# DNA Simulation of Boolean Circuits*

Martyn Amos† Paul E. Dunne
Department of Computer Science, University of Liverpool, Liverpool L69 3BX, England

**Abstract**

In this paper we describe a simulation of Boolean circuits using standard bio-molecular techniques. Previously proposed simulations have been shown to run in time proportional to the size of the circuit. The simulation we present here runs in time proportional to the *depth* of the circuit. We describe the abstract model and its laboratory implementation, before concluding with a brief analysis.

## 1. Introduction

Since the publication of Adleman's original paper [1], several authors have described how various models of computation may be simulated using bio-molecular methods. Such models include Turing Machines [14, 18], splicing systems [6, 9] and Boolean circuits [12]. It is the last of these models that this paper concentrates on.

Boolean circuits are an important Turing-equivalent model of parallel computation (see [7, 8]). An $n$-input *bounded fan-in* Boolean circuit may be viewed as a directed, acyclic graph, $S$, with two types of node: $n$ *input* nodes with in-degree (i.e., input lines) zero, and *gate* nodes with maximum in-degree two. Each input node is associated with a unique Boolean variable $x_i$ from the input set $X_n = (x_1, x_2, \ldots, x_n)$. Each gate node, $g_i$ is associated with some Boolean function $f_i \in \Omega$. We refer to $\Omega$ as the circuit *basis*. A *complete* basis is a set of functions that are able to express all possible Boolean functions.

The first DNA-based simulation of Boolean circuits was described by Ogihara and Ray in [12]. In this paper the authors claim real-time simulation of the class $NC^1$ [13] in time proportional to the depth of the circuit. Recall that $NC^1$ defines the class of problems of size $n$ solved by bounded fan-in circuits of $O(\log n)$ depth and polynomial size. In [3], we showed that, in practical terms, this estimate of the time complexity should be proportional to the *size* of the circuit. We present a rather more detailed analysis in Section 4.

In this paper we describe a DNA simulation of Boolean circuits that runs in time proportional to the *depth* of the circuit. Apart from the reduced running time, our model has the advantage of being much easier to implement in the laboratory than those previously described.

The rest of this paper is organized as follows. In Section 2., we describe our abstract model of Boolean circuit simulation. We then show in Section 3. how the

*Report number CTAG-97009
†Partially supported by a Leverhulme Special Research Fellowship. Email:martyn@csc.liv.ac.uk

model may be implemented in the laboratory. We then present an analysis of the model in Section 4., and conclude with a few remarks and suggestions for further work.

## 2. Our simulation

Since it is well-known [7, 8, 19] that the NAND function provides a complete basis by itself, we restrict our model to the simulation of such gates. In fact, the realisation in DNA of this basis provides a far less complicated simulation than using other complete bases. It is interesting to observe that the fact that NAND offers the most suitable basis for Boolean network simulation within DNA computation continues the traditional use of this basis as a fundamental component within new technologies. Thus, from the work of Sheffer [17] that established the completeness of NAND with respect to propositional logic, through classical gate-level design techniques [8], and, continuing, in the present day, with VLSI technologies both in nMOS [11], and CMOS [20, pp. 9–10].

The simulation proceeds as follows. An $n$-input, $m$-output Boolean network is modelled as a directed acyclic graph, $S(V, E)$, in which the set of vertices $V$ is formed from two disjoint sets: $X_n$, the *inputs* of the network (of which there are exactly $n$); and $G$, the *gates* (of which exactly $m$ are distinguished as output gates). Each input vertex has in-degree 0 and is associated with a single Boolean variable, $x_i$. Each gate has in-degree 2 and is associated with the Boolean operation NAND. The $m$ distinguished output gates - $t_1, t_2, \ldots, t_m$ - are conventionally regarded as having out-degree equal to 0. An assignment of Boolean variables from $< 0, 1 >^n$ to the inputs $X_n$ ultimately induces Boolean values at the output gates $< t_1, \ldots, t_m >$. An $n$-input, $m$-output Boolean network, $S$, is said to compute an $n$-input, $m$-output Boolean function,

$f(X_n) :< 0, 1 >^n \rightarrow < 0, 1 >^m =_{def} < f^{(i)}(X_n) :< 0, 1 >^n \rightarrow \{0, 1\} : 1 \leq i \leq m >$ if $\forall \alpha \in < 0, 1 >^n \ \forall 1 \leq i \leq m \ t_i(\alpha) = f^{(i)}(\alpha)$.

The two standard complexity measures for Boolean networks are *size* and *depth*: the size of a network, $S$, denoted $C(S)$, is the number of gates in $S$; its depth, denoted by $D(S)$, is the number of gates in the *longest* directed path connecting an input vertex to an output gate.

The simulation takes place in three distinct phases:

1. Set-up
2. Level simulation
3. Final read-out of output gates

We now describe each phase in detail.

### 2.1. Set-up

In what follows we use the term *tube* to denote a set of strings over some alphabet $\sigma$. We denote the $j$th gate at level $k$ by $g_k^j$. We first create a tube, $T_0$, containing

unique strings of length $l$, each of which corresponds to only those input gates that have the value 1. We then create, for each level $1 \leq k < D(S)$, a tube $T_k$ containing unique strings of length $3l$ representing each gate at level $k$. We also create a tube $S_k$, containing strings corresponding to the complement of positions $2l-5$ to $2l+5$ for each $g_k^j$. We define the concept of complementarity in the next section, but for the moment we assume that if sequence $x$ and its complement $\overline{x}$ are present in the same tube, the string containing sequence $x$ is in some way "marked".

We then create tube $T_{D(S)}$, containing unique strings representing the output gates $< t_1, \ldots, t_m >$. These strings representing gates at level $1 \leq k < D(S)$ are of the form $x_k^j y_k^j z_k^j$. If gate $g_k^j$ takes its input from gates $g_{k-1}^m$ and $g_{k-1}^n$, then the sequence representing $x_k^j$ is the complement of the sequence representing $z_{k-1}^m$, and $y_j^k$ is the complement of the sequence representing $z_{k-1}^n$. The presence of $z_k^j$ therefore signifies that $g_k^j$ has an output value of 1.

The strings in tube $T_{D(S)}$ are similar, but the length of the sequence $z_{D(S)}^j$ is in some way proportional to $j$. Thus, the length of each string in $T_{D(S)}$ is linked to the index of the output gate it represents.

### 2.2. Level simulation

We now describe how levels $1 \leq k < D(S)$ are simulated. We create the set union of tubes $T_{k-1}$ and $T_k$. Strings representing gates which take either of their inputs from a gate with an output value of 1 are "marked", due to their complementary nature. We then remove from $T_k$ all strings that have been marked twice (i.e., those representing gates with both inputs equal to one). We then split the remaining strings after section $y_k^j$, retaining the sequences representing $z_k^j$. This subset then forms the input to tube $T_{k+1}$.

### 2.3. Final read-out of output gates

At level D(S) we create the set union of tubes $T_{D(S)-1}$ and $T_{D(S)}$ as described above. We then, as before, remove from this set all strings that have been marked twice. By checking the length of each string in this set we are therefore able to say which output gate has the value 1, and which has the value zero by the presence or absence of a string representing the gate in question.

## 3. Physical implementation

We now describe how the abstract model detailed in the previous section may be implemented in the laboratory using standard bio-molecular manipulation techniques. The implementation is similar to that of our *parallel filtering model*, described in [2, 4].

We first describe the design of strands representing the input gates $X_n$. For each $X_n$ that has the value 1 we synthesise a unique strand of length $l$. We now describe the design of strands representing gates at level 1. We have already synthesised a unique strand to represent each $g_k^j$ at the set-up stage. Each strand is comprised of three components of length $l$, representing the gate's inputs and output. Positions 0 to $l$ represent the first input, positions $l+1$ to $2l$ represent the second input,

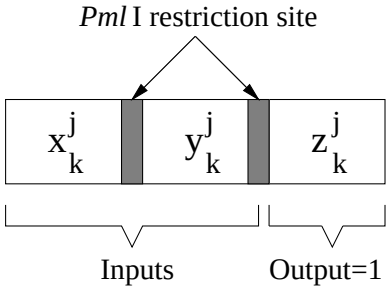


Figure 1. Structure of a gate strand

and positions $2l+1$ to $3l$ represent the gate's output. Positions $l-3$ to $l+3$ and positions $2l-3$ to $2l+3$ correspond to the restriction site $CACGTG$. This site is recognized and cleaved exactly at its mid-point by the restriction enzyme $Pml$I, leaving blunt ends. Due to the inclusion of these restriction sites, positions 0 to 2, $l+1$ to $l+3$ and $2l+1$ to $2l+3$ correspond to the sequence $GTG$, and positions $l-3$ to $l$, $2l-3$ to $2l$ and $3l-3$ to $3l$ correspond to the sequence $CAC$. The design of the other sub-sequences is described in Section 2.1.. A graphical depiction of the structure of each gate strand is shown in Figure 1.

The simulation proceeds as follows for levels $1 \leq k < D(S)$.

1. At $k$ pour into $T_k$ the strands in tube $T_{k-1}$. These anneal to the gate strands at the appropriate position.

2. Add ligase to $T_k$ in order to seal any "nicks".

3. Add to $T_k$ the restriction enzyme $Pml$I. Because of the strand design, the enzyme cleaves only those strands that have *both* input strands annealed to them. This is due to the fact that the *first* restriction site $CACGTG$ is only made fully double-stranded if both of these strands have annealed correctly. This process is depicted in Figure 2.

4. Denature the strands and run $T_k$ through a gel, retaining only those strands of length $3l$. This may be achieved in a single step by using a denaturing gel [16].

5. Add tube $S_k$ to tube $T_k$. The strands in tube $S_k$ anneal to the *second* restriction site embedded within each retained gate strand.

6. Add enzyme $Pml$I to tube $T_k$, which "snips" off the $z_k^j$ section (i.e., the output section) of each strand representing a retained gate.

7. Denature and run $T_k$ through another gel, this time retaining only strands of length $l$. This tube, $T_k$ of retained strands forms the input to $T_{k+1}$. We now proceed to the simulation of level $k+1$.

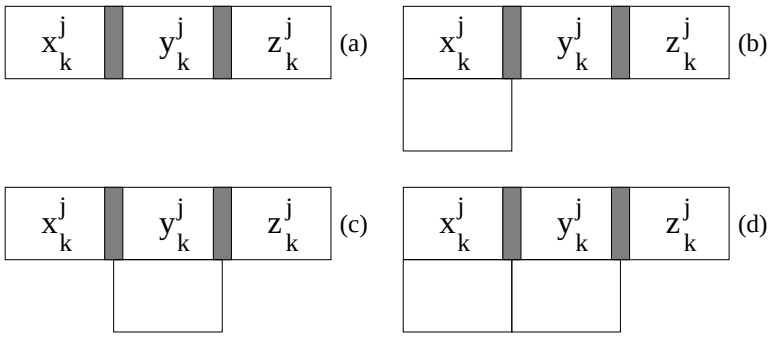


Figure 2. (a) Both inputs=0 (b) First input=1 (c) Second input=1 (d) Both inputs=1

At level $D(S)$ we carry out steps 1-7, as described above. However, at steps 4 and 7 we retain all strands of length $\geq 3l$. We are now then ready to implement the final read-out phase. This involves a simple interpretation of the final gel visualisation. Since we know the unique length $u_j$ of each $z^j_{D(S)}$ section of the strand for each output gate $t_j$, the presence or absence of a strand of length $u_j + 2l$ in the gel signifies that $t_j$ has the value one or zero respectively.

## 4. Analysis

We first analyse our model in terms of the feasibility of its biological implementation. During the course of the simulation, we use the following operations; primer annealing, ligation, restriction, and denaturing gel electrophoresis. We note that in the description of their implementation [12], Ogihara and Ray use all of these operations, plus the polymerase chain reaction (PCR). The inherent problems with PCR have been well-documented in the literature [2, 10], and it is clear that reliance on this technique should be minimised if possible. As well as avoiding the need for PCR, our proposed implementation greatly reduces the degree of physical manipulation of tubes of DNA. This minimises potential problems such as strand shear and material loss due to strands sticking to the surface of tubes.

We now compare both our and Ogihara and Ray's models by describing how Batcher sorting networks [5] may be implemented within them. Batcher networks sort $n$ inputs in $O(log^2 n)$ stages. In [19], Wegener showed that if $n = 2^k$ then the number of *comparison* modules is $0.5n(\log n)(\log n - 1) + 2n - 2$. The circuit *depth*, (again expressed in terms of the number of comparison modules) is $0.5(\log n)(\log n + 1)$. A comparison module has two (Boolean) inputs, $x, y$, and two outputs $MIN(x, y)$ (which is just $x\ AND\ y$); $MAX(x, y)$ (which is just $x\ OR\ y$).

Using NAND we can build a comparison module with five NAND gates and having depth 2 (the module is levelled, so since the Batcher network is levelled with respect to comparison modules, the whole realisation in NAND gates will be levelled). The NAND gate realisation is

| *Model* | *Volume* | *Time* |
|---|---|---|
| O&R | $(K1)(n(\log n)(\log n - 1) + 4n - 4)$ | $n(\log n)(\log n - 1) + 4n + 4$ |
| A&D | $(K2)(2.5(\log n)(logn - 1) + 10n - 10)$ | $7(\log n)(\log n + 1)$ |

Table 1. Model comparison for Batcher network simulation

| $n$ | $k$ | *O&R* | *A&D* |
|---|---|---|---|
| 1024 | 10 | 92196 | 770 |
| $2^{20}$ | 20 | $4 * 10^8$ | 2940 |
| $2^{40}$ | 40 | $1.7 * 10^{15}$ | 11480 |

Table 2. Time comparisons for different values of $k$

$MIN(x, y) = NAND(NAND(x, y), NAND(x, y))$ - 2 gates, depth 2;
$MAX(x, y) = NAND(NAND(x, x), NAND(y, y))$ - 3 gates, depth 2;

If we assume that $n = 2^k$ this gives the total size (in terms of number of gates) as $2.5(\log n)(\log n - 1) + 10n - 10$ and depth (in gates) as $(\log n)(logn + 1)$.

Ogihara and Ray use an AND, OR gate simulation, so they would realise a comparison module with two gates ($MAX(x, y) = x\ OR\ y$; $MIN(x, y) = x\ AND\ y$)) and depth 1, giving the size of the network as $n(\log n)(\log n - 1) + 4n - 4$ and the depth of the network as $0.5(\log n)(\log n + 1)$.

Within the context of our *strong model* [3], the volumes of DNA and the time required to simulate an $n$-input Batcher network within each model are depicted in Table 1. $K1$ and $K2$ are constants, representing the number of copies of a single strand required to give reasonable guarantees of correct operation. The coefficient of 7 in the A&D time figure represents the number of separate stages in a single level simulation. If we concentrate on the time measure for $n = 2^k$ we arrive at the figures shown in Table 2.

Roweis *et al.* claim that their *sticker model* [15] is feasible using $2^{56}$ distinct strands. We therefore conclude that our implementation is technically feasible for input sizes that could not be physically realised *in silico* using existing fabrication techniques.

## 5. Conclusions

In this paper we described an abstract model for the simulation of Boolean circuits using DNA, as well as a description of its laboratory implementation. Our simulation runs in time proportional to the circuit *depth*, in contrast to previously proposed simulations, which run in time proportional to the circuit *size*. In addition, the proposed implementation of our model avoids the use of error-prone techniques, such as PCR. However, we have not yet attempted to physically realise our model in the laboratory. We acknowledge the substantial practical difficulties in implementing the model for even small circuits, and emphasise that much more

work needs to be done on establishing the error-resistance of basic operations, such as strand removal.

## 6. Acknowledgements

The authors are grateful to Alan Gibbons for his comments on an earlier draft of this paper.

## References

[1] Leonard M. Adleman. Molecular computation of solutions to combinatorial problems. *Science*, 266:1021–1024, 1994.

[2] Martyn Amos. *DNA Computation.* PhD thesis, Department of Computer Science, University of Warwick, UK, September 1997. Available at http://www.csc.liv.ac.uk/~ctag/archive/th/amos-thesis.ab.html.

[3] Martyn Amos, Alan Gibbons, and Paul E. Dunne. The complexity and viability of DNA computations. In Lundh, Olsson, and Narayanan, editors, *Proceedings of the First International Conference on Bio-Computing and Emergent Computation*, pages 165–173, University of Skövde, Sweden, 1997. World Scientific.

[4] Martyn Amos, Steve Wilson, David A. Hodgson, Gerald Owenson, and Alan Gibbons. Practical implementation of DNA computations. In C.S. Calude, J. Casti, and M.J. Dinneen, editors, *Unconventional Models of Computation*, Discrete Mathematics and Theoretical Computer Science. Springer-Verlag, Singapore, 1998. To appear.

[5] K. E. Batcher. Sorting networks and their applications. In *Proc. AFIPS 1968 Spring Jt. Computer Conf.*, volume 32, pages 307–314, Washington, D.C., 1968. Thompson Book,Co.

[6] Erzsébet Csuhaj-Varjú, R. Freund, Lila Kari, and Gheorghe Păun. DNA computation based on splicing: universality results. In Lawrence Hunter and Teri Klein, editors, *Biocomputing: Proceedings of the 1996 Pacific Symposium.* World Scientific, January 1996.

[7] Paul E. Dunne. *The Complexity of Boolean Networks.* Academic Press, 1988.

[8] M.A. Harrison. *Introduction to switching and automata theory.* McGraw-Hill, 1965.

[9] Thomas Head. Formal language theory and DNA: an analysis of the generative capacity of specific recombinant behaviors. *Bulletin of Mathematical Biology*, 49(6):737–759, 1987.

[10] Peter D. Kaplan, Guillermo Cecchi, and Albert Libchaber. DNA based molecular computation: template-template interactions in PCR. In *Proceedings of the Second Annual Meeting on DNA Based Computers*, June 1996.

[11] Carver Mead and Lynn Conway. *Introduction to VLSI systems.* Addison Wesley, 1980.

[12] Mitsunori Ogihara and Animesh Ray. Simulating Boolean circuits on a DNA computer. Technical Report 631, University of Rochester, August 1996.

[13] N. Pippenger. On simultaneous resource bounds. In *20th Annual Symposium on Foundations of Computer Science*, pages 307–311, Long Beach, Ca., USA, October 1979. IEEE Computer Society Press.

[14] Paul Wilhelm Karl Rothemund. A DNA and restriction enzyme implementation of Turing machines. In *DNA Based Computers*, 1996.

[15] Sam Roweis, Erik Winfree, Richard Burgoyne, Nickolas Chelyapov, Myron Goodman, Paul Rothemund, and Leonard Adleman. A sticker based architecture for DNA computation. In *Proceedings of the Second Annual Meeting on DNA Based Computers.*

[16] J. Sambrook, E.F. Fritsch, and T. Maniatis. *Molecular Cloning: A Laboratory Manual.* Cold Spring Harbor Press, second edition, 1989.

[17] H.M. Sheffer. A set of five independent postulates for boolean algebras, with application to logical constants. *Trans. Amer. Math. Soc.*, 14:481–488, 1913.

[18] Warren D. Smith and Allan Schweitzer. DNA computers in vitro and vivo. In *DNA Based Computers*, 1996.

[19] Ingo Wegener. *The Complexity of Boolean Functions.* Wiley-Teubner, 1987.

[20] N.E. Weste and K. Eshragan. *Principles of CMOS VLSI Design.* Addison-Wesley, 1993.