An experimental realization of a universal computer

Nicolas Schabanel
CNRS
LIP & IXXI - ÉNS de Lyon

Slides mainly borrowed from Damien Woods et al (Nature 2019)
Single Stranded Tiles Nanotubes
Single stranded Nanotubes

10-helix nanotube schematic, Yin et al. ’08

4 domains = 4 glues

b9
a9

a8*
b8*
Single stranded Nanotubes

10-helix nanotube schematic, Yin et al. ’08

4 domains = 4 glues
Growing them

Seed = DNA Origami encoding an input

STT assembly = Computation

S3 System design abstraction level: binding-domain schematics

In this section we give the binding-domain schematics level of abstraction, which is below the abstraction level of aTAM proofreading tiles and above that of DNA sequence design.

S3.1 Strand-level system design of SST lattice, input-adapter strands, seed attachment

Concerns at this level of abstraction include choosing (a) appropriate domain lengths (in number of bases) for good DNA crossover positions between helices and for structural stability, for SSTs, input adapters, and origami seed staples/ scaffolds, and (b) appropriate positions on SSTs for biotin modifications. As shown in Figure 1 in the main text, each square tile, post-proofreading, is mapped to an abstract SST strand. We used the SST motif from ref. [13], which has domains of length 10 and 11 bases. Altogether 355 SSTs were designed. The number 355 was calculated in Section S2.3.1, at the abstraction level of 2x2 proofreading tiles; here at the SST binding-domain level of abstraction we simply convert each square (proofreading) tile to an abstract SST strand with four binding domains.

Figure S10 gives the domain-level design of our SST set, and the caption describes some of the key features. SST strands are designed to grow a nanotube lattice, so that the top of the SST lattice in Figure S10 is

U1
U2
U3
U4
U5
U6
U7
U8

wire 1
wire 2
wire 3
wire 4
wire 5

= 1
= 0

Figure S10: SST lattice showing from left to right the DNA origami seed scaffold (light blue), input-adapter strands (red, plus other colours), and SSTs (yellow, brown and blue), seam (grey), seam block (grey), and biotin locations (red and green disks/circles, coloured according to their wire index being even or odd). Each of the four domains on an SST are colour coded: yellow represents bit 1, brown represents bit 0, blue represents a domain that is unique to the proofreading block it is in. The presence of biotins is indicated by red and green solid disks, and is used to denote the encoding of a 1 bit on the proofreading block output domains that are on the same wire. A biotin is present on a strand if its closest bit-encoding output domain, along the same strand, is yellow (encodes bit 1), and otherwise the biotin is not present (denoted as a hollow circle, to help visualize all possible positions where biotins could be on other arbitrary computations). Relevant portions of the DNA origami seed scaffold strand are shown as light blue on the left-hand side. Input-adapter strands are shown with red on the domains connecting to the scaffold, and the domains binding to tiles are colored using the same bit convention as the tiles. Although input adapters do not have biotin modifications, red or green 'x's are used to indicate locations were biotins would have been had we chosen to include them.
Seeded growth: barrier to nucleation at [tile]=100nM

Experiments give a (narrow) temperature range at which we make good-quality, long, nanotubes!

Higher concentration => longer nanotubes!

Lower concentration => bigger barrier to nucleation! 2.5°C gap!

Too hot: free tiles do not bind to each other
Too cold: nanotubes along with blobs & nanotube tangles
Just right: long nanotubes

Fluorescence microscopy images: cy3 label
Error bars show SEM for n=5 experiments for blue, and n=2 for red

All scale bars 10µm

Each datapoint is a separate ~24 hour temperature hold experiment

8-helix, 100nM, 53.0°C
12-helix, 100nM, 52.2°C
14-helix, 100nM, 52.2°C
16-helix, 100nM, 52.2°C
8-helix, 1µM, 57.0°C
10-helix, 1µM, 57.0°C
12-helix, 1µM, 56.2°C
14-helix, 1µM, 56.2°C
16-helix, 1µM, 56.2°C
Seeded growth: barrier to nucleation at [tile] = 100nM

Growth from seed only

[tile] = 100nM

Lower concentration => bigger barrier to nucleation! 2.5°C gap!

Everything melts

Everything sticks together

Too cold: nanotubes along with blobs & nanotube tangles

Just right: long nanotubes

Too hot: free tiles do not bind to each other

Seeds: 8-helix nanotubes, Alexa647 labelled

growth from seed: 8-helix tiles, Cy3 labelled

scale bar: 10µm. 100nM tile concentration

Controls: 0 seed nanotubes => 0 nanotubes/image

Many nanotubes (per image)

0 nanotubes (per image)

Take pre-formed seed nanotubes, at room temperature and heat to temperature T

Single stranded

Many nanotubes (per image)

Seed

End-to-end joining

-scale bars 10µm. ~24 hour temperature hold experiments. cy3 label

Damien Woods
Seeded growth only

Many nanotubes were observed at temperatures below the nucleation temperature, while single-stranded DNA was present at temperatures above the melting temperature. Between these two temperatures, seeded growth occurred. The diagram illustrates the temperature range where seeded growth may occur, with data from Figure S27(b) showing a temperature range of approximately 2.5°C.

Seeded growth experiments were performed using pre-formed 8-helix SST nanotubes with one tile type having a 5′ cy3 modification for visualization via fluorescence microscopy. Without seeds, nanotubes did not form.

Two distinct temperature regimes were observed: a high temperature range where no structures were visible (nanotubes melted) and a low temperature range where nanotubes were visible (not melts). The melting temperature refers to the transition temperature between these two ranges, which lies in the range 58–60°C in Figure S27(a), depending on nanotube circumference.

For a given circumference, a larger gap between the nucleation temperature and melting temperature is consistent with a larger kinetic barrier to nucleation. Although some circumferences showed a measurable gap between their nucleation and melting temperature at 1µM, the gap was deemed too small for seeded growth.

We hypothesized that a lower tile type concentration might result in a larger gap between nucleation and melting temperatures for SST nanotubes. Initial experiments showed that at a tile type concentration of 100 nM, SST nanotubes formed well, but at significantly lower concentration yields were low and nanotubes were short. Hence, we chose a monomer concentration of 100 nM for each tile type.

Data set II: Tile type concentration of 100 nM. We repeated the experiments for h ∈ {8, 12, 14, 16} at a lower tile type concentration of 100 nM (instead of 1µM). Figure S27(b) shows the data. There were two clear differences between the 100 nM and 1µM data sets: the nucleation and melting temperature for the 100 nM set is significantly lower than that of the 1µM data set. The 100 nM data set has a wider gap between nucleation and melting temperatures than the 1µM data set. The gap of roughly 2.5°C between nucleation and melting temperatures was deemed sufficient for our intended seeded growth experiments.

The largest circumference tested was 16-helix, hence we used a 16-helix DNA nanotube implementation of IBC, which via our abstraction corresponded to a 6-bit IBC.
Imaging the results
Principle of Atomic Force Microscopy

The microscope works by scanning the surface with a sharp probe and gently touching the DNAs that arrange on the mica.

(Artwork: Ebbe Andersen- Slide by Cody Geary)
Laser deflection
The forces involved in AFM

They are interaction forces between the atoms of the end of the tip and the atoms on the sample surface.
Tip convolution

Tip radius 2-20 nm

\[ W = w + 2 \sqrt{h(2R-h)} \]

\[ W = 2 \sqrt{2Rh} \]

Lateral Dimension(um)
High resolution imaging

The Chemical Structure of a Molecule Resolved by Atomic Force Microscopy
Leo Gross, et al.
Science 325, 1110 (2009);
DOI: 10.1126/science.1176210

Fig. 1. STM and AFM imaging of pentacene on Cu(111). (A) Ball-and-stick model of the pentacene molecule. (B) Constant-current STM and (C and D) constant-height AFM images of pentacene acquired with a CO-modified tip. Imaging parameters are as follows: (B) set point $I = 110 \ pA, V = 170 \ mV$; (C) tip height $z = -0.1 \ \AA$ (with respect to the STM set point above Cu(111)), oscillation amplitude $A = 0.2 \ \AA$; and (D) $z = 0.0 \ \AA, A = 0.8 \ \AA$. The asymmetry in the molecular imaging in (D) (showing a “shadow” only on the left side of the molecules) is probably caused by asymmetric adsorption geometry of the CO molecule at the tip apex.
About AFM scale

... how to shake the Mont Blanc over little men heads without crushing them

Oscillation : 20 m
Cantilever 100 km
Mont Blanc : 4807 m
Man: 2 m

expanding to our scale : multiply all by $10^9$
Marking 0s and 1s
Streptavidin-biotin marker

Streptavidin: a "huge blob"

Biotin can easily be attached to DNA strand at order

Together they make one of the strongest non-covalent bond
Streptavidin-biotin marks

When added to the solution while imaging, Streptavidin attaches to biotin, marking the corresponding single stranded tiles.
Streptavidin-biotin marks

In this section we give the binding-domain schematics level of abstraction, which is below the abstraction level of aTAM proofreading tiles and above that of DNA sequence design.

S3.1 Strand-level system design of SST lattice, input-adapter strands, seed attachment

Concerns at this level of abstraction include choosing (a) appropriate domain lengths (in number of bases) for good DNA crossover positions between helices and for structural stability, for SSTs, input adapters, and origami seed staples/scaffolds, and (b) appropriate positions on SSTs for biotin modifications. As shown in Figure 1 in the main text, each square tile, post-proofreading, is mapped to an abstract SST strand. We used the SST motif from ref. [13], which has domains of length 10 and 11 bases. Altogether 355 SSTs were designed. The number 355 was calculated in Section S2.3.1, at the abstraction level of proofreading tiles; here at the SST binding-domain level of abstraction we simply convert each square (proofreading) tile to an abstract SST strand with four binding domains.

Figure S10 gives the domain-level design of our SST set, and the caption describes some of the key features. SST strands are designed to grow a nanotube lattice, so that the top of the SST lattice in Figure S10 is

```
wire 1
wire 2
wire 3
wire 4
wire 5
wire 6
```

= 0

= 1

Figure S10: SST lattice showing from left to right the DNA origami seed scaffolding (light blue), input-adapter strands (red, plus other colours), and SSTs (yellow, brown and blue), seam (grey), seam block (grey), and biotin locations (red and green disks/circles, coloured according to their wire index being even or odd). Each of the four domains on an SST are colour coded: yellow represents bit 1, brown represents bit 0, blue represents a domain that is unique to the proofreading block it is in. The presence of biotins is indicated by red and green solid disks, and is used to denote the encoding of a 1 bit on the proofreading block output domains that are on the same wire. A biotin is present on a strand if its closest bit-encoding output domain, along the same strand, is yellow (encodes bit 1), and otherwise the biotin is not present (denoted as a hollow circle, to help visualize all possible positions where biotins could be on other arbitrary computations). Relevant portions of the DNA origami seed scaffolding strand are shown as light blue on the left-hand side. Input-adapter strands are shown with red on the domains connecting to the scaffolding, and the domains binding to tiles are colored using the same bit convention as the tiles. Although input adapters do not have biotin modifications, red or green 'x's are used to indicate locations were biotins would have been had we chosen to include them.
kTAM model for algorithmic assembly
Erik Winfree had the idea that a growing lattice of DNA tiles could run a computer program, like Wang tiles or a CA.

tiles = program

crystal growth = program execution


Rothemund, Papadakis, Winfree 2004
Thermodynamical model

Attachement rate

\[ k_f \cdot [\text{Strand}] = k_f \cdot e^{-Gmc} \]

(mainly entropy)

Detachment rate

\[ k_f \cdot e^{-(b \cdot Gse)} \]

where \( b \) is the number of bonds

and \( Gse = \frac{\Delta G}{RT} \)

the bonding unit energy in RT units

(mix of entropy and enthalpy)

\( mc = \text{monomer concentration} \)

\( se = \text{sticky end bond strength} \)
Simulations

![Simulations Diagram]

- **low [monomer]**
- **G_{mc}**
- **high [monomer]**

- **weak bonds (hot)**
- **G_{se}**
- **strong bonds (cold)**

**\( \tau = 2 \)**

- optimal growth

**\( \tau = 1 \)**

- good crystals

- fast random aggregation

- constanrt \( \varepsilon \)

- no growth

*Winfree, Bekbolatov DNA9*
Simulations

- Good crystals
- Fast random aggregation
- Optimal growth
- Constant \( \varepsilon \)

\[
G_{se} < G_{mc} < 2G_{se}
\]

Fastest when
\[
G_{mc} \sim 2G_{se} - \varepsilon
\]

Sweet spot
Minimizing errors

Desired

Obtained
Proofreading tiles

- Cut every tile into $k \times k$ tiles
- Now, you need to make an other error to compensate for an error
- The error rate is squared for $k = 2$

Fig. 6. (a) The general proofreading construction for rule tiles. (b) The original Sierpinski tiles. (c) The $2 \times 2$ proofreading Sierpinski tiles. (d) Growth of the proofreading Sierpinski tiles. Small tiles illustrate that when a mismatched tile is incorporated, further growth on one side must involve a second mismatch. This is illustrated by the small tiles in figure 6d: after the initial (lowest) small tile arrives, forming a mismatch on one side, any further tile assembly on that side will either (a) agree with the initial tile but, because it therefore must be part of the same proofreading block, mismatch on its lower right side, or (b) agree with its lower right input, but therefore form a mismatch with the initial small tile. The assembly process stalls, giving time for the initial mismatch tile to fall off and be replaced by a correct tile. The final assembly therefore has no record of the mishap having occurred.

Winfree, Bekbolatov DNA9
Proofreading tiles

$k = 2$

$k = 3$
Proofreading tiles compared to other tiles

Figure S35 shows a non-algorithmic tile set that was designed to reuse tile types along the circumference of a 16-helix nanotube, along with AFM images showing the results.

The AFM images are not totally conclusive. However, it would appear that the tubes successfully grow from the origami seeds, but occasionally shrink in diameter as the tube grows, and in a few cases the tubes appear to split into two. We did not carefully test this hypothesis, but the images suggest that single-stranded tiles are possibly too floppy to prevent such lattice errors.

In addition to the approach used in this paper, wherein each row is hard-coded by sequence, general theoretical methods for ensuring correct self-algorithmic self-assembly in the limit of "arbitrarily floppy" tiles have been explored [91].

S5.7 Comparison of DX, TX, and SST motifs for algorithmic self-assembly

(a) DX motif (b) TX motif (c) SST (d) SST proofreading

Figure S36: Comparison of tile motifs used in algorithmic self-assembly. The top diagram in each panel shows the tile structure, with the binding domains used for tile-tile attachments colored green, red, blue, and orange, and the tile core colored cyan. For the four SST shown in a proofreading motif, the uniquely addressed binding domains specific to that proofreading block are shown in cyan. The bottom diagram in each panel displays a connectivity graph for the structure. The green, red, blue, and orange dots represent strands on a hypothetical neighboring tile. For DX and TX motifs, each (light or dark) cyan dot represents a strand of the motif. For the SST motif and the SST proofreading block, each black dot represents a single strand. Edges between dots indicate hybridization to form a double-helix, either within the tile or involving binding domains attaching a tile to a neighbor. Thin edges indicate the strength of a tile-tile interactions; thick edges indicate intra-tile binding that is twice as strong or more, in terms of the number of base pairs formed.

To date, three general DNA tile motifs have been used for algorithmic self-assembly. The DX tile motif was introduced in ref. [76], shown to form periodic two-dimensional (2D) lattices [12] and nanotubes [86, 92], used in finite uniquely-addressed arrays [16], and exploited for 2D algorithmic self-assembly [6, 93, 94, 7, 8, 9, 10, 11, 95]. The DAO variant of the DX tile motif is shown in Figure S36(a). The TX tile motif was introduced in ref. [96] and used to make 2D lattices in the same work; it has also been shown capable of forming nanotubes [97] and exploited for one-dimensional (1D) algorithmic self-assembly [5]. A variant of the TX tile motif is shown in Figure S36(b). The SST motif was introduced in ref. [13] and used to make nanotubes in the same work; it has been used in finite uniquely-addressed arrays [15] and in simple 2D algorithmic self-assembly [98]. The SST variant used in this work is shown in Figure S36(c). While periodic three-dimensional (3D) crystals have been designed using a DNA tensegrity triangle tile [99] and finite uniquely addressed 3D structures have been created using SST [80, 17], algorithmic self-assembly in 3D has not yet been demonstrated.

The self-assembly of DX and TX tiles share several notable features that are in contrast to SST. Both DX and TX tiles are designed to have a substantial "rigid" core that, during an anneal from a high temperature to a low temperature, assembles from the tile’s constituent strands (four of them, for the variants shown in Figure S36) before the tiles have significant interactions with each other to form lattices, arrays, or nanotubes. One could think of this tile set as logically equivalent (ignoring the repeating of tile types along the circumference) to the tile set of Section S8.9, but without the presence of any tiles representing 1, if one thinks of glues a, a′, b, b′ as both representing 0.
Implementing boolean circuits
Tile as gates

4 domains = 4 glues

Tiles assembly is a rewriting system
DNA nanotube circuit model

input

1

2 layers

3
A DNA strand implementing a gate
DNA nanotube circuit model

S3 System design abstraction level: binding-domain schematics

In this section we give the binding-domain schematics level of abstraction, which is below the abstraction level of aTAM proofreading tiles and above that of DNA sequence design.

S3.1 Strand-level system design of SST lattice, input-adapter strands, seed attachment

Concerns at this level of abstraction include choosing (a) appropriate domain lengths (in number of bases) for good DNA crossover positions between helices and for structural stability, for SSTs, input adapters, and origami seed staples/scaffolds, and (b) appropriate positions on SSTs for biotin modifications. As shown in Figure 1 in the main text, each square tile, post-proofreading, is mapped to an abstract SST strand. We used the SST motif from ref. [13], which has domains of length 10 and 11 bases. Altogether 355 SSTs were designed. The number 355 was calculated in Section S2.3.1, at the abstraction level of proofreading tiles; here at the SST binding-domain level of abstraction we simply convert each square (proofreading) tile to an abstract SST strand with four binding domains.

Figure S10 gives the domain-level design of our SST set, and the caption describes some of the key features. SST strands are designed to grow a nanotube lattice, so that the top of the SST lattice in Figure S10 is...
Example nanotube circuits

- **n-bit copying**: $n+1$ copy gates

  ![6-bit copying circuit](image)

<table>
<thead>
<tr>
<th>$i_1$</th>
<th>$i_2$</th>
<th>$o_1$</th>
<th>$o_2$</th>
</tr>
</thead>
<tbody>
<tr>
<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>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

  Note that 2 gates are single input, single output

- **n-bit binary sorting**: $n+1$ sort gates

  ![6-bit sorting circuit](image)

<table>
<thead>
<tr>
<th>$i_1$</th>
<th>$i_2$</th>
<th>$o_1$</th>
<th>$o_2$</th>
</tr>
</thead>
<tbody>
<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>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

  Note that 2 gates are single input, single output
**Example nanotube circuits**

- **Lazy sorting!** Take the union of the copy gate set and the sort gate set. Copying fights to slow down the sorting process, but assuming a fair execution, sorting will eventually win.

- Since, in any given circuit, each gate “knows” its row number $r$, we will also write circuits (programs) that exploit this feature, do something that is interesting *and* (more importantly) provably impossible without that feature.

---

6-bit slow randomised sorting

<table>
<thead>
<tr>
<th>Input 1</th>
<th>Input 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Output:

<table>
<thead>
<tr>
<th>Output 1</th>
<th>Output 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**Copy gate**

- $i_1$ flows into $i_1$
- $i_2$ flows into $i_2$

**Sort gate**

- $i_1$ flows into $i_1$
- $i_2$ flows into $i_2$

$\max(i_1, i_2)$

$\min(i_1, i_2)$
Circuits

Function computation

Sorting

Parity

Glider: A common cellular automata primitive

Zig-zag

Solving a “hard” decision problem

Pattern: DNA

Zig-zag

Behaviour: 63 layers to see the same thing twice!

Long repeat

Rule 110
Randomised programs may be a useful tool to calculate energetics of tile binding, or groups of tiles binding, from AFM data. A nice method to assess the quality of our sequence design.
Circuits

Glider: A common cellular automata primitive
Pattern: Monotone / horizontally connected

Nonmonotonic widely-spaced patterns are provably impossible in the deterministic circuit model

Diamonds are forever

Blowing bubbles
Computational power of DNA (DNA = DNA nanotube algorithms)

- What is the computational power of our circuit model?
- With $n$ input bits, depth-2 layer, and poly(n) depth circuit, what can be solved?
  - No more than P (proof: simulate poly(n) depth circuit in polynomial time on a Turing machine)
  - We’ve seen already that the model can solve SORTING, PARITY both of which are outside $AC^0$
• **Theorem:** Rule 110 is an **efficient** and **general purpose** computer


ICALP 2006
Computational power of DNA (DNA = DNA nanotube algorithms)

- What is the computational power of our circuit model?
- With $n$ input bits, depth-2 layer, and poly($n$) depth circuit, what can be solved?
  - No more than P. Proof: simulate poly($n$) depth circuit in polynomial time on a Turing machine
- All of P: Proof: simulate Rule 110

\[
\begin{align*}
F(0, 0, 0) &= 0 & F(1, 0, 0) &= 0 \\
F(0, 0, 1) &= 1 & F(1, 0, 1) &= 1 \\
F(0, 1, 0) &= 1 & F(1, 1, 0) &= 1 \\
F(0, 1, 1) &= 1 & F(1, 1, 1) &= 0
\end{align*}
\]
Computational power of DNA (DNA = DNA nanotube algorithms)

- What is the computational power of our circuit model?
- With $n$ input bits, depth-2 layer, and poly($n$) depth circuit, what can be solved?
  - Answer: Exactly P, via Rule 110 simulation

From gate abstraction to tile abstraction

1. Compile gates to tiles

Each glue encodes a bit and a layer row

Gate truth table

<table>
<thead>
<tr>
<th>$i_1$</th>
<th>$i_2$</th>
<th>$o_1$</th>
<th>$o_2$</th>
</tr>
</thead>
<tbody>
<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>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Each row of a gate’s truth table is encoded by a tile

Tiles (4 per gate)
6-bit universal tileset: overview

For each gate we have 4 tiles, 1 or which sticks

Glues encode rows
2. Wrap into a tube along boundary/seam ("_" = no bit here)

2.1. U_ does not encode input/output bits. U_ encodes "boundary"

2.2. U2,…,U6 have 2 input and 2 output bits. U1 & U7 have only 1 input and 1 output bit.

3. Asynchronous update semantics: assembly frontier grows asynchronously rather than layer-by-layer (does not change expressivity of circuit versus tile model, roughly speaking)
But can we afford all those tiles?
From gates to tiles: savings

- Let’s convert the set of $R$-bit universal gates into tiles, and examine at the resulting $R$-bit universal tile set.

- Suppose I have two different gates, e.g. copying and sorting. If I convert each into 4 tiles I get 8 tiles, but let's look closer at some tile-savings:

<table>
<thead>
<tr>
<th>$i_1$</th>
<th>$i_2$</th>
<th>$o_1$</th>
<th>$o_2$</th>
</tr>
</thead>
<tbody>
<tr>
<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>
</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>
</tbody>
</table>

$\text{copy gate}$

$\begin{align*}
\text{copy tiles:} & \\
& \begin{array}{cccc}
0 & 0 & 1 & 1 \\
0 & 0 & 1 & 1 \\
0 & 0 & 1 & 1 \\
0 & 0 & 1 & 1 \\
\end{array}
\end{align*}$

$\text{sort tiles:} \\
\begin{align*}
& \begin{array}{cccc}
0 & 0 & 1 & 1 \\
0 & 0 & 1 & 1 \\
0 & 0 & 1 & 1 \\
0 & 0 & 1 & 1 \\
\end{array}
\end{align*}$

$\text{sort gate:}$

$\begin{align*}
\text{sort gate:} & \\
& \begin{array}{cccc}
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
\end{array}
\end{align*}$

$\text{max}(i_1, i_2)$

$\text{min}(i_1, i_2)$
From gates to tiles: savings

- Let’s convert the set of $R$-bit universal gates into tiles, and examine at the resulting $R$-bit universal tile set.

- Suppose I have two different gates, e.g. copying and sorting. If I convert each into 4 tiles I get 8 tiles, but let’s look closer at some tile-savings:

Truth tables: 3 identical row-pairs!

3 identical tile-pairs!

$i_1, i_2 \in \{0,1\}$
From gates to tiles: savings

- Let’s convert the set of $R$-bit universal gates into tiles, and examine at the resulting $R$-bit universal tile set.

- Suppose I have two different gates, e.g. copying and sorting. If I convert each into 4 tiles I get 8 tiles, but let’s look closer at some tile-savings:

Truth tables: 3 identical row-pairs!

Only 5 tile types needed to do both copying and sorting!
6-bit universal tileset: overview

- Intuition from previous slide: Tiles separate the 4 “elementary operations” of a gate into 4 individual tiles, which results in **fewer tile types** in our universal tile set than gates in the universal gate set.

- So how many tiles in the $R$-bit universal tile set?

  E.g. U4: There are **16 U3 tile types** that can go here (a tile is defined by its row & 4 bits), as opposed to 256 gates in the circuit model.

The user may plug and play with these 16 tile types!

<table>
<thead>
<tr>
<th>Tile Type</th>
<th>Description</th>
<th>Number of Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>U1</td>
<td>4 input and 1 output bit</td>
<td>16</td>
</tr>
<tr>
<td>U2, U3, U4, U5, U6</td>
<td>2 input and 2 output bits, each type</td>
<td>16 each</td>
</tr>
<tr>
<td>U7</td>
<td>1 input and 1 output bit</td>
<td>4</td>
</tr>
<tr>
<td>U_</td>
<td>seam tile (represents no bits)</td>
<td>1</td>
</tr>
</tbody>
</table>

Total: **89 tile types**

1,288 for 2-output gates.
168 for 1 output gates.
6-bit universal tileset: details

8 rows U1–U8; each has disjoint subset of tile types

Glues are between even and odd rows: always named after even row

U4;00→00  U4;01→00  U4;10→00  U4;11→00

U4;00→01  U4;01→01  U4;10→01  U4;11→01

U4;00→10  U4;01→10  U4;10→10  U4;11→10

U4;00→11  U4;01→11  U4;10→11  U4;11→11

Pic by Dave Doty

Damien Woods
6-bit universal tileset: details

glues are between even and odd rows: always named after even row

8 rows U1–U8; each has disjoint subset of tile types

other possible outputs on input 00

other possible inputs
to compute a function, e.g.:

<table>
<thead>
<tr>
<th></th>
<th>i1</th>
<th>i2</th>
<th>o1</th>
<th>o2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

select one tile from each column

tiles in rows U3,U5,U6,U7 selected similarly

each computes a function f : \{0,1\}^2 \rightarrow \{0,1\}^2

pic by Dave Doty

Damien Woods
6-bit universal tileset: details

Special cases for rows near seam

- U2: \_0 → \_0
- U2: \_1 → \_0
- U2: \_0 → \_1
- U2: \_1 → \_1
- U8: 0 → 0
- U8: 1 → 0
- U8: 0 → 1
- U8: 1 → 1

U2 and U8 have no bit on the helix they share with U1, so they compute a function \( f: \{0,1\} \to \{0,1\} \).

Only 1 tile type on position U1, computes trivial function \( f: (\_,\_) \to (\_,\_) \).

8 rows U1–U8, each has disjoint subset of tile types.

Pic by Dave Doty
6-bit universal proofreading (PR) tileset

- Linear/polynomial redundancy for exponential error reduction
- 2x2 PR transformation: each tile type $t$ is transformed into a 2x2 block of 4 tiles types that uniquely represent, or hardcode for, $t$

- 3-bit copying experiments show that 2x2 PR significantly reduces errors
- Transforms 89 tiles into $356$ proofreading tiles
- Caveat: we will use only a single tile type along the seam (hence, the 2x2 “U_” block at the seam is not a proofreading block). => $4\times9-1=355$ unique strands
3-bit proofreading copying tileset

- To give an idea of what a 2x2 proof-reading transformation is here is a 3-bit proofreading copying applied to the 3-bit copying tile set (i.e. for a different tile set)

pic by Dave Doty
Sequence design
Random sequences will not work

Spread them apart!

Random sequences over 3-letter code with 1 base exception, and domain-pairs ending with AT stack
What do we want?

1. No "self-folding"
2. Clean lattice boundary
3. Minimize interactions between strand pairs
4. Uniform correct binding: in a tight range
5. Incorrect binding should have a much higher energy
An iterative process

Set of sequences

Corrections

Evaluate

NUPACK nucleic acid package

Analysis    Design    Utilities
Input

Nucleic acid type: RNA
Temperature: 37.0 °C
Number of strand species: 1

Strand species

strand1: ATGCATATGCATATGCATATGCATATGCATATGCAT
Designed sequences

Figure S18: Evidence that using random, or almost random, DNA sequences could lead to higher tile attachment error rates than designed sequences. (a) Random sequences for our 355 tiles, but with a 3-letter code (each strand using either the alphabet A,T,C or the alphabet A,T,G) and where runs of CCCC and GGGG were forbidden within a domain. The red histogram shows binding energies for a tile correctly binding to a valid lattice, by its two input domains (see Figure S19(a) for the definition of input domains). The blue histogram shows binding energies for a tile erroneously binding to a correct lattice where one of the tile’s input domains correctly matches and the other mismatches (called a tile attachment error). Binding energies were calculated using the function lattice_binding_spacer() described in Section S4.3.3. Since more negative energy implies more favourable binding, it can be seen in (a) that many of the erroneous tile attachments (blue) are stronger than many of the correct tile attachments (red). (b) The same analysis applied to our designed DNA sequence set of 355 tile-strands for comparison; all erroneous tile attachments (blue) are significantly weaker than all correct tile attachments (red). Plot (b) is explained in Section S4.3.3.

In the main text it is noted that the three main principles employed to inhibit growth errors are: (i) ensuring that desired interactions are isoenergetic, (ii) minimizing erroneous binding through minimizing mismatch binding energies, and (iii) employing proofreading tile sets in the logical design. Also, because SSTs are floppy, there are other undesired interactions, including (iv) minimizing off-lattice interactions (unintended binding of tiles to themselves or each other) and (v) minimizing near-lattice interactions (unintended interactions of strands at the lattice growth frontier).

A variety of different models are used for different criteria; one of the main reasons for this was to balance concerns about computational efficiency (some models are faster to evaluate than others) against concerns about how well the resulting strands would perform in self-assembly experiments (some models are perhaps more predictive than others). As part of the design process, designed sets of sequences were subsequently analysed using a suite of models and criteria that was broader than the design models and criteria, and in the end, the sequences that were chosen performed satisfactorily on that entire analysis suite (Section S4.3). All energy values and thresholds are in units of kcal/mol, and energetics calculations were performed at a temperature of 35.0°C. NUPACK pfunc energies were computed using the parameter sets dna1998.dH and dna1998.dG (that ship with NUPACK) and invoked via: pfunc -T 53 -multi -material dna. RNAduplex energies were computed from the parameter set dna_matthews1999.par (that ships with ViennaRNA), and invoked via: RNAduplex -P dna_matthews1999.par -T 53 –noGU.
The experiments
The seed: a DNA origami

<table>
<thead>
<tr>
<th>Classic rectangle</th>
<th>Twist correction</th>
<th>Barrel correction</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Classic rectangle" /></td>
<td><img src="image2.png" alt="Twist correction" /></td>
<td><img src="image3.png" alt="Barrel correction" /></td>
</tr>
</tbody>
</table>

**tiles:** idealized cross-section of 16-helix nanotube of single-stranded tiles with crossover between all adjacent helices: regular 16-gon

**seed:** idealized cross-section of 16-helix DNA origami barrel without crossover from top to bottom helix: irregular 16-gon
Seed barcodes allow to image many circuits/inputs at the same time.
Preparing the tiles

• Mix of the tile strands for each of the circuits in an individual properly labelled tube
Protocol

1. **Origami**
   1.1. Mix scaffold and staples and adapters
   1.2. Heat at 90°C and let it cold down to 58.1°C slowly (1h)

2. **Growth**
   2.1. Add tiles
   2.2. Let it grow at 58.1°C for 1 day

3. **Guards**
   3.1. Add Guard staples
   3.2. Let it attach for 4h

4. **Unzip**
   4.1. Add the unzipers
   4.2. Let it rest for 1 night

5. **Cool down to room temperature and Image!**
The result
### Influence of Temperature

<table>
<thead>
<tr>
<th>Temperature °C</th>
<th>S' biotin</th>
<th>Internal biotin</th>
<th>No biotin</th>
<th>No seed or biotin</th>
</tr>
</thead>
<tbody>
<tr>
<td>51.9</td>
<td><img src="image1.png" alt="Image" /></td>
<td><img src="image2.png" alt="Image" /></td>
<td><img src="image3.png" alt="Image" /></td>
<td><img src="image4.png" alt="Image" /></td>
</tr>
<tr>
<td>52.4</td>
<td><img src="image5.png" alt="Image" /></td>
<td><img src="image6.png" alt="Image" /></td>
<td><img src="image7.png" alt="Image" /></td>
<td><img src="image8.png" alt="Image" /></td>
</tr>
<tr>
<td>53.0</td>
<td><img src="image9.png" alt="Image" /></td>
<td><img src="image10.png" alt="Image" /></td>
<td><img src="image11.png" alt="Image" /></td>
<td><img src="image12.png" alt="Image" /></td>
</tr>
<tr>
<td>53.4</td>
<td><img src="image13.png" alt="Image" /></td>
<td><img src="image14.png" alt="Image" /></td>
<td><img src="image15.png" alt="Image" /></td>
<td><img src="image16.png" alt="Image" /></td>
</tr>
<tr>
<td>53.8</td>
<td><img src="image17.png" alt="Image" /></td>
<td><img src="image18.png" alt="Image" /></td>
<td><img src="image19.png" alt="Image" /></td>
<td><img src="image20.png" alt="Image" /></td>
</tr>
<tr>
<td>54.1</td>
<td><img src="image21.png" alt="Image" /></td>
<td><img src="image22.png" alt="Image" /></td>
<td><img src="image23.png" alt="Image" /></td>
<td><img src="image24.png" alt="Image" /></td>
</tr>
</tbody>
</table>
Rule 110: Turing complete!

Simulation of a cellular automaton

Tile-attachment error rate 0.03%±0.009
Number of tiles attached 48,789

Landscape of circuit decision problems

Rule 110 prediction

Parity

Complex

Simple
Lazy sorting

LaZYSORTING

Sort 1s to the top

Tile-attachment error rate 0.03%±0.005
Parity

Is the number of 1s odd?

Yes  No  Yes  No  Yes  No

Tile-attachment error rate 0.03%±0.001
Number of tiles attached 1,318,163

$2^6 = 64$ inputs

Number of tiles attached 354,355

Reprogramming IBCs.
Is the input binary number a multiple of 3?

- Yes
- Yes
- Yes
- No
- No

Tile-attachment error rate 0.03% ± 0.002
Number of tiles attached 354,355
Unbiasing a biased coin

Probability(result = yes)

Bias P and barcode

Distance to yes/no result (nm)

Tile-attachment error rate 0.01% ± 0.001
Number of tiles attached 545,785

Unbiasing a biased coin

P = 0.5

P = 0.9

P = 0.1

No

Yes

No

Yes

No

Yes

Tile-attachment error rate 0.03%
Number of tiles attached 48,789

Bias P and barcode

Distance to yes/no result (nm)

Probability(result = yes)

Bias P and barcode

Distance to yes/no result (nm)

Tile-attachment error rate 0.01% ± 0.001
Number of tiles attached 545,785
Conclusion

• A 6-bits universal "efficient" DNA computer based on CA rule 110

• 3-5 years of hard work

• Beautiful results

• OPEN: interface computation for other circuits? reduce errors? have the circuits react to something?