nature nanotechnology

ARTICLES

https://doi.org/10.1038/s41565-019-0544-5

Fast and compact DNA logic circuits based on single-stranded gates using strand-displacing polymerase

Tianqi Song[1], Abeer Eshra[2], Shalin Shah[3], Hieu Bui[4], Daniel Fu[1], Ming Yang[1], Reem Mokhtar[1] and John Reif[1,3]*

DNA is a reliable biomolecule with which to build molecular computation systems. In particular, DNA logic circuits (diffusion-based) have shown good performance regarding scalability and correctness of computation. However, previous architectures of DNA logic circuits have two limitations. First, the speed of computation is slow, often requiring hours to compute a simple function. Second, the circuits are of high complexity regarding the number of DNA strands. Here, we introduce an architecture of DNA logic circuits based on single-stranded logic gates using strand-displacing DNA polymerase. The logic gates consist of only single DNA strands, which largely reduces leakage reactions and signal restoration steps such that the circuits are improved in regard to both speed of computation and the number of DNA strands needed. Large-scale logic circuits can be constructed from the gates by simple cascading strategies. In particular, we have demonstrated a fast and compact logic circuit that computes the square-root function of four-bit input numbers.

Deoxyribonucleic acid is a highly programmable biomolecule because of its precise base-pairing property[1], which makes it possible to construct DNA circuits for complex tasks. To date, DNA circuits have been developed for multiple purposes[2–6]. In particular, considerable progress has been made on (diffusion-based) DNA logic circuits[2,7,8]. Previous work has demonstrated the capability of DNA logic circuits regarding scalability and accuracy of computation[2,7,8]. However, these architectures of DNA logic circuits have limitations that need to be addressed if we want DNA computation of higher efficiency. First, the speed of computation is slow and it can take hours for a circuit to respond (half-completion time). Second, the 'strand complexity' (number of DNA strands) of the circuits is high. For example, for a DNA logic circuit to compute the square-root function of four-bit input numbers by an existing architecture[2], it requires hours for this circuit (of >100 DNA strands) to respond for some inputs. Recently, some effort has been made to speed up DNA logic circuits, including the use of localized DNA circuits[4,9] and leakless strand displacement[10,11]. However, both methods require non-trivial additional work on DNA origami or gates, which increases design, operation and strand complexity.

To mitigate the limitations detailed above, an architecture requires certain features. First, the logic gates should have very low leakage (unintended reactions) such that the number of signal restoration steps is minimized, since frequent signal restoration is a major reason for slow speed of computation and high 'strand complexity'[2]. Second, logic gates with DNA complexes of multiple DNA strands should be avoided, because such DNA complexes increase 'strand complexity' and are more prone to leakage caused by imperfect sequence design and purification compared to simple DNA structures (for example, single DNA strand).

Here, we introduce an architecture of DNA logic circuits that has both of the above features and that utilizes strand-displacing DNA polymerase. There is a long history of enzyme use in DNA circuits. Early papers are insightful and are mainly focused on the theoretical exploration of computation by DNA using enzymes[12–14], where the designs are quite elegant for theoretical usage but need to be simplified for experimental demonstration because of the use of multiple enzymes or complex operations. A recently introduced toolbox, called PEN DNA, is utilized for programming of DNA circuits[15] using multiple enzymes, and several multi-enzyme DNA circuits have been developed based on it[16–20]. This provides motivation for the design of our architecture, but the main difference is that our circuits use only one enzyme—DNA polymerase—and this helps to avoid the additional design complexity, potential slow-down by cleavage of DNA and leakage from catalytic behaviours in multi-enzyme DNA circuits. Primer exchange reaction[21] is a technique developed for automatic synthesis of single-stranded DNA using DNA hairpins. It can also be used to construct DNA logic circuits, although its performance regarding the speed of computation needs more detailed investigation for large-scale, complex logic circuits.

Specifically, our architecture is based on logic OR (disjunction) and AND (conjunction) gates using Bst 2.0 DNA polymerase. Each logic gate consists initially of only single DNA strands, to reduce the leakage and 'strand complexity' caused by prefabricated DNA complexes in previous architectures[2,7]. The reason why the gates can be simplified to such extent is the use of DNA polymerase. The logic gates can be easily cascaded to form large-scale circuits, and each logic gate was first tested separately. To demonstrate the scalability of our architecture, multiple two- and three-layer linear cascades were implemented. Also, fan-in and -out functions were tested to show that our architecture was able to support diverse circuit layouts. Finally, we demonstrated a large-scale circuit that computes (the floor of) the square-root function of four-bit input numbers. This responds in minutes (half-completion time) and

[1]Department of Computer Science, Duke University, Durham, NC, USA. [2]Department of Computer Science and Engineering, Menoufia University, Menouf, Egypt. [3]Department of Electrical and Computer Engineering, Duke University, Durham, NC, USA. [4]National Research Council, Washington, DC, USA. *e-mail: reif@cs.duke.edu

comprises about 37 DNA strands while, on the other hand, even if implemented by the best previous architecture[2], response would be measured in hours and it would comprise around 100 DNA strands.

Single-stranded DNA logic gates

Logic OR (disjunction) and AND (conjunction) gates are the foundation of our architecture. Using dual-rail logic, any Boolean logic function can be evaluated by a circuit made from these two gates[2].

The OR gate has two inputs (A, B) and an output (O) (Fig. 1a), where each is encoded by the high (logic '1') or low (logic '0') concentration of a corresponding DNA strand. Each DNA domain in the OR gate has 30 nucleotides (nt). The OR gate consists of two DNA strands ($O^*F^*A^*$ and $O^*F^*B^*$), where we use the sequence of domain names (5′–3′) as a strand name. To function, the OR gate also needs fuel strand F. The OR gate works as shown in Fig. 1b,c: fuel strand F is first added to the reaction solution to prepare the gate before addition of the input strands. F binds to $O^*F^*A^*$ or $O^*F^*B^*$ by DNA hybridization, and then initiates DNA polymerization to form an intermediate DNA complex ($O^*F^*A^*$:FO or $O^*F^*B^*$:FO, respectively) using Bst 2.0 DNA polymerase (Fig. 1b). In this paper, we combine the names of the DNA strands to give the name of the corresponding DNA complex. For example, the name of the DNA complex comprising $O^*F^*A^*$ and FO is $O^*F^*A^*$:FO. DNA complex $O^*F^*A^*$:FO or $O^*F^*B^*$:FO takes in an input strand A or B, respectively, by DNA hybridization and then produces strand FO by polymerase-based strand displacement (Fig. 1c), where the O domain in strand FO represents the output of the OR gate. Only when either input strand exists does the gate produce the output strand, which is an OR function. The OR gate will also work correctly to produce output strand FO when an input strand and a fuel strand simultaneously hybridize to a gate strand and start polymerization. This is because the input and fuel strands have the same length and sequence design criteria, and thus they should have very similar rates of DNA hybridization and polymerization. The probability that the fuel strand is displaced before its extension should be very low.

The OR gate was tested under all possible input combinations, and it gave the correct outputs in all scenarios (Fig. 1a). Each strand of the OR gate was at 1× (relative) concentration, where 1× was 100 nM. Fuel strand F was at 2× concentration, where each gate strand required 1×. If an input was logic '1', the corresponding DNA strand was at 1× concentration; otherwise, no corresponding DNA strand existed. The gate computed the output by a half-completion time of only ~3 min. To report the computation result, we let the output strand of the OR gate react with a reporter complex (Fig. 1d). The reporter complex takes in an input strand O and generates fluorescence. Specifically, the fluorophore (red star) is quenched by a quencher (black spot) in the reporter complex. When the input strand O arrives, it hybridizes to the reporter complex and initiates polymerase-based strand displacement to knock off the R strand such that the fluorophore is no longer quenched, thus generating fluorescence. Let the O domain be identical to the output domain of a gate, and then the reporter can be used to monitor the output of that gate. Note that the bottom strand of the reporter may also be extended by polymerase during the reporting in some cases, but this kind of reaction does not influence the performance of a gate or reporter. However, in the interests of conciseness, we do not show this in Fig. 1. It can be prevented simply by the addition of a poly-T domain to the 3′ end of the bottom strand such that it cannot be extended by polymerase[21]. The experiments were conducted using Bst 2.0 DNA polymerase at 55 °C. In the experiments, all reactants (input strands, fuel strands, gate strands, deoxyribonucleoside triphosphates and so on) except the polymerase were first mixed and placed in a cuvette for warming to 55 °C in a fluorescence spectrophotometer for about 5 min; the polymerase was then introduced to start the polymerization reaction. The details of experimental and sequence design are included in Methods and Supplementary Tables 1–13.

The AND gate has two inputs (A, B) (Fig. 1e) and an output (O), where each is encoded by either the high (logic '1') or low (logic '0') concentration of a corresponding DNA strand. The AND gate consists of two DNA strands, $O^*B^*A^*$ and $O^*A^*B^*$. Each DNA domain in the AND gate has 30 nt. The AND gate operates by two possible reaction pathways to produce the output strand. In the first pathway (Fig. 1f), input strand A first hybridizes to strand $O^*A^*B^*$ and initiates polymerization to form DNA complex $O^*A^*B^*$:AO, then input strand B hybridizes to DNA complex $O^*A^*B^*$:AO and initiates polymerase-based strand displacement to produce output strand AO, where the O domain in strand AO represents the output of the AND gate. In the second pathway (Fig. 1g), input strand B first hybridizes to strand $O^*B^*A^*$ and initiates polymerization to form DNA complex $O^*B^*A^*$:BO, then input strand A hybridizes to DNA complex $O^*B^*A^*$:BO and initiates polymerase-based strand displacement to produce output strand BO, where the O domain in strand BO represents the output of the AND gate. Only when both input strands exist does the gate produce an output strand, which is an AND function. The AND gate will also work correctly to produce output strand AO or BO when input strands A and B simultaneously hybridize to a gate strand and initiate polymerization. The AND gate was tested under all possible input combinations (Fig. 1e), and it gave the correct outputs in all scenarios. Each gate strand of the AND gate was at 1× (relative) concentration, where 1× was 100 nM. For inputs, if an input was logic '1', the corresponding DNA strand was at 2× concentration (1× for each gate strand). Otherwise, no corresponding DNA strand existed. The gate computed the output by a half-completion time of only ~3 min.

Side-reactions can happen in the AND gate (Fig. 1h). Input A can hybridize to strand $O^*B^*A^*$ and initiate DNA polymerization to form DNA complex $O^*B^*A^*$:ABO, such that no site suitable for hybridization is available to input strand B, and then strand $O^*B^*A^*$ is consumed without generating any output strand. Also, input strand B can hybridize to strand $O^*A^*B^*$ and initiate DNA polymerization to form DNA complex $O^*A^*B^*$:BAO, such that no site for hybridization is available for input strand A, and then strand $O^*A^*B^*$ is wasted. This would be an issue when we cascaded the gates into a circuit. The worst-case scenario occurs when one of the input strands appears much earlier than the other, and then one of the gate strands will be totally wasted. The output of the AND gate will also be compromised when the input strands are insufficient. For example, if we use 1× concentration to encode logic '1' for an input (rather than 2×) in the case of A = 1 and B = 1, each gate strand has a possibility of only $0.5 \times 0.5 = 0.25$ to acquire both input strands and generate the output. Hence, the total output expected is only 0.5× concentration, which means that half of the input strands are wasted.

Note that for neither gate is the signal of logic '0' detectable, indicating that both have low leakage. Because of this low leakage, our architecture does not need to perform a signal restoration for each logic gate, which can dramatically improve the speed of computation and 'strand complexity' of logic circuits. Further discussion on signal restoration in our architecture is provided in Supplementary Section 1.

Cascade gates into circuits

The logic gates in our architecture are modular, since both the input and output strands have the same domain motif (30 nt in length). Therefore, we can cascade the logic gates into circuits. Specifically, the output domain O of an upstream gate can be designed to be identical to an input strand of a downstream gate to serve as its input.

To show the scalability of our architecture, two-layer cascades OR–AND (upstream to downstream) and AND–OR were tested as in Fig. 2a (left and right, respectively). Both cascades produced correct results for all representative input combinations. For the

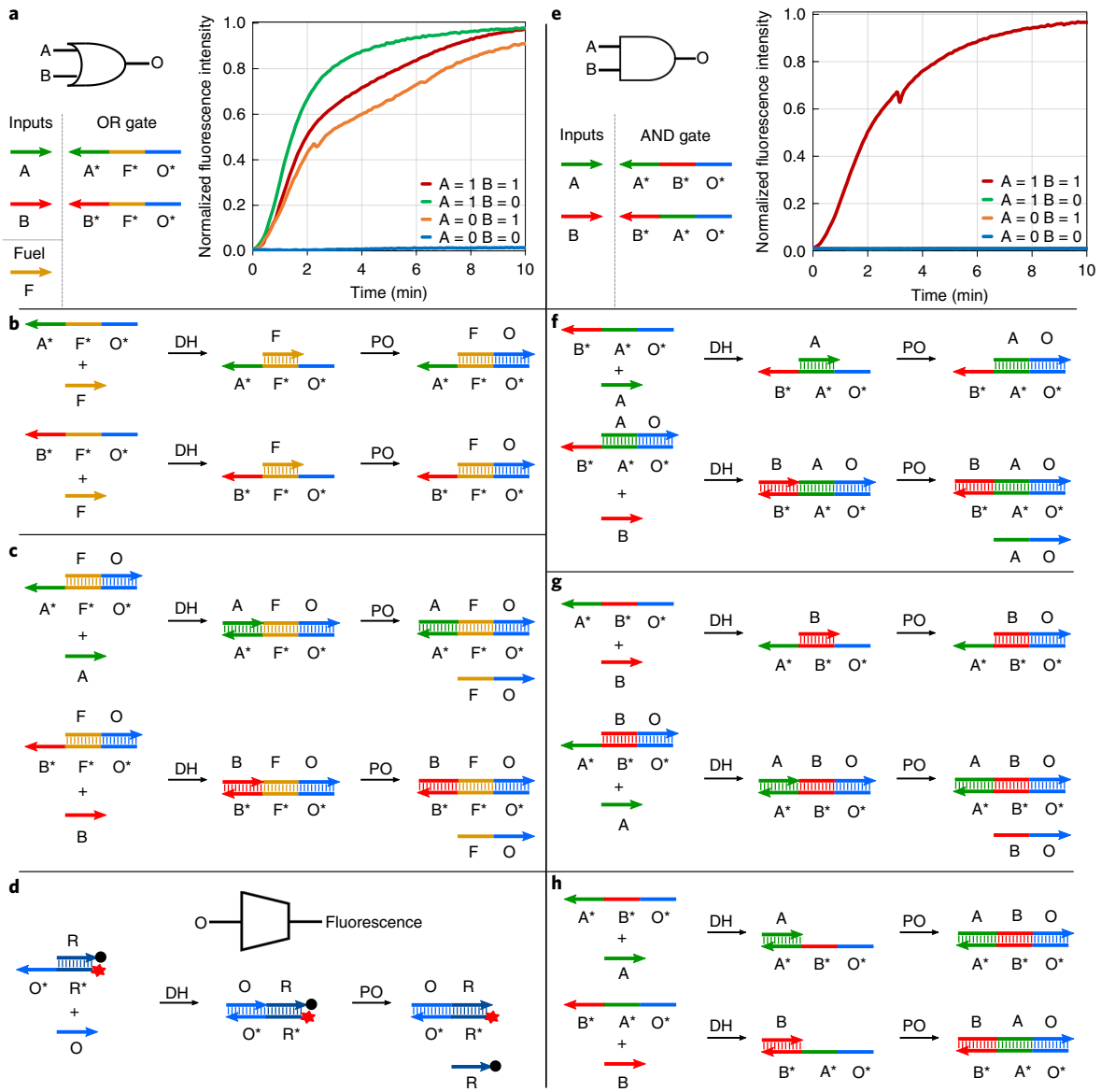

Fig. 1 | Design and performance of the OR and AND gates. a, Abstraction, domain design and experimental results for the OR gate. **b**,**c**, DNA hybridization (DH) and polymerization (PO) reactions in the OR gate. In the experiments, DNA strands O*F*A* and O*F*B* were at 1×, where 1× relative concentration was 100 nM. Fuel F was at 2× while inputs A and B were at either 1× (logic '1') or 0× (logic '0'). **d**, Abstraction, domain design and reactions of a reporter complex. Red stars and black spots indicate fluorophores and quenchers, respectively. **e**, Abstraction, domain design and experimental results for the AND gate. **f**,**g**, Two potential reaction pathways for generation of output strand in the AND gate. In the experiments, DNA strands O*B*A* and O*A*B* were at 1×, where 1× relative concentration was 100 nM. Inputs A and B were at either 2× (logic '1', because each input strand is needed by both gate strands) or 0× (logic '0'). **h**, Side-reactions in the AND gate where the gate strands were consumed without generating any output strand. The output of each gate was monitored by fluorescence intensity.

OR–AND cascade, each gate strand of the AND gate was at 1×, where 1× was 100 nM. Each gate strand of the OR gate was at 2×, to ensure that the AND gate could acquire sufficient input strands (2×) from the OR gate even if only one of those input strands existed. For the AND–OR cascade, each gate strand of the OR gate was at 1× while each gate strand of the AND gate was at 1×, ensuring that the OR gate could acquire sufficient input strands (1×) from the AND gate, even in the worst-case scenario when one of the input strands of the AND gate appeared much later than the other and only one of the gate strands of the AND gate produced output. The essential strategy was that the concentration of a logic gate was determined such that this gate could always generate sufficient output strands (if needed) for its downstream gates. This strategy for determination of concentrations in cascades was applied to all circuits. Three-layer cascades, AND–OR–AND and OR–AND–OR, were also tested and these gave correct results for all representative input combinations (Fig. 2b).

The OR gate supports fan-in function. To accept *n* inputs, an OR gate needs to have *n* gate strands and the design strategy follows the two-input OR gate. Fan-in function based on a four-input OR gate was tested (Fig. 3a). As designed, the inputs of the downstream OR gate could be derived from any of four possible upstream gates, and this gate gave an output of logic '1' only when one of the upstream gates produced an output of logic '1'.

Fan-out function was also demonstrated (Fig. 3b), where the output of an upstream OR gate served as the inputs to four

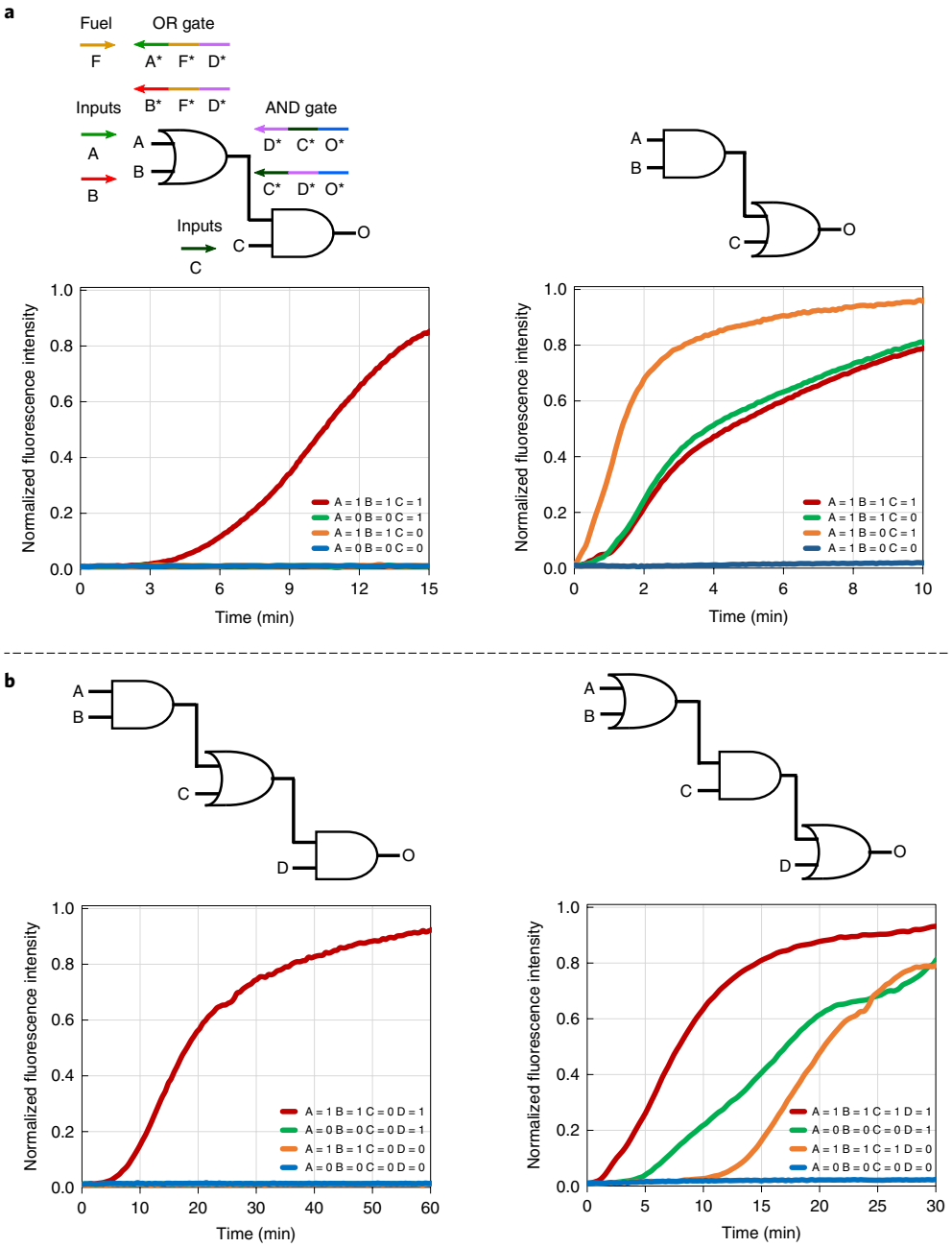

Fig. 2 | Two- and three-layer cascades. a, Two-layer cascades OR–AND (with domain-level design as an example) and AND–OR. **b**, Three-layer cascades AND–OR–AND and OR–AND–OR. The output of each cascade was monitored by fluorescence intensity. Details of concentrations used are included in Supplementary Fig. 2.

downstream OR gates. As designed, when one of the input strands was presented, the upstream OR gate produced output strands and all downstream OR gates gave an output of logic '1'. Each gate strand of each downstream OR gate was at 1×, where 1× was 100 nM. To ensure that the upstream OR gate could produce sufficient output strands for all downstream OR gates in all scenarios, each gate strand of the upstream OR gate was at 4×, as was each input strand of the upstream OR gate, if they existed.

A moderate-scale circuit of four gates was then tested (Fig. 3c), and it gave correct results for all representative input combinations. All the circuits described above computed their outputs in a half-completion time of about 25 min at most. Although the concentration strategy used for cascades can influence the scalability of this architecture, we were still able to construct large-scale logic circuits such as one that computed (the floor of) the square-root function of four-bit input numbers, described in the following section. Note that for all circuits tested above, the goal was to demonstrate the basic ways of organizing logic gates by showing that the output of an upstream gate could serve as the inputs of its downstream gates in different layouts, as in a previous work[2], and that there was no need to test all possible input combinations.

A circuit for computation of square-root

To show that our architecture is suitable for large-scale circuits, we demonstrated a circuit made from ten gates to compute (the floor of) the square-root function of four-bit numbers where F1F0 $= \lfloor\sqrt{\mathrm{DCBA}}\rfloor$ (Fig. 4a). Since there is no NOT gate in our architecture, the original circuit must be transformed to its dual-rail logic form (Fig. 4b)[2].

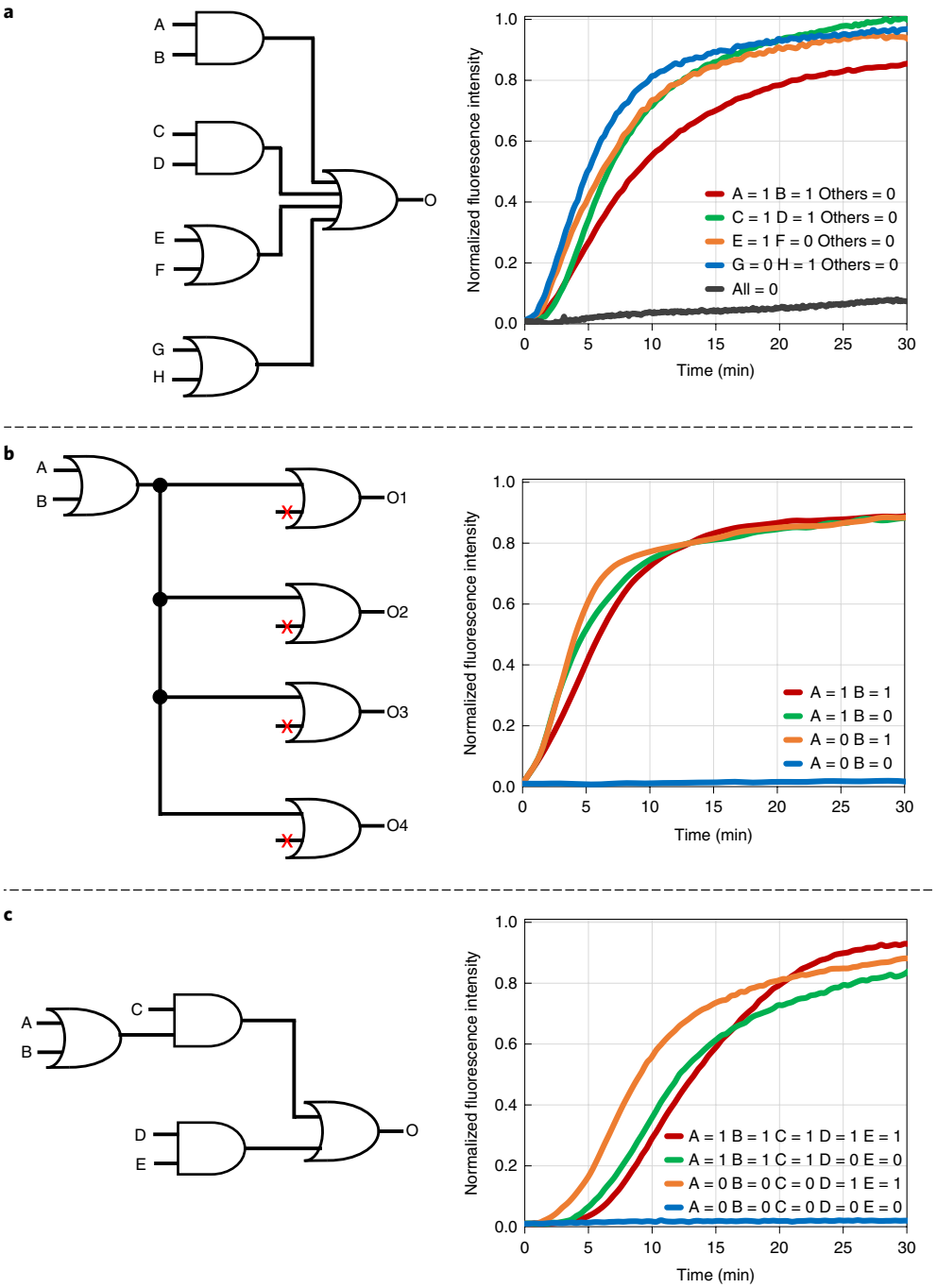

Fig. 3 | Fan-in, fan-out and moderate-scale circuits. a, Fan-in by a four-input OR gate. **b**, Fan-out, where the output of an OR gate is directed to four downstream OR gates. The four outputs were monitored simultaneously by four different types of reporter. The fluorescence result here is for output O1; fluorescence results for outputs O2, O3 and O4 are included in Supplementary Fig. 3. **c**, A moderate-scale circuit of four gates. The output of each circuit was monitored by fluorescence intensity.

By dual-rail logic, each Boolean variable in the original circuit is represented by two variables in the transformed circuit. For example, variable A is represented by variables A-1 and A-0, where we have A-1 = 1, A-0 = 0 when A = 1, and A-1 = 0, A-0 = 1 when A = 0. Specifically, A-1 is responsible for the case of A = 1, where we have A-1 = 1. A-0 is responsible in the case of A = 0, where we have A-0 = 1. A-1 and A-0 cannot both be 1 or 0 simultaneously.

All 16 possible input combinations were tested, and the circuit always gave correct outputs. Figure 4c shows the fluorescence outputs of four representative input combinations. To explain the result, using the case of input = 0000 as an example, the result was F0-0 = 1, F0-1 = 0, F1-0 = 1, F1-1 = 0, which means that F0 = 0 and F1 = 0 (as expected). The experimental results of all input combinations are shown in Supplementary Figs. 4–7. The circuit computed the outputs in a half-completion time of ~25 min at most for all input combinations.

Here, we might want to compare our square-root circuit to the previous one based on seesaw gates[2]. It is difficult to provide a direct comparison because the circuits have different layouts. We did not use the same layout because our architecture did not support the

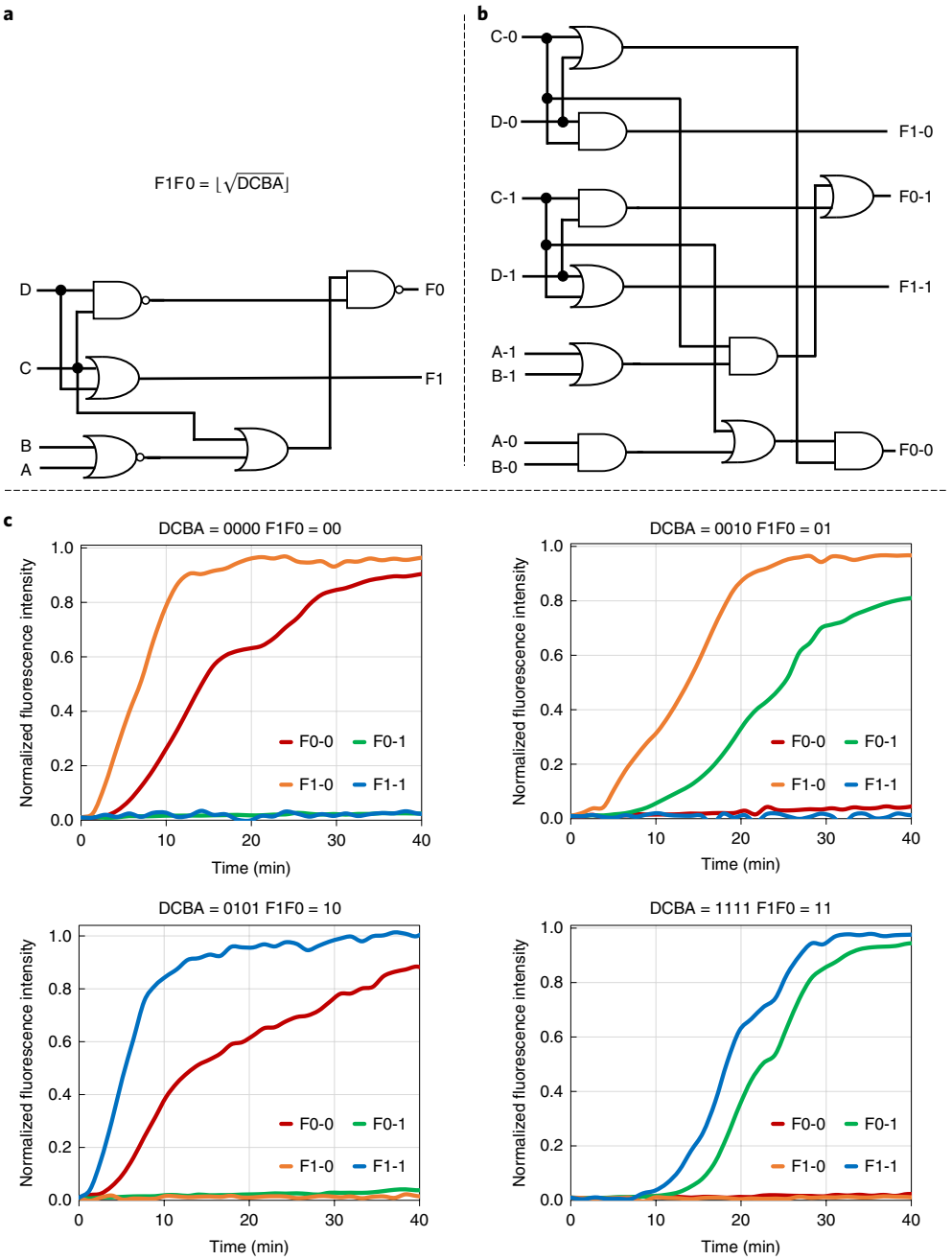

Fig. 4 | A square-root circuit. a, The logic circuit used to compute F1F0 = $\lfloor\sqrt{\mathrm{DCBA}}\rfloor$. **b**, The logic circuit transformed by dual-rail logic. Each logic variable in the original circuit is represented by two variables in the transformed circuit. For example, variable A is represented by variables A-1 and A-0, where we have A-1 = 1, A-0 = 0 when A = 1, and A-1 = 0, A-0 = 1 when A = 0. **c**, The experimental results of four representative input combinations. F0-0, F0-1, F1-0 and F1-1 were monitored simultaneously by four different types of reporter. Further results are given in Supplementary Figs. 4–7.

three-input AND gate used in ref. [2]. However, a square-root circuit based on seesaw gates using our layout will still respond in hours (half-completion time) in some cases, and have around 100 DNA strands. This can be estimated from the circuit layout.

Our square-root circuit responds in about 25 min for all cases and comprises ~37 DNA strands. Another aspect regarding the comparison is concentration. Since no thresholding is needed in our architecture, the logic gates here respond more rapidly than those based on seesaw gates at the same concentration level. For example, the OR gate in Fig. 1 was tested at a concentration of 100 nM for each gate strand (as in ref. [2]), but the computation rate of our OR gate was still much faster.

It is unfair to claim that our architecture is better, because the two architectures discussed above focus on different aspects of computation. Our architecture focuses on improving the speed of computation and 'strand complexity'. We designed the logic gates to have only single DNA strands such that they are of low leakage and require a minimal number of DNA strands. Low leakage allows our architecture to contain few signal restoration steps, which accelerates computation and also reduces the number of DNA strands required. For architecture based on seesaw gates[2], this is focused mainly on scalability. Compared to our single-stranded motif, the seesaw gate motif is catalytic and leakier but is also more scalable, through the incorporation of a signal restoration step for each gate.

Conclusions

In this paper, we present an architecture for fast and compact DNA logic circuits based on Bst 2.0 DNA polymerase. The logic gates in our architecture are designed to be single-stranded, to minimize the leakage and number of DNA strands. The domain motif of our logic gates is modular and convenient for cascading. Fan-in and -out functions are also supported in our architecture. Experimental demonstrations of single gates and circuits of various layouts were successful. In particular, we demonstrated a large-scale circuit to compute (the floor of) the square-root function of four-bit input numbers. The computation took around 25 min at most (half-completion time), and the circuit had about 37 DNA strands.

The DNA logic circuits in our architecture may be used for simultaneous sensing of multiple nucleic acid sequences in medical applications, but potentially with a shorter response time compared to circuits having multiple or no enzymes with previous architectures. For such applications, the gate strands may be modified to prevent degradation by enzymes from patient samples. For future applications of DNA logic circuits, the choice of architecture will depend on the practical requirements of an application. For example, enzyme-free circuits may be preferred when the application cannot fulfill the environments required by certain enzymes.

Compared to previous architectures, our architecture has several advantages including simpler design, lower leakage and shorter response time. We concede, however, that DNA circuits using multiple enzymes or multi-strand DNA structures (but no enzyme) may have advantages in several scenarios, such as building complex dynamical systems.

Online content

Any methods, additional references, Nature Research reporting summaries, source data, statements of code and data availability and associated accession codes are available at https://doi.org/10.1038/s41565-019-0544-5.

Received: 13 February 2019; Accepted: 9 August 2019;
Published online: 23 September 2019

References

1. Watson, J. D. & Crick, F. H. Molecular structure of nucleic acids. *Nature* **171**, 737–738 (1953).
2. Qian, L. & Winfree, E. Scaling up digital circuit computation with DNA strand displacement cascades. *Science* **332**, 1196–1201 (2011).
3. Qian, L., Winfree, E. & Bruck, J. Neural network computation with DNA strand displacement cascades. *Nature* **475**, 368–372 (2011).
4. Chatterjee, G., Dalchau, N., Muscat, R. A., Phillips, A. & Seelig, G. A spatially localized architecture for fast and modular DNA computing. *Nat. Nanotechnol.* **12**, 920 (2017).
5. Srinivas, N., Parkin, J., Seelig, G., Winfree, E. & Soloveichik, D. Enzyme-free nucleic acid dynamical systems. *Science* **358**, eaal2052 (2017).
6. Cherry, K. M. & Qian, L. Scaling up molecular pattern recognition with DNA-based winner-take-all neural networks. *Nature* **559**, 370 (2018).
7. Seelig, G., Soloveichik, D., Zhang, D. Y. & Winfree, E. Enzyme-free nucleic acid logic circuits. *Science* **314**, 1585–1588 (2006).
8. Thubagere, A. J. et al. Compiler-aided systematic construction of large-scale DNA strand displacement circuits using unpurified components. *Nat. Commun.* **8**, 14373 (2017).
9. Bui, H. et al. Localized DNA hybridization chain reactions on DNA origami. *ACS Nano* **12**, 1146–1155 (2018).
10. Wang, B., Thachuk, C., Ellington, A. D., Winfree, E. & Soloveichik, D. Effective design principles for leakless strand displacement systems. *Proc. Natl Acad. Sci. USA* **115**, E12182–E12191 (2018).
11. Thachuk, C., Winfree, E. & Soloveichik, D. in *International Workshop on DNA-Based Computers* Vol. 9211 (eds Phillips, A. & Yin, P.) 133–153 (Springer, 2015).
12. Boneh, D., Dunworth, C., Lipton, R. J. & Sgall, J. On the computational power of DNA. *Discrete Appl. Math.* **71**, 79–94 (1996).
13. Ogihara, M. & Ray, A. Simulating boolean circuits on a DNA computer. *Algorithmica* **25**, 239–250 (1999).
14. Winfree, E. *Whiplash PCR for O(1) Computing* (California Institute of Technology, 1998).
15. Baccouche, A., Montagne, K., Padirac, A., Fujii, T. & Rondelez, Y. Dynamic DNA-toolbox reaction circuits: a walkthrough. *Methods* **67**, 234–249 (2014).
16. Montagne, K., Plasson, R., Sakai, Y., Fujii, T. & Rondelez, Y. Programming an in vitro DNA oscillator using a molecular networking strategy. *Mol. Syst. Biol.* **7**, 466 (2011).
17. Padirac, A., Fujii, T. & Rondelez, Y. Bottom-up construction of in vitro switchable memories. *Proc. Natl Acad. Sci. USA* **109**, E3212–E3220 (2012).
18. Aubert, N., Mosca, C., Fujii, T., Hagiya, M. & Rondelez, Y. Computer-assisted design for scaling up systems based on DNA reaction networks. *J. R. Soc. Interface* **11**, 20131167 (2014).
19. van Roekel, H. W. et al. Programmable chemical reaction networks: emulating regulatory functions in living cells using a bottom-up approach. *Chem. Soc. Rev.* **44**, 7465–7483 (2015).
20. Yordanov, B. et al. Computational design of nucleic acid feedback control circuits. *ACS Synth. Biol.* **3**, 600–616 (2014).
21. Kishi, J. Y., Schaus, T. E., Gopalkrishnan, N., Xuan, F. & Yin, P. Programmable autonomous synthesis of single-stranded DNA. *Nat. Chem.* **10**, 155 (2018).

Acknowledgements

This work is supported by NSF Grants nos. CCF-1320360, CCF-1217457, CCF-1617791 and CCF-1813805.

Author contributions

T.S. and J.R. designed the logic gates and circuits. T.S. designed and performed the experiments. T.S. analysed the data. T.S. and J.R. wrote the paper. A.E., S.S., H.B., D.F., M.Y. and R.M. contributed to creating the logic gates and circuits. All authors discussed the design and results of the experiments. All authors commented on the manuscript.

Competing interests

The authors declare no competing interests.

Additional information

Supplementary information is available for this paper at https://doi.org/10.1038/s41565-019-0544-5.

Reprints and permissions information is available at www.nature.com/reprints.

Correspondence and requests for materials should be addressed to J.R.

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Methods

Sequence design and DNA preparation. Each domain has 30 nt. We used the software to design seesaw gate circuits[2] to generate a large set of orthogonal DNA sequences of 15 nt, because the sequence design principles utilized are suitable for the design of PCR primers (about 40% guanine-cytosine content, minimal secondary structures). The software can be found at http://qianlab.caltech.edu/SeesawCompiler/. Each 30-nt domain was created by combining two such 15-nt sequences, with additional checking to exclude secondary structures. The domain length of 30 nt was chosen because it is safe for stable primer binding at the temperature selected in our experiments. The DNA strands were purchased from Integrated DNA Technologies, without purification. We purified the strands by polyacrylamide gel electrophoresis using 10% denaturing PAGE. Electrophoresis was conducted in 1× TBE buffer under 300 V for 90 min.

Polymerase reaction. Bst 2.0 DNA polymerase and deoxynucleotide solution mix were purchased from New England Biolabs. The polymerase reactions were conducted at 55 °C in cuvettes in a fluorescence spectrophotometer. The reaction buffer was 1× isothermal amplification buffer provided by New England Biolabs. Ten units of Bst 2.0 DNA polymerase and 1 mM of each type of deoxynucleotide were used in all experiments. For each experiment, all DNA strands and chemicals, except the polymerase, were mixed well (78 μl), added to a cuvette and placed in a spectrophotometer for warming to 55 °C for about 5 min. Two microlitres of polymerase (ten units) was then added to the cuvette and mixed to start polymerization reactions. While adding the polymerase, the cuvette remained in the spectrophotometer to maintain a constant temperature. Also, the process was completed in only several seconds to avoid temperature fluctuation. The 2 μl of polymerase was initially at room temperature, but no significant temperature fluctuation was likely because of its tiny volume compared to the whole reaction solution.

Fluorescence experiments and data normalization. A Varian Cary Eclipse Fluorescence Spectrophotometer was used for fluorescence experiments. Four types of fluorophore were used in the reporters: 6-FAM (excitation 495 nm, emission 520 nm), ROX (excitation 588 nm, emission 608 nm), TYE 563 (excitation 549 nm, emission 563 nm) and TYE 665 (excitation 645 nm, emission 665 nm). Existing research shows that these fluorophores do not interfere with each other[2].

Each fluorescence curve, representing an output of logic '1', was normalized using its minimum value as 0 and its maximum value (average of last three data points) as 1. The curves of logic '1' had to be normalized separately to identify the half-completion time of each curve, since our circuits were not catalytic and the curves of logic '1' had different maximum values even when using the same reporter complex. This was different to previous enzyme-free catalytic DNA circuits, where all curves of logic '1' have almost the same maximum value when using the same reporter complex. From a digital perspective, this did not impede the performance of our circuits because the maximum values of all curves of logic '1' were quite different from those of the corresponding curves of logic '0'.

For each curve representing an output of logic '0', because the increase in fluorescence was almost zero during the experiment we could not use its maximum value as 1 for normalization. Instead, we chose another curve representing an output of logic '1' for the same circuit (also the same reporter and cuvette), and took the maximum value of this curve as 1 to normalize that curve representing an output of logic '0'. If multiple such curves were available, the one whose maximum value was the lowest was used to avoid the improper suppression of a curve of logic '0' caused by normalization, which gave a fair comparison between the curves of logic '1' and those of logic '0'. Fluorescence data in the warm-up period before addition of the polymerase are not shown for each individual curve.

Data availability

The data used in this paper are available from the corresponding author upon proper request.