SUPPLEMENTARY INFORMATION

doi:10.1038/nature10262

# Neural network computation with DNA strand displacement cascades

*Supplementary information*

**Lulu Qian[1], Erik Winfree[1,2,3] and Jehoshua Bruck[3,4]**

[1] *Bioengineering,* [2] *Computer Science,* [3] *Computation & Neural Systems, and* [4] *Electrical Engineering*
*California Institute of Technology, Pasadena, CA 91125, USA*

## Contents

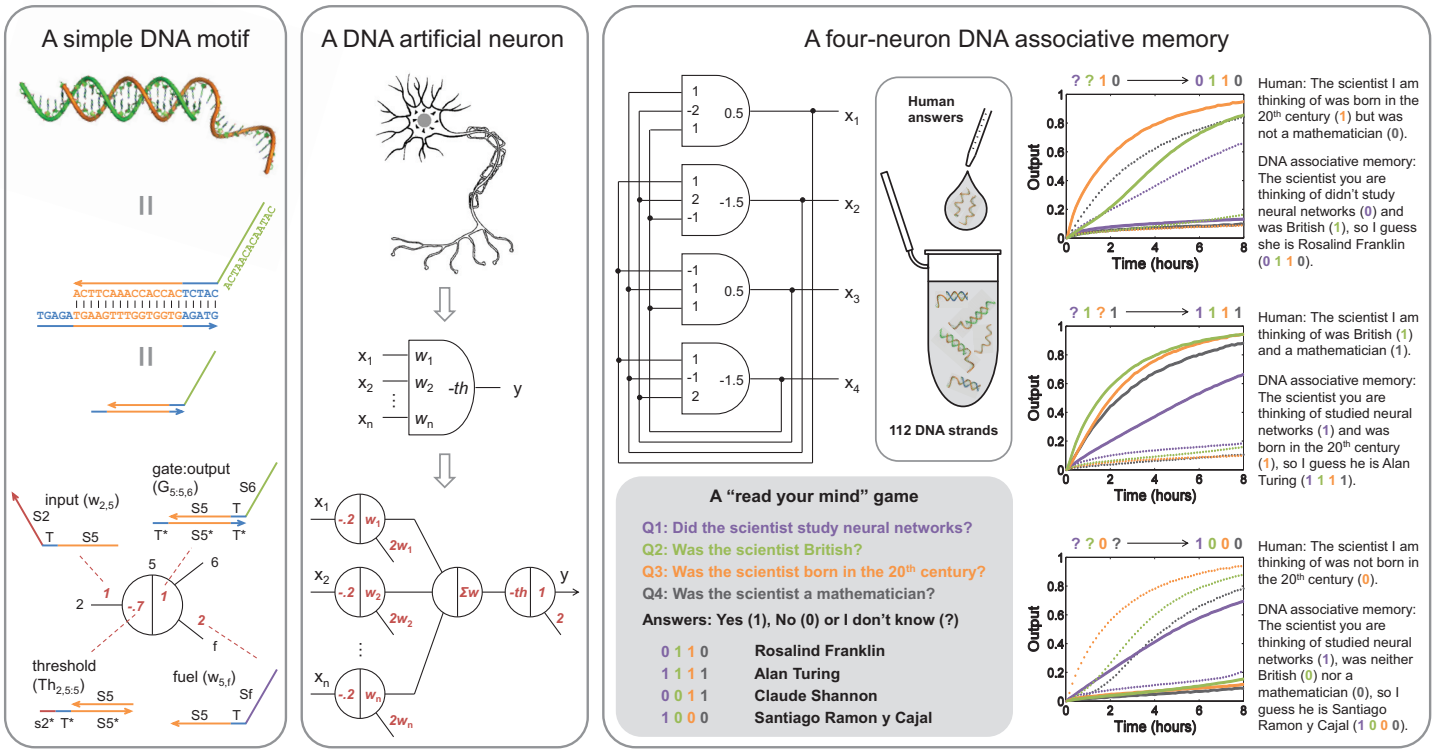



Figure S1: Summary of neural network computation with DNA strand displacement cascades. **Left panel**: a simple DNA motif (which we call the "seesaw" motif) presented as an abstract diagram with wires and nodes. The abstract diagram provides a concise representation of a full DNA implementation and can be systematically translated first to the domain level, then to the sequence level, and finally to the molecular level. The seesaw motif theoretically suffices for implementing a variety of analog and digital circuits[1] and has been demonstrated experimentally in multilayer AND-OR-NOT circuits.[2] **Middle panel**: an artificial neuron constructed with the simple DNA motif. Seesaw networks are reconfigured to capture the essential features of a neuron's behavior. As characterized in an artificial neural network model – the linear threshold gate – these features include analog weights, analog summation, tunable thresholding, and arbitrary degrees of fan-in and fan-out. An $n$-input linear threshold gate can be systematically implemented with a seesaw network with $n+2$ nodes. **Right panel**: a four-neuron Hopfield associative memory constructed with recurrent linear threshold circuits. It is trained to remember four patterns and to recall the most similar one upon the introduction of an incomplete pattern represented as a set of single-stranded DNA molecules. Training is done *in silico* to determine the values of weights and thresholds, which are then used to determine the concentrations of the 72 DNA species that comprise the memory. In principle, the memory could be trained to remember any of roughly 500 distinct sets of patterns. Unlike the traditional Hopfield associative memory with two states of each neuron: ON ("1") and OFF ("0"), we use a dual-rail monotone model that allows two additional states: unknown ("?") and invalid ("x"), which leads to new features such as identifying a pattern that is compatible with no single remembered pattern. We demonstrated the Hopfield memory in the context of a game called "read your mind", which is played between a human and our four-neuron DNA associative memory in a cuvette. The human thinks of a scientist and "tells" the DNA associative memory some of the answers to questions Q1 to Q4 by adding corresponding DNA strands into the cuvette. The DNA associative memory will "guess" who is in the human's mind and "tell" the human the rest of the answers by fluorescence signals. The experimental data can be fancifully interpreted as specific "conversations". The simplicity of the seesaw architecture – all components are no more complex than a double-stranded DNA with single-stranded tails, and all reactions are no more involved than a back-and-forth movement – provides a consistency that may be essential for scaling up circuit complexity. This is confirmed dramatically by modeling and simulations that semiquantitatively reproduced all the experimental data. Previous attempts to implement neural networks with DNA mostly used hybridization-based DNA circuitry, where DNA is used as "static Legos" that require manual experimental steps to be assembled, disassembled, and otherwise processed. In contrast, our approach exploits strand-displacement-based DNA circuitry, where DNA is used as "dynamic Legos" performing autonomous, programmable tasks within a single reaction vessel. By providing a general construction for implementing arbitrary circuits of use-once linear threshold gates, and by demonstrating several feedforward and recurrent neural network circuits, we have shown that an extremely simple DNA strand displacement mechanism is sufficient to imbue a chemical system with the ability to recognize patterns and make intelligent decisions. The picture of a neuron is from http://faculty.washington.edu/chudler/colorb1.html

## Methods

**Sequence design.** All DNA strands in seesaw circuits consist of long recognition domains and short toehold domains, and these domains are functionally independent. Thus, sequence design was done at the domain level. Software was written to generate distinct domain pools of the requested length. These domain sequences use a three letter code (A, C and T) to reduce secondary structures and interactions,[3–5] no more than 4 A's or T's in a row and no more than 3 C's in a row to reduce synthesis errors, and 30% to 70% C-content to ensure comparable melting temperatures.[6] For any two sequences in a pool, we require at least 30% of bases to be different, and the longest run of matches can be at most 35% of the domain length. These criteria were proposed[1] to reduce undesired displacement of mismatched invading strands, and they were used successfully to design experimental systems involving over 130 strands.[2] As that work discovered a gate-gate leak in strand displacement circuitry, "clamps" were included in the sequence design (see Table S1) to further reduce undesired strand displacement. To build circuits that were demonstrated in this work, a pool of 15 nt recognition domains was generated, with a specified 2 nt clamp sequence on both 5' and 3' ends. Each seesaw gate $i$ was assigned one recognition domain $Si$ from the pool. A 3 nt universal toehold core is chosen. With clamps, the 5 nt universal toehold $T$ becomes a variable element, containing the toehold core and the clamp on one side or the other. All DNA strands were then generated by concatenating $Si$ and $T$ domains, with a different interpretation of $T$, depending on whether it is on the left of, right of, or in between recognition domains. Further details and discussion of clamps were published previously.[2]

**Circuit preparation.** After being designed, DNA oligonucleotides were purchased from Integrated DNA Technologies (IDT). All gate, threshold and fuel strands were ordered unpurified (standard desalting); all input strands were ordered PAGE purified; and all reporter strands with fluorophore and quencher modifications were ordered HPLC purified. After being chemically synthesized and shipped as powder, DNA oligonucleotides were suspended in Milli-Q water (Millipore) and stored at 4°C at ∼100 $\mu$M. For more accurate concentrations, input strands and reporter strands were first quantified with BioPhotometer (Eppendorf). Reporter complexes were then annealed together at 20 $\mu$M, with a 20% excess of top strands. Because reporter top strands have no toehold domains and are modified with quenchers, this excess will ensure high yield formation of complexes even with imperfect stoichiometry, without producing residual active single-stranded DNA or changing the fluorescence baseline. Annealing was performed in a PCR machine (Thermal Cycler, Eppendorf), first heating up to 95°C, and then slowly cooling down to 20°C at the rate of 1°C/min. Gate and threshold complexes need purification in addition to annealing, because either top strands or bottom strands in excess will cause undesired activities in a circuit. Polyacrylamide gel electrophoresis (PAGE) purification was employed to remove single-stranded DNA and poorly formed double-stranded complexes. Finally, the purified gate and threshold complexes were quantified and extinction coefficients were calculated use $e = e(\text{top strand}) + e(\text{bottom strand}) - 3200N_{AT} - 2000N_{GC}$, where $N_{AT}$ and $N_{GC}$ are the number of AT pairs and GC pairs in the double-stranded domain, respectively.[7]

**Kinetics experiments.** Kinetics experiments were performed with two spectrofluorimeters (both SPEX Fluorolog-3, Horiba), which allow a maximum of four experiments to be run in parallel. Matched sets of 1.5 mL cuvettes were purchased from Hellma, either 119.004F-QS(round top) or 109.004F-QS(square top). Each set of four cuvettes was tested in our hands to show very little difference in fluorescence readout among cuvettes. The instrument can record multiple excitation / emission pairs in one experiment. For circuits using a reporter with the ROX fluorophore, excitation / emission at 588 nm / 608 nm was included. For circuits using a reporter with the FAM fluorophore, excitation / emission at 495 nm / 520 nm was included. For circuits using a reporter with the TYE563 fluorophore, excitation / emission at 549 nm / 563 nm was included. For circuits using a reporter with the TYE665 fluorophore, excitation / emission at 647 nm/ 664 nm was included. Bandwidth was 2 nm and integration time was 10 s for all experiments. Experiments for general linear threshold gates were done at 20°C while all other experiments were done at 25°C, because higher temperature decreases the spurious reactions in larger circuits.[2] Cuvettes were cleaned with distilled water and 70% ethanol between experiments.

**Reporter Crosstalk test.** Before four different reporters were used in the same fluorescence kinetics experiment, crosstalk tests were done to verify the lack of significant interference among the four distinct fluorophores (Fig. S2). Very little crosstalk was observed in an hour (Fig. S2c, left plots in samples 1 through 4) – each channel only responded to the signal strand that triggers the reporter with the corresponding fluorophore. More crosstalk was observed in 10 hours (Fig. S2c, right plots in samples 1 through 4) – in the worst case, there was roughly 5% crosstalk between the channels of FAM and TYE563, which is still neglectable.

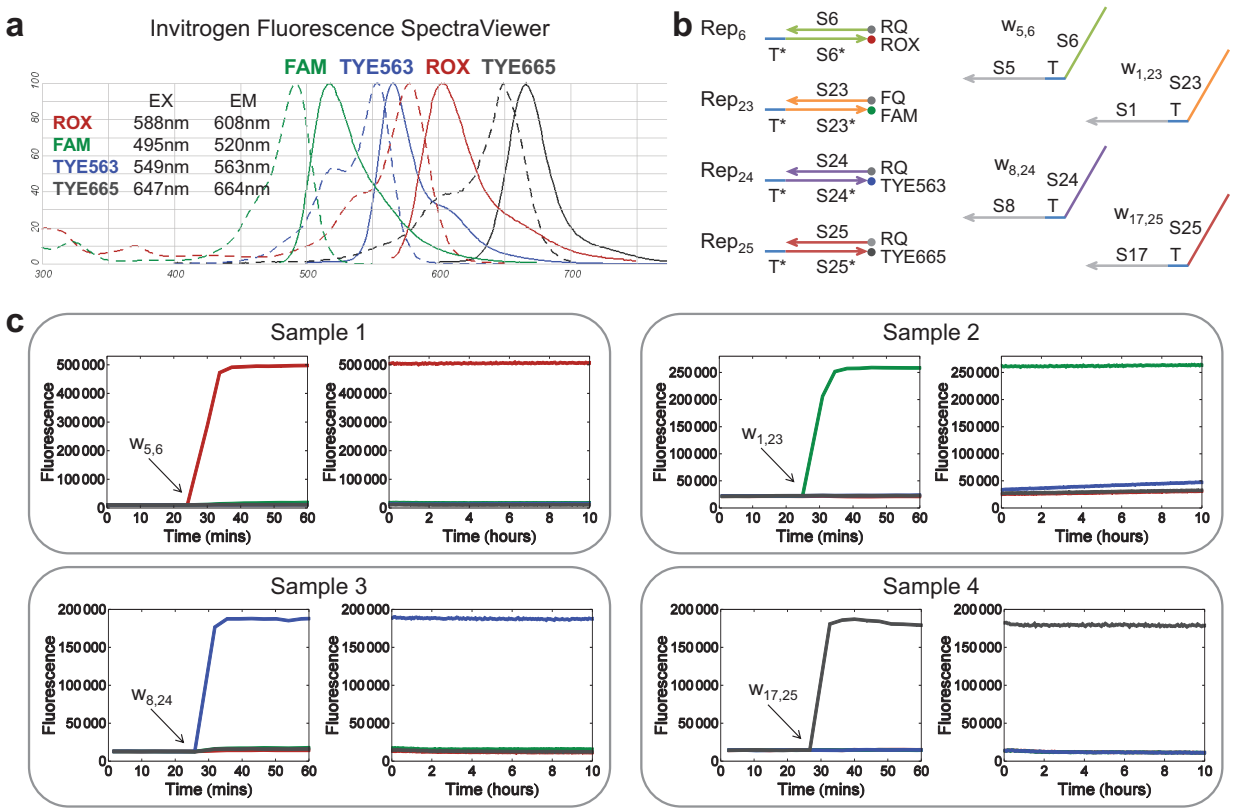



Figure S2: Crosstalk test among four reporters with distinct fluorophores. **(a)** Fluorescence spectra of ROX, FAM, TYE563 and TYE665, generated by Invitrogen Fluorescence SpectraViewer. Dotted curves show the excitation spectra and solid curves show the emission spectra. Four sets of excitation and emission wavelengths were determined to minimize the interference among different fluorophores. **(b)** Four reporters with fluorophores ROX, FAM, TYE563 and TYE665, and four signal strands that can trigger the four reporters, respectively. **(c)** Four parallel kinetics experiments to test the crosstalk among the four reporters. Four channels were set with the wavelengths in subfigure a. All four reporters were added to each of the four cuvettes at the beginning of the experiment (t=0, left plots in samples 1 through 4). Four signal strands were then added to the four cuvettes separately (roughly t=25 mins, left plots in samples 1 through 4). The right plots in samples 1 through 4 show data collected after the first hour. The four trajectories in each plot and the four fluorophores are shown with matching colors.

**Fluorescence data normalization.** All raw data of fluorescence signals were normalized to relative concentrations of output signals when plotted, which allows us to quantitatively analyze the data despite the difference of fluorophores, instruments, instrument performance, circuit functions and even molecular implementations. As mentioned above, our spectrofluorimeters can run four parallel kinetics experiments and the difference in fluorescence readout caused by the instrument is neglectable among these parallel experiments. Each set of parallel experiments we performed was of the same circuit with different inputs. The minimum level (output=0) was determined by the minimum of all four data points at t=0. For a given fluorophore, if the set of parallel experiments had at least one output signal that went high and flattened out at the end of the experiment (i.e. a maximally ON completion level), the maximum level (output=1) was determined by the average of the last 5 data points for the highest signal (Fig. S3a). Fluorescence data shown in Fig. 1f were normalized in this way. In the sets of experiments where none of the four output signals went all the way to completion at the end of the experiments, we performed a post-experiment triggering step in which we added excess of the inputs to the gates that directly generate the circuit outputs (i.e., the gates immediately upstream of the reporters), and then stopped data collection after the signals flattened out. In this case, the maximum level was determined by the average of the last 5 data points after the output signals were triggered to completion (Fig. S3b). Fluorescence data shown in Figs. 2c and 3e were normalized in this way. Note that if two sets of parallel experiments are of the same circuit with different inputs, run on the same instrument, and the interval between them is less than a day, the maximum level of one set can be used to normalize another. For example, the top two plots in Fig. S3a are raw data from two sets of parallel experiments that satisfy the above description, and the difference between two maximum completion levels are within 5%. Also note that the raw data taken by the spectrofluorimeter with a new lamp (Fig. S3b, ROX channel) can be as much as five times higher than with an old lamp (Fig. S3a), and with much less noise.

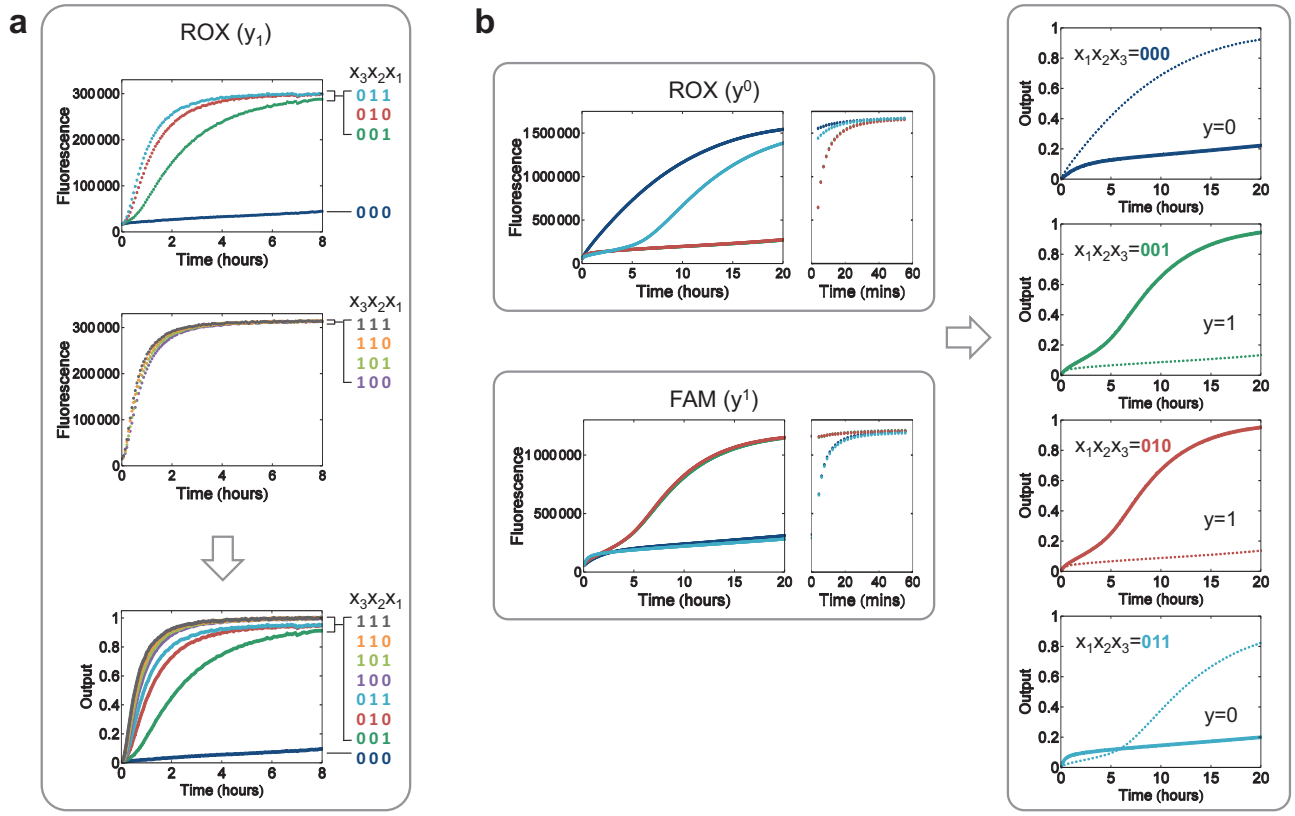



Figure S3: Fluorescence data normalization. **(a)** Raw data of ROX channel of the general linear threshold gate with inputs 000 to 111 and their normalization. The normalized data is the same data that is presented in the top plot in Fig. 1f. **(b)** Raw data of the 3-bit XOR circuit with inputs 000 to 011 and their normalization. The plot immediately to the right of the ROX (resp. FAM) raw data is the raw data from post-experiment triggering. The normalized data are the same data that are presented in the top four plots in Fig. 2c.

# S1. The seesaw DNA gate motif

## From abstract diagram to DNA implementation

In the seesaw abstraction, each DNA gate is represented by a node that has two sides (Fig. 1a). Each DNA signal is represented by a wire. Each side of the node can be connected to any number of wires. Each wire connects two different sides of two nodes. A reporter that can transform a DNA signal into a fluorescence signal is represented by half a node with a zig-zag arrow (Fig. 1b).

Each DNA signal is implemented as a single-stranded DNA molecule that has two recognition domains, identifying the two gates it connects, one on either side of a central toehold domain. All DNA signals are either initially free (e.g. input and fuel) or bound to a gate (e.g. gate:output). Each gate corresponds to a gate base strand (e.g. bottom strand of the gate:output complex) that has (the complement of) one recognition domain in the middle, identifying this gate, and (the complement of) two toehold domains on either side. At any given moment, a gate base strand always has a signal strand bound to one side or another, leaving the toehold on the other side uncovered. Each wire may have a threshold that absorbs the corresponding DNA signal when it arrives at a gate. The threshold is implemented as a double-stranded recognition domain, identifying this gate, with an extended toehold on one side of the bottom strand. Each reporter is implemented as a threshold-like DNA complex with a fluorophore/quencher pair attached to the top and bottom strand respectively.

Each red number in the abstract diagram indicates one DNA species with its initial relative concentration. Each red number on a wire from the right side of gate $i$ to the left side of gate $j$ indicates the initial relative concentration of $w_{i,j}$ (e.g. $1\times$ $w_{2,5}$ and $2\times$ $w_{5,f}$ in Fig. 1a). It is implemented as a free signal strand with two recognition domains $Si$ and $Sj$ connected by a toehold domain $T$. Each positive red number within a node $i$ at the end of a wire $w_{i,j}$ indicates the initial relative concentration of $G_{i:i,j}$ (e.g. $1\times$ $G_{5:5,6}$ in Fig. 1a). It is implemented as a complex of the gate $i$ base strand, with one recognition domain $Si^*$ flanked by two toehold domains $T^*$, bound to the left side of signal strand $w_{i,j}$. Each negative red number within a node $i$ at the end of a wire $w_{k,i}$ indicates the initial relative concentration of $Th_{k,i:i}$ (e.g. $0.7\times$ $Th_{2,5:5}$ in Fig. 1a). It is implemented as a complex with double-stranded domain $Si$ and a extended toehold domain $sk^*T^*$ on the left side of the bottom strand. Each negative red number within a half node $i$ indicates the initial relative concentration of $Rep_i$ (e.g. $1.5\times$ $Rep_6$ in Fig. 1b). It is implemented as a complex with double-stranded domain $Si$, a toehold domain $T^*$ on the left side of the bottom strand, a quencher (e.g. Iowa Black RQ) attached to the 5' end of the top strand, and a fluorophore (e.g. ROX) attached to the 3' end of the bottom strand. For more details, see our prior work.[1,2]

## Three basic reactions

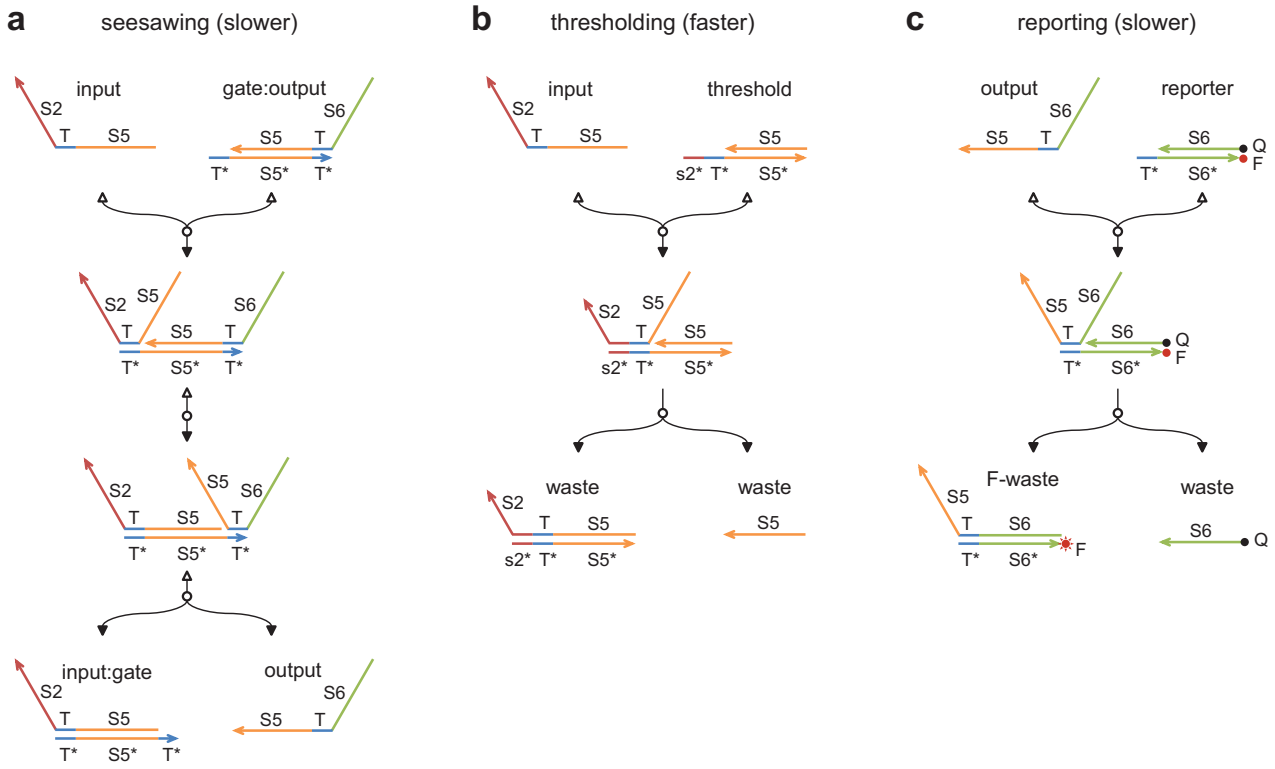



Figure S4: Three basic reactions of the seesaw DNA gate motif. Solid and outlined arrowheads distinguish reactants and products for each reversible reaction. **(a)** Seesawing. A free signal strand on one side of a gate (e.g. input signal $w_{2,5}$) hybridizes to a bound signal on the other side (e.g. gate:output complex $G_{5:5,6}$) via the uncovered toehold domain $T^*$, branch migrates through the recognition domain (e.g. $S5$), then the previously bound signal (e.g. output signal $w_{5,6}$) will fall off when it is only attached to the gate base strand by the short toehold. The now-bound signal (e.g. input:gate complex $G_{2,5:5}$) will have an uncovered toehold on the other side, and therefore the now-free signal (e.g. output signal $w_{5,6}$) can reverse the process symmetrically. **(b)** Thresholding. A free signal strand (e.g. input signal $w_{2,5}$) binds to an associated threshold complex (e.g. $Th_{2,5:5}$) via a extended toehold (e.g. $s2^*T^*$) and produces only inert waste molecules that have no toehold revealed. This reaction is much faster than seesawing because the strand displacement rate grows exponentially with toehold length for short toeholds.[8,9] **(c)** Reporting. A free signal strand (e.g. output signal $w_{5,6}$) binds to an associated reporter complex (e.g. $Rep_6$), releases the top strand, separates the quencher and the fluorophore, and results in an increasing fluorescence signal.

## The seesaw catalyst

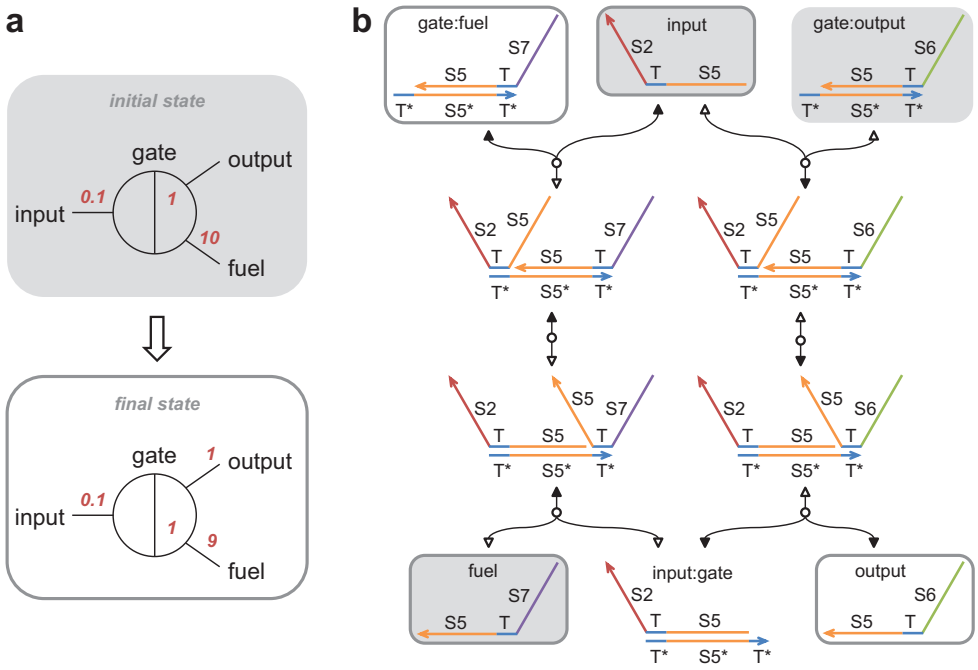



Figure S5: The seesaw catalyst. **(a)** Abstract diagram of the seesaw gate motif with an example of an initial state and its final state. Red numbers indicate relative concentrations of initial species (0.1× input, 1× gate:output complex and 10× fuel) and final species (0.1× input, 1× gate:fuel complex, 1× output and 9× fuel). Numbers in the final state are approximate. **(b)** Reaction pathways of a seesaw catalyst. Shaded boxes highlight the initial species; outlined boxes highlight the final species; shaded boxes with outlines indicate that the species is present in both the initial state and the final state. Solid arrows indicate flows of the forward reactions and outlined arrows indicate flows of the backward reactions. In this catalytic cycle, input initially reacts with the gate:output complex, releasing the output and in turn producing the input:gate complex (righthand pathway). Because there is much more fuel than output, the input:gate complex will prefer to react with the fuel, generating the gate:fuel complex while freeing the input (lefthand pathway). At this point, the input has a choice to either react with the gate:output complex or the gate:fuel complex, but it will always get bounced back by the fuel (because there is always a lot more fuel than output), resulting in accumulation of the gate:fuel complex and consumption of the gate:output complex until the circuit reaches its equilibrium. At the final state (equilibrium), the free/bound ratio for fuel cannot be less than 9, and this same ratio must apply to both input and output. This suggests that almost all input and output will stay free. Accurate calculation of equilibria for arbitrary seesaw circuits was discussed previously.[1]

# S2. Four types of seesaw gates

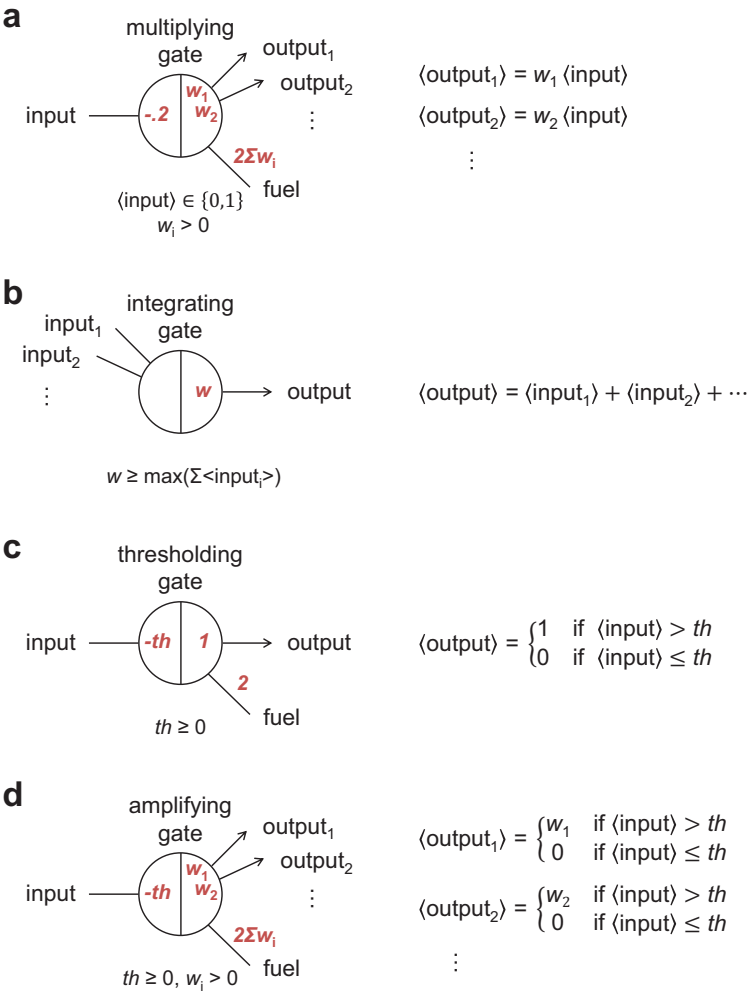



Figure S6: Four types of seesaw gates for linear threshold circuits. Each arrow indicates an irreversible downstream drain. ⟨signal⟩ is the total amount of the signal strand ever produced (signal = input, output, etc.). **(a)** Multiplying gates that produce analog weights. To sufficiently drive the release of all outputs, the initial amount of free fuel is twice the sum of all initially bound outputs. **(b)** Integrating gates that produce the sum of all inputs. To make sure all free inputs can be transformed into free output, the initial amount of bound output is at least the maximum sum of all inputs that can possibly arrive. **(c)** Thresholding gates that turn on the output when the input exceeds a threshold. **(d)** Multiplying gates and integrating gates are combined and generalized to amplifying gates that reduce the circuit size in cascades.

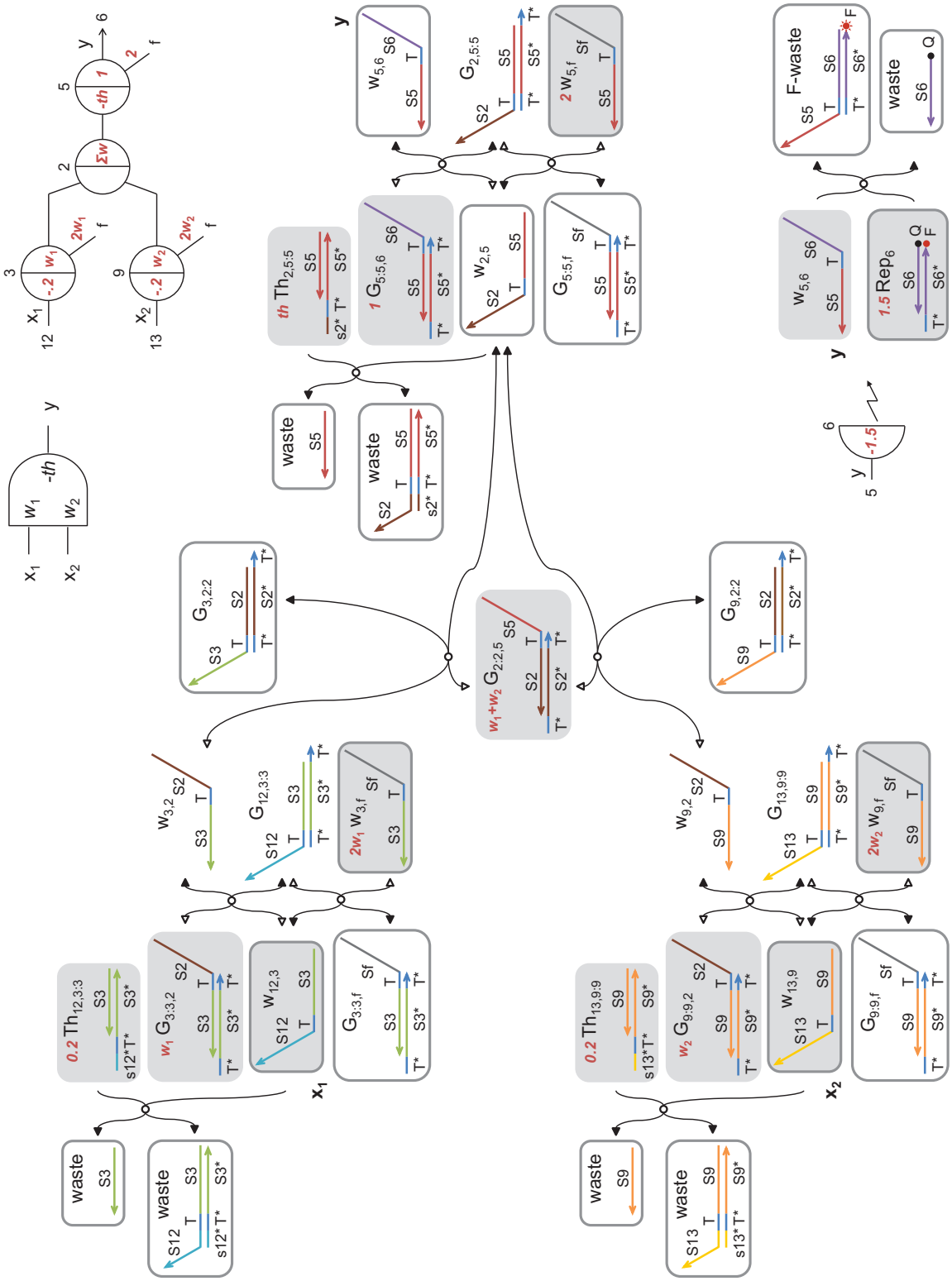



Figure S7: DNA implementation of a 2-input linear threshold gate and its reaction pathways. Shaded boxes highlight the initial species; outlined boxes highlight the final species; shaded boxes with outlines indicate that the species is present in both the initial state and the final state. Solid arrows indicate flows of the forward reactions and outlined arrows indicate flows of the backward reactions.

# S3. Four transformation rules

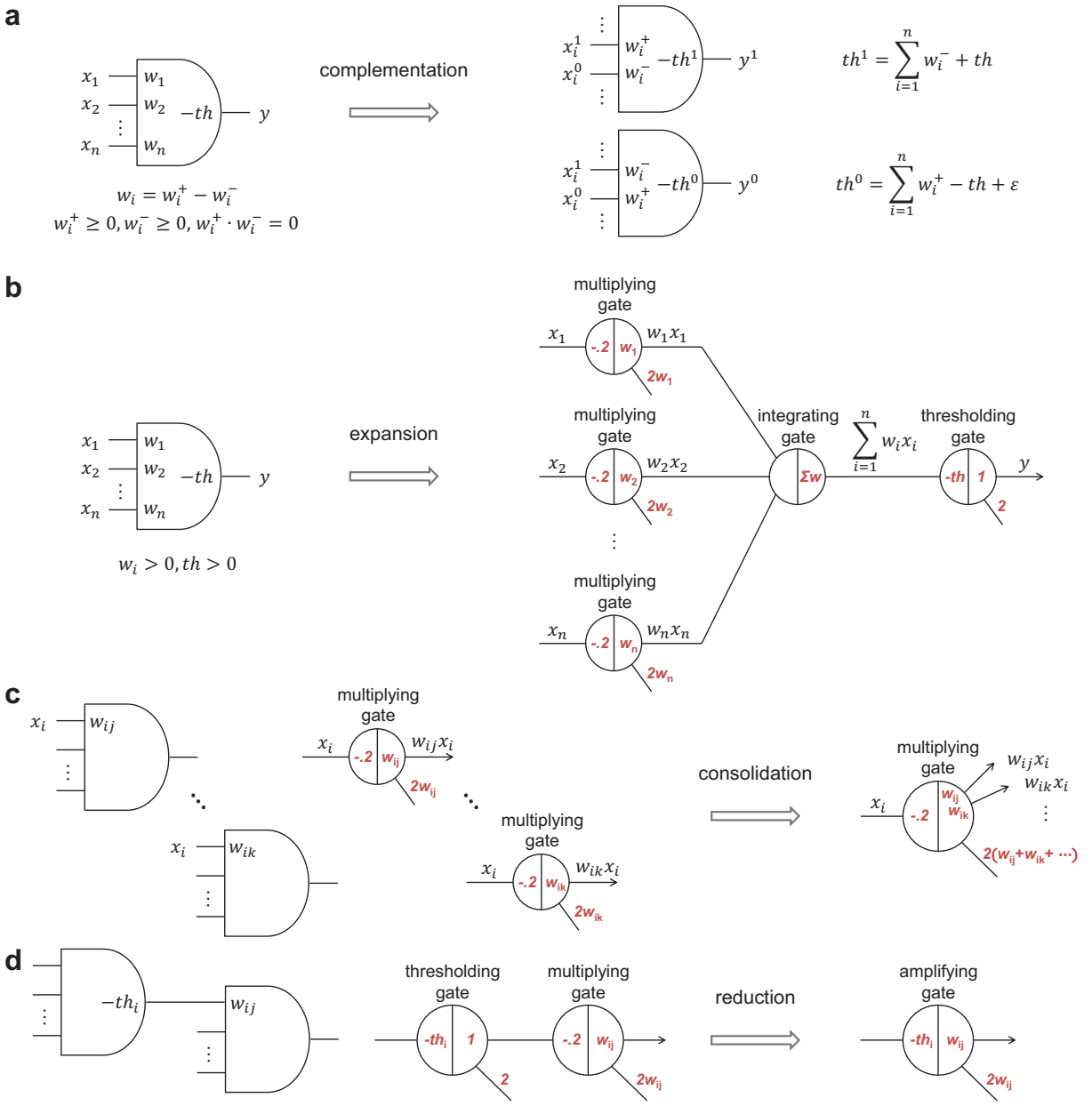



Figure S 8: Four transformation rules for arbitrary linear threshold circuits to seesaw circuits. **(a)** Complementation removes all negative weights. If $w_i$ is positive, it is represented as $w_i^+$ while $w_i^- = 0$; if $w_i$ is negative, it is represented as $-w_i^-$ while $w_i^+ = 0$. Each input $x_i$ is replaced by a pair of inputs $x_i^0$ and $x_i^1$, representing logic OFF and logic ON separately. If $x_i = 0$, $x_i^0 = 1$ and $x_i^1 = 0$; if $x_i = 1$, $x_i^0 = 0$ and $x_i^1$=1. Each linear threshold gate is replaced by a pair of gates with only positive weights, producing a pair of outputs as well. In each linear threshold gate after complementation, although both wires with inputs $x_i^0$ and $x_i^1$ are shown, only one wire will actually exist because one of $w_i^+$ and $w_i^-$ will be 0. Proof for the dual-rail linear threshold gate that outputs $y^1$: $y = 1 \Leftrightarrow \sum w_i x_i \geq th \Leftrightarrow \sum (w_i^+ - w_i^-) x_i \geq th \Leftrightarrow \sum w_i^+ x_i + \sum w_i^- (1 - x_i) \geq \sum w_i^- + th \Leftrightarrow \sum w_i^+ x_i^1 + \sum w_i^- x_i^0 \geq th^1 \Leftrightarrow y^1 = 1$. Similarly, $y = 0 \Leftrightarrow \sum w_i x_i < th \Leftrightarrow -\sum w_i x_i > -th \Leftrightarrow \sum w_i^- x_i^1 + \sum w_i^+ x_i^0 \geq \sum w_i^+ - th + \epsilon \Leftrightarrow y^0 = 1$ ($\epsilon$ is added to address the situation when $\sum w_i^- x_i^1 + \sum w_i^+ x_i^0 = \sum w_i^+ - th$; it is a real number that can be arbitrarily small, and must be smaller than the minimum of positive $\sum w_i^- x_i^1 + \sum w_i^+ x_i^0 - \sum w_i^+ + th$). **(b)** Expansion transforms each linear threshold gate with positive weights and threshold into a seesaw network. (If a non-positive threshold is derived from complementation, the linear threshold gate can be replaced by a wire that is always ON.) For each input $x_i$, a multiplying gate outputs $w_i x_i$. An integrating gate then outputs $\sum w_i x_i$. A thresholding gate finally outputs logic ON if $\sum w_i x_i \geq th$, and otherwise outputs logic OFF. **(c)** Consolidation collects all multiplying gates associated with the same input or internal output into a single multiplying gate with fan-out. **(d)** Reduction applies in between cascading linear threshold gates: each upstream thresholding gate and its directly downstream multiplying gate are combined to create an amplifying gate with the upstream threshold and the downstream weight.

## S4. Circuit design lessons learned from experiments

**Weights and thresholds are tunable.** With our implementation, all weights and thresholds in a linear threshold circuit are tunable – with the same set of DNA molecules comprising the circuit, new weight and threshold values can be obtained by adjusting the concentrations of certain initial species. For example, in Figs. S9 and S10, two general linear threshold gates with different thresholds were implemented with the same set of DNA molecules but different initial concentrations of two threshold complexes $Th_{2,5:5}$ and $Th_{4,1:1}$.

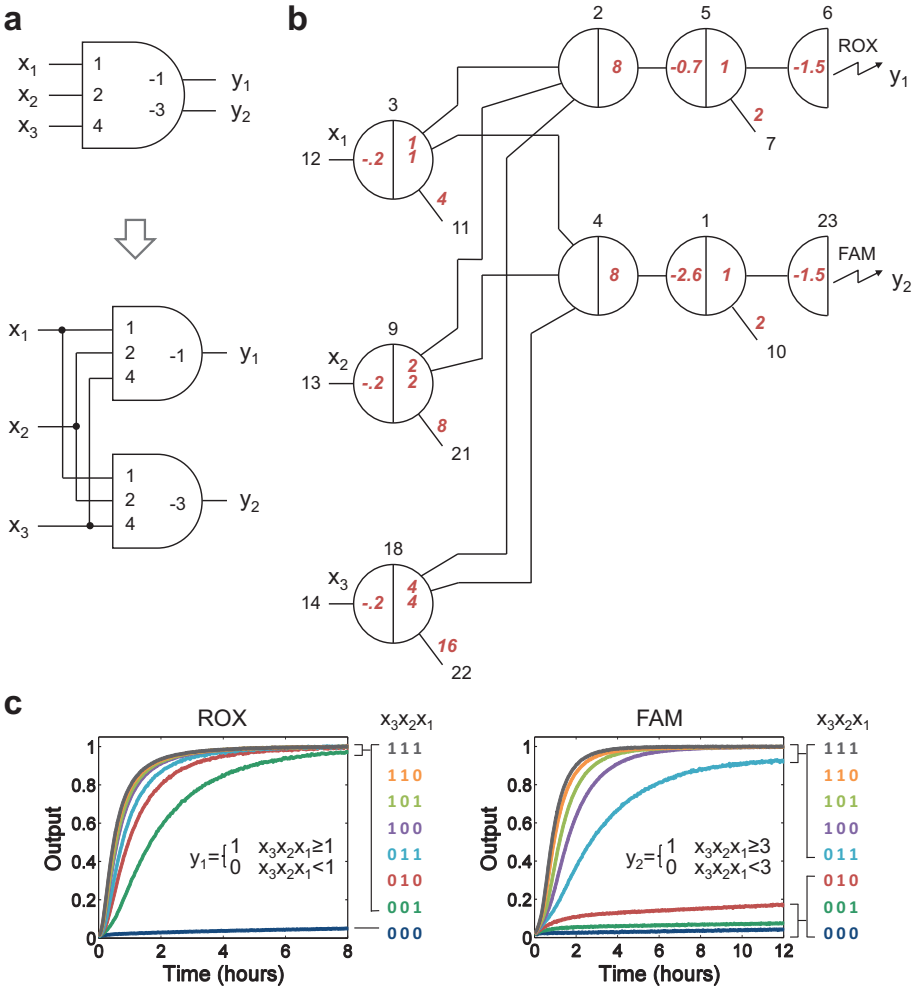



Figure S9: Demonstration of a 3-input 2-output linear threshold gate with thresholds 1 and 3. **(a)** A 3-input 2-output linear threshold gate that is equivalent to a linear threshold circuit with two gates in parallel and three inputs with fan-out of two. It calculates the analog value of a 3-bit binary number ($1 \cdot x_1 + 2 \cdot x_2 + 4 \cdot x_3 = x_3x_2x_1$), then compares the value to 1 and 3. **(b)** Equivalent seesaw circuit. ROX and FAM fluorophores are associated with two reporters that translate output signals $y_1$ and $y_2$ into fluorescence signals, respectively. **(c)** Kinetics experiments. A total of 36 DNA strands assembled to form 22 initial DNA species (indicated by the red numbers in subfigure **b**) were mixed in solution at their respective concentrations. The standard concentration was $1\times = 16.67$ nM. Input strands $x_1$, $x_2$ and $x_3$ were then added with relative concentrations of $0.1\times$ (0, logic OFF) or $0.9\times$ (1, logic ON). Trajectories for corresponding inputs have matching colors. Sequences of strands are listed in Tables S2 to S4, circuit 1. Experiments were performed at 20°C in Tris-acetate-EDTA buffer containing 12.5 mM $Mg^{2+}$. Output signals were inferred by fluorescence signals normalized to the maximum completion level. Both outputs achieved the correct ON or OFF states with the complete 8 sets of inputs, even though the inputs were not at ideal $0\times$ and $1\times$ values. For output $y_1$, with threshold 1, the ON/OFF gap was between 000 and 001; for output $y_2$, with threshold 3, the ON/OFF gap was between 010 and 011.

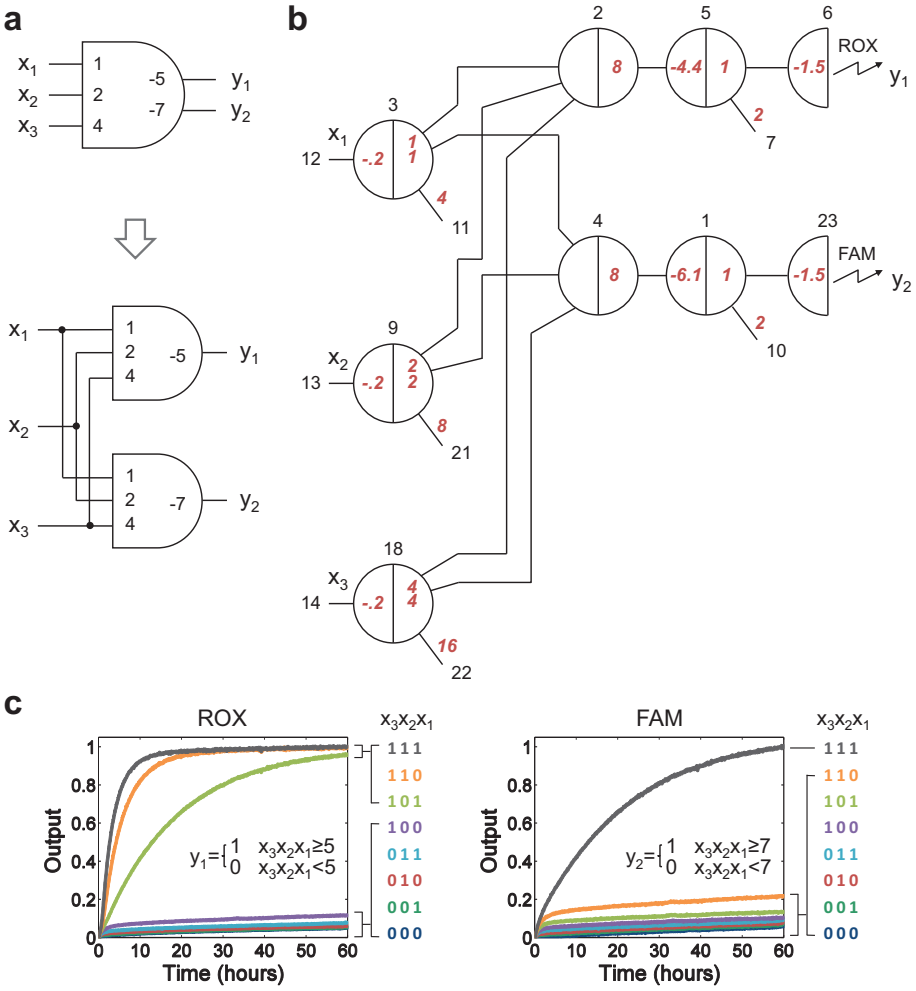



Figure S10: Demonstration of a 3-input 2-output linear threshold gate with thresholds 5 and 7. **(a)** A 3-input 2-output linear threshold gate. **(b)** Equivalent seesaw circuit. **(c)** Kinetics experiments. Experimental conditions were the same as Fig. S9. For output $y_1$, with threshold 5, the ON/OFF gap was between 100 and 101; for output $y_2$, with threshold 7, the ON/OFF gap was between 110 and 111.

**Smaller weights are preferable.** In a linear threshold gate, large weights will result in high concentrations of bound signal species, which by design have a few nucleotides (5 nt in our experiments) that can temporarily bind to the extended toehold of the threshold species (e.g. in Fig. S7, $G_{3:3,2}$ and $G_{9:9,2}$ bind to $Th_{2,5:5}$ via the $s2$ domain) – this will in turn decrease the effectiveness of thresholding. If the linear threshold gate also have large number of outputs, it will result in especially high concentrations of fuel species because initially the fuel is twice the sum of all bound signals (Fig. S6a) – this will in turn increase the leaky reactions between the bound signals and the fuel (e.g. in Fig. S7, $w_{3,f}$ binds to $G_{3:3,2}$ when either end of the double-stranded domain breathes, branch migrates to the other end and release the signal strand $w_{3,2}$). Therefore, for the general linear threshold gate (Fig. 1) with weights up to 4 and 4 outputs, we had to decrease the standard concentration to $1\times = 16.67$ nM in the kinetics experiments to obtain better performance.

**Smaller thresholds are preferable.** Because the signals can only be effectively amplified after overcoming the threshold, large thresholds require more time for the computation to go to completion. In Fig. 1e, the four outputs have roughly the same completion level but with time varying from 8 to 60 hours, $y_1$ with threshold 1 being the fastest and $y_4$ with threshold 7 being the slowest.

**Smaller subthreshold sums are preferable.** Large subthreshold sums produce more leak due to a small percentage of signal reacting with the bound signal molecules without encountering the threshold molecules. In Fig. 1e, the output with the largest subthreshold sum (i.e. the $y_4$ trajectory with input 110) had the worst leak (the highest OFF level).

**Delay of computation occurs with cascades.** The 3-bit XOR gate (Fig. 2) is implemented with two linear threshold gates in cascade, and then translated into four linear threshold gates with the dual-rail representation. When inputs $x_1x_2x_3 = 000$ or 111, the downstream linear threshold gates will be turned ON directly by the input, while in other cases the downstream linear threshold gates will have to wait for the upstream linear threshold gates to produce their outputs. The fact that the output $y^0$ ($y^1$) responded sooner with inputs 000 (111) shows that the computation delay increases with circuit layers. However, all outputs reached roughly the same level after 20 hours. This suggests that with increasing depth, further slowdown can be expected – although presumably not worse than linear as in our previous work.[2]

**Performance variation occurs with different DNA sequences.** The shape of kinetics trajectories, and therefore also the speed of computation, varies among pathways that have the same function but are implemented with different DNA molecules. For example, in Fig. 2c, $y^1 = 1$ (the solid lines) with inputs $x_1x_2x_3 = 001$, 010 and 100 had slightly differently shaped curves than $y^0 = 1$ (the dotted lines) with inputs $x_1x_2x_3 = 011$, 101 and 110, while they computed the same function (with 0 and 1 swapped). Sequences of the extended toeholds on threshold molecules are mostly responsible for three behaviors: (1) The effectiveness of thresholding. Spurious binding happens between single-stranded domains on any signal strand (free or bound) and extended toehold domains on any threshold. Although this spurious binding is unproductive (two strands held together by short domains fall apart quickly), it will decrease the effectiveness of thresholding by temporarily disabling a fraction of threshold molecules. (2) The level of initial activity in signals that stay OFF. Threshold complexes have longer toeholds than gate:output complexes, thus input signals will react faster with a threshold than with a gate:output complex, which means input signals will only effectively release output signals after the threshold molecules are consumed. However, supposing the strand displacement rate of thresholding is 20 times faster than seesawing, there will be roughly 5% of the input signals that react with the gate:output molecules without encountering the threshold molecules first, and we call this skipped-threshold stoichiometric triggering of initial activity in the output signal. The level of initial activity varies because it is determined by the ratio of thresholding and seesawing reaction rates, and the rate of thresholding is sequence dependent. (3) The slope of catalytic reactions in signals that go ON. When the input signal is larger than the threshold, the output signal will be catalytically produced. If the input signal comes from an upstream gate, there will be a delay before the downstream threshold is exceeded. There will nonetheless be initial activity before the output signal goes ON, and the effectiveness of the upstream threshold will change the timing of when the input signal exceeds the downstream threshold, thus changing the slope of the catalytic reaction that releases the output signal.

The experimental data in Fig. 2c imply that the thresholding reaction rates in the $y^1$ pathway are slower (resulting in a larger slope of catalytic reactions and a higher level of initial activities), and those in the $y^0$ pathway are faster (resulting in a smaller slope of catalytic reactions and a lower level of initial activities). A consistent picture is shown: (1) The slopes of $y^1$ turning ON were larger than those of $y^0$. (2) When $y = 1$, the initial activity of $y^1$ was higher and that of $y^0$ was lower. Since $y^1$ turned ON and $y^0$ stayed OFF, these initial activities resulted in an earlier crossing point (roughly at t=0). When $y = 0$, the initial activity of $y^1$ was also higher and that of $y^0$ was also lower. Since $y^1$ stayed OFF and $y^0$ turned ON, these initial activities resulted in a later crossing point (roughly at t=5 hours). Thus, variations caused by DNA sequence differences are apparent, but from the point of view of functionality (computing the correct ON/OFF outputs) they can be neglected.

**Comparison with other implementations.** The 3-bit XOR function has been implemented with four deoxyribozyme-based three-input logic gates,[10] where the computation achieved distinguishable ON/OFF states in 3 hours. Comparing this implementation and ours, there are four things to note: (1) The deoxyribozyme-based 3-bit XOR circuit has the same issue as we encountered with implementing cascadable NOT gates - 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, computations cannot be undone. Thus, to implement a cascadable 3-input XOR gate, the deoxyribozyme-based logic circuit might also need to use dual-rail representation – which would double the size of the circuit. (2) The activated deoxyribozyme logic gates cut nucleic-acid substrates into two short fragments, which can be used to generate increasing fluorescence signals, but cannot directly serve as input signals to downstream gates. Ligating deoxyribozymes might be included to create signals that can activate deoxyribozyme logic gates in cascades.[11] (3) The output signals of the deoxyribozyme-based 3-bit XOR circuit were presented as raw fluorescence levels, and it was sometimes difficult to tell how close to completion the reaction was at the end of the run. In contrast, our output signals were normalized to the maximum completion level. Thus, quantitatively comparing the computation speed of the two implementations is difficult – for example, the ON/OFF states in Fig. 2c could be distinguished much earlier before 20 hours, if we don't require them to go nearly to completion. (4) Our 3-bit XOR implementation works with digital abstraction: input signals can be imperfect (0-0.2 for ON signals and 0.8-1 for OFF signals), which requires extra thresholding and is one of the reasons that the circuit is larger and slower than it could have been. The deoxyribozyme-based 3-bit XOR circuit does not work with digital abstraction, and thus is not capable of handling imperfect signals and errors that would build up through cascades. However, it is clear that 3-bit XOR function can be implemented simpler and faster with deoxyribozymes when downstream cascades and digital abstraction are not desired.

It is not obvious how best to scale up the deoxyribozyme approach[10] to implement the $n$-bit XOR function. Since no existing deoxyribozyme gate takes more than 3 inputs directly, it seems natural to use some mechanism for cascading[11,12] (and possibly also signal restoration[13]) to construct a tree of 3-bit XOR elements with $O(\log n)$ depth and circuit size $O(n)$, which is optimal for AND-OR-NOT circuits.[14] In contrast, our linear threshold circuit implementation can in principle employ the approach of Kautz[15] to implement the $n$-bit XOR function in $O(\log n)$ depth and circuit size $O(\log n)$. This is an exponential gap. It is interesting to note that when constant circuit depth is required (rather than depth that increases with $n$), the difference is even more dramatic: AND-OR-NOT circuits require $O(2^{O(n)})$ gates[16] while linear threshold circuits require only $O(n)$ gates.[17] This difference in complexity applies not only to $n$-bit XOR but to most symmetric functions ($n$-bit functions whose output depends only on the number of 1s in the input, but not on specifically which input variables are 1) and related results show the advantages of linear threshold circuits for many other functions,[18,19] even when weights are restricted to be "small", i.e. polynomial in $n$.

Concerning other implementations of neural network computation, there were proposals of using DNA to store information and create content addressable databases,[20] performing matrix multiplication[21] and vector algebra,[22] and implementing *in vitro* transcriptional networks.[23] These theoretical works ranged from partial aspects to full constructions of neural network computation with DNA. There were also demonstrations of building small DNA databases with associative search capability,[24] addition of two DNA vectors,[25] the weighted-sum operation on a four-bit DNA pattern,[26] and transcriptional circuit implementations of a two node bistable switch[27] and a three node oscillator.[28] These experimental demonstrations showed some basic elements of neural network computation, but did not yield a reliable and complete implementation of even a single neuron – each of the experimental systems lacked at least one of the following aspects: variable weights, variable thresholding, linear integration, the capability of supporting multiple inputs and outputs in multilayer cascades, or predictable quantitative behavior.

Previous game-playing molecular automata – extending the tradition of using game-playing automata as a benchmark for new computing technologies[29] – were constructed using deoxyribozymes. The initial automaton played a symmetrized version of tic-tac-toe,[30] the second automaton played the full tic-tac-toe game,[31] and the third automaton was a generic machine that could be "trained" by the user to play a smaller game.[32] All three automata were implemented as direct lookup tables: for each possible combination of human inputs, a unique deoxyribozyme molecule was prepared to detect that specific input combination and provide the appropriate response – the molecular automaton's computation involved no cascading signals. Further, all three automata, during game play, relied on having distinct mixtures of molecules in several distinct test tubes, so that computation was distributed throughout these non-interacting solutions. In contrast, our DNA strand displacement circuit that plays "read your mind" exists entirely within a single solution, and the computation relies on recurrent (and cascaded) neural network information processing – which can in principle be exponentially more compact than lookup table architectures. (But see the discussion at the end of SI text S5 for analysis of the specific computation performed by the network demonstrated here.)

# S5. In silico training of dual-rail monotone Hopfield associative memories

## Determining weights and thresholds for remembering certain patterns

The perceptron learning algorithm[33–35] is a supervised learning algorithm for training a single linear threshold gate to respond correctly on training set of target input and output pairs. By considering each neuron in a Hopfield associative memory separately, a set of training pairs can be constructed (based on the set of target patterns to be memorized) to ensure that the given neuron behaves correctly to stabilize the target patterns. That is, for any given target pattern, each neuron should be able to examine the states of the other three neurons and output a state that agrees with the pattern. (Thus the pattern is "stable" in the sense that no neuron will want to to change the pattern when it is updated.)

In the example implemented in this paper, we have a set of four patterns for the Hopfield associative memory to "remember", which gives rise to 16 input / output pairs. For example, for pattern (0, 1, 1, 0), the first neuron should take input (1, 1, 0) and output 0; the second neuron should take input (0, 1, 0) and output 1; the third neuron should take input (0, 1, 0) and output 1; the fourth neuron should take input (0, 1, 1) and output 0.

Perceptron learning for associative memories proceeds as follows. Initially, weights and thresholds are assigned to zero for all neurons. At the start of every iteration, each neuron's response to each pattern is computed; if they are all correct, training is done. Otherwise, if the output of a neuron disagrees with a desired pattern, a correction step will be taken so that the weights and threshold yield better output. More specifically, if the output is 0 while the desired state is 1, it suggests that either the weights are not large enough or the threshold is too large. Thus, the correction step will be to increase the weights and decrease the threshold. Similarly, if the output is 1 while the desired state is 0, the correction step will be to decrease the weights and increase the threshold. The perceptron learning theorem guarantees that a finite number of correction steps (which can be bounded above for binary input patterns) will lead to successful training with the desired patterns if the set of patterns is **learnable** – i.e., if there exist weights and thresholds that result in correct behavior for all neurons, then the perceptron learning theorem will find suitable weights and thresholds. However, for some sets of patterns (e.g. sets of very similar patterns) no such weights and thresholds will exist, and the algorithm will continue forever unless aborted after the maximum number of correction steps have been taken. The basic algorithm is summarized below:

```
number of neurons = 4
number of patterns = 4
pattern 1 = (0, 1, 1, 0)
pattern 2 = (1, 1, 1, 1)
pattern 3 = (0, 0, 1, 1)
pattern 4 = (1, 0, 0, 0)
set all weights and thresholds to zero
while not all neurons compute correctly for every pattern do
  choose a random neuron and a random pattern
  if the output of the neuron disagrees with the pattern then
    if the output of the neuron should be 1 then
      add the pattern to the weights (omitting self-weights)
      subtract 1 from the threshold
    else
      subtract the pattern from the weights (omitting self-weights)
      add 1 to the threshold
    end if
  end if
end while
```

Because the classical Hopfield associative memory uses symmetric weights to ensure convergence, we used a variant of the above training algorithm (as described in chapter 13.4 of Rojas[35]) that (1) updates $w_{i,j}$ (the weight of the input to neuron $j$, coming from neuron $i$) whenever $w_{j,i}$ (the weight of the input to neuron $i$, coming from neuron $j$) is updated; (2) represents ON and OFF with $-1$ and $+1$ as in the classical networks, rather than with 0 and 1 as presented here; (3) converts the weights and thresholds to work with the 0 / 1 representation after training. Note that due to randomness in the algorithm, different runs may result in different weights and thresholds, all of which satisfy the requirements.

To establish the number of distinct sets of memories that can be learned by a four-neuron Hopfield associative memory, we enumerated all $\binom{16}{1}+\binom{16}{2}+\binom{16}{3}+\binom{16}{4} = 2516$ such non-empty sets, and found that 500 of them could be learned successfully, 188 of which are sets of four distinct patterns, 208 are sets of three, 88 are sets of 2, and 16 are individual patterns. This is considerably more than would be predicted from the asymptotic capacity[36] of $m = \frac{n}{4 \ln n}$ randomly-chosen patterns that can be perfectly stored in an $n$-bit Hopfield associative memory with high probability. We attribute this difference to the effectiveness of the perceptron learning algorithm for non-orthogonal patterns and to the lack of a sharp boundary between learnable and unlearnable regimes for small $n$. Interestingly, **unlearnable sets** either contained a pair of patterns that differ by a single bit (and hence that neuron cannot uniquely determine its value based on knowing the other three bits) or else implicitly contained the XOR problem (which cannot be solved by a single perceptron[34]).

## A comparison between classical and dual-rail monotone Hopfield associative memories

The dual-rail Hopfield associative memory construction was developed to respect the molecular implementation constraints we encountered with seesaw circuits – specifically, when a signal's ON/OFF states are represented by the presence/absence of the signal molecule, any computation that takes an OFF signal and produce an ON signal might encounter an error because the signal could just hasn't been turned ON yet; once the error is made, the computation cannot be undone for use-once networks such as seesaw circuits. However, having now characterized the dual-rail memory theoretically, it appears to be quite interesting in its own right. The original Hopfield network has two states (0 and 1) for each signal, thus the four-neuron associative memory has $2^4 = 16$ possible initial states and 4 final states (the 4 remembered patterns). That the original Hopfield network is able to recover a corrupted pattern means that when the initial states of the four neurons are not one of the remembered patterns, the network will update the states (either from 0 to 1 or from 1 to 0) until they stabilize at the most similar remembered pattern. The dual-rail monotone Hopfield network has three initial states (?, 0, 1) and four final states (?, 0, 1, x) for each signal, thus the four-neuron associative memory has $3^4 = 81$ possible initial states, and the final state could be one of the remembered patterns, could contain "x" (identifying a pattern that is incompatible with all remembered patterns, as in Fig. 3e bottom right), or could contain "?" (identifying a pattern that is compatible with multiple remembered patterns, as in Fig. 3e bottom left). The dual-rail Hopfield monotone network cannot recover a corrupted pattern because it can only update the states from unknown to 0 or 1 or from there to invalid, but not from 0 to 1 or 1 to 0. The capability of identifying initial patterns that are compatible with multiple remembered patterns or incompatible with all patterns shows the character of being "finicky", while the capability of recovering a corrupted pattern shows the character of being "sloppy". The original "sloppy" feature is desirable when one needs to recognize the species of a cat based on previous memories of different cats; the new "finicky" feature could also be desirable when one needs to search for a lost cat based on previous memories of that cat – i.e., they might be required for different circumstances.

## Elementary facts about dual-rail monotone Hopfield associative memories

After a classical Hopfield associative memory with state variables $x_i$ has been trained to obtain real-valued weights $w_{i,j}$ and thresholds $th_i$, the complementation transformation of Fig. S8a can be used to produce a dual-rail monotone Hopfield associative memory with neuron $i$ state variables $x_i^0$ and $x_i^1$, non-negative weights $w_{i,j}^+$ and $w_{i,j}^-$, and thresholds $th_i^0$ and $th_i^1$. The behavior of this network, while closely related to that of the classical Hopfield associative memory, is not identical.

There are two reasons that a dual-rail monotone Hopfield associative memory is different from a classical one: First, the state of neuron $i$ may be considered to be the pair $x_i^0/x_i^1$; there are thus **four states** for each neuron, 0/1 (**ON**, $x_i = 1$), 1/0 (**OFF**, $x_i = 0$), 0/0 (**unknown**), and 1/1 (**invalid**). The latter two states are not present in the classical model. Second, state changes proceed monotonically. Neuron $i$'s state variable $x_i^1$ may be updated from 0 to 1 if $\sum_j (w_{i,j}^+ x_j^1 + w_{i,j}^- x_i^0) > th_i^1$. Similarly, $x_i^0$ may be updated from 0 to 1 if $\sum_j (w_{i,j}^- x_j^1 + w_{i,j}^+ x_i^0) > th_i^0$. Neither state variable is allowed to change from 1 to 0, which implies that the neuron's state can only be updated from unknown to ON or OFF, and from there to invalid – but not from OFF to ON or from ON to OFF.

Like the classical version, when a dual-rail monotone network is run, state variables are updated asynchronously, one at a time. If a neuron's state is unknown and all its inputs are either ON or OFF, then (as derived in Fig. S8a) update of the neuron's state variables $x_i^0$ and/or $x_i^1$ will turn it ON or OFF in agreement with the corresponding classical neuron with the same input. Accordingly, if this neuron was already in the correct ON or OFF state, then updates will do nothing. However, if the neuron was originally in the incorrect state, it will become invalid after the update.

There are four elementary facts about dual-rail monotone Hopfield associative memories.

**Fact 1: If a pattern is stable in a classical network (i.e. no update will change any neuron's state) then the pattern is also stable in the corresponding dual-rail monotone network.** Furthermore, if $n-1$ bits of this pattern are provided as input, leaving the remaining bit unknown, then the dual-rail monotone network will correctly recall the full pattern. Conversely, if a pattern is unstable in the classical network, then in the dual-rail monotone network updates will eventually yield at least one invalid state.

Now consider the behavior of the dual-rail monotone network starting with arbitrary inputs containing some combination of ON, OFF, and unknown states. (Since the state of the network is considered invalid if any neuron is invalid, we don't consider inputs containing an invalid state.) The neuron's state variables can only go from 0 to 1 but not back, which immediately implies that the network will eventually converge to a stable state – possibly containing unknown or invalid neurons. Note that although the classical network requires symmetric weights to guarantee convergence, the dual-rail monotone network will also converge for asymmetric weights. (This potentially allows for more powerful perceptron learning, but we found that for four-neuron networks, the same number of sets of patterns are learnable, so we used symmetric weights for elegance.) Another difference is that while classical networks starting in the same initial state may settle into a different stable state depending upon the order of asynchronous updates, the monotone properties give rise to deterministic behavior:

**Fact 2: Given any initial state, a dual-rail monotone network will converge to the same stable state regardless of the order of state variable updates.** To see why, note that because all weights are positive, if a state variable *could* change from 0 to 1 were it to be updated (i.e. its weighted sum of inputs exceeds its threshold), then even after many other state variables change from 0 to 1, this state variable will still change from 0 to 1 when next updated (since its weighted sum of inputs has only increased). Thus, the full set of state variables that *could* change to become 1 can be calculated before any updates are performed; eventually they all *will* become 1.

This monotonicity property has implications for the correctness of pattern completion from an incomplete initial state (i.e. initial states with some unknown values). An input is considered **compatible** with a pattern if every neuron in an ON or OFF state agrees with the pattern, and no neurons are in an invalid state.

**Fact 3: If the input is compatible with more than one memorized pattern, then all differing bits will remain unknown.** The proof is by contradiction: suppose neuron $i$ differs in patterns A and B, and it is the first such bit to be updated to either ON or OFF. Prior to this update, the network state P must still be compatible with both A and B. Without loss of generality, consider the case where the update turns neuron $i$ ON, in agreement with A. We conclude that B is not a stable pattern, because B consists of P plus some additional state variables changed from 0 to 1, hence by monotonicity any state variable that can change in P can also change in B, and therefore neuron $i$ (which was OFF in B) will become invalid when updated. But B must be stable, since it is a correctly memorized pattern.

**Fact 4: If the input is compatible with at least one memorized pattern, then no bit will become invalid.** The argument is similar: consider the first bit to become invalid; the pre-update network state is compatible with the memorized pattern, and therefore the same bit could also become invalid in the memorized pattern. An alternative way of stating this fact is that if *any* neuron becomes invalid, we can conclude that the input was incompatible with *every* memorized pattern.

We can now classify the behavior of dual-rail monotone Hopfield associative memories when presented with inputs containing no invalid states:

(1) Input compatible with exactly one memorized pattern may (or may not) be completely reconstructed; no neuron will become invalid, and no neuron will become incompatible with the memorized pattern.

(2) Input compatible with more than one memorized pattern may (or may not) be extended to the maximal set of common bits; no neuron will become invalid, and no neuron will become incompatible with any of these memorized patterns.

(3) Input compatible with no memorized patterns may (or may not) be declared invalid.

For a particular network, if "may (or may not) be" can be replaced by "will be" for all inputs, we say that the network performs **perfect inference**.

While dual-rail linear threshold circuits have been considered previously for optical[37] and nanoelectronic[38] implementations, we are not aware of a previous model that employs monotone updates and exploits the invalid state; perhaps conceptually closest are the more general concept of implicit check bits in neural memories[39] and the framework of inference in networks of relations.[40]

## Testing circuit function with all 81 possible input patterns

After perceptron learning has successfully trained a classical Hopfield associative memory and the corresponding dual-rail monotone Hopfield associative memory has been constructed, the behavior of the dual-rail monotone network is guaranteed to be correct, but not necessarily perfect.

We define two scores to evaluate the performance of an $n$-neuron dual-rail monotone Hopfield associative memory:

**Inference score**. We count how many of the $3^n$ inputs result in *all* logical deductions being made.

(1) For each input compatible with exactly one memorized pattern, if the network fully reconstructs the pattern, score 1.

(2) For each input compatible with more than one memorized pattern, if the network reconstructs all the common bits and leaves the uncommon bits unknown, score 1.

(3) For each input compatible with no memorized pattern, if the network turns all bits to invalid, score 1.

**Association score**. Consider only input that match the first category of the inference score:

(1) For each input containing 0 or 1 unknown bit, if the network reconstructs the pattern, score 0.

(2) For each input containing 2 unknown bits, if the network reconstructs the pattern, score 1.

(3) For each input containing 3 unknown bits, if the network reconstructs the pattern, score 2.

Since dual-rail monotone networks converge deterministically to a stable output pattern, it is straightforward to enumerate all $3^n$ possible input patterns, run the network to convergence, and calculate the inference score and the association score. Different successful training runs that memorize the same set of patterns will yield different weights and thresholds due to random steps in the perceptron learning algorithm. We found that, for all 500 learnable sets of patterns for a four-neuron Hopfield associative memory, the different weights and thresholds typically resulted in widely varying inference and association scores (Fig. S11ab), with only a few sets of patterns ever achieving a perfect inference score (81) or the best association score (20). Interestingly, a hill-climbing procedure was typically able to reduce the number of inference errors by changing the network's threshold values, without altering the weights or stability of the memorized patterns. Applying this procedure multiple times to each of the 500 learnable sets, we found weights and thresholds for every set that result in inference scores consistently larger than 75, including 156 that scored perfectly (Fig. S11cd). Association scores also improved. The network shown in Fig. 3 has an inference score of 78 and an association score of 17, which is typical for networks optimized by hill climbing.

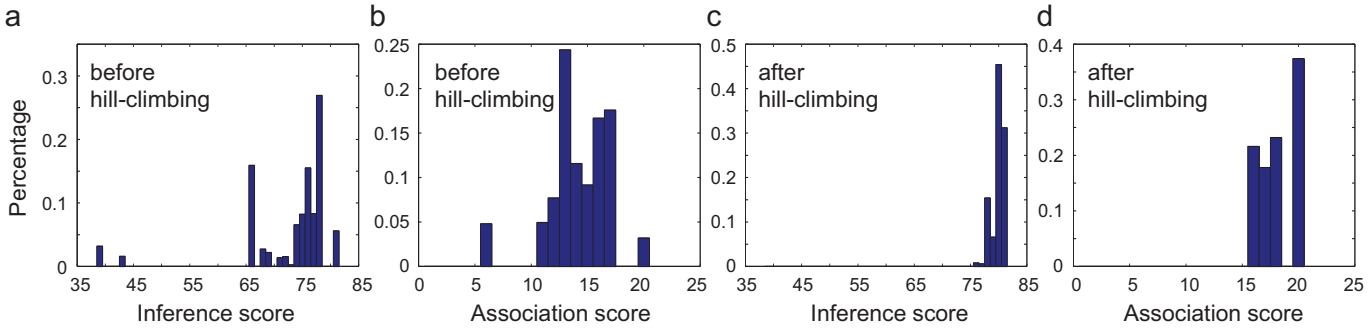



Figure S11: Inference and association scores. **(a)** Histogram of inference scores directly after perceptron learning. Each of the 500 learnable sets was trained 10 times and all scores were reported **(b)** Histogram of association scores directly after perceptron learning, for the same set of trained networks. **(c)** Histogram of inference scores after optimization. Each of the 500 learnable sets was trained by perceptron learning, followed by 50 steps of hill climbing where a small random threshold change was accepted if the inference score improved or stayed the same. This process was repeated 10 times, and the best network selected. **(d)** Histogram of association scores after hill climbing optimization, for the same set of optimized networks.

The perceptron learning algorithm used here is quite primitive compared to modern machine learning techniques, and it would be interesting to consider improved training algorithms for dual-rail monotone Hopfield associative memories. On the other hand, it is satisfying that such a simple learning algorithm, which attempts only to make the target patterns stable memories, nonetheless results in highly effective inference in so many cases.

## Choice of patterns, weights, and thresholds for experimental demonstration

For our experimental system, we wanted a set of four patterns to memorize that had no obvious symmetries, had no constant bit, had few bit positions with a value unique to a pattern (to ensure that our example was of at least typical difficulty), and where learning generated a reasonable set of weights and thresholds and produced a network that performed well on inference tasks. Through ad-hoc manual search, we quickly arrived at the set 0110, 1111, 0011, and 1000, which has only a single bit (the third one) where a value (0) is unique to one of the patterns (the last one). A few training runs were sufficient to find weights and thresholds with which only four inference errors are made. Manual adjustment of thresholds decreased the inference errors to three and simultaneously increased the linear threshold margins – the minimal distance on either side between weighted input sums and the threshold, for any state variable after convergence from any of the 81 possible input patterns. (Interestingly, blindly maximizing the margins sometimes decreases the inference and association scores, as partial inputs sometimes failed to exceed the adjusted thresholds, and the network converged prematurely.) As a final step, following the lesson that smaller weights and thresholds are preferable, we separately rescaled each neuron's weights and thresholds such that the minimum absolute weight was 1.0, which does not change the logical behavior of the neuron. This yielded the weights shown in Fig. 3a. Prior to converting to dual-rail, the weights and thresholds before and after rescaling were (in matrix and vector form):

$$w = \begin{bmatrix} 0 & 4 & -8 & 4 \\ 4 & 0 & 8 & -4 \\ -8 & 8 & 0 & 8 \\ 4 & -4 & 8 & 0 \end{bmatrix} \quad th = \begin{bmatrix} -2 \\ 6 \\ -4 \\ 6 \end{bmatrix} \quad \text{and} \quad w = \begin{bmatrix} 0 & 1 & -2 & 1 \\ 1 & 0 & 2 & -1 \\ -1 & 1 & 0 & 1 \\ 1 & -1 & 2 & 0 \end{bmatrix} \quad th = \begin{bmatrix} -0.5 \\ 1.5 \\ -0.5 \\ 1.5 \end{bmatrix}$$

After rescaling, the linear threshold margin for this network is 0.5, which is a significant fraction of the largest weight. This is good, since our experience suggests that DNA seesaw circuit implementations will perform better when no weights or thresholds are much larger than others, and when subthreshold sums are significantly below the threshold values (SI test S5). It is instructive to consider other weight matrices and thresholds also produced by perceptron learning that successfully stabilize the same four patterns, but which yield worse inference behavior. For example, before and after rescaling:

$$w = \begin{bmatrix} 0 & 2 & -7 & 2 \\ 2 & 0 & 9 & -2 \\ -7 & 9 & 0 & 9 \\ 2 & -2 & 9 & 0 \end{bmatrix} \quad th = \begin{bmatrix} -3.5 \\ 7.5 \\ 2 \\ 7.5 \end{bmatrix} \quad \text{and} \quad w = \begin{bmatrix} 0 & 1.0 & -3.5 & 1.0 \\ 1.0 & 0 & 4.5 & -1.0 \\ -1.0 & 1.3 & 0 & 1.3 \\ 1.0 & -1.0 & 4.5 & 0 \end{bmatrix} \quad th = \begin{bmatrix} -1.75 \\ 3.75 \\ 0.29 \\ 3.75 \end{bmatrix}$$

The inference score is 71 (i.e. 10 inference errors – two failures to invalidate inputs that match no known patterns, and 8 failures to complete inputs that are compatible with a unique known pattern), the association score is 12, and the margin is 0. The large weight and threshold ratios would be likely to cause experimental difficulties, and the small margin could lead to unreliable behavior. In contrast, perceptron learning on the four patterns 0111, 1011, 0100, and 1000 resulted in a network that performed inference perfectly and had an association score of 16. Here, rather than rescaling, we show a logically equivalent network that was simplified by eye. It has a margin of 0.5.

$$w = \begin{bmatrix} 0 & -6 & 1 & 1 \\ -6 & 0 & -1 & -1 \\ 1 & -1 & 0 & 8 \\ 1 & -1 & 8 & 0 \end{bmatrix} \quad th = \begin{bmatrix} -2.5 \\ -3.5 \\ 3.0 \\ 4.0 \end{bmatrix} \quad \text{and} \quad w = \begin{bmatrix} 0 & -1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad th = \begin{bmatrix} -0.5 \\ -0.5 \\ 0.5 \\ 0.5 \end{bmatrix}$$

## Comparison of concentration-trainable molecular networks vs hard-coded molecular networks

Using seesaw circuits, the same set of 72 DNA species – with different initial concentrations – can implement a four-neuron associative memory that stores any of 500 distinct sets of patterns. (Note that 24 of the 72 species are used for additional signal restoration and fluorescence readout.) For a specific memory, such as the four patterns used in Fig. 3, an alternative design approach would be to devise a Boolean expression for each neuron state variable that expresses the logic for perfect inference. As shown in Fig. S12, these Boolean expressions can be simplified using Karnaugh maps,[41] resulting in 6 IDENTITY gates, 18 two-input AND gates, 1 three-input AND gate, 7 three-input OR gates and 1 four-input OR gate. Each logic gate could then be implemented in DNA using strand displacement circuits[13] or using deoxyribozymes.[10] In both example systems, OR gates would not require additional DNA species, thus a total of 25 modified deoxyribozymes or DNA complexes with up to 5 strands would be suffice. (Admittedly, additional complexity would be necessary to allow signal restoration and fluorescence readout.) However, this "hard-coded" approach requires designing and synthesizing new molecules for every new memory to be created, or preparing a library of $8 \cdot [\binom{3}{1} \cdot 2^1 + \binom{3}{2} \cdot 2^2 + \binom{3}{3} \cdot 2^3] = 208$ AND gates and IDENTITY gates for all four-bit learnable sets of patterns. (Here, sets of patterns containing a pair of patterns that differ by a single bit are still unlearnable, but sets of patterns containing the XOR problem are learnable, resulting in a total of 540 learnable sets.) Furthermore, implementing an $n$-bit associative memory with all learnable sets of patterns requires $2n \cdot \sum_{i=1}^{n-1} \binom{n-1}{i} \cdot 2^i \geq n \cdot 2^n$ DNA species for the "hard-coded" digital logic circuits, but requires only $2n \cdot (n+1) = 2n^2 + 4n$ DNA species for the seesaw linear threshold circuits. Note that the complexity of each DNA species in both "hard-coded" example systems will scale with $n$ (presuming it is possible to construct, e.g. $n$-input AND gates with $n > 3$), while in contrast the complexity of each DNA species in the seesaw circuits will always stay the same.

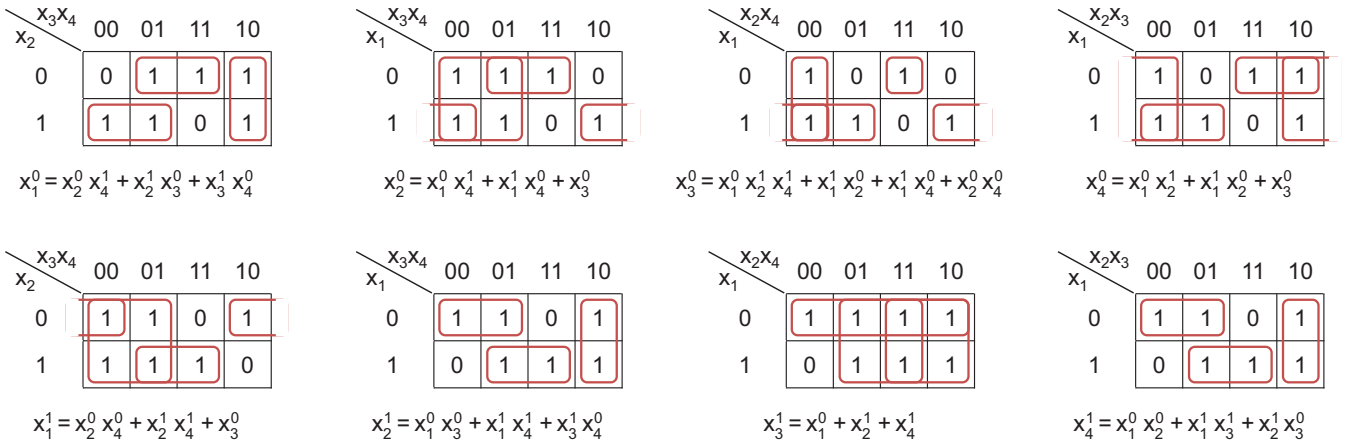



Figure S12: Implementing a memory that stores the four patterns 0110, 1111, 0011 and 1000 with digital logic circuits. Boolean expressions for all neuron state variables are simplified using Karnaugh maps. For example, $x_1^0$ should stay OFF when the input states agree with pattern 1111 or 1000, and turn ON in all other cases (either agreeing with pattern 0110 or 0011, or disagreeing with all four patterns) – so two 0s are written in the positions of $x_2x_3x_4 = 111$ and 000 in the top left Karnaugh map. Each red box encloses the entries computed by a single minterm.

## S6. A four-neuron dual-rail monotone Hopfield associative memory

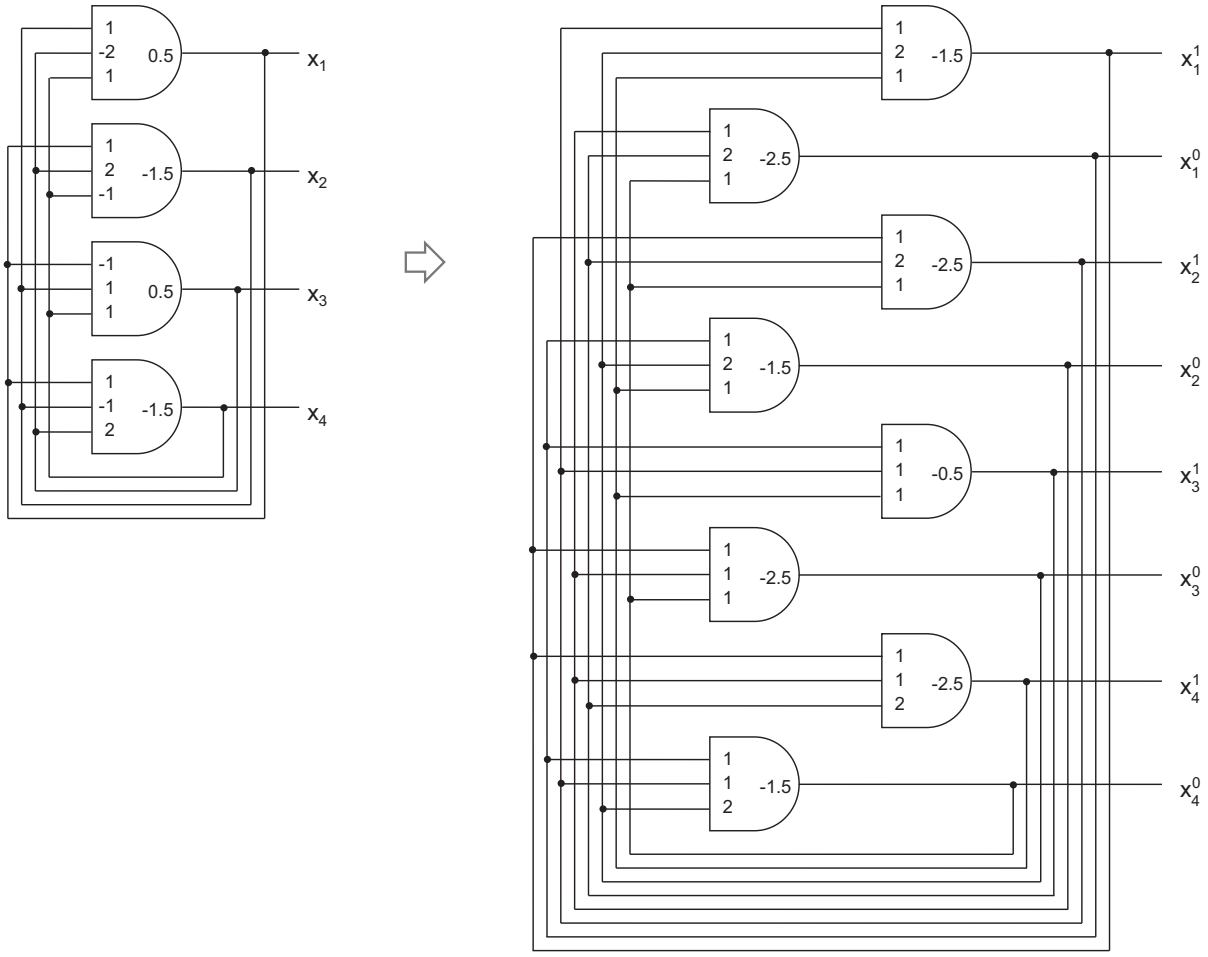



Figure S13: Diagram of a dual-rail four-neuron Hopfield associative memory. Each signal $x_i$ is replaced by a pair of signals $x_i^0$ and $x_i^1$, and each linear threshold gate is replaced by a pair of gates, following the complementation rule (Fig. S5a).

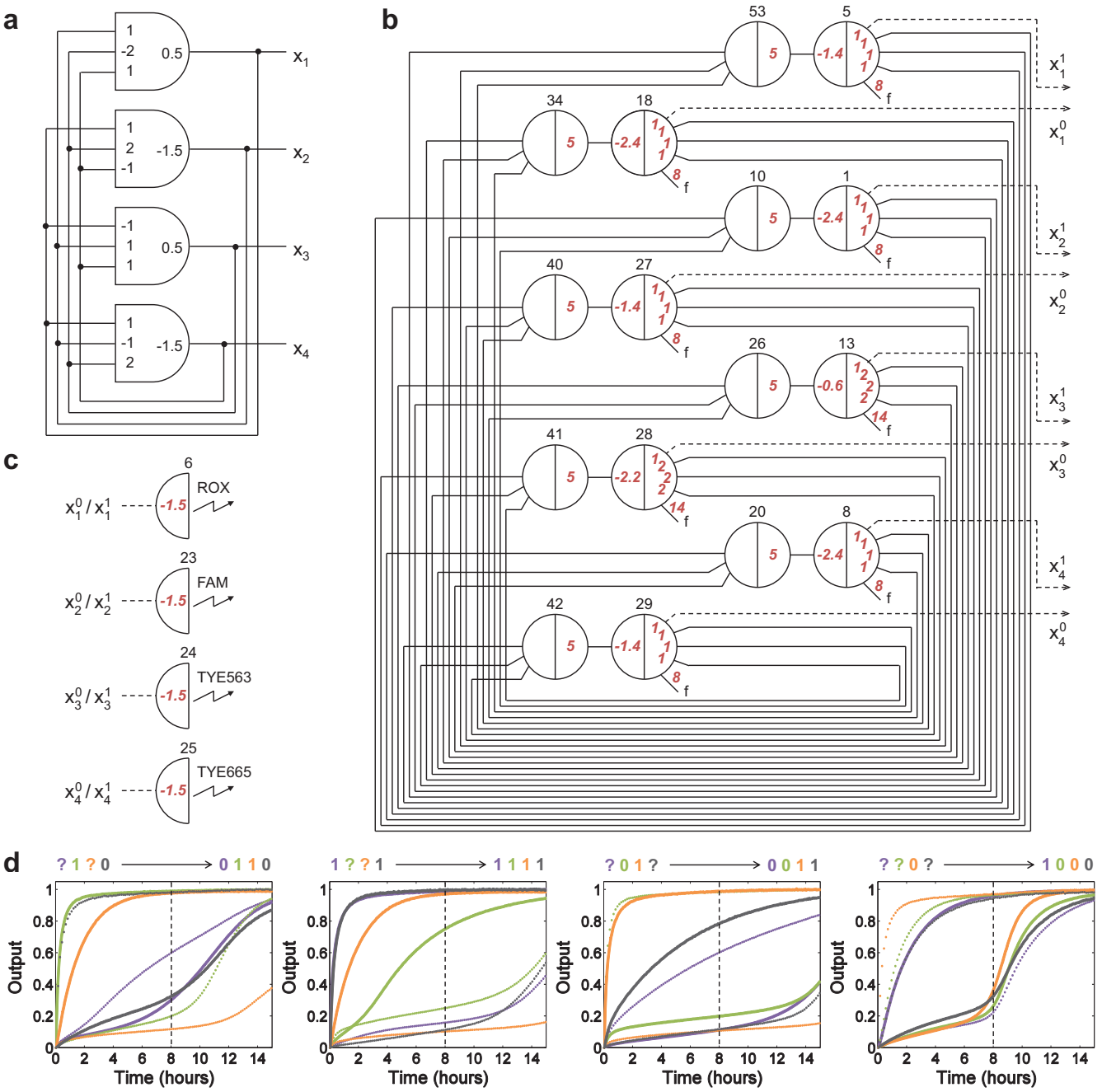



Figure S14: A four-neuron Hopfield associative memory without signal restoration built into reporter pathways. **(a)** A four-neuron Hopfield associative memory constructed as recurrent linear threshold circuit. $x_1$ to $x_4$ represent the states of the four neurons. **(b)** Equivalent seesaw circuit of the four-neuron Hopfield associative memory in its dual-rail implementation. Dotted lines indicate the connections to reporters. $x_i^0$ and $x_i^1$ are dual-rail signals of $x_i$ for each neuron. **(c)** Four reporters without signal restoration. They are connected to either $x_i^0$ or $x_i^1$ at any given time, and associated with ROX, FAM, TYE563 and TYE665 fluorophores. **(d)** Kinetics experiments of the associative memory, performing the "read your mind" game (see Fig. 4a). A total of 84 DNA strands assembled to form 56 initial DNA species (as indicated by the red numbers in subfigures **b** and **c**) were mixed in solution at their respective concentrations. The standard concentration was $1\times = 25$ nM. Selected inputs corresponding to the human's answers were then added with relative concentrations of $5\times$. (To set the initial states, we use inputs that trigger the fan-out, e.g. $w_{53,5}$ for $x_1^1$, $w_{34,18}$ for $x_1^0$, etc.) Dotted and solid lines indicate dual-rail outputs $x_i^0$ and $x_i^1$, respectively. If the dotted line rises high and the solid line stays low, the logic value is 0; if the solid line rises high and the dotted line stays low, the logic value is 1. Arrows connect initial states of the four neurons (inputs) to the expected final states. The expected final states were roughly achieved at 8 hours, but leaky reactions went too high afterwards. These initial experiments suggested that extra signal restoration should be added to suppress the leaky reactions amplified by the feedback loops, and thresholds should be adjusted to approximately synchronize the output times despite rate differences due to the DNA sequences. Sequences of strands are listed in Tables S5 to S7. Experiments were performed at 25°C.


Figure S15: Initial patterns that recall a remembered pattern. Boxes highlight the experimentally demonstrated cases. **(a)** 9 initial patterns that recall Rosalind Franklin. **(b)** 9 initial patterns that recall Alan Turing. **(c)** 9 initial patterns that recall Claude Shannon. **(d)** 9 initial patterns that recall Santiago Ramon y Cajal.

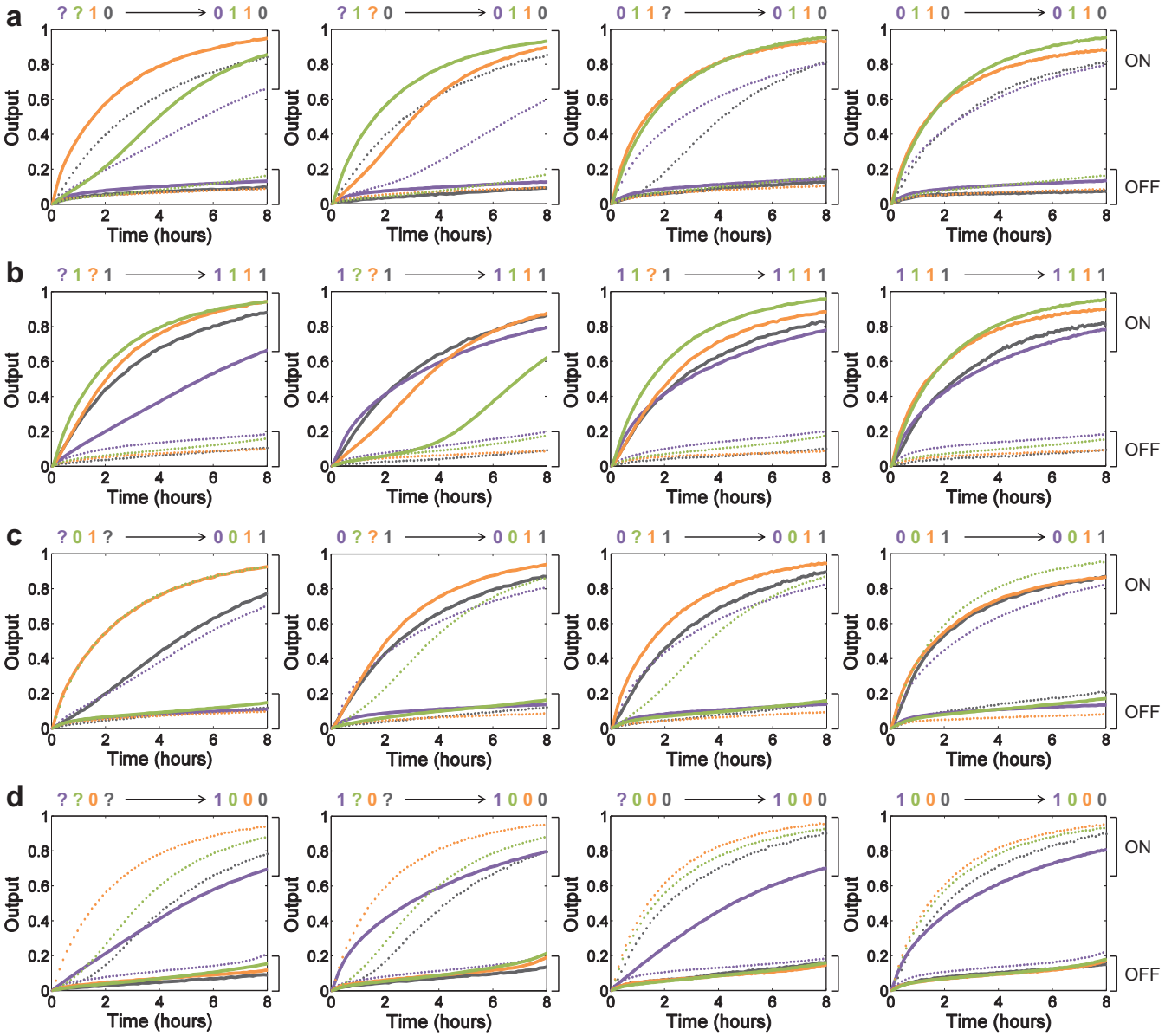



Figure S 16: Kinetics experiments from the "read your mind" game (see Fig. 3d) with initial patterns that recall a remembered pattern. The first plots in subfigures **abd** and the second plot in subfigure **c** are the same data as that presented in the top four plots in Fig. 3e. **(a)** Initial patterns that recall Rosalind Franklin. The more information that was known initially, the faster the pattern was recalled. However, different unknown information resulted in different recalling speed (the first and second data from the left), because the number of neural update cycles before reaching the final states differs. **(b)** Initial patterns that recall Alan Turing. **(c)** Initial patterns that recall Claude Shannon. **(d)** Initial patterns that recall Santiago Ramon y Cajal. A total of 112 DNA strands assembled to form 72 initial DNA species (as indicated by the red numbers in Fig. 3bc) were mixed in solution at their respective concentrations. The standard concentration was $1\times = 25$ nM. Selected inputs corresponding to the human's answers were then added with relative concentrations of $5\times$. Dotted and solid lines indicate dual-rail outputs $x_i^0$ and $x_i^1$, respectively. For each signal, if both dotted and solid lines stay in the range of OFF, the logic value is unknown, "?"; if the dotted (solid) line rises to the range of ON and the solid (dotted) line stays OFF, the logic value is "0" ("1"); if both dotted and solid lines go ON, the logic value is invalid, "x". Arrows connect initial states of the four neurons (inputs) to the final states (outputs at 8 hours). Sequences of strands are listed in Tables S5 to S7. Experiments were performed at 25°C.

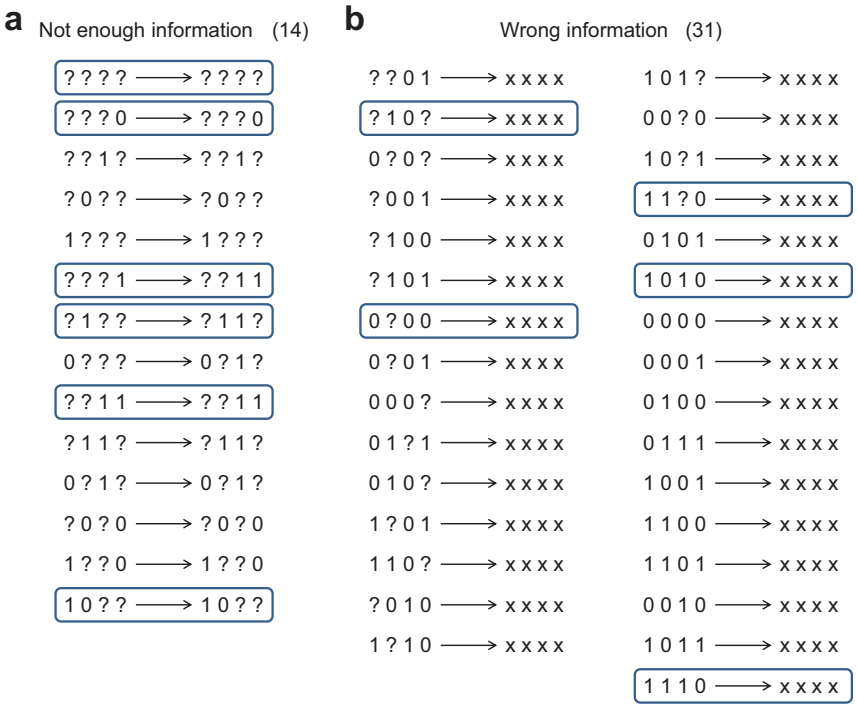



Figure S17: Initial patterns that cannot recall a remembered pattern. Boxes highlight the experimentally demonstrated cases. **(a)** 14 initial patterns that are compatible (as determined by the trained neural network) with more than one remembered patterns, which suggests not enough information. Note that for the last three initial patterns, the neural network fails to realize that there is exactly one remembered pattern that is compatible with the initial pattern. In all other cases, the neural network performs optimally. **(b)** 31 initial patterns that are incompatible with all remembered patterns, which suggest wrong information.

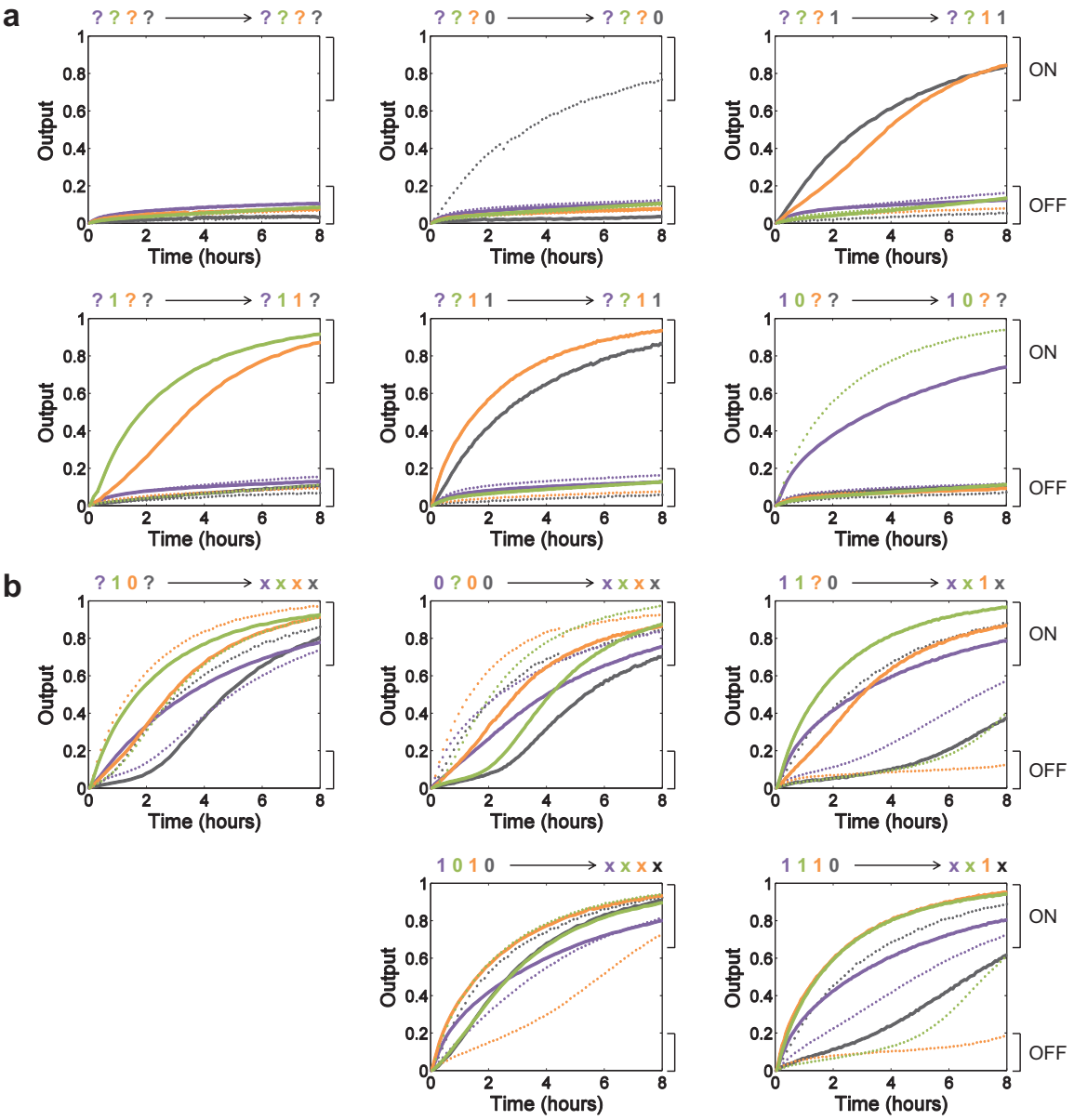



Figure S18: Kinetics experiments of the "read your mind" game (see Fig. 3d) with initial patterns that cannot recall a remembered pattern. The top right plot in subfigure **a** and the top left plot in subfigure **b** are the same data that are presented in the bottom two plots in Fig. 3e. **(a)** Initial patterns that suggest not enough information. Some of the final patterns included more information than the initial patterns (the top right plot and the bottom left plot), indicating more common features shared by the multiple scientists that were compatible with the initial patterns. **(b)** Initial patterns that suggest wrong information. The DNA associative memory was able to determine if the initial pattern was incompatible with all remembered patterns. Note that the process of identifying wrong information could involve as many as four layers of cascades within the feedback loops, while matching a pattern or identifying not enough information involves at most two layers of cascades – this is why in the two right plots, the third bit signal did not turn invalid ("x") yet in 8 hours. However, it is sufficient to identify the wrong information as soon as one bit signal turns invalid. Notations and experimental conditions are the same as Fig. S16.

# S7. Modeling and simulations

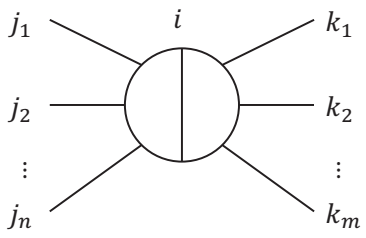
Simplicity and uniformity of modeling and characterization were among the motivating principles for the formulation of the seesaw circuit architecture.[1] In prior work, we found that a simple model with five experimentally-fit parameters was able to semiquantitatively reproduce the behavior of a series of seesaw networks with as many as 33 seesaw nodes.[2] Here, we use this model and the two experimentally-fit parameter sets (one for experiments at 20 °C and one for experiments at 25 °C) with no modifications from the prior work. The model is expressed as a set of formal chemical reaction equations that are derived systematically from the seesaw node representation, then translated into a system of ordinary differential equations (ODEs) for mass action chemical kinetics that may be simulated by a standard ODE solver. We use Mathematica to conveniently represent complex seesaw networks, with functions for translating high-level components (e.g. a linear threshold unit (LTU)) into lower-level seesaw nodes and then into formal chemical reaction networks. Mathematica is also used to perform the numerical integration. An on-line compiler is available that automatically constructs executable Mathematica files.[42]

In the following, the general model framework is presented, then specifications are given for all experiments reported in this work, along with side-by-side comparisons of the experimental data and the simulation data. The remarkable level of agreement achieved suggests that simulations can be a useful guide for predicting performance when developing new seesaw circuits – so long as well-behaved sequences are used. In fact, the occasional deviations between simulations and experiments can be used to pinpoint sequence-dependent problems in the molecular implementation – and, on the other hand, unexpected behavior in the experiments can be provided with an explanation if the simulations demonstrate the same behavior.

As an example, consider the experiments and simulations of the 3-input 4-output general linear threshold gate (Fig. S19). Although the experiments go to completion somewhat faster than the simulations, there are no major qualitative differences in behavior. Prior to examining the simulations, one might wonder what is the explanation of the initial signal activity in the experimental runs that produce an OFF output – is it due to a leak reaction between standard components or to a subpopulation of faulty gates that release their output prematurely? The simulations suggest that neither is the main explanation: First, the model has no subpopulation of faulty gates, yet it reproduces the elevated OFF signals. Second, the 1 /M/s leak rates in the model cannot account for the observed signals (and setting them to zero doesn't significantly change the simulation results in the time periods shown). Thus, the high OFF signals must be attributable to the normal seesaw reaction pathways – in fact, they arise from stoichiometric triggering of seesaw gates by high subthreshold input, some fraction of which will react with the gates before it has a chance to react with the threshold complexes, which is why inputs that produce higher subthreshold sums also exhibit higher initial activity. (This is explained in more detail in SI text S4.) Incidentally, we can also identify the role played by leak reactions in this network: in both experiments and simulations, output signals $y_1$ to $y_4$ each go to completion shortly after the time period shown, much like the plots in Fig. S21. In the model, this can be attributed to the leak reactions in the first two layers, which eventually exceed the thresholds of the third layer threshold gates, and circuit output is at that point catalytically produced. This argument suggests that if we wanted all four outputs to be produced at roughly the same time so that they can be read simultaneously, then we would want to renormalize each of the four LTU subcircuits separately such that their thresholds are comparable. (Rescaling both the weights and the thresholds of an LTU does not change its logical function.)

The XOR circuit experiments also have several features that can be explained in terms of the model (Fig. S20). First, $y^0$ for $x_1x_2x_3 = 000$ and $y^1$ for $x_1x_2x_3 = 111$ go up immediately, while in the other experiments the ON output signals go up only after a delay. This is reproduced by the simulations, and can be attributed to the fact that for those experiments only, signals directly from the inputs is sufficient to activate the output layer LTU, whereas in the other cases the output layer LTU must wait for signal propagation from the upstream first layer LTU. Second, the OFF output signals show some initial activity, which can be explained by skipped-threshold stoichiometric triggering as in the 3-input 4-output general linear threshold gate. More interesting in this case are the differences between the experiments and simulations. We see that OFF $y^0$ signals are lower in experiments than in simulations, while OFF $y^1$ signals are higher; for signals that go ON, we see that $y^0$ exhibits a more pronounced delay phase in experiments, while $y^1$ exhibits a less pronounced delay phase – and as a consequence the point where $y^0$ crosses $y^1$ is delayed relative to simulations when $y = 0$, while it is advanced (or non-existent) when $y = 1$. Here we must look for a sequence-dependent difference between the molecular implementations of the seesaw gates mediating these two pathways. Specifically, we can infer that the thresholds are more effective in the $y^0$ pathway (nodes 5 and 8) than in the $y^1$ pathway (nodes 1 and 17), leading to more pronounced delays and less skipped-threshold stoichiometric triggering.

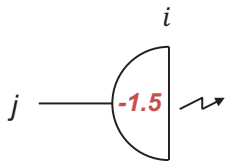
Examining the four-neuron associative memory allows us to identify additional prospective sequence-dependent effects. For the circuit without signal restoration in the reporter pathways (Fig. S21), the trajectory of $x_1^0$ in the ?1?0 and ?01? experiments is noticeably slower than in the simulations, perhaps suggesting a particularly effective threshold in node 18. In the 27 experiments using the circuit with additional signal restoration (Figs. S22-S24), signals $x_1^0$ and $x_1^1$ regularly went up slower than in simulations, suggesting more effective thresholds in nodes 5 and 18. Comparisons between experimental and simulation trajectories of other signals can also be used to infer the sequence-dependent threshold effectiveness in a similar way. The lack of delay (relative to simulations) in many signals that go ON perhaps can be explained by the initial leak caused by impurities and poorly formed gate molecules, which we attribute to synthesis errors (see supplementary information section S10 of our work on seesaw digital circuits[2]).




seesaw$[i, \{j_1, j_2, \cdots, j_n\}, \{k_1, k_2, \cdots, k_m\}]$

$$
= \left\{ \begin{array}{l}
\text{for all } j, j' \in \{j_1, j_2, \cdots, j_n\}, k, k' \in \{k_1, k_2, \cdots, k_m\} \\
\left.\begin{array}{rcl} w_{j,i} + G_{i:i,k} & \underset{k_s}{\overset{k_s}{\rightleftharpoons}} & G_{j,i:i} + w_{i,k} \end{array}\right\} \text{Seesawing reactions} \\
\left.\begin{array}{rcl} w_{j,i} + Th_{j,i:i} & \overset{k_f}{\rightarrow} & waste \end{array}\right\} \text{Thresholding reactions} \\
\left.\begin{array}{rcl} W + G_{i:i,k} & \underset{k_{rf}}{\overset{k_f}{\rightleftharpoons}} & G_{W:i:i,k} \\ G_{j,i:i} + W & \underset{k_{rf}}{\overset{k_f}{\rightleftharpoons}} & G_{j,i:i:W} \\ W + Th_{j,i:i} & \underset{k_{rs}}{\overset{k_f}{\rightleftharpoons}} & Th_{W:j,i:i} \\ w_{j,i} + G & \underset{k_{rf}}{\overset{k_f}{\rightleftharpoons}} & G_{j,i:G} \\ w_{j,i} + TH & \underset{k_{rs}}{\overset{k_f}{\rightleftharpoons}} & Th_{j,i:TH} \end{array}\right\} \text{Universal toehold binding reactions} \\
\left.\begin{array}{rcl} G_{j,i:i} + w_{j',i} & \underset{k_l}{\overset{k_l}{\rightleftharpoons}} & G_{j',i:i} + w_{j,i} \\ G_{i:i,k} + w_{i,k'} & \underset{k_l}{\overset{k_l}{\rightleftharpoons}} & G_{i:i,k'} + w_{i,k} \end{array}\right\} \text{Leak reactions}
\end{array} \right.
$$

| Total signal, gate and threshold | | | |
|---|---|---|---|
| $W(t=0)$ | $=$ | $\sum[w_{i,j}](t=0)$ | for all $i, j$ in the seesaw circuit |
| $G(t=0)$ | $=$ | $\sum[G_{j,i:i}](t=0) + [G_{i:i,k}](t=0) + [Rep_i](t=0)$ | for all $i, j, k$ in the seesaw circuit |
| $TH(t=0)$ | $=$ | $\sum[Th_{j,i:i}](t=0)$ | for all $i, j$ in the seesaw circuit |



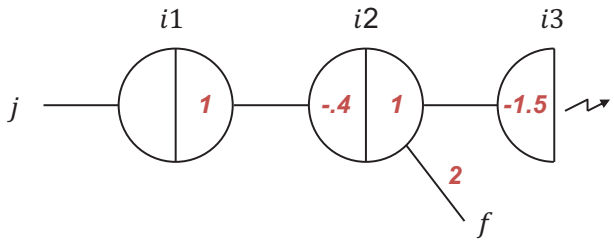

reporter$[i,j]$

$$
= \begin{cases}
w_{j,i} + Rep_i \xrightarrow{k_s} Fluor_i & \text{Reportering reactions} \\
W + Rep_i \underset{k_{rf}}{\overset{k_f}{\rightleftharpoons}} Rep_{W:i} & \text{Universal toehold binding reactions} \\
w_{j,i} + G \underset{k_{rf}}{\overset{k_f}{\rightleftharpoons}} G_{j,i:G} & \\
w_{j,i} + TH \underset{k_{rs}}{\overset{k_f}{\rightleftharpoons}} Th_{j,i:TH} & \\
\text{conc}[Rep_i] = 1.5 \times c & \text{Initial concentration of the reporter}
\end{cases}
$$

| Temperature: 20 ° C | | |
|---|---|---|
| $k_s$ | = | $5 \times 10^4$ M$^{-1}$s$^{-1}$ |
| $k_f$ | = | $2 \times 10^6$ M$^{-1}$s$^{-1}$ |
| $k_{rs}$ | = | 0.5 s$^{-1}$ |
| $k_{rf}$ | = | 10 s$^{-1}$ |
| $k_l$ | = | 1 M$^{-1}$s$^{-1}$ |

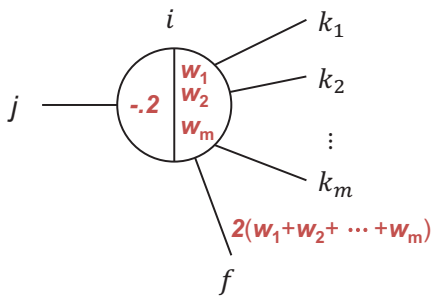
| Temperature: 25 ° C | | |
|---|---|---|
| $k_s$ | = | $5 \times 10^4$ M$^{-1}$s$^{-1}$ |
| $k_f$ | = | $2 \times 10^6$ M$^{-1}$s$^{-1}$ |
| $k_{rs}$ | = | 1.3 s$^{-1}$ |
| $k_{rf}$ | = | 26 s$^{-1}$ |
| $k_l$ | = | 10 M$^{-1}$s$^{-1}$ |

$k_s$ is the slow strand displacement rate of seesawing and reporting reactions. $k_f$ is the fast strand displacement rate of thresholding reactions, and the forward rate of spurious toehold binding reactions due to the universal toehold sequence, both of which reach the maximum DNA hybridization rate. $k_{rs}$ is the backward reaction rate of spurious toehold binding between signal strands and threshold complexes – it is a slow toehold disassociation rate because threshold complexes have extended toeholds that are likely to bind to the signal strands by more than the universal toehold domain. $k_{rf}$ is the backward reaction rate of spurious toehold binding between signal strands and gate:signal complexes – it is a fast toehold disassociation rate. $k_l$ is the leak reaction rate of zero-toehold strand displacement.



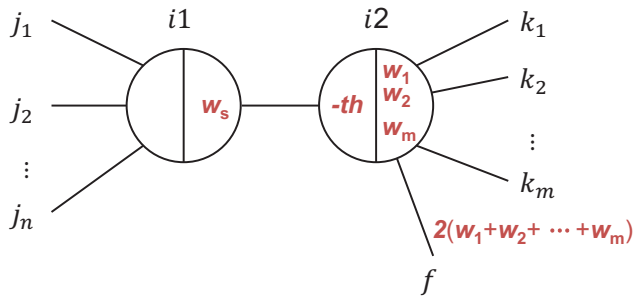

reporterREST$[i1, i2, i3, j]$

$$
= \left\{
\begin{array}{ll}
\left.\begin{array}{c} \text{seesaw}[i1, \{j\}, \{i2\}] \\ \text{seesaw}[i2, \{i1\}, \{i3, f\}] \end{array}\right\} & \text{Signal restoration} \\
\left.\begin{array}{c} w_{i2,i3} + Rep_{i3} \xrightarrow{k_s} Fluor_{i3} \end{array}\right\} & \text{Reportering reactions} \\
\left.\begin{array}{c} W + Rep_{i3} \underset{k_{rf}}{\overset{k_f}{\rightleftharpoons}} Rep_{W:i3} \\ w_{i2,i3} + G \underset{k_{rf}}{\overset{k_f}{\rightleftharpoons}} G_{i2,i3:G} \\ w_{i2,i3} + TH \underset{k_{rs}}{\overset{k_f}{\rightleftharpoons}} Th_{i2,i3:TH} \end{array}\right\} & \text{Universal toehold binding reactions} \\
\left.\begin{array}{rcl} \text{conc}[G_{i1:i1,i2}] & = & 1 \times c \\ \text{conc}[Th_{i1,i2:i2}] & = & 0.4 \times c \\ \text{conc}[G_{i2:i2,i3}] & = & 1 \times c \\ \text{conc}[w_{i2,f}] & = & 2 \times c \\ \text{conc}[Rep_{i3}] & = & 1.5 \times c \end{array}\right\} & \text{Concentrations of the initial species}
\end{array}
\right.
$$




$$\text{inputfanout}[i, j, \{k_1, k_2, \cdots, k_m\}, \{w_1, w_2, \cdots, w_m\}]$$

$$= \begin{cases} \text{seesaw}[i, \{j\}, \{k_1, k_2, \cdots, k_m, f\}] & & \\ \qquad \text{conc}[G_{i:i,k}] & = & w \times c \quad \text{for all } k \in \{k_1, k_2, \cdots, k_m\} \\ & & \quad \text{and their corresponding } w \in \{w_1, w_2, \cdots, w_m\} \\ \qquad \text{conc}[w_{i,f}] & = & 2 \times (w_1 + w_2 + \cdots + w_m) \times c \\ \qquad \text{conc}[Th_{j,i:i}] & = & 1.1 \times 0.2 \times c \end{cases}$$




$$\text{seesawLTU}[i1, i2, \{j_1, j_2, \cdots, j_n\}, \{k_1, k_2, \cdots, k_m\}, w_s, th, \{w_1, w_2, \cdots, w_m\}]$$

$$= \begin{cases} \quad \text{seesaw}[i1, \{j_1, j_2, \cdots, j_n\}, \{i2\}] & & \\ \text{seesaw}[i2, \{i1\}, \{k_1, k_2, \cdots, k_m, f\}] & & \\ \qquad \text{conc}[G_{i1:i1,i2}] & = & w_s \times c \\ \qquad \text{conc}[G_{i2:i2,k}] & = & w \times c \quad \text{for all } k \in \{k_1, k_2, \cdots, k_m\} \\ & & \quad \text{and their corresponding } w \in \{w_1, w_2, \cdots, w_m\} \\ \qquad \text{conc}[w_{i2,f}] & = & 2 \times (w_1 + w_2 + \cdots + w_m) \times c \\ \qquad \text{conc}[Th_{i1,i2:i2}] & = & 1.1 \times th \times c \end{cases}$$

Note that in all simulations presented in this section, nominal threshold concentrations were multiplied by 1.1 to account for empirically higher-than-expected concentrations.

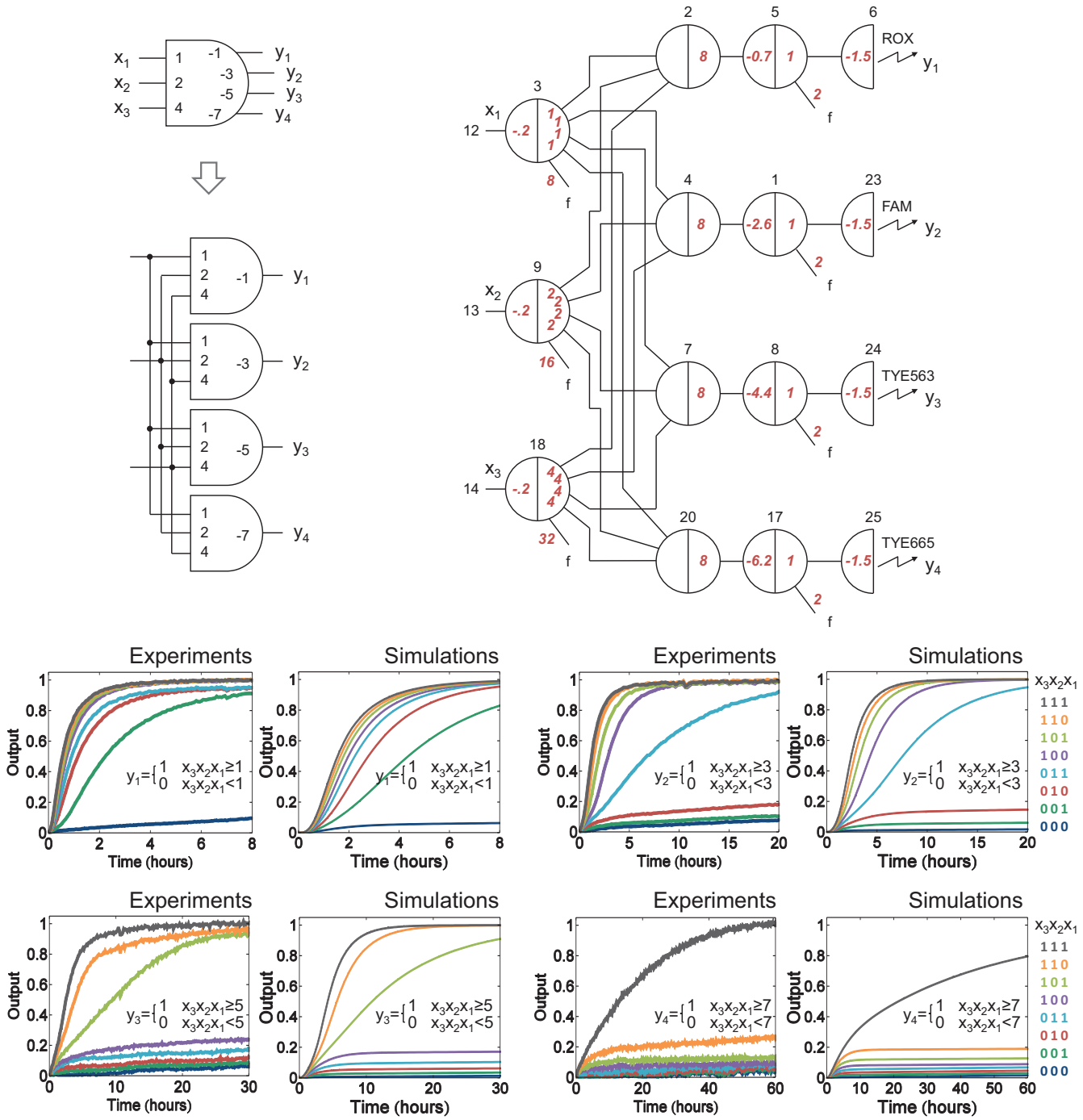



Figure S19: Deterministic simulations of the 3-input 4-output general linear threshold gate. (Experimental data are from Fig. 1f.) Standard concentration $c = 16.67 \times 10^{-9}$ M. Rate parameters were set according to 20° C. Inputs $x_1$ to $x_3$ were either 0.1 (0, logic OFF) or 0.9 (1, logic ON). All deterministic simulations in this section were done by solving ordinary differential equations (ODEs) in Mathematica.

$$
\begin{aligned}
\text{inputfanout}[3, 12, \{2, 4, 7, 20\}, \{1, 1, 1, 1\}] \\
\text{inputfanout}[9, 13, \{2, 4, 7, 20\}, \{2, 2, 2, 2\}] \\
\text{inputfanout}[18, 14, \{2, 4, 7, 20\}, \{4, 4, 4, 4\}] \\
\text{seesawLTU}[2, 5, \{3, 9, 18\}, \{6\}, 8, 0.7, \{1\}] \\
\text{seesawLTU}[4, 1, \{3, 9, 18\}, \{6\}, 8, 2.6, \{1\}] \\
\text{seesawLTU}[7, 8, \{3, 9, 18\}, \{6\}, 8, 4.4, \{1\}] \\
\text{seesawLTU}[20, 17, \{3, 9, 18\}, \{6\}, 8, 6.2, \{1\}] \\
\text{reporter}[6, 5] \\
\text{reporter}[23, 1] \\
\text{reporter}[24, 13] \\
\text{reporter}[25, 8] \\
\text{conc}[w_{12,3}] &= x_1 \times c \\
\text{conc}[w_{13,9}] &= x_2 \times c \\
\text{conc}[w_{14,18}] &= x_3 \times c \\
\text{conc}[W] &= 64 \times c \\
\text{conc}[G] &= 70 \times c \\
\text{conc}[TH] &= 1.1 \times 14.5 \times c \\
y_1 &= Fluor_6 \\
y_2 &= Fluor_{23} \\
y_3 &= Fluor_{24} \\
y_4 &= Fluor_{25}
\end{aligned}
$$

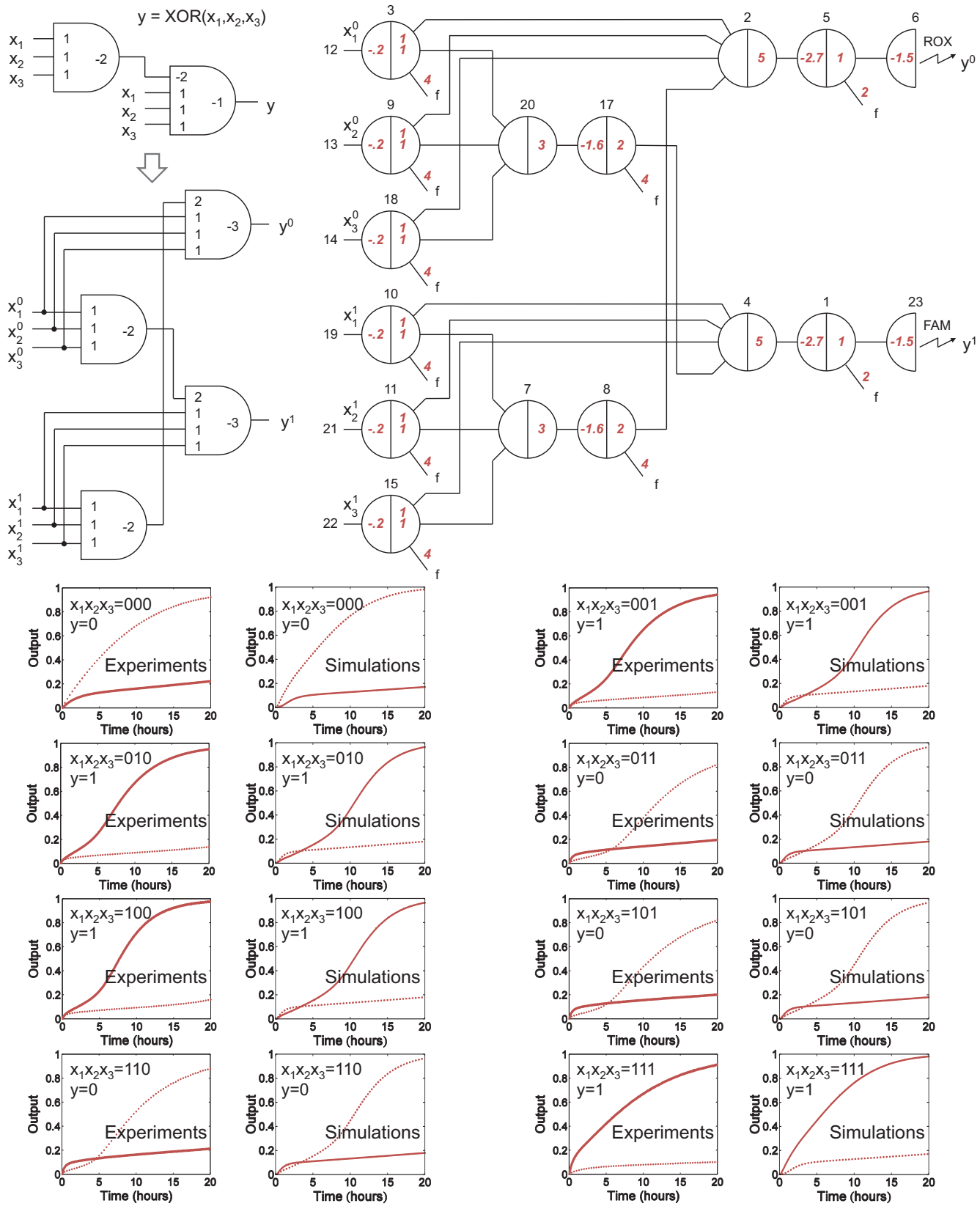



Figure S20: Deterministic simulations of the 3-input XOR circuit. (Experimental data are from Fig. 2c.) Standard concentration $c = 50 \times 10^{-9}$ M. Rate parameters were set according to 25° C. Inputs $x_1$ to $x_3$ were either 0.1 (0, logic OFF) or 0.9 (1, logic ON).

$$
\begin{aligned}
&\text{inputfanout}[3, 12, \{2, 20\}, \{1, 1\}] \\
&\text{inputfanout}[9, 13, \{2, 20\}, \{1, 1\}] \\
&\text{inputfanout}[18, 14, \{2, 20\}, \{1, 1\}] \\
&\text{inputfanout}[10, 19, \{4, 7\}, \{1, 1\}] \\
&\text{inputfanout}[11, 21, \{4, 7\}, \{1, 1\}] \\
&\text{inputfanout}[15, 22, \{4, 7\}, \{1, 1\}] \\
&\text{seesawLTU}[2, 5, \{3, 9, 18, 8\}, \{6\}, 5, 2.7, \{1\}] \\
&\text{seesawLTU}[4, 1, \{10, 11, 15, 17\}, \{23\}, 5, 2.7, \{1\}] \\
&\text{seesawLTU}[20, 17, \{3, 9, 18\}, \{4\}, 3, 1.6, \{2\}] \\
&\text{seesawLTU}[7, 8, \{10, 11, 15\}, \{2\}, 3, 1.6, \{1\}] \\
&\text{reporter}[6, 5] \\
&\text{reporter}[23, 1] \\
\text{conc}[w_{12,3}] &= (1 - x_1) \times c \\
\text{conc}[w_{13,9}] &= (1 - x_2) \times c \\
\text{conc}[w_{14,18}] &= (1 - x_3) \times c \\
\text{conc}[w_{19,10}] &= x_1 \times c \\
\text{conc}[w_{21,11}] &= x_2 \times c \\
\text{conc}[w_{22,15}] &= x_3 \times c \\
\text{conc}[W] &= 36 \times c \\
\text{conc}[G] &= 37 \times c \\
\text{conc}[TH] &= 1.1 \times 9.8 \times c \\
y^0 &= Fluor_6 \\
y^1 &= Fluor_{23}
\end{aligned}
$$

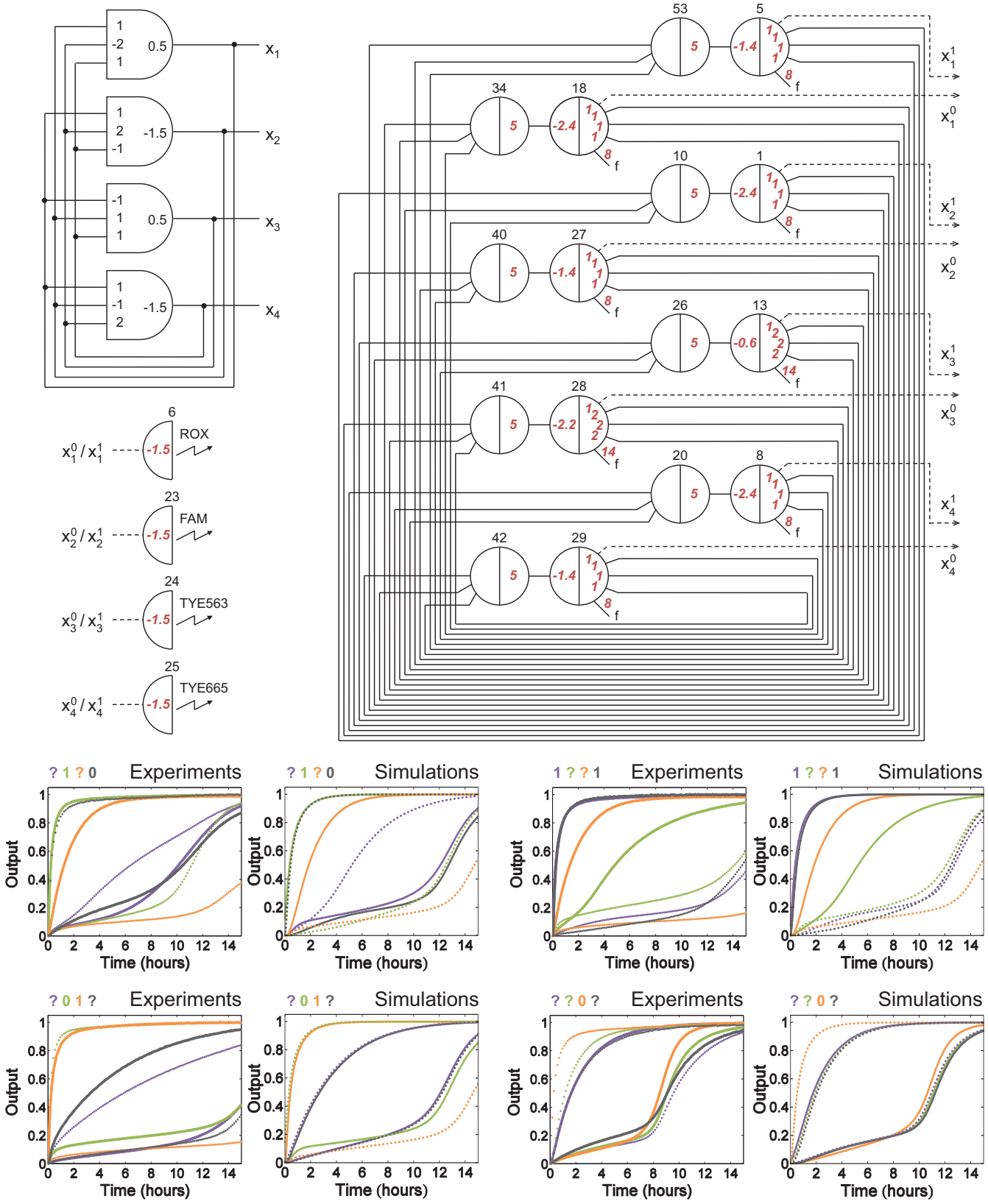



Figure S21: Deterministic simulations of the four-neuron Hopfield associative memory without signal restoration built into reporter pathways. (Experimental data are from Fig. S14.) Standard concentration $c = 25 \times 10^{-9}$ M. Rate parameters were set according to 25° C. Initial values of $x_1^0/x_1^1$ to $x_4^0/x_4^1$ were either 0/0 (?, unknown), 1/0 (0, logic OFF), or 0/1 (1, logic ON). Overlapping trajectories were shifted to be visible.

$$
\begin{aligned}
&\text{seesawLTU}[53,5,\{1,28,8\},\{6,10,41,20\},5,1.4,\{1,1,1,1\}]\\
&\text{seesawLTU}[34,18,\{27,13,29\},\{6,40,26,42\},5,2.4,\{1,1,1,1\}]\\
&\text{seesawLTU}[10,1,\{5,13,19\},\{23,53,26,42\},5,2.4,\{1,1,1,1\}]\\
&\text{seesawLTU}[40,27,\{18,28,8\},\{23,34,41,20\},5,1.4,\{1,1,1,1\}]\\
&\text{seesawLTU}[26,13,\{18,1,8\},\{24,34,10,20\},5,0.6,\{1,2,2,2\}]\\
&\text{seesawLTU}[41,28,\{5,27,29\},\{24,53,40,42\},5,2.2,\{1,2,2,2\}]\\
&\text{seesawLTU}[20,8,\{5,27,13\},\{25,53,40,26\},5,2.4,\{1,1,1,1\}]\\
&\text{seesawLTU}[42,29,\{18,1,28\},\{25,34,10,41\},5,1.4,\{1,1,1,1\}]\\
&\text{reporter}[6,5]\\
&\text{reporter}[23,1]\\
&\text{reporter}[24,13]\\
&\text{reporter}[25,8]\\
&\text{conc}[w_{53,5}] = 5\times x_1^1\times c\\
&\text{conc}[w_{34,18}] = 5\times x_1^0\times c\\
&\text{conc}[w_{10,1}] = 5\times x_2^1\times c\\
&\text{conc}[w_{40,27}] = 5\times x_2^0\times c\\
&\text{conc}[w_{26,13}] = 5\times x_3^1\times c\\
&\text{conc}[w_{41,28}] = 5\times x_3^0\times c\\
&\text{conc}[w_{20,8}] = 5\times x_4^1\times c\\
&\text{conc}[w_{42,29}] = 5\times x_4^0\times c\\
&\text{conc}[W] = 76\times c\\
&\text{conc}[G] = 84\times c\\
&\text{conc}[TH] = 1.1\times 14.2\times c\\
&x_1^1 = Fluor_6\\
&x_2^1 = Fluor_{23}\\
&x_3^1 = Fluor_{24}\\
&x_4^1 = Fluor_{25}\\
&\cdots\cdots\\
&\text{reporter}[6,18]\\
&\text{reporter}[23,27]\\
&\text{reporter}[24,28]\\
&\text{reporter}[25,29]\\
&\cdots\cdots\\
&x_1^0 = Fluor_6\\
&x_2^0 = Fluor_{23}\\
&x_3^0 = Fluor_{24}\\
&x_4^0 = Fluor_{25}
\end{aligned}
$$

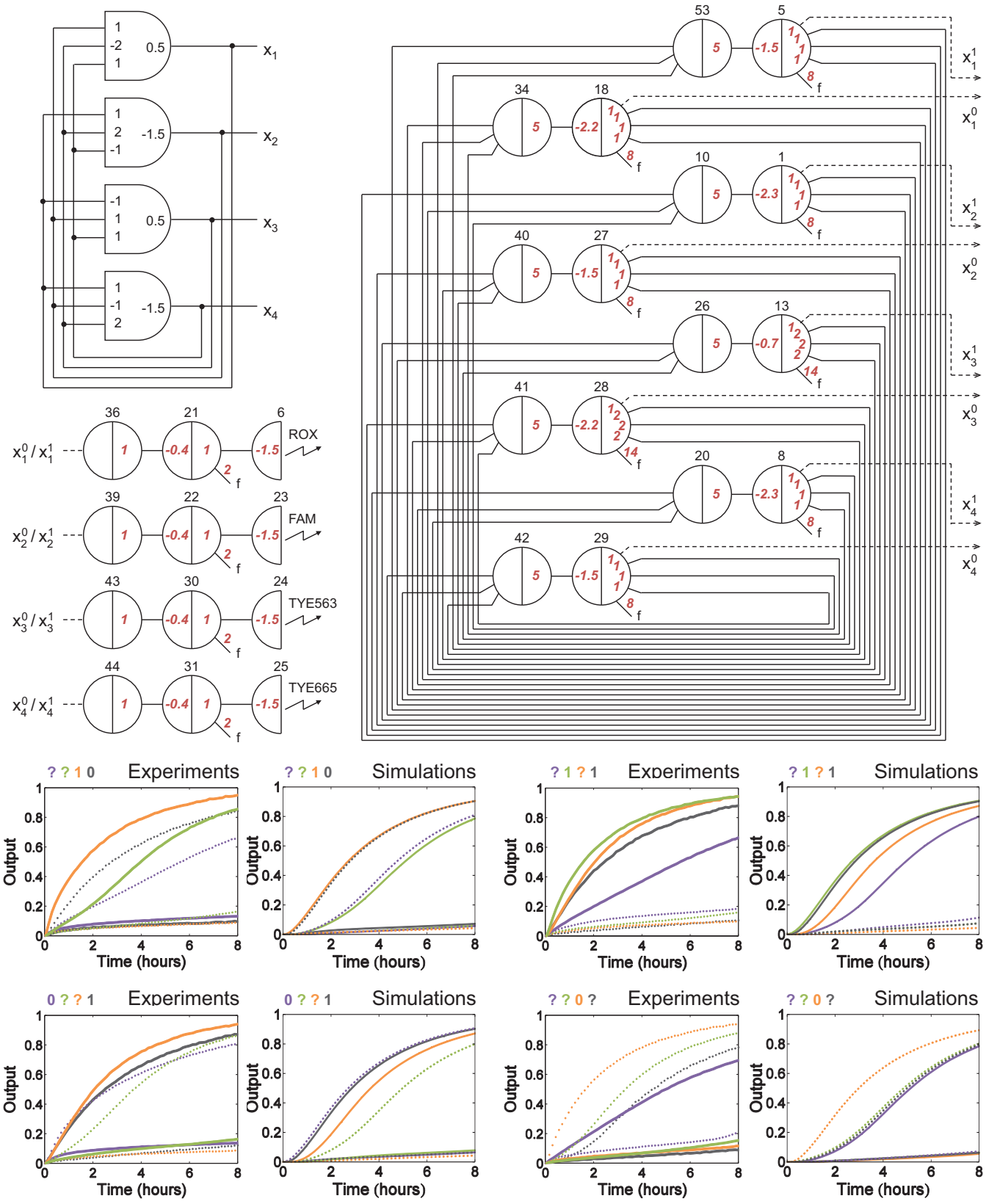



Figure S 22: Deterministic simulations of the four-neuron Hopfield associative memory with signal restoration built into reporter pathways. (Experimental data are from Fig. 3e.) Standard concentration $c = 25 \times 10^{-9}$ M. Rate parameters were set according to 25° C. Initial values of $x_1^0/x_1^1$ to $x_4^0/x_4^1$ were either 0/0 (?, unknown), 1/0 (0, logic OFF), or 0/1 (1, logic ON). Overlapping trajectories were shifted to be visible.

$$
\begin{aligned}
&\text{seesawLTU}[53, 5, \{1, 28, 8\}, \{36, 10, 41, 20\}, 5, 1.5, \{1, 1, 1, 1\}]\\
&\text{seesawLTU}[34, 18, \{27, 13, 29\}, \{36, 40, 26, 42\}, 5, 2.2, \{1, 1, 1, 1\}]\\
&\text{seesawLTU}[10, 1, \{5, 13, 19\}, \{39, 53, 26, 42\}, 5, 2.3, \{1, 1, 1, 1\}]\\
&\text{seesawLTU}[40, 27, \{18, 28, 8\}, \{39, 34, 41, 20\}, 5, 1.5, \{1, 1, 1, 1\}]\\
&\text{seesawLTU}[26, 13, \{18, 1, 8\}, \{43, 34, 10, 20\}, 5, 0.7, \{1, 2, 2, 2\}]\\
&\text{seesawLTU}[41, 28, \{5, 27, 29\}, \{43, 53, 40, 42\}, 5, 2.2, \{1, 2, 2, 2\}]\\
&\text{seesawLTU}[20, 8, \{5, 27, 13\}, \{44, 53, 40, 26\}, 5, 2.3, \{1, 1, 1, 1\}]\\
&\text{seesawLTU}[42, 29, \{18, 1, 28\}, \{44, 34, 10, 41\}, 5, 1.5, \{1, 1, 1, 1\}]\\
&\text{reporterREST}[36, 21, 6, 5]\\
&\text{reporterREST}[39, 22, 23, 1]\\
&\text{reporterREST}[43, 30, 24, 13]\\
&\text{reporterREST}[44, 31, 25, 8]\\
&\text{conc}[w_{53,5}] = 5 \times x_1^1 \times c\\
&\text{conc}[w_{34,18}] = 5 \times x_1^0 \times c\\
&\text{conc}[w_{10,1}] = 5 \times x_2^1 \times c\\
&\text{conc}[w_{40,27}] = 5 \times x_2^0 \times c\\
&\text{conc}[w_{26,13}] = 5 \times x_3^1 \times c\\
&\text{conc}[w_{41,28}] = 5 \times x_3^0 \times c\\
&\text{conc}[w_{20,8}] = 5 \times x_4^1 \times c\\
&\text{conc}[w_{42,29}] = 5 \times x_4^0 \times c\\
&\text{conc}[W] = 84 \times c\\
&\text{conc}[G] = 92 \times c\\
&\text{conc}[TH] = 1.1 \times 15.8 \times c\\
&x_1^1 = Fluor_6\\
&x_2^1 = Fluor_{23}\\
&x_3^1 = Fluor_{24}\\
&x_4^1 = Fluor_{25}
\end{aligned}
$$

$$
\begin{aligned}
&\cdots\cdots\\
&\text{reporterREST}[36, 21, 6, 18]\\
&\text{reporterREST}[39, 22, 23, 27]\\
&\text{reporterREST}[43, 30, 24, 28]\\
&\text{reporterREST}[44, 31, 25, 29]\\
&\cdots\cdots\\
&x_1^0 = Fluor_6\\
&x_2^0 = Fluor_{23}\\
&x_3^0 = Fluor_{24}\\
&x_4^0 = Fluor_{25}
\end{aligned}
$$

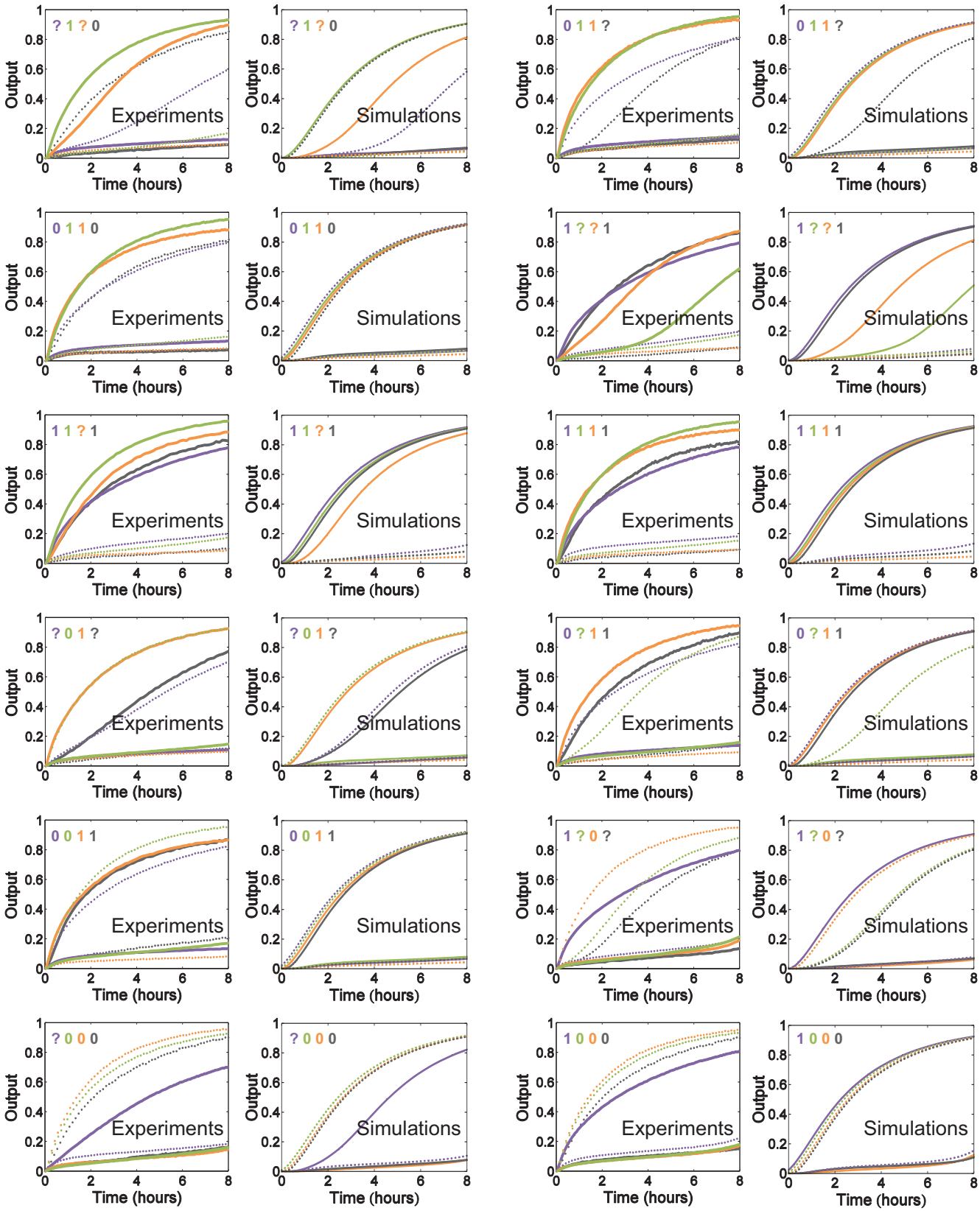



Figure S23: More deterministic simulations of the four-neuron Hopfield associative memory with signal restoration built into reporter pathways, where initial patterns recall a remembered pattern. (Experimental data are from Fig. S16.) Standard concentration $c = 25 \times 10^{-9}$ M. Rate parameters were set according to 25° C. Initial values of $x_1^0/x_1^1$ to $x_4^0/x_4^1$ were either 0/0 (?, unknown), 1/0 (0, logic OFF), or 0/1 (1, logic ON). Overlapping trajectories were shifted to be visible.

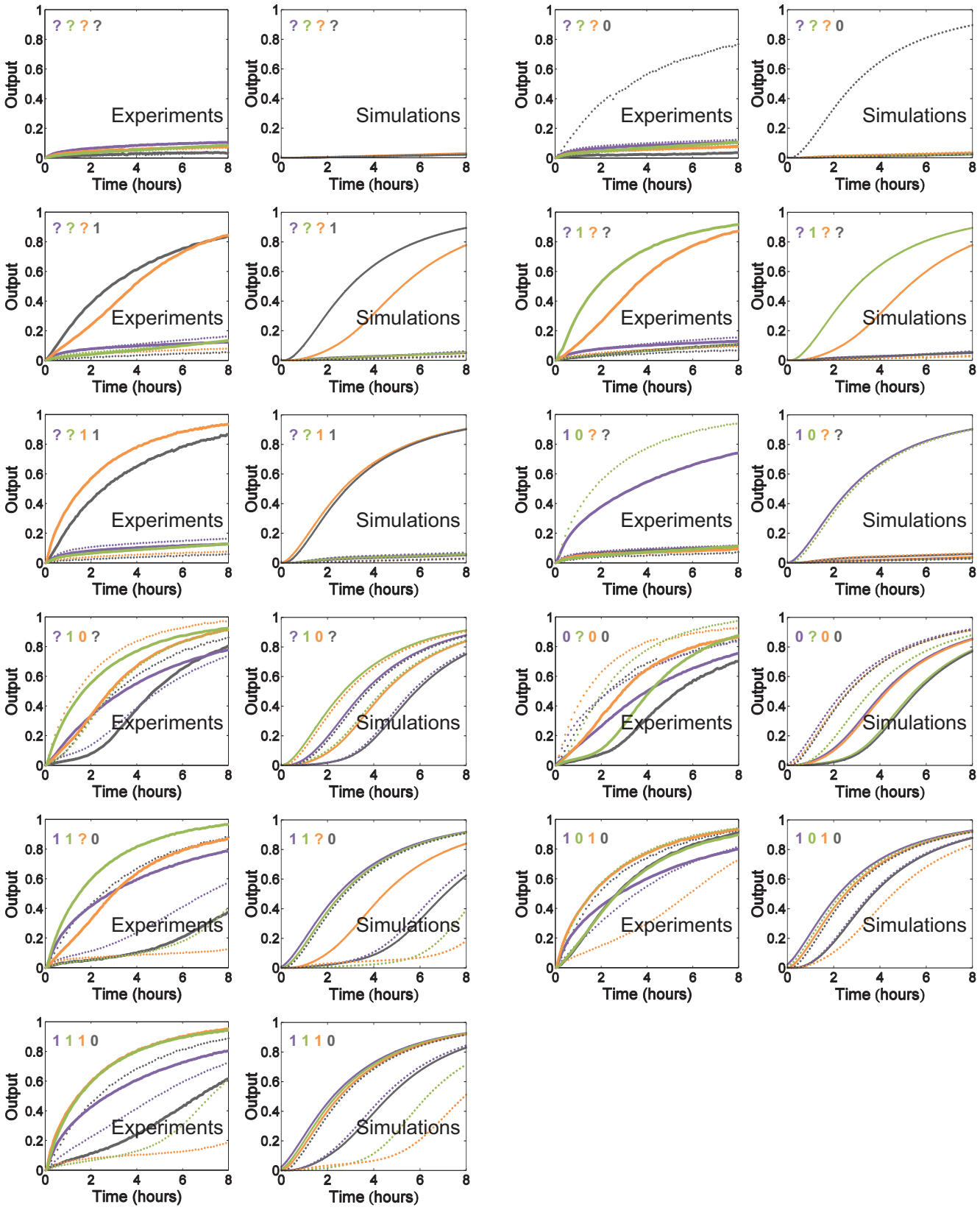



Figure S24: More deterministic simulations of the four-neuron Hopfield associative memory with signal restoration built into reporter pathways, where initial patterns cannot recall a remembered pattern. (Experimental data are from Fig. S18.) Standard concentration $c = 25 \times 10^{-9}$ M. Rate parameters were set according to 25° C. Initial values of $x_1^0/x_1^1$ to $x_4^0/x_4^1$ were either 0/0 (?, unknown), 1/0 (0, logic OFF), or 0/1 (1, logic ON). Overlapping trajectories were shifted to be visible.

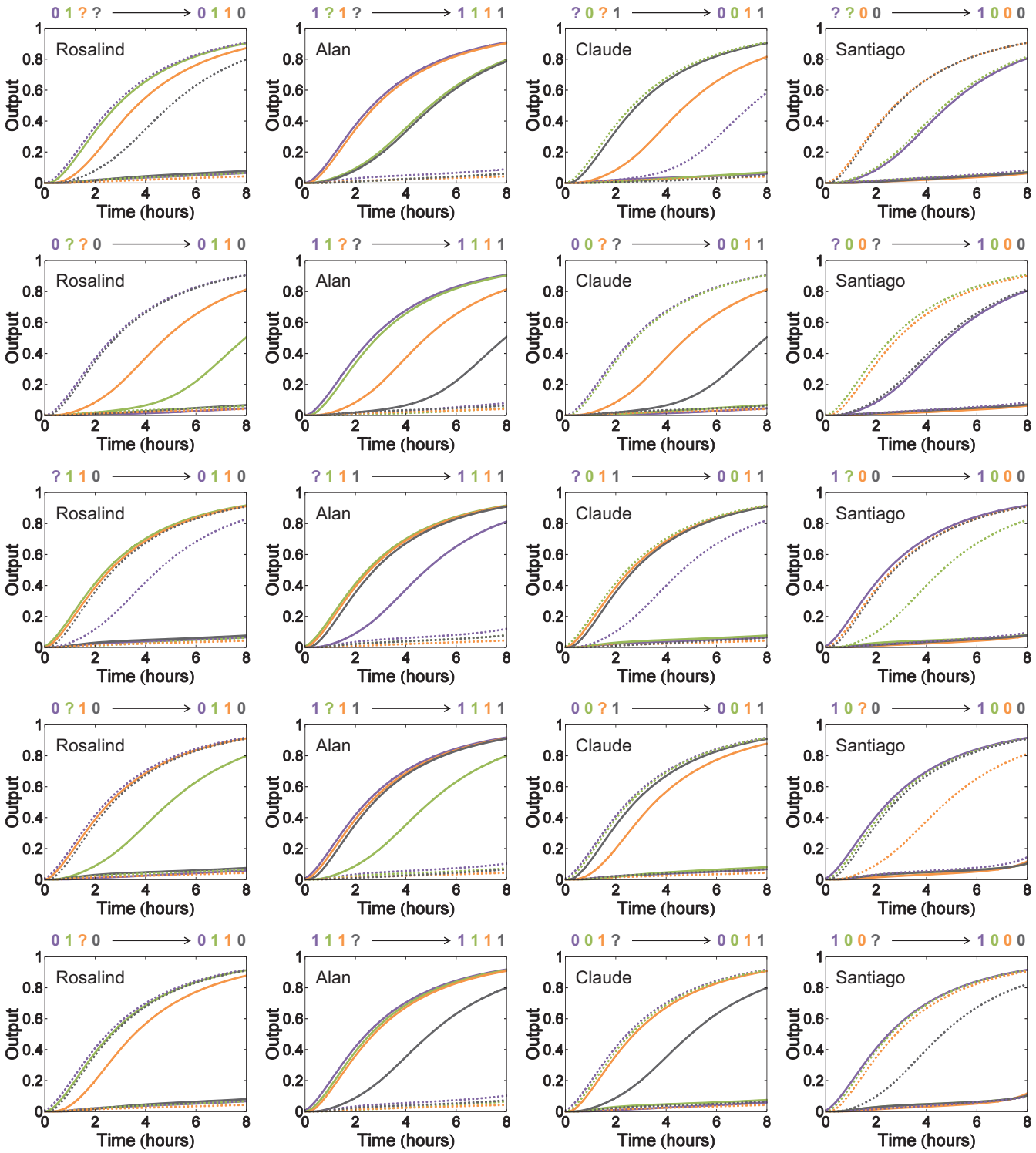



Figure S 25: Deterministic simulations of the four-neuron Hopfield associative memory with signal restoration built into reporter pathways, where initial patterns recall a remembered pattern, predicting the untested games. Standard concentration $c = 25 \times 10^{-9}$ M. Rate parameters were set according to 25° C. Initial values of $x_1^0/x_1^1$ to $x_4^0/x_4^1$ were either 0/0 (?, unknown), 1/0 (0, logic OFF), or 0/1 (1, logic ON). Overlapping trajectories were shifted to be visible.

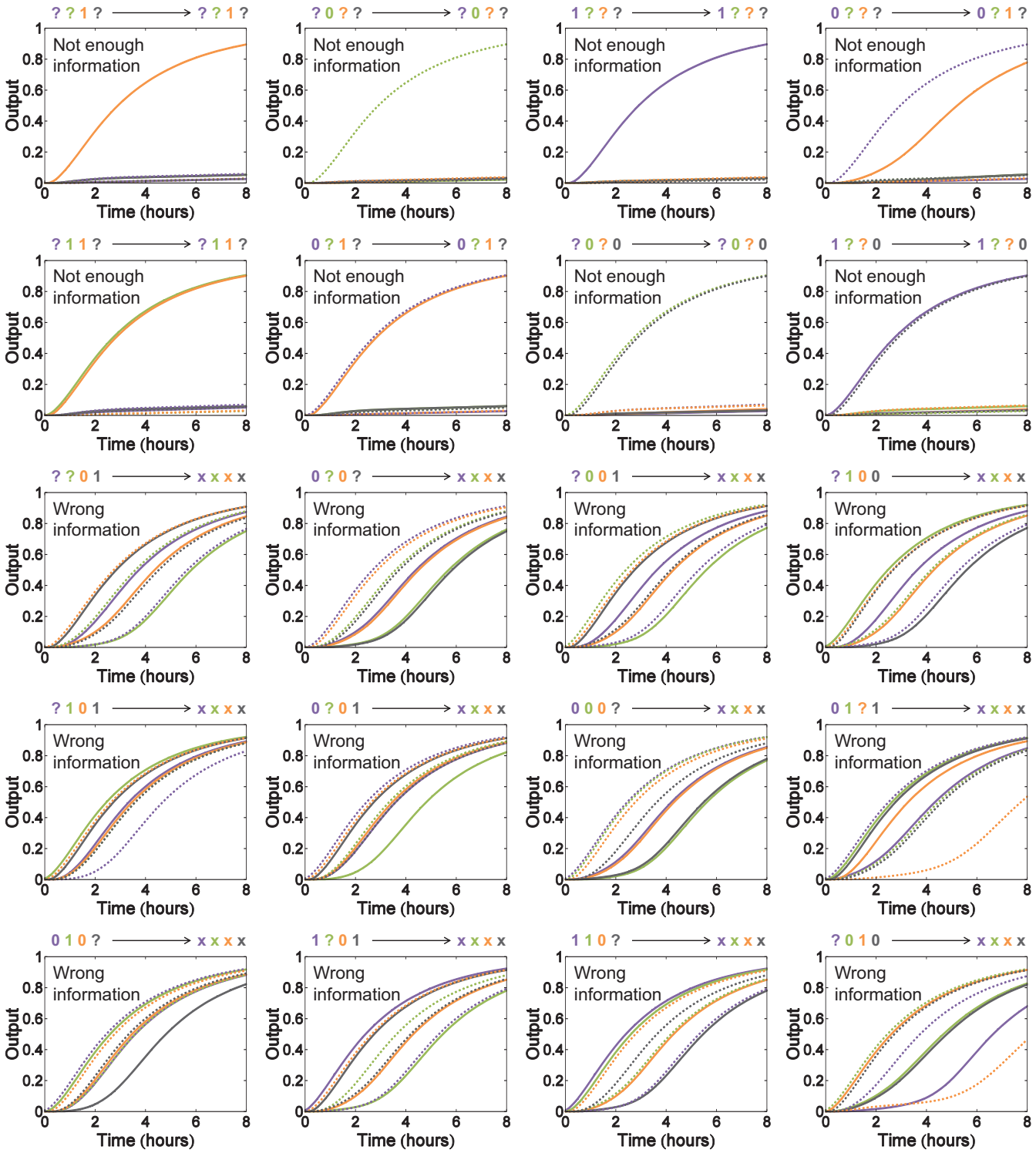



Figure S 26: Deterministic simulations of the four-neuron Hopfield associative memory with signal restoration built into reporter pathways, where initial patterns cannot recall a remembered pattern, predicting the untested games. Standard concentration $c = 25 \times 10^{-9}$ M. Rate parameters are set according to 25° C. Initial values of $x_1^0/x_1^1$ to $x_4^0/x_4^1$ were either 0/0 (?, unknown), 1/0 (0, logic OFF), or 0/1 (1, logic ON). Overlapping trajectories were shifted to be visible. Final states of $x_1x_2x_3x_4$ were written as "xxxx" if as least one pair of $x_i^0/x_i^1$ was invalid (both signals went high) at 8 hours – all four bits will turn invalid in a longer time.

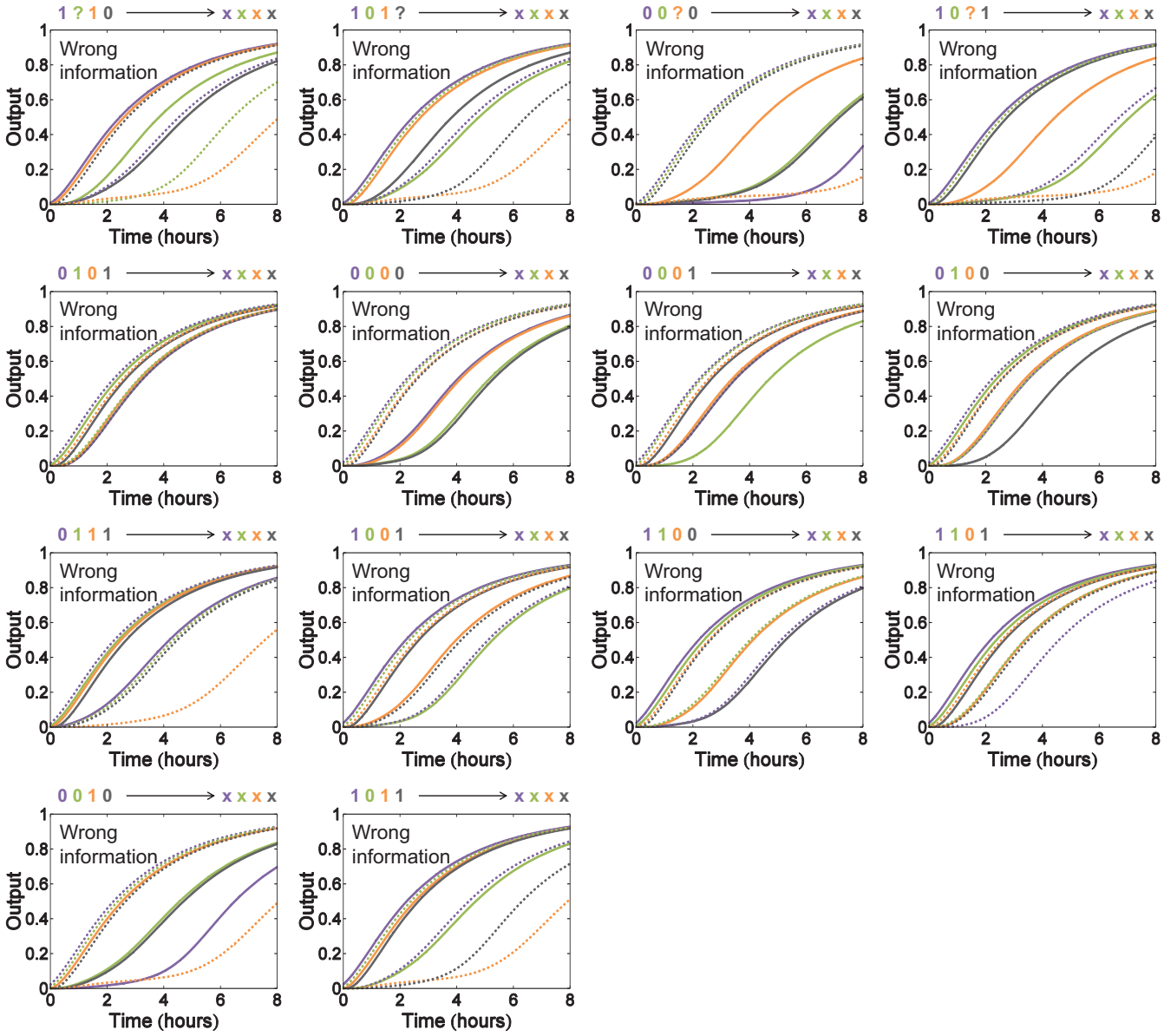



Figure S27: More deterministic simulations of the four-neuron Hopfield associative memory with signal restoration built into reporter pathways, where initial patterns cannot recall a remembered pattern, predicting the untested games. Standard concentration $c = 25 \times 10^{-9}$ M. Rate parameters are set according to 25° C. Initial values of $x_1^0/x_1^1$ to $x_4^0/x_4^1$ were either 0/0 (?, unknown), 1/0 (0, logic OFF), or 0/1 (1, logic ON). Overlapping trajectories were shifted to be visible. Final states of $x_1x_2x_3x_4$ were written as "xxxx" if as least one pair of $x_i^0/x_i^1$ was invalid (both signals went high) at 8 hours – all four bits will turn invalid in a longer time.

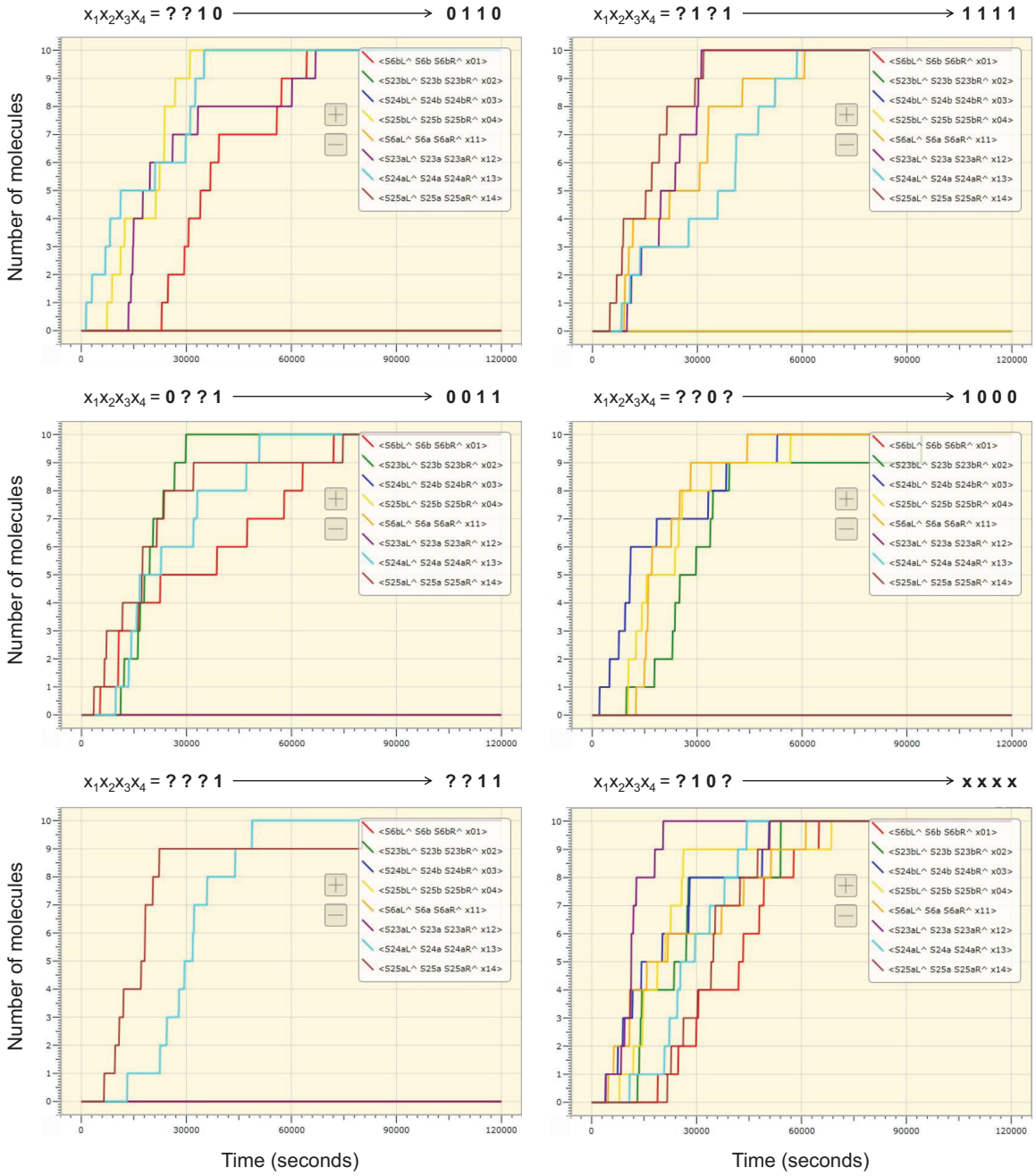



Figure S28: Stochastic simulations of the four-neuron Hopfield associative memory in Visual DSD[43] – a software package that takes a domain level specification of a DNA strand displacement cascade, visualizes the species and reactions at the domain level, generates stochastic and deterministic simulations, etc. The memory was simulated with $k_f = 2 \times 10^6$ $M^{-1}s^{-1}$ and $k_s = 5 \times 10^4$ $M^{-1}s^{-1}$ (same rate constants used in the deterministic simulations). Each species at a 1× concentration was set to 10 copies (all species scale the same, e.g. a 0.7× threshold species would be set to 7 copies). Inputs triggering the update of multiple neurons were set to either 0× or 5×. The semantic model was chosen to be "infinite" and other options were left unchanged as default (which means the spurious toehold binding reactions and leak reactions were not included, and 1× = 10 nM). To specify the extended toeholds of threshold species in DSD, each long recognition domain $Si$ was substituted with three domains: short domain $SiL$, long domain $Si$ and short domain $SiR$. The six games shown in Fig. 1e were run 10 times each, and the first trials are shown here. The shapes of the trajectories vary in all trials of each game, while the signal species all went to the correct ON/OFF states by 120,000 seconds (∼33 hours). In at least 9 trials of each game, the number of all OFF signal species stayed at 0; while in at most 1 trial of each game, the number of one OFF signal species jumped to and then stayed at 1 (which suggests the thresholding is not perfect). In at least 8 trials of each game, the number of all ON signal species went to 10; while in at most 2 trials of each game, the number of one ON signal species went to 9 (which suggests it will take longer than 120,000 seconds to reach the completion level). Visual DSD is online at http://lepton.research.microsoft.com/webdna/. Note that in part because of their relevance to genetic regulatory networks within biological cells, where small numbers of regulatory molecules may be present, stochastic chemical neural networks have previously been considered theoretically.[44, 45]

# S8. Sequences

| | |
|---|---|
| 2 nt clamp | c = [..] |
| 3 nt toehold core | t = […] |
| 15 nt recognition domain | Si = c [……….] c (∀i) |
| 7 nt of Si on the 5' end | si = c […..] (∀i) |

| | shorthand | explicit | |
|---|---|---|---|
| $w_{i,j}$ | = Sj T Si | = Sj t Si | $G_{i:i,j} = w_{i,j} + G_i$-b |
| $G_i$-b | = T* Si* T* | = c* t* Si* t* c* | |
| $Th_{k,i:i}$-t | = Si | | $Th_{k,i:i} = Th_{k,i:i}$-t + $Th_{k,i:i}$-b |
| $Th_{k,i:i}$-b | = sk* T* Si* | = sk* t* Si* | |

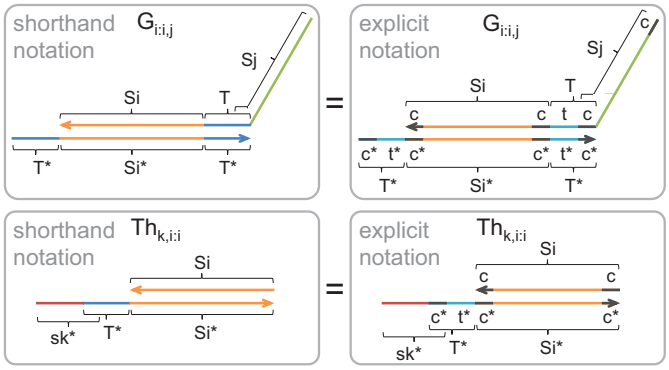



Table S1: Seesaw domain level sequences. Here and elsewhere all sequences start from the 5' end.

| | | | | | |
|---|---|---|---|---|---|
| c | = CA | t | = TCT | | |
| c* | = TG | t* | = AGA | | |
| S1 | = CATCCATTCCACTCA | S2 | = CAAAACAAAACCTCA | S3 | = CACCCTAAAATCTCA |
| S1* | = TGAGTGGAATGGATG | S2* | = TGAGGTTTTGTTTTG | S3* | = TGAGATTTTAGGGTG |
| S4 | = CACATAACAACCACA | S5 | = CACCACCAAACTTCA | S6 | = CATAACACAATCACA |
| S4* | = TGTGGTTGTTATGTG | S5* | = TGAAGTTTGGTGGTG | S6* | = TGTGATTGTGTTATG |
| S7 | = CAACATATCAATTCA | S8 | = CACTAACATACAACA | S9 | = CACCATCAAATAACA |
| S7* | = TGAATTGATATGTTG | S8* | = TGTTGTATGTTAGTG | S9* | = TGTTATTTGATGGTG |
| S10 | = CATACAACATCTACA | S11 | = CAATATCCATAACCA | S12 | = CATCAATCAACACCA |
| S10* | = TGTAGATGTTGTATG | S11* | = TGGTTATGGATATTG | S12* | = TGGTGTTGATTGATG |
| S13 | = CACAACTCATTACCA | S14 | = CATTATTCAAACCCA | S15 | = CACACTATAATTCCA |
| S13* | = TGGTAATGAGTTGTG | S14* | = TGGGTTTGAATAATG | S15* | = TGGAATTATAGTGTG |
| S17 | = CAACTCCTAATATCA | S18 | = CATCTTCTAACATCA | S19 | = CACCTCTTAAACACA |
| S17* | = TGATATTAGGAGTTG | S18* | = TGATGTTAGAAGATG | S19* | = TGTGTTTAAGAGGTG |
| S20 | = CAATCTAACACTCCA | S21 | = CAACCATACTAAACA | S22 | = CATTCCTACATTTCA |
| S20* | = TGGAGTGTTAGATTG | S21* | = TGTTTAGTATGGTTG | S22* | = TGAAATGTAGGAATG |
| S23 | = CAAATCTTCATCCCA | S24 | = CACTCATCCTTTACA | S25 | = CAATTCACTCAATCA |
| S23* | = TGGGATGAAGATTTG | S24* | = TGTAAAGGATGAGTG | S25* | = TGATTGAGTGAATTG |
| S26 | = CATTCATTACCTCCA | S27 | = CAAACACTCTATTCA | S28 | = CATCTACAATTCACA |
| S26* | = TGGAGGTAATGAATG | S27* | = TGAATAGAGTGTTTG | S28* | = TGTGAATTGTAGATG |
| S29 | = CACCAATACTCCTCA | S30 | = CACCATTACAATCCA | S31 | = CAATCCACACTTCCA |
| S29* | = TGAGGAGTATTGGTG | S30* | = TGGATTGTAATGGTG | S31* | = TGGAAGTGTGGATTG |
| S34 | = CACATAACAAAACCA | S36 | = CAAACTAAACAACCA | S39 | = CACTATACACACCCA |
| S34* | = TGGTTTTGTTATGTG | S36* | = TGGTTGTTTAGTTTG | S39* | = TGGGTGTGTATAGTG |
| S40 | = CAATACAAATCCACA | S41 | = CAACAAACCATTACA | S42 | = CACTTTTCACTATCA |
| S40* | = TGTGGATTTGTATTG | S41* | = TGTAATGGTTTGTTG | S42* | = TGATAGTGAAAAGTG |
| S43 | = CATCATACCTACTCA | S44 | = CAAAACTCTCTCTCA | S53 | = CATATCTAATCTCCA |
| S43* | = TGAGTAGGTATGATG | S44* | = TGAGAGAGAGTTTTG | S53* | = TGGAGATTAGATATG |
| Sf | = CATTTTTTTTTTTCA | | | | |

Table S2: Small linear threshold circuit sequences (part one). Shading indicates which strands were used in which circuits. Circuit 1 is the 3-input 2-output general linear threshold gate shown in Figs. S9 and S10. Circuit 2 is the 3-input 4-output general linear threshold gate shown in Fig. 1. Circuit 3 is the XOR circuit shown in Fig. 2. P indicates PAGE purification by IDT, used only for input strands. H indicates HPLC purification by IDT, used only for reporter strand with fluorophores or quenchers. All other strands were ordered without any purification.

| | | Circuit number | 1 | 2 | 3 |
|---|---|---|---|---|---|
| $w_{5,6}$ | = S6 T S5 | = CATAACACAATCACA TCT CACCACCAAACTTCA | | | |
| $G_5$-b | = T* S5* T* | = TG AGA TGAAGTTTGGTGGTG AGA TG | | | |
| $w_{5,7}$ | = S7 T S5 | = CAACATATCAATTCA TCT CACCACCAAACTTCA | | | |
| $w_{5,f}$ | = Sf T S5 | = CATTTTTTTTTTTCA TCT CACCACCAAACTTCA | | | |
| $Th_{2,5:5}$-t | = S5 | = CACCACCAAACTTCA | | | |
| $Th_{2,5:5}$-b | = s2* T* S5* | = TGTTTTG AGA TGAAGTTTGGTGGTG | | | |
| $w_{2,5}$ | = S5 T S2 | = CACCACCAAACTTCA TCT CAAAACAAAACCTCA | | | |
| $G_2$-b | = T* S2* T* | = TG AGA TGAGGTTTTGTTTTG AGA TG | | | |
| $w_{1,23}$ | = S23 T S1 | = CAAATCTTCATCCCA TCT CATCCATTCCACTCA | | | |
| $G_1$-b | = T* S1* T* | = TG AGA TGAGTGGAATGGATG AGA TG | | | |
| $w_{1,10}$ | = S10 T S1 | = CATACAACATCTACA TCT CATCCATTCCACTCA | | | |
| $w_{1,f}$ | = Sf T S1 | = CATTTTTTTTTTTCA TCT CATCCATTCCACTCA | | | |
| $Th_{4,1:1}$-t | = S1 | = CATCCATTCCACTCA | | | |
| $Th_{4,1:1}$-b | = s4* T* S1* | = TTATGTG AGA TGAGTGGAATGGATG | | | |
| $w_{4,1}$ | = S1 T S4 | = CATCCATTCCACTCA TCT CACATAACAACCACA | | | |
| $G_4$-b | = T* S4* T* | = TG AGA TGTGGTTGTTATGTG AGA TG | | | |
| $w_{8,24}$ | = S24 T S8 | = CACTCATCCTTTACA TCT CACTAACATACAACA | | | |
| $G_8$-b | = T* S8* T* | = TG AGA TGTTGTATGTTAGTG AGA TG | | | |
| $w_{8,f}$ | = Sf T S8 | = CATTTTTTTTTTTCA TCT CACTAACATACAACA | | | |
| $Th_{7,8:8}$-t | = T8 | = CACTAACATACAACA | | | |
| $Th_{7,8:8}$-b | = s7* T* S8* | = TATGTTG AGA TGTTGTATGTTAGTG | | | |
| $w_{7,8}$ | = S8 T S7 | = CACTAACATACAACA TCT CAACATATCAATTCA | | | |
| $G_7$-b | = T* S7* T* | = TG AGA TGAATTGATATGTTG AGA TG | | | |
| $w_{17,25}$ | = S25 T S17 | = CAATTCACTCAATCA TCT CAACTCCTAATATCA | | | |
| $G_{17}$-b | = T* S17* T* | = TG AGA TGATATTAGGAGTTG AGA TG | | | |
| $w_{17,f}$ | = Sf T S17 | = CATTTTTTTTTTTCA TCT CAACTCCTAATATCA | | | |
| $Th_{20,17:17}$-t | = S17 | = CAACTCCTAATATCA | | | |
| $Th_{20,17:17}$-b | = s20* T* S17* | = TAGAT TG AGA TGATATTAGGAGTTG | | | |
| $w_{20,17}$ | = S17 T S20 | = CAACTCCTAATATCA TCT CAATCTAACACTCCA | | | |
| $G_{20}$-b | = T* S20* T* | = TG AGA TGGAGTGTTAGATTG AGA TG | | | |

Table S3: Small linear threshold circuit sequences (part two). Shading indicates which strands were used in which circuits. Circuit 1 is the 3-input 2-output general linear threshold gate shown in Figs. S9 and S10. Circuit 2 is the 3-input 4-output general linear threshold gate shown in Fig. 1. Circuit 3 is the XOR circuit shown in Fig. 2. P indicates PAGE purification by IDT, used only for input strands. H indicates HPLC purification by IDT, used only for reporter strand with fluorophores or quenchers. All other strands were ordered without any purification.

| | | Circuit number | 1 | 2 | 3 |
|---|---|---|---|---|---|
| $w_{3,2}$ | = S2 T S3 | = CAAAACAAAACCTCA TCT CACCCTAAAATCTCA | | | |
| $w_{3,4}$ | = S4 T S3 | = CACATAACAACCACA TCT CACCCTAAAATCTCA | | | |
| $w_{3,7}$ | = S7 T S3 | = CAACATATCAATTCA TCT CACCCTAAAATCTCA | | | |
| $w_{3,20}$ | = S20 T S3 | = CAATCTAACACTCCA TCT CACCCTAAAATCTCA | | | |
| $G_3$-b | = T* S3* T* | = TG AGA TGAGATTTTAGGGTG AGA TG | | | |
| $w_{3,11}$ | = S11 T S3 | = CAATATCCATAACCA TCT CACCCTAAAATCTCA | | | |
| $w_{3,f}$ | = Sf T S3 | = CATTTTTTTTTTTCA TCT CACCCTAAAATCTCA | | | |
| $Th_{12,3:3}$-t | = S3 | = CACCCTAAAATCTCA | | | |
| $Th_{12,3:3}$-b | = s12* T* S3* | = ATTGATG AGA TGAGATTTTAGGGTG | | | |
| $w_{12,3}$ | = S3 T S12 | = CACCCTAAAATCTCA TCT CATCAATCAACACCA | P | P | P |
| $w_{9,2}$ | = S2 T S9 | = CAAAACAAAACCTCA TCT CACCATCAAATAACA | | | |
| $w_{9,4}$ | = S4 T S9 | = CACATAACAACCACA TCT CACCATCAAATAACA | | | |
| $w_{9,7}$ | = S7 T S9 | = CAACATATCAATTCA TCT CACCATCAAATAACA | | | |
| $w_{9,20}$ | = S20 T S9 | = CAATCTAACACTCCA TCT CACCATCAAATAACA | | | |
| $G_9$-b | = T* S9* T* | = TG AGA TGTTATTTGATGGTG AGA TG | | | |
| $w_{9,21}$ | = S21 T S9 | = CAACCATACTAAACA TCT CACCATCAAATAACA | | | |
| $w_{9,f}$ | = Sf T S9 | = CATTTTTTTTTTTCA TCT CACCATCAAATAACA | | | |
| $Th_{13,9:9}$-t | = S9 | = CACCATCAAATAACA | | | |
| $Th_{13,9:9}$-b | = s13* T* S9* | = AGTTGTG AGA TGTTATTTGATGGTG | | | |
| $w_{13,9}$ | = S9 T S13 | = CACCATCAAATAACA TCT CACAACTCATTACCA | P | P | P |
| $w_{18,2}$ | = S2 T S18 | = CAAAACAAAACCTCA TCT CATCTTCTAACATCA | | | |
| $w_{18,4}$ | = S4 T S18 | = CACATAACAACCACA TCT CATCTTCTAACATCA | | | |
| $w_{18,7}$ | = S7 T S18 | = CAACATATCAATTCA TCT CATCTTCTAACATCA | | | |
| $w_{18,20}$ | = S20 T S18 | = CAATCTAACACTCCA TCT CATCTTCTAACATCA | | | |
| $G_{18}$-b | = T* S18* T* | = TG AGA TGATGTTAGAAGATG AGA TG | | | |
| $w_{18,22}$ | = S22 T S18 | = CATTCCTACATTTCA TCT CATCTTCTAACATCA | | | |
| $w_{18,f}$ | = Sf T S18 | = CATTTTTTTTTTTCA TCT CATCTTCTAACATCA | | | |
| $Th_{14,18:18}$-t | = S18 | = CATCTTCTAACATCA | | | |
| $Th_{14,18:18}$-b | = s14* T* S18* | = AATAATG AGA TGATGTTAGAAGATG | | | |
| $w_{14,18}$ | = S18 T S14 | = CATCTTCTAACATCA TCT CATTATTCAAACCCA | P | P | P |

Table S4: Small linear threshold circuit sequences (part three). Shading indicates which strands were used in which circuits. Circuit 1 is the 3-input 2-output general linear threshold gate shown in Figs. S9 and S10. Circuit 2 is the 3-input 4-output general linear threshold gate shown in Fig. 1. Circuit 3 is the XOR circuit shown in Fig. 2. P indicates PAGE purification by IDT, used only for input strands. H indicates HPLC purification by IDT, used only for reporter strand with fluorophores or quenchers. All other strands were ordered without any purification.

| | | | Circuit number | 1 | 2 | 3 |
|---|---|---|---|---|---|---|
| $w_{10,4}$ | = S4 T S10 | = | CACATAACAACCACA TCT CATACAACATCTACA | | | |
| $w_{10,7}$ | = S7 T S10 | = | CAACATATCAATTCA TCT CATACAACATCTACA | | | |
| $G_{10}$-b | = T* S10* T* | = | TG AGA TGTAGATGTTGTATG AGA TG | | | |
| $w_{10,f}$ | = Sf T S10 | = | CATTTTTTTTTTTCA TCT CATACAACATCTACA | | | |
| $Th_{19,10:10}$-t | = S10 | = | CATACAACATCTACA | | | |
| $Th_{19,10:10}$-b | = s19* T* S10* | = | AGAGGTG AGA TGTAGATGTTGTATG | | | |
| $w_{19,10}$ | = S10 T S19 | = | CATACAACATCTACA TCT CACCTCTTAAACACA | | | P |
| $w_{11,4}$ | = S4 T S11 | = | CACATAACAACCACA TCT CAATATCCATAACCA | | | |
| $w_{11,7}$ | = S7 T S11 | = | CAACATATCAATTCA TCT CAATATCCATAACCA | | | |
| $G_{11}$-b | = T* S11* T* | = | TG AGA TGGTTATGGATATTG AGA TG | | | |
| $w_{11,f}$ | = Sf T S11 | = | CATTTTTTTTTTTCA TCT CAATATCCATAACCA | | | |
| $Th_{21,11:11}$-t | = S11 | = | CAATATCCATAACCA | | | |
| $Th_{21,11:11}$-b | = s21* T* S11* | = | ATGGTTG AGA TGGTTATGGATATTG | | | |
| $w_{21,11}$ | = S11 T S21 | = | CAATATCCATAACCA TCT CAACCATACTAAACA | | | P |
| $w_{15,4}$ | = S4 T S15 | = | CACATAACAACCACA TCT CACACTATAATTCCA | | | |
| $w_{15,7}$ | = S7 T S15 | = | CAACATATCAATTCA TCT CACACTATAATTCCA | | | |
| $G_{15}$-b | = T* S15* T* | = | TG AGA TGGAATTATAGTGTG AGA TG | | | |
| $w_{15,f}$ | = Sf T S15 | = | CATTTTTTTTTTTCA TCT CACACTATAATTCCA | | | |
| $Th_{22,15:15}$-t | = S15 | = | CACACTATAATTCCA | | | |
| $Th_{22,15:15}$-b | = s22* T* S15* | = | AGGAATG AGA TGGAATTATAGTGTG | | | |
| $w_{22,15}$ | = S15 T S22 | = | CACACTATAATTCCA TCT CATTCCTACATTTCA | | | P |
| $w_{8,2}$ | = S2 T S8 | = | CAAAACAAAACCTCA TCT CACTAACATACAACA | | | |
| $w_{17,4}$ | = S4 T S17 | = | CACATAACAACCACA TCT CAACTCCTAATATCA | | | |
| $Rep_6$-t | = RQ S6 | = | /5IAbRQ/ CATAACACAATCACA | H | H | H |
| $Rep_6$-b | = T* S6* ROX | = | TG AGA TGTGATTGTGTTATG /3ROXN/ | H | H | H |
| $Rep_{23}$-t | = FQ S23 | = | /5IAbFQ/ CAAATCTTCATCCCA | H | H | H |
| $Rep_{23}$-b | = T* S23* FAM | = | TG AGA TGGGATGAAGATTTG /36-FAM/ | H | H | H |
| $Rep_{24}$-t | = RQ S24 | = | /5IAbRQ/ CACTCATCCTTTACA | | H | |
| $Rep_{24}$-b | = T* S24* TYE563 | = | TG AGA TGTAAAGGATGAGTG /3TYE563/ | | H | |
| $Rep_{25}$-t | = RQ S25 | = | /5IAbRQ/ CAATTCACTCAATCA | | H | |
| $Rep_{25}$-b | = T* S25* TYE665 | = | TG AGA TGATTGAGTGAATTG /3TYE665/ | | H | |

Table S5: Hopfield associative memory sequences (part one). Reporter strands are the same as Table S4. circuit 2.

| Gate strands (IDT unpurified) | | |
|---|---|---|
| $w_{5,10}$ | = S10 T S5 | = CATACAACATCTACA TCT CACCACCAAACTTCA |
| $w_{5,41}$ | = S41 T S5 | = CAACAAACCATTACA TCT CACCACCAAACTTCA |
| $w_{5,20}$ | = S20 T S5 | = CAATCTAACACTCCA TCT CACCACCAAACTTCA |
| $w_{5,f}$ | = Sf T S5 | = CATTTTTTTTTTTCA TCT CACCACCAAACTTCA |
| $G_5$-b | = T* S5* T* | = TG AGA TGAAGTTTGGTGGTG AGA TG |
| $Th_{53,5:5}$-t | = S5 | = CACCACCAAACTTCA |
| $Th_{53,5:5}$-b | = s53* T* S5* | = AGATATG AGA TGAAGTTTGGTGGTG |
| $w_{53,5}$ | = S5 T S53 | = CACCACCAAACTTCA TCT CATATCTAATCTCCA |
| $G_{53}$-b | = T* S53* T* | = TG AGA TGGAGATTAGATATG AGA TG |
| $w_{1,53}$ | = S53 T S1 | = CATATCTAATCTCCA TCT CATCCATTCCACTCA |
| $w_{1,26}$ | = S26 T S1 | = CATTCATTACCTCCA TCT CATCCATTCCACTCA |
| $w_{1,42}$ | = S42 T S1 | = CACTTTTCACTATCA TCT CATCCATTCCACTCA |
| $w_{1,f}$ | = Sf T S1 | = CATTTTTTTTTTTCA TCT CATCCATTCCACTCA |
| $G_1$-b | = T* S1* T* | = TG AGA TGAGTGGAATGGATG AGA TG |
| $Th_{10,1:1}$-t | = S1 | = CATCCATTCCACTCA |
| $Th_{10,1:1}$-b | = s10* T* S1* | = TTGTATG AGA TGAGTGGAATGGATG |
| $w_{10,1}$ | = S1 T S10 | = CATCCATTCCACTCA TCT CATACAACATCTACA |
| $G_{10}$-b | = T* S10* T* | = TG AGA TGTAGATGTTGTATG AGA TG |
| $w_{13,34}$ | = S34 T S13 | = CACATAACAAAACCA TCT CACAACTCATTACCA |
| $w_{13,10}$ | = S10 T S13 | = CATACAACATCTACA TCT CACAACTCATTACCA |
| $w_{13,20}$ | = S20 T S13 | = CATACAACATCTACA TCT CACAACTCATTACCA |
| $w_{13,f}$ | = Sf T S13 | = CATTTTTTTTTTTCA TCT CACAACTCATTACCA |
| $G_{13}$-b | = T* S13* T* | = TG AGA TGGTAATGAGTTGTG AGA TG |
| $Th_{26,13:13}$-t | = S13 | = CACAACTCATTACCA |
| $Th_{26,13:13}$-b | = s26* T* S13* | = ATGAATG AGA TGGTAATGAGTTGTG |
| $w_{26,13}$ | = S13 T S26 | = CACAACTCATTACCA TCT CATTCATTACCTCCA |
| $G_{26}$-b | = T* S26* T* | = TG AGA TGGAGGTAATGAATG AGA TG |
| $w_{8,53}$ | = S53 T S8 | = CATATCTAATCTCCA TCT CACTAACATACAACA |
| $w_{8,40}$ | = S40 T S8 | = CAATACAAATCCACA TCT CACTAACATACAACA |
| $w_{8,26}$ | = S26 T S8 | = CATTCATTACCTCCA TCT CACTAACATACAACA |
| $w_{8,f}$ | = Sf T S8 | = CATTTTTTTTTTTCA TCT CACTAACATACAACA |
| $G_8$-b | = T* S8* T* | = TG AGA TGTTGTATGTTAGTG AGA TG |
| $Th_{20,8:8}$-t | = S8 | = CACTAACATACAACA |
| $Th_{20,8:8}$-b | = s20* T* S8* | = TAGATTG AGA TGTTGTATGTTAGTG |
| $w_{20,8}$ | = S8 T S20 | = CACTAACATACAACA TCT CAATCTAACACTCCA |
| $G_{20}$-b | = T* S20* T* | = TG AGA TGGAGTGTTAGATTG AGA TG |

Table S6: Hopfield associative memory sequences (part two). Reporter strands are the same as Table S4. circuit 2.

| Gate strands (IDT unpurified) | | |
|---|---|---|
| $w_{18,40}$ | = S40 T S18 | = CAATACAAATCCACA TCT CATCTTCTAACATCA |
| $w_{18,26}$ | = S26 T S18 | = CATTCATTACCTCCA TCT CATCTTCTAACATCA |
| $w_{18,42}$ | = S42 T S18 | = CACTTTTCACTATCA TCT CATCTTCTAACATCA |
| $w_{18,f}$ | = Sf T S18 | = CATTTTTTTTTTTCA TCT CATCTTCTAACATCA |
| $G_{18}$-b | = T* S18* T* | = TG AGA TGATGTTAGAAGATG AGA TG |
| $Th_{34,18:18}$-t | = S18 | = CATCTTCTAACATCA |
| $Th_{34,18:18}$-b | = s34* T* S18* | = TTATGTG AGA TGATGTTAGAAGATG |
| $w_{34,18}$ | = S18 T S34 | = CATCTTCTAACATCA TCT CACATAACAAAACCA |
| $G_{34}$-b | = T* S34* T* | = TG AGA TGGTTTTGTTATGTG AGA TG |
| $w_{27,34}$ | = S34 T S27 | = CACATAACAAAACCA TCT CAAACACTCTATTCA |
| $w_{27,41}$ | = S41 T S27 | = CAACAAACCATTACA TCT CAAACACTCTATTCA |
| $w_{27,20}$ | = S20 T S27 | = CAATCTAACACTCCA TCT CAAACACTCTATTCA |
| $w_{27,f}$ | = Sf T S27 | = CATTTTTTTTTTTCA TCT CAAACACTCTATTCA |
| $G_{27}$-b | = T* S27* T* | = TG AGA TGAATAGAGTGTTTG AGA TG |
| $Th_{40,27:27}$-t | = S27 | = CAAACACTCTATTCA |
| $Th_{40,27:27}$-b | = s40* T* S27* | = TGTATTG AGA TGAATAGAGTGTTTG |
| $w_{40,27}$ | = S27 T S40 | = CAAACACTCTATTCA TCT CAATACAAATCCACA |
| $G_{40}$-b | = T* S40* T* | = TG AGA TGTGGATTTGTATTG AGA TG |
| $w_{28,53}$ | = S53 T S28 | = CATATCTAATCTCCA TCT CATCTACAATTCACA |
| $w_{28,40}$ | = S40 T S28 | = CAATACAAATCCACA TCT CATCTACAATTCACA |
| $w_{28,42}$ | = S42 T S28 | = CACTTTTCACTATCA TCT CATCTACAATTCACA |
| $w_{28,f}$ | = Sf T S28 | = CATTTTTTTTTTTCA TCT CATCTACAATTCACA |
| $G_{28}$-b | = T* S28* T* | = TG AGA TGTGAATTGTAGATG AGA TG |
| $Th_{41,28:28}$-t | = S28 | = CATCTACAATTCACA |
| $Th_{41,28:28}$-b | = s41* T* S28* | = TTTGTTG AGA TGTGAATTGTAGATG |
| $w_{41,28}$ | = S28 T S41 | = CATCTACAATTCACA TCT CAACAAACCATTACA |
| $G_{41}$-b | = T* S41* T* | = TG AGA TGTAATGGTTTGTTG AGA TG |
| $w_{29,34}$ | = S34 T S29 | = CACATAACAAAACCA TCT CACCAATACTCCTCA |
| $w_{29,10}$ | = S10 T S29 | = CATACAACATCTACA TCT CACCAATACTCCTCA |
| $w_{29,41}$ | = S41 T S29 | = CAACAAACCATTACA TCT CACCAATACTCCTCA |
| $w_{29,f}$ | = Sf T S29 | = CATTTTTTTTTTTCA TCT CACCAATACTCCTCA |
| $G_{29}$-b | = T* S29* T* | = TG AGA TGAGGAGTATTGGTG AGA TG |
| $Th_{42,29:29}$-t | = S29 | = CACCAATACTCCTCA |
| $Th_{42,29:29}$-b | = s42* T* S29* | = AAAAGTG AGA TGAGGAGTATTGGTG |
| $w_{42,29}$ | = S29 T S42 | = CACCAATACTCCTCA TCT CACTTTTCACTATCA |
| $G_{42}$-b | = T* S42* T* | = TG AGA TGATAGTGAAAAGTG AGA TG |

Table S7: Hopfield associative memory sequences (part three). Reporter strands are the same as Table S4. circuit 2.

| Gate strands (IDT unpurified) | | |
|---|---|---|
| $w_{21,6}$ | = S6 T S21 | = CATAACACAATCACA TCT CAACCATACTAAACA |
| $w_{21,f}$ | = Sf T S21 | = CATTTTTTTTTTTCA TCT CAACCATACTAAACA |
| $G_{21}$-b | = T* S21* T* | = TG AGA TGTTTAGTATGGTTG AGA TG |
| $Th_{36,21:21}$-t | = S21 | = CAACCATACTAAACA |
| $Th_{36,21:21}$-b | = s36* T* S21* | = TAGTTTG AGA TGTTTAGTATGGTTG |
| $w_{36,21}$ | = S21 T S36 | = CAACCATACTAAACA TCT CAAACTAAACAACCA |
| $G_{36}$-b | = T* S36* T* | = TG AGA TGGTTGTTTAGTTTG AGA TG |
| $w_{5,36}$ | = S36 T S5 | = CAAACTAAACAACCA TCT CACCACCAAACTTCA |
| $w_{18,36}$ | = S36 T S18 | = CAAACTAAACAACCA TCT CATCTTCTAACATCA |
| $w_{22,23}$ | = S23 T S22 | = CAAATCTTCATCCCA TCT CATTCCTACATTTCA |
| $w_{22,f}$ | = Sf T S22 | = CATTTTTTTTTTTCA TCT CATTCCTACATTTCA |
| $G_{22}$-b | = T* S22* T* | = TG AGA TGAAATGTAGGAATG AGA TG |
| $Th_{39,22:22}$-t | = S22 | = CATTCCTACATTTCA |
| $Th_{39,22:22}$-b | = s39* T* S22* | = TATAGTG AGA TGAAATGTAGGAATG |
| $w_{39,22}$ | = S22 T S39 | = CATTCCTACATTTCA TCT CACTATACACACCCA |
| $G_{39}$-b | = T* S39* T* | = TG AGA TGGGTGTGTATAGTG AGA TG |
| $w_{1,39}$ | = S39 T S1 | = CACTATACACACCCA TCT CATCCATTCCACTCA |
| $w_{27,39}$ | = S39 T S27 | = CACTATACACACCCA TCT CAAACACTCTATTCA |
| $w_{30,24}$ | = S24 T S30 | = CACTATACACACCCA TCT CAAACACTCTATTCA |
| $w_{30,f}$ | = Sf T S30 | = CATTTTTTTTTTTCA TCT CACCATTACAATCCA |
| $G_{30}$-b | = T* S30* T* | = TG AGA TGGATTGTAATGGTG AGA TG |
| $Th_{43,30:30}$-t | = S30 | = CACCATTACAATCCA |
| $Th_{43,30:30}$-b | = s43* T* S30* | = TATGATG AGA TGGATTGTAATGGTG |
| $w_{43,30}$ | = S30 T S43 | = CACCATTACAATCCA TCT CATCATACCTACTCA |
| $G_{43}$-b | = T* S43* T* | = TG AGA TGAGTAGGTATGATG AGA TG |
| $w_{13,43}$ | = S43 T S13 | = CATCATACCTACTCA TCT CACAACTCATTACCA |
| $w_{28,43}$ | = S43 T S28 | = CATCATACCTACTCA TCT CATCTACAATTCACA |
| $w_{31,25}$ | = S25 T S31 | = CAATTCACTCAATCA TCT CAATCCACACTTCCA |
| $w_{31,f}$ | = Sf T S31 | = CATTTTTTTTTTTCA TCT CAATCCACACTTCCA |
| $G_{31}$-b | = T* S31* T* | = TG AGA TGGAAGTGTGGATTG AGA TG |
| $Th_{44,31:31}$-t | = S31 | = CAATCCACACTTCCA |
| $Th_{44,31:31}$-b | = s44* T* S31* | = AGTTTTG AGA TGGAAGTGTGGATTG |
| $w_{44,31}$ | = S31 T S44 | = CAATCCACACTTCCA TCT CAAAACTCTCTCTCA |
| $G_{44}$-b | = T* S44* T* | = TG AGA TGAGAGAGAGTTTTG AGA TG |
| $w_{8,44}$ | = S44 T S8 | = CAAAACTCTCTCTCA TCT CACTAACATACAACA |
| $w_{29,44}$ | = S44 T S29 | = CAAAACTCTCTCTCA TCT CACCAATACTCCTCA |

# References


[1] L. Qian and E. Winfree. A simple DNA gate motif for synthesizing large-scale circuits. *Journal of The Royal Society Interface*. Published online February 4, 2011 [doi: 10.1098/rsif.2010.0729].

[2] L. Qian and E. Winfree. Scaling up digital circuit computation with DNA strand displacement cascades. *Science*, 332:1196–1201, 2011.

[3] K. U. Mir. A restricted genetic alphabet for DNA computing. *DNA based computers 2*, pages 243–246, 1999.

[4] R. S. Braich, N. Chelyapov, C. Johnson, P. W. K. Rothemund, and L. M. Adleman. Solution of a 20-variable 3-SAT problem on a DNA computer. *Science*, 296:499–502, 2002.

[5] D. Faulhammer, A. R. Cukras, R. J. Lipton, and L. F. Landweber. Molecular computation: RNA solutions to chess problems. *Proceedings of the National Academy of Sciences USA*, 97:1385–1389, 2000.

[6] M. Y. Kao, M. Sanghi, and R. Schweller. Randomized fast design of short DNA words. *ACM Transactions on Algorithms (TALG)*, 5:1–24, 2009.

[7] J. D. Puglisi and I. Tinoco, Jr. Absorbance melting curves of RNA. In J. E. Dahlberg and J. N. Abelson, editors, *Methods in Enzymology*, volume 180, pages 304–325, San Diego, 1989. Academic Press.

[8] B. Yurke and A. P. Mills Jr. Using DNA to power nanostructures. *Genetic Programming and Evolvable Machines*, 4:111–122, 2003.

[9] D. Y. Zhang and E. Winfree. Control of DNA strand displacement kinetics using toehold exchange. *Journal of the American Chemical Society*, 131:17303–17314, 2009.

[10] H. Lederman, J. Macdonald, D. Stefanovic, and M. N. Stojanovic. Deoxyribozyme-based three-input logic gates and construction of a molecular full adder. *Biochemistry*, 45:1194–1199, 2006.

[11] M. N. Stojanovic, S. Semova, D. Kolpashchikov, J. Macdonald, C. Morgan, and D. Stefanovic. Deoxyribozyme-based ligase logic gates and their initial circuits. *Journal of the American Chemical Society*, 127:6914–6915, 2005.

[12] J. Elbaz, O. Lioubashevski, F. Wang, F. Remacle, R.D. Levine, and I. Willner. DNA computing circuits using libraries of DNAzyme subunits. *Nature Nanotechnology*, 5:417–422, 2010.

[13] G. Seelig, D. Soloveichik, D.Y. Zhang, and E. Winfree. Enzyme-free nucleic acid logic circuits. *Science*, 314:1585–1588, 2006.

[14] I. Wegener. The complexity of the parity function in unbounded fan-in, unbounded depth circuits. *Theoretical Computer Science*, 85:155–170, 1991.

[15] W. H. Kautz. The realization of symmetric switching functions with linear-input logical elements. *IRE Transactions on Electronic Computers*, 10:371–378, 1961.

[16] J. Hastad. Almost optimal lower bounds for small depth circuits. In *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing*, pages 6–20. ACM, 1986.

[17] S. Muroga. *Threshold Logic and Its Applications*, volume 18. Wiley-Interscience New York, 1971.

[18] K. Y. Siu and J. Bruck. On the power of threshold circuits with small weights. *SIAM Journal on Discrete Mathematics*, 4:423, 1991.

[19] A. A. Razborov. On small depth threshold circuits. In *Proceedings of the 3rd Scandinavian Workshop on Algorithm Theory (SWAT92)*, volume LNCS 621, pages 42–52. Springer, 1992.

[20] E. B. Baum. Building an associative memory vastly larger than the brain. *Science*, 268:583–585, 1995.

[21] J. S. Oliver. Matrix multiplication with DNA. *Journal of Molecular Evolution*, 45:161–167, 1997.

[22] A. P. Mills, B. Yurke, and P. M. Platzman. Article for analog vector algebra computation. *Biosystems*, 52:175–180, 1999.

[23] J. Kim, J. J. Hopfield, and E. Winfree. Neural network computation by in vitro transcriptional circuits. *Advances in Neural Information Processing Systems*, 17:681–688, 2004.

[24] J. Reif, T. LaBean, M. Pirrung, V. Rana, B. Guo, C. Kingsford, and G. Wickham. Experimental construction of very large scale DNA databases with associative search capability. In *DNA Computing*, volume LNCS 2340, pages 231–247. Springer, 2002.

[25] A. P. Mills Jr, M. Turberfield, A. J. Turberfield, B. Yurke, and P. M. Platzman. Experimental aspects of DNA neural network computation. *Soft Computing*, 5:10–18, 2001.

[26] H. W. Lim, S. H. Lee, K. Yang, J. Y. Lee, S. I. Yoo, T. H. Park, and B. T. Zhang. In vitro molecular pattern classification via DNA-based weighted-sum operation. *Biosystems*, 100:1–7, 2010.

[27] J. Kim, K. S. White, and E. Winfree. Construction of an in vitro bistable circuit from synthetic transcriptional switches. *Molecular Systems Biology*, 2:68, 2006.

[28] J. Kim and E. Winfree. Synthetic in vitro transcriptional oscillators. *Molecular Systems Biology*, 7:465, 2011.

[29] F. Hsu. IBM's Deep Blue chess grandmaster chips. *IEEE Micro*, 19:70–81, 1999.

[30] M. N. Stojanovic and D. Stefanovic. A deoxyribozyme-based molecular automaton. *Nature Biotechnology*, 21:1069–1074, 2003.

[31] J. Macdonald, Y. Li, M. Sutovic, H. Lederman, K. Pendri, W. Lu, B.L. Andrews, D. Stefanovic, and M. N. Stojanovic. Medium scale integration of molecular logic gates in an automaton. *Nano Letters*, 6:2598–2603, 2006.

[32] R. Pei, E. Matamoros, M. Liu, D. Stefanovic, and M. N. Stojanovic. Training a molecular automaton to play a game. *Nature Nanotechnology*, 5:773–777, 2010.

[33] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–408, 1958.

[34] M. L. Minsky and S. Papert. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge, MA, 1988.

[35] R. Rojas. *Neural Networks: A Systematic Introduction*. Springer, 1996.

[36] R. McEliece, E. Posner, E. Rodemich, and S. Venkatesh. The capacity of the Hopfield associative memory. *IEEE Transactions on Information Theory*, 33:461–482, 1987.

[37] Y. Owechko. Cascaded-grating holography for artificial neural networks. *Applied Optics*, 32:1380–1398, 1993.

[38] K.K. Likharev. Neuromorphic CMOL circuits. In *Third IEEE Conference on Nanotechnology (IEEE-NANO 2003)*, pages 339–342. IEEE Press, 2003.

[39] J. J. Hopfield. Searching for memories, Sudoku, implicit check bits, and the iterative use of not-always-correct rapid neural computation. *Neural Computation*, 20:1119–1164, 2008.

[40] M. Cook. *Networks of Relations*. PhD thesis, California Institute of Technology, 2005.

[41] M. Karnaugh. The map method for synthesis of combinational logic circuits. *Transactions of the AIEE, pt. I*, 72:593–599, 1953.

[42] `http://www.dna.caltech.edu/SeesawCompiler/`.

[43] A. Phillips and L. Cardelli. A programming language for composable DNA circuits. *Journal of The Royal Society Interface*, 6:S419–S436, 2009.

[44] A. Hjelmfelt and J. Ross. Chemical implementation and thermodynamics of collective neural networks. *Proceedings of the National Academy of Sciences USA*, 89:388–391, 1992.

[45] N. E. Buchler, U. Gerland, and T. Hwa. On schemes of combinatorial transcription logic. *Proceedings of the National Academy of Sciences USA*, 100:5136–5141, 2003.