Module 5: Logic circuits with DNA strand displacement (part 1)

CSE590: Molecular programming and neural computation.
Goal: Engineering embedded controllers for biochemical systems

A cell-based “computer” needs to be biocompatible, and sense, analyze and act on biological information.

Biological Information is encoded in the sequences and amounts of biomolecules (DNA, RNA, proteins, etc.)

```
ugagguaguagguuuguauaguuu
ugagguaguagguuguugaguuu
ugagguaguagguugauagguuu
agagguaguaguugucauaguu
ugagguagggagguuguauagguuu
ugagguaguagguauuauagguuu
ugagguaguaguuuuguacaguuu
ugagguaguaguuuugugcuguuu
```
Q: Why digital logic? Biology is not digital.

A: Because adherence to digital logic design has enabled incredibly complex, manmade information technology. We don’t need to do exactly what biology does.

Q: Why DNA strand displacement?

A: Because it’s a surprisingly powerful building block.
Basic rules

Short domains bind **reversibly**

\[ \text{5'} - \text{TTAAGT} - \text{3'} \]

\[ \text{3'} - \text{AATTCA} - \text{5'} \]

Long domains bind **irreversibly**

\[ \text{5'} - \text{TTAAGTCTAGGTGGGTTTCT} - \text{3'} \]

\[ \text{3'} - \text{AATTCAGATCCACCCAAAGA} - \text{5'} \]
DNA strand displacement mechanism

RNA sequence:
5'-AAUUCAGAUCCACCCAAAGAG-3'

For a review see D.Y. Zhang and G. Seelig, Nature Chemistry (2011)
DNA strand displacement mechanism

For a review see D.Y. Zhang and G. Seelig, Nature Chemistry (2011)
Strand displacement is initiated at the single-stranded toeholds. Toehold binding is a reversible process.
DNA strand displacement mechanism

Strand displacement proceeds through a branch migration. Branch migration is a random walk.
DNA strand displacement mechanism

Release of the output strand is (almost) irreversible in the absence of a toehold for the reverse reaction.
Signals can propagate through multiple layers

The sequences of inputs and outputs can be completely independent.
Signals can propagate through multiple layers

The sequences of inputs and outputs can be completely independent.
Signals can propagate through multiple layers

The sequences of inputs and outputs can be completely independent.
Signals can propagate through multiple layers

The sequences of inputs and outputs can be completely independent.
Signals can propagate through multiple layers

The sequences of inputs and outputs can be completely independent.
Signals can propagate through multiple layers

The sequences of inputs and outputs can be completely independent.
Signals can propagate through multiple layers

The sequences of inputs and outputs can be completely independent.
Signals can propagate through multiple layers

The sequences of inputs and outputs can be completely independent.
Signals can propagate through multiple layers

The sequences of inputs and outputs can be completely independent.
OR logic / fan-in
OR logic / fan-in
OR logic / fan-in

Input 2

Gate

Gate

Gate

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

Friday, January 24, 14
AND logic

<table>
<thead>
<tr>
<th>In 2</th>
<th>In 1</th>
<th>Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
AND logic

In 1 | In 2 | Out
0   | 0   | 0

Gate 1 | Gate 2

Input 2

Friday, January 24, 14
AND logic

spurious non-productive interaction

<table>
<thead>
<tr>
<th>In 2</th>
<th>In 1</th>
<th>Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
AND logic

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

Input 1

Gate

Gate

In 1

In 2

Out

Out

Friday, January 24, 14
AND logic

In 1 | In 2 | Out
0  | 0  | 0
1  | 0  | 0
AND logic

\[
\begin{array}{c|c|c|c}
\text{In 2} & \text{In 1} & \text{Out} \\
\hline
0 & 0 & 0 \\
1 & 0 & 0 \\
\end{array}
\]
AND logic

In 1

In 2

Out

Out

<table>
<thead>
<tr>
<th>In 2</th>
<th>In 1</th>
<th>Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
AND logic

toehold releases spontaneously but reaction is reversible!

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

Friday, January 24, 14
AND logic

<table>
<thead>
<tr>
<th>In 2</th>
<th>In 1</th>
<th>Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
AND logic

Input 2

active/productive toehold

In 2 | In 1 | Out
--- | --- | ---
0   | 0   | 0
0   | 1   | 0
1   | 0   | 0
AND logic

<table>
<thead>
<tr>
<th>In 2</th>
<th>In 1</th>
<th>Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
AND logic

<table>
<thead>
<tr>
<th>In 2</th>
<th>In 1</th>
<th>Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
AND logic

In 1 | In 2 | Out
---|---|---
0  | 0  | 0
0  | 1  | 0
1  | 0  | 0
1  | 1  | 1

Friday, January 24, 14
AND logic

\[ \begin{array}{cccc|c}
\text{In 1} & \text{In 2} & \text{Out} \\
0 & 0 & 0 \\
0 & 1 & 0 \\
1 & 0 & 0 \\
1 & 1 & 1 \\
\end{array} \]
Why is NOT difficult?

Absence of a signal could be “NOT” or could simply mean that computation hasn’t occurred yet.
Dual-rail logic: AND and OR are sufficient for feed-forward digital circuits

Replace X by the pair (X0, X1):

- X0 on: logical “0”
- X1 on: logical “1”
- X0, X1 off: not yet computed
- X0, X1 on: error

Single wire circuit using NOT, AND, OR, NAND, ... can be replaced by a dual rail representation using AND and OR only. This implementation requires maximally 2x as many gates.
Differences and similarities between electronic and molecular circuits

1. **Lack of spatial isolation:** All gates and signals diffuse in solution and interact stochastically.

2. **Computation energy and non-reusable gates:** Both inputs and gates are consumed as the circuit is evaluated by cascade reactions, so they cannot be reused.

3. **Data encoding:** Information is encoded in the sequences and concentration of biomolecules.

4. **Lack of clear hardware software separation:** Gates and circuits come pre-programmed for the specific computation they are meant to carry out.

5. **Speed of computation:** A circuits evaluation under typical reaction conditions takes minutes to hours.

6. **Need for dual-rail logic:** NOT is difficult to implement.
visual DSD: A tool for simulating DNA strand displacement systems


Use links to “web simulator” and “tutorial” for hw.

visual DSD: A tool for simulating DNA strand displacement systems

\[
\begin{align*}
\text{tb} & \rightarrow (5') \text{TACCA} (3') \\
\text{tx} & \rightarrow (5') \text{TATTCC} (3') \\
\text{to} & \rightarrow (5') \text{GTCA} (3') \\
\text{b} & \rightarrow (5') \text{CCCTTTACATTACATAACAA} \\
\text{x} & \rightarrow (5') \text{CCCAAAACAAAAACAAACAAACAA} \\
\end{align*}
\]

\[
\{\text{tb}^*\}[\text{b} \text{ tx}^*];[\text{x} \text{ to}^*]
\]

Slide credit: Andrew Phillips (MSR)
visual DSD: A tool for simulating DNA strand displacement systems

Slide credit: Andrew Phillips (MSR)
visual DSD: A tool for simulating DNA strand displacement systems

Slide credit: Andrew Phillips (MSR)
visual DSD: A tool for simulating DNA strand displacement systems

**Strand::=**

\[<S> \quad \underline{s} \quad \text{Upper strand} \]

\[\{S\} \quad \underline{s} \quad \text{Lower strand} \]

**Segment::=**

\[\{L'\} <L>[S] <R>\{R'\} \quad \underline{s} \quad \text{Double stranded complex with overhangs} \]

**Segment concatenation**

\[
\begin{align*}
\text{L_1} & \quad \text{S_1} & \quad \text{R_1} \\
& \quad \text{S_1}^* & \quad \text{R_1}^*
\end{align*}
\]

\[
\begin{align*}
\vdots
\end{align*}
\]

\[
\begin{align*}
\text{L_2} & \quad \text{S_2} & \quad \text{R_2} \\
& \quad \text{S_2}^* & \quad \text{R_2}^*
\end{align*}
\]

\[=\]

\[
\begin{align*}
\text{L_1} & \quad \text{S_1} & \quad \text{R_1} & \quad \text{L_2} & \quad \text{S_2} & \quad \text{R_2} \\
& \quad \text{S_1}^* & \quad \text{R_1'} & \quad \text{L_2'} & \quad \text{S_2}^* & \quad \text{R_2'}
\end{align*}
\]

Slide credit: Andrew Phillips (MSR)
visual DSD: Syntax of strands and complexes

Slide credit: Andrew Phillips (MSR)
visual DSD: Reduction rules

Slide credit: Andrew Phillips (MSR)