ACS SyntheticBiology

pubs.acs.org/synthbio

Research Article

Automated Leak Analysis of Nucleic Acid Circuits

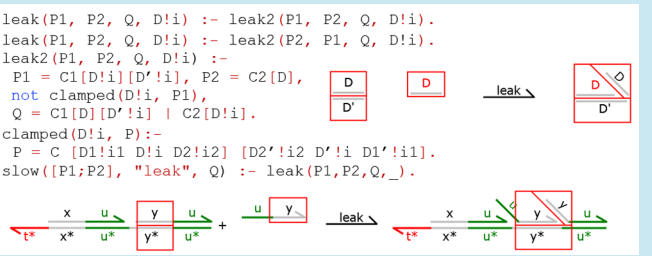
Iuliia Zarubiieva, Carlo Spaccasassi, Vishwesh Kulkarni,* and Andrew Phillips*

Cite This: https://doi.org/10.1021/acssynbio.2c00084

Read Online

ACCESS | Metrics & More | Article Recommendations | Supporting Information

ABSTRACT: Nucleic acids are a powerful engineering material that can be used to implement a broad range of computational circuits at the nanoscale, with potential applications in high-precision biosensing, diagnostics, and therapeutics. However, nucleic acid circuits are prone to leaks, which result from unintended displacement interactions between nucleic acid strands. Such leaks can grow combinatorially with circuit size, are challenging to mitigate, and can significantly compromise circuit behavior. While several techniques have been proposed to partially mitigate leaks, computational methods for designing new leak mitigation strategies and comparing their effectiveness on circuit behavior are limited. Here we present a general method for the automated leak analysis of nucleic acid circuits, referred to as *DSD Leaks*. Our method extends the logic programming functionality of the Visual DSD language, developed for the design and analysis of nucleic acid circuits, with predicates for leak generation, a leak reaction enumeration algorithm, and predicates to exclude low probability leak reactions. We use our method to identify the critical leak reactions affecting the performance of control circuits, and to analyze leak mitigation strategies by automatically generating leak reactions. Finally, we design new control circuits with substantially reduced leakage including a sophisticated proportional-integral controller circuit, which can in turn serve as building blocks for future circuits. By integrating our method within an open-source nucleic acid circuit design tool, we enable the leak analysis of a broad range of circuits, as an important step toward facilitating robust and scalable nucleic acid circuit design.



Nucleic acids are a powerful material for engineering computational circuits at the nanoscale, where interactions between nucleic acid strands can be precisely programmed by the choice of nucleotide sequence. *DNA Strand Displacement* (DSD)[1] is a well-established technique for implementing computation in both DNA and RNA,[2−4] which involves an invading single strand of DNA displacing an incumbent strand hybridized to a template. DSD is generally mediated by a short, single-stranded region of DNA referred to as a *toehold*, which is typically 4−10 nucleotides long to enable reversible binding. DSD has been used to implement a broad range of nucleic acid circuits, including combinatorial logic,[5,6] neural networks,[7] distributed consensus,[8] biochemical oscillators,[9,10] dynamic nanomachines,[11−16] reconfigurable nanomaterials,[17−20] molecular amplifiers,[21−23] and genetic riboregulators.[24,25] Such circuits show great potential for a broad range of applications, including high-precision biosensing,[26−28] molecular manufacturing,[29,30] diagnostics,[21,31−35] and therapeutics.[36−39]

Despite this potential, nucleic acid circuits are prone to *leaks*,[6,10,40] which result from unintended displacement interactions between nucleic acid strands. A leak can occur if the nucleotides at one end of a double-stranded complex spontaneously unbind, referred to as *fraying*, thus creating a very short toehold that allows a DSD reaction to take place. Although leak reactions are orders of magnitude slower than ordinary toehold-mediated DSD reactions, they typically grow combinatorially with circuit size, are challenging to mitigate,[40] and can significantly compromise circuit behavior. In some cases leaks can be used intentionally, for instance to ensure a slow and constant production of signal;[41] however, in most cases the presence of leaks is undesirable.

A number of techniques have been proposed to partially mitigate leaks, many of which have been summarized previously.[40] Sequence-level techniques include G−C bonds at the ends of double-stranded complexes to reduce the rate of spontaneous nucleotide fraying,[10,42] sequence mismatches at targeted locations to reduce the consequences of fraying,[43−46] and additional base pairs known as *clamps* at the ends of double-stranded complexes to increase the amount of fraying required for leaks to occur.[5,6,10,40] Such sequence-level approaches can be analyzed using a range of computational methods, including secondary structure prediction with NUPACK[47] and nucleotide dynamic simulation with OxDNA.[48] However, sequence-level techniques are unable to eliminate leaks in a systematic way,[40]

Received: February 16, 2022

and their overall effects on circuit behavior can be challenging to predict.

Circuit-level techniques for reducing leaks include toehold-sized clamps,[49] additional long domains in molecular cascades,[40] translocating a single nucleotide between domains using *interdomain bridging*,[50] and multiarm junction substrates to make leakage energetically unfavorable.[23] Further techniques include spatial localization of species to DNA origami[51−54] or protocells[55] to reduce interference through physical separation, threshold gates to consume leaks on inputs to logic circuits,[5,6] and creating a shadow copy of the circuit to cancel out leaks by *shadow cancellation*.[56] While each of these approaches is able to mitigate leaks in specific contexts, leak mitigation of DSD circuits in general still remains a challenge. Computational methods for designing new leak mitigation strategies and comparing their effectiveness on circuit behavior could substantially accelerate progress; however, such methods are currently limited.

A number of software tools have been developed for the analysis of DSD circuits, including Multistrand,[57] Peppercorn,[58] the DyNAMiC Workbench[59] and Visual DSD[60,61] (https://ph1ll1ps.github.io/project/visualdsd). However, their ability to analyze leak reactions has so far been limited. For example, Peppercorn cannot detect leak pathways automatically, since strand displacement in the absence of toeholds is not a part of its current enumeration semantics,[58] while Visual DSD currently only detects a limited subset of leaks involving linear polymers, which excludes most leak reactions. We refer to this version of Visual DSD as *Classic DSD* and to this limited subset of leaks involving linear polymers as *classic leaks*. Visual DSD has also recently been extended to support circuits with rich secondary structures, including branched junctions and pseudoknots,[62] together with a logic programming component referred to as *Logic DSD*[61] that supports custom logical rules for model generation. The Logic DSD component enables the design and analysis of a broad range of nucleic acid circuits, including not only DSD circuits but also circuits that incorporate nucleic acid enzymes such as the Polymerase-Exonuclease-Nickase (PEN) DNA toolbox,[63,64] as well RNA translation circuits such as ribocomputing devices.[34] However, Logic DSD does not yet support the automated generation of leak reactions.

In this paper, we present a methodology for the automated leak analysis of nucleic acid circuits, referred to as *DSD Leaks*, by building on the Logic DSD language. The paper is structured as follows. The Methods section presents an overview of the Logic DSD language, using a catalysis circuit as an example. The Results section presents the DSD Leaks methodology, including predicates for leak generation, a leak reaction enumeration algorithm and predicates to exclude low probability leak reactions, illustrated using the example catalysis circuit from the Methods. We then use DSD Leaks to analyze previous elementary control circuits for catalysis, annihilation and degradation, implemented using 2-domain and 4-domain circuit architectures, and to identify the critical leak reactions affecting circuit performance. We also analyze gain circuits that combine all three elementary circuits. We further use DSD Leaks to analyze a previous leak mitigation strategy, by automatically generating complex leak pathways that were previously manually derived while also generating additional leak pathways. Finally, we use DSD Leaks to design new elementary control circuits with substantially reduced leakage, and combine these to design a sophisticated proportional integral controller circuit that functions effectively in the presence of leaks.

METHODS

Our DSD Leaks methodology extends the Visual DSD programming language,[60] developed for the design and analysis of nucleic acid circuits. Since its conception,[65] Visual DSD has progressively been updated with new functionality over time, including most recently a logic programming language component referred to as *Logic DSD*,[61] which supports the definition of logic predicates for automatically generating a computational model of nucleic acid circuit behavior. This is achieved using logical rules in combination with nucleic acid molecular motifs to define constraints for model generation.

This section provides an overview of Logic DSD, the core foundation on which DSD Leaks is built. A summary of Logic DSD is provided in Figure 1, illustrated using a simplified version of a previously defined Catalysis circuit,[66] in which a signal *X* catalyzes the production of a signal *Y*.

The syntax of Logic DSD is based on the standard syntax of Prolog,[67,68] where logic programs are defined in terms of *logic predicates*. Each predicate has a name starting with a lower case letter and one or more arguments. A new predicate can be defined in terms of existing predicates using the (:-) operator, which indicates the existing predicates that need to be true in order for the new predicate to be true. A *logic variable* is represented by a name starting with an uppercase letter, where the *wildcard* (_) represents the logical variable that matches any term. Logic predicates are defined in terms of collections of *facts* and *rules*, which can be used in a proof search procedure to determine whether user-defined *queries* are satisfiable given those facts and rules. For example, the logic program

```
human("Socrates").
mortal(X):- human(X).
```

specifies a fact: Socrates is human, and a rule: if X is human, then X is also mortal. These can be used by a *resolution* engine to prove that Socrates is therefore mortal. Logic DSD extends the standard semantics of Prolog with an equational theory of strands.[61] The equational theory defines a *strand* in which individual domains can be logical variables, a *pattern* for matching parts of a strand, and a *context* for matching parts of a program.

Figure 1A shows the Visual DSD code for the species of the Catalysis circuit, together with their corresponding graphical representations. The Visual DSD language relies on the notion of a *domain*,[1−3] which denotes a unique nucleic acid sequence. We represent a domain by a lower-case variable and its Watson−Crick *complement* by appending the (*) character to the domain. For example, `x*` denotes the complement of `x`. We assume that a domain can only bind to its complement and indicate that two complementary domains are bound using the notation `x!i` and `x*!i`, where `i` is a unique identifier called a *bond*. A *toehold* denotes a domain that is short enough to spontaneously unbind from its complement and is represented by appending the (^) character to the domain. There are two types of *species* in the Visual DSD language: strands and complexes. A *strand* is represented as a sequence of domains enclosed in angle brackets, where the 3′ end of the strand is assumed to be on the right and is indicated by an arrowhead in the graphical representation. In the example Catalysis circuit (Figure 1A), the strand `<t^ x>` consists of the toehold `t^` followed by the domain `x` and represents the signal *X*, which denotes the catalyst. Similarly, the strand `<t^ y>` represents the signal *Y*, which denotes the output. This representation of signals is referred to as a *two-domain architecture*,[69] since each signal is represented by a strand

https://doi.org/10.1021/acssynbio.2c00084
ACS Synth. Biol. XXXX, XXX, XXX−XXX

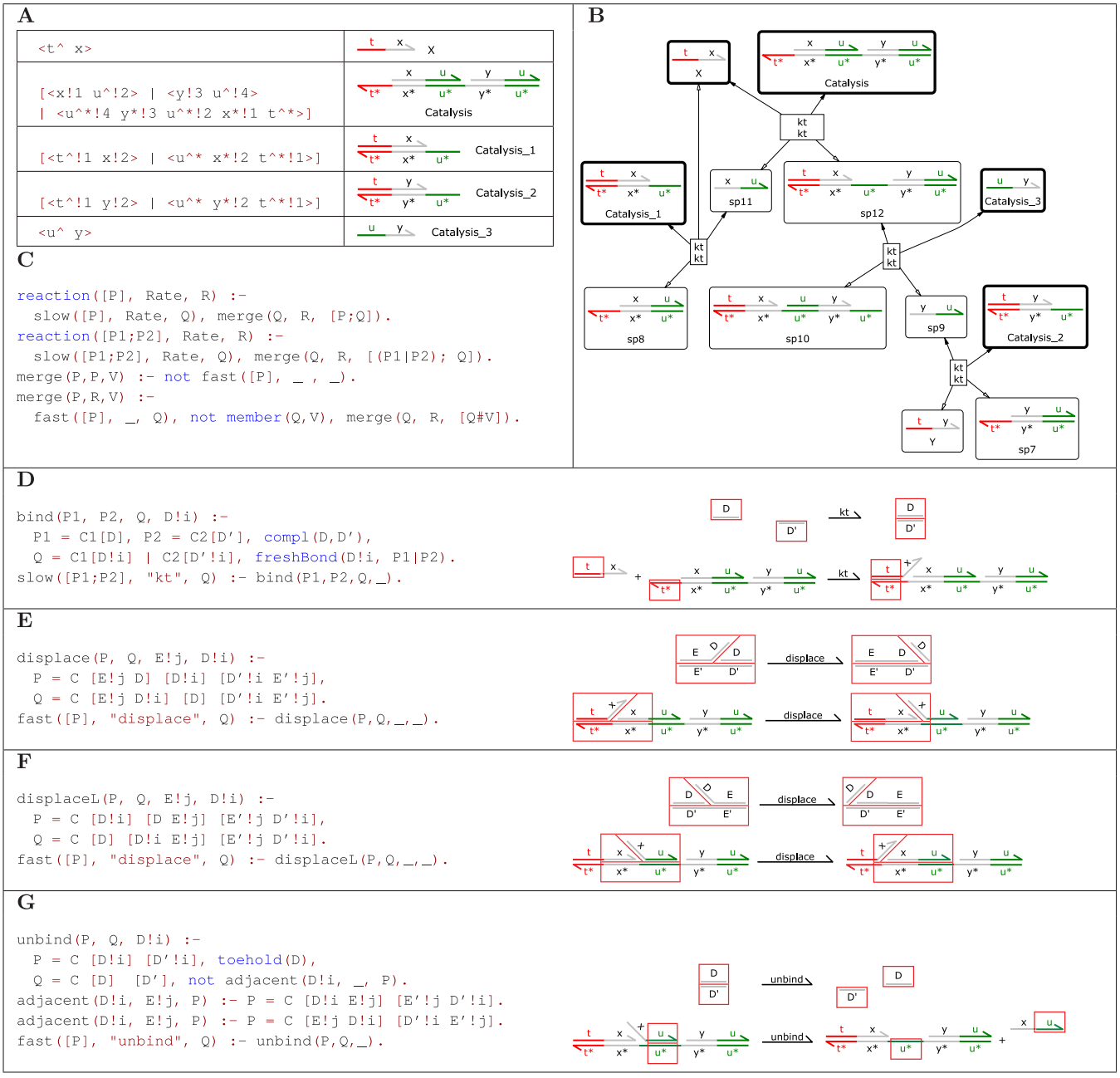


Figure 1. Logic DSD language, illustrated using a simplified version of a previously defined Catalysis circuit.[66] The circuit enables a signal *X* to catalyze the production of a signal *Y*, where built-in predicates of the Logic DSD language are highlighted in blue. (A) Visual DSD code for the species of the Catalysis circuit (left column) and their corresponding graphical representations (right column). The full Visual DSD program is shown in Figure S1. (B) Chemical Reaction Network (CRN) generated from the species in (A) according to the logic predicates defined in (C−G). (C) Logic predicates for automated CRN generation, defined using a special `reaction` predicate, which takes as arguments a list of reactant species, a reaction rate, and a product process. A reaction is generated by merging a single elementary slow reaction with a maximal sequence of elementary fast reactions, defined using the `merge` predicate, which also ensures that the same process is not visited more than once. The `slow` and `fast` predicates define elementary slow and fast reactions, respectively. Separate `reaction` rules are defined for unimolecular and bimolecular reactions. (D) The `bind` predicate states that two processes `P1` and `P2` can bind if they contain complementary domains `D` and `D'`. The product is a process `Q` in which a fresh bond `i` is shared between the two domains. (E) The `displace` predicate states that an unbound domain `D` can displace a bound domain `D!i` in the 5′ to 3′ direction if the unbound domain directly overhangs the bound domain. (F) A symmetric `displaceL` predicate defines strand displacement in the 3′ to 5′ direction. (G) The `unbind` predicate states that complementary bound domains `D!i` and `D'!i` can unbind if `D` is a toehold and if there are no adjacent bound domains on either side of the toehold, as defined by the `adjacent` predicate.

with two domains. A *complex* is represented as a collection of strands separated by a parallel composition operator (|) and enclosed in square brackets. The domains in the strands can contain bonds, such that each bond can only be shared between two complementary domains. In addition, the strands in a complex are assumed to form a *connected component*, meaning that they are all connected to each other via bonds. This simple syntax allows a broad range of structures to be encoded.[61,62] In the example Catalysis circuit (Figure 1A), the *Catalysis* complex consists of two short strands, `<x!1 u^!2>` and `<y!3 u^!4>`,

https://doi.org/10.1021/acssynbio.2c00084
ACS Synth. Biol. XXXX, XXX, XXX−XXX

bound to a longer strand <u^*!4 y*!3 u^*!2 x*!1 t^*> such that the toehold t^* is exposed. Note that bound strands are antiparallel to each other, such that the 3′ end of one strand aligns with the 5′ end of its complement, as shown in the graphical representation. The bonds are omitted from the graphical representation for conciseness since they are only needed to indicate connectivity. Together, the species *Catalysis*, $Catalysis_1$, ..., $Catalysis_3$ implement the functionality of the Catalysis circuit.

Figure 1B shows the chemical reaction network (CRN) generated from the species in Figure 1A. The CRN is generated automatically by a *reaction enumeration* method, according to the logical rules defined in Figure 1C−G. The CRN is defined as a set of *reactions*, where each reaction consists of a multiset of *reactant* species, a reaction rate, and a multiset of *product* species. The CRN is represented graphically as a bipartite graph, where nodes with rounded corners represent species and nodes with sharp corners represent reactions. Each reaction node is labeled with its reaction rate, where inbound edges represent reactants and outbound edges labeled with hollow arrows represent products. Reactions can be reversible, in which case the rate of the forward reaction is shown on top, and the reactant edges are labeled with solid arrows to indicate that the reactants of the forward reaction are also the products of the reverse reaction. In this CRN the signal *X* interacts with the *Catalysis* complex to produce intermediates *sp*11 and *sp*12, where *sp*12 can in turn interact with $Catalysis_3$ to produce an intermediate *sp*9. The intermediates *sp*11 and *sp*9 can interact with $Catalysis_1$ and $Catalysis_2$, respectively, to produce the signals *X* and *Y*, respectively. Thus, *X* produces *X* and *Y*, which is the desired behavior of the Catalysis circuit. In doing so, the species *Catalysis*, $Catalysis_1$, ..., $Catalysis_3$ are consumed as *fuel*, and the species *sp*7, *sp*8, and *sp*10 are produced as *waste*.

Figure 1C defines the top-level Logic DSD predicates that generate the CRN from the Visual DSD program. The reaction enumeration method for generating the CRN follows a previous approach[70] defined in terms of *processes*, where a process in this case is simply a multiset of strands. The `reaction` predicate is a special predicate used by the system for reaction enumeration. It takes as arguments a list of one or more processes `[P1;...;PN]` representing the reactant species, a reaction rate `R`, and a process `Q` representing the product. To generate a CRN, the reaction enumeration method first calls the `reaction` predicate on each new species or pair of species and then automatically splits the resulting process into a multiset of product species, by identifying subsets of strands that form a connected component. These steps are repeated until no new species are generated. The `reaction` predicate states that a reaction with reactant `[P]`, rate `Rate`, and product `R` can take place provided a slow reaction with reactant `[P]`, rate `Rate`, and product `Q` can take place, such that `Q` can produce `R` by a merged sequence of one or more fast reactions. And similarly for a reaction with two reactants `[P1;P2]`. The `slow` and `fast` predicates define elementary slow and fast reactions, respectively, and the `merge` predicate merges a maximal sequence of consecutive fast reactions while ensuring that the same process is not visited more than once.

Figure 1D−G defines the logic predicates that generate elementary slow and fast reactions. Each predicate is defined as Logic DSD code, where the graphical representation next to each predicate provides an informal visualization of the predicate, together with an example of its application to the Catalysis circuit. The rules encode the assumption that binding interactions between two species are considered slow, since they are limited by the relatively slow rate of molecular diffusion, while unimolecular interactions involving a single species are considered fast. This assumption is referred to as the *Infinite semantics*.[61,71] In the CRN of Figure 1B, each reaction of the CRN is generated by a slow elementary `bind` reaction followed by fast elementary `displace` and `unbind` reactions. If more detailed models are required, unimolecular reactions can also be considered slow. The case where all reactions are considered slow is referred to as the *Detailed semantics*.[61,71]

Figure 1D defines the `bind` predicate, which states that two species `P1` and `P2` can bind to each other if `P1` matches a context `C1[D]` and `P2` matches a context `C2[D']` such that `D` and `D'` are complementary, as defined by the built-in predicate `compl(D,D')`. The resulting process `Q` is obtained by adding the bond `i` to the domains `D` and `D'` in their respective contexts. Additionally, the bond `i` should not occur anywhere in processes `P1` and `P2`, which is ensured by the built-in predicate `freshBond(D!i, P1|P2)`. When this rule is applied to the example Catalysis circuit, the algorithm locates the unbound domain `t^` from process `P1` and its complement `t^*` from process `P2`. The bond `i` is then added to these domains in the resulting process `Q`. The reaction rate can either be fixed, as shown here using the rate constant "`kt`", or defined as a function of the domains and their context.[61] For the graphical representation of the `bind` predicate, the two patterns `[D]` and `[D']` on the left of the arrow are represented as two boxes with red outlines. These patterns are then updated on the right of the arrow. When the predicate is applied to the Catalysis circuit, the patterns are matched to corresponding domains in the circuit.

Figure 1E defines the `displace` predicate, which states that if a process `P` matches a context in which the sequence `D'!i E'!j` is bound to the sequence `D!i` on bond `i` and to the sequence `E!j D` on bond `j`, such that the unbound domain `D` overhangs the bound domain `D!i` and is held in place by the bound domains `E` and `E'`, then the unbound domain `D` can replace the bound domain `D!i`. The bond i is transferred to the unbound domain in the resulting process `Q`. This defines strand displacement in the 5′ to 3′ direction. For the graphical representation of the `displace` predicate, the three patterns `[E!j D]`, `[D!i]`, and `[D'!i E'!j]` on the left of the arrow are represented as three distinct regions with red outlines. These patterns are then updated on the right of the arrow. When the predicate is applied to the Catalysis circuit, the patterns are matched to corresponding domains in the circuit. Subsequent predicates in this paper are represented graphically in a similar manner. A similar `displaceL` predicate defines strand displacement in the 3′ to 5′ direction (Figure 1F).

Figure 1G defines the `unbind` predicate, which states that complementary bound domains `D!i` and `D'!i` can unbind if `D` is a toehold, as defined by the built-in `toehold` predicate, and if there are no adjacent bound domains on either side of the toehold, as defined by the `adjacent` predicate. The bond `i` is removed from both domains in the resulting product `Q`. Note that the predicate takes `D!i` as an argument, which contains both the domain `D` and its corresponding bond `i`. Since a bond can only occur twice in a process, this allows the inference system to pinpoint the specific domain on which unbinding occurs in order to test for adjacent bound domains, even if there are multiple occurrences of domain `D` in the system.

We note that the predicates shown in Figure 1 represent a subset of the predicates needed for the Catalysis circuit, where

https://doi.org/10.1021/acssynbio.2c00084
ACS Synth. Biol. XXXX, XXX, XXX−XXX

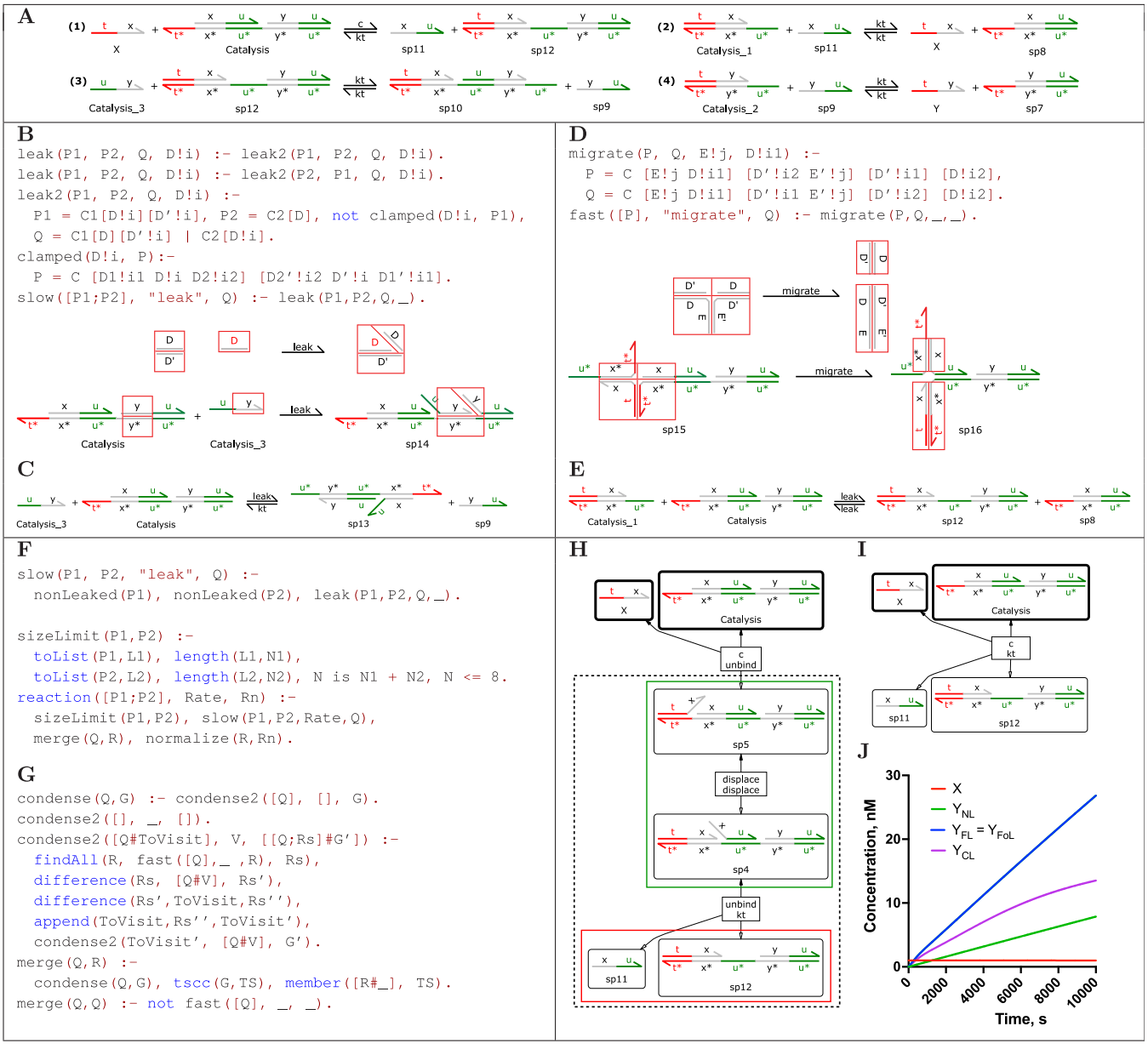


Figure 2. DSD Leaks methodology, illustrated using the Catalysis circuit from Figure 1. (A) Reactions of the Catalysis circuit, which are the same as in Figure 1 but with a slower binding rate `c` for reaction (1). (B) The `leak` predicate states that an unbound domain from one species can displace a bound domain from another species, provided the bound domain does not itself contain adjacent bound domains on both sides, as defined by the `clamped` predicate. (C) When applied to the Catalysis circuit, this allows the fuel strand to displace the output in the absence of the catalyst. (D) The `migrate` predicate states that two opposing pairs of bound domains can swap complementary domains if they are held next to each other. (E) When combined with the `leak` predicate and applied to the Catalysis circuit, this allows two fuel complexes to leak, resulting in the displacement of the output in the absence of the catalyst. (F) The number of leak reactions can be reduced by considering only *first order leaks*, defined as leaks in which both species are generated from nonleak reactions only, as defined by the `nonLeaked` predicate. A limit on the size of polymers can also be enforced, by requiring that the combined number `N` of strands in the reactants is less than a given threshold, where `toList` and `length` are built-in predicates that, respectively, turn a complex into a list of strands and compute the length of a list. (G) To improve the efficiency of leak enumeration we encoded a generalized method for merging fast reactions. (H) We illustrate this method using the Catalysis circuit. A slow elementary bind reaction is followed by fast elementary displace and unbinding reactions, represented as a reactions graph enclosed in a dashed outline. Strongly connected components (SCCs) of fast reactions are enclosed in solid boxes, with the terminal SCC shown in red. (I) The merging results in a single reaction whose products are given by the terminal SCC. (J) Simulation results of the Catalysis circuit with four different leak hypotheses: full leaks (FL, Figure S4), first order leaks (FoL, Figure S5), no leaks, (NL, Figure S3) and *classic leaks* (CL, Figure S6).

the full set of predicates is shown in Figure S1. More complex circuits, such as those involving localized strands or enzymatic activity, will require additional predicates that are beyond the scope of this paper but have that been defined in prior work.[61] In general, Logic DSD allows new predicates to be defined by the user and dynamically executed by the logic programming engine; however, most users will not be required to program logic predicates. We envisage two categories of users: regular users who use predefined logic predicates and write model code, and expert users who may wish to also change the logic predicates. The current set of built-in predicates covers a broad range of strand displacement behaviors. We also envisage

https://doi.org/10.1021/acssynbio.2c00084
ACS Synth. Biol. XXXX, XXX, XXX−XXX

different built-in predicate sets representing different modeling hypotheses, which can be loaded by the user depending on the experimental conditions, for example to model polymerase, nickase, or exonuclease enzymes, or transcription and translation.[61] Expert users can further customize or extend these predicates to model new behaviors.

RESULTS

DSD Leaks Methodology. We developed a leak analysis methodology for nucleic acid circuits referred to as *DSD Leaks* (Figure 2), building on the Logic DSD language[61] summarized in the Methods. We illustrate our methodology using the example Catalysis circuit from Figure 1, which was modified by tuning the *degree of complementarity*[4,66] of one of the toeholds, for consistency with previously defined control circuits.[66] In practice, this is implemented by introducing DNA sequence mismatches between the toehold and its complement. We encoded the degree of complementarity in Logic DSD (Figure S2) and used it to tune the degree of complementarity of the toehold `t^*` on the fuel *Catalysis*, such that binding to the catalyst *X* took place with a scaled rate *c* (Figure 2A).

Leak Predicates. We defined logic predicates to automatically generate leak reactions (Figure 2B), by leveraging the flexibility of the Logic DSD language. A leak occurs when an invading strand displaces an incumbent strand in the absence of a toehold. The `leak2` predicate states that two species `P1` and `P2` can leak if the second species contains an unbound domain `D` and the first species contains a bound domain `D!i` that is not clamped, meaning that it does not have adjacent bound domains on both the left and right, as defined by the `clamped` predicate. This allows the bound domain to fray apart at one end, creating a short toehold for the unbound domain to initiate a displacement reaction, resulting in the process `Q`. The `leak` predicate generalizes this by allowing the order of the two species to be swapped. A graphical representation of the `leak` predicate is shown below the Logic DSD code, where the domain `D` of the second species is colored in red to distinguish it from the domain `D` of the first species. An example of the `leak` predicate applied to the Catalysis circuit is also shown, resulting in an elementary leak reaction in which the $Catalysis_3$ and *Catalysis* fuels leak on the y domain.

This elementary leak reaction is then merged with a subsequent elementary unbinding reaction on the domain `u^`, resulting in a merged reaction (Figure 2C) in which the $Catalysis_3$ and *Catalysis* fuels leak to displace the *sp*9 intermediate. This can in turn interact with the $Catalysis_2$ fuel to displace the *Y* output (Figure 2A reaction 4). Together, these two reactions allow the output *Y* to be produced even in the absence of the catalyst *X*.

We defined an additional logic predicate for 4-way branch migration (Figure 2D), which is required to model key leak reactions. The `migrate` predicate states that if a species contains two opposing pairs of bound complementary domains `D` and `D'`, such that the domain `D` of the first pair is to the right of a domain `E`, and the domain `D'` of the second pair is to the left of a domain `E'` bound to `E`, then the two pairs can swap complementary domains. As a result, the domains `D`, `D'` of the first pair can bind to the domains `D'`, `D` of the second pair, respectively. A graphical representation of the `migrate` predicate is shown below the Logic DSD code. An example of the predicate applied to the Catalysis circuit is also shown, resulting in an elementary 4-way branch migration reaction in which the two pairs of bound domains `x` and `x*` exchange complementary domains.

This elementary migrate reaction is then merged with a prior elementary leak reaction between the $Catalysis_1$ and *Catalysis* fuels on the domain `t^`, and a subsequent elementary unbinding reaction on the domain `u^`, resulting in a merged reaction (Figure 2E) in which the $Catalysis_1$ and *Catalysis* fuels leak to produce waste *sp*8 and intermediate *sp*12. This can in turn bind the $Catalysis_3$ fuel to displace the intermediate *sp*9 (Figure 2A, reaction 3), which can in turn displace the output (Figure 2A, reaction 4). Together, these three reactions allow the output *Y* to be produced even in the absence of the catalyst *X*.

Beyond this specific example, the logic predicates support leak analysis of complex structures (Figure S8) including pseudoknots, which are structures with non-nested binding patterns.

Leak Approximations. Leak reactions are highly combinatorial, where even simple systems with a few dozen species can generate thousands of leak reactions (Table 1). We developed approximations to analyze systems with large numbers of leaks by removing lower probability leak reactions. This builds on previous experimental results indicating that not all leaks are equally probable.[72,73] Conceptually, suppose that species `A` is produced by a leak reaction and can also participate in a subsequent leak reaction. Since leak reactions are typically several orders of magnitude slower than toehold-mediated reactions, species `A` will likely be at low concentration and therefore have even lower probability of participating in a subsequent leak reaction. To generalize this principle, we define *second order leaks* as leak reactions for which at least one of the reactants is *leak-induced*, meaning it is directly or indirectly produced by a leak reaction. Disregarding these second order leak reactions allows us to model only *first order leaks*, which are leak reactions for which both reactants are produced in the absence of leaks. This allows us to focus on a smaller set of the most likely leak reactions. To generate CRNs with only first order leaks, we first generate the CRN in the absence of leaks and store the species of this CRN in a list. We then generate the CRN in the presence of leaks and, for each candidate leak reaction, the reaction enumeration algorithm checks if both reactants are in the list, using the `nonLeaked` predicate. If they are, the leak reaction is included. If not, this means that at least one of the reactants was produced by a leak-induced reaction, and the leak reaction is excluded. We note that the products of first order leak reactions can still participate in toehold-mediated reactions. More generally, considering only first order leaks is an approximation whose accuracy will depend on the specific circuit being considered.

In some cases, leak reactions can also give rise to polymers of connected strands that grow indefinitely, resulting in an unbounded number of reactions that cannot be modeled using a finite CRN, even if the CRN is finite in the absence of leaks. In these cases, considering only first order leaks will likely also prevent the formation of unbounded polymers, since successive leak reactions to grow the polymer will be excluded. However, in some cases a combination of first order leaks and an additional limit on the size of polymers is required for reaction enumeration to remain tractable. In addition, imposing a size limit is computationally simpler than computing first order leaks, and can be used as an alternative approximation. The size limit is implemented using the `sizeLimit` predicate (Figure 1E), which is used as an additional constraint whenever a reaction involving two species is executed. The predicate checks that the combined number of strands `N` in the reactants is less than a

https://doi.org/10.1021/acssynbio.2c00084
ACS Synth. Biol. XXXX, XXX, XXX−XXX

given threshold, where N1 is the number of strands in the first reactant and N2 is the number of strands in the second reactant. Here, the limit is set to 8, where toList and length are built-in Logic DSD predicates that, respectively, turn a complex into a list of strands and compute the length of a list.

Leak Reaction Enumeration Algorithm. To efficiently compute large numbers of leak reactions, we introduced a new reaction enumeration method to our logic programming framework, based on a previous approach.[58] The product of a reaction is represented internally as a process containing one or more species, where a species is defined as a set of connected strands. The method first performs a slow reaction and then explores all possible fast reactions from the resulting process. Fast reactions are stored in a directed graph, where vertices are processes and edges are fast reactions between processes. We call this graph the *reactions graph* and define a predicate condense(Q, G), which computes the reactions graph G from a given process Q (Figure 2G). The predicate relies on a new built-in findAll() predicate, which computes all possible solutions for an input predicate, similar to the findall() predicate in SWI Prolog.

Once the reactions graph is computed, we divide it into *strongly connected components* (SCC), defined as equivalence classes of vertices such that any two vertices in the same class are reachable from each other. We then further distinguish *terminal* SCCs, where a SCC is terminal if no other SCC is reachable from it. The SCCs partition the original graph into a directed acyclic graph (DAG) of SCCs, because if two SCCs were reachable from each other then they would form a single SCC. Therefore, the terminal SCCs are the leaves of the DAG of SCCs described by the fast reactions that a process can perform. We extended Logic DSD with new built-in predicates scc() and tscc(), which compute the SCCs and terminal SCCs, respectively. The underlying SCC algorithm is Kosaraju's algorithm,[74] implemented in F#.

Once the terminal SCCs of the reactions graph are computed, our semantics creates a new reaction for each terminal SCC, where the reactants are the input species of the slow reaction, and the product is a process from the terminal SCC. Leveraging the canonical form of species and the strand ordering used in bisimulation sorting,[61] we choose the least process in each terminal SCC. We define a predicate merge(Q, R), which merges the fast reactions from process Q, resulting in the process R (Figure 2D). The predicate makes use of the condense predicate, together with the built-in tscc predicate. In general, a process might contain multiple disconnected strands. For example, an unbinding reaction can start with a single reactant species and produce a process with two product species that are disconnected from each other. The resulting process is automatically split into multiple species, one for each set of connected strands.

An example of the reaction enumeration method applied to the Catalysis circuit from Figure 1 is shown in Figure 2H,I. Initially, the X and Catalysis species can perform a slow binding reaction at rate c to produce the process sp7. The reactions graph starting from this process is shown with a dashed outline and consists of two strongly connected components (SCCs). The first SCC consists of two processes with one species each, sp4 and sp5, while the second SCC consists of a single process with two species, sp11 + sp12, and is a terminal SCC. Figure 2H illustrates the result of merging the fast reactions, which produces a single reaction with reactants X + Catalysis and products sp11 + sp12. We then follow the same approach for each subsequent reaction. Crucially, this is performed dynamically during reaction enumeration, as opposed to at the end once the graph has been generated. This allows reaction enumeration to remain tractable.

Our reaction enumeration method is based on a previous approach,[58] which we adapted to our logic programming framework through a new merge predicate (Figure 2G). This allowed us to precisely tune the interplay between reaction enumeration and leak analysis, which underpins the extensibility of our approach. This enables the construction of the reactions graph and the terminal SCCs to be performed exactly once for each input state. In addition, rather than expanding the entire reaction network and then identifying connected components on the fully expanded CRN, we identify connected components for a single step in the reaction enumeration process, by considering a single slow reaction followed by multiple consecutive fast reactions. This is needed for the reaction enumeration method to effectively handle the generation of leak reactions, since computing the full detailed CRN with leaks when all reactions are assumed to be slow is often intractable, even for simple circuits. Furthermore, our logic programming framework allows us to customize which reactions are fast and which are slow by altering the logic predicates in the model code, which then determines which reactions are merged.

Critical Leak Analysis. Using our methodology, the automatically generated leak reactions can then be simulated to evaluate the effects of different leak hypotheses. For illustration, we simulated the Catalysis circuit with four leak hypotheses (Figure 2J): full leaks (Figure S4), first order leaks (Figure S5), no leaks (Figure S3), and *classic leaks* (Figure S6), defined as leaks generated by a previous version of Visual DSD,[71] also referred to as *Classic DSD*, that only considered linear structures with no double stranded branches. The output *Y* for the first order leak approximation is almost identical with the circuit with full leaks; however, the classic leaks circuit misses major sources of leaks (Figure S7).

Model simulation can also be used to identify which leak-induced reactions have the greatest effect on model output. To support this, we developed an automated method to disable leak-induced reactions one at a time, to identify the reactions that contribute the most to the production of a signal of interest, referred to as *critical leak-induced reactions*. The automated workflow relies on MATLAB and the SimBiology toolbox and takes as input the CRN file generated by Visual DSD. It automatically executes an ODE simulation of the CRN file with the leak-induced reactions disabled one at a time, and ranks the reactions by their effect on the production of the signal of interest (Section S2.2). The workflow is suitable for the analysis of large CRNs, where a system of 939 reactions took approximately 30 min. When applied to the Catalysis circuit under the four leak hypotheses, the analysis with full leaks identified three main critical leak reactions, including the reactions from Figure 2C,E described previously, together with a third reaction in which the fuel $Catalysis_1$ binds with the intermediate *sp*13 produced in Figure 1C. This prevents the intermediate *sp*9 from rebinding to *sp*13, which in turn increases the amount of output *Y* that will be leaked in the absence of the catalyst *X* by the reaction in Figure 1C. The same critical leak induced reactions were identified with first order leaks. However, analysis with classic leaks only identified one reaction, from Figure 2C.

Leak Analysis of Control Circuits. We used our DSD Leaks methodology to analyze previously defined elementary

https://doi.org/10.1021/acssynbio.2c00084
ACS Synth. Biol. XXXX, XXX, XXX−XXX

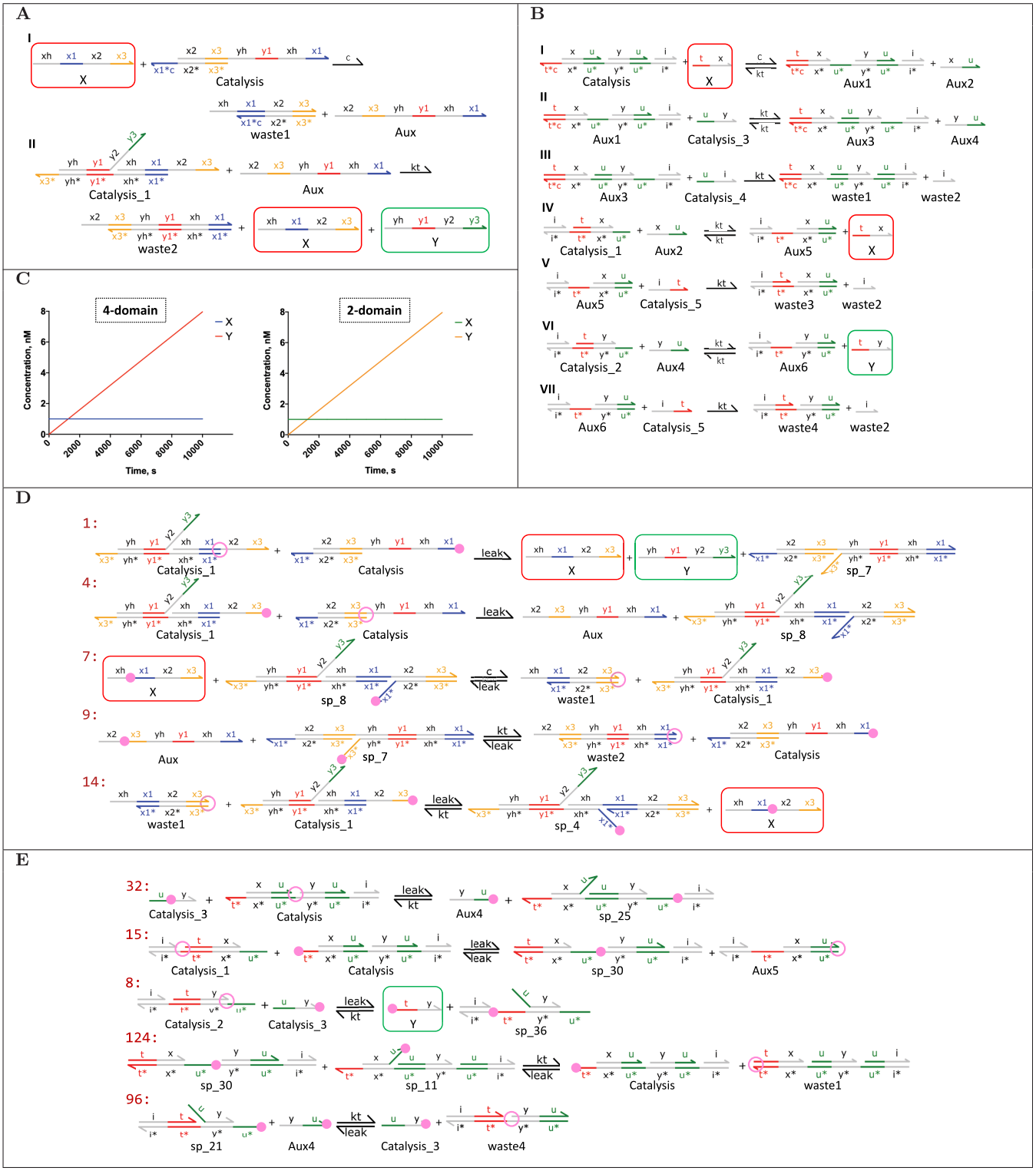


Figure 3. DSD Leaks analysis of 2-domain and 4-domain Catalysis circuits,[66] which implement the idealized reaction $X \xrightarrow{c} X + Y$. (A) Reactions for the 4-domain scheme. (B) Reactions for the 2-domain scheme. (C) Simulation of the 4-domain and 2-domain circuits in the absence of leaks, with initial concentration of signal X set to 1 nM, initial concentration of fuel species set to C_{max} = 1000 nM, toehold binding rate $kt = 10^{-3}$ nM^{-1} s^{-1}, leak rate $leak = 10^{-9}$ nM^{-1} s^{-1}, and scaled binding rate $c = 8 \times 10^{-7}$ nM^{-1} s^{-1}. (D,E) Leak-induced reactions that have the most significant effect on the behavior of the 4-domain and 2-domain circuits, respectively. Pink circles denote the initiation points of a reaction, where empty circles specify fraying sites and filled circles specify regions complementary to the fraying sites.

control circuits[66] for *catalysis*, *degradation* and *annihilation*, which can in turn be used as building blocks for complex control systems.[66,75] We further analyzed more complex *gain* circuits, which combine all three elementary control circuits. We analyzed all circuits using two representative nucleic acid implementation schemes: a 2-domain[69] scheme and a 4-

https://doi.org/10.1021/acssynbio.2c00084
ACS Synth. Biol. XXXX, XXX, XXX−XXX

Table 1. Reactions Generated by DSD Leaks for 2-Domain and 4-Domain Control Circuits[a]

	catalysis			degradation			annihilation			gain		
	R	L	S	R	L	S	R	L	S	R	L	S
4-Domain Scheme												
Full leaks	27	17	10	2	1	4	184	112	25	2046	1438	114
First order leaks	16	6	10	2	1	4	74	24	23	379	61	74
	↓41%	↓ 65%					↓60%	↓79%		↓81%	↓96%	
Classic leaks	3	1	7	2	1	4	72	44	24	90	54	42
No leaks	2	–	7	1	–	4	6	–	12	12	–	28
2-Domain Scheme												
Full leaks	553	313	41	2	1	4	188	116	24	7008	3662	179
First order leaks	197	45	37	2	1	4	76	26	22	1537	193	141
	↓64%	↓86%					↓60%	↓78%		↓78%	↓95%	
Classic leaks	144	86	41	2	1	4	72	44	24	1076	752	179
No leaks	11	–	18	1	–	4	6	–	12	30	–	47

[a]Analysis under four hypotheses: full leaks, first order leaks, classic leaks and no leaks, with maximum size limit N set to 8. S stands for *number of species*, R stands for *number of reactions*, and L stands for *number of leak only reactions*, where each reversible reaction is counted as two reactions.

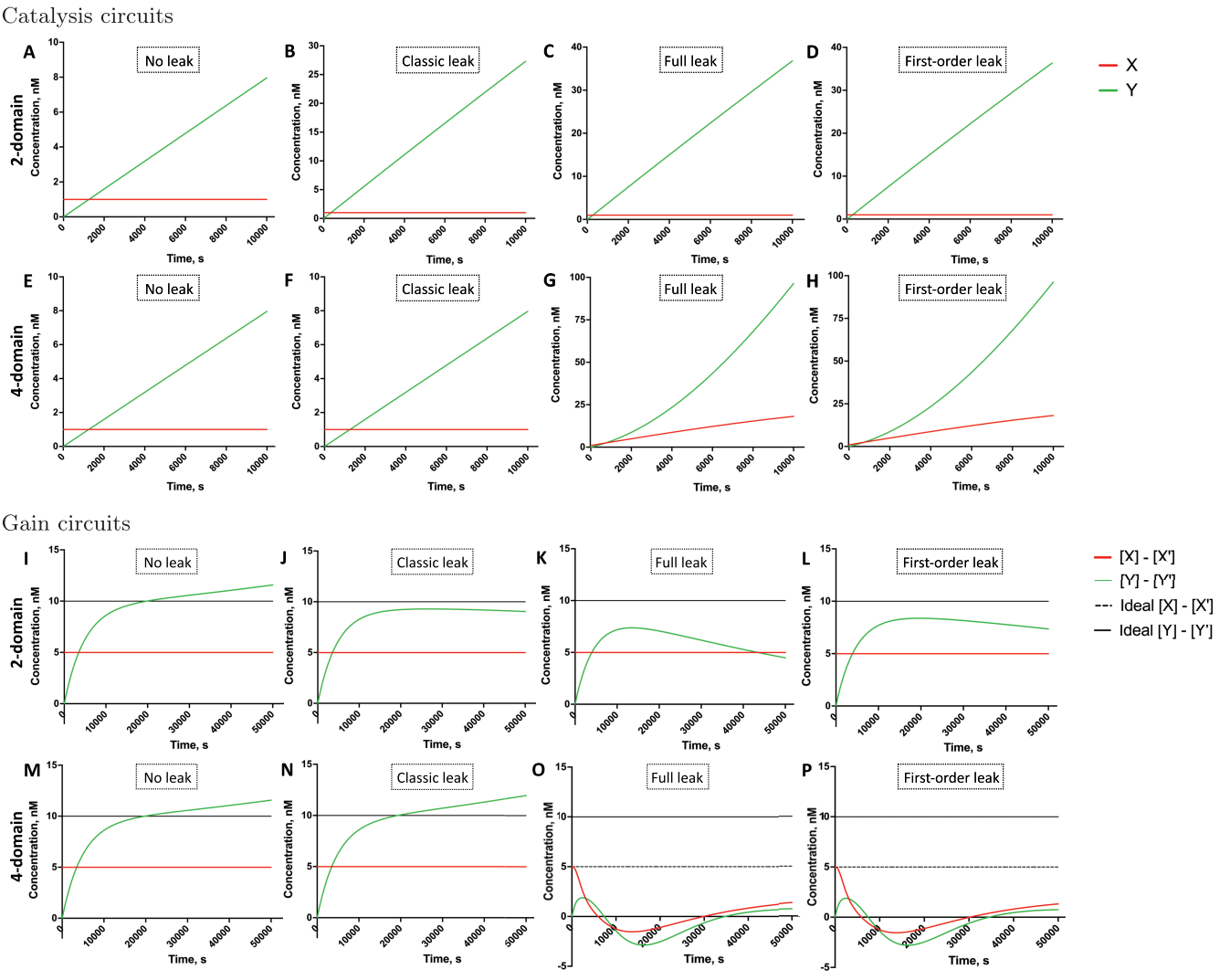


Figure 4. Simulations of 2-domain and 4-domain catalysis and gain circuits with 4 different leak hypotheses. The circuits were implemented without leaks (A,E,I,M), with classic leaks (B,F,J,N), with full leaks (C,G,K,O) and with first order leaks (D,H,L,P). All simulations used initial concentration of fuel species C_{max} = 1000 nM, toehold binding rate $kt = 10^{-3}$ nM^{-1} s^{-1}, scaled binding rate $c = 8 \times 10^{-7}$ nM^{-1} s^{-1}, leak rate $leak = 10^{-9}$ nM^{-1} s^{-1}, and maximum size limit N = 8. (A−H) Simulation of Catalysis circuits with catalyst X set to 1 nM. (I−P) Simulation of Gain circuits with input X set to 5, where X should ideally stay at the same level throughout the simulation. The gain was set to 2, such that the ideal output Y = 10.

https://doi.org/10.1021/acssynbio.2c00084
ACS Synth. Biol. XXXX, XXX, XXX−XXX

domain[4] scheme. While several versions of these schemes have previously been defined, here we used versions defined for nucleic acid control circuits.[66] We analyzed all circuits with four leak hypotheses: full leaks, first order leaks, no leaks and classic leaks.

The Catalysis circuits (Figure 3) implement the idealized reaction $X \xrightarrow{c} X + Y$, in which X catalyzes the production of Y at rate c. The 2-domain circuit is identical with the circuit in Figure 2, except that additional domains `i` allow the toehold `t^*` to be closed off by an additional helper strand `<i t^>`, which we refer to as *capping*. In the 4-domain scheme, the catalyst X is implemented as a single strand comprising four distinct domains, x_h, x_1, x_2, x_3, where x_1 and x_3 are toeholds, x_2 is the recognition domain, and x_h is the history domain. Catalysis occurs in 2 steps in the 4-domain scheme (Figure 3A). In the first step, fuel complex *Catalysis* binds to species X via toehold x_1 to displace an auxiliary species *Aux*. In the second step, another fuel complex $Catalysis_1$ reacts with species *Aux* via toehold x_3 to produce the output species X and Y. Catalysis in the 2-domain scheme occurs in 7 steps (Figure 3B), including capping. Both schemes involve additional fuel species, labeled with the prefix *Catalysis*. For all simulations of Catalysis circuits, the initial concentration of catalyst X was set to 1 nM, the initial concentration of all fuel species was set to C_{max} = 1000 nM, and the maximum complex size N was set to 8 strands, which enabled the main leak reactions to be enumerated while allowing the analysis to remain tractable.

The 2-domain Catalysis circuit generated 553, 197, and 11 reactions for the full leaks, first order leaks, and no leaks models, respectively, compared to 27, 16, and 2 reactions, respectively, for the 4-domain circuit (Table 1), where each reversible reaction was counted as two reactions. In particular, the first order leak approximation decreased the number of reactions by 41% and 64% for the 4-domain and 2-domain circuits, respectively, while maintaining almost identical simulation results compared with full leaks (Figure 4A−H). Thus, first order leaks are a good approximation for the Catalysis circuits.

Although the 4-domain circuit generated substantially fewer leak reactions overall compared to the 2-domain circuit, the performance of the 4-domain circuit was severely compromised for both full leaks and first order leaks (Figure 4G,H). This is likely due to the presence of long overhanging strands, which are not present in the 2-domain circuit. We note that a number of modifications can be made to the 4-domain scheme to further reduce leaks,[10,40,72] both at the domain level, such as through the use of clamps, and at the nucleotide level, such as through optimized sequence design methods. For future circuit design, our leak analysis methodology can be used to compare different leak mitigation strategies at the domain level and identify potential areas of intervention.

Given the similarity between the full leak and first order leak simulations, a critical leak analysis was performed using first order leaks for increased computational efficiency. The critical leak-induced reactions for the 4-domain circuit were identified as reactions 1, 4, 7, 9, 14 (Figure 3D, Figure S13), while the critical leak-induced reactions for the 2-domain circuit were identified as reactions 8, 15, 32, 96, 124 (Figure 3E). Interestingly, reaction 8 was not critical for the 2-domain Catalysis circuit in the absence of capping (Figure 2), since the leaked output was able to quickly rebind. However, in the presence of capping, the caps quickly seal off exposed `t^*` toeholds, meaning the leaked output is much less likely to rebind. This illustrates the subtle interplay between circuit design and critical leaks. Most of the critical leak-induced reactions for both circuits relied on the formation of intermediate branched structures and hence were not captured by classic leaks, which produced substantially different behavior (Figure 4E−H). In addition, 3 out of 5 critical leak reactions in the 4-domain circuit involved the catalyst X (Figure 3D), whereas no critical leak reactions involved the catalyst in the 2-domain circuit (Figure 3E). Hence, although the 2-domain circuit generated more reactions than the 4-domain circuit, its reactions had less effect on the production of output (Table S2A,B). While it was possible to generate reactions with full leaks enabled for both circuits, in general the first order leak approximation allows critical leak reactions to be identified even for circuits where the generation of full leaks is intractable.

We further analyzed 2-domain and 4-domain Annihilation and Degradation circuits (see Section S3.2 for additional details). The annihilation circuits were highly prone to leaks, while the degradation circuits were relatively leak-resistant. The leak profiles of the 2-domain and 4-domain schemes were similar for both circuits (Table S2, Figure S14).

We then analyzed Gain circuits, which map an input X to an output Y as $Y = KX$, where K is the gain. This corresponds to two idealized catalysis reactions $X \xrightarrow{K \cdot c} X + Y$ and $X' \xrightarrow{K \cdot c} X' + Y'$, two annihilation reactions $X + X' \xrightarrow{kt}$ and $Y + Y' \xrightarrow{kt}$ with $kt \gg c$, and a degradation reaction $Y \xrightarrow{c}$, where X and X' represent the positive and negative components, respectively, of a real-valued signal. The gain circuit can be used either independently or as a component of more complex circuits, such as circuits for computing complex functions,[76] proportional-integral controllers,[66] or linear classifiers.[77]

The 2-domain Gain circuit generated 7008, 1537, and 30 reactions for the full leaks, first order leaks and no leaks models, respectively, compared to 2046, 370, and 12 reactions, respectively, for the 4-domain circuit (Table 1, Table S13). The first order leak approximation decreased the number of reactions by 81% and 78% for the 4-domain and 2-domain circuits, respectively, while maintaining similar but still noticeably different simulation results compared with full leaks (Figure 4I−P). We note also that the percentage reduction increased with circuit complexity, compared with the simpler Catalysis circuits. Overall, the 2-domain scheme appears more favorable for building complex circuits such as the Gain circuit (Figure 4).

We also analyzed the effect of increasing the reaction rate c for the Catalysis circuit via increasing the degree of complementarity (Figure S15). Increasing c to its maximum of kt enabled the 2-domain scheme to mitigate leaks almost completely, whereas the 4-domain scheme continued to exhibit a high degree of leaks. However, increasing c has its limitations: in control circuits, the degree of complementarity was introduced to ensure that annihilation is significantly faster than catalysis and degradation. Increasing the value of c will increase the catalysis reaction rate, bringing it closer to the annihilation reaction rate, which will result in less accurate control. Moreover, the system will run out of fuel more quickly, which will in turn reduce its operating time. As a result, increasing c in catalysis and degradation substantially reduces gain circuit performance (Figure S16).

Analysis of Leak Reduction Strategies. We used our DSD Leaks methodology to analyze leak reduction strategies whose leak reactions were previously manually derived.[40] A *Translator* circuit that translates a signal X into a signal Y was implemented using three schemes—*Single-Long Domain* (SLD),

https://doi.org/10.1021/acssynbio.2c00084
ACS Synth. Biol. XXXX, XXX, XXX−XXX

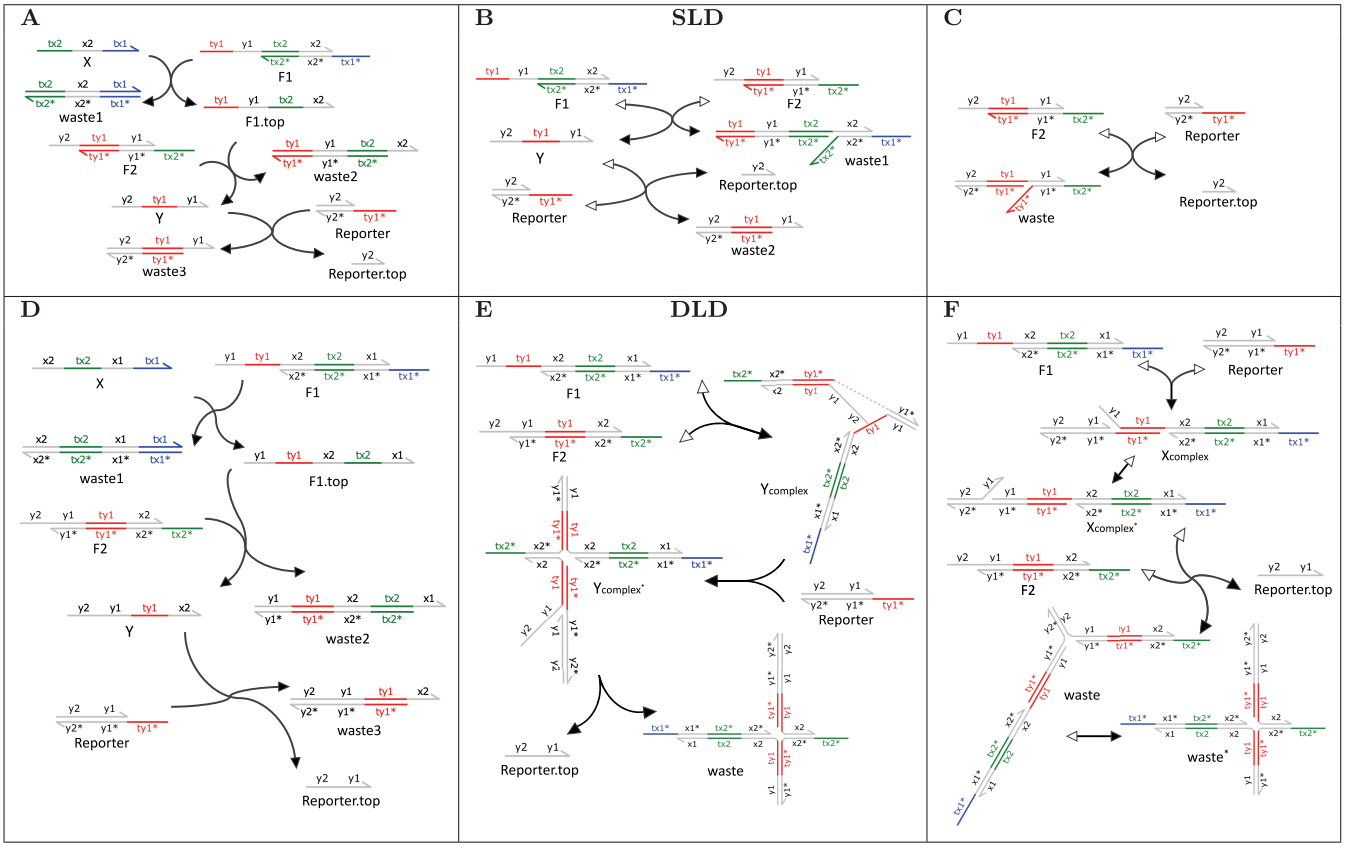


Figure 5. DSD Leaks analysis of previously proposed leak reduction strategies.[40] Our method automatically generated previously reported and additional leak reactions. (A–C) Single Long Domain (SLD) Translator, showing the intended pathway (A) and previously reported leak pathways (B,C). (D–F) Double Long Domain (DLD) Translator, showing the intended pathway (D), a previously reported leak pathway (E) and an additional leak pathway (F). Overall, 22 reactions that produced the *Reporter.top* strand were generated automatically (Figure S19).

Double-Long Domain (DLD), and *Triple-Long Domain* (TLD)—where increasing the number of long domains was shown to substantially reduce leaks.[40] We used our methodology to analyze the SLD and DLD Translator circuits (Figure 5, Section S4.1) by automatically generating their leak reactions.

The SLD Translator circuit (Figure 5) is almost identical with the 4-domain Catalysis circuit (Figure 3D), except that the history of the X signal is omitted, the domains are numbered from 3′ to 5′ instead of the other way around, and only a single output *Y* is produced instead of two outputs *X* and *Y*. The automatically generated reactions (Figure S17) include the intended reactions (Figure 5A), together with a leak reaction in which the two fuel complexes *F*1 and *F*2 interact due to fraying of the bound domains `y1` and `y1*` on *F*2 (Figure 5B). This results in the production of the output *Y*, which can further react with the *Reporter* complex to produce a detectable signal *Reporter.top*. In addition, a direct leak reaction between *F*2 and *Reporter* results in the release of *Reporter.top* (Figure 5C).

To reduce these leaks, the DLD scheme encodes the input signal *X* using two long domains instead of one, represented as `<x2 tx2^ x1 tx1^>`, where the fuel complexes are amended accordingly. The intended reactions of the DLD Translator (Figure 5D) are similar to those of the SLD Translator. However, the leak reaction between the two fuel complexes *F*1 and *F*2 differs in that the unbound domains `<y1 ty1^>` in *F*1 can now only partially displace the output strand from *F*2, since the output strand remains bound by a long domain `x2` (Figure 5E). For a leak to occur, the resulting $Y_{complex}$ must further react with the *Reporter* complex to produce $Y^*_{complex}$, which can then release *Reporter.top*. This requires the use of *Detailed mode*, in which all reactions are considered slow, since if displacement is considered fast, then $Y_{complex}$ always dissociates before the *Reporter* has a chance to bind, meaning that $Y^*_{complex}$ never forms. Detailed mode also results in different conformations of $Y^*_{complex}$ being represented as different species (Figure S20), where a displacement or 4-way branch migration reaction is required to transition from one conformation to another. Since we do not know in advance which conformations will give rise to leaks, all conformations need to be considered.

The use of Detailed mode for the DLD Translator also allows complexes of potentially unbounded size to form. As a result, a limit on the size of complexes is required, as defined in Figure 2F. The limit was set to 6, to generate the main leak reactions while ensuring that the analysis remained tractable. Our methodology identified 22 leak reactions in which the *Reporter.top* strand was produced (Figure S19). These included leak pathways that were previously manually derived[40] together with additional leak pathways, such as a pathway in which *F*1 interacts with *Reporter* and the resulting product then interacts with *F*2 (Figure 5F). For comparison, we also generated the leak reactions for the SLD Translator in Detailed mode (Figure S18). These included the higher probability leaks of Figure S17, together with additional lower probability leaks that required the formation of transient complexes only present in Detailed mode. Overall, 39 reactions

https://doi.org/10.1021/acssynbio.2c00084
ACS Synth. Biol. XXXX, XXX, XXX−XXX

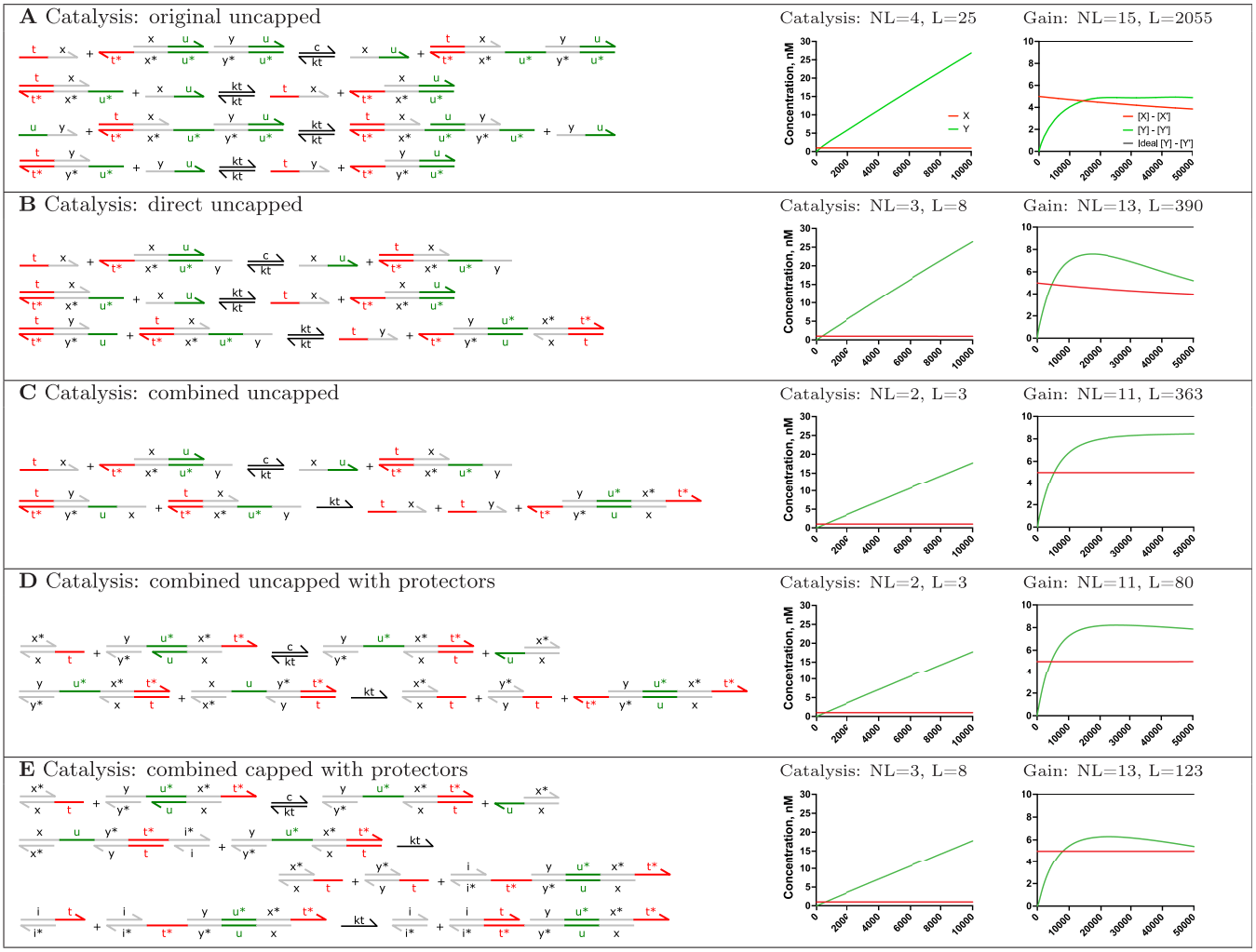


Figure 6. Application of DSD Leaks to reduce leakage of the 2-domain Catalysis circuit. NL and L denote the number of reactions with no leaks and full leaks, respectively. Plots show full leak simulations, where Catalysis plots show input X (red) and output Y (green), and Gain plots show the ideal output (black), the difference $X-X'$ between the catalyst and its complement (red) and the difference $Y-Y'$ between the output and its complement (green). Simulations for the proposed circuits in the absence of leaks are shown in Figure S21. (A) Catalysis circuit from Figure 2A, equivalent to the original catalysis circuit[66] but without caps. (B) Direct displacement of output following binding of catalyst. (C) Combined displacement of output and catalyst following binding of catalyst. (D) Combined displacement of output and catalyst with protector strands. (E) Combined displacement of output and catalyst with protector strands and caps.

that produced the *Reporter.top* strand were enumerated for the SLD Translator.

The fact that *Detailed mode* was required to detect the DLD Translator leaks is consistent with the low probability of these leaks, which were shown to be an order of magnitude lower than the SLD leaks.[40] However, moving from Infinite to Detailed mode also substantially increased the number of generated reactions, resulting in increased computational cost. As a result, reaction enumeration of the TLD scheme,[40] in which signals are represented using three long domains, did not terminate. Additional work is therefore needed to increase the computational efficiency of the reaction enumeration algorithm. In the future, our logic programming framework could also be used to encode reaction enumeration approaches that make Detailed mode more tractable, such as considering migrate, displace, and unbind reactions as fast relative to leak reactions. Despite the current limitations of Detailed mode, in practice we can still use Infinite mode to detect the primary sources of leaks and to inform corresponding leak reduction strategies.

More generally, while manual analysis is valuable for circuit design,[40] it does not scale to complex circuits due to the combinatorial explosion in the number of leak reactions, meaning that important leak pathways could potentially be missed. An automated approach is therefore beneficial for the design and simulation of more sophisticated circuits, as illustrated below.

Leak Reduction of Control Circuits. Building on our leak analysis of existing control circuits, we used our DSD Leaks methodology to perform multiple iterations of circuit design and analysis, in order to substantially reduce control circuit leaks (Figures 6–8).

We focused initially on the 2-domain Catalysis circuit of Figure 3, which was previously used as a key component of a proportional integral (PI) controller circuit.[66] Catalysis has also been highlighted as an important component for scalable nucleic acid circuits more generally.[6] We first investigated the effects of removing the `<i t^>` fuel and the bound `i` and `i*` domains, which together allow the exposed `t^*` toeholds to be *capped* following the displacement of the X and Y signals. This reduces

https://doi.org/10.1021/acssynbio.2c00084
ACS Synth. Biol. XXXX, XXX, XXX−XXX

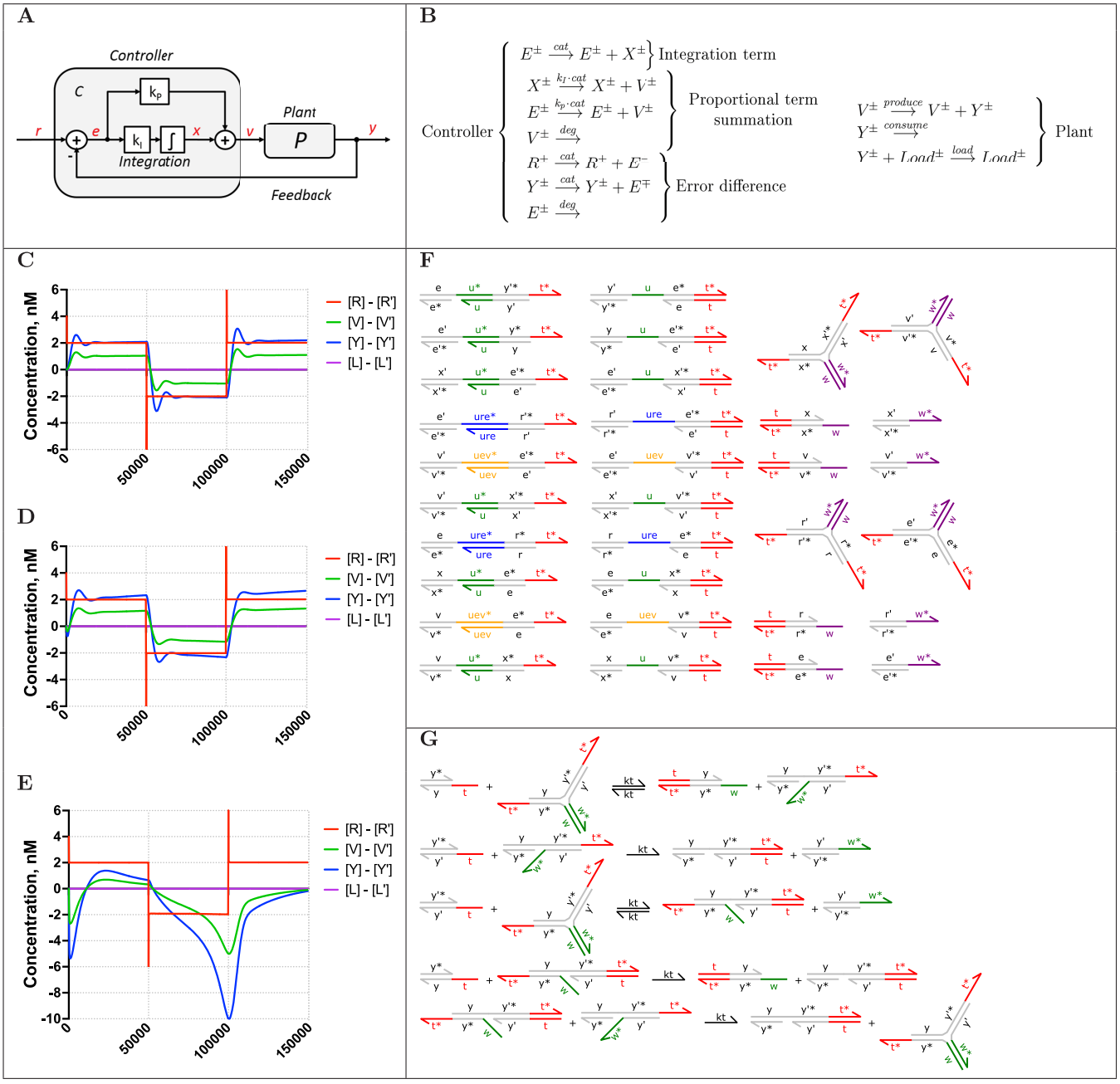


Figure 7. DSD Leaks analysis of an optimized proportional integral (PI) controller. (A) Block diagram of the PI controller and a production Plant with input v and output y, reproduced from previous work.[66] The PI controller automatically adjusts the input of the Plant so that the output tracks the reference r, by computing a weighted sum of the error e and its integral, where e is the difference between the output and the reference, and K_p and K_i are the weight parameters. (B) Idealized chemical reaction model of the PI controller and Plant,[66,75] where each idealized chemical reaction is implemented as a Catalysis, Annihilation, or Degradation circuit. X^+ and X^- denote the positive and negative components of a real-valued signal, and the reaction $X^{\pm} \rightarrow X^{\pm} + V^{\mp}$ is short for two separate catalysis reactions $X^+ \rightarrow X^+ + V^-$ and $X^- \rightarrow X^- + V^+$. The plant produces the output Y from the input V at a constant rate, where the output is also consumed at a constant rate. (C) Simulation of the optimized PI controller tracking a reference signal R that switches over time. (D) Simulation of the optimized PI controller with full leaks enabled. (E) Simulation of a variant of the optimized PI controller with additional *caps* and with full leaks enabled. The introduction of caps disrupts the functioning of the circuit in the presence of leaks. (F) Initial conditions for the optimized PI controller. Additional toeholds were introduced for four of the Catalysis circuits to prevent transient binding of the combined gates. (G) Reactions for the optimized Annihilation circuit used in the PI controller. The circuit enables cooperative displacement via reversible strand exchange.

the concentration of exposed `t^*` toeholds, which in turn reduces potential interference and limits the rebinding of displaced X and Y signals, increasing the forward bias of the circuit.

We analyzed the Catalysis circuit without caps (Figure 6A), which also coincides with the Catalysis circuit in Figure 2A. This produced 25 reactions with full leaks enabled, compared to 553 reactions for the original Catalysis circuit. Simulation of the circuit also revealed reduced leakage. We further combined the circuit with an optimized Annihilation circuit designed using our methodology (Figure 7G, Figure S23), in order to form a Gain circuit. This produced 2055 reactions with full leaks enabled,

https://doi.org/10.1021/acssynbio.2c00084
ACS Synth. Biol. XXXX, XXX, XXX−XXX

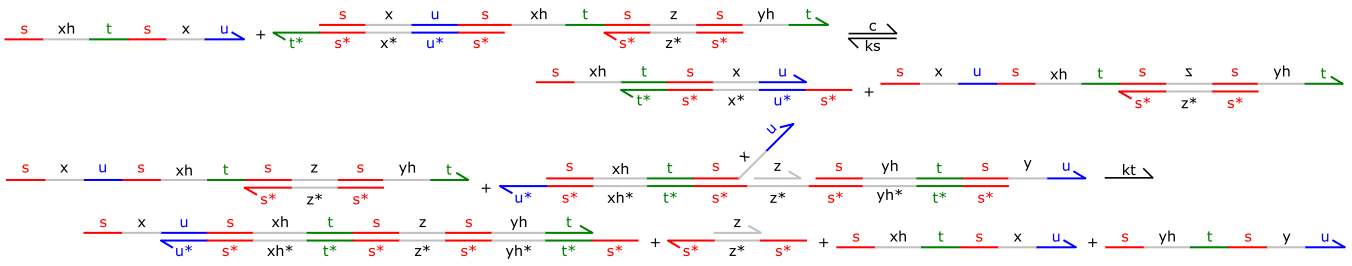


Figure 8. Application of DSD Leaks to reduce leakage of the 4-domain Catalysis circuit. The design relies on a combination of toehold clamps and protector strands, and does not exhibit any leaks. Its performance is therefore the same as that of the original circuit (Figure 3C).

compared to 7008 reactions for the original Gain circuit. However, simulation performance was reduced, suggesting that caps provide functional benefit for the Gain circuit.

We hypothesized that leaks could be further reduced and performance improved by allowing the catalysis complex to directly displace the output after binding the catalyst, without the need for additional excess helper (Figure 6B). This gave rise to a Catalysis circuit with only 8 reactions and a Gain circuit with only 390 reactions and improved performance. However, a gradual decrease of the catalyst remained an issue, due to reduced forward bias caused by rebinding of the catalyst.

We hypothesized that leaks could be further reduced and the forward bias of reactions improved by allowing the catalysis complex to simultaneously displace both the catalyst and output using a combined gate (Figure 6C), without the need for additional fuel. This resulted in a Catalysis circuit with only 3 reactions and a Gain circuit with only 363 reactions, with significantly improved performance.

We then hypothesized that leaks could be further reduced in the Gain circuit, and more generally in any combination of elementary control circuits, by using *protector* strands to cover the long domains of signals (Figure 6D). The main principle behind protector strands is that, in order for a leak to occur, fraying needs to take place in two double-stranded complexes simultaneously, which has a much lower probability than fraying of a single complex. This resulted in a Catalysis circuit with only 3 reactions and a Gain circuit with only 80 reactions and good performance. A potential disadvantage of protectors is that they rely on 4-way branch migration, which is known to be slower than 3-way branch migration. Our models used a standard assumption that both 4-way and 3-way branch migration are fast compared to binding; however, further investigation would be beneficial.

We also explored a design in which caps were reintroduced (Figure 6E), but this led to increased leaks and reduced performance of the Gain circuit. Several other designs were also explored (Figure S22), but the design with the fewest leaks remained the uncapped circuit with a combined gate and protectors (Figure 6D).

We performed a similar iterative design process for the Annihilation circuit and converged on a branched design (Figure 7G, Figure S23), which produced 5 reactions with full leaks enabled, compared to 188 reactions for the original Annihilation circuit. The design was partly inspired by previous cooperative displacement circuits,[78] with the key difference that here we designed reversibility of reactions using toehold mediated strand exchange. This allows the reverse rate of the reaction to be tuned by changing the concentration of reverse complexes, whereas cooperative displacement circuits[78] require direct tuning of the toehold unbinding and branch migration rates. Furthermore, cooperative displacement assumes that toehold unbinding is slow, in order to allow the second input to bind before the first one unbinds. This substantially complicates the leak analysis, since a more detailed reaction enumeration mode is required to detect potential leaks involving the intermediate complexes. To mitigate this, it is possible to assume that toehold unbinding is only slow for the annihilation circuit but not for the other circuits, for instance due to the increased degree of complementarity. Such mixed assumptions are possible in the Logic DSD framework by further refining the model generation rules, which could be the subject of future work.

Finally, we used our optimized Catalysis (Figure 6D) and Annihilation (Figure 7G) circuits to implement a proportional integral (PI) controller circuit (Figure 7), based on a previous design[66,75] (Figures S25, S26). The PI controller consists of 10 Catalysis, 4 Annihilation, and 2 Degradation circuits (Figure 7F), where additional toeholds were introduced for four of the Catalysis circuits to prevent transient binding of the combined gates. The circuit generated 1187 reactions with full leaks enabled, compared to 7008 reactions for the original Gain circuit alone, where leak analysis of the original PI controller circuit failed to terminate due to the large number of leaks. Simulation of the optimized PI controller in the absence of leaks produced almost identical results to the original design,[66] where the controller was able to regulate the input V of a production Plant such that the output Y tracked a reference R. When full leaks were enabled, the optimized PI controller continued to function with only a minor reduction in performance (Figure 7D). For comparison, we also simulated an alternative PI controller design (Figure 7E) that used a Catalysis circuit with caps (Figure 6E). However, even this minor change was sufficient to completely disrupt PI controller function when full leaks were enabled. Leak analysis of PI controller circuits using other Catalysis circuits from Figures 6 and S22 failed to terminate, due to the large number of leaks.

We also investigated the effect of the complex size limit on the PI controller circuits (Section S5.2). Removing the size limit altogether for the optimized PI controller circuit resulted in 5633 reactions compared to 1187 for a size limit of 8, with no observable difference in simulation results (Figure S24). This suggests that a size limit of 8 is a good approximation that substantially reduces the number of reactions without significantly affecting circuit performance.

We also applied our leak analysis methodology to the 4-domain Catalysis circuit (Figure 8), by introducing toehold clamps at strategic locations, inspired by previous work.[10,40,49] A key difference between the 2-domain and 4-domain designs is that reactions in the 2-domain design can proceed in two directions, from 5′ to 3′ and vice versa, which means that some leakage is inevitable for at least one of the directions, even when clamps are introduced. However, 2-domain circuits have the

https://doi.org/10.1021/acssynbio.2c00084
ACS Synth. Biol. XXXX, XXX, XXX−XXX

Table 2. Overview of Existing Approaches for Mitigating Leaks, Together with Some Advantages and Challenges

approach	advantages	challenges
G−C bonds at ends of complexes[10,42]	Effective leak reduction by reducing fraying; Maintains circuit design at the domain level.	Used in specific contexts, may not eliminate all leaks.
Base pair mismatches[43−46]	Maintains circuit design at the domain level; Substantial leak reduction with modestly decreasing rates in catalytic hairpin assembly;[43] Identification of target areas for leak reduction and quantification of secondary structure effects,[45] allowing a 4-fold leak reduction.	May slow down circuit behavior; Requires careful tuning of circuit; Used in specific contexts.
Clamps[5,6,10,40,49]	Successfully implemented in complex DSD circuits; Simple experimental implementation.	Used in specific contexts, may not eliminate all leaks; May require modification of circuit design.
Molecular cascades with additional long domains[40]	Additional domains can be introduced to further reduce leaks; Capable of reducing leaks to arbitrarily low levels at the cost of increased complexity.	Validated for translator circuits, further work required for AND circuits with two inputs.
Interdomain bridging[50]	Helps to eliminate toehold-independent leakages while enhancing strand displacement kinetics across a three-way junction.	Used in specific contexts, may not eliminate all leaks.
Multiarm junctions with four-way branch migration[23]	Increases leakage energy barrier through four-way branch migration; Potential to design complex, robust, and efficient molecular systems.	Limitations for implementing some circuit types such as AND gates.
Shadow cancellation[56]	Does not require modification of existing circuit; Agnostic to leak mechanism; Works in conjunction with other leak mitigation strategies.	Requires a shadow copy of the circuit and additional cancellation complexes; Requires a sufficiently good match between the primary and shadow circuits; Slows down circuit behavior.
Thresholds[5,6]	Mitigates the effects of initial leakage; Works well for digital logic circuits.	Limited application to continuous leakage in dynamic circuits; Requires sufficiently strong binding to threshold, may not eliminate all leaks.
Spatial localization[51−55]	Uses spatial separation to reduce interference; Allows for faster circuit dynamics and reduced leakage between separated components.	Higher experimental complexity required to achieve localization; Barriers to communication between locations may affect circuit behavior.
Enzymatic DNA synthesis[8,79]	Reduces leakage due to malformed complexes caused by synthesis errors. Enzymatically prepared DNA hairpins exhibit substantially reduced leakage compared to chemically synthesized hairpins.[79]	Specific to certain architectures amenable to enzymatic synthesis; Increased complexity of experimental preparation.
Hybrid DNA/Locked Nucleic Acid (LNA) systems[80]	High leakage rate reduction (over 50-fold) while maintaining high circuit performance; Maintains original circuit architecture.	Requires synthesis of hybrid oligonucleotides; Requires care when incorporating LNA due to their strong binding affinity.

advantage that they contain fewer single-stranded overhangs, making them generally less prone to leaks. They are also amenable to plasmid production in cells, which has previously been shown to reduce leaks due to imperfections that can occur when strands are synthesized using chemical methods.[8] In contrast, 4-domain circuits can be designed to proceed in a single direction. This allows clamps to be used to prevent leakage prior to displacement. Unlike previous Translator circuits,[40] Catalytic circuits require two outputs to be produced. Following multiple iterations of our leak analysis methodology, including several failed designs, we converged on a design in which protector strands were used to prevent leakage from exposed single-stranded overhangs. This resulted in a Catalytic circuit with no leaks (Figure 8). This scheme also allows the same toeholds *t*, *u*, and *s* to be shared across all signals, where long domains are used for specificity. In addition, the scheme supports an Annihilation circuit similar to the 2-domain design of Figure 7G, resulting in 15 reactions with full leaks enabled, compared to 184 for the original 4-domain Annihilation circuit. Future work could investigate how to further reduce leaks for 4-domain circuits with two inputs, for instance based on previous approaches.[72]

CONCLUSIONS

In this paper, we presented the DSD Leaks methodology for automating the leak analysis of nucleic acid circuits. We extended the Logic DSD language[61] with predicates for leak generation, a leak reaction enumeration algorithm, and predicates to exclude low probability leak reactions, which support the analysis of complex circuits with large numbers of leaks. We used DSD Leaks to analyze previous control circuits implemented with 2-domain and 4-domain architectures under four different leak hypotheses, and to analyze a leak mitigation strategy that was previously analyzed by hand, resulting in the automated generation of both existing and new leak pathways. We further used DSD Leaks in an iterative cycle of design, simulation, and analysis to develop new elementary control circuits with substantially reduced leakage, which were then combined to design a sophisticated proportional integral controller circuit that functioned effectively in the presence of leaks. These elementary control circuits could in the future be used to design more complex circuits with reduced leakage. Catalysis in particular has been highlighted as an important component of scalable nucleic acid circuits more generally,[6] where catalysis circuits with reduced leakage could enable increased speed and scalability of larger circuits. We integrated DSD Leaks with the open-source nucleic acid circuit design tool Visual DSD, to aid in the future design of a broad range of complex nucleic acid circuits that are robust to leaks.

Table 2 provides a partial summary of existing leak mitigation strategies, together with some of their advantages and challenges. Most strategies can be directly modeled at the circuit level and can therefore be analyzed using DSD Leaks, including an explicit representation of clamps,[5,6,10,40,49] molecular cascades with additional long domains,[40] Interdomain bridging,[50] multiarm junctions with 4-way branch migration,[23] shadow cancellation,[56] thresholds,[5,6] and spatial localization.[51−55] Other strategies require intervention at the sequence level, including G−C bonds at the ends of complexes[10,42] and base pair mismatches.[43−46] While these

https://doi.org/10.1021/acssynbio.2c00084
ACS Synth. Biol. XXXX, XXX, XXX−XXX

cannot be modeled explicitly in DSD Leaks, approximations are possible in some cases to account for the effects of sequence level interventions, for instance by changing the degree of complementarity of a domain to account for sequence mismatches, or by introducing logical rules that modulate the effects of sequence on kinetic rates. For example, the leak predicate `leak(P1,P2,Q,D!i)` could in the future be updated so that the leak rate is dependent on the domain `D` by using a rate lookup predicate, which assigns different leaks rates based on whether the domain is known to have A−T or G−C bonds where the fraying takes place. Reduced leakage resulting from enzymatic synthesis can also be modeled at the circuit level.[8] In addition, hybrid DNA/LNA (Locked Nucleic Acid) systems that exhibit reduced leakage[80] could in the future be modeled using the built-in tag mechanism of Logic DSD to tag LNA strands, in combination with revised logical predicates to encode leak generation rules for these hybrid systems. More generally, we anticipate a combined approach in which DSD Leaks can aid in the reduction of leaks through effective circuit-level designs, while sequence-level and experimental techniques can be used to further reduce the remaining leaks in ways that are compatible with the design of the circuit.

Future work will aim to increase the efficiency of the reaction enumeration implementation of DSD Leaks, and to encode additional rules to provide approximations for more detailed semantics that require explicit analysis of transient complexes. Future work will also seek to analyze additional leak mitigation strategies and to compare computational analysis with experimental implementations. In particular, experimental implementations of the proposed control circuits with reduced leakage remain to be investigated.

Overall, our method enables the leak analysis of a broad range of nucleic acid circuits, from purely DSD circuits with complex topologies to circuits that incorporate DNA enzymes or RNA translation. It also provides a logic programming framework in which reaction generation rules can be flexibly encoded alongside circuit designs, and readily extended as new types of nucleic acid circuit implementation strategies are developed in the future.

■ ASSOCIATED CONTENT

Supporting Information

The Supporting Information is available free of charge at https://pubs.acs.org/doi/10.1021/acssynbio.2c00084.

Additional DSD data files (ZIP)

Logic DSD method; DSD Leaks methodology; Leak analysis of control circuits; Analysis of leak reduction strategies; Leak reduction of control circuits; CRNs for elementary reactions under no-leak and first order leak implementations (PDF)

Special Issue Paper

Invited contribution from the 13th International Workshop on Bio-Design Automation.

■ AUTHOR INFORMATION

Corresponding Authors

Vishwesh Kulkarni − *AiM Macromed Ltd., London WC2H 9JQ, U.K.*; Email: vishwesh@aimacromed.com

Andrew Phillips − *Microsoft Research, Cambridge CB1 2FB, U.K.*; orcid.org/0000-0001-9725-1073; Email: andrew.phillips@live.com

Authors

Iuliia Zarubiieva − *University of Warwick, Coventry CV4 7AL, U.K.*

Carlo Spaccasassi − *Microsoft Research, Cambridge CB1 2FB, U.K.*

Complete contact information is available at: https://pubs.acs.org/10.1021/acssynbio.2c00084

Author Contributions

IZ performed the simulations and analysis, wrote the Visual DSD model code, and implemented the critical leak analysis method; VK and IZ conceived the critical leak analysis method; CS and AP encoded the Logic DSD rules; CS developed and implemented the reaction enumeration method; AP designed the control circuits with reduced leakage; IZ, AP, VK, and CS conceived the leak approximation methods and wrote the manuscript.

Notes

The authors declare no competing financial interest.

■ ACKNOWLEDGMENTS

We thank Francesca Mantellino (University of Warwick) for her help on this manuscript. We thank Martina Sciola of MathWorks Inc. (Cambridge, UK) for her help on SimBiology Toolbox implementations. This research is supported, in parts, by the EPSRC INDUSTRIAL CASE AWARD (CASE Voucher 16000070), Microsoft Research, Erasmus+ 2018-1-UK01-KA107-047454 grant, and the EPSRC/BBSRC grant BB/M017982/1 to the Warwick Integrative Synthetic Biology Centre.

■ REFERENCES

(1) Yurke, B.; Turberfield, A. J.; Mills, A. P.; Simmel, F. C.; Neumann, J. L. A DNA-fuelled molecular machine made of DNA. *Nature* **2000**, *406*, 605−608.

(2) Simmel, F.; Yurke, B.; Singh, H. Principles and applications of nucleic acid strand displacement reactions. *Chem. Rev.* **2019**, *119* (10), 6326−6369.

(3) Zhang, D.; Seelig, G. Dynamic DNA nanotechnology using strand-displacement reactions. *Nat. Chem.* **2011**, *3*, 103−113.

(4) Soloveichik, D.; Seelig, G.; Winfree, E. DNA as a universal substrate for chemical kinetics. *Proc. Natl. Acad. Sci. U.S.A.* **2010**, *107*, 5393−5398.

(5) Seelig, G.; Soloveichik, D.; Zhang, D.; Winfree, E. Enzyme-free nucleic acid logic circuits. *Science* **2006**, *314* (5805), 1585−1588.

(6) Qian, L.; Winfree, E. Scaling up digital circuit computation with DNA strand displacement cascades. *Science (New York, N.Y.)* **2011**, *332*, 1196−201.

(7) Qian, L.; Winfree, E.; Bruck, J. Neural network computation with DNA strand displacement cascades. *Nature* **2011**, *475*, 368−372.

(8) Chen, Y.-J.; Dalchau, N.; Srinivas, N.; Philips, A.; Cardelli, L.; Soloveichik, D.; Seelig, G. Programmable chemical controllers made from DNA. *Nat. Nanotechnol.* **2013**, *8*, 755−762.

(9) Weitz, M.; Kim, J.; Kapsner, K.; Winfree, E.; Franco, E.; Simmel, F. C. Diversity in the dynamical behavior of a compartmentalized programmable biochemical oscillator. *Nat. Chem.* **2014**, *6*, 295−302.

(10) Srinivas, N.; Parkin, J.; Seelig, G.; Winfree, E.; Soloveichik, D. Enzyme-free nucleic acid dynamical systems. *Science* **2017**, *358*, No. eaal2052.

(11) Simmel, F.; Yurke, B. A DNA-based molecular device switchable between three distinct mechanical states. *Appl. Phys. Lett.* **2002**, *80*, 883−885.

(12) Shin, J.; Pierce, N. A synthetic DNA walker for molecular transport. *J. Am. Chem. Soc.* **2004**, *126*, 10834−10835.

(13) He, Y.; Liu, D. R. Autonomous multistep organic synthesis in a single isothermal solution mediated by a DNA walker. *Nat. Nanotechnol.* **2010**, *5*, 778−782.
(14) Muscat, R. A.; Bath, J.; Turberfield, A. J. A Programmable Molecular Robot. *Nano Lett.* **2011**, *11*, 982−987.
(15) Modi, S.; Nizak, C.; Surana, S.; Halder, S.; Krishnan, Y. Two DNA nanomachines map pH changes along intersecting endocytic pathways inside the same cell. *Nat. Nanotechnol.* **2013**, *8*, 459−467.
(16) Thubagere, A. J.; Li, W.; Johnson, R. F.; Chen, Z.; Doroudi, S.; Lee, Y. L.; Izatt, G.; Wittman, S.; Srinivas, N.; Woods, D.; Winfree, E.; Qian, L. A cargo-sorting DNA robot. *Science (New York, N.Y.)* **2017**, *357*, No. eaan6558.
(17) Chen, H.; Weng, T.-W.; Riccitelli, M. M.; Cui, Y.; Irudayaraj, J.; Choi, J. H. Understanding the mechanical properties of DNA origami tiles and controlling the kinetics of their folding and unfolding reconfiguration. *J. Am. Chem. Soc.* **2014**, *136*, 6995−7005.
(18) Peng, R.; Wang, H.; Lyu, Y.; Xu, L.; Liu, H.; Kuai, H.; Liu, Q.; Tan, W. Facile assembly/disassembly of DNA nanostructures anchored on cell-mimicking giant vesicles. *J. Am. Chem. Soc.* **2017**, *139*, 12410−12413.
(19) Simmel, F. C.; Schulman, R. Self-organizing materials built with DNA. *MRS Bull.* **2017**, *42*, 913−919.
(20) Kahn, J. S.; Hu, Y.; Willner, I. Stimuli-responsive DNA-based hydrogels: from basic principles to applications. *Acc. Chem. Res.* **2017**, *50*, 680−690.
(21) Zhang, D.; Turberfield, A.; Yurke, B.; Winfree, E. Engineering entropy-driven reactions and networks catalyzed by DNA. *Science* **2007**, *318*, 1121−1125.
(22) Jiang, Y.; Li, B.; Chen, X.; Ellington, D. A. Coupling two different nucleic acid circuits in an enzyme-free amplifier. *Molecules* **2012**, *17*, 13211−13220.
(23) Kotani, S.; Hughes, W. L. Multi-Arm Junctions for Dynamic DNA Nanotechnology. *J. Am. Chem. Soc.* **2017**, *139*, 6363−6368.
(24) Isaacs, F. J.; Dwyer, D. J.; Ding, C. M.; Pervouchine, D. D.; Cantor, C. R.; Collins, J. J. Engineered riboregulators enable posttranscriptional control of gene expression. *Nat. Biotechnol.* **2004**, *22*, 841−847.
(25) Pardee, K.; Green, A. A.; Ferrante, T.; Cameron, D. E.; Daleykeyser, A.; Yin, P.; Collins, J. J. Paper-based synthetic gene networks. *Cell* **2014**, *159*, 940−954.
(26) Gao, Z. F.; Ling, Y.; Lu, L.; Chen, N. Y.; Luo, H. Q.; Li, H. B. Detection of single-nucleotide polymorphisms using an ON-OFF switching of regenerated biosensor based on a locked nucleic acidintegrated and toehold-mediated strand displacement reaction. *Anal. Chem.* **2014**, *86*, 2543−2548.
(27) Song, W. L.; Zhang, Q.; Sun, W. B. Ultrasensitive detection of nucleic acids by template enhanced hybridization followed by rolling circle amplification and catalytic hairpin assembly. *Chem. Commun.* **2015**, *51*, 2392−2395.
(28) Peng, P.; Shi, L.; Wang, H.; Li, T. A DNA nanoswitchcontrolled reversible nanosensor. *Nucleic Acids Res.* **2017**, *45*, 541−546.
(29) Meng, W.; Muscat, R. A.; McKee, M. L.; Milnes, P. J.; El-Sagheer, A. H.; Bath, J.; Davis, B. G.; Brown, T.; O'Reilly, R. K.; Turberfield, A. J. An autonomous molecular assembler for programmable chemical synthesis. *Nat. Chem.* **2016**, *8*, 542−548.
(30) Chen, R. P.; Blackstock, D.; Sun, Q.; Chen, W. Dynamic protein assembly by programmable DNA strand displacement. *Nature Chem.* **2018**, *10*, 474−481.
(31) Li, B.; Ellington, A. D.; Chen, X. Rational, modular adaptation of enzyme-free DNA circuits to multiple detection methods. *Nucleic Acids Res.* **2011**, *39*, e110.
(32) Jung, C.; Ellington, A. D. Diagnostic applications of nucleic acid circuits. *Acc. Chem. Res.* **2014**, *47*, 1825−1835.
(33) Pardee, K.; Slomovic, S.; Nguyen, P. Q.; Lee, J. W.; Donghia, N.; Burrill, D.; Ferrante, T.; McSorley, F. R.; Furuta, Y.; Vernet, A.; Lewandowski, M.; Boddy, C. N.; Joshi, N. S.; Collins, J. J. Portable, on-demand biomolecular manufacturing. *Cell* **2016**, *167* (1), 248−259.
(34) Green, A. A.; Kim, J.; Ma, D.; Silver, P. A.; Collins, J. J.; Yin, P. Complex cellular logic computation using ribocomputing devices. *Nature* **2017**, *548*, 117−121.
(35) Lopez, R.; Wang, R.; Seelig, G. A molecular multi-gene classifier for disease diagnostics. *Nat. Chem.* **2018**, *10* (7), 746−754.
(36) Douglas, S. M.; Bachelet, I.; Church, G. M. A logic gated nanorobot for targeted transport of molecular payloads. *Science* **2012**, *335*, 831−834.
(37) You, M.; Peng, L.; Shao, N.; Zhang, L.; Qiu, L.; Cui, C.; Tan, W. DNA "nano-claw": logic-based autonomous cancer targeting and therapy. *J. Am. Chem. Soc.* **2014**, *136*, 1256−1259.
(38) Zhang, P.; Wang, C.; Zhao, J.; Xiao, A.; Shen, Q.; Li, L.; LI, J.; Zhang, J.; Min, Q.; Chen, J.; Chen, H.-Y.; Zhu, J.-J. Near infraredguided smart nanocarriers for microRNA-controlled release of doxorubicin/siRNA with intracellular ATP as fuel. *ACS Nano* **2016**, *10*, 3637−3647.
(39) Chen, Y.-j.; Groves, B.; Muscat, R. A.; Seelig, G. DNA nanotechnology from the test tube to the cell. *Nat. Nanotechnol.* **2015**, *10*, 748−760.
(40) Wang, B.; Thachuk, C.; Ellington, A. D.; Winfree, E.; Soloveichik, D. Effective design principles for leakless strand displacement systems. *Proc. Natl. Acad. Sci. U.S.A.* **2018**, *115* (52), E12182−E12191.
(41) Fern, J.; Scalise, D.; Cangialosi, A.; Howie, D.; Potters, L.; Schulman, R. DNA Strand-Displacement Timer Circuits. *ACS Synth. Biol.* **2017**, *6*, 190−193.
(42) SantaLucia, J.; Allawi, H. T.; Seneviratne, P. A. Improved Nearest-Neighbor Parameters for Predicting DNA Duplex Stability. *Biochemistry* **1996**, *35*, 3555−3562.
(43) Jiang, Y. S.; Bhadra, S.; Li, B.; Ellington, A. D. Mismatches Improve the Performance of Strand-Displacement Nucleic Acid Circuits. *Angew. Chem., Int. Ed.* **2014**, *53*, 1845−1848.
(44) Machinek, R. R. F.; Ouldridge, T. E.; Haley, N. E. C.; Bath, J.; Turberfield, A. J. Programmable energy landscapes for kinetic control of DNA strand displacement. *Nat. Commun.* **2014**, *5*, 5324.
(45) Olson, X.; Kotani, S.; Padilla, J. E.; Hallstrom, N.; Goltry, S.; Lee, J.; Yurke, B.; Hughes, W. L.; Graugnard, E. Availability: A Metric for Nucleic Acid Strand Displacement Systems. *ACS Synth. Biol.* **2017**, *6*, 84−93.
(46) Irmisch, P.; Ouldridge, T. E.; Seidel, R. Modeling DNA-Strand Displacement Reactions in the Presence of Base-Pair Mismatches. *J. Am. Chem. Soc.* **2020**, *142*, 11451−11463.
(47) Zadeh, J. N.; Steenberg, C. D.; Bois, J. S.; Wolfe, B. R.; Pierce, M. B.; Khan, A. R.; Dirks, R. M.; Pierce, N. A. NUPACK: Analysis and design of nucleic acid systems. *J. Comput. Chem.* **2011**, *32* (1), 170−173.
(48) Šulc, P.; Romano, F.; Ouldridge, T. E.; Rovigatti, L.; Doye, J. P. K.; Louis, A. A. Sequence-dependent thermodynamics of a coarse-grained DNA model. *J. Chem. Phys.* **2012**, *137*, 135101.
(49) Wang, B.; Thachuk, C.; Ellington, A. D.; Soloveichik, D. The design space of strand displacement cascades with toehold-size clamps. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*; LNCS, 2017; Vol. *10467*, pp 64−81.
(50) Ang, Y. S.; Tong, R.; Yung, L.-Y. L. Engineering a robust DNA split proximity circuit with minimized circuit leakage. *Nucleic Acids Res.* **2016**, *44* (14), No. e121.
(51) Frezza, B. M.; Cockroft, S. L.; Ghadiri, M. R. Modular multi-level circuits from immobilized DNA-based logic gates. *J. Am. Chem. Soc.* **2007**, *129* (48), 14875.
(52) Teichmann, M.; Kopperger, E.; Simmel, F. C. Robustness of localized DNA strand displacement cascades. *ACS Nano* **2014**, *8* (8), 8487.
(53) Chatterjee, G.; Dalchau, N.; Muscat, R. A.; Phillips, A.; Seelig, G. A spatially localized architecture for fast and modular DNA computing. *Nat. Nanotechnol.* **2017**, *12* (9), 920.
(54) Bui, H.; Shah, S.; Mokhtar, R.; Song, T.; Garg, S.; Reif, J. Localized DNA Hybridization Chain Reactions on DNA Origami. *ACS Nano* **2018**, *12*, 1146−1155.
(55) Joesaar, A.; Yang, S.; Bögels, B.; der Linden, A. v.; Kumar, B. P.; Dalchau, N.; Phillips, A.; Mann, S.; de Greef, T. F. A. Distributed DNA-

https://doi.org/10.1021/acssynbio.2c00084
ACS Synth. Biol. XXXX, XXX, XXX−XXX

based Communication in Populations of Synthetic Protocells. *bioRxiv*, January 4, 2019. DOI: 10.1101/511725.
(56) Song, T.; Gopalkrishnan, N.; Eshra, A.; Garg, S.; Mokhtar, R.; Bui, H.; Chandran, H.; Reif, J. Improving the Performance of DNA Strand Displacement Circuits by Shadow Cancellation. *ACS Nano* **2018**, *12* (11), 11689−11697.
(57) Schaeffer, J. *Stochastic simulation of the kinetics of multiple interacting nucleic acid strands*. Master's thesis, California Institute of Technology, 2012.
(58) Badelt, S.; Grun, C.; Sarma, K. V.; Wolfe, B.; Shin, S. W.; Winfree, E. A domain-level DNA strand displacement reaction enumerator allowing arbitrary non-pseudoknotted secondary structures. *J. R. Soc. Interface* **2020**, *17*, 20190866.
(59) Grun, C.; Werfel, J.; Zhang, D. Y.; Yin, P. DyNAMiC Workbench: an integrated development environment for dynamic DNA nanotechnology. *Journal of The Royal Society Interface* **2015**, *12*, 20150580.
(60) Lakin, M. R.; Youssef, S.; Polo, F.; Emmott, S.; Phillips, A. Visual DSD: a design and analysis tool for DNA strand displacement systems. *Bioinformatics* **2011**, *27* (22), 3211−3213.
(61) Spaccasassi, C.; Lakin, M.; Phillips, A. A logic programming language for computational nucleic acid devices. *ACS Synth. Biol.* **2019**, *8* (7), 1530−1547.
(62) Petersen, R. L.; Lakin, M. R.; Phillips, A. A strand graph semantics for DNA-based computation. *Theoretical Computer Science* **2016**, *632*, 43−73.
(63) Montagne, K.; Plasson, R.; Sakai, Y.; Fujii, T.; Rondelez, Y. Programming an *in vitro* DNA oscillator using a molecular networking strategy. *Mol. Syst. Biol.* **2011**, *7* (466), 1−7.
(64) Baccouche, A.; Montagne, K.; Padirac, A.; Fujii, T.; Rondelez, Y. Dynamic DNA-toolbox reaction circuits: A walkthrough. *Methods* **2014**, *67* (2), 234−249.
(65) Phillips, A.; Cardelli, L. A programming language for composable DNA circuits. *J. R. Soc., Interface* **2009**, *6* (Suppl 4), S419−S436.
(66) Yordanov, B.; Kim, J.; Petersen, R. L.; Shudy, A.; Kulkarni, V. V.; Phillips, A. Computational design of nucleic acid feedback control circuits. *ACS Synth. Biol.* **2014**, *3*, 600−616.
(67) Doets, K. *From Logic to Logic Programming*; MIT Press: Cambridge, MA, 1994.
(68) Nilsson, U.; Maluszynski, J. *Logic, Programming, and PROLOG*, 2nd ed.; John Wiley & Sons, Inc.: New York, NY, 1995.
(69) Cardelli, L. Two-domain DNA strand displacement. *Mathematical Structures in Computer Science* **2013**, *23* (02), 247−271.
(70) Lakin, M.; Pauleve, L.; Phillips, A. Stochastic simulation of multiple process calculi for biology. *Theor. Comput. Sci.* **2012**, *431*, 181−206.
(71) Lakin, M. R.; Youssef, S.; Cardelli, L.; Phillips, A. Abstractions for DNA circuit design. *J. R. Soc., Interface* **2012**, *9* (68), 470−486.
(72) Thachuk, C.; Winfree, E.; Soloveichik, D. Leakless DNA strand displacement systems. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* **2015**, *9211*, 133−153.
(73) Li, X.; Wang, X.; Song, T.; Lu, W.; Chen, Z.; Shi, X. A Novel Computational Method to Reduce Leaky Reaction in DNA Strand Displacement. *J. Anal. Methods Chem.* **2015**, *2015*, 1.
(74) Aho, A. V.; Hopcroft, J. E.; Ullman, J. *Data Structures and Algorithms*, 1st ed.; Addison-Wesley Longman Publishing Co., Inc., 1983.
(75) Oishi, K.; Klavins, E. Biomolecular implementation of linear I/O systems. *IET Systems Biology* **2011**, *5*, 252−260.
(76) Zarubiieva, I.; Kulkarni, V. Accurate computation of logarithm using abstract chemical reaction networks. In *ICSBB 2018: 20th International Conference on Systems Biology and Bioengineering*; London, UK, November 2018.
(77) Zhang, D. Y.; Seelig, G. DNA-Based Fixed Gain Amplifiers and Linear Classifier Circuits. In *DNA Computing and Molecular Programming*; Sakakibara, Y., Mi, Y., Eds.; Springer Berlin Heidelberg: Berlin, Heidelberg, 2011; pp 176−186.
(78) Zhang, D. Y. Cooperative hybridization of oligonucleotides. *J. Am. Chem. Soc.* **2011**, *133*, 1077−86.
(79) Chen, X.; Briggs, N.; McLain, J. R.; Ellington, A. D. Stacking nonenzymatic circuits for high signal gain. *Proc. Natl. Acad. Sci. U.S.A.* **2013**, *110* (14), 5386−5391.
(80) Olson, X.; Kotani, S.; Yurke, B.; Graugnard, E.; Hughes, W. L. Kinetics of DNA Strand Displacement Systems with Locked Nucleic Acids. *J. Phys. Chem. B* **2017**, *121*, 2594−2602.

https://doi.org/10.1021/acssynbio.2c00084
ACS Synth. Biol. XXXX, XXX, XXX−XXX