Arithmetic and Logic Operations with DNA

Vineet Gupta, Srinivasan Parthasarathy and Mohammed J. Zaki

V. Gupta, Dept. of Chemistry, University of Rochester, Rochester, NY 14627.

E-mail: vtga@uhura.cc.rochester.edu

S. Parthasarathy, M. J. Zaki, Dept. of Computer Science, University of Rochester, Rochester, NY 14627.

E-mail: (srini,zaki)@cs.rochester.edu

All authors contributed equally to this work.

Abstract

The use of DNA molecules to solve hard computational problems has been demonstrated in recent studies. However, the question of suitability of DNA for solving simple computer operations, such as boolean or arithmetic operations, has largely been unaddressed. Incorporation of these operations in DNA computing is essential for solving a wide range of applications. We present a fixed bit encoding scheme, modeling the input/output mechanisms of an electronic computer, and show how a sequence of such operations can be executed in a single test tube producing a unique result.

In recent work, Adleman [1] and Lipton [2] presented the idea of solving difficult combinatorial search problems using DNA molecules. These studies showed that DNA computing may have an advantage over electronic computers for such problem domains due to the massive parallelism inherent in DNA reactions. However, for DNA computing to be applicable on a wider range of problems, support for simple computational operations is necessary [3]. Boolean operators such as AND, OR and NOT, and arithmetic operators such as addition and subtraction are the fundamental operations of an electronic computer. In contrast to search problems, which can be solved by generating all possible combinations and extracting the correct output, these operations mandate that only a unique output be generated by specific inputs. Although DNA computers might not improve on current silicon technology for these operations, as Adleman [4] has pointed out, "they can contribute to our understanding of the nature of computation." Recently, Guarnieri *et al.* [5] have proposed a clever way to add two binary numbers. The novelty of their approach is the introduction of a place holder for the carry position while performing additions [6]. As they themselves point out, a limitation of their approach is that the output strand of one operation cannot serve as the input strand for another round of addition. This has been a constraining factor so far in performing a sequence of operations in a single vessel. In this paper we show that this limitation can be overcome (in DNA), allowing a number of basic series and series-parallel bit operation sequences in solution.

In an electronic computer an operator is succinctly represented by a truth table, a table of all possible combinations of the input bit values and their corresponding output values (Table 1A). Our approach is to encode these truth tables in DNA using a three-level scheme. An operation is represented in terms of DNA hybridization. For each binary operation, the two bit strings are represented with two different DNA single strands. The first string is called the "input" and the second the "operand" strand. Each bit is represented with a dinucleotide unit, and a bit string with a sequence of dinucleotides. The natural DNA bases Adenine (A), Thymine (T), Uracil (U) and a non-natural base 2,6-diaminopurine (P) (Fig. 1) are used for constructing the dinucleotides. The input DNA strand is constructed with dinucleotides 5'-AU [7] representing bit 1, and 5'-UA, the bit 0. The operand strand is constructed with dinucleotides 3'-TA and 3'-PT, representing bit 0, and the dinucleotides 3'-AT and 3'-TP, representing bit 1 [8]. The input strand is constructed using only A and U (level 1), while the operand strand with A, T, and P (level 2). This

keeps the input distinct from the operand, yet retains the same base-pairing structure and allows all possible combinations of the input and operand bits (Table 1). As a result of an operation the input strand hybridizes with its complementary operand strand to form a double stranded DNA complex. This output is interpreted according to the truth table (level 3) of the operator applied. Table 1B shows our encoding scheme along with the truth tables for different boolean and arithmetic operations such as NAND, AND, XOR and ADD (addition). The truth table encoding can easily be extended to a number of other operators.

The NAND operator is a *universal* boolean operator, which means that any boolean operation can be represented in terms of a sequence of NAND operators. Our first example shows the execution, in DNA, of the operator NAND on the two binary strings 1001 and 0101 (Table 2). The input bit string 1001, is represented as the DNA sequence 5'-AUUAUAAU. All 16 possible DNA sequences are used to represent the operand 0101, since there are two alternative dinucleotides for each bit in the level 2 encoding. However, as shown in Table 2, only one of the operand strands carries a DNA sequence complementary to the sequence of the input strand. This strand (3'-TAATPTTP) is thus the only one that can hybridize with the input strand to yield a unique output duplex. The output, when decoded using the truth table for NAND, yields the correct answer of 1110. In fact, the same output can be interpreted differently if decoded using a different operator's truth table. For instance, if we were to perform the operation 1001 XOR 0101 instead of 1001 NAND 0101, the input and all of the operand strands would still be the same, and would yield the same output duplex. However, interpreting this output duplex according to the truth table for XOR (Table 1B) would generate a different but correct answer of 1100. This shows that single bit operations can be done efficiently with DNA using the proposed approach.

In order to extend this framework to handle a *series* operation, i.e. a linear sequence of operations, we had to address two main issues: 1) how can the output of one operation, a double strand, be used as the input for the next operation, and 2) how to ensure that the operations take place only in the desired order. As shown in the first example (Table 2), *each input strand can bind with only one of the operand strands producing a duplex representing the unique output.* As a solution to the first problem we attach, *a priori*, the corresponding output value to each of the operand strands in the form of a single stranded DNA. The DNA sequence of the output is constructed using the level 1 encoding (Table 1B). Thus, each operand strand

carries a covalently linked output strand which does not interfere with its operation of hybridizing with the input strand, but instead results in a duplex with a sticky-end upon one such complexation. This sticky-end can then act as the input strand for the next operation. Not only does this allow a series operation involving a single operator, but also mixed operators, since the output strand attached to the operand is dependent only on a particular operator.

To solve the second problem, we use a unique DNA sequence tag on each strand. This tag is just a length 4 DNA segment comprised of four bases, the natural bases Guanine(G) and Cytidine (C) along with the non-natural bases iso-Guanine (M) and iso-Cytidine (N) [9] (Fig. 1). Each input strand is linked with a unique tag, while the operand strand next in order in the sequence, is linked with the complementary tag. The input can thus only hybridize with the next operand. A unique tag for the next step is also linked to the output strand, since it serves as the input for the next operation. Since each of the strands is made up of an operand and an output, they may become self-complementary. The tags also prevent the formation of such unproductive duplexes. Using this approach, we generate unique DNA sequences as inputs and obtain a unique final output resulting from the desired sequence of operations.

In Figure 2, we present an example of how three different operators – NAND, XOR and ADD – can be performed in succession on DNA. The example solves the series operation sequence (((1001 NAND 0101) XOR 0001) ADD 0001). The first two bit strings are the same as those used in our first example (Table 2). The input strand (5'-AUUAUAAU-GCMN) has the bit string 5'-AUUAUAAU (1001) linked with the tag 5'-GCMN. Figure 2A shows the 16 possible representations of the operand strand (0101) that could possibly bind with the input (similar to our first example). Only one strand hybridizes with the input, namely the strand 3'-TAATPTTP-CGNM-AUAUAUUA-CCGG, where 3'-TAATPTTP is the bit string 0101, 3'-CGNM is the complement of the tag 5'-GCMN, 3'-AUAUAUUA is the attached output (1110) of the operation and 3'-CCGG is the tag for the next operation. The result of the first operation is thus a duplex with a single stranded overhang (3'-AUAUAUUA-CCGG), which is ideally suited to serve as the input strand for the next operation – XOR 0001.

Our second operand strand, representing 0001, also has 16 possible representations (not shown). Only 5'-TATATAATGGCC-AUAUAUAUCGCG (Fig. 2B) hybridizes with the output of the previous opera-

tion, producing a duplex with the overhang 5'-AUAUAUAUCGCG as the output (1111) of the current operation. Out of the 16 possible third operand strand (0001) DNA sequences, for ADD 0001, only 3'-TATATATPGCGCAUUAUAUAUA$(NNNN)_3$ hybridizes with the output of the previous round of operations. 5'-$(NNNN)_3$ is a special tag signifying that the sequence of operations has completed and is used for extracting the result. The unique result of our operation 3'-AUUAUAUAUA, when decoded using the truth table, is 10000, which is the correct result of our operation sequence. This examples shows that our scheme generates a unique output from a series operation in solution.

In order to emulate the more complex boolean circuits as in an electronic computer, support for *series-parallel* operation sequences is desirable. A series-parallel operation sequence merges two or more series operation sequences. In Figure 3 we show how the above architecture can be extended to handle a series-parallel operation sequence in a single test tube. The output of the series operation (from the previous example, Fig. 2B), a duplex with an overhang is used as both the input and the operand complex, performing the operation [(((1001 NAND 0101) XOR 0001) ADD 0001) **NAND** (((1001 NAND 0101) XOR 0001)]. However, instead of the typical operand strand which has an output attached to it, the operand complex for the series-parallel operation has two attached outputs – the output of the series-parallel operation followed by the output of the series operation sequence (Fig. 3A). The final step is a copy-operation – AND 11111 (which simply replicates the output). The result of this operation sequence is a three-arm junction [10] with the correct output (01111) as the overhang, ideally suited for performing more operations.

Although only a theoretical model is presented in this paper, it is based on well-established techniques of biological chemistry. To practically carry out the operations in a test tube, we can attach the input strand 5'-AUUAUAAUGCMN to a magnetic bead, and add all the operand strands for the different operations to be performed, to the solution. The solution can then be heat-denatured, annealed and ligated [11]. All strands with the magnetic bead attached can be filtered off, and the output strand end-sequenced to produce the result.

For an n bit long string ($2n$ length DNA strand), with tag length m, our technique must satisfy some constraints. The length of an input/operand strand is $m + 2n$. Since the mismatch tolerance of a duplex increases with its length, at present duplexes only about 20 base-pairs long can distinguish single-base

mismatches [12], introducing the constraint $m + 2n \leq 20$ on our system. Furthermore, for m operations, we need at least $m \cdot 2^n$ distinct DNA strands in the solution. For nanomolar quantities of each, thermodynamical limitations would put the upper bound at approximately 10^{14} different strands. The first constraint, though, clearly subsumes the second. The total length of the output complex, $2n \cdot m$, might also be limited by various factors that affect the reliablility, such as the hybridization conditions, nearest-neighbor interactions (sequence context), and the efficiency of both the successive ligations and the result extraction process. To improve the reliability, the end-tag can be used for PCR amplification of the output. The PCR amplified output can be sequenced to also verify the operations performed. Finally, the number of series-parallel operations may be limited due to the steric constraints (branching-out) that three-arm junctions introduce.

There are 4^{2n} different sequences possible for a $2n$ length DNA strand, if random encodings are used. An advantage of using our fixed encoding scheme is that we need to synthesize only 2^n DNA sequences per operation [13], which is a very small subset of 4^{2n}. The increase in demand of DNA strands is also linear in the number of operations rather than exponential ($m \cdot 2^n$ instead of $2^m \cdot 2^n$). Furthermore, since all possible representations of the operands, for each of the operations, are added to the solution in the beginning, all possible outputs are produced in the solution. We can simply add a given input to the solution, which will bind to its complementary sequence, and then extract the unique output. The same solution can then be reused with different inputs, for sequential extraction of the corresponding outputs. Thus the vessel serves as a black-box (representing some function to be performed), which takes in different input values to produce the corresponding outputs, without having to perform the operation again. Moreover, the fact that all the intermediate results are present in the final output, prevents the loss of any information about the computation, and can be used for implementing reversible logic gates [14].

There are some other advantages to our approach. The use of dinucleotides as single bit units lets us represent information at a much higher bit density than any previous methodology, and brings it very close to the theoretical limit of a bit per base [15]. Our methodology also makes possible a succinct representation of basic computational operators. The use of different bases for encoding the bits and the operation order permits series as well as series-parallel operation sequences in one test tube, producing a unique result, and simplifies the extraction process. The unique tags help keep the operation sequence in order. They

also avoid the *fan-out* problem where a randomized sequence of molecules can produce an exponentially increasing number of product strands with the progression of the operation sequence. We use 2^n molecules for each step, which interact with 2^n molecules from the previous step to generate exactly 2^n molecules for the next step.

Generation of DNA molecules carrying sticky-ends as the output of each operation, in the reaction vessel, is another nice feature of our scheme. This output is well suited to serve as the input for the next operation which enables us to accomplish the generic input/output semantics of an electronic computer. This is an important first step in solving problems using DNA computing where the DNA molecules have to go through multiple rounds of computation. Moreover, in most of the prior approaches where DNA was used for computing, the encoding was application specific and based on the particular application in mind. One of the most important advantages of our proposed mechanism is that it makes both encoding and computation more general (uniform) and application independent. As our examples show, almost any basic computational operation can be carried out on our "DNA computer".

In conclusion, we present a new approach for computing with DNA. The ability to perform complex bit operations in solution might help us learn more about the nature of computation and lead to the development of better DNA based computers, capable of solving a wide range of complex problems.

References

[1] L. M. Adleman, *Science* **266**, 1021 (1994).

[2] R. J. Lipton, *Science* **268**, 542 (1995).

[3] D. K. Gifford, *Science* **266**, 993 (1994); R. Pool, *Science* **268**, 498 (1995); H. Rubin, *Nature Str. Biol.* **3**, 656 (1996).

[4] L. M. Adleman, *Science* **268**, 483 (1995).

[5] F. Guarnieri, M. Fliss and C. Bancroft, *Science* **273**, 220 (1996).

[6] In contrast to boolean operators, ADD can generate two output bits (e.g. 1+1 = 10). In this case that bit position is assigned a value of zero (0) and the one (1) is carried over. The carried over bit is added to the next bit position and the value of that position adjusted accordingly. The operation is repeated as many times as there is a carry over.

[7] 5' and 3' denote directionality of the DNA strand.

[8] If the input strand is constructed in the 5' to 3' direction the operand strand is constructed in the 3' to 5' direction and vice versa.

[9] This allows for a maximum of 256 unique tags, which gives us the option of performing 256 consecutive operations. This number can be increased by simply using a longer tag sequence.

[10] N. C. Seeman, *J. Theor. Biol.* **99**, 237 (1982).

[11] Ligation can be performed either enzymatically, using DNA ligase enzyme, or chemically using Letsinger's method [M. K. Herrlein et al, *J. Am. Chem. Soc.* **117**, 10151, (1995)]. The DNA strands will have to be properly end-modified for either of the cases. The modifications should not affect their hybridizing properties.

[12] R. B. Wallace et al., *Nucleic Acids Res.* **6**, 3543 (1979).

[13] All the strands can be synthesized in a single combinatorial synthesis cycle using techniques of photolithography and nucleic acid chemistry [S. P. A. Fodor et al., *Science* **251**, 767 (1991); A. C. Pease et al., *Proc. Natl. Acad. Sci. U. S. A.* **91**, 5022 (1994)].

[14] C. H. Bennet and R. Landauer, *Scientific American* **253**, 48 (1985).

[15] Although, in principle mono-nucleotides can be used as single bits, there may be some complications to using such strands for performing different operations. For example, a long sequence of a mono-nucleotide can form unwanted triplexes.

[16] We thank R. Jakubiak, Prof. E. T. Kool, H. S. Malik, Prof. B. L. Miller, Prof. M. Ogihara and Prof. A. Ray for their insightful comments and helpful discussions.

Table 1 A. The truth table for various binary operations

Input 1	Input 2	Output			
		NAND	AND	XOR	ADD
0	0	1	0	0	0
0	1	1	0	1	1
1	0	1	0	1	1
1	1	0	1	0	10

Table 1 B. The truth table encoding using dinucleotide "bits"

Level 1	Level 2	Level 3 (Interpretation of the Duplex for the Output)			
Input Strand	Operand Strand	NAND	AND	XOR	ADD
UA = 0	PT = 0	1	0	0	0
	AT = 1	1	0	1	1
AU = 1	TA = 0	1	0	1	1
	TP = 1	0	1	0	10

Table 2 Example of a NAND operation

Input Strand	Operand Strands	Unique Output
1 0 0 1	0 1 0 1	**1 1 1 0**
5′-AUUAUAAU	3′-TATPPTAT 3′-TATPPTTP	5′-AUUAUAAU
	3′-TATPTAAT 3′-TATPTATP	3′-TAATPTTP
	3′-TAATPTAT **3′-TAATPTTP**	
	3′-TAATTAAT 3′-TAATTATP	
	3′-PTTPPTAT 3′-PTTPPTTP	
	3′-PTTPTAAT 3′-PTTPTATP	
	3′-PTATPTAT 3′-PTATPTTP	
	3′-PTATTAAT 3′-PTATTATP	

Figure 1

Chemical structure of the bases used; Adenine (A), Thyamine (T), Uracil (U), 2,6-diaminopurine (P), Guanine (G), Cytidine (C), iso-Guanine (M) and iso-Cytidine (N). Also shown are some example base-pairs, and a dinucleotide "bit" unit.

Figure 2

Example of a series operation.

A. The first operation : (1001 NAND 0101)

B. The whole series : (((1001 NAND 0101) XOR 0001) ADD 0001)

C. A cartoon of the structure generated.

Figure 3

Example of a series-parallel combination operation (Input NAND operand).

Input : (((1001 NAND 0101) XOR 0001) ADD 0001) = 10000 (same as in Fig. 2)

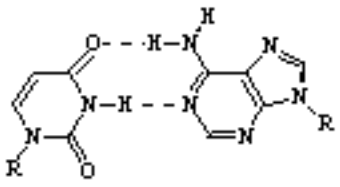
Operand : (((1001 NAND 0101) XOR 0001) ADD 0001) = 10000

Intermediate Result : 10000 NAND 10000 = 01111

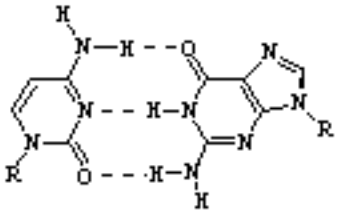
Copy-operation : 01111 AND 11111 = 01111 (Final Output).

A. The sequence of the input complex, operand complex, and the copy-operation strands.

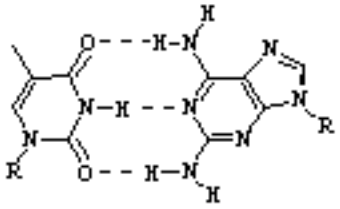
B. The three-arm junction generated as a result of the series-parallel operation. (Inset) A cartoon of the three-arm junction structure.



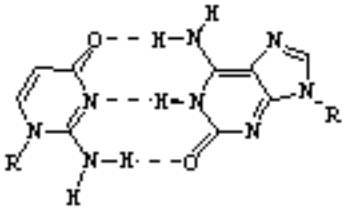
U A



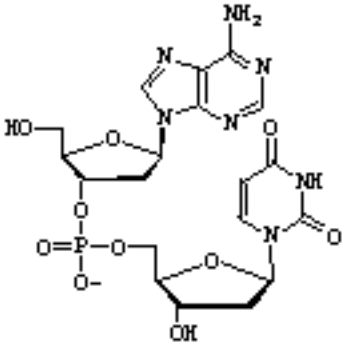
C G



T P

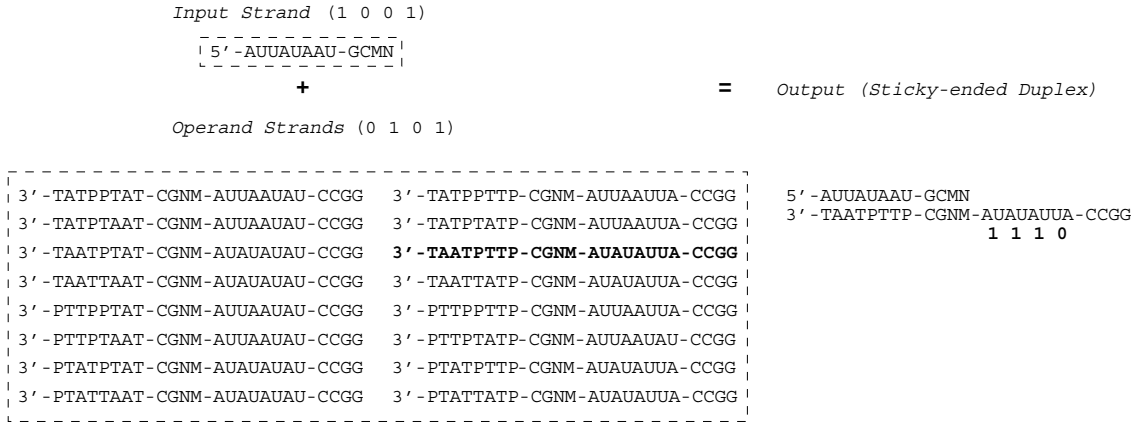


N M

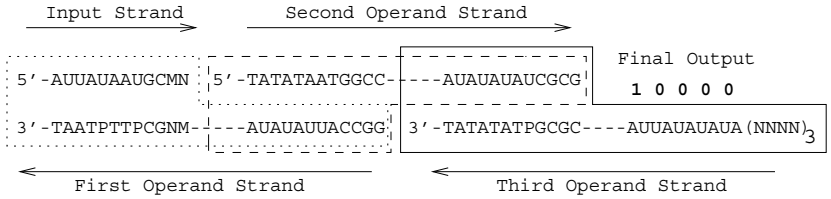


A dinucleotide unit = One Bit
(5'-AU)

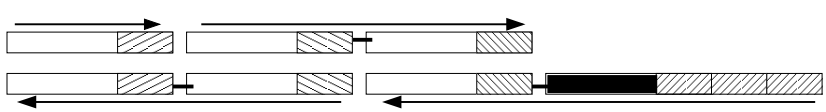
A

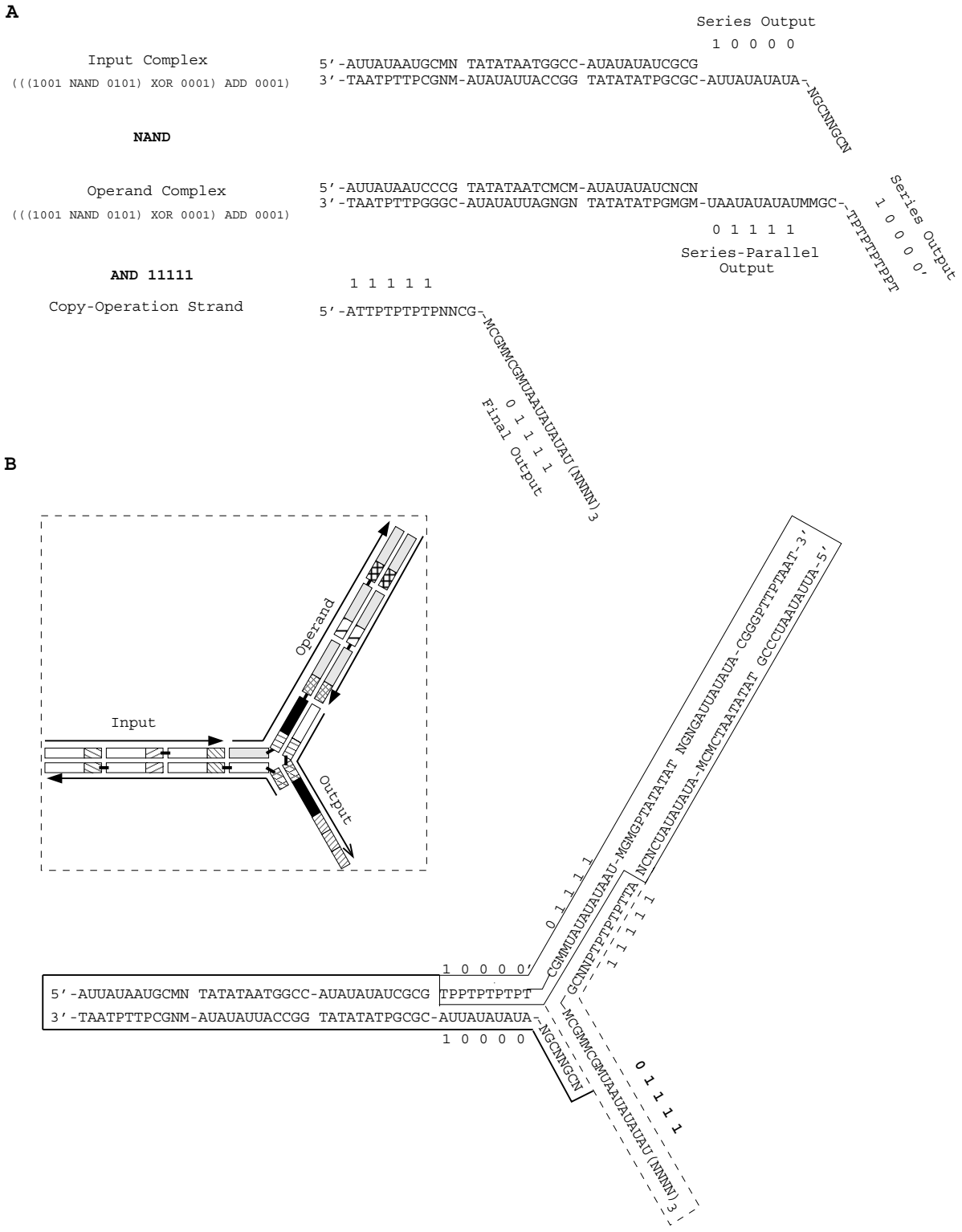

Input Strand (1 0 0 1)
5’-AUUAUAAU-GCMN
+
=
Output (Sticky-ended Duplex)
Operand Strands (0 1 0 1)
3’-TATPPTAT-CGNM-AUUAAUAU-CCGG
3’-TATPPTTP-CGNM-AUUAAUUA-CCGG
3’-TATPTAAT-CGNM-AUUAAUAU-CCGG
3’-TATPTATP-CGNM-AUUAAUUA-CCGG
3’-TAATPTAT-CGNM-AUAUAUAU-CCGG
3’-TAATPTTP-CGNM-AUAUAUUA-CCGG
3’-TAATTAAT-CGNM-AUAUAUAU-CCGG
3’-TAATTATP-CGNM-AUAUAUUA-CCGG
3’-PTTPPTAT-CGNM-AUUAAUAU-CCGG
3’-PTTPPTTP-CGNM-AUUAAUUA-CCGG
3’-PTTPTAAT-CGNM-AUUAAUAU-CCGG
3’-PTTPTATP-CGNM-AUUAAUAU-CCGG
3’-PTATPTAT-CGNM-AUAUAUAU-CCGG
3’-PTATPTTP-CGNM-AUAUAUUA-CCGG
3’-PTATTAAT-CGNM-AUAUAUAU-CCGG
3’-PTATTATP-CGNM-AUAUAUUA-CCGG
5’-AUUAUAAU-GCMN
3’-TAATPTTP-CGNM-AUAUAUUA-CCGG
1 1 1 0

B


Input Strand
Second Operand Strand
5’-AUUAUAAUGCMN
5’-TATATAATGGCC-----AUAUAUAUCGCG
Final Output
1 0 0 0 0
3’-TAATPTTPCGNM-----AUAUAUUACCGG
3’-TATATATPGCGC----AUUAUAUAUA(NNNN)3
First Operand Strand
Third Operand Strand

C




A
Input Complex
(((1001 NAND 0101) XOR 0001) ADD 0001)
NAND
Operand Complex
(((1001 NAND 0101) XOR 0001) ADD 0001)
AND 11111
Copy-Operation Strand
Series Output
1 0 0 0 0
5′-AUUAUAAUGCMN TATATAATGGCC-AUAUAUAUCGCG
3′-TAATPTTPCGNM-AUAUAUUACCGG TATATATPGCGC-AUUAUAUAUA-
-NGCNNNGCN
5′-AUUAUAAUCCCG TATATAATCMCM-AUAUAUAUCNCN
3′-TAATPTTPGGGC-AUAUAUUAGNGN TATATATPGMGM-UAAUAUAUAUMMGC-
-TPTPTPTPTPPT
Series Output
1 0 0 0 0′
0 1 1 1 1
Series-Parallel
Output
1 1 1 1 1
5′-ATTPTPTPTPNNCG-
-MCGMMCGMUAAUAUAUAU(NNNN)3
0 1 1 1 1
Final Output
B
Operand
Input
Output
1 0 0 0 0′
5′-AUUAUAAUGCMN TATATAATGGCC-AUAUAUAUCGCG TPPTPTPTPT
3′-TAATPTTPCGNM-AUAUAUUACCGG TATATATPGCGC-AUUAUAUAUA-
1 0 0 0 0
-NGCNNNGCN
0 1 1 1 1
CGMMUAUAUAUAAU-MGMGPTATATAT NGNGAUUAUAUA-CGGGPTTPTAAT-3′
GCNNPTPTPTPTPTA
1 1 1 1 1
NCNCUAUAUAUA-MCMCTAATATAT GCCCUAAUAUUA-5′
MCGMMCGMUAAUAUAUAU(NNNN)3
0 1 1 1 1