A Framework for Genetic Logic Synthesis

In this paper, the authors introduce a framework to address the challenges of applying logic synthesis techniques to genetic logic devices.

By Prashant Vaidyanathan, *Student Member IEEE*, Bryan S. Der, Swapnil Bhatia, *Member IEEE*, Nicholas Roehner, Ryan Silva, Christopher A. Voigt, and Douglas Densmore, *Senior Member IEEE*

ABSTRACT | Digital electronic circuits have inspired synthetic biologists to program living cells with synthetic decision-making circuits by creating multilevel genetic logic gates. In both genetic and electronic circuits, logic synthesis translates an abstract functional description into a representation that can be physically implemented. However, inherent differences in genetic logic devices present new synthesis challenges. Here we provide a starting point for a growing set of biodesign automation tools to tackle this challenge in a unified way. This framework will enable direct comparison of new approaches, results that are transferable and standardized, and research into independent areas of the problem formulation with a clear path toward future integration.

KEYWORDS | Boolean; digital; genetic; logic; synthesis; synthetic biology; transcription

I. INTRODUCTION TO GENETIC LOGIC

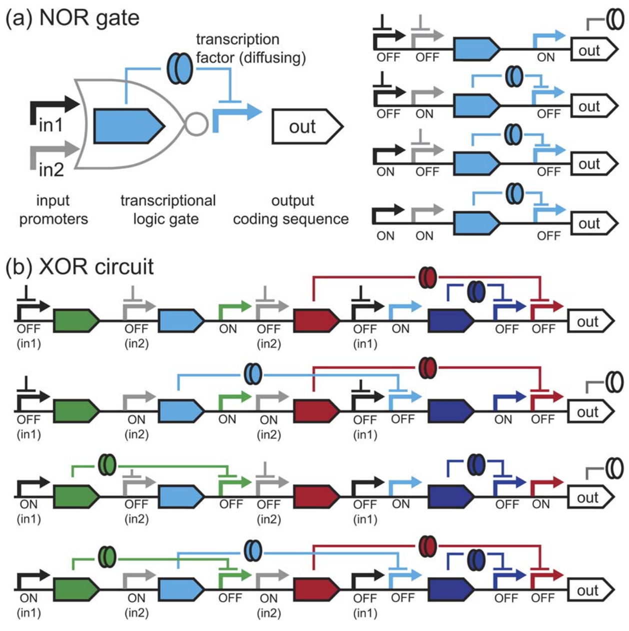
Boolean algebra is a foundation for understanding and defining logic that has enabled robust engineering of many of today's large scale electronic systems. In the field of synthetic biology, Boolean logic functions are now being engineered into living cells [1]–[4]. Genetically encoded logic operates within the central dogma of molecular biology, which describes the flow of information from DNA to RNA to protein. Transcription is the biological process by which DNA is decoded to RNA by a polymerase (RNAP), while translation is the process by which RNA is decoded to protein. High/low levels of RNAP flux can be abstracted to ON/OFF logic states, which is the basis for transcriptional genetic logic.

While genetic logic can be implemented in many cell types at the transcriptional, translational, or posttranslational stages of the central dogma, our synthesis framework is primarily focused on transcriptional genetic logic circuits. Transcriptional genetic logic circuits are made up of one or more transcriptional units. A transcriptional unit contains several different types of DNA components (genetic parts), but promoters and coding sequences are sufficient for a basic understanding of a unit's function. Promoters are start sites for transcription. One or more promoters preceding a coding sequence for a transcription factor enables activation or repression of RNAP flux at a different promoter. These genetic regulatory relationships, illustrated using directed arcs from coding sequences to promoters (see Fig. 1), enable the implementation of transcriptional genetic logic gates, such as BUFFER, NOT, OR, NOR, AND, and NAND. For example, an input promoter, coding sequence for a transcriptional repressor, and output promoter implement a genetic NOT gate. A NOR gate can be built from the same parts, but two input promoters are required [see Fig. 1(a)]. Furthermore, an XOR circuit can be built by connecting two NOT gates, two NOR gates, and one OR output [see Fig. 1(b)]. In this circuit, promoters are ON unless repressed by a transcription factor, and the conditional presence/absence of repression arcs results in XOR logic for the output coding sequence.

Manuscript received October 15, 2014; revised April 4, 2015; accepted June 2, 2015. Date of publication August 18, 2015; date of current version October 26, 2015. This work was supported by the National Science Foundation under Awards 1147158 and 1253856, and by the Air Force Academy Faculty Pipeline Fellowship. The work of B. S. Der and C.A. Voigt was supported by the Office of Naval Research (ONR) Grant N00014-13-1-0074, by the National Institutes of Health Grant GMO98792, and by the National Science Foundation Synthetic Biology Engineering Research Center (SynBERC) EEC0540879.

P. Vaidyanathan, **S. Bhatia**, **N. Roehner**, **R. Silva**, and **D. Densmore** are with the Department of Electrical and Computer Engineering, Boston University, Boston, MA 02215 USA (e-mail: prash@bu.edu; bhatia.swapnil@gmail.com; nicholasroehner@gmail.com; rjsilva@bu.edu; dougd@bu.edu).
B. S. Der and **C. A. Voigt** are with the Department of Biological Engineering, Massachusetts Institute of Technology, Cambridge, MA 02139 USA (e-mail: bder@mit.edu; cavoigt@gmail.com).

Digital Object Identifier: 10.1109/JPROC.2015.2443832



(a) NOR gate
transcription factor (diffusing)
in1
in2
out
input promoters
transcriptional logic gate
output coding sequence
OFF OFF ON out
OFF ON OFF out
ON OFF OFF out
ON ON OFF out
(b) XOR circuit
OFF (in1) OFF (in2) ON OFF (in2) OFF (in1) ON OFF OFF out
OFF (in1) ON (in2) ON ON (in2) OFF (in1) OFF ON OFF out
ON (in1) OFF (in2) OFF OFF (in2) OFF (in1) ON OFF ON out
ON (in1) ON (in2) OFF ON (in2) OFF (in1) OFF OFF OFF out

Fig. 1. ***Genetic implementation of combinational logic. (a) Promoter and coding sequence symbols (bent arrows and block arrows) are derived from standardized iconography [5]. Promoters are transcriptional start sites, which can be ON or OFF. Coding sequences encode transcription factors that can diffuse and regulate promoter state (repression arc). When a coding sequence encodes a protein-based transcription factor, transcription and translation occur, but we do not explicitly show the RNA transcript in this illustration. A NOR gate requires two input promoters, a coding sequence encoding a transcriptional repressor, and an output promoter. (b) A multilevel XOR circuit has a unique transcriptional state for each row of the truth table as indicated by the presence/absence of specific repression arcs. The output (white coding sequence) is transcribed when either upstream promoter is ON.***

II. SYNTHESIS CHALLENGES FOR GENETIC LOGIC

The first section introduced how specific layouts of DNA components (promoters and coding sequences) can implement genetically encoded logic functions. This section describes three engineering challenges for genetic circuit design (see Fig. 2, with new terms defined in Section III).

1) *Crosstalk:* Unlike metal wires, the signal carriers in genetic circuits are regulatory molecules that can diffuse and mix within the liquid medium of a cell. Sometimes this diffusion also occurs between cells, for example, if the molecule is involved in cell-to-cell communication [6], [7]. Due to diffusion within and/or between cells, each gate must encode unique regulatory proteins or RNA to avoid unintended logic computations. In synthetic biology, this phenomenon is known as crosstalk. Crosstalk limits the scalable design of genetic logic circuits and requires significant experimental effort to identify and quantify regulators that have unique binding partners [8], [9]. For example, reusing the same promoter-coding sequence pair for each gate in the XOR circuit results in multiple undesired regulatory arcs [see Fig. 2(a)].

2) *Signal Matching:* In contrast to electronic logic gates, for which input/output signals have definable system-wide high/low levels, genetic logic gates have unique response functions that describe experimentally measured input-output relationships and determine whether an input/

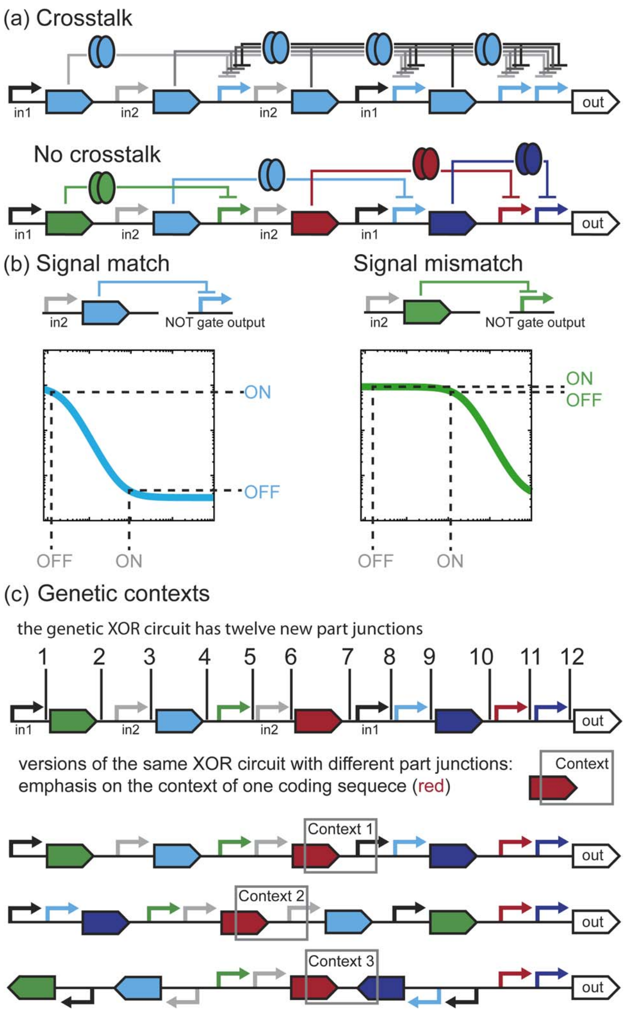


Fig. 2. ***Key challenges to genetic logic synthesis: (a) Crosstalk. Signal carriers are transcription factors, which are diffusible molecules that mix within a cell. Crosstalk occurs when the same signal carrier is reused in more than one logic gate in a cell. To avoid crosstalk, each gate must use a unique signal carrier. (b) Signal matching. The dynamic range of the blue NOT gate matches the ON/OFF dynamic range of the input promoter. The dynamic range of the green NOT gate does not match and results in a logic error where both outputs are high. (c) Genetic context effects. While there are several different types of genetic context effects, we illustrate new DNA component junctions in this figure. A larger circuit will have more junctions and carries a higher risk for nonmodular component behavior. In circuits with the same genetic gates, variations in order and orientation of the transcriptional units result in different neighboring DNA components.***

output signal is high or low [9], [10]. Thus, assigning particular Boolean functions to genetic logic gates does not guarantee successful genetic logic, as signal mismatches can result in logic errors [see Fig. 2(b)].

3) Context Effects: The physical properties of biological systems are challenging to quantify and understand. Even after extensive efforts to characterize a series of genetic logic gates and avoid both crosstalk and signal mismatches, logic errors can occur due to unanticipated nonmodularity that arises when genetic components are used in new genetic, cellular, and environmental contexts [4], [11]. For example, the performance of a component can be altered by its neighboring components [12], [13], and unintended components can form at component junctions [14]. The risk of local component interference increases as circuit size increases [see Fig. 2(c)]. Context effects can also act at a distance, such as retroactivity when a transcription factor binds multiple instances of a promoter [15]. Other context effects are not as well understood, such as resource competition within the host cell, growth rate of the host cell [16], and environmental factors that affect host cell physiology [11], [17].

Despite these challenges, the field of genetic circuit design continues to mature. Crosstalk is being mitigated by programmable transcription factors that can bind specifiable DNA sequences, thus expanding the number of orthogonal molecular "wires" (RNA-guided dCas9 [18]–[20], zinc fingers [21], TAL effectors [22], and small RNAs [23], [24]). Signal matching can be improved by more precise and reproducible experimental measurement and characterization techniques [25], [26]. Context effects are also being identified and insulated [11], [17]: genetic components have been developed to flank promoters [12], [13], and retroactivity can be quantitatively modeled [15]. As genetic components increase in both number and modularity, computer-aided design will play a vital role in genetic circuit design. With this goal in mind, we define a framework for genetic logic synthesis.

III. A FRAMEWORK FOR GENETIC LOGIC SYNTHESIS

Genetic logic synthesis is the process by which an abstract functional description is mapped to a design for a genetic logic circuit, one that includes a complete DNA sequence to inform its physical construction. Genetic circuits broadly include combinational logic circuits for decision-making, sequential circuits for switches and counters [27], [28], and analog circuits for linear sensing and ratio calculation [29], [30]. These are all active areas of research, but this framework focuses on combinational logic circuits. In order to define our framework for genetic logic synthesis, we first describe the general process of logic synthesis, then define terms and concepts for formulating the genetic logic synthesis problem.

A. Logic Synthesis

We define logic synthesis with this workflow: specification, design, build (see also Section V). The input to the system is any abstract logic description that is able to be specified using a description language or description tool. The output is a technology-specific mapping of the input logic.

Specification is the process of creating a correct representation of the description that is canonical for optimization, such as a truth table or a reduced ordered binary decision diagram (ROBDD) [31]. This point in the workflow is closest to the abstract functional description of an electronic logic circuit and is the step with the least concern for the target technology. An example of specification for electronic circuits is the act of accepting or rejecting a description written in Verilog for RTL synthesis, as per the IEEE Standard for Verilog RTL Synthesis (1364.1-2002) [32]. The standard enables reproducibility among different synthesis tools and defines a specific subset of Verilog that can be reliably instantiated in hardware.

Next, the design step of the synthesis process takes the output of specification and produces an optimized gate-level schematic. This optimization weighs several cost functions, such as reducing the total number of components in the schematic or the total amount of component fan-in [33] (examples of genetic cost functions in Table 1). The functional characteristics of the gate-level components, but not necessarily their physical attributes, must be defined using an external library, which usually consists of logic gates at the level of NOT, AND, OR, and NOR.

Build is the final step in the process of converting an abstract design into a realizable form. This step translates the optimized gate-level schematic into structures that can be found in a specific target technology. Since this step is technology specific, it marks the farthest point in the workflow from the abstract functional description.

B. Terms

- **Graphs**: In our framework, gate-level schematics are represented using **directed acyclic graphs** (**DAGs**).
  1) A **graph** $G = (V, E)$ is a data structure consisting of a set of **vertices** V and a set of **edges** E connecting these vertices [34]. In our framework, each vertex in a graph represents a gate in a gate-level schematic or an input/output component for the schematic as a whole.
  2) A **directed graph** is a graph in which each edge is an ordered pair of elements (v_i, v_t) in V. An edge e_{it} is said to start with its **initial vertex** v_i and end with its **terminal vertex** v_t.
  3) A **path** from v_0 to v_n is an ordered set of edges $(v_0,v_1), (v_1,v_2), (v_2,v_3), \ldots, (v_{n-1},v_n)$, such that the terminal vertex of each edge is the same as the initial vertex of the next edge in the path.

4) In an **acyclic graph**, there exists no path that starts with a vertex v_i and ends with the same vertex v_i.
5) A **colored graph** is a graph in which a **color** is assigned to each vertex of the graph, such that no two vertices are assigned the same color. This restriction of color uniqueness is necessary to avoid crosstalk during genetic logic synthesis.

- **Synthetic Biology**: We now define the terms from synthetic biology that are necessary for a basic understanding of genetic logic synthesis.
  1) A **DNA component** (also referred to as a genetic part in synthetic biology) is a finite string of characters $(p_1, p_2, p_3, p_4, \ldots, p_n)$, where $p_i \in \{A, T, G, C\}$. A, T, G, and C represent the nucleotides adenine, thymine, guanine, and cytosine, respectively, which are the basic building blocks of DNA. A new junction of two DNA components results in a new **genetic context** for each component.
  2) **Transcription** is the biological process by which DNA is "read" to produce RNA.
  3) **Translation** is the biological process by which RNA is "read" to produce protein.
  4) A **promoter** is a type of DNA component (bent arrows in Fig. 1 and Fig. 2) at which transcription starts. Promoter locations determine fan-in and fan-out for a gate.[1]
  5) A **coding sequence** is a type of DNA component (box-arrows in Figs. 1 and 2) that encodes and can be transcribed to RNA. RNA typically encodes and can be translated to protein, though RNA-based regulators do not require translation.
  6) **RNAP** (RNA polymerase) binds a promoter in the ON state and transcribes the downstream coding sequences to RNA. Levels of transcription at a promoter can be expressed in terms of **RNAP flux**, the number of polymerases per second that start transcription at that promoter.
  7) A **transcription signal** is a measure of transcription based on RNAP flux at a promoter that serves as a Boolean input or output in transcriptional genetic logic.
  8) A **transcription factor** is a diffusible signal carrier (RNA or protein) that regulates transcription at a specific promoter. This regulation is the basis for implementing transcriptional genetic logic.
  9) A **genetic module** is a set of DNA components and any of their encoded or associated biochemical components (such as RNA and protein) that interact to perform some biological function.
  10) A **transcriptional genetic logic gate** is a genetic module comprised of a set of one or more coding sequences that regulate a promoter via their encoded transcription factor(s). This regulation is abstracted to a logical relationship between the input transcription signals from any promoters composed with the gate's coding sequences and the output transcription signal from the gate's promoter.
  11) Each transcriptional genetic logic gate has an empirically determined **response function** ϕ

$$\phi : \mathbb{R}^k_{\geq 0} \rightarrow \mathbb{R}_{\geq 0}.$$

  The response function specifies how the combination of input transcription signals of gate k is mapped to the output transcription signal of gate k. Examples of response functions for NOR gates, AND gates, and an OR gate are shown as heatmaps in Fig. 3 (white is low, black is high).
  12) A **transcriptional unit** is a partially ordered set of DNA components that is capable of undergoing transcription, typically including one or more adjacent promoters followed by a coding sequence. The set is partially ordered because the input promoters must precede the coding sequence, but the order of the input promoters can vary. Transcriptional units form when transcriptional genetic logic gates are connected to form a circuit and a promoter from one gate is composed with a coding sequence from another gate.
  13) A **transcriptional genetic logic library** is a set of promoters for generating transcription signals, a set of coding sequences for reading/displaying transcription signals, and a set of transcriptional genetic logic gates for transforming input transcription signals to output transcription signals in accordance with a set of response functions.
  14) A **transcriptional genetic logic circuit** is a subset of a transcriptional genetic logic library with a colored DAG to indicate the

[1]Fan-in is determined by the number of physically adjacent, nonidentical promoters driving expression of a coding sequence. Note that genetic logic gates with fan-in greater than two have not yet been reported. Limitations to fan-in include a high OFF level due to accumulation of leaky expression from each promoter, and physical interference between RNAP and downstream promoters bound by transcription factors. Fan-out of a primitive gate is determined by the number of disjoint instances of the same promoter. Homologous recombination can delete DNA between repeated sequences, which may place restrictions on fan-out for certain gates. Recombination can occur at 20–30 base pairs of identical sequence, but larger sequences will be more susceptible. Additionally, high fan-out might affect the response function due to transcription factor titration effects (retroactivity).

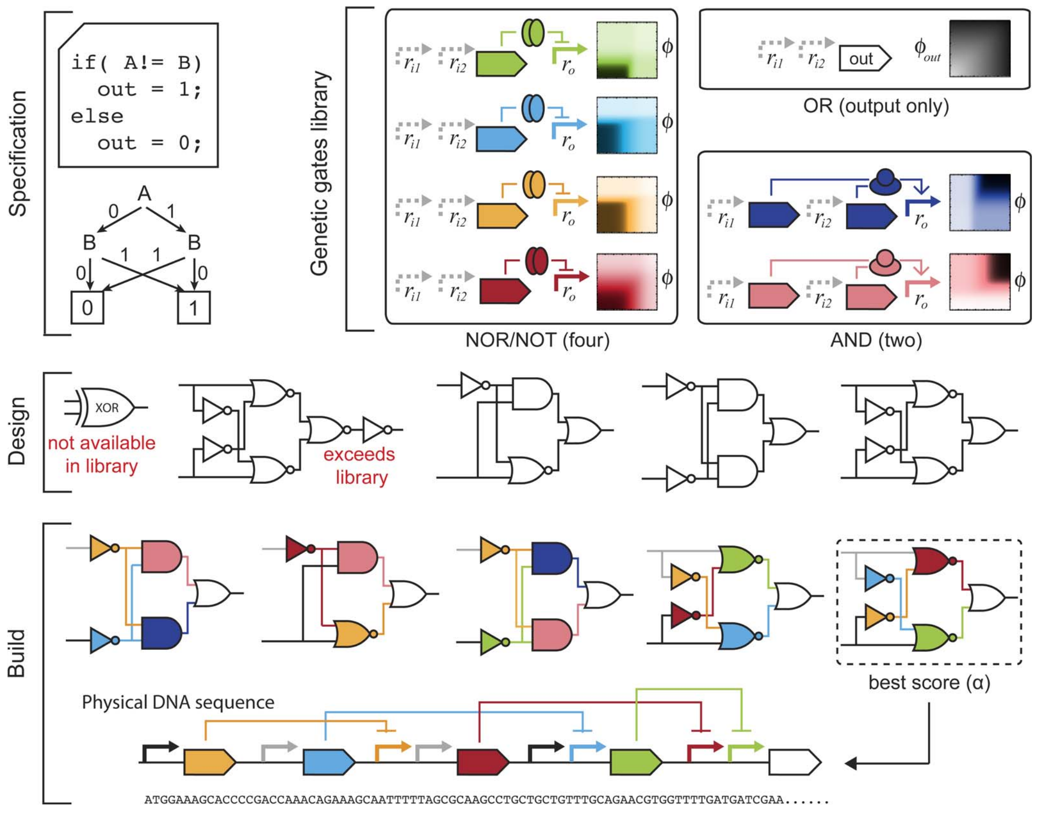


Fig. 3. ***Genetic logic synthesis workflow. Genetic gates library. The library consists of four NOR/NOT gates, two AND gates, and an OR output. Dashed gray input promoters will be replaced by promoters from other gates in a connected circuit. In the response function heatmaps showing $r_o = \phi(r_{i1}, r_{i2})$, white indicates low output and black indicates high output. Regulation arcs ending with a flat line indicate repression, arcs ending with an arrow indicate activation. Specification. The logic function of interest and the library of experimentally characterized DNA components are required inputs. Design. The logic function is mapped to an uncolored graph, where gates are represented as generic functional primitives with no physical characteristics. The type of gates and number of gates must be available in the genetic gates library. Five functionally equivalent Boolean circuits are shown, but only three can be fully colored by the genetic gates library. Build. The graph is colored using available DNA components from the library. Black and gray promoters are the input vertices, the white coding sequence is the output vertex, and other colors represent a transcription factor, which can only be used once per circuit to avoid crosstalk. The goal is to color the graph to maximize circuit score α. Once the graph is colored, all DNA components are concatenated into a final DNA sequence S, which can be ordered from a company or constructed in a molecular biology lab.***

connectivity of its elements. In general, each color in the DAG represents a transcriptional genetic logic gate that encodes a unique transcription factor. The colors of the input and output vertices of the graph (see Section III-C1), however, represent individual promoters and coding sequences, respectively. This exception is necessary for complete transcriptional units to be formed at the boundaries of the graph (see Fig. 3).

15) The **DNA sequence** S produced by genetic logic synthesis is an ordered set of DNA components. This ordered set must conform to a ligation or physical concatenation of the set of transcriptional units that arise from the connectivity of the synthesized transcriptional genetic logic circuit.

C. Concepts

1) Types of Vertices: In our framework for genetic logic synthesis, a graph can contain three types of vertices.

- **Input**: A vertex is an input when it is not a terminal vertex for any edge in the graph. A colored input vertex represents a single promoter and generates an output transcription signal on the outgoing edges for which it is an initial vertex.
- **Output**: A vertex is an output when it is not an initial vertex for any edge in the graph. A colored output vertex represents a coding sequence and reads/displays an input transcription signal from the incoming edges for which it is a terminal vertex.
- **Primitive**: A vertex is a primitive when it is a terminal vertex for at least one edge and an initial vertex for at least one other edge in the graph. A colored primitive vertex represents a transcriptional genetic

logic gate. Hence, a primitive vertex reads one or more input transcription signals from its incoming edges and generates an output transcription signal on its outgoing edges. In the case of digital logic synthesis, a primitive vertex would map n input signals to one output signal using a completely specified Boolean function

$$f:\{0,1\}^n \rightarrow \{0,1\}.$$

In the case of genetic logic synthesis, however, a primitive vertex must map n input signals to one output signal using the response function of its genetic logic gate.

2) Boolean Signal Representation: In an electronic logic circuit, the discrete state of each electrical signal is determined by measuring an electrical property, such as the voltage or current in a wire. An electrical signal can be high or low, depending on whether the value of the measured electrical property is above or below a specified threshhold.

For example, in a typical complementary metal-oxide-semiconductor (CMOS) circuit that is supplied with a voltage of V_{DD}, a wire with a voltage ranging from 0 to $V_{DD}/2$ transmits a low signal and is represented in Boolean logic as 0. A wire with a voltage ranging from $V_{DD}/2$ to V_{DD}, transmits a high signal and is represented in Boolean logic as 1. Accordingly, the state of an electronic logic gate depends on the properties of the gate and the states of its input signals in the context of the circuit as a whole.

In a genetic logic circuit, however, it is the response function of each genetic logic gate that determines the Boolean representation of signals within the circuit [see Fig. 2(b)]. Like an electronic logic gate, the state of a genetic logic gate depends on the properties of the gate and the states of its input signals. The state of each input signal, however, is determined by comparing the value of the signal to the response function of the gate, rather than to a circuit-wide threshold. This poses a challenge for defining discrete high and low states for genetic logic circuits.

3) Response Functions: Transcriptional genetic logic circuits transmit data via transcription signals. A transcription signal provides a measure $r \in \mathbb{R}_{\geq 0}$ of RNAP flux within a genetic logic circuit. In the colored DAG that represents such a circuit, each primitive vertex relates the values of its k input transcription signals $(r_{i1}, r_{i2}, \ldots, r_{i_k})$ to the value of its output transcription signal (r_o) using a response function ϕ

$$r_o = \phi(r_{i1}, r_{i2}, \ldots, r_{i_k}).$$

Generally, each response function ϕ of a primitive vertex captures how its input transcription signals are combined over the coding sequences of its corresponding gate and subsequently transformed to an output transcription signal via genetic regulation.

Unlike primitive vertices, the input vertices of a graph do not possess response functions of the sort defined here. Instead, an input vertex supplies its output transcription signal with a value r_o from a closed interval in $\mathbb{R}_{\geq 0}$. Hence, this output signal has both maximum and minimum values ON and OFF.

An output vertex calculates the combined display value r_d of its input transcription signals $(r_{i1}, r_{i2}, \ldots, r_{i_k})$ in accordance with the response function ϕ_{out}.

$$r_d = \phi_{\text{out}}(r_{i1}, r_{i2}, \ldots, r_{i_k}).$$

In general, ϕ_{out} represents how input transcription signals are combined over a coding sequence that encodes the final molecular output of a transcriptional genetic logic circuit. For instance, this output could be a fluorescent protein for experimental measurement, an enzyme for biochemical production, or any natural or recombinant cellular function.

Finally, a response function can also relate the input signals and output signal of a transcriptional genetic logic circuit containing multiple gates. Fig. 3 shows an example of this type of circuit: an XOR gate composed of two NOT gates, two NOR gates, and one OR gate. In order to reuse such a circuit as a library gate, its response function ϕ_C would likely need to be determined empirically (beyond composing the function from that of each individual gate in the circuit). In addition, this composite gate would have multiple colors that would need to be taken into account to avoid introducing crosstalk during genetic logic synthesis.

4) How a Genetic Circuit Implements a Boolean Function: Since transcription signals cannot be defined as having high or low states in the context of a transcriptional genetic logic circuit as a whole, it is challenging to define how such a genetic logic circuit implements a Boolean function.

For example, consider a Boolean function

$$f:\{0,1\}^n \rightarrow \{0,1\} \tag{1}$$

and a colored DAG G that has:

- n input vertices that generate transcription signals with maximum and minimum values ON and OFF;
- m primitive vertices that map their transcription signals in accordance with m response functions $(\phi_1, \phi_2, \ldots, \phi_m)$;
- a single output vertex that displays a transcription signal calculated with ϕ_{out}.

The value of the function ϕ_C represented by G can be determined by calculating the value of ϕ_{out}, which depends in turn on the values of the m response functions and the n

transcription signals generated by the input vertices. By setting the values of these generated input signals to either ON or OFF and mapping them to the input set $\{0,1\}^n$ of (1), we obtain 2^n values for ϕ_C. Let $\mathrm{ON}_{\min} = \{\phi_C | f(x) = 1\}_{\min}$ and $\mathrm{OFF}_{\max} = \{\phi_C | f(x) = 0\}_{\max}$, where $f(x)$ is the Boolean function and $x \in \{0,1\}^n$. We can then say that G realizes the Boolean function specified in (1) if

$$ON_{\min} > OFF_{\max}. \tag{2}$$

This conservative definition for a Boolean function allows the condition specified in (2) to realize multiple Boolean functions. For instance, consider the following possible values of ϕ_C: {16,0.2,0.4,0.5} for inputs a and b.

These values can realize all of the following:

a	b	$(a+b)'$	$(a \oplus b)'$	$(a+b')$	1	0	*Value*
0	0	1	1	1	1	0	16
0	1	0	0	0	1	0	0.2
1	0	0	0	1	1	0	0.4
1	1	0	1	1	1	0	0.5

To solve this ambiguity, we introduce a score α, to define the extent to which a circuit realizes the Boolean function (1). One possible form for α is

$$\alpha = 1 - \frac{(r_{d_{\max}} - \mathrm{ON}_{\min}) + (\mathrm{OFF}_{\max} - r_{d_{\min}})}{2(r_{d_{\max}} - r_{d_{\min}})} \tag{3}$$

where $r_{d_{\max}}$ is the maximum value of the codomain of ϕ_C and $r_{d_{\min}}$ is the minimum value of the codomain of ϕ_C.

In the case of a Boolean Tautology: $f : \{0,1\}^n \rightarrow \{1\}, \alpha$ reduces to

$$\alpha = 1 - \frac{(r_{d_{\max}} - \mathrm{ON}_{\min})}{(r_{d_{\max}} - r_{d_{\min}})} \tag{4}$$

and for Boolean Contradictions: $f : \{0,1\}^n \rightarrow \{0\}, \alpha$ can be defined as

$$\alpha = 1 - \frac{(\mathrm{OFF}_{\max} - r_{d_{\min}})}{(r_{d_{\max}} - r_{d_{\min}})}. \tag{5}$$

We now demonstrate how α defined in (3), (4), and (5) can be used to quantify the Boolean functions that realized the values $\phi_C : \{16.0, 0.2, 0.4, 0.5\}$ using the condition specified in (2). Let $r_{d_{min}} = 0.1$ and $r_{d_{max}} = 17$. Note that these are hypothetical numbers that do not reflect simulation results

Function	$r_{d_{\max}}$	$r_{d_{\min}}$	$\mathrm{ON}_{\min}$	$\mathrm{OFF}_{\max}$	α
$(a+b)'$	17.0	0.1	16.0	0.5	0.9586
$(a \oplus b)'$	17.0	0.1	0.5	0.4	0.5030
$(a+b')$	17.0	0.1	0.4	0.2	0.5059
1	17.0	0.1	0.2	N/A	0.0059
0	17.0	0.1	N/A	16.0	0.0592

This table illustrates that the Boolean function $(a+b)'$ can best realize the values $\phi_C : \{16, 0.2, 0.4, 0.5\}$.

IV. THE PROBLEM STATEMENT

Within the established framework, we define the problem of genetic logic synthesis as follows:

Given a Boolean function

$$f : \{0,1\}^n \rightarrow \{0,1,x\}^m \tag{6}$$

(where x denotes a don't care) and a finite transcriptional genetic logic library, with uniquely colored input promoters with known ON/OFF levels, logic gates with known response functions, output coding sequences and a well defined α, synthesize a DNA sequence S which can realize the logic function specified by (6) (see Fig. 3).

A. Objectives

- minimize the number of primitive vertices;
- represent each vertex with a unique color from the genetic gates library [see Fig. 2(a)];
- color the sequence S to maximize the value of α [see Fig. 2(b)].

B. Problem Complexity

We now remark briefly on the complexity of the genetic logic synthesis (GLS) problem. We consider the special case when α is given to be 1, since if GLS for any α were easy, it could be used to solve for $\alpha = 1$ too.

Consider the following subset sum problem (SSP): given a set S of positive integers and a positive integer k, decide whether a subset of S sums to k. Via a reduction from the Knapsack problem, SSP is known to be NP-complete [35]. A polynomial time reduction exists from SSP to the GLS. Each element $s_i \in S$ is translated into a Buffer gate—one that leaves its input logically unchanged—with a piecewise linear response function

$$\phi_i(x) = x + s_i, \text{ if } x < (k+1) + 2\sum_{s \in S} s; \tag{7}$$

$$= 0, \text{ otherwise} \tag{8}$$

resulting in a gate library L of $|S|$ gates. Let the boolean function to be synthesized be $f = \{(0 \mapsto 1), (1 \mapsto 0)\}$—the unary negation function. Let the input promoter operating points for low and high inputs be 0 and $k+1$ respectively.

We first argue that any circuit synthesized using Buffer gates alone will have an $\alpha < 1$. This is because for $f(0)$, the output $\text{ON}_{\min} = \sum_{s \in B} s$ for the set of Buffer gates $B \subseteq L$ used in the synthesized circuit. For $f(1)$, the output $\text{OFF}_{\max} = (k+1) + \sum_{s \in B} s$.

But, for the final gate

$$r_{d_{\min}} = s_i, \quad \text{if the final gate is gate } i, \text{ and} \tag{9}$$

$$r_{d_{\max}} = (k+1) + 2\sum_{s \in S} s. \tag{10}$$

We note that as per the definition of α, no circuit whose $\text{OFF}_{\max} \neq r_{d_{\min}}$ or whose $\text{ON}_{\min} \neq r_{d_{\max}}$ can achieve an $\alpha = 1$, since α measures the relative dynamic range of the output gate utilized by the circuit. Thus, for the circuit comprising solely buffer gates, $\alpha < 1$.

Next, consider an additional gate g with

$$\begin{aligned} \phi_g(x) &= x \text{ if } x \leq k, \\ &= 0 \text{ otherwise.} \end{aligned}$$

Note that a circuit comprising g alone will also have $\alpha < 1$ since $\phi_g(0) = 0$ and $\phi_g(k+1) = 0$, which implies that $\text{ON}_{\min} = 0$ and $\text{OFF}_{\max} = 0$. But $r_{d_{\max}} = k$ and $r_{d_{\min}} = 0$. Thus

$$\alpha = 1 - \frac{(k-0) + (0-0)}{2(k-0)} = \frac{1}{2}. \tag{11}$$

Also note that g cannot act as an input gate at the given operating points.

With the gate library $L \cup \{g\}$, consider an instance of SSP where the answer to the SSP is "yes." Thus, there is a subset of S that sums to k. We must show that a corresponding circuit with $\alpha = 1$ exists. Consider a circuit C comprising buffer gates $B \subseteq L$, with

$$\begin{aligned} \phi_C(0) &= \sum_{s \in B} s = k, \text{ and} \\ \phi_C(k+1) &= (k+1) + \sum_{s \in B} s = 2k+1. \end{aligned}$$

Now consider C plugged into g; call the full circuit C'. In this case

$$\begin{aligned} \phi_{C'}(0) = k \,&\big|\, r_{d_{max}} = k \,\big|\, \text{ON}_{\min} = k, \\ \phi_{C'}(k+1) = 0 \,&\big|\, r_{d_{\min}} = 0 \,\big|\, \text{OFF}_{\max} = 0 \end{aligned}$$

$$\alpha(C') = 1 - \frac{(k-k) + (0-0)}{2(k-0)} = 1. \tag{12}$$

Now consider an SSP where the answer to the SSP is "no." Thus, all subsets of S sum to a value either greater or smaller than k. As we have shown above independent of the status of the SSP, the buffer gates alone and the gate g alone both result in a circuit with $\alpha < 1$. So, consider the case where some circuit D comprising buffer gates alone is plugged into gate g. (Call this circuit D'.) Since this is a "no" instance of SSP, $\phi_D(0) \neq k$. Therefore, $\phi_{D'}(0) \in \{0, \phi_D(0)\}$. (If g were an internal gate, the argument still holds.) Hence $\text{ON}_{\min} \neq r_{d_{max}} = k$. Thus, $\alpha(D) < 1$.

Thus, we have shown that an instance of SSP can be translated into an instance of GLS such that a circuit with $\alpha = 1$ is obtained if and only if the SSP instance is a positive one. But it is well known that SSP is complete for NP. Thus, it must be the case that GLS is also NP-hard. Furthermore, the decision version of GLS—"is there a circuit C implementing f with library L with $\alpha(C) = 1$?"—is clearly in NP, since given C, its α can be computed and its gates checked to be in L in time polynomial in $|L|$ and $|C|$. Thus, the decision version of GLS is NP-complete.

Thus, our proof reveals a fundamental challenge in designing multistage genetic circuits: the selection of a response function matched subset of dynamic range-maximizing gates from an arbitrary collection of gates.

V. DISCUSSION

A major goal of both genetic and electronic circuit synthesis is to translate an abstract functional description into a technology-specific representation that can be directly implemented [32]. Here we discuss similarities and differences between synthesis in the domains of living technology versus electronic technology in the context of our generalized, three-step synthesis process.

A. Specification

Specification is the process of describing abstract logic, vetting the description for synthesis, and transforming it to a canonical form. Each of these terms are described in this section.

Abstract logic requires a language for circuit description. For genetic circuits, this language could be Verilog, VHDL [36], or more genetic-specific languages, such as the genetic engineering of living cells (GEC) [37], Proto [38], or the synthetic biology open language (SBOL) [5] standard. Currently, SBOL describes the structure but not the higher-order function of a genetic design. With proper extension [39], SBOL could form the basis for a standard genetic synthesis language: a standard language originally used for circuit description can become a powerful tool for circuit synthesis, as did VHDL for electronic circuit design.

All of the previous languages potentially contain instructions that may or may not be synthesizable. As a result, the chosen language for circuit description must be vetted for constructibility via a standard, although the requirements for "appropriateness" of a genetic circuit will be different than that of an electronic circuit. For example, genetic circuits lack a clocking function, rendering edge-sensitive Verilog models as not appropriate for genetic construction. Additionally, genetic synthesis currently lacks a standard for reproducibility and appropriateness as the IEEE standards for VHDL and Verilog provide for electronic circuit synthesis.

Finally, the vetted description is transformed into a correct representation of the description that is canonical for optimization, such as a truth table or a reduced ordered binary decision diagram (ROBDD) [31].

B. Design

The design step takes the output of specification and uses an optimized number of generic primitive gates to implement its logic operation. For example, ABC [44] uses an AND-Inverter graph library to heuristically optimize the specified logic operation. The gates are considered generic because they have no detailed information regarding their physical characteristics. In approaches to genetic logic synthesis, generic logic gates often take the form of abstract genetic modules, DNA components, and/or biochemical components (such as RNA and protein) that fix the types of biological mechanisms implementing the logic, but not the exact physical structures of the components involved in these mechanisms.

Optimization during the design step must be defined by at least one cost function. For electronic circuits, wires could be considered cheap, as in the case of an optimal construct for a programmable logic array. This construct is achieved via two-level synthesis and results in a circuit expressed in sum-of-products form, thus having a higher fan-in than a circuit produced by multilevel synthesis (i.e., synthesis using multiple levels of gates) [33]. For genetic circuits, additional wires are frequently considered expensive and thus multilevel synthesis is preferred.

Other costs are challenging to quantify. First, unknown genetic context effects that can degrade circuit performance carry a risk factor that increases with circuit size. Second, since genetic circuits are run by a living host cell, each additional signal competes for the same metabolic energy and resources the cell needs to survive. These resource limitations and other modes of circuit toxicity are challenging to predict, so the size of the genetic program needed to implement the logic function should be minimized (see Table 1 for examples of cost functions). This can be done at the generic gate-level during the design step or a merged design-and-build step. Since genetic logic circuits have orders of magnitude fewer gates (less than 20 promoters or coding sequences [2], [8]) than electronic circuits, more exhaustive methods for gate-level optimizations might prove useful.

C. Build

The build step takes the optimized generic gate structure and composes a functionally identical structure that is specific to a particular technology. Whereas a gate component is defined primarily by its function, gates and components at the build layer are further characterized by their physical attributes [33]. The build process for both electronic and genetic domains begins by finding physical constructs that behave reliably and making these constructs modular. These physical constructs can draw from multiple biological mechanisms for the same logic function [4]. For example, an AND gate can be constructed from RNA toehold switches, riboswitches, RNA-aptamers, metabolic enzymes [45], coregulatory transcriptional activators [8], recombinases [46], split polymerases [47], or by combining NOR and NOT gates (also having multiple biological mechanisms) [9], [20]. The problem becomes a matter of routing intermediate signals between reliable modules, taking care to match signal types. Not all biological mechanisms of logic share the same type of signal, which in our framework is a measure of transcription from a promoter.

The output of the build step is a fully mapped structural representation of the optimized logic construct expressed solely in components found within the list of reliable modules. This list of reliable modules can be thought of as an Application Specific Integrated Circuit (ASIC) standard cell library in electronics and a transcriptional genetic logic library in genetics. Synthesis targeting an ASIC further behaves as an analog to synthesis targeting genetic circuits in that both require postsynthesis fabrication. In contrast, there is not currently a genetic counterpart to a premanufactured field-programmable gate array (FPGA).

Though genetic circuit mapping resembles that for ASICs, the genetic equivalent to the standard cell library also has a finite number of unrepeatable intermediate signal carriers from which to draw. Graph coloring from a finite library is required to avoid crosstalk from transcriptional factors that coexist and mix within the same intracellular medium. Furthermore, genetic gates cannot currently conform to a single static discipline that defines a standard noise margin and system-wide thresholds for input/output signal strengths. Instead of using a static discipline, each gate interaction in genetics requires signal matching. Finally, even a correctly colored and tuned circuit could reliably produce failures during experimentation. These proven failures should be captured in a library and avoided during the build process.

VI. CURRENT APPROACHES

There are many methodologies that can be implemented to solve the genetic logic synthesis problem. While the field of genetic logic synthesis is very new, previously published methodologies for solving the genetic logic synthesis problem have been implemented in various software tools and approaches, including BioJADE [48], GEC [37], AutoBioCad [49], OptCircuit [50], TASBE [51] (includes the Proto BioCompiler [38] and MatchMaker [10]), SBROME [52]–[54], iBioSim [55], Parts & Pools [56], and a top-down

Table 1 Genetic Logic Synthesis Implementations

Concept	**GEC**	**AutoBioCad**	**OptCircuit**	**TASBE**	**SBROME**	**iBioSim**	**Parts&Pools**
				Specification			
Languages	GEC [37] LBS [42] SBML [40] Kappa [43]	SBML [40]	n/a	Proto [38] SBOL [5]	SBOL [5]	SBOL [5] SBML [40]	ProMoT [41] SBML [40]
				Design			
Synthesis Categories	Multi-level	Multi-level	Multi-level	Multi-level	Multi-level	Multi-level	Two-level
Cost Functions	n/a	n/a	n/a	# of TU	# of GM	n/a	# of TF
				Build			
Technology Libraries	DNAC	DNAC GM	DNAC	DNAC	DNAC GM	GLG	AGLG
Mapping Considerations	Crosstalk Simulation	Crosstalk Simulation	Crosstalk Simulation	Crosstalk SM Simulation	Crosstalk LC Simulation	Crosstalk SL Simulation	Simulation
Computational Approaches	ES	MINLP	MINLP	GS	GS DP, B&B MINLP	DP, B&B	ES

Step	**Key**
Specification	GEC = Genetic Engineering of Living Cells, LBS = Langauge for Biochemical Systems, SBML = Systems Biology Markup Language, SBOL = Synthetic Biology Open Language, ProMoT = Process Modeling Tool
Design	GM = Genetic Module, TF = Transcription Factor, TU = Transcriptional Unit
Build	AGLG = Abstract Genetic Logic Gate, B&B = Branch&Bound, DNAC = DNA Component, DP = Dynamic Programming, ES = Exhaustive Search, GLG = Genetic Logic Gate, GS = Greedy Search, LC = Ligation Count, MINLP = Mixed Integer Nonlinear Programming, SL = Sequence Length, SM = Signal Matching

approach with stochastic simulations for assessment [57]. Table I briefly summarizes a selection of these approaches.

VII. CONCLUSION

Synthetic biology continues to see technological gains in its ability to "write" DNA (gene synthesis) and "read" DNA (gene sequencing). As this capacity grows it will be crucial that the level of abstraction at which new designs are created be raised. This phenomenon parallels the increasing silicon resources seen in computing. Similarly, this abstraction will be eased if logical expressions of required behavior can be expressed at a high level and automatically transformed to physical realities. This paper has presented an outline of a framework that will allow for such formalism to exist. Additionally, it has detailed some of the unique challenges facing genetic logic synthesis and identified potential approaches to meet these challenges. The approach presented has explicitly separated functional specification from physical implementation. This separation will allow for this framework to persist as genetic logic technologies advance [4]. Performance metrics are also modular and can be captured electronically to incorporate into computational frameworks. We hope to see additional concepts such as debugging, verification, and automated library development become rich research topics in the future. ■

Acknowledgment

The authors would like to thank J. Madsen and A. Mishchenko for their valuable comments and feedback. Collaborations made possible by the MIT/Broad Foundry and Thermo Fisher Scientific Inc. were also vital.

REFERENCES

[1] T. K. Lu, A. S. Khalil, and J. J. Collins, "Next-generation synthetic gene networks," *Nat. Biotechnol.*, vol. 27, pp. 1139–1150, Dec. 2009.

[2] P. E. Purnick and R. Weiss, "The second wave of synthetic biology: From modules to systems," *Nat. Rev. Mol. Cell Biol.*, vol. 10, pp. 410–422, Jun. 2009.

[3] A. A. Nielsen, T. H. Segall-Shapiro, and C. A. Voigt, "Advances in genetic circuit design: Novel biochemistries, deep part mining, precision gene expression," *Curr. Opin. Chem. Biol.*, vol. 17, no. 6, pp. 878–892, 2013.

[4] J. A. Brophy and C. A. Voigt, "Principles of genetic circuit design," *Nat. Methods*, vol. 11, no. 5, pp. 508–520, 2014.

[5] M. Galdzicki *et al.*, "The Synthetic Biology Open Language (SBOL) provides a community standard for communicating designs in synthetic biology," *Nat. Biotechnol.*, vol. 32, pp. 545–550, 2014.

[6] N. Saeidi, M. Arshath, M. W. Chang, and C. L. Poh, "Characterization of a quorum sensing device for synthetic biology design: Experimental and modeling validation," *Chem. Eng. Sci.*, vol. 103, pp. 91–99, 2013.

[7] A. Tamsir, J. J. Tabor, and C. A. Voigt, "Robust multicellular computing using genetically encoded NOR gates and chemical 'wires'," *Nature*, vol. 469, pp. 212–215, Jan. 2011.

[8] T. S. Moon, C. Lou, A. Tamsir, B. C. Stanton, and C. A. Voigt, "Genetic programs constructed from layered logic gates in single cells," *Nature*, vol. 491, pp. 249–253, Nov. 2012.

[9] B. C. Stanton *et al.*, "Genomic mining of prokaryotic repressors for orthogonal logic gates," *Nat. Chem. Biol.*, vol. 10, pp. 99–105, Feb. 2014.

[10] F. Yaman, S. Bhatia, A. Adler, D. Densmore, and J. Beal, "Automated selection of synthetic biology parts for genetic regulatory networks," *ACS Synth. Biol.*, vol. 1, no. 8, pp. 332–344, 2012.

[11] S. Cardinale and A. P. Arkin, "Contextualizing context for synthetic biology-identifying causes of failure of synthetic biological systems," *Biotechnol. J.*, vol. 7, pp. 856–866, Jul. 2012.

[12] C. Lou, B. Stanton, Y. J. Chen, B. Munsky, and C. A. Voigt, "Ribozyme-based insulator parts buffer synthetic circuits from genetic context," *Nat. Biotechnol.*, vol. 30, pp. 1137–1142, Nov. 2012.

[13] V. K. Mutalik *et al.*, "Precise and reliable gene expression via standard transcription and translation initiation elements," *Nat. Methods*, vol. 10, no. 4, pp. 354–360, 2013.

[14] A. I. Yao *et al.*, "Promoter element arising from the fusion of standard biobrick parts," *ACS Synth. etic Biol.*, vol. 2, no. 2, pp. 111–120, 2013.

[15] D. Del Vecchio, A. J. Ninfa, and E. D. Sontag, "Modular cell biology: Retroactivity and insulation," *Mol. Syst. Biol.*, vol. 4, p. 161, 2008.

[16] S. Cardinale, M. P. Joachimiak, and A. P. Arkin, "Effects of genetic variation on the e. coli host-circuit interface," *Cell Reports*, vol. 4, no. 2, pp. 231–237, 2013.

[17] A. P. Arkin, "A wise consistency: Engineering biology for conformity, reliability, predictability," *Current Opin. Chem. Biol.*, vol. 17, no. 6, pp. 893–901, 2013.

[18] L. S. Qi *et al.*, "Repurposing crispr as an rna-guided platform for sequence-specific control of gene expression," *Cell*, vol. 152, no. 5, pp. 1173–1183, 2013.

[19] D. Bikard *et al.*, "Programmable repression and activation of bacterial gene expression using an engineered crispr-cas system," *Nucleic Acids Res.*, vol. 41, no. 15, pp. 7429–7437, 2013.

[20] A. A. Nielsen and C. A. Voigt, "Multi-input crispr/cas genetic circuits that interface host regulatory networks," *Mol. Syst. Biol.*, vol. 10, no. 11, 2014.

[21] J. J. Lohmueller, T. Z. Armel, and P. A. Silver, "A tunable zinc finger-based framework for boolean logic computation in mammalian cells," *Nucleic Acids Res.*, vol. 40, no. 11, pp. 5180–5187, 2012.

[22] F. Lienert *et al.*, "Two-and three-input tale-based and logic computation in embryonic stem cells," *Nucleic Acids Res.*, vol. 41, no. 21, pp. 9967–9975, 2013.

[23] J. Chappell, M. K. Takahashi, and J. B. Lucks, "Creating small transcription activating rnas," *Nat. Chem. Biol*, 2015.

[24] A. A. Green, P. A. Silver, J. J. Collins, and P. Yin, "Toehold switches: De-novo-designed regulators of gene expression," *Cell*, vol. 159, no. 4, pp. 925–939, 2014.

[25] J. Beal, R. Weiss, F. Yaman, N. Davidsohn, and A. Adler, "A method for fast, high-precision characterization of synthetic biology devices," MIT, Cambridge, MA, USA, Tech. Rep. MIT-CSAIL-TR-2012-008, 2012.

[26] J. R. Kelly *et al.*, "Measuring the activity of biobrick promoters using an *in vivo* reference standard," *J. Biol. Eng.*, vol. 3, no. 1, p. 4, 2009.

[27] T. S. Gardner, C. R. Cantor, and J. J. Collins, "Construction of a genetic toggle switch in Escherichia coli," *Nature*, vol. 403, pp. 339–342, Jan. 2000.

[28] A. E. Friedland *et al.*, "Synthetic gene networks that count," *Science*, vol. 324, no. 5931, pp. 1199–1202, 2009.

[29] R. Daniel, J. R. Rubens, R. Sarpeshkar, and T. K. Lu, "Synthetic analog computation in living cells," *Nature*, vol. 497, no. 7451, pp. 619–623, 2013.

[30] R. Sarpeshkar, "Analog synthetic biology," *Philos. T. R. Soc. A.*, vol. 372, no. 2012, p. 20130110, 2014.

[31] R. E. Bryant, "Graph-based algorithms for boolean function manipulation," *IEEE Trans. Comput.*, vol. C-35, no. 8, pp. 677–691, Aug. 1986.

[32] *IEEE Standard for Verilog Register Transfer Level Synthesis*, IEEE Std 1364.1-2002, 2002.

[33] P. P. Chu, *RTL Hardware Design Using VHDL: Coding for Efficiency, Portability, Scalability*. New York, NY, USA: Wiley-IEEE Press, 2006.

[34] K. H. Rosen, *Discrete Mathematics and Its Applications*. New York, NY, USA: WCB/McGraw-Hill, 1999.

[35] J. C. Lagarias and A. Odlyzko, "Solving low-density subset sum problems," *J. ACM*, vol. 32, no. 1, pp. 229–246, 1985.

[36] Y. Gendrault, M. Madec, C. Lallement, and J. Haiech, "Modeling biology with hdl languages: A first step toward a genetic design automation tool inspired from microelectronics," *IEEE Trans. Bio-Med. Eng.*, vol. 61, no. 4, pp. 1231–1240, Apr. 2014.

[37] M. Pedersen and A. Phillips, "Towards programming languages for genetic engineering of living cells," *J. Roy. Soc. Interface*, vol. 6, no. Suppl. 4, pp. S437–S450, 2009.

[38] J. Beal, T. Lu, and R. Weiss, "Automatic compilation from high-level biologically-oriented programming language to genetic regulatory networks," *PLoS ONE*, vol. 6, 2011.

[39] N. Roehner *et al.*, "Proposed data model for the next version of the Synthetic Biology Open Language," *ACS Synth. Biol.*, vol. 4, pp. 57–71, 2015.

[40] M. Hucka *et al.*, "The Systems Biology Markup Language (SBML): A medium for representation and exchange of biochemical network models," *Bioinformatics*, vol. 19, no. 4, pp. 524–531, 2003.

[41] M. Ginkel, A. Kremling, T. Nutsch, R. Rehner, and E. D. Gilles, "Modular modeling of cellular systems with ProMoT/Diva," *Bioinformatics*, vol. 19, pp. 1169–1176, 2003.

[42] M. Pedersen and G. D. Plotkin, "A language for biochemical systems: Design and formal specification," *Proc. CMSB*, pp. 77–145, 2010.

[43] V. Danos, J. Feret, W. Fontana, R. Harmer, and J. Krivine, "Rule-based modelling, symmetries, refinements," in *Formal Methods in Systems Biology*. Berlin, Germany: Springer-Verlag, 2008, pp. 103–122.

[44] R. Brayton and A. Mishchenko, "Abc: An academic industrial-strength verification tool," *Comput.-Aided Verification, LNCS Comput. Sci.*, pp. 24–40, 2010.

[45] T. Miyamoto, S. Razavi, R. DeRose, and T. Inoue, "Synthesizing biomolecule-based boolean logic gates," *ACS Synth. Biol.*, vol. 2, no. 2, pp. 72–82, 2012.

[46] P. Siuti, J. Yazbek, and T. K. Lu, "Synthetic circuits integrating logic and memory in living cells," *Nat. Biotechnol.*, vol. 31, pp. 448–452, May 2013.

[47] D. L. Shis and M. R. Bennett, "Library of synthetic transcriptional and gates built with split t7 rna polymerase mutants," *PNAS*, vol. 110, no. 13, pp. 5028–5033, 2013.

[48] J. Goler, "BioJADE: A design and simulation tools for synthetic biological systems," master's thesis, MIT, Cambridge, MA, USA, 2004.

[49] G. Rodrigo and A. Jaramillo, "AutoBioCAD: Full biodesign automation of genetic circuits," *ACS Synth. Biol.*, vol. 2, no. 5, pp. 230–236, 2013.

[50] M. S. Daika and C. D. Maranas, "OptCircuit: An optimization based method for computational design of genetic circuits," *BMC Syst. Biol.*, vol. 2, no. 24, 2008.

[51] J. Beal *et al.*, "An end-to-end workflow for engineering of biological networks from high-level specifications," *ACS Synth. Biol.*, vol. 1, no. 8, pp. 317–331, 2012.

[52] L. Huynh, J. Kececioglu, M. Koppe, and I. Tagkopoulos, "Automatic design of synthetic gene circuits through mixed integer non-linear programming," *PloS ONE*, vol. 7, no. 4, 2012.

[53] L. Huynh, A. Tsoukalas, M. Koppe, and I. Tagkopoulos, "SBROME: A scalable optimization and module matching framework for automated biosystems design," *ACS Synth. Biol.*, vol. 2, no. 5, pp. 1073–1089, 2013.

[54] L. Huynh and I. Tagkopoulos, "Optimal part and module selection for synthetic gene circuit design automation," *ACS Synth. Biol.*, vol. 3, no. 8, pp. 556–564, 2014.

[55] N. Roehner and C. J. Myers, "Directed acyclic graph-based technology mapping of genetic circuit models," *ACS Synth. Biol.*, vol. 3, no. 8, pp. 543–555, 2014.

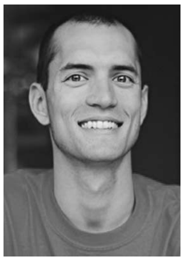
[56] M. A. Marchisio, "Parts & Pools: A framework for modular design of synthetic gene circuits," *Front Bioeng. Biotechnol.*, vol. 2, no. 42, 2014.

[57] S. M. Laursen, J. J. Boysen, and J. Madsen, "A top-down approach for genetic logic synthesi.," in *Proc. IWBDA Con.*, Jun. 2014, vol. 1, pp. 62–63.

ABOUT THE AUTHORS

Prashant Vaidyanathan (Student Member, IEEE) received the M.S. degree in computer engineering from Boston University, Boston, MA, USA, and the B.E. (Hons.) degree in electrical and electronics engineering from the Birla Institute of Technology and Science. He is currently working toward the Ph.D. in computer engineering at Boston University, focusing on functional specification in genetic circuits.

As a part of the Cross-disciplinary Integration of Design Automation Research (CIDAR) group, he developed various computational tools for Synthetic Biology including a Logic synthesis tool for Genetic Circuits. His research interests include Computer Architecture, Computer Arithmetic, Reconfigurable Computing, Embedded Systems, Logic Synthesis and Synthetic Biology.



Bryan S. Der received the Ph.D. degree in biochemistry and biophysics, in 2013, from the University of North Carolina, Chapel Hill, NC, USA, where he designed new protein-protein interactions and zinc binding sites using the Rosetta molecular modeling software.

His current postdoctoral research at MIT and Boston University is to develop Cello, a tool that improves and automates the predictive design of transcriptional logic circuits from experimentally characterized genetic gates.

Swapnil Bhatia (Member, IEEE) received the doctorate in computer science from the University of New Hampshire, Durham, NH, USA, in 2010, where he worked on computer networks, storage systems, and intermittently connected distributed computing systems.

He was a Postdoctoral Researcher at the Palo Alto Research Center where he developed Conovo, a software tool for sequence-constrained *de novo* peptide sequencing from mass spectrometry data. He was a Postdoctoral Scholar at Boston University where he developed Finch—a tool for combinatorial design of genetic devices, and Puppeteer, a platform for laboratory automation for synthetic biology. He is now a Research Assistant Professor at Boston University, Boston, MA, USA, and co-founder of Lattice Automation, a startup focused on developing algorithms and software for synthetic biology.

Nicholas Roehner received the B.S. and Ph.D. degrees in bioengineering from the University of Washington, Seattle, WA, USA, in 2010, and the University of Utah, Salt Lake City, UT, USA, in 2014, respectively.

He is currently working as a Postdoctoral Researcher at Boston University, Boston, MA, USA. He has developed methodologies and algorithms for technology mapping of genetic circuits, which are implemented as part of the genetic design automation software tool iBioSim, and he has contributed as an Editor to the development of the Synthetic Biology Open Language (SBOL), a community standard for the digital reuse and sharing of biological designs. He is currently developing Double Dutch, a web application for designing libraries of variant metabolic pathways that are tailored for use in a design of experiments framework. His research interests include biodesign automation and standards and their reconciliation with laboratory practice.

Ryan Silva is a Major on active-duty in the United States Air Force. He is currently assigned to Boston University, where he is working toward the Ph.D. degree in computer engineering.

After graduating with his degree, Ryan will be assigned to an Air Force unit before returning to teach at the United States Air Force Academy, where he graduated with a degree in Electrical Engineering in 2005.

Christopher A. Voigt received the B.S.E. degree from the University of Michigan, Ann Arbor, MI, USA, in 1998, and the Ph.D. degree from the California Institute of Technology, Pasadena, CA, USA, in 2002.

He is a Professor in the Department of Biological Engineering at MIT, Cambridge, MA, USA, Co-director of the Synthetic Biology Center (SBC), and Editor-in-Chief of *ACS Synthetic Biology*. Previously, he was a Professor in the Department of Pharmaceutical Chemistry at the University of California, San Francisco (2003-2011). He is an Adjunct Professor at the Korea Advanced Institute of Science and Technology (KAIST) and an Honorary Fellow at Imperial College. He has been honored as a Sloan Fellow, Pew Scholar, Packard Fellow, MIT TR35, NSF CAREER Award, and Vaughan Lecturer. His current research interests include reprogramming of bacterial organisms to perform coordinated, complex tasks for pharmaceutical, and industrial applications.

Douglas Densmore (Senior Member, IEEE) received the B.S.E. degree in computer engineering from the University of Michigan, Ann Arbor, MI, USA, in 2001, and the M.S. and Ph.D. degrees in electrical engineering from the University of California, Berkeley, Berkeley, CA, USA, in 2004 and 2007, respectively.

He is a Kern Faculty Fellow at Hariri Institute for Computing and Computational Science and Engineering Junior Faculty Fellow, and an Associate Professor in the Department of Electrical and Computer Engineering at Boston University, Boston, MA, USA. His research focuses on the development of tools for the specification, design, and assembly of synthetic biological systems, drawing upon his experience with embedded system level design and electronic design automation (EDA). He is the director of the Cross-disciplinary Integration of Design Automation Research (CIDAR) group at Boston University, which develop computational and experimental tools for synthetic biology. His research interests include synthetic biology, bioelectronic systems, cyber physical systems, digital logic design, and system level design.