Logical Design with Molecular Components

Filomena de Santis and Gennaro Iaccarino

Department of Informatica e Applicazioni "R.M. Capocelli" University of Salerno, Via S. Allede, 84081 - Baronissi, Italy

Abstract. We propose a theoretical model to realize DNA made circuits based on *in-vitro* algorithms, to perform arithmetic and logical operations. The physical components of the resulting Arithmetic-Logic Unit are a variety of elements such as biochemical laboratories, test tubes and human operators. The advantage of the model is the possibility to perform arithmetic operations with huge binary numbers.

1 Introduction

As it is well known, arithmetic and logical operations are done in a conventional computer by the Arithmetic-Logic Unit that often incurs in overflow or underflow problems, due to the minimal and maximal size of the representable numbers. We present some basic instruments to realize simple circuits based on DNA algorithms that constitute the bio-hardware of a DNA Arithmetic-Logic Unit overcoming overflow and underflow arithmetic limitations. Table 1 summarizes the notations and the chemical operations used in the sequel.

Table 1. Notations and Chemical Operations

Symbols	
x, ¬x	Generic DNA sequence and its complement
x^i	i repetition of x sequence
$\uparrow$ x, $\downarrow$ x, $\updownarrow$ x	Upper, lower and double strand x
Chemical Operations	
Synthesis:	Generating of DNA single strands in vitro.
Annealing:	Bounding of two complementary DNA single strands
Cutting:	Cutting a DNA double strand by restriction enzyme
Ligation:	Pasting two DNA double strands by restriction enzyme
Polymerization:	Generating a complete double strand from a portion of it
Gel Electrophoresis:	Separating DNA strands by electric charges
PCR:	Cloning and amplifying a particular DNA piece in solution

2 Representing Binary Strings

Each binary number can be encoded by a set of integers indicating the positions where bits are set to 1 [2]; correspondently, its biochemical representation can be done by a set of DNA double strands test tubes $T[\alpha]_m ... T[\alpha]_1$

associated to the positions where bits are set to 1 [3]. An example of DNA double strand, representing an integer, is the following:

$$\updownarrow\underbrace{(aagctct^5)^i}_{S^i}\, aagctt\, \underbrace{(ctgcatg^5)^k}_{X^k}\, ctgcag\, \underbrace{(gaattgc^5t^5g^5c)^j}_{Y^i}\underbrace{gaattc}_{E_0}$$

where S^i encodes the test tube containing the strand and X^k the byte in which the molecular bit is contained; Y^j represents the offset into the byte and E_0 the end of each DNA strand. All the subsequences are linked by the restriction sites *aagctt*, *ctgcag* and *gaattc*, respectively for HandIII, PstI and EcoRI enzymes. According with this schemes, the first position is encoded with $\updownarrow\ (aagctct^5)^1 aagctt (ctgcatg^5)^1 ctgcag (gaattgc^5t^5g^5c)^1 gaattc$ and so on for the successive ones. In order to simplify the biological operations we need, each sequence has different size and Y has to be $l-1$ times longer than X, where l is the maximum value of k. Thus, a generical value α is encoded by the set of test tube $T[\alpha]_m ... T[\alpha]_1$ that contain all the integers of $X[\alpha]$.

3 Logical and Arithmetic Operations

3.1 Logical Operations

In [5] Weiss and Basu report *in-vivo* experimental results which examine the steady state behavior of cellular logic gates, with mRNA support. Here we propose the logical design for *in-vitro* logic gates, using DNA algorithms. Let $T[\alpha]_m ... T[\alpha]_1$ and $T[\beta]_m ... T[\beta]_1$ be the test tubes encoding binary strings α and β respectively, with m<<n. The **OR**, $\alpha \vee \beta$, is executed synthesizing the test tubes $T[\alpha]_i$ and $T[\beta]_i$, $\forall i = 1..n$, and mixing them together. The **XOR**, $\alpha \oplus \beta$, is executed extracting all the double strands that belong exclusively to $T[\alpha]_i$ or $T[\beta]_i$. The **AND**, $\alpha \wedge \beta$, is executed extracting all molecular bits that belong contemporarily to $T[\alpha]_i$ and $T[\beta]_i$. The **NOT**, $\neg\alpha$, is executed synthesizing a set of test tube $T_m ... T_i$, applying the AND operation with $T[\alpha]_m ... T[\alpha]_1$ and eventually extracting these sequences from the solutions. Each operation is performed efficiently whatever is the size of the input binary strings. Each algorithm requires a constant number of bio-steps [3], related to the number of test tubes used in the representation of binary numbers. Thus the expected number of bio-steps for each logical operation is $O(m)$, where m is the unfixed number of test tubes requested by molecular bits. If the number of bits is fixed the complexity is $O(1)$. Previous analyses carry out theoretical schemes where each gate has input and output strings composed by DNA test tubes, circuits are constituted by bio-steps, and the computational site is a biological laboratory.

3.2 Arithmetic Operations

As it is shown in [1], we can implement addition and multiplication operations by recursive procedures. Let $\alpha = \alpha_n ... \alpha_1$ and $\beta = \beta_n ... \beta_1$ be two binary

strings, and $X[\alpha] = \{i : \alpha_i = 1\}$ and $X[\beta] = \{j : \beta_j = 1\}$, it results:

$$\mathbf{Add}(\alpha, \beta) = Val(RecursiveAdd(X[\alpha], X[\beta])); \tag{1}$$

where

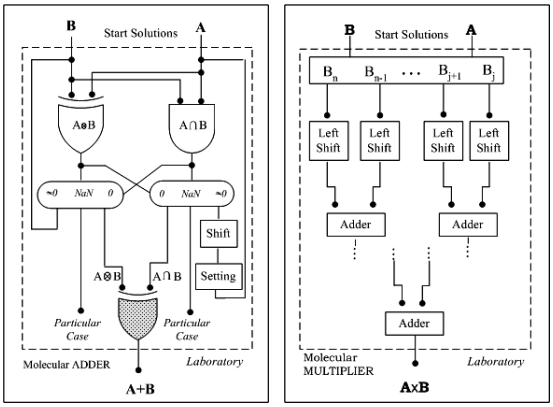
$$ReacursiveAdd(Y, Z) = \begin{cases} X & if \quad Z = \emptyset \\ Z & if \quad Y = \emptyset \\ RecursiveAdd((Y \oplus Z)(Y \cap Z)^{+}) & otherwise \end{cases}$$

The multiplication procedure can be realized using progressive additions of values, left shifted. So the multiplication operation results in:

$$\mathbf{Mul}(\alpha, \beta) = \mathbf{Add}(\{\, Val\ (X[\alpha] + (j-1))\}\beta_j = 1) \tag{2}$$

Subtraction and division are trivial consequences of them.

For each $T[\alpha]_i$ and $T[\beta]_i$ with $i = 1...m$, the addition is performed as follows:
Step1. Divide molecular bits in two different test tubes $T[\alpha \oplus \beta]_i$ and $T[\alpha \cap \beta]_i$. Check whether the set $T[\alpha \oplus \beta]_i$ or $T[\alpha \ \cap \ \beta]_i$ is empty. If that's true, then the set of test tubes $T[\alpha + \beta]_i$ are equal to the not empty test tubes. Else go to step2.
Step2. Shift on the left all the bits contained in $T[\alpha \cap \beta]_i$, that is increase by one position all the double strands in solution, producing $(X[\alpha] \cap X[\beta])^{+}$. Repeat this two simple steps until one of the set of test tube in step 1 is empty. Figure 1(a) shows a logical circuit that faithfully reproduces the molecular algorithm steps. The procedure is realized as follows. With the help of restriction enzyme *EcoRI*, cut all the double strands at their $3'$ end. Add the upper strands $\uparrow$ $attgc^5t^5g^5cgaattc$ with the ligation enzyme and attend that the polymerization process forms double strands of this kind: $\updownarrow (aagctct^5)^i aagctt(ctgcatg^5)^k ctgcag(gaattgc^5t^5g^5c)^{j+1}gaattc$. Thus each bit position in $T[\alpha \ \cap \ \beta]_i$ has been shifted on the left. To reorganize the solution in bytes, move surplus bits from a byte to the next and from a test tube to the next [3]. Restriction enzymes and ligation are used, enzyme *SalI* to increase X^k and *HandIII* to increase S^i. At the end of this process, the surplus bits will be: $\updownarrow\ (aagctct^5)^i aagctt(ctgcatg^5)^{k+1}ctgcag\ \ (gaattgc^5t^5g^5c)^1 gaattc$ and $\updownarrow (aagctct^5)^{i+1}aagctt(ctgcatg^5)^1 ctgcag(gaattgc^5t^5g^5c)^1 gaattc$.
For the multiplication first calculate all shifted values of α bits, in comparison to 1-bit$\in\ X[\beta]$, then sum them as in a tree data structure. Thus progressive additions are not made with the same solutions but, concurrently with successive pairs of tubes. This procedure, besides to improve complexity, avoids that biological errors might affect DNA in solutions. Figure 1(b) shows a simple multiplier circuit. As shown in [3], the expected number of bio-steps, for the addition, is $O(m \cdot \log_2 n)$ and becomes $O(\log_2 n)$, if the number of bits is finite and limited to one test tube. Multiplication complexity is $O(m \cdot (\log_2 n)^2)$; it depends on the logarithmic number of addition in the tree. Also for multiplication it became $O(\log_2 n)^2)$ in the best case. Arithmetic operations can be described as conventional circuits.



(a) (b)

Fig. 1. Logical design of molecular circuits. (**a**) Molecular Adder: dark gates represent not DNA made logical selections. (**b**) Molecular Multiplier.

4 Floating Point Arithmetic

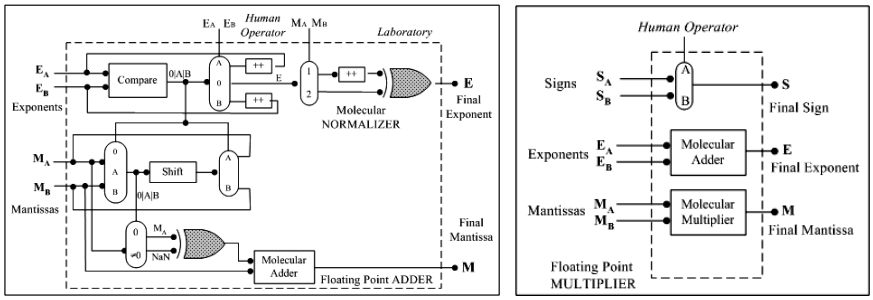
The DNA representation of floating numbers we propose is comparable to the IEEE 754 [4]. Using the DNA representation presented above, we can divide the set of test tubes as follow: $T[\alpha]_{sign}T[\alpha]_{exp}T[\alpha]_m...T[\alpha]_1$, where $T[\alpha]_{sign}$ encodes the sign of α. $T[\alpha]_{exp}$ the molecular bits for the exponent and $T[\alpha]_m...T[\alpha]_1$ the mantissa f. As it happens in IEEE 754 standard, we need to choose a set of DNA strings to represent a few of mathematical significative values ($0, \pm\infty$ and NaN) [3].
For the addition, compare α and β exponents (by *gel electrophoreses*) and increase the mantissa of the greater of $|exp_\alpha - exp_\beta|$ positions. The result is the left shift of the mantissa toward greatest positions. Perform integer addition with α and β mantissas and then normalize the result [3] comparing the new most significant bit with the old, and shifting it if smaller.
Multiplication is realized by adding α and β exponents and multiplying their mantissas (integer multiplication). At the end of this process compare $T[\alpha]_{sign}$ and $T[\beta]_{sign}$ and determine resulting sign as shown in Table 2. The

Table 2. Sign Choice

Choice	$T[\alpha \cdot \beta]_{sign}$	Sign
$T[\alpha]_{sign} = T[\beta]_{sign} = \phi$	$T[\alpha \cdot \beta]_{sign} = \phi$	Positive
$T[\alpha]_{sign} = T[\beta]_{sign} \neq \phi$	$T[\alpha \cdot \beta]_{sign} = \phi$	Positive
$T[\alpha]_{sign} \neq T[\beta]_{sign}$	$T[\alpha \cdot \beta]_{sign} \neq \phi$	Negative

expected bio-steps for the addition depend on the bio-steps in each computational step. Only one gel electrophoresis is required to choose the exponent, so the complexity is $O(1)$. The integer subtraction requires $O(\log_2 q)$ [3] and

the mantissa left shift $O(q)$, where q is the number of bits in the representation of the exponent. The integer addition takes $O(m \cdot \log_2 n)$ and the final gel electrophoresis and the increment $O(\log_2 q)$. In conclusion, the expected bio-steps are: $O(1) + O(\log_2 q) + O(q) + O(m \cdot \log_2 n)$ that is $O(m \cdot \log_2 n)$. They become $O(\log_2 n)$, if the bits of the mantissa are fixed and limited to one test tube. The expected bio-steps for the multiplication depends on the integer addition and multiplication: $O(\log_2 q) + O(m \cdot (\log_2 n)^2)$ namely $O(m \cdot (\log_2 n)^2)$. They become $O(\log_2 n)^2$, if the bits of the mantissa are fixed. Logical circuits for floating point adder and multiplier are respectively shown in Figure 2.



(a) (b)

Fig. 2. Floating Point Adder (**a**) and Multiplier (**b**). Dark sections represent logical selections, not implemented with Dna molecules.

5 Conclusion

The purpose of this paper was to introduce a concrete approach for the logical design of a real DNA-ALU, defeating limitations of results presented in [1], such as the fixed number of bits available for the user, and in [5], such as difficulties in building real biological circuits due to the *in-vivo* nature of its experimental basis.

References

1. Barua R. (2002) Binary Arithmetic for DNA Computer, 8^{th} International workshop on DNA-Based Computers: Dna Computing, pp. 124-132
2. Biswas S. (1998) Computing with Bio-Molecules. Theory and Experiment, Ed G. Paun
3. de Santis F., Iaccarino G. (2004) A DNA Arithmetic Logic Unit, WSEAS Transactions on Biology and Biomedicine, vol. **1**, October 2004, pp. 436-440
4. Kahan W. (1996) IEEE Standard 754 for Binary Floating Point Arithmetic, Lecture Notes on status of IEEE 754, University of California Berkeley CA.
5. Weiss R., Basu S. (2002) The Device Physics of Cellular Logic Gates, First Workshop on Non-Silicon Computing, Cambridge, Mass.