, we choose the path $\{g_1, g_5, g_7, g_{10}\}$ that consists of four AND gates. The correct output probability for each AND gate is determined as $(1-R)^{1/4}$. Next, the algorithm can choose a path from three paths $\{g_{10}, g_6, g_2\}$, $\{g_6, g_2\}$, and $\{g_8, g_4, g_2\}$; note that p_{10} is already assigned. The algorithm selects the path with the smallest cumulative correct output probabilities. In this case $\{g_{10}, g_6, g_2\}$ is chosen; p_2 and p_6 are determined as $((1-R)/p_{10})^{1/2} = (1-R)^{3/8}$. Finally, the path $\{g_8, g_2\}$ is picked, and then p_8 is assigned as $(1-R)/p_2 = (1-R)^{5/8}$. We determine the size of each AND gate as the smallest one whose correct output probability is larger than the assigned value of p.

B. Assessments

We implemented the AND gate sizing algorithm in C. Several test circuits are taken from ISCAS benchmark [14], and gate-level netlists were obtained using an industrial logic synthesis tool. Logic synthesis was done in such a way that every gate-level netlist only contains 2-input AND and OR gates as well as inverters. Each of them was then converted into a dual-rail circuit network using the same manner of Fig. 8. The resulting dual-rail gate-level netlists are given to our AND gate sizing algorithm as inputs. Maximum tolerable error ratio R is set to 0.25, which corresponds to the error probability of the smallest possible AND gate (See Fig. 7). To evaluate the effectiveness of our algorithm, we implemented a simple conservative algorithm that assigns the same size for all the AND gates; the size is determined as the smallest size with which the path containing the most AND gates satisfies Eq. (3).

Table II shows the results of AND gate sizing. The third and forth columns present the total AND gate sizes in nm^2, and the fifth column shows the difference of the two methods. The last two columns are the maximum and the average size of AND

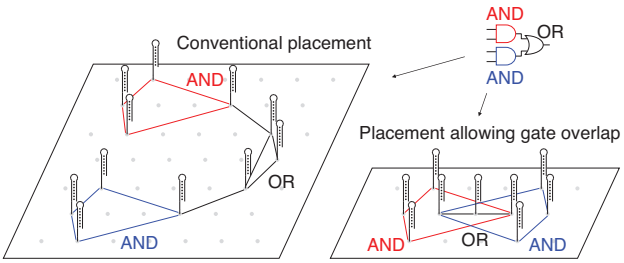

Fig. 10. Placement of DNA circuit with spatially localized gates.

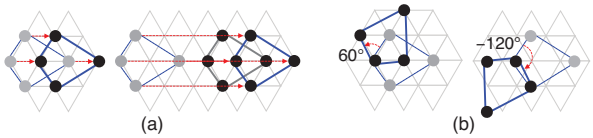

Fig. 11. Functions for simulated annealing: (a) Displace(G), (b) Rotate(G).

gates. Our algorithm reduces total AND gate size by 27%, on average, compared to the simple method. More reduction of AND gate sizes is observed as the difference between the maximum and average sizes of AND gates becomes larger.

IV. Placement of DNA Gates

Since there is no general methodology to locate gates on a DNA origami, the placement of a DNA circuit has been done manually; gates are placed based on the relative locations in the schematic of the circuit [8]. However, it incurs larger circuit area, and thus more DNA origamis to be used for arranging gates. We take the strategy of allowing overlap among gates to reduce circuit footprint as shown in Fig. 10.

To find a placement solution, we leverage simulated annealing [15]. We set the cost function as the placement area of the given circuit, because it directly affects the cost for manufacturing a DNA origami. To generate new placement in simulated annealing, we define three perturbation functions: *Displace*(g), *Rotate*(g), and *Transform*(g).

Displace(g) is a function that displaces a gate g by the minimum possible number of grids. The direction to which g displaces is selected randomly from multiples of 60°. Fig. 11(a) shows an example. If all the adjacent grids of hairpin of g in the target direction are empty, *Displace*(g) displaces g to that grids; if one of them is already occupied by another gate, *Displace*(g) displaces g to more amounts until g can be placed. Fig. 11(b) depicts the *Rotate*(g) function. It rotates g by a multiple of 60° about the threshold-output axis; the rotation angle is also chosen randomly.

We also devised a dedicated function for OR gates, *Transform*(g), which tries to modify the shape of an OR gate g. It takes a hairpin of an OR gate randomly, and displaces it to randomly chosen direction by the minimum possible number of grids. Fig. 12 shows some possible shapes of OR gates. Three hairpins of an OR gate can be more freely placed; even

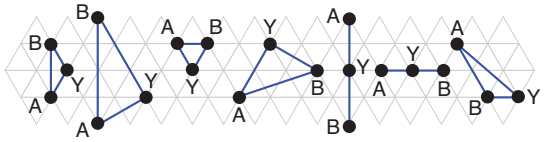

Fig. 12. Some possible footprints of an OR gate.

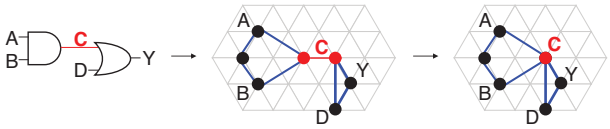

Fig. 13. Hairpin merging.

TABLE III
Placement Results

Circuit	#Gates	ISCA [8] (nm^2)	Prop. (nm^2)	Diff. (%)
s298	158	115763	17474	85
s386	192	165102	21605	87
s400	222	162654	24941	85
c432	280	312417	31987	90
s510	424	365575	47684	87
c880	610	681811	69273	90
c1908	726	804636	82071	90
c2670	1072	1183368	122896	90

asymmetric shape can be taken. The distance between each input hairpins and the output hairpin only need to be smaller than the influence range of a hairpin.

During placement perturbations, some hairpins can be merged. Consider an input hairpin of a gate that accepts its input signal from the output of other gates, as illustrated in Fig. 13. The output hairpin of the upstream gate can be used directly as an input of its fanout gate. Therefore, the two gates can share the hairpin.

A. Assessments

We implemented the simulated annealing DNA circuit placer using Python. It takes the netlist after AND gate sizing as its input. The benchmark circuits were the same as the experiment of Section III. We also generated placements of the test circuits using the placement method presented in [8] to compare with our results.

Table III presents the placement results. The third and forth columns present the area occupied by hairpins in each circuit. Because the main objective of the placement of [8] is to reduce the number of necessary DNA sequences, the placement determines the relative location of each gate based on the topological locations in the circuit schematic, and then places the gate sufficiently apart from other gates; this makes different DNA gates can leverage the same DNA sequences. Since the placement method in [8] intentionally locates gates

Input: Placed hairpin netlist $G(H,E)$, influence ranges of hairpins
Output: Constraint graph

L1: **for** each $h \in H$ **do**
L2: **if** h is a circuit output or a threshold **then** $V \leftarrow V \cup \{h_i\}$
L3: **else if** h is a circuit input **then** $V \leftarrow V \cup \{h_o\}$
L4: **else** $V \leftarrow V \cup \{h_i, h_o\}$
L5: **for** each vertex representing hairpin loop h_o in V **do**
L6: $R \leftarrow n_i$ for each hairpin n in the interference range of h
L7: $S \leftarrow$ fanouts of h_o, and fanins of the fanouts
L8: $C \leftarrow$ vertices already connected to h_o
L9: $D \leftarrow R \setminus (S \cup C \cup \{h_i\})$
L10: **if** h has unique fanin f, and h is f's unique fanout **then**
L11: Remove f_i from D
L12: **for** each pair (d, d'), where $d \in D$ and $d' \in S$ **do**
L13: Create an edge (d, d')

Fig. 14. Algorithm for graph formulation.

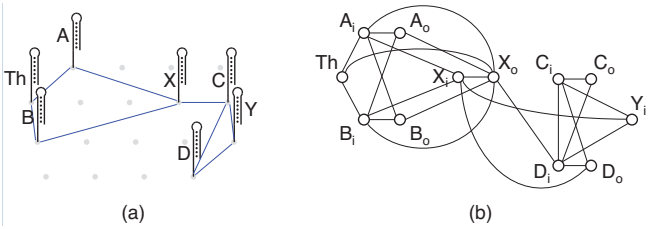

Fig. 15. (a) an example DNA circuit, (b) graph formulation.

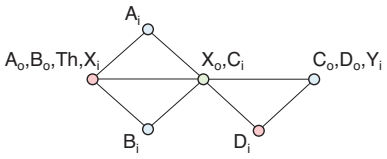

Fig. 16. Vertex merging and graph coloring.

apart, the placement area becomes too larger compared to our placements as shown in Table III; our placement method which allows overlaps among gates can reduce the circuit area by 88% on average.

V. Domain Assignment

Domains of hairpins must be assigned carefully so that connected gates can communicate without interference. Consider a placement of a DNA circuit shown in Fig. 15(a) which consists of an AND gate and an OR gate. Every hairpin has two domains, one for the toehold and the other for the loop. Each toehold (loop) domain should be assigned to the identical domain of its fanin loop (fanout toehold) to ensure the connectivity of hairpins; for example, the domains of the toehold of X and the loops of A as well as B should be the same. All other pairs of a loop and a toehold of different hairpins that are located close to each other should be assigned to different domains. For simplicity, let the inference range of hairpins, which is defined as the distance a hairpin can reach to, be three times of grid pitch in this example. For instance, the loop of X and the toehold of D can be combined, and, therefore, the domains of them must be different.

The problem of the domain assignment can be reduced to a vertex coloring problem through a graph formulation as shown in Fig. 15(b). Each vertex of the graph represents either a toehold or a loop domain of a hairpin. An edge exists when a pair of vertices has to be assigned with different domains.

A. Formulation

We develop a graph formulation algorithm as outlined in Fig. 14. The algorithm first generates vertices that represent two domains needed to be determined of each hairpin (L1–L4); for a hairpin h, h_i denotes the vertex of the toehold domain, and h_o denotes the vertex of the loop. There are some exceptions. If a hairpin is a circuit input or output, there is only one domain necessary to assign for loop and toehold, respectively (L2–L3). The thresholds for AND gates also have one domain to assign, because their loop domain do not combine with any hairpin toeholds (L2).

Now edges are created; two vertices constituting an edge should have different domains (L5–L13). For each vertex h_o that represents a loop of h, the algorithm generates a set R that includes the toehold vertices of all hairpins which are in the h's inference range because those vertices can be combined with h_o (L6). Then the algorithm finds the fanouts of h_o and the fanins of the fanouts, and stores them in a set S; they should have the same domain because of the connectivity (L7). Then D, which is given by $R \setminus S$, is a set of the vertices that has to be different to h_o; because h_i and h_o are not bound to be different each other, h_i also removed from D (L9). When h has the unique fanin f, and f has the unique fanout h, f_i cannot combined with h_o. This is because h_o is exposed only when f_o is combined with h_i. Therefore, such f_i is also removed from D (L10–L11). At last, the algorithm creates edges between all the element pairs of D and S.

After the graph formulation, we can now determine the domain of each hairpin's toehold and loop. We merge the vertices of the graph that should have the same domain. It is easily done by merging toehold vertices and their corresponding fanin vertices, as illustrated in Fig. 16. For example, X_o and C_i are merged into the same vertex because the toehold of hairpin C and the loop of hairpin X must be able to bind. A vertex coloring is performed on the graph of Fig. 16; vertices of the same color, i.e. the chromatic number of the graph, can be assigned to the same domain.

B. Assessments

The graph generation algorithm for domain assignment was implemented using Python. The placement results of Section IV were given as the inputs of the algorithm. Influence ranges of a hairpin was determined by the maximum size of AND gate in a circuit. Conflict graph of each placement was then generated, and its chromatic number was calculated using a greedy vertex coloring algorithm.

Table IV shows the results of domain assignments. The third

TABLE IV
SEQUENCE ASSIGNMENT RESULTS

Circuit	#Hairpins	Exclusive	Proposed
s298	553	288	45
s386	672	358	44
s400	777	404	43
c432	980	546	43
s510	1484	822	44
c880	2135	1186	42
c1908	2541	1402	42
c2670	3752	2068	41

column of the table presents the number of necessary DNA domains when we exclusively assign each hairpin's domain; it is a conventional method of domain assignments used for global DNA circuit. The forth column shows the results of our algorithm. The number of necessary domains obtained by our algorithm does not depend on the number of gates. It is determined only by the influence range of hairpins; a domain can be reused for different hairpins that are spatially separated. On the other hand, the number of necessary domains increases as the circuit size becomes larger when we use the conventional method; for example, it is about 50 times larger than that of our algorithm for a benchmark circuit c2670.

VI. CONCLUSION

We have addressed three key problems on the physical synthesis of the spatially localized DNA circuits. The first problem was about the gate sizing of AND gates to guarantee proper circuit operations. Given a maximum tolerable error ratio, we presented an algorithm to determine the size of each AND gate in a circuit network. We then moved to the placement problem of DNA circuits, and presented a placement method based on the simulated annealing. It allows overlap among gates as well as various shapes of OR gates to reduce the placement area of DNA circuits. Packing DNA gates in small area may cause unwanted interference among hairpins; the third problem was on the domain assignment. A graph-based methodology to determine the domains of the hairpin in a circuit was proposed.

APPENDIX

Consider a path P from a circuit input to a circuit output shown in Fig. 17. Let x_i be the ith side-input signal sensitized to logic value 1, and y_j be the jth side-input signal sensitized to 0. Now, suppose that P is not sensitizable to the worst sensitization as the side-input signal of the gate g, the ith AND gate on P from the circuit input, cannot be sensitized to logic value 1 because of a logic value correlation. As the output f of the gate g takes logic value 0 in that case, Boolean expression of f can be written as

$$f = \left(\bar{A} + \bar{x}_1 + \cdots + \bar{x}_{i-1} + x_i + y_1 + \cdots + y_j\right) \cdot u, \qquad (4)$$

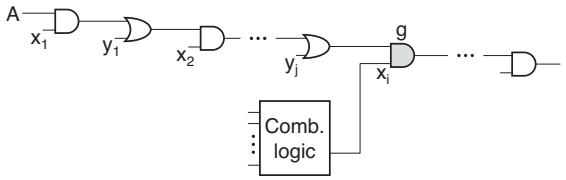

Fig. 17. An example path in a dual-rail circuit.

where A is the circuit input of P, j is the number of OR gates before g, and u is a Boolean expression. The side input x_i becomes 0 by the assumption, so it can be expressed as

$$x_i = \left(\bar{A} + \bar{x}_1 + \cdots + \bar{x}_{i-1} + y_1 + \cdots + y_j\right) \cdot v, \qquad (5)$$

where v denotes a Boolean expression. Replacing $\bar{A} + \bar{x}_1 + \cdots + \bar{x}_{i-1} + y_1 + \cdots + y_j$ by w, and substituting Eq. (5) into Eq. (4), we have

$$f = (w + w \cdot v) \cdot u = w \cdot u, \qquad (6)$$

which contradicts the assumption that the circuit is after logic minimization. Therefore, every AND gate in a path of a circuit network after logic minimization is sensitizable to the worst sensitization. The proof for the case that g is an OR gate can be done in a similar way. ■

REFERENCES

[1] L. Adleman, "Molecular computation of solutions to combinatorial problems," *Science*, vol. 266, no. 5187, pp. 1021–1024, Nov. 1994.
[2] Y. Benenson, B. Gil, U. Ben-Dor, R. Adar, and E. Shapiro, "An autonomous molecular computer for logical control of gene expression," *Nature*, vol. 429, no. 6990, pp. 423–429, May 2004.
[3] J. Bath and A. J. Turberfield, "DNA nanomachines," *Nature nanotechnology*, vol. 2, no. 5, pp. 275–284, May 2007.
[4] S. M. Douglas, I. Bachelet, and G. M. Church, "A logic-gated nanorobot for targeted transport of molecular payloads," *science*, vol. 335, no. 6070, pp. 831–834, Feb. 2012.
[5] G. Seelig, D. Soloveichik, D. Zhang, and E. Winfree, "Enzyme-free nucleic acid logic circuits," *Science*, vol. 314, no. 5805, pp. 1585–1588, Dec. 2006.
[6] L. Qian and E. Winfree, "Scaling up digital circuit computation with DNA strand displacement cascades," *Science*, vol. 332, no. 6034, pp. 1196–1201, Jun. 2011.
[7] H. Chandran, N. Gopalkrishnan, A. Phillips, and J. Reif, "Localized hybridization circuits," in *Proc. DNA Computing and Molecular Programming*, Jan. 2011, pp. 64–83.
[8] R. Muscat, K. Strauss, L. Ceze, and G. Seelig, "DNA-based molecular architecture with spatially localized components," in *Proc. Int. Symp. on Computer Architecture*, Jun. 2013, pp. 177–188.
[9] "M13mp18 single-stranded DNA." [Online]. Available: https://www.neb.com/products/n4040-m13mp18-single-stranded-dna
[10] D. Zhang and E. Winfree, "Control of DNA strand displacement kinetics using toehold exchange," *Journal of the American Chemical Society*, vol. 131, no. 47, pp. 17 303–17 314, Nov. 2009.
[11] D. Zhang and G. Seelig, "Dynamic DNA nanotechnology using strand-displacement reactions," *Nature chemistry*, vol. 3, no. 2, pp. 103–113, Feb. 2011.
[12] P. Rothemund, "Folding DNA to create nanoscale shapes and patterns," *Nature*, vol. 440, no. 7082, pp. 297–302, Mar. 2006.
[13] M. Lakin, R. Petersen, K. Gray, and A. Phillips, "Abstract modelling of tethered DNA circuits," in *Proc. DNA Computing and Molecular Programming*, Jan. 2014, pp. 132–147.
[14] "ISCAS'85 benchmark circuits." [Online]. Available: http://www.cbl.ncsu.edu/benchmarks/ISCAS85
[15] K. Shahookar and P. Mazumder, "VLSI cell placement techniques," *ACM Comput. Surv.*, vol. 23, no. 2, pp. 143–220, Jun. 1991.