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

input X

input Y

AND gate
Strand Displacement Cascades Example: AND gate

release Z if and only if X and Y are present

input X

input Y

bind

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

input Y
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
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 $Z$

waste

waste
Strand Displacement Cascades Example: AND gate

release Z if and only if X and Y are present

before

after

input X

output Z

input Y

waste

AND gate

waste
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

Composable
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

\[ A \rightarrow B \rightarrow X \]

\[ Y \rightarrow W \rightarrow Z \]
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”): $X \rightarrow Y$

input $X$

F1

F2
Sequence Independence

Translator (a “wire”): \( X \to Y \)
Sequence Independence
Translator (a “wire”): $X \rightarrow Y$
Reading Output
$$h v_{\text{ex}}$$

![Chemical Structure of Cy3](attachment:image.png)
displace

quencher

flourophore
displace

quencher

fluorophore

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
Signal Fanout Gates

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

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

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

5’-ACAGATCACCAGATCATTATCAGAG-3’
strand representing signal S

Fanout 2

Fanout 3

Fanout 4
Handling OR gates

\[ \text{AND}( A1, \text{OR}( B1, \text{AND}( A2, B2 ) ) ) \]
Handling OR gates

Relevant gates function as “AND of OR” gates

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

Dual-rail input, Dual-rail output

5'-CCCCCCC-3'  
S

5'-TTTTTTTT-3'  
T

Dual-rail input, Dual-rail output

S

¬S

T

¬T

S

¬S

T

¬T

S

¬S

T

¬T

R1

R2

X

X

X
Dual rail logic

AND

OR

NOT
Handling NOT gates

Dual-rail input, Dual-rail output

5'-CCCCCCC-3'
S

5'-TTTTTTTT-3'
\neg S
Handling NOT gates

5'-CCCCCCC-3'
S

5'-TTTTTTTT-3'
\sim S

Dual-rail input, Dual-rail output

Dual-rail input
Handling NOT gates

5'-CCCCCCC-3'

S

5'-TTTTTTTT-3'

S

Dual-rail input, Dual-rail output

Dual-rail input

5'-CCCCCCC-3'

S

5'-TTTTTTTT-3'

S

S

T

X

Dual-rail input, Dual-rail output

Dual-rail input

S

T

X
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
module FullAdder(a, b, cin, cout, sum); 
input a, b, cin; // inputs
output cout, sum;    // output
wire w1, w2, w3, w4; // internal nets
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?
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>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

**Nuskell project**
- **CRN-to-DSD translation schemes**
- **Nuskell project**
  - **A ⇋ B**
  - **CRN trajectory equivalence**
  - **CRN enumeration**

**Peppercorn project**
- **CRN condensation**
- **Domain-level reaction rates**
- **Peppercorn project**
  - **condensed reaction rates**
  - **nucleotide-level reaction rates**

**KinDA project**
- **nucleotide sequences**
- **KinDA project**

**References**
- 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

- **Binding/Unbinding**
  - Forward: $a \rightarrow b \rightarrow a^*$
  - Reverse: $a^* \rightarrow a \rightarrow b$

- **Remote Toehold Branch Migration**
  - Forward: $a^* \rightarrow a \rightarrow c^* \rightarrow a$
  - Reverse: $a \rightarrow a^* \rightarrow c^* \rightarrow a$

- **3-Way Branch Migration**
  - Forward: $a \rightarrow b \rightarrow c\rightarrow a^*$
  - Reverse: $a^* \rightarrow b^* \rightarrow c^*$

- **4-Way Branch Migration**
  - Forward: $a \rightarrow b \rightarrow c \rightarrow d \rightarrow a^*$
  - Reverse: $a^* \rightarrow b^* \rightarrow c^* \rightarrow d^*$

Additional Resources:

Grun et al. 2014
Designing Sequences

TGTACCTGTCGA
ACATGGACAGCTATAC

Global energy barrier

Local energy barrier

Folding pathway events

Toehold long domain

Free energy (arbitrary units)

-15
-12
-10
-8
-6
-4
-2
0
2
4
6
8
10
12
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

MFES structure at 25.0 C

Free energy of secondary structure: -10.70 kcal/mol

MFES structure at 25.0 C

Free energy of secondary structure: 0.00 kcal/mol
Multistrand.org to determine reaction rates

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.

DNA and Natural Algorithms Group @ Caltech

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?
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

F1
\[ \delta x_1^* \quad x_2^* \quad y_1 \]
F2
\[ \delta x_2^* \quad y_1^* \quad y_2 \]

translator cascade with imperfect 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

F1
\[\delta x_1^* \quad x_2^* \quad y_1 \quad y_2 \]

F2
\[\delta x_2^* \quad y_1^* \]

translator cascade with imperfect molecules

F1
\[\delta x_1^* \quad x_2^* \quad y_1 \quad y_2 \]

F2
\[\delta x_2^* \quad y_1^* \]

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

$X \rightarrow Y$

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

State 2: after leak
- 2 bound long domains
- 2 complexes

$\Delta$ Energy
- 0 bound long domains
- 0 units of entropy
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 \textit{composable}, \textit{leakless} DSD gates?
What do we mean by leakless?

“Golf funnel with deep groove” pathway
For a redundancy parameter $\mathbf{N}$, there exist \textit{translator} and \textit{AND} gates using $\mathbf{N}$ long domains that have the following property:

\textit{even at thermodynamic equilibrium},

the net leak decreases exponentially with $\mathbf{N}$.

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.

$\Delta$ Energy

0 bound long domains

-2 units of entropy
Translator using \( N \) Long Domain (NLD) motif

\[ F_1 \quad x_{1bc} \quad x_2 \quad \ldots \quad x_n \quad y_1 \]

\[ F_2 \quad x_{1}^* \quad x_{2}^* \quad x_3 \quad \ldots \quad y_1 \quad y_2 \]

\[ F_n \quad x_{nbc} \quad y_1 \quad \ldots \quad y_{n-1} \quad y_n \]

\( N \) fuel complexes must combine to activate output signal.

\( \square \) Energy to leak state

0 bound long domains

\(-(N-1)\) units of entropy
[fuel]=[input]=1000nM
[reporter]=500nM

Boya Wang, Thachuk, Ellington, Winfree, David Soloveichik. (In Review)
Effective Design Principles for Leakless Strand Displacement Systems
Building OR circuits from DLD translators

(a) Input X1

[b] Input X2

(c) Input X3

(c) Input X4

(d) Input X5

(e) 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

input signals
A1  B1
A2  B2
A3  B3
A4  B4
A7  B7

reaction gates
1
2
3
4

wires
X1 ----- R1
Y1 ----- R2
X2 ----- B1
X2 ----- R2
X3 ----- B1
X3 ----- R2
Y3 ----- A2
Y3 ----- B2
X4 ----- B1
X4 ----- B2
Y4 ----- A2
Y4 ----- R2
X7 ----- A1

reporters
R1
R2

Built using leakless motif
Molecular breadboard 1.0

Load breadboard components onto 384-well plate

Built using leakless motif
# 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 2</td>
<td>X1—R1 Y3—A2</td>
<td>R1</td>
</tr>
<tr>
<td>A2 B2</td>
<td>3 4</td>
<td>Y1—R2 Y3—B2</td>
<td>R2</td>
</tr>
<tr>
<td>A3 B3</td>
<td></td>
<td>X2—B1 X4—B1</td>
<td></td>
</tr>
<tr>
<td>A4 B4</td>
<td></td>
<td>X2—R2 X4—B2</td>
<td></td>
</tr>
<tr>
<td>A7 B7</td>
<td></td>
<td>Y2—R1 Y4—A2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X3—B1 Y4—R2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X3—B4 X7—A1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X3—R2</td>
<td></td>
</tr>
</tbody>
</table>

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

**Built using leakless motif**
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>2</td>
<td>Y1----R2</td>
<td>R2</td>
</tr>
<tr>
<td>A3 B3</td>
<td>3</td>
<td>X2----B1</td>
<td></td>
</tr>
<tr>
<td>A4 B4</td>
<td>4</td>
<td>Y2----R1</td>
<td></td>
</tr>
<tr>
<td>A7 B7</td>
<td>7</td>
<td>X3----B1</td>
<td></td>
</tr>
</tbody>
</table>

Load breadboard components onto 384-well plate

Built using leakless motif

Breadboard plate

Circuit 1

Circuit 2
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

- red: and(0,0)
- blue: and(0,1)
- purple: and(1,0)
- gray: and(1,1)
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

- 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

Fluorescence (a.u.) vs. time (minutes)

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

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

Half-time completion on order of tens of seconds

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

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

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

Diagram showing the multiplexer-demultiplexer circuit with inputs B, s, A, and r, and outputs Y and X. The circuit includes a truth table below the diagrams.
Multiplexer-Demultiplexer

X

<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>
</tbody>
</table>

fluorescence (a.u.)
Multiplexer-Demultiplexer

[Diagram showing a multiplexer-Demultiplexer circuit with inputs A, B, and r, and outputs Y and X, along with a table and graph showing fluorescence values]
Multiplexer-Demultiplexer

[Diagram of multiplexer-demultiplexer circuit]

[X-axis graph showing fluorescence (a.u.)]
Multiplexer-Demultiplexer

Diagram showing a multiplexer and demultiplexer circuit with inputs B, S, and A, and outputs Y and X. The circuit includes logic gates and connections labeled as r and s. The graphs on the right show fluorescence data for combinations of inputs r, s, A, and B, with red bars indicating 6 minutes and blue bars indicating 720 minutes.

Legend:
- Blue: 2-in, 2-out Multiplexer
- Orange: 2-input MUX
- Red: 2-output DEMUX
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.

Acoustic Liquid Handler

All 8 input combinations for 6 circuits

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

Roadmap
Molecular Breadboard 2.0:
More components

input signals
A1   B1
A2   B2
A3   B3
A4   B4
A5   B5
A25  B25

reaction gates
1
3
5
2
4
6

wires
X1 --- A3
X1 --- B4
Y1 --- A4
Y1 --- A7
Y1 --- B12

reporters
R1
R2
R3
R4
R5
R6
R7
R8

X24 --- R7
Y25 --- R1
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
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 Lanry Yung
Design of Split Proximity Circuit as a Plug-and-Play Translator for Discriminating Single Nucleotide Mutation

Yan Shan Ang and Lin-Yue Lanry 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)*