Theory & Practice
of DNA strand displacement circuits

October 8, 2018 @ DNA 24

Chris Thachuk
Winfree Lab, California Institute of Technology
Molecular Programming Coat of Arms
Today’s tutorial in a nutshell
Molecular Circuits
Built upon DNA strand displacement cascades

<table>
<thead>
<tr>
<th>Input</th>
<th>Computation</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>5’–ACCACGATCACATTAC–3’</td>
<td>DNA strand displacement cascades</td>
<td>5’–GAGCTACATCAC–3’</td>
</tr>
<tr>
<td>5’–GCAACATACAT–3’</td>
<td></td>
<td>5’–TAAATCATGATCAG–3’</td>
</tr>
<tr>
<td>5’–CCCATACTACACAG–3’</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5’–TACCACATGAGCAGCA–3’</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Molecular Circuits
Built upon DNA strand displacement cascades

<table>
<thead>
<tr>
<th>Input</th>
<th>Computation</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>5’-ACCACGATCACATTAC-3’</td>
<td></td>
<td>5’-GAGCTACATCAC-3’</td>
</tr>
<tr>
<td>5’-GCAACATACAT-3’</td>
<td></td>
<td>5’-TAAATCATGATCAG-3’</td>
</tr>
<tr>
<td>5’-CCCATACATCACCAG-3’</td>
<td>DNA strand displacement cascades</td>
<td></td>
</tr>
<tr>
<td>5’-TACCACATGAGCAGCA-3’</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Molecular Circuits
Built upon DNA strand displacement cascades

Input
- 5’-ACCACGATCACATTAC-3’
- 5’-GCAACATACAT-3’
- 5’-CCCATACATCACCAG-3’
- 5’-TACCACATGAGCAGCA-3’

Computation
- DNA strand displacement cascades

Output
- 5’-GAGCTACATCAC-3’
- 5’-TAAATCATGATCAG-3’
Tutorial Outline

- Review of strand displacement
- Building and composing logic gates
- Tools for designing and verifying circuits
- Robustness of strand displacement
Review of DNA Strand Displacement (DSD)


Video Courtesy of Biocomputation Group @ Microsoft Research
Review of DNA Strand Displacement (DSD)


Video Courtesy of Biocomputation Group @ Microsoft Research
2. Domain 2 undergoes branch migration.
3. Strand displacement completes.
toehold

long domain

1

* 1

2

* 2

TGTACCTGTGCA

ACATGGACAGCTATAAC

1*

2*

Global energy barrier

Local energy barrier

Complex Y (old)

Output B (old)

2. Domain 2 undergoes branch migration

3. Strand displacement completes

Adapted from Zhang & Seelig 2011 (simplified energy model)
1. Domain 2 undergoes branch migration
2. Strand displacement completes

Adapted from Zhang & Seelig 2011
Strand Displacement Cascades

= Three Rules

I Bind
II Release
III Displace
Domain level rules for DSD

Rule 1: \textit{Bind}

Example

\[
\begin{array}{c}
1 \\
\vdots \\
1^* \\
\end{array}
\]
Domain level rules for DSD

Rule 1: *Bind*

Example

1

single-stranded complementary domains

1*
Domain level rules for DSD

Rule 1: \textit{Bind}

Example
Domain level rules for DSD

Rule 1: \textit{Bind}

Two single-stranded complementary domains can bind

Example
Domain level rules for DSD

Rule 2: *Release*

Example
Domain level rules for DSD

Rule 2: *Release*

Example
Domain level rules for DSD

Rule 2: *Release*

Example

*blue strand* bound by only a short domain
Domain level rules for DSD

Rule 2: *Release*

Example

\[1 \rightarrow 1^*\]
Any strand bound by only a short domain can release

Example

1

......

1*

......
Domain level rules for DSD

Rule 3: *Displace*

Example

[Diagram showing the concept of Displace with numbered points and arrows.]
Domain level rules for DSD

Rule 3: *Displace*

Example

identical domains, one bound, one free
Domain level rules for DSD

Rule 3: \textit{Displace}

Example

\begin{itemize}
  \item Already bound
\end{itemize}
Domain level rules for DSD

Rule 3: *Displace*

Example
Domain level rules for DSD

Rule 3: **Displace**

A domain can **displace** an identical domain of another strand, *if neighboring domains are already bound*.

**Example**

```
1  2
1* 2*
```
Why do we use toeholds?

Adapted from Zhang & Seelig 2011

long domain

toehold

1

tgtacctgtgca

acatggacagctatac

1*

2*

log 10 k (M-1 s-1)

Toehold length (nt)

0 10 15 5

0 2 4 6 8

2* 3*

23 21

1 1*

2*

Why do we use toeholds?
Why do we use toeholds?

Toeholds introduce energy barriers.

Adapted from Zhang & Seelig 2011

Toehold length (nt)

log_{10} k (M^{-1} s^{-1})

- Toehold length (nt)
- Adhesive energy barrier
- Global energy barrier

Folding pathway events

(simplified energy model)
Toehold-mediated DNA strand displacement

T: toehold domain (typically 3-7 nucleotides)
S: branch migration domain (typically 15-20 nucleotides)

\[ A + B \xrightarrow{k} C + D \]

The rate of strand displacement grows exponentially with toehold length for short toeholds.
Toehold-mediated DNA strand displacement

T: toehold domain (typically 3-7 nucleotides)
S: branch migration domain (typically 15-20 nucleotides)

\[ A + B \stackrel{k}{\rightarrow} C + D \]

The rate of strand displacement grows exponentially with toehold length for short toeholds.

\[ k \approx 10^L / \text{M/s} \quad \text{when} \ L \leq 6 \]
\[ k \approx 10^6 / \text{M/s} \quad \text{otherwise} \]

L: toehold length

Kinetics of toehold-mediated strand displacement

\[ A + B \rightleftharpoons AB \rightarrow C + D \]

\[ k_{\text{eff}} \approx \frac{10^6}{M/s} \text{ when } L \leq 6 \]

\[ k_{\text{eff}} \approx \frac{10^6}{M/s} \]

\[ k_{\text{eff}} \approx \frac{10^6}{M/s} \]

This approximation is valid for low concentrations of A and B (e.g. \([A]=[B]=100\text{nM}\)) such that the unimolecular reaction is sufficiently faster than the bimolecular reaction.
Kinetics of toehold-mediated strand displacement

\[ A + B \rightleftharpoons AB \rightarrow C + D \]

Simplified kinetic scheme:

\[ \text{collision rate: } 10^6 \text{M/s} \]

\[ \text{collision success probability: } \frac{1}{s} + 10^6 - \frac{L}{s} \]

\[ \text{net rate of success: } 10^6 \cdot \left( 1 + 10^6 - \frac{L}{s} \right) \]

This approximation is valid for low concentrations of A and B (e.g. [A]=[B]=100nM) such that the unimolecular reaction is sufficiently faster than the bimolecular reaction.

\[ k_{\text{eff}} \approx 10^L \text{M/s} \] when \( L \leq 6 \)

Otherwise \[ k_{\text{eff}} \approx 10^6 \text{M/s} \]

\( L \): toehold length

References:
- Zhang et al., JACS 2009
- Srinivas et al., NAR 2013
Kinetics of toehold-mediated strand displacement

\[ A + B \leftrightarrow AB \rightarrow C + D \]

\[ \text{Collision rate: } 10^6 \text{M/s} \]
\[ \text{Collision success probability: } \frac{1}{s} + 10^6 - \frac{L}{s} \]
\[ \text{Net rate of success: } 10^6 \cdot \frac{1}{1 + 10^6 - \frac{L}{s}} \]

This approximation is valid for low concentrations of A and B (e.g. \([A]=[B]=100\text{nM}\)) such that the unimolecular reaction is sufficiently faster than the bimolecular reaction.

\[ k_{\text{eff}} \approx \frac{10^6}{L} \text{M/s} \]
\[ \text{when } L \leq 6 \]
\[ k_{\text{eff}} \approx 10^6 \text{M/s} \]
\[ \text{otherwise} \]

\( L \): toehold length

\( Zhanget al., JACS 2009 \)
\( Srinivas et al., NAR 2013 \)
Kinetics of toehold-mediated strand displacement

\[ A + B \leftrightarrow AB \rightarrow C + D \]

\[ \begin{align*}
T1 & \quad S1 \\
T1^* & \quad S1^* \\
\text{10}^6/M/s & \quad 10^6-L/s \\
\end{align*} \]

\[ \begin{align*}
1/s & \\
T1 & \quad S1 \\
T1^* & \quad S1^* \\
\end{align*} \]

This approximation is valid for low concentrations of A and B (e.g. [A]=[B]=100nM) such that the unimolecular reaction is sufficiently faster than the bimolecular reaction.

\[ k_{eff} \approx \text{10}^6/L/M/s \quad \text{otherwise} \]

\[ k_{eff} \approx \text{10}^6/M/s \]

\( L \): toehold length

Zhang et al., JACS 2009

Srinivas et al., NAR 2013
Kinetics of toehold-mediated strand displacement

\[ A + B \leftrightarrow AB \rightarrow C + D \]

simplify: \[ A + B \xrightarrow{k_{eff}} C + D \]

collision rate: \[ 10^6 [A][B] \]

collision success probability: \[ \frac{1/s}{1/s + 10^{6-L}/s} \]

net rate of success: \[ \frac{10^6 \cdot \frac{1}{1 + 10^{6-L}[A][B]}}{k_{eff}} \]

This approximation is valid for low concentrations of A and B (e.g. \([A]=[B]=100nM\)) such that the unimolecular reaction is sufficiently faster than the bimolecular reaction.
Kinetics of toehold-mediated strand displacement

\[ A + B \leftrightarrow AB \rightarrow C + D \]

simplify: \[ A + B \xrightarrow{k_{eff}} C + D \] \[ k_{eff} = ? \]

collision rate: \[ 10^6[A][B] \]

collision success probability: \[ \frac{1/s}{1/s + 10^{6-L}/s} \]

net rate of success: \[ \frac{10^6 \cdot \frac{1}{1 + 10^{6-L}[A][B]}}{k_{eff}} \]

This approximation is valid for low concentrations of A and B (e.g. \([A]=[B]=100\text{nM}\)) such that the unimolecular reaction is sufficiently faster than the bimolecular reaction.

Zhang et al, JACS 2009
Srinivas et al, NAR 2013
Strand displacement in the lab
In all tested cases, the output went to the correct ON or OFF state. A three-OR cascade (fig. S6, A and B) and a four-OR cascade (fig. S6, C and D) also worked. The delay time required for circuit computation increased linearly with the number of layers (Fig. 3A). However, once the threshold for the output gate was exceeded, the signal increased at roughly the same rate as in the smaller circuit (Fig. 3B). In a circuit with four layers, two AND gates, and three OR gates, with 12 different combinations of inputs, the output went to clear and correct ON or OFF states in 8 hours (Fig. 3C).

Because integrating gates support multiple inputs and amplifying gates support multiple outputs, logic gates built from a pair of them can easily support fan-in and fan-out. In a circuit with a four-input OR gate, only when all inputs from the upstream OR gates were OFF did the output stay OFF (Fig. 3D). In a circuit with a four-output OR gate, each output copied the correct logic from the upstream OR gate (Fig. 3E). Circuits with a four-input AND gate and a four-output AND gate are shown in fig. S8C and fig. S9C, respectively.

To demonstrate a digital circuit with an interesting function, we built a circuit that computes the floor of the square root of a four-bit binary number (Fig. 4A). It is not an optimized digital logic circuit; it is designed to showcase AND, OR, NOT, NAND, NOR, fan-in, and fan-out of logic gates, as well as fan-out of input signals. NOT gates are difficult to implement directly using representations where the ON or OFF state of an input is determined by the presence or absence of a single DNA species: A circuit might compute a false output before all input strands are added, because NOT gates already produce ON signals in the absence of their inputs, and for use-once circuits (such as seesaw circuits), computations cannot be undone. Therefore, we use dual-rail logic (fig. S10B). Each input is replaced by a pair of inputs, representing logic ON and OFF separately. Each logic gate is replaced by a pair of AND or OR gates. (Taking the NOR gate as an example, output being OFF is the OR of both inputs being ON; output being ON is the AND of both inputs being OFF.) Initially, the pair of inputs is absent, indicating that the logic value of this signal is unknown. At the beginning of computation, one input of the pair will be added, indicating either logic ON or OFF. In this way, no computation will take place before the input signals arrive. With dual-rail logic, any AND-OR-NOT circuit can be transformed into an equivalent circuit with AND or OR gates only. Then, any AND-OR circuit can be further transformed into an equivalent seesaw circuit.
Strand displacement in the lab

- Large autonomous biochemical networks built from scratch
  Qian, Winfree, Science 2011

- Biochemical system doing inference
  Qian, Winfree, Bruck Nature 2011
Strand displacement in the lab

molecular logic circuits

- Large autonomous biochemical networks built from scratch
  Qian, Winfree, Science 2011

controlling assembly of nanoscale structures

- Prescribed nanoscale structures seen under atomic force microscope
  Yin, Choi, Calvert, Yurke, Pierce Nature 2008

molecular artificial neural networks

- Biochemical system doing inference
  Qian, Winfree, Bruck Nature 2011
Strand displacement in the lab

molecular logic circuits

- Large autonomous biochemical networks built from scratch
  Qian, Winfree, Science 2011

controlling assembly of nanoscale structures

- Prescribed nanoscale structures seen under atomic force microscope
  Yin, Choi, Calvert, Yurke, Pierce Nature 2008

molecular artificial neural networks

- Biochemical system doing inference
  Qian, Winfree, Bruck Nature 2011

strand displacement cascades

strand displacement in vivo

- Logic on biological signals
  Hemphill, Deiters J Am Chem Soc 2013
Strand displacement in the lab

molecular logic circuits

- Large autonomous biochemical networks built from scratch
  Qian, Winfree, Science 2011

controlling assembly of nanoscale structures

- Prescribed nanoscale structures seen under atomic force microscope
  Yin, Choi, Calvert, Yurke, Pierce Nature 2008

molecular artificial neural networks

- Biochemical system doing inference
  Qian, Winfree, Bruck Nature 2011

strand displacement cascades

strand displacement in vivo

- Logic on biological signals
  Hemphill, Deiters J Am Chem Soc 2013
Tutorial Outline

- Review of strand displacement
- Building and composing logic gates
- Tools for designing and verifying circuits
- Robustness of strand displacement
AND gate

release Z if and only if X and Y are present
AND gate
release Z if and only if X and Y are present
AND gate
release Z if and only if X and Y are present

voltages

strands

gates get consumed!
(need to have many copies)
gate=complex
Strand Displacement Cascades Example: AND gate

release $Z$ if and only if $X$ and $Y$ are present

AND gate
Strand Displacement Cascades Example: AND gate

release $Z$ if and only if $X$ and $Y$ are present

### Strand Displacement Cascades Example: AND gate

**Input X**

**Input Y**

**AND gate**
Strand Displacement Cascades Example: AND gate

release Z if and only if X and Y are present

input Y
Strand Displacement Cascades Example: AND gate

release Z if and only if X and Y are present
Strand Displacement Cascades Example: AND gate

release Z if and only if X and Y are present
Strand Displacement Cascades Example: AND gate

release Z if and only if X and Y are present
Strand Displacement Cascades Example: AND gate

release Z if and only if X and Y are present
Strand Displacement Cascades Example: AND gate

release Z if and only if X and Y are present

input Y

displace
Strand Displacement Cascades Example: AND gate

release Z if and only if X and Y are present
Strand Displacement Cascades Example: AND gate

release Z if and only if X and Y are present

input Y
Strand Displacement Cascades Example: AND gate

release Z if and only if X and Y are present

input Y
Strand Displacement Cascades Example: AND gate

release $Z$ if and only if $X$ and $Y$ are present

waste

input $Y$
Strand Displacement Cascades Example: AND gate

release Z if and only if X and Y are present

waste

input Y

Strand Displacement Cascades Example: AND gate

release Z if and only if X and Y are present

waste

input Y
Strand Displacement Cascades Example: AND gate

release Z if and only if X and Y are present

waste
Strand Displacement Cascades Example: AND gate

release Z if and only if X and Y are present

waste
Strand Displacement Cascades Example: AND gate

release Z if and only if X and Y are present

waste
Strand Displacement Cascades Example: AND gate

release Z if and only if X and Y are present

waste
Strand Displacement Cascades Example: AND gate

release Z if and only if X and Y are present

output Z
Strand Displacement Cascades Example: AND gate

release Z if and only if X and Y are present

before

input X

input Y

AND gate

after

output Z

waste

AND gate

waste
Strand Displacement Cascades Example: AND gate

release Z if and only if X and Y are present

before

input X

4 5 6 7

input Y

2 3 5 6 7

Composable

after

output Z

2 3 5

waste

AND gate

0 1 2 3

6 7 8 9

waste

0 1 2 3 5

1* 2* 3* 5* 6* 7* 8* 9*
Strand Displacement Cascades Example: AND gate

release Z if and only if X and Y are present

before

input X

input Y

Composable
gate is “used up”

AND gate

after

output Z

waste

waste
Composing AND gates

A
B
X

Y
W
Z
We need a “wire”
Sequence Independence

Translator (a “wire”): $X \rightarrow Y$

input $X$

output $Y$

Different coloring scheme to emphasize sequence (in)dependence!
Sequence Independence

Translator (a “wire”): X→Y

input X

F₁

F₂
Sequence Independence
Translator (a “wire”): X→Y

input X

$F_1$

bind

$F_2$
Sequence Independence

Translator (a “wire”): $X \rightarrow Y$
Sequence Independence

Translator (a “wire”): X→Y
Reading Output

A

B

X

Y

W

Z
displace

read many different samples
A reaction gate

\[ A_1 + B_1 \rightarrow X_1 + Y_1 \]

This *universal component* can realize a number of logic gates
A reaction gate

\[ A_1 + B_1 \rightarrow X_1 + Y_1 \]

This *universal component* can realize a number of logic gates.
To implement this:
We start with large excess of DNA complexes (fuels) that mediate the reaction:

adopted from Srinivas et al, Science, 2017
AND gate

C

D

1

R1

B1

Y1

A1

X1
Signal Fanout Gates

5’-GCACCTTTTTACATTACATTACATTAC-3’
fuel (always present)

5’-ACAGATCACCAGATCATTATCAGAG-3’
strand representing signal S
Signal Fanout Gates

5’-GCACCTTTACATTACATTACATTACATTAC-3’
fuel (always present)

5’-ACAGATCACCAGATCATTATCAGAG-3’
strand representing signal S
Handling OR gates

AND( A1, OR( B1, AND( A2, B2 ) ) )
Handling OR gates

\[ \text{AND}(\ A1, \ \text{OR}(\ B1, \ \text{AND}(\ A2, \ B2 ) \ ) \ ) \]
Handling NOT gates
Handling NOT gates

Dual-rail input, Dual-rail output

5'-CCCCCCC-3'

S

5'-TTTTTTT-3'

T

\[ \sim S \]

\[ \sim T \]
Dual rail logic

AND

OR

NOT
Handling NOT gates

Dual-rail input, Dual-rail output

5'-CCCCCCC-3'

5'-TTTTTTTT-3'

$\overline{s}$

$\overline{s}$
Handling NOT gates

Dual-rail input, Dual-rail output

5'-CCCCCCC-3'

\( S \)

5'-TTTTTTTT-3'

\( \sim S \)

Dual-rail input

5'-CCCCCCC-3'

\( S \)

5'-TTTTTTTT-3'

\( \sim S \)

Dual-rail input, Dual-rail output
Handling NOT gates

Dual-rail input, Dual-rail output

Dual-rail input

$5'$-CCCCCCC-3'$
$s$

$5'$-TTTTTTTT-3'$
$\neg s$
With reaction gates, wires, and dual-rail encoding, we can build any combinatorial circuit.
Tutorial Outline

- Review of strand displacement
- Building and composing logic gates
- **Tools for designing and verifying circuits**
- Robustness of strand displacement
How do you design the circuit?

<table>
<thead>
<tr>
<th>2-input MUX</th>
<th>2-output DEMUX</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

```verilog
module FullAdder(a, b, cin, cout, sum);
    input a, b, cin;
    output cout, sum;
    wire w1, w2, w3, w4;
    xor #(10) (w1, a, b);
    and #(8) (w2, a, b);
    and #(8) (w3, a, cin);
    and #(8) (w4, b, cin);
    or #(10, 8)(cout, w2, w3, w4);
endmodule
```
How do you design the circuit?

Compile from Verilog, or truth table, into AND-OR-NOT circuit
How do you design the circuit?

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$</td>
<td>$B$</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
How do you design the circuit?

Find minimized AND-OR-NOT circuit using ABC

How do you design the circuit?

Find minimized AND-OR-NOT circuit using ABC

Need to remove explicit NOT gates

How do you design the circuit?

- Find minimized AND-OR-NOT circuit using ABC
- “Tree-ify” circuit

How do you design the circuit?

- Find minimized AND-OR-NOT circuit using ABC
- “Tree-ify” circuit
- Push negations to literal level (dual-rail inputs)
How do you design the circuit?

- Find minimized AND-OR-NOT circuit using ABC
- "Tree-ify" circuit
- Push negations to literal level (dual-rail inputs)
- Compress circuit

---

From circuit to DSD system

DSD: formal language for describing and modeling strand displacement cascades

http://lepton.research.microsoft.com/webdna/

From circuit to DSD system

DSD: formal language for describing and modeling strand displacement cascades

http://lepton.research.microsoft.com/webdna/

From circuit to DSD system

\[ A + B \rightarrow C + D \]

Soloveichik et al. (2010)

Lakin et al. (2012)

Cardelli (2011)

Qian et al. (2011)

Chen et al. (2012), Cardelli (2013), Srinivas (2015), Lakin et al. (2016), ...

Images drawn using VisualDSD, Lakin et al. (2012)
The Nuskell compiler framework

Badelt et al. (2017) - Nuskell
Grun et al. (2014) - Peppercorn
Shin et al. (2017) - CRN pathway decomposition equivalence
Johnson et al. (2018) - CRN bisimulation equivalence
Berleant et al. (submitted) - KinDA
The Nuskell compiler framework

Badelt et al. (2017) - Nuskell
Grun et al. (2014) - Peppercorn
Shin et al. (2017) - CRN pathway decomposition equivalence
Johnson et al. (2018) - CRN bisimulation equivalence
Berleant et al. (submitted) - KinDA
Reaction Enumeration

Grun et al. 2014
CRN equivalence verification

**formal input CRN**

- 3 species
- 7 reactions
  - A → A + A
  - A + A → A
  - A + B → B + B
  - B →
  - A + C →
  - C → C + C
  - C + C → C

**enumerated CRN**

- 360 species
- 668 reactions

translation scheme: qian2011_3D_var1.ts
CRN equivalence verification

**formal input CRN**
- 3 species
- 7 reactions
  - A \rightarrow A + A
  - A + A \rightarrow A
  - A + B \rightarrow B + B
  - B \rightarrow
  - A + C \rightarrow
  - C \rightarrow C + C
  - C + C \rightarrow C

**enumerated CRN**
- 360 species
- 668 reactions

translation scheme: qian2011_3D_var1.ts
CRN equivalence verification

<table>
<thead>
<tr>
<th>formal input CRN</th>
<th>condensed CRN</th>
<th>enumerated CRN</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 species</td>
<td>42 species</td>
<td>360 species</td>
</tr>
<tr>
<td>7 reactions</td>
<td>32 reactions</td>
<td>668 reactions</td>
</tr>
<tr>
<td>A → A + A</td>
<td>f14 + C → e1428 + f15</td>
<td></td>
</tr>
<tr>
<td>A + A → A</td>
<td>e853 + f12 → C + f13</td>
<td></td>
</tr>
<tr>
<td>A + B → B + B</td>
<td>A + f4 → f3 + e71</td>
<td></td>
</tr>
<tr>
<td>B →</td>
<td>f2 + e25 → A + f1</td>
<td></td>
</tr>
<tr>
<td>A + C →</td>
<td>A + e25 → f2 + e7</td>
<td></td>
</tr>
<tr>
<td>C → C + C</td>
<td>e996 + f3 → A + f10</td>
<td></td>
</tr>
<tr>
<td>C + C → C</td>
<td>e1428 + f15 → f14 + C</td>
<td></td>
</tr>
<tr>
<td></td>
<td>f3 + e71 → A + f4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>e465 + B → e418 + f6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>e614 + f9 → e611 + e730</td>
<td></td>
</tr>
<tr>
<td></td>
<td>e996 + C → e1040 + f12</td>
<td></td>
</tr>
<tr>
<td></td>
<td>e465 + f5 → e514 + e368</td>
<td></td>
</tr>
<tr>
<td></td>
<td>e308 + f7 → f8 + B</td>
<td></td>
</tr>
<tr>
<td></td>
<td>e418 + B → e371 + f6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>C + f13 → e853 + f12</td>
<td></td>
</tr>
<tr>
<td></td>
<td>A + f1 → f2 + e25</td>
<td></td>
</tr>
<tr>
<td></td>
<td>B + e71 → e319 + f7</td>
<td></td>
</tr>
<tr>
<td></td>
<td>f2 + e7 → A + e25</td>
<td></td>
</tr>
<tr>
<td></td>
<td>e1040 + f11 → e1162 + e1163 + e1158</td>
<td></td>
</tr>
<tr>
<td></td>
<td>e319 + f7 → B + e71</td>
<td></td>
</tr>
<tr>
<td></td>
<td>e308 + f9 → e614 + e615 + e611</td>
<td></td>
</tr>
<tr>
<td></td>
<td>e371 + f6 → e418 + B</td>
<td></td>
</tr>
<tr>
<td></td>
<td>e1040 + f12 → e996 + C</td>
<td></td>
</tr>
<tr>
<td></td>
<td>f8 + B → e308 + f7</td>
<td></td>
</tr>
<tr>
<td></td>
<td>e319 + f5 → e372 + e371 + e368</td>
<td></td>
</tr>
<tr>
<td></td>
<td>f3 + e7 → A + f0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>e853 + f15 → e1428 + C</td>
<td></td>
</tr>
<tr>
<td></td>
<td>e1428 + C → e853 + f15</td>
<td></td>
</tr>
<tr>
<td></td>
<td>A + f10 → e996 + f3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>e1163 + f11 → e1158 + e1246</td>
<td></td>
</tr>
<tr>
<td></td>
<td>e418 + f6 → e465 + B</td>
<td></td>
</tr>
<tr>
<td></td>
<td>A + f0 → f3 + e7</td>
<td></td>
</tr>
</tbody>
</table>

Are these networks equivalent?

translation scheme: qian2011_3D_var1.ts
<table>
<thead>
<tr>
<th>formal input CRN</th>
<th>interpreted CRN</th>
<th>condensed CRN</th>
<th>enumerated CRN</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 species</td>
<td>7 reactions</td>
<td>42 species</td>
<td>360 species</td>
</tr>
<tr>
<td>A → A + A</td>
<td>C → C</td>
<td>f14 + C → e1428 + f15</td>
<td></td>
</tr>
<tr>
<td>A + A → A</td>
<td>C + C → C</td>
<td>e853 + f12 → C + f13</td>
<td></td>
</tr>
<tr>
<td>A + B → B + B</td>
<td>A → A</td>
<td>A + f4 → f3 + e71</td>
<td></td>
</tr>
<tr>
<td>B →</td>
<td>B →</td>
<td>f2 + e25 → A + f1</td>
<td></td>
</tr>
<tr>
<td>A + C →</td>
<td>A + A → A + A</td>
<td>A + e25 → f2 + e7</td>
<td></td>
</tr>
<tr>
<td>C → C + C</td>
<td>A → A</td>
<td>e996 + f3 → A + f10</td>
<td></td>
</tr>
<tr>
<td>C + C → C</td>
<td>B → B</td>
<td>e1428 + f15 → f14 + C</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>f3 + e71 → A + f4</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>e465 + B → e418 + f6</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>e614 + f9 → e611 + e730</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>e996 + C → e1040 + f12</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>e465 + f5 → e514 + e368</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>e308 + f7 → f8 + B</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>e418 + B → e371 + f6</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>C + f13 → e853 + f12</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>A + f1 → f2 + e25</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>B + e71 → e319 + f7</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>f2 + e7 → A + e25</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>e1040 + f11 → e1162 + e1163 + e1158</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>e319 + f7 → B + e71</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>e308 + f9 → e614 + e615 + e611</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>e371 + f6 → e418 + B</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>e1040 + f12 → e996 + C</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>f8 + B → e308 + f7</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>e319 + f5 → e372 + e371 + e368</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>f3 + e7 → A + f0</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>e853 + f15 → e1428 + C</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>e1428 + C → e853 + f15</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>A + f10 → e996 + f3</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>e1163 + f11 → e1158 + e1246</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>e418 + f6 → e465 + B</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>A + f0 → f3 + e7</td>
<td></td>
</tr>
</tbody>
</table>

**Johnson et al. (2016) - CRN bisimulation equivalence**

**translation scheme: qian2011_3D_var1.ts**
CRN equivalence verification

<table>
<thead>
<tr>
<th>formal input CRN</th>
<th>interpreted CRN</th>
<th>condensed CRN</th>
<th>enumerated CRN</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 species</td>
<td>3 species</td>
<td>42 species</td>
<td>360 species</td>
</tr>
<tr>
<td>7 reactions</td>
<td>7 non-trivial reactions</td>
<td>32 reactions</td>
<td>668 reactions</td>
</tr>
<tr>
<td>A → A + A</td>
<td>C → C</td>
<td>f14 + C → e1428 + f15</td>
<td></td>
</tr>
<tr>
<td>A + A → A</td>
<td>C + C → C</td>
<td>e853 + f12 → C + f13</td>
<td></td>
</tr>
<tr>
<td>A + B → B + B</td>
<td>A → A</td>
<td>A + f4 → f3 + e71</td>
<td></td>
</tr>
<tr>
<td>B →</td>
<td>A + A → A + A</td>
<td>f2 + e25 → A + f1</td>
<td></td>
</tr>
<tr>
<td>A + C →</td>
<td>A → A</td>
<td>A + e25 → f2 + e7</td>
<td></td>
</tr>
<tr>
<td>C → C + C</td>
<td>A → A</td>
<td>e996 + f3 → A + f10</td>
<td></td>
</tr>
<tr>
<td>C + C → C</td>
<td>B → B</td>
<td>e1428 + f15 → f14 + C</td>
<td></td>
</tr>
<tr>
<td></td>
<td>A + C → A + C</td>
<td>f3 + e71 → A + f4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>B → B</td>
<td>e465 + B → e418 + f6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>B + B → B + B</td>
<td>e614 + f9 → e611 + e730</td>
<td></td>
</tr>
<tr>
<td></td>
<td>C → C + C</td>
<td>e996 + C → e1040 + f12</td>
<td></td>
</tr>
<tr>
<td></td>
<td>A → A</td>
<td>e465 + f5 → e514 + e368</td>
<td></td>
</tr>
<tr>
<td></td>
<td>B + A → A + B</td>
<td>e308 + f7 → f8 + B</td>
<td></td>
</tr>
<tr>
<td></td>
<td>A + A → A + A</td>
<td>e418 + B → e371 + f6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>A + C →</td>
<td>C + f13 → e853 + f12</td>
<td></td>
</tr>
<tr>
<td></td>
<td>B →</td>
<td>A + f1 → f2 + e25</td>
<td></td>
</tr>
<tr>
<td></td>
<td>B + B → B + B</td>
<td>B + e71 → e319 + f7</td>
<td></td>
</tr>
<tr>
<td></td>
<td>A + B → B + A</td>
<td>f2 + e7 → A + e25</td>
<td></td>
</tr>
<tr>
<td></td>
<td>B →</td>
<td>e1040 + f11 → e1162 + e1163 + e1158</td>
<td></td>
</tr>
<tr>
<td></td>
<td>B + C → A + C</td>
<td>e319 + f7 → B + e71</td>
<td></td>
</tr>
<tr>
<td></td>
<td>B →</td>
<td>e308 + f9 → e614 + e615 + e611</td>
<td></td>
</tr>
<tr>
<td></td>
<td>A + B → B + B</td>
<td>e371 + f6 → e418 + B</td>
<td></td>
</tr>
<tr>
<td></td>
<td>A + A → A</td>
<td>e1040 + f12 → e996 + C</td>
<td></td>
</tr>
<tr>
<td></td>
<td>C + C → C + C</td>
<td>f8 + B → e308 + f7</td>
<td></td>
</tr>
<tr>
<td></td>
<td>C + C → C + C</td>
<td>e319 + f5 → e372 + e371 + e368</td>
<td></td>
</tr>
<tr>
<td></td>
<td>A → A</td>
<td>f3 + e7 → A + f0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>B →</td>
<td>e853 + f15 → e1428 + C</td>
<td></td>
</tr>
<tr>
<td></td>
<td>A → A + A</td>
<td>e1428 + C → e853 + f15</td>
<td></td>
</tr>
</tbody>
</table>

Johnson et al. (2016) - CRN bisimulation equivalence translation scheme: qian2011_3D_var1.ts
CRN equivalence verification

formal input CRN
3 species
7 reactions

A → A + A
A + A → A
A + B → B + B
B →
A + C →
C → C + C
C + C → C

interpreted CRN
3 species
7 non-trivial reactions

C → C
C + C → C
A → A
A → A
A + A → A + A
A → A
C → C
A → A
B → B
A + C → A + C
B → B
B + B → B + B
C → C + C
A → A
B + A → A + B
A + A → A + A
A + C →
A + B → B + A
B →
B + B → B + B
A + C → A + C
B → B
A + B → B + B
A + A → A
C + C → C + C
C + C → C + C
A → A
B → B
A → A + A

condensed CRN
42 species
32 reactions

f14 + C → e1428 + f15
e853 + f12 → C + f13
A + f4 → f3 + e71
f2 + e25 → A + f1
A + e25 → f2 + e7
e996 + f3 → A + f10
e1428 + f15 → f14 + C
f3 + e71 → A + f4
e465 + B → e418 + f6
e614 + f9 → e611 + e730
e996 + C → e1040 + f12
e465 + f5 → e514 + e368
e308 + f7 → f8 + B
e418 + B → e371 + f6
C + f13 → e853 + f12
A + f1 → f2 + e25
B + e71 → e319 + f7
f2 + e7 → A + e25
e1040 + f11 → e1162 + e1163 + e1158
e319 + f7 → B + e71
e308 + f9 → e614 + e615 + e611
e371 + f6 → e418 + B
e1040 + f12 → e996 + C
f8 + B → e308 + f7
e319 + f5 → e372 + e371 + e368
f3 + e7 → A + f0
e853 + f15 → e1428 + C
e1428 + C → e853 + f15
A + f10 → e996 + f3
e1163 + f11 → e1158 + e1246
e418 + f6 → e465 + B
A + f0 → f3 + e7

enumerated CRN
360 species
668 reactions

Johnson et al. (2016) - CRN bisimulation equivalence translation scheme: qian2011_3D_var1.ts
Designing Sequences

Toehold long domain free energy (arbitrary units)

Global energy barrier

Local energy barrier

Folding pathway events
**Designing Sequences**

**NUPACK** is a growing software suite for the analysis and design of nucleic acid structures, devices, and systems.

The NUPACK web application enables analysis and design of the equilibrium base-pairing properties of one or more test tubes of interacting nucleic acid strands:

Please cite the web application and algorithms appropriately; usage statistics are an important component in helping to secure funding for NUPACK development. We are happy to provide advice and technical support.

— The NUPACK Team

News: Constrained multistate test tube design for reaction pathway engineering is now published! (pdf, supp info, source code, user guide)
Designing Sequences

input X

poor sequence for a signal strand

good sequence for a signal strand

MFE structure at 25.0 C

Free energy of secondary structure: -10.70 kcal/mol

MFE structure at 25.0 C

Free energy of secondary structure: 0.00 kcal/mol
MULTISTRAND is a software package for simulating the kinetics of multiple interacting nucleic acid strands. It is developed at the Winfree lab at the California Institute of Technology.

Key Features

- Kinetic simulations of nucleic acids as random walk on thermodynamic energy model
- Supports multiple interacting strands
- Equilibrium consistent with NUPACK
- Various usage modes to study kinetic trajectories
- Distributed as a Python package
- MIT License
Tutorial Outline

- Review of strand displacement
- Building and composing logic gates
- Tools for designing and verifying circuits
- Robustness of strand displacement
Why is this circuit not *robust*?

![Graph showing concentration over time for ON and OFF cases.](image)
What causes signal leak?
Problem 1: Molecules are not perfect

Imperfect strands from imperfect synthesis
Problem 1: Molecules are not perfect

Imperfect strands from imperfect synthesis
Problem 1: Molecules are not perfect

translator cascade with perfect molecules

F1

\[ \delta x_1^* \quad x_2^* \quad y_1 \]

F2

\[ \delta x_2^* \quad y_1^* \quad y_2 \]
Problem 1: Molecules are not perfect

translator cascade with perfect molecules

\[ \delta x_1^* \]

F1

\[ \delta x_2^* \]

\[ y_1^* \]

\[ y_2 \]

F2

translator cascade with imperfect molecules

\[ \delta x_1^* \]

F1

\[ \delta x_2^* \]

\[ y_1^* \]

\[ y_2 \]

F2
Problem 1: Molecules are not perfect

Translator cascade with perfect molecules

Translator cascade with imperfect molecules

displacement now possible
(Partial) solution to Problem 1
(Partial) solution to Problem 1
Complexes can also be purified by gel.
Problem 2: Spurious reactions occur (even with perfect molecules)

\[ X \rightarrow Y \]

Y has been spuriously “produced”
Some rough energy accounting

State 1: before leak
- 2 bound long domains
- 2 complexes

State 2: after leak
- 2 bound long domains
- 2 complexes
Some rough energy accounting

State 1: before leak
- 2 bound long domains
- 2 complexes

△Energy
0 bound long domains
0 units of entropy

State 2: after leak
- 2 bound long domains
- 2 complexes
A Motivating Question

Can we rationally design *composable, leakless* DSD gates?
A Motivating Question

Can we rationally design *composable, leakless* DSD gates?
A Motivating Question

Can we rationally design *composable, leakless* DSD gates?
What do we mean by leakless?

“Golf funnel with deep groove” pathway

(Partial) solution to Problem 2

For a redundancy parameter $N$, there exist translator and AND gates using $N$ long domains that have the following property:

> **even at thermodynamic equilibrium,**

the net leak decreases exponentially with $N$.

Typical translator using “Single Long Domain” (SLD)

- Designed pathways: bimolecular
- Leak pathways: bimolecular

DLD translator using “Double Long Domain” (DLD)

- Designed pathways: bimolecular
- Leak pathways: trimolecular
Typical translator using “Single Long Domain” (SLD)

- Designed pathways: bimolecular
- Leak pathways: bimolecular

DLD translator using “Double Long Domain” (DLD)

- Designed pathways: bimolecular
- Leak pathways: trimolecular
Typical translator using “Single Long Domain” (SLD)

- Designed pathways: bimolecular
- Leak pathways: bimolecular

DLD translator using “Double Long Domain” (DLD)

- Designed pathways: bimolecular
- Leak pathways: trimolecular

Lengthening recognition domains does not help
DLD translators are intrinsically less “leaky”
DLD translators are intrinsically less “leaky”
DLD translators are intrinsically less “leaky”
Can we generalize the DLD motif?
Translator using Triple Long Domain (TLD) motif

Three fuel complexes must combine to activate output signal.

\[ \text{Energy} \quad \Delta = 0 \text{ bound long domains } -2 \text{ units of entropy} \]
Translator using $N$ Long Domain (NLD) motif

$N$ fuel complexes must combine to activate output signal.

$\triangle$ Energy to leak state
0 bound long domains
$-(N-1)$ units of entropy
Building OR circuits from DLD translators

Boya Wang, Thachuk, Ellington, Winfree, David Soloveichik. (In Review)
Effective Design Principles for Leakless Strand Displacement Systems

[fuel]=[input]=1000nM
[reporter]=500nM

desired triggering: reporter+fuels+input
leak: reporter+fuels
Building OR circuits from DLD translators

[a] input X1
[b] input X2
[c] input X3
[d] input X4
[e] input X5
[f] input X6

[fuel]=[input]=1000nM
[reporter]=500nM

Boya Wang, Thachuk, Ellington, Winfree, David Soloveichik. (In Review)
Effective Design Principles for Leakless Strand Displacement Systems
Tutorial Outline

- Review of strand displacement
- Building and composing logic gates
- Tools for designing and verifying circuits
- Robustness of strand displacement
- (Bonus) DSD circuits the easy way
Does it need to be this difficult to build a circuit?
# Molecular breadboard 1.0

<table>
<thead>
<tr>
<th>input signals</th>
<th>reaction gates</th>
<th>wires</th>
<th>reporters</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1 B1</td>
<td>![1]</td>
<td>X1----R1</td>
<td>R1</td>
</tr>
<tr>
<td>A2 B2</td>
<td>![3]</td>
<td>Y1----R2</td>
<td>R2</td>
</tr>
<tr>
<td>A3 B3</td>
<td>![4]</td>
<td>Y3----A2</td>
<td></td>
</tr>
<tr>
<td>A4 B4</td>
<td>![7]</td>
<td>X2----B1</td>
<td></td>
</tr>
<tr>
<td>A7 B7</td>
<td></td>
<td>X4----B2</td>
<td></td>
</tr>
</tbody>
</table>

Built using leakless motif
Molecular breadboard 1.0

Load breadboard components onto 384-well plate

Built using leakless motif
Molecular breadboard 1.0

Load breadboard components onto 384-well plate

Built using leakless motif
### Molecular breadboard 1.0

#### Input Signals
- A1 B1
- A2 B2
- A3 B3
- A4 B4
- A7 B7

#### Reaction Gates
- 1
- 2
- 3
- 4
- 7

#### Wires
- X1 --- R1
- X2 --- B1
- X3 --- B1
- Y1 --- R2
- Y2 --- R1
- Y3 --- A2
- Y4 --- A2
- X4 --- B2
- X4 --- B2
- X7 --- A1
- X3 --- R2

#### Reporters
- R1
- R2

---

**Breadboard Plate**

**Built using leakless motif**

---

**Load breadboard components onto 384-well plate**

---

**Circuit 1**

- B --- 4
- A --- 3
- 3
- 1
- 1
- x2
- x3
- ~x3

**Circuit 2**

- B --- 4
- A --- 3
- ~r
- 1
- 2
- R2
- Y

---

**Built using leakless motif**

---
Testing breadboard components

- Typical DSD circuits are 50nM - 200nM concentration (our circuits can operate at these concentrations)

- To demonstrate *robustness*, all experiments will be at 2uM (~20x higher than typical concentrations)
AND gate
Ideal AND gate simulation @ 2 µM
Ideal AND gate simulation @ 2 µM

Designed reaction half-life on order of seconds
Ideal AND gate simulation @ 2 µM

Designed reaction half-life on order of seconds

- and(0,0)
- and(0,1)
- and(1,0)
- and(1,1)
Ideal AND gate simulation @ 2 µM

- Designed reaction half-life on order of seconds

- Ideal AND gate simulation @ 2 µM

- and(0,0)
- and(0,1)
- and(1,0)
- and(1,1)
Ideal AND gate simulation @ 2 μM

- Designed reaction half-life on order of seconds
- Spurious reactions not observable on order of hours
[fuel]=[input]=2uM, [reporter]=1uM

AND gate @ 2 µM

[fuel]=[input]=2uM, [reporter]=1uM
AND gate @ 2 µM

Half-time completion on order of tens of seconds

[fuel]=[input]=2µM,  [reporter]=1µM
AND gate @ 2 µM (12 hours)

[fuel]=[input]=2uM, [reporter]=2.5uM
AND gate @ 2 µM (12 hours)

[fuel]=[input]=2uM,  [reporter]=2.5uM
Multiplexer-Demultiplexer

The diagram illustrates a multiplexer-demultiplexer circuit with inputs B, s, A, and r. The circuit consists of two main components:

1. **2-in, 2-out Multiplexer**: This component combines two inputs (B and A) with select inputs (s and r) to produce two outputs (Y and X).
2. **2-input MUX**: The first multiplexer block with inputs B and A and select s.
3. **2-output DEMUX**: The second multiplexer block with inputs Y and X and select r.

The table below shows the truth table for the multiplexer-demultiplexer, with inputs r, s, A, and B, and outputs X and Y.

<table>
<thead>
<tr>
<th>r</th>
<th>s</th>
<th>A</th>
<th>B</th>
<th>X</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

The graph on the right side of the diagram shows the connections and outputs (R1 and R2) for different input states.
Multiplexer-Demultiplexer

- 2-in, 2-out Multiplexer
- 2-input MUX
- 2-output DEMUX

<table>
<thead>
<tr>
<th>r</th>
<th>s</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0.0</td>
<td>0.0</td>
<td>0.4</td>
<td>0.8</td>
</tr>
</tbody>
</table>

 fluorscence (a.u.)
Multiplexer-Demultiplexer
Multiplexer-Demultiplexer

2-in, 2-out Multiplexer  2-input MUX  2-output DEMUX

X

Y

<table>
<thead>
<tr>
<th>r s A B</th>
<th>6 minutes</th>
<th>720 minutes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 1 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 1 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1 0 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1 0 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1 1 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1 1 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 0 0 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 0 0 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 0 1 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 0 1 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1 0 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1 0 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1 1 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1 1 1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

fluorescence (a.u.)

<table>
<thead>
<tr>
<th>r s A B</th>
<th>6 minutes</th>
<th>720 minutes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 1 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 1 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1 0 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1 0 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1 1 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1 1 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 0 0 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 0 0 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 0 1 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 0 1 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1 0 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1 0 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1 1 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1 1 1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

fluorescence (a.u.)
Large circuits that are fast

In all tested cases, the output went to the correct ON or OFF state. A three-OR cascade (fig. S6, A and B) and a four-OR cascade (fig. S6, C and D) also worked. The delay time required for circuit computation increased linearly with the number of layers (Fig. 3A). However, once the threshold for the output gate was exceeded, the signal increased at roughly the same rate as in the smaller circuit (Fig. 3B). In a circuit with four layers, two AND gates, and three OR gates, with 12 different combinations of inputs, the output went to clear and correct ON or OFF states in 8 hours (Fig. 3C).

Because integrating gates support multiple inputs and amplifying gates support multiple outputs, logic gates built from a pair of them can easily support fan-in and fan-out. In a circuit with a four-input OR gate, only when all inputs from the upstream OR gates were OFF did the output stay OFF (Fig. 3D). In a circuit with a four-output OR gate, each output copied the correct logic from the upstream OR gate (Fig. 3E). Circuits with a four-input AND gate and a four-output AND gate are shown in fig. S8C and fig. S9C, respectively.

To demonstrate a digital circuit with an interesting function, we built a circuit that computes the floor of the square root of a four-bit binary number (Fig. 4A). It is not an optimized digital logic circuit; it is designed to showcase AND, OR, NOT, NAND, NOR, fan-in, and fan-out of logic gates, as well as fan-out of input signals. NOT gates are difficult to implement directly using representations where the ON or OFF state of an input is determined by the presence or absence of a single DNA species: A circuit might compute a false output before all input strands are added, because NOT gates already produce ON signals in the absence of their inputs, and for use-once circuits (such as seesaw circuits), computations cannot be undone. Therefore, we use dual-rail logic (fig. S10B). Each input is replaced by a pair of inputs, representing logic ON and OFF separately. Each logic gate is replaced by a pair of AND or OR gates. (Taking the NOR gate as an example, output being OFF is the OR of both inputs being ON; output being ON is the AND of both inputs being OFF.) Initially, the pair of inputs is absent, indicating that the logic value of this signal is unknown. At the beginning of computation, one input of the pair will be added, indicating either logic ON or OFF. In this way, no computation will take place before the input signals arrive. With dual-rail logic, any AND-OR-NOT circuit can be transformed into an equivalent circuit with AND or OR gates only. Then, any AND-OR circuit can be further transformed into an equivalent seesaw circuit.
Breadboard compiler produces a mixing protocol.

All 8 input combinations for 6 circuits:

- 0x1
- 0x3
- 0xE
- Rule 30
- 0x41
- 0xC9

Destination plate
First measurement 6 minutes after mixing start time
Molecular Circuit Breadboard

Roadmap
## Molecular Breadboard 2.0:

### More components

<table>
<thead>
<tr>
<th>input signals</th>
<th>reaction gates</th>
<th>wires</th>
<th>reporters</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1 B1</td>
<td>1 2</td>
<td>X1 –- A3</td>
<td>R1 R5</td>
</tr>
<tr>
<td>A2 B2</td>
<td>3 4</td>
<td>X1 –- B4</td>
<td>R2 R6</td>
</tr>
<tr>
<td>A3 B3</td>
<td>5 6</td>
<td>Y1 –- A4</td>
<td>R3 R7</td>
</tr>
<tr>
<td>A4 B4</td>
<td></td>
<td>Y1 –- A7</td>
<td>R4 R8</td>
</tr>
<tr>
<td>A5 B5</td>
<td></td>
<td>Y1 –- B12</td>
<td></td>
</tr>
<tr>
<td>A25 B25</td>
<td>24 25</td>
<td>X24 –- R7</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Y25 –- R1</td>
<td></td>
</tr>
</tbody>
</table>
Molecular Breadboard 2.0: More circuits

Breadboard 2.0 can realize > 130 K circuits
Molecular Breadboard 2.0:

Larger circuits
Building circuits with feedback loops

Chemical Reaction Networks

Asynchronous Sequential Logic Circuits

Finite state machines
Providing input amplifiers & output signal restoration

Linear input amplifier

Exponential input amplifier

Output signal restoration

New component
Increased speed
Robustness to error
Automation

http://DSDbreadboard.org

Diagram of a circuit with labels B, s, A, r, Y, and X, showing connections and expected outputs over time.
Related talks & posters @ DNA 24

Dominic Scalise, Nisita Dutta and Rebecca Schulman
DNA strand–displacement buffers

Si-Ping Han, Lisa Scherer, Matt Gethers, Marwa Ben Hadj Salah, Rebecca Mancusi, Sahil Sagar, Robin Hu, Julia Derogatis, Ya–Huei Kuo, Guido Marcucci, John Rossi and William A. Goddard iii
Development and optimization of strand displacement based conditional small interfering RNAs for operation inside mammalian cells

Eyal Nir, Yaron Berger and Miran Liber
Computer Controlled DNA Bipedal Walker that Perform Several Steps a Minute

Abhinav Singh and Manoj Gopalkrishnan
EM Algorithm with DNA Molecules

Wooli Bae, Thomas Ouldridge and Guy–Bart Stan
Autonomous generation of multi-stranded RNA complexes for synthetic molecular circuits

Yan Shan Ang and Lin–Yue Lanr Yung
Design of Split Proximity Circuit as a Plug–and–Play Translator for Discriminating Single Nucleotide Mutation

Yan Shan Ang and Lin–Yue Lanr Yung
Dynamically Elongated Association Toehold for Tuning Circuit Kinetics and Thermodynamics

Patrick Irmisch and Ralf Seidel
Modelling DNA–strand displacement reactions in the presence of base–pair mismatches

Boya Wang and David Soloveichik
Experimentally characterizing the design space of strand displacement translators with toehold–size clamps

Allison Tai and Anne Condon
Error–free stable computation with stack–supplemented chemical reaction networks

Kevin Cherry, Gokul Gowri and Lulu Qian
DNA–based neural networks that learn from their molecular environment

Robert F. Johnson and Erik Winfree
Using Bisimulation for Verification of Polymer Reaction Networks
Acknowledgments

• **Winfree lab** (Caltech)

• **Soloveichik lab** (University of Texas at Austin)

• **Qian lab** (Caltech)

• **Murray lab** (Caltech)

• Thanks to DNA 24 organizers for the invitation
Tools discussed in tutorial

ABC: logic synthesis and verification
https://people.eecs.berkeley.edu/~alanmi/abc

VisualDSD
https://lepton.research.microsoft.com/webdna

Nuskell compiler framework
https://github.com/DNA-and-Natural-Algorithms-Group

DSD breadboard
http://dsdbreadboard.org (online later this year)
(a) SLD intended reaction

(b) SLD leak reaction

**SLD leak**

<table>
<thead>
<tr>
<th>SLD leak</th>
<th>before</th>
<th>after</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of base pairs</td>
<td>49</td>
<td>≈ 50</td>
</tr>
<tr>
<td>separate components</td>
<td>3</td>
<td>= 3</td>
</tr>
</tbody>
</table>

Waste composition:
- Waste 1
- Waste 2
- Waste 3
(a) **DLD intended reaction**

- $X \rightarrow waste_1$
- $F1 \rightarrow waste_2$
- reporter $\rightarrow waste_3$

(b) **DLD leak reaction**

- $F1 \rightarrow waste_1$
- $F2 \rightarrow waste_2$
- reporter $\rightarrow waste_3$

### Table: DLD leak

<table>
<thead>
<tr>
<th>Number of Base Pairs</th>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>$79 \approx 80$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Separate Components

- $3 > 2$

**Entropic Penalty**
TLD leak

before

describe components
166

number of base pairs
165

separate components
4

> 2

entropic penalty

TLD leak reaction

reporter

reporter.top

TLD intended reaction

reporter

reporter.top

waste1

waste2

waste3

waste4

F1

F1.top

F2

F2.top

F3

reporter

Y

Y_complex

waste

waste

waste

waste