Hardware addition over finite fields based on Booth-Karatsuba algorithm

J. Ayuso Pérez

Abstract—Two algorithms, both based around multiplication, one defined by Andrew Donald Booth [1] in 1950 and the other defined by Anatoly Alexeevitch Karatsuba [2] in 1960 can be applied to other types of operations. We know from recent results [3-14], that to perform some algebraic transformations it is more efficient to calculate an inverse effect of a smaller magnitude together with an original action with a higher rank, than the initial operation, reaching the final element with less transitions. In this paper, we present an addition algorithm on $\mathbb{Z}/m\mathbb{Z}$ of big-integer numbers based on these concepts, using an alternative to traditional hardware implementation for binary addition based on FULL-ADDER cells, allowing the reduction of space complexity compared to other techniques, such as carry-lookahead, letting us calculate a modular addition in an optimal order complexity of $\mathcal{O}(n)$ without adding more complexity due to reduction operations.

Index Terms—modular addition, Booth algorithm, Karatsuba algorithm, hardware binary adder, finite field operations, Galois field arithmetic

1 Introduction

Our aim is to describe a hardware solution capable of calculating modular additions without representational problems or overflow on abelian groups $\mathbb{Z}/m\mathbb{Z}$ being $m \in \mathbb{N}$.

In the following, we consider finite fields, even knowing that our results only require the addition and therefore it could be applied on finite abelian groups.

We are assuming that Booth-Karatusuba (B-K) concept is applicable to:

- group operations defined as the succession of other operations [3]
- transformations on the same algebraic context [4], so that they could be cancelled [6]
- structures in which every element has an inverse and therefore congruence relation [5]
- other algebraic contexts [8]
- functions of arity greater than 2 [10]
- group relations that have inverse functions, which are different to one another [11]
- composite primitives which satisfy both, commutative and transitive properties [13]

With the previous premises, in this example we denote $a, b \in \mathbb{N}$, in 4-bit binary representation, assuming that: $a = a_3 \cdot 2^3 + a_2 \cdot 2^2 + a_1 \cdot 2^1 + a_0 \cdot 2^0$ and $b = b_3 \cdot 2^3 + b_2 \cdot 2^2 + b_1 \cdot 2^1 + b_0 \cdot 2^0$ to apply B-K to the addition operation [9,14]. This is our starting point in this analysis: these criteria allow us to avoid the FULL-ADDER and HALF-ADDER cells used in current solutions, describing new hardware capable of operating on finite fields without representational problems.

- *This work was supported in part by the Complutense University of Madrid and Research Service of UCM and its scholarship programs.*
- *Corresponding author: J. Ayuso Pérez (ORCID: https://orcid.org/0000-0001-8287-6089).*

Manuscript received May 8, 2023; revised May 8, 2023.

2 Addition scheme for $\mathbb{Z}/m\mathbb{Z}$ with B-K algorithm

Two contexts should be defined: additive and subtractive, where the actions of B-K change their meaning. For that reason the destination of the outputs of the upper level of the electronic schematics changes.

We use the cancellation behavior of transformations in the first layer of AND gates, together with building actions, when identical binary relations are joined on the same part of the structure. All actions must be pre-calculated as far in advance as possible, of course.

TABLE 1
BOOTH-KARATSUBA 2-ARITY ACTIONS

bit $i+1$	bit i	Interpretation	Action
0	0	intermediate 0s sequence	no operation
0	1	end 1s sequence	group operation
1	0	start 1s sequence	reverse operation or inverse with group operation
1	1	intermediate 1s sequence	no operation

In the first approximation, we seek to apply the actions described in the previous table to the neutral element of the group on which we work. In other words, we apply successor and predecessor actions for each position of the binary representation. We understand that in each digit there is a 0 for using positional representation.

Still outside finite fields, using simultaneously the schema of **figure 1** in two operands, we denote $a, b \in \mathbb{N}$ in binary representation, and we define $0 + a + b$ as:

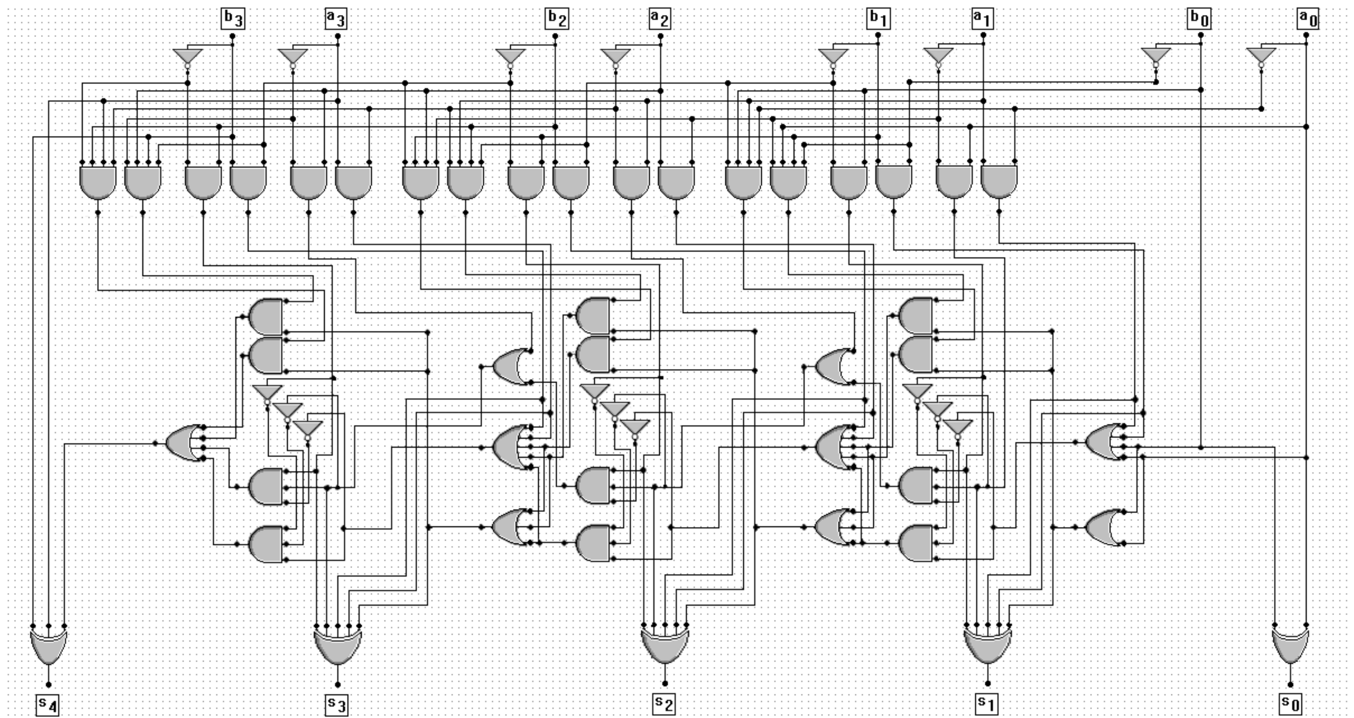

Fig. 1. Symmetric addition algorithm with 4 bits by Booth-Karatsuba.

Note that in **figure 4** there is an asymmetry in the management of the bits of the operands: on the projective weight of the ordinal successor, the OR gate only takes into account one of the 2 values involved. However, it is interchangeable for the $i-th$ bit of the other operand, for the same i. This is an interesting implementation detail, since we look for a result with correct symmetry.

In the second approximation, we change to a subtractive context, based on the previous circuit. We conceive the subtractive relationship with the scheme of **figure 5**: let $a, b \in \mathbb{N}$, both with a length of n binary digits, we compute the result of: $0 - a - b$.

Of course, B-K actions are not implemented with neutral element, and we also know that B-K actions change their meaning in **figure 5** with respect to **figure 4**. For this reason, the conductive tracks in the electronic circuit change target in the outputs of higher-level gates: the upper level of the circuit, consisting only of AND gates and that implements B-K logic. On the other hand, the predecessor calculation is more complicated, due to the accumulation of subtractive transformations: an additional 1 logic gate is included to ensure the correctness of the result.

In **figure 5** there is a reduction in the number of logic gates for the second bit as a result of overlap of operations in the additive solution, but that disappears in its dual algebraic: subtraction. In any case, these savings are negligible for large operand sizes, and do not significantly impact on the space complexity.

As a side-effect, one more bit is added to the design, in order to be able to represent all the cases of overflows and propagation. In this paper, the word *overflow* is not referenced in computational terms; we use it with an algebraic approach, as a consequence of the problem of representation of an element through a positional system of finite fields.

These solutions definitely do not use classical operational logic to calculate the addition between 2 binary bits; only the B-K simplification is implemented. Note that the normal method for adding binary digits and the *carry-out* calculation disappears in the first input level: logic gates closest to the inputs; the circuits in **figure 4** and **figure 5** implement **table 1** logic and they use it to project against the predecessor or successor operand: the successor or the predecessor, in case of a subtractive context. For this reason, the first layer gates are free of XOR cells.

It should be noted that the circuits in **figure 4** and **figure 5** are simply previous steps to the physical architecture that we propose, but helps to follow the paper. Of course, they do not support finite fields and they would not be efficient alternatives to current solutions to implement the addition and subtraction of integers, having a higher latency and greater spatial complexity.

However, the designs of the two figures are the base on which we can build 2 functions of three-arity, adding a third operand [10]. This does not mean that a nested function is needed. Avoiding further logic gate levels is critical to limit space complexity and order complexity in this work.

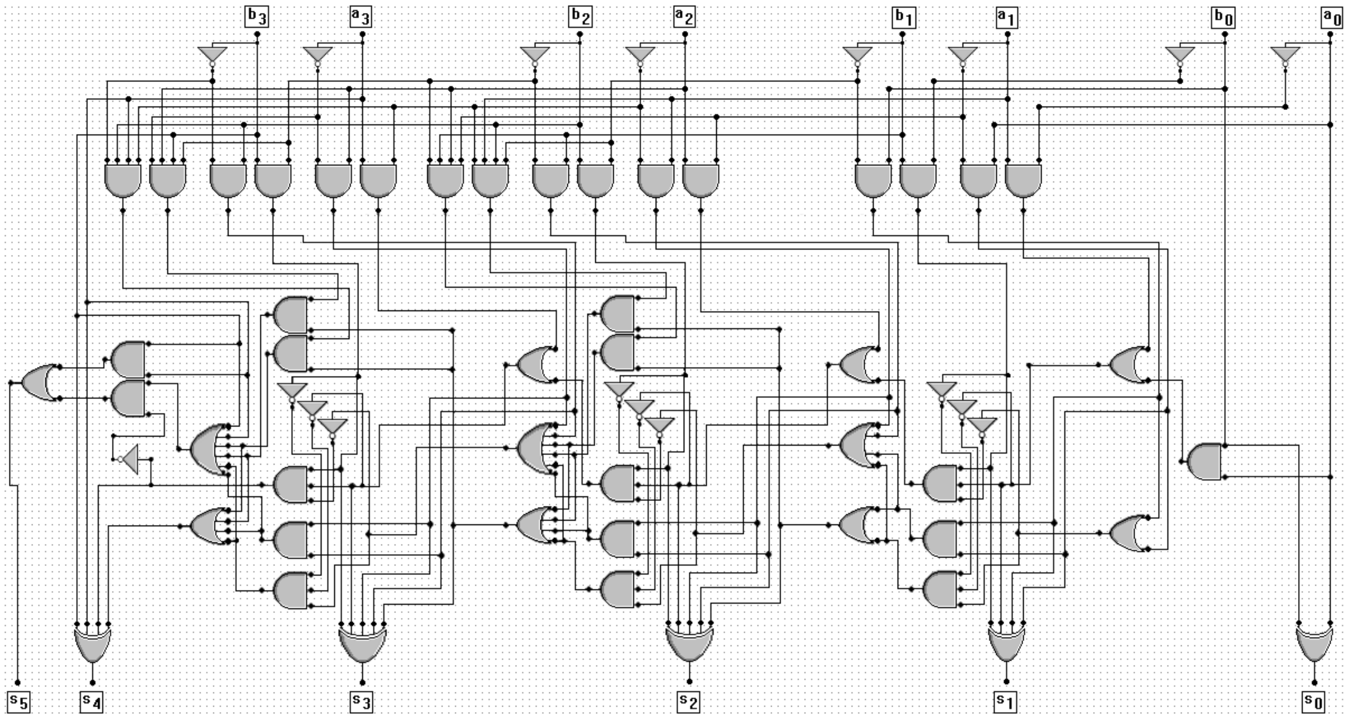

Fig. 2. Symmetric subtraction algorithm with 4 bits by Booth-Karatsuba.

We incorporate to **figure 4** the ability to work with three elements, in this case subtracting the modulus defined for the finite field. We must distinguish multiple situations for the successor:

TABLE 2
BOOTH-KARATSUBA 3-ARITY SUCCESSOR

$i - bit$ start	$i - bit$ end	Action
yes		subtractive sequences, with a transformative weight of the module
		successors, with a transformative weight of the module, without reaching the end of subtractive sequences
yes		subtractive sequences and successors, with a transformative weight of the module without predecessors
yes		subtractive sequences, without a transformative weight of the module and without predecessors

And for the predecessor we manage the cases:

TABLE 3
BOOTH-KARATSUBA 3-ARITY PREDECESSOR

$i - bit$ start	$i - bit$ end	Action
		transformative cancellation, without module weight and without predecessors
		predecessors, without a transformative weight of the module and without successors
	yes	subtractive sequence, no successors
	yes	subtractive sequences with predecessors

We incorporate to **figure 5** the ability to work with three elements, in this case, adding the modulus defined for the finite field. We have to distinguish for the successor:

TABLE 4
BOOTH-KARATSUBA 3-ARITY SUCCESSOR

$i - bit$ start	$i - bit$ end	Action
yes		subtractive sequence from the module and end of additive sequence, with a transformative weight of an operand
		successors, with a transformative weight of an operand, without end of subtractive sequence in the module or beginning of additive
yes		subtractive sequence of modulus and successors, with a transformative weight of an operand without predecessors
	yes	additive sequence of an operand and successors, without cancellations, with a transformative weight of an operand without predecessors
yes		subtractive and additive sequence without predecessor

Similarly, and for the predecessor we manage the cases:

TABLE 5
BOOTH-KARATSUBA 3-ARITY PREDECESSOR

$i - bit$ start	$i - bit$ end	Action
		transformative cancellation, without weight of the pivot operand and without predecessors
		predecessors, without a transformative weight and without successors
	yes	subtractive sequence of the module, without successors
	yes	additive sequence with predecessors

We define c^x, being $x \in \mathbb{N} \mid 1 \geq x \leq 4$, to denote carry-in and carry-out values. As soon as the results of **table 2** and **table 3** and taking account of the recent results [3-14], we explore all options and we defined the following truth table:

TABLE 6
MODULAR FULL ADDER

Inputs												Outputs					Inputs												Outputs				
a_{i-1}	b_{i-1}	a_i	b_i	m_i	a_{i+1}	b_{i+1}	m_{i+1}	c^1	c^2	c^3	c^4	s_i	c^1	c^2	c^3	c^4	a_{i-1}	b_{i-1}	a_i	b_i	m_i	a_{i+1}	b_{i+1}	m_{i+1}	c^1	c^2	c^3	c^4	s_i	c^1	c^2	c^3	c^4
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	1	0	0	1	1
0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	1	1	1	0	0	0	0	0	0	0	0	0	1	1	1	0	0	1	1
0	0	0	0	0	0	0	1	0	0	0	0	1	0	1	0	0	1	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	1	1	0	0	1	1	1	1	0	0	0	0	0	0	1	0	0	1	0	0	0	1	1	1
0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	1	0	0	1	1	0	0	1	1	1
0	0	0	0	0	0	1	0	0	0	1	1	0	0	0	1	1	1	0	0	0	0	0	0	1	1	1	1	0	1	0	1	0	0
0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	1	0	0	0	1	0	0	0	0	1	1
0	0	0	0	0	0	1	1	0	0	1	1	1	0	1	1	1	1	0	0	0	0	0	1	0	0	0	1	1	0	0	0	1	1
0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	1	0	1	1	1	0	1	0	0	0	0
0	0	0	0	0	1	0	0	0	0	1	1	0	0	0	1	1	1	0	0	0	0	0	1	1	0	0	1	0	1	0	1	1	1
0	0	0	0	0	1	0	1	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	1	1	0	0	1	1	1	0	1	1	1
0	0	0	0	0	1	0	1	0	0	1	1	1	0	1	1	1	1	0	0	0	0	0	1	1	1	1	1	0	0	0	1	0	0
0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	1	1
0	0	0	0	0	1	1	0	0	0	1	1	1	0	0	1	1	1	0	0	0	0	1	0	0	0	0	1	1	0	0	0	1	1
0	0	0	0	0	1	1	1	0	0	0	0	1	0	1	0	0	1	0	0	0	0	1	0	0	1	1	1	0	1	0	0	0	0
0	0	0	0	0	1	1	1	0	0	1	1	0	0	1	1	1	1	0	0	0	0	1	0	1	0	0	1	0	1	0	1	1	1
0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	1	0	1	0	0	1	1	1	0	1	1	1
0	0	0	0	1	0	0	0	0	1	1	1	0	0	0	0	0	1	0	0	0	0	1	0	1	1	1	1	0	0	0	1	0	0
0	0	0	0	1	0	0	1	0	1	0	0	1	0	1	0	0	1	0	0	0	0	1	1	0	0	0	1	0	1	0	0	1	1
0	0	0	0	1	0	0	1	0	1	1	1	1	0	1	0	0	1	0	0	0	0	1	1	0	0	0	1	1	1	0	0	1	1
0	0	0	0	1	0	1	0	0	1	0	0	1	0	0	0	0	1	0	0	0	0	1	1	0	1	1	1	0	0	0	0	0	0
0	0	0	0	1	0	1	0	0	1	1	1	1	0	0	0	0	1	0	0	0	0	1	1	1	0	0	1	0	0	0	1	1	1
0	0	0	0	1	0	1	1	0	1	0	0	0	0	1	0	0	1	0	0	0	0	1	1	1	0	0	1	1	0	0	1	1	1
0	0	0	0	1	0	1	1	0	1	1	1	0	0	1	0	0	1	0	0	0	0	1	1	1	1	1	1	0	1	0	1	0	0
0	0	0	0	1	1	0	0	0	1	0	0	1	0	0	0	0	1	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	1	1	0	0	0	1	1	1	1	0	0	0	0	1	0	0	0	1	0	0	0	0	1	1	1	1	0	0	1	1
0	0	0	0	1	1	0	1	0	1	0	0	0	0	1	0	0	1	0	0	0	1	0	0	0	1	1	0	0	0	0	0	0	0
0	0	0	0	1	1	0	1	0	1	1	1	0	0	1	0	0	1	0	0	0	1	0	0	1	0	1	0	0	1	0	1	0	0
0	0	0	0	1	1	1	0	0	1	0	0	0	0	0	0	0	1	0	0	0	1	0	0	1	0	1	1	1	0	0	1	1	1
0	0	0	0	1	1	1	0	0	1	1	1	0	0	0	0	0	1	0	0	0	1	0	0	1	1	1	0	0	1	0	1	0	0
0	0	0	0	1	1	1	1	0	1	0	0	1	0	1	0	0	1	0	0	0	1	0	1	0	0	1	0	0	1	0	0	0	0
0	0	0	0	1	1	1	1	0	1	1	1	1	0	1	0	0	1	0	0	0	1	0	1	0	0	1	1	1	0	0	0	1	1
0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0	0	0	1	0	1	0	1	1	0	0	1	0	0	0	0
0	0	0	1	0	0	0	0	0	0	1	1	1	0	0	1	0	1	0	0	0	1	0	1	1	0	1	0	0	0	0	1	0	0
0	0	0	1	0	0	0	1	0	0	0	0	0	0	1	0	0	1	0	0	0	1	0	1	1	0	1	1	1	1	0	1	1	1
0	0	0	1	0	0	0	1	0	0	1	1	0	0	1	0	0	1	0	0	0	1	0	1	1	1	1	0	0	0	0	1	0	0
0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	1	0	0	0	1	0	0	1	0	0	0	0
0	0	0	1	0	0	1	0	0	0	1	1	0	0	0	0	0	1	0	0	0	1	1	0	0	0	1	1	1	0	0	0	1	1
0	0	0	1	0	0	1	1	0	0	0	0	1	0	1	0	0	1	0	0	0	1	1	0	0	1	1	0	0	1	0	0	0	0
0	0	0	1	0	0	1	1	0	0	1	1	1	0	1	0	0	1	0	0	0	1	1	0	1	0	1	0	0	0	0	1	0	0
0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	1	1	0	1	0	1	1	1	1	0	1	1	1
0	0	0	1	0	1	0	0	0	0	1	1	0	0	0	1	0	1	0	0	0	1	1	0	1	1	1	0	0	0	0	1	0	0
0	0	0	1	0	1	0	1	0	0	0	0	1	0	1	0	0	1	0	0	0	1	1	1	0	0	1	0	0	0	0	0	0	0
0	0	0	1	0	1	0	1	0	0	1	1	1	0	1	0	0	1	0	0	0	1	1	1	0	0	1	1	1	1	0	0	1	1
0	0	0	1	0	1	1	0	0	0	0	0	1	0	0	0	0	1	0	0	0	1	1	1	0	1	1	0	0	0	0	0	0	0
0	0	0	1	0	1	1	0	0	0	1	1	1	0	0	0	0	1	0	0	0	1	1	1	1	0	1	0	0	1	0	1	0	0
0	0	0	1	0	1	1	1	0	0	0	0	0	0	1	0	0	1	0	0	0	1	1	1	1	0	1	1	1	0	0	1	1	1
0	0	0	1	0	1	1	1	0	0	1	1	0	0	1	0	0	1	0	0	0	1	1	1	1	1	1	0	0	1	0	1	0	0
0	0	0	1	1	0	0	0	0	1	0	0	0	1	1	1	0	1	0	0	1	0	0	0	0	0	0	1	0	1	0	0	1	0
0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	1	0	1	0	0	1	0	0	0	0	0	0	1	1	0	0	0	1	1
0	0	0	1	1	0	0	1	0	1	0	0	1	1	1	0	0	1	0	0	1	0	0	0	0	1	1	1	0	1	0	0	1	0
0	0	0	1	1	0	0	1	0	1	1	1	0	0	1	0	0	1	0	0	1	0	0	0	1	0	0	1	0	0	0	1	0	0
0	0	0	1	1	0	1	0	0	1	0	0	1	1	1	0	0	1	0	0	1	0	0	0	1	0	0	1	1	1	0	1	1	1
0	0	0	1	1	0	1	0	0	1	1	1	0	0	0	0	0	1	0	0	1	0	0	0	1	1	1	1	0	0	0	1	0	0
0	0	0	1	1	0	1	1	0	1	0	0	0	1	1	0	0	1	0	0	1	0	0	1	0	0	0	1	0	0	0	0	0	0
0	0	0	1	1	0	1	1	0	1	1	1	1	0	1	0	0	1	0	0	1	0	0	1	0	0	0	1	1	1	0	0	1	1
0	0	0	1	1	1	0	0	0	1	0	0	1	1	1	1	0	1	0	0	1	0	0	1	0	1	1	1	0	0	0	0	0	0
0	0	0	1	1	1	0	0	0	1	1	1	0	0	0	1	0	1	0	0	1	0	0	1	1	0	0	1	0	1	0	1	0	0
0	0	0	1	1	1	0	1	0	1	0	0	0	1	1	0	0	1	0	0	1	0	0	1	1	0	0	1	1	0	0	1	1	1
0	0	0	1	1	1	0	1	0	1	1	1	1	0	1	0	0	1	0	0	1	0	0	1	1	1	1	1	0	1	0	1	0	0
0	0	0	1	1	1	1	0	0	1	0	0	0	1	1	0	0	1	0	0	1	0	1	0	0	0	0	1	0	0	0	0	1	0
0	0	0	1	1	1	1	0	0	1	1	1	1	0	0	0	0	1	0	0	1	0	1	0	0	0	0	1	1	1	0	0	1	1
0	0	0	1	1	1	1	1	0	1	0	0	1	1	1	0	0	1	0	0	1	0	1	0	0	1	1	1	0	0	0	0	1	0
0	0	0	1	1	1	1	1	0	1	1	1	0	0	1	0	0	1	0	0	1	0	1	0	1	0	0	1	0	1	0	1	0	0
0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0	0	1	0	1	0	1	0	0	1	1	0	0	1	1	1
0	0	1	0	0	0	0	0	0	0	1	1	1	0	0	1	0	1	0	0	1	0	1	0	1	1	1	1	0	1	0	1	0	0
0	0	1	0	0	0	0	1	0	0	0	0	0	0	1	0	0	1	0	0	1	0	1	1	0	0	0	1	0	1	0	0	0	0
0	0	1	0	0	0	0	1	0	0	1	1	0	0	1	0	0	1	0	0	1	0	1	1	0	0	0	1	1	0	0	0	1	1
0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	1	0	1	0	0	1	0	1	1	0	1	1	1	0	1	0	0	0	0
0	0	1	0	0	0	1	0	0	0	1	1	0	0	0	1	0	1	0	0	1	0	1	1	1	0	0	1	0	0	0	1	0	0
0	0	1	0	0	0	1	1	0	0	0	0	1	0	1	0	0	1	0	0	1	0	1	1	1	0	0	1	1	1	0	1	1	1
0	0	1	0	0	0	1	1	0	0	1	1	1	0	1	0	0	1	0	0	1	0	1	1	1	1	1	1	0	0	0	1	0	0
0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1	0	0	0	0	1	0	0	1	0	0	1	0
0	0	1	0	0	1	0	0	0	0	1	1	0	0	0	0	0	1	0	0	1	1	0	0	0	0	1	1	1	1	0	0	1	0
0	0	1	0	0	1	0	1	0	0	0	0	1	0	1	0	0	1	0	0	1	1	0	0	1	1	0	0	0	1	1	1	1	0
0	0	1	0	0	1	0	1	0	0	1	1	1	0	1	0	0	1	0	0	1	1	0	0	1	0	1	0	0	0	0	1	0	0
0	0	1	0	0	1	1	0	0	0	0	0	1	0	0	0	0	1	0	0	1	1	0	0	1	0	1	1	0	0	0	1	0	0
0	0	1	0	0	1	1	0	0	0	1	1	1	0	0	0	0	1	0	0	1	1	0	0	1	1	1	0	0	1	1	1	0	0
0	0	1	0	0	1	1	1	0	0	0	0	0	0	1	0	0	1	0	0	1	1	0	1	0	0	1	0	0	0	0	0	0	0
0	0	1	0	0	1	1	1	0	0	1	1	0	0	1	0	0	1	0	0	1	1	0	1	0	0	1	1	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	1	0	0	0	1	1	1	0	1	0	0	1	1	0	1	0	1	1	0	0	1	1	1	0	0
0	0	1	0	1	0	0	0	0	1	1	1	1	0	0	1	0	1	0	0	1	1	0	1	1	0	1	0	0	1	0	1	0	0
0	0	1	0	1	0	0	1	0	1	0	0	1	1	1	0	0	1	0	0	1	1	0	1	1	0	1	1	1	0	1	0	0	0
0	0	1	0	1	0	0	1	0	1	1	1	0	0	1	0	0	1	0	0	1	1	0	1	1	1	1	0	0	0	1	1	0	0
0	0	1	0	1	0	1	0	0	1	0	0	1	1	1	1	0	1	0	0	1	1	1	0	0	0	1	0	0	0	0	0	1	0
0	0	1	0	1	0	1	0	0	1	1	1	0	0	0	1	0	1	0	0	1	1	1	0	0	0	1	1	0	0	0	1	0	0
0	0	1	0	1	0	1	1	0	1	0	0	0	1	1	0	0	1	0	0	1	1	1	0	0	1	1	0	0	1	1	1	1	0
0	0	1	0	1	0	1	1	0	1	1	1	1	0	1	0	0	1	0	0	1	1	1	0	1	0	1	0	0	1	0	1	0	0
0	0	1	0	1	1	0	0	0	1	0	0	1	1	1	0	0	1	0	0	1	1	1	0	1	0	1	1	1	1	0	1	0	0
0	0	1	0	1	1	0	0	0	1	1	1	0	0	0	0	0	1	0	0	1	1	1	0	1	1	1	0	0	0	1	1	0	0
0	0	1	0	1	1	0	1	0	1	0	0	0	1	1	0	0	1	0	0	1	1	1	1	0	0	1	0	0	1	0	0	0	0
0	0	1	0	1	1	0	1	0	1	1	1	1	0	1	0	0	1	0	0	1	1	1	1	0	0	1	1	1	0	0	0	0	0
0	0	1	0	1	1	1	0	0	1	0	0	0	1	1	0	0	1	0	0	1	1	1	1	0	1	1	0	0	0	1	1	0	0
0	0	1	0	1	1	1	0	0	1	1	1	1	0	0	0	0	1	0	0	1	1	1	1	1	0	1	0	0	0	0	1	0	0
0	0	1	0	1	1	1	1	0	1	0	0	1	1	1	0	0	1	0	0	1	1	1	1	1	0	1	1	0	0	0	1	0	0
0	0	1	0	1	1	1	1	0	1	1	1	0	0	1	0	0	1	0	0	1	1	1	1	1	1	1	0	0	1	1	1	0	0

Inputs												Outputs					Inputs												Outputs				
a_{i-1}	b_{i-1}	a_i	b_i	m_i	a_{i+1}	b_{i+1}	m_{i+1}	c^1	c^2	c^3	c^4	s_i	c^1	c^2	c^3	c^4	a_{i-1}	b_{i-1}	a_i	b_i	m_i	a_{i+1}	b_{i+1}	m_{i+1}	c^1	c^2	c^3	c^4	s_i	c^1	c^2	c^3	c^4
0	0	1	1	0	0	0	0	0	0	0	0	1	1	1	1	0	1	0	1	0	0	0	0	0	0	0	0	0	1	0	0	1	0
0	0	1	1	0	0	0	0	0	0	1	1	0	0	0	1	0	1	0	1	0	0	0	0	0	0	0	1	1	0	0	0	1	1
0	0	1	1	0	0	0	1	0	0	0	0	0	1	1	0	0	1	0	1	0	0	0	0	0	1	1	0	0	1	0	0	1	0
0	0	1	1	0	0	0	1	0	0	1	1	1	0	1	0	0	1	0	1	0	0	0	0	1	0	0	0	0	0	0	1	0	0
0	0	1	1	0	0	1	0	0	0	0	0	0	1	1	1	0	1	0	1	0	0	0	0	1	0	0	1	1	1	0	1	1	1
0	0	1	1	0	0	1	0	0	0	1	1	1	0	0	1	0	1	0	1	0	0	0	0	1	1	1	0	0	0	0	1	0	0
0	0	1	1	0	0	1	1	0	0	0	0	1	1	1	0	0	1	0	1	0	0	0	1	0	0	0	0	0	0	0	0	1	0
0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	0	0	1	0	1	0	0	0	1	0	0	0	1	1	1	0	0	1	1
0	0	1	1	0	1	0	0	0	0	0	0	0	1	1	1	0	1	0	1	0	0	0	1	0	1	1	0	0	0	0	0	1	0
0	0	1	1	0	1	0	0	0	0	1	1	1	0	0	1	0	1	0	1	0	0	0	1	1	0	0	0	0	1	0	1	0	0
0	0	1	1	0	1	0	1	0	0	0	0	1	1	1	0	0	1	0	1	0	0	0	1	1	0	0	1	1	0	0	1	1	1
0	0	1	1	0	1	0	1	0	0	1	1	0	0	1	0	0	1	0	1	0	0	0	1	1	1	1	0	0	1	0	1	0	0
0	0	1	1	0	1	1	0	0	0	0	0	1	1	1	0	0	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	0	1	1	0	0	0	1	1	0	0	0	0	0	1	0	1	0	0	1	0	0	0	0	1	1	1	0	0	1	1
0	0	1	1	0	1	1	1	0	0	0	0	0	1	1	0	0	1	0	1	0	0	1	0	0	1	1	0	0	0	0	0	0	0
0	0	1	1	0	1	1	1	0	0	1	1	1	0	1	0	0	1	0	1	0	0	1	0	1	0	0	0	0	1	0	1	0	0
0	0	1	1	1	0	0	0	0	1	0	0	1	1	1	1	0	1	0	1	0	0	1	0	1	0	0	1	1	0	0	1	1	1
0	0	1	1	1	0	0	0	0	1	1	1	1	1	1	1	0	1	0	1	0	0	1	0	1	1	1	0	0	1	0	1	0	0
0	0	1	1	1	0	0	1	0	1	0	0	0	1	1	0	0	1	0	1	0	0	1	1	0	0	0	0	0	1	0	0	0	0
0	0	1	1	1	0	0	1	0	1	1	1	0	1	1	0	0	1	0	1	0	0	1	1	0	0	0	1	1	0	0	0	1	1
0	0	1	1	1	0	1	0	0	1	0	0	0	1	1	1	0	1	0	1	0	0	1	1	0	1	1	0	0	1	0	0	0	0
0	0	1	1	1	0	1	0	0	1	1	1	0	1	1	1	0	1	0	1	0	0	1	1	1	0	0	0	0	0	0	1	0	0
0	0	1	1	1	0	1	1	0	1	0	0	1	1	1	0	0	1	0	1	0	0	1	1	1	0	0	1	1	1	0	1	1	1
0	0	1	1	1	0	1	1	0	1	1	1	1	1	1	0	0	1	0	1	0	0	1	1	1	1	1	0	0	0	0	1	0	0
0	0	1	1	1	1	0	0	0	1	0	0	0	1	1	1	0	1	0	1	0	1	0	0	0	0	1	0	0	1	0	0	1	0
0	0	1	1	1	1	0	0	0	1	1	1	0	1	1	1	0	1	0	1	0	1	0	0	0	0	1	1	1	1	0	0	1	0
0	0	1	1	1	1	0	1	0	1	0	0	1	1	1	0	0	1	0	1	0	1	0	0	0	1	1	0	0	0	1	1	1	0
0	0	1	1	1	1	0	1	0	1	1	1	1	1	1	0	0	1	0	1	0	1	0	0	1	0	1	0	0	0	0	1	0	0
0	0	1	1	1	1	1	0	0	1	0	0	1	1	1	0	0	1	0	1	0	1	0	0	1	0	1	1	1	0	0	1	0	0
0	0	1	1	1	1	1	0	0	1	1	1	1	1	1	0	0	1	0	1	0	1	0	0	1	1	1	0	0	1	1	1	0	0
0	0	1	1	1	1	1	1	0	1	0	0	0	1	1	0	0	1	0	1	0	1	0	1	0	0	1	0	0	0	0	0	1	0
0	0	1	1	1	1	1	1	0	1	1	1	0	1	1	0	0	1	0	1	0	1	0	1	0	0	1	1	1	0	0	0	1	0
0	1	0	0	0	0	0	0	0	0	1	0	1	0	0	1	1	1	0	1	0	1	0	1	0	1	1	0	0	1	1	1	1	0
0	1	0	0	0	0	0	0	0	0	1	1	1	0	0	1	1	1	0	1	0	1	0	1	1	0	1	0	0	1	0	1	0	0
0	1	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	1	0	1	0	1	0	1	1	0	1	1	1	1	0	1	0	0
0	1	0	0	0	0	0	1	0	0	1	0	0	0	1	1	1	1	0	1	0	1	0	1	1	1	1	0	0	0	1	1	0	0
0	1	0	0	0	0	0	1	0	0	1	1	0	0	1	1	1	1	0	1	0	1	1	0	0	0	1	0	0	0	0	0	0	0
0	1	0	0	0	0	0	1	1	1	1	0	1	0	1	0	0	1	0	1	0	1	1	0	0	0	1	1	1	0	0	0	0	0
0	1	0	0	0	0	1	0	0	0	1	0	0	0	0	1	1	1	0	1	0	1	1	0	0	1	1	0	0	1	1	1	0	0
0	1	0	0	0	0	1	0	0	0	1	1	0	0	0	1	1	1	0	1	0	1	1	0	1	0	1	0	0	1	0	1	0	0
0	1	0	0	0	0	1	0	1	1	1	0	1	0	0	0	0	1	0	1	0	1	1	0	1	0	1	1	1	1	0	1	0	0
0	1	0	0	0	0	1	1	0	0	1	0	1	0	1	1	1	1	0	1	0	1	1	0	1	1	1	0	0	0	1	1	0	0
0	1	0	0	0	0	1	1	0	0	1	1	1	0	1	1	1	1	0	1	0	1	1	1	0	0	1	0	0	1	0	0	0	0
0	1	0	0	0	0	1	1	1	1	1	0	0	0	1	0	0	1	0	1	0	1	1	1	0	0	1	1	1	1	0	0	0	0
0	1	0	0	0	1	0	0	0	0	1	0	0	0	0	1	1	1	0	1	0	1	1	1	0	1	1	0	0	0	1	1	0	0
0	1	0	0	0	1	0	0	0	0	1	1	0	0	0	1	1	1	0	1	0	1	1	1	1	0	1	0	0	0	0	1	0	0
0	1	0	0	0	1	0	0	1	1	1	0	1	0	0	0	0	1	0	1	0	1	1	1	1	0	1	1	1	0	0	1	0	0
0	1	0	0	0	1	0	1	0	0	1	0	1	0	1	1	1	1	0	1	0	1	1	1	1	1	1	0	0	1	1	1	0	0
0	1	0	0	0	1	0	1	0	0	1	1	1	0	1	1	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0
0	1	0	0	0	1	0	1	1	1	1	0	0	0	1	0	0	1	0	1	1	0	0	0	0	0	0	1	1	0	0	0	1	0
0	1	0	0	0	1	1	0	0	0	1	0	1	0	0	1	1	1	0	1	1	0	0	0	0	1	1	0	0	1	1	1	1	0
0	1	0	0	0	1	1	0	0	0	1	1	1	0	0	1	1	1	0	1	1	0	0	0	1	0	0	0	0	1	0	1	0	0
0	1	0	0	0	1	1	0	1	1	1	0	0	0	0	0	0	1	0	1	1	0	0	0	1	0	0	1	1	1	0	1	0	0
0	1	0	0	0	1	1	1	0	0	1	0	0	0	1	1	1	1	0	1	1	0	0	0	1	1	1	0	0	0	1	1	0	0
0	1	0	0	0	1	1	1	0	0	1	1	0	0	1	1	1	1	0	1	1	0	0	1	0	0	0	0	0	1	0	0	1	0
0	1	0	0	0	1	1	1	1	1	1	0	1	0	1	0	0	1	0	1	1	0	0	1	0	0	0	1	1	1	0	0	1	0
0	1	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	1	0	1	1	0	0	1	0	1	1	0	0	0	1	1	1	0
0	1	0	0	1	0	0	0	0	1	1	1	1	0	0	1	1	1	0	1	1	0	0	1	1	0	0	0	0	0	0	1	0	0
0	1	0	0	1	0	0	0	1	1	0	0	0	0	0	0	0	1	0	1	1	0	0	1	1	0	0	1	1	0	0	1	0	0
0	1	0	0	1	0	0	1	0	1	0	0	1	0	1	0	0	1	0	1	1	0	0	1	1	1	1	0	0	1	1	1	0	0
0	1	0	0	1	0	0	1	0	1	1	1	0	0	1	1	1	1	0	1	1	0	1	0	0	0	0	0	0	1	0	0	1	0
0	1	0	0	1	0	0	1	1	1	0	0	1	0	1	0	0	1	0	1	1	0	1	0	0	0	0	1	1	1	0	0	1	0
0	1	0	0	1	0	1	0	0	1	0	0	1	0	0	0	0	1	0	1	1	0	1	0	0	1	1	0	0	0	1	1	1	0
0	1	0	0	1	0	1	0	0	1	1	1	0	0	0	1	1	1	0	1	1	0	1	0	1	0	0	0	0	0	0	1	0	0
0	1	0	0	1	0	1	0	1	1	0	0	1	0	0	0	0	1	0	1	1	0	1	0	1	0	0	1	1	0	0	1	0	0
0	1	0	0	1	0	1	1	0	1	0	0	0	0	1	0	0	1	0	1	1	0	1	0	1	1	1	0	0	1	1	1	0	0
0	1	0	0	1	0	1	1	0	1	1	1	1	0	1	1	1	1	0	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	1	0	1	1	1	1	0	0	0	0	1	0	0	1	0	1	1	0	1	1	0	0	0	1	1	0	0	0	0	0
0	1	0	0	1	1	0	0	0	1	0	0	1	0	0	0	0	1	0	1	1	0	1	1	0	1	1	0	0	1	1	1	0	0
0	1	0	0	1	1	0	0	0	1	1	1	0	0	0	1	1	1	0	1	1	0	1	1	1	0	0	0	0	1	0	1	0	0
0	1	0	0	1	1	0	0	1	1	0	0	1	0	0	0	0	1	0	1	1	0	1	1	1	0	0	1	1	1	0	1	0	0
0	1	0	0	1	1	0	1	0	1	0	0	0	0	1	0	0	1	0	1	1	0	1	1	1	1	1	0	0	0	1	1	0	0
0	1	0	0	1	1	0	1	0	1	1	1	1	0	1	1	1	1	0	1	1	1	0	0	0	0	1	0	0	1	1	1	1	0
0	1	0	0	1	1	0	1	1	1	0	0	0	0	1	0	0	1	0	1	1	1	0	0	0	0	1	1	1	0	0	0	1	0
0	1	0	0	1	1	1	0	0	1	0	0	0	0	0	0	0	1	0	1	1	1	0	0	0	1	1	0	0	1	1	1	1	0
0	1	0	0	1	1	1	0	0	1	1	1	1	0	0	1	1	1	0	1	1	1	0	0	1	0	1	0	0	0	1	1	0	0
0	1	0	0	1	1	1	0	1	1	0	0	0	0	0	0	0	1	0	1	1	1	0	0	1	0	1	1	1	1	0	1	0	0
0	1	0	0	1	1	1	1	0	1	0	0	1	0	1	0	0	1	0	1	1	1	0	0	1	1	1	0	0	0	1	1	0	0
0	1	0	0	1	1	1	1	0	1	1	1	0	0	1	1	1	1	0	1	1	1	0	1	0	0	1	0	0	0	1	1	1	0
0	1	0	0	1	1	1	1	1	1	0	0	1	0	1	0	0	1	0	1	1	1	0	1	0	0	1	1	1	1	0	0	1	0
0	1	0	1	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0	1	1	1	0	1	0	1	1	0	0	0	1	1	1	0
0	1	0	1	0	0	0	0	0	0	1	1	0	0	0	1	1	1	0	1	1	1	0	1	1	0	1	0	0	1	1	1	0	0
0	1	0	1	0	0	0	0	1	1	0	0	1	0	0	1	0	1	0	1	1	1	0	1	1	0	1	1	1	0	0	1	0	0
0	1	0	1	0	0	0	1	0	0	0	0	0	0	1	0	0	1	0	1	1	1	0	1	1	1	1	0	0	1	1	1	0	0
0	1	0	1	0	0	0	1	0	0	1	1	1	0	1	1	1	1	0	1	1	1	1	0	0	0	1	0	0	0	1	1	1	0
0	1	0	1	0	0	0	1	1	1	0	0	0	0	1	0	0	1	0	1	1	1	1	0	0	0	1	1	1	1	0	0	1	0
0	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	1	0	0	1	1	0	0	0	1	1	1	0
0	1	0	1	0	0	1	0	0	0	1	1	1	0	0	1	1	1	0	1	1	1	1	0	1	0	1	0	0	1	1	1	0	0
0	1	0	1	0	0	1	0	1	1	0	0	0	0	0	0	0	1	0	1	1	1	1	0	1	0	1	1	1	0	0	1	0	0
0	1	0	1	0	0	1	1	0	0	0	0	1	0	1	0	0	1	0	1	1	1	1	0	1	1	1	0	0	1	1	1	0	0
0	1	0	1	0	0	1	1	0	0	1	1	0	0	1	1	1	1	0	1	1	1	1	1	0	0	1	0	0	1	1	1	0	0
0	1	0	1	0	0	1	1	1	1	0	0	1	0	1	0	0	1	0	1	1	1	1	1	0	0	1	1	1	0	0	0	0	0
0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	1	0	1	0	1	1	1	1	1	0	1	1	0	0	1	1	1	0	0
0	1	0	1	0	1	0	0	0	0	1	1	1	0	0	1	1	1	0	1	1	1	1	1	1	0	1	0	0	0	1	1	0	0
0	1	0	1	0	1	0	0	1	1	0	0	0	0	0	1	0	1	0	1	1	1	1	1	1	0	1	1	1	1	0	1	0	0
0	1	0	1	0	1	0	1	0	0	0	0	1	0	1	0	0	1	0	1	1	1	1	1	1	1	1	0	0	0	1	1	0	0
0	1	0	1	0	1	0	1	0	0	1	1	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0	1	0	1	0	0	1	1
0	1	0	1	0	1	0	1	1	1	0	0	1	0	1	0	0	1	1	0	0	0	0	0	0	1	1	1	0	1	0	0	1	1
0	1	0	1	0	1	1	0	0	0	0	0	1	0	0	0	0	1	1	0	0	0	0	0	1	0	0	1	0	0	0	1	1	1
0	1	0	1	0	1	1	0	0	0	1	1	0	0	0	1	1	1	1	0	0	0	0	0	1	1	1	1	0	0	0	1	1	1
0	1	0	1	0	1	1	0	1	1	0	0	1	0	0	0	0	1	1	0	0	0	0	1	0	0	0	1	0	0	0	0	1	1
0	1	0	1	0	1	1	1	0	0	0	0	0	0	1	0	0	1	1	0	0	0	0	1	0	1	1	1	0	0	0	0	1	1
0	1	0	1	0	1	1	1	0	0	1	1	1	0	1	1	1	1	1	0	0	0	0	1	1	0	0	1	0	1	0	1	1	1
0	1	0	1	0	1	1	1	1	1	0	0	0	0	1	0	0	1	1	0	0	0	0	1	1	1	1	1	0	1	0	1	1	1
0	1	0	1	1	0	0	0	0	1	0	0	1	0	0	1	0	1	1	0	0	0	1	0	0	0	0	1	0	0	0	0	1	1

Inputs												Outputs					Inputs												Outputs				
a_{i-1}	b_{i-1}	a_i	b_i	m_i	a_{i+1}	b_{i+1}	m_{i+1}	c^1	c^2	c^3	c^4	s_i	c^1	c^2	c^3	c^4	a_{i-1}	b_{i-1}	a_i	b_i	m_i	a_{i+1}	b_{i+1}	m_{i+1}	c^1	c^2	c^3	c^4	s_i	c^1	c^2	c^3	c^4
0	1	0	1	1	0	0	0	0	1	1	1	1	0	0	1	0	1	1	0	0	0	1	0	0	1	1	1	0	0	0	0	1	1
0	1	0	1	1	0	0	0	1	1	0	0	0	1	1	1	0	1	1	0	0	0	1	0	1	0	0	1	0	1	0	1	1	1
0	1	0	1	1	0	0	1	0	1	0	0	0	0	1	0	0	1	1	0	0	0	1	0	1	1	1	1	0	1	0	1	1	1
0	1	0	1	1	0	0	1	0	1	1	1	0	0	1	0	0	1	1	0	0	0	1	1	0	0	0	1	0	1	0	0	1	1
0	1	0	1	1	0	0	1	1	1	0	0	1	1	1	0	0	1	1	0	0	0	1	1	0	1	1	1	0	1	0	0	1	1
0	1	0	1	1	0	1	0	0	1	0	0	0	0	0	0	0	1	1	0	0	0	1	1	1	0	0	1	0	0	0	1	1	1
0	1	0	1	1	0	1	0	0	1	1	1	0	0	0	0	0	1	1	0	0	0	1	1	1	1	1	1	0	0	0	1	1	1
0	1	0	1	1	0	1	0	1	1	0	0	1	1	1	0	0	1	1	0	0	1	0	0	0	0	1	0	0	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	0	0	1	0	1	0	0	1	1	0	0	1	0	0	0	1	1	0	0	0	0	0	0	0
0	1	0	1	1	0	1	1	0	1	1	1	1	0	1	0	0	1	1	0	0	1	0	0	1	0	1	0	0	0	0	1	1	1
0	1	0	1	1	0	1	1	1	1	0	0	0	1	1	0	0	1	1	0	0	1	0	0	1	1	1	0	0	1	0	1	0	0
0	1	0	1	1	1	0	0	0	1	0	0	0	0	0	1	0	1	1	0	0	1	0	1	0	0	1	0	0	0	0	0	1	1
0	1	0	1	1	1	0	0	0	1	1	1	0	0	0	1	0	1	1	0	0	1	0	1	0	1	1	0	0	1	0	0	0	0
0	1	0	1	1	1	0	0	1	1	0	0	1	1	1	1	0	1	1	0	0	1	0	1	1	0	1	0	0	1	0	1	1	1
0	1	0	1	1	1	0	1	0	1	0	0	1	0	1	0	0	1	1	0	0	1	0	1	1	1	1	0	0	0	0	1	0	0
0	1	0	1	1	1	0	1	0	1	1	1	1	0	1	0	0	1	1	0	0	1	1	0	0	0	1	0	0	0	0	0	1	1
0	1	0	1	1	1	0	1	1	1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	0	0	0	0
0	1	0	1	1	1	1	0	0	1	0	0	1	0	0	0	0	1	1	0	0	1	1	0	1	0	1	0	0	1	0	1	1	1
0	1	0	1	1	1	1	0	0	1	1	1	1	0	0	0	0	1	1	0	0	1	1	0	1	1	1	0	0	0	0	1	0	0
0	1	0	1	1	1	1	0	1	1	0	0	0	1	1	0	0	1	1	0	0	1	1	1	0	0	1	0	0	1	0	0	1	1
0	1	0	1	1	1	1	1	0	1	0	0	0	0	1	0	0	1	1	0	0	1	1	1	0	1	1	0	0	0	0	0	0	0
0	1	0	1	1	1	1	1	0	1	1	1	0	0	1	0	0	1	1	0	0	1	1	1	1	0	1	0	0	0	0	1	1	1
0	1	0	1	1	1	1	1	1	1	0	0	1	1	1	0	0	1	1	0	0	1	1	1	1	1	1	0	0	1	0	1	0	0
0	1	1	0	0	0	0	0	0	0	1	0	1	0	0	1	0	1	1	0	1	0	0	0	0	0	0	1	0	0	0	0	1	1
0	1	1	0	0	0	0	0	0	0	1	1	0	0	0	1	1	1	1	0	1	0	0	0	0	1	1	1	0	1	0	0	1	0
0	1	1	0	0	0	0	0	1	1	1	0	1	0	0	1	0	1	1	0	1	0	0	0	1	0	0	1	0	1	0	1	1	1
0	1	1	0	0	0	0	1	0	0	1	0	0	0	1	0	0	1	1	0	1	0	0	0	1	1	1	1	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	1	1	1	0	1	1	1	1	1	0	1	0	0	1	0	0	0	1	0	1	0	0	1	1
0	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0	1	1	0	1	0	0	1	0	1	1	1	0	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	1	0	0	0	0	1	0	1	1	0	1	0	0	1	1	0	0	1	0	0	0	1	1	1
0	1	1	0	0	0	1	0	0	0	1	1	1	0	0	1	1	1	1	0	1	0	0	1	1	1	1	1	0	1	0	1	0	0
0	1	1	0	0	0	1	0	1	1	1	0	0	0	0	1	0	1	1	0	1	0	1	0	0	0	0	1	0	1	0	0	1	1
0	1	1	0	0	0	1	1	0	0	1	0	1	0	1	0	0	1	1	0	1	0	1	0	0	1	1	1	0	0	0	0	1	0
0	1	1	0	0	0	1	1	0	0	1	1	0	0	1	1	1	1	1	0	1	0	1	0	1	0	0	1	0	0	0	1	1	1
0	1	1	0	0	0	1	1	1	1	1	0	1	0	1	0	0	1	1	0	1	0	1	0	1	1	1	1	0	1	0	1	0	0
0	1	1	0	0	1	0	0	0	0	1	0	0	0	0	0	0	1	1	0	1	0	1	1	0	0	0	1	0	0	0	0	1	1
0	1	1	0	0	1	0	0	0	0	1	1	1	0	0	1	1	1	1	0	1	0	1	1	0	1	1	1	0	1	0	0	0	0
0	1	1	0	0	1	0	0	1	1	1	0	0	0	0	0	0	1	1	0	1	0	1	1	1	0	0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1	0	0	1	0	1	0	1	0	0	1	1	0	1	0	1	1	1	1	1	1	0	0	0	1	0	0
0	1	1	0	0	1	0	1	0	0	1	1	0	0	1	1	1	1	1	0	1	1	0	0	0	0	1	0	0	1	0	0	1	0
0	1	1	0	0	1	0	1	1	1	1	0	1	0	1	0	0	1	1	0	1	1	0	0	0	1	1	0	0	1	0	0	1	0
0	1	1	0	0	1	1	0	0	0	1	0	1	0	0	0	0	1	1	0	1	1	0	0	1	0	1	0	0	0	0	1	0	0
0	1	1	0	0	1	1	0	0	0	1	1	0	0	0	1	1	1	1	0	1	1	0	0	1	1	1	0	0	0	0	1	0	0
0	1	1	0	0	1	1	0	1	1	1	0	1	0	0	0	0	1	1	0	1	1	0	1	0	0	1	0	0	0	0	0	0	0
0	1	1	0	0	1	1	1	0	0	1	0	0	0	1	0	0	1	1	0	1	1	0	1	0	1	1	0	0	0	0	0	0	0
0	1	1	0	0	1	1	1	0	0	1	1	1	0	1	1	1	1	1	0	1	1	0	1	1	0	1	0	0	1	0	1	0	0
0	1	1	0	0	1	1	1	1	1	1	0	0	0	1	0	0	1	1	0	1	1	0	1	1	1	1	0	0	1	0	1	0	0
0	1	1	0	1	0	0	0	0	1	0	0	1	0	0	1	0	1	1	0	1	1	1	0	0	0	1	0	0	0	0	0	1	0
0	1	1	0	1	0	0	0	0	1	1	1	1	0	0	1	0	1	1	0	1	1	1	0	0	1	1	0	0	0	0	0	1	0
0	1	1	0	1	0	0	0	1	1	0	0	0	1	1	1	0	1	1	0	1	1	1	0	1	0	1	0	0	1	0	1	0	0
0	1	1	0	1	0	0	1	0	1	0	0	0	0	1	0	0	1	1	0	1	1	1	0	1	1	1	0	0	1	0	1	0	0
0	1	1	0	1	0	0	1	0	1	1	1	0	0	1	0	0	1	1	0	1	1	1	1	0	0	1	0	0	1	0	0	0	0
0	1	1	0	1	0	0	1	1	1	0	0	1	1	1	0	0	1	1	0	1	1	1	1	0	1	1	0	0	1	0	0	0	0
0	1	1	0	1	0	1	0	0	1	0	0	0	0	0	1	0	1	1	0	1	1	1	1	1	0	1	0	0	0	0	1	0	0
0	1	1	0	1	0	1	0	0	1	1	1	0	0	0	1	0	1	1	0	1	1	1	1	1	1	1	0	0	0	0	1	0	0
0	1	1	0	1	0	1	0	1	1	0	0	1	1	1	1	0	1	1	1	0	0	0	0	0	0	0	1	0	0	0	0	1	1
0	1	1	0	1	0	1	1	0	1	0	0	1	0	1	0	0	1	1	1	0	0	0	0	0	1	1	1	0	1	0	0	1	0
0	1	1	0	1	0	1	1	0	1	1	1	1	0	1	0	0	1	1	1	0	0	0	0	1	0	0	1	0	1	0	1	1	1
0	1	1	0	1	0	1	1	1	1	0	0	0	1	1	0	0	1	1	1	0	0	0	0	1	1	1	1	0	0	0	1	0	0
0	1	1	0	1	1	0	0	0	1	0	0	0	0	0	0	0	1	1	1	0	0	0	1	0	0	0	1	0	1	0	0	1	1
0	1	1	0	1	1	0	0	0	1	1	1	0	0	0	0	0	1	1	1	0	0	0	1	0	1	1	1	0	0	0	0	1	0
0	1	1	0	1	1	0	0	1	1	0	0	1	1	1	0	0	1	1	1	0	0	0	1	1	0	0	1	0	0	0	1	1	1
0	1	1	0	1	1	0	1	0	1	0	0	1	0	1	0	0	1	1	1	0	0	0	1	1	1	1	1	0	1	0	1	0	0
0	1	1	0	1	1	0	1	0	1	1	1	1	0	1	0	0	1	1	1	0	0	1	0	0	0	0	1	0	1	0	0	1	1
0	1	1	0	1	1	0	1	1	1	0	0	0	1	1	0	0	1	1	1	0	0	1	0	0	1	1	1	0	0	0	0	0	0
0	1	1	0	1	1	1	0	0	1	0	0	1	0	0	0	0	1	1	1	0	0	1	0	1	0	0	1	0	0	0	1	1	1
0	1	1	0	1	1	1	0	0	1	1	1	1	0	0	0	0	1	1	1	0	0	1	0	1	1	1	1	0	1	0	1	0	0
0	1	1	0	1	1	1	0	1	1	0	0	0	1	1	0	0	1	1	1	0	0	1	1	0	0	0	1	0	0	0	0	1	1
0	1	1	0	1	1	1	1	0	1	0	0	0	0	1	0	0	1	1	1	0	0	1	1	0	1	1	1	0	1	0	0	0	0
0	1	1	0	1	1	1	1	0	1	1	1	0	0	1	0	0	1	1	1	0	0	1	1	1	0	0	1	0	1	0	1	1	1
0	1	1	0	1	1	1	1	1	1	0	0	1	1	1	0	0	1	1	1	0	0	1	1	1	1	1	1	0	0	0	1	0	0
0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	0	1	0	0	0	0	1	0	0	1	0	0	1	0
0	1	1	1	0	0	0	0	0	0	1	1	0	0	0	1	0	1	1	1	0	1	0	0	0	1	1	0	0	1	0	0	1	0
0	1	1	1	0	0	0	0	1	1	0	0	1	1	1	1	0	1	1	1	0	1	0	0	1	0	1	0	0	0	0	1	0	0
0	1	1	1	0	0	0	1	0	0	0	0	1	0	1	0	0	1	1	1	0	1	0	0	1	1	1	0	0	0	0	1	0	0
0	1	1	1	0	0	0	1	0	0	1	1	1	0	1	0	0	1	1	1	0	1	0	1	0	0	1	0	0	0	0	0	1	0
0	1	1	1	0	0	0	1	1	1	0	0	0	1	1	0	0	1	1	1	0	1	0	1	0	1	1	0	0	0	0	0	1	0
0	1	1	1	0	0	1	0	0	0	0	0	1	0	0	1	0	1	1	1	0	1	0	1	1	0	1	0	0	1	0	1	0	0
0	1	1	1	0	0	1	0	0	0	1	1	1	0	0	1	0	1	1	1	0	1	0	1	1	1	1	0	0	1	0	1	0	0
0	1	1	1	0	0	1	0	1	1	0	0	0	1	1	1	0	1	1	1	0	1	1	0	0	0	1	0	0	0	0	0	0	0
0	1	1	1	0	0	1	1	0	0	0	0	0	0	1	0	0	1	1	1	0	1	1	0	0	1	1	0	0	0	0	0	0	0
0	1	1	1	0	0	1	1	0	0	1	1	0	0	1	0	0	1	1	1	0	1	1	0	1	0	1	0	0	1	0	1	0	0
0	1	1	1	0	0	1	1	1	1	0	0	1	1	1	0	0	1	1	1	0	1	1	0	1	1	1	0	0	1	0	1	0	0
0	1	1	1	0	1	0	0	0	0	0	0	1	0	0	1	0	1	1	1	0	1	1	1	0	0	1	0	0	1	0	0	0	0
0	1	1	1	0	1	0	0	0	0	1	1	1	0	0	1	0	1	1	1	0	1	1	1	0	1	1	0	0	1	0	0	0	0
0	1	1	1	0	1	0	0	1	1	0	0	0	1	1	1	0	1	1	1	0	1	1	1	1	0	1	0	0	0	0	1	0	0
0	1	1	1	0	1	0	1	0	0	0	0	0	0	1	0	0	1	1	1	0	1	1	1	1	1	1	0	0	0	0	1	0	0
0	1	1	1	0	1	0	1	0	0	1	1	0	0	1	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0
0	1	1	1	0	1	0	1	1	1	0	0	1	1	1	0	0	1	1	1	1	0	0	0	0	1	1	0	0	0	0	0	1	0
0	1	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	1	0	0	0	0	1	0	1	0	0
0	1	1	1	0	1	1	0	0	0	1	1	0	0	0	0	0	1	1	1	1	0	0	0	1	1	1	0	0	1	0	1	0	0
0	1	1	1	0	1	1	0	1	1	0	0	1	1	1	0	0	1	1	1	1	0	0	1	0	0	0	0	0	1	0	0	1	0
0	1	1	1	0	1	1	1	0	0	0	0	1	0	1	0	0	1	1	1	1	0	0	1	0	1	1	0	0	1	0	0	1	0
0	1	1	1	0	1	1	1	0	0	1	1	1	0	1	0	0	1	1	1	1	0	0	1	1	0	0	0	0	0	0	1	0	0
0	1	1	1	0	1	1	1	1	1	0	0	0	1	1	0	0	1	1	1	1	0	0	1	1	1	1	0	0	0	0	1	0	0
0	1	1	1	1	0	0	0	0	1	0	0	1	1	1	1	0	1	1	1	1	0	1	0	0	0	0	0	0	1	0	0	1	0
0	1	1	1	1	0	0	0	0	1	1	1	0	0	0	1	0	1	1	1	1	0	1	0	0	1	1	0	0	1	0	0	1	0
0	1	1	1	1	0	0	0	1	1	0	0	1	1	1	1	0	1	1	1	1	0	1	0	1	0	0	0	0	0	0	1	0	0
0	1	1	1	1	0	0	1	0	1	0	0	0	1	1	0	0	1	1	1	1	0	1	0	1	1	1	0	0	0	0	1	0	0
0	1	1	1	1	0	0	1	0	1	1	1	1	0	1	0	0	1	1	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0
0	1	1	1	1	0	0	1	1	1	0	0	0	1	1	0	0	1	1	1	1	0	1	1	0	1	1	0	0	0	0	0	0	0
0	1	1	1	1	0	1	0	0	1	0	0	0	1	1	1	0	1	1	1	1	0	1	1	1	0	0	0	0	1	0	1	0	0
0	1	1	1	1	0	1	0	0	1	1	1	1	0	0	1	0	1	1	1	1	0	1	1	1	1	1	0	0	1	0	1	0	0
0	1	1	1	1	0	1	0	1	1	0	0	0	1	1	1	0	1	1	1	1	1	0	0	0	0	1	0	0	0	0	0	1	0

Inputs												Outputs					Inputs												Outputs				
a_{i-1}	b_{i-1}	a_i	b_i	m_i	a_{i+1}	b_{i+1}	m_{i+1}	c^1	c^2	c^3	c^4	s_i	c^1	c^2	c^3	c^4	a_{i-1}	b_{i-1}	a_i	b_i	m_i	a_{i+1}	b_{i+1}	m_{i+1}	c^1	c^2	c^3	c^4	s_i	c^1	c^2	c^3	c^4
0	1	1	1	1	0	1	1	0	1	0	0	1	1	1	0	0	1	1	1	1	1	0	0	0	1	1	0	0	1	1	1	1	0
0	1	1	1	1	0	1	1	0	1	1	1	0	0	1	0	0	1	1	1	1	1	0	0	1	0	1	0	0	1	0	1	0	0
0	1	1	1	1	0	1	1	1	1	0	0	1	1	1	0	0	1	1	1	1	1	0	0	1	1	1	0	0	0	1	1	0	0
0	1	1	1	1	1	0	0	0	1	0	0	0	1	1	1	0	1	1	1	1	1	0	1	0	0	1	0	0	1	0	0	1	0
0	1	1	1	1	1	0	0	0	1	1	1	1	0	0	1	0	1	1	1	1	1	0	1	0	1	1	0	0	0	1	1	1	0
0	1	1	1	1	1	0	0	1	1	0	0	0	1	1	1	0	1	1	1	1	1	0	1	1	0	1	0	0	0	0	1	0	0
0	1	1	1	1	1	0	1	0	1	0	0	1	1	1	0	0	1	1	1	1	1	0	1	1	1	1	0	0	1	1	1	0	0
0	1	1	1	1	1	0	1	0	1	1	1	0	0	1	0	0	1	1	1	1	1	1	0	0	0	1	0	0	1	0	0	1	0
0	1	1	1	1	1	0	1	1	1	0	0	1	1	1	0	0	1	1	1	1	1	1	0	0	1	1	0	0	0	1	1	1	0
0	1	1	1	1	1	1	0	0	1	0	0	1	1	1	0	0	1	1	1	1	1	1	0	1	0	1	0	0	0	0	1	0	0
0	1	1	1	1	1	1	0	0	1	1	1	0	0	0	0	0	1	1	1	1	1	1	0	1	1	1	0	0	1	1	1	0	0
0	1	1	1	1	1	1	0	1	1	0	0	1	1	1	0	0	1	1	1	1	1	1	1	0	0	1	0	0	0	0	0	0	0
0	1	1	1	1	1	1	1	0	1	0	0	0	1	1	0	0	1	1	1	1	1	1	1	0	1	1	0	0	1	1	1	0	0
0	1	1	1	1	1	1	1	0	1	1	1	1	0	1	0	0	1	1	1	1	1	1	1	1	0	1	0	0	1	0	1	0	0
0	1	1	1	1	1	1	1	1	1	0	0	0	1	1	0	0	1	1	1	1	1	1	1	1	1	1	0	0	0	1	1	0	0

Symmetrically, as soon as the results of **table 4** and **table 5** and taking account of the recent results [3-14], we can define the following truth table:

TABLE 7
MODULAR FULL ADDER

Inputs												Outputs					Inputs												Outputs				
a_{i-1}	m_{i-1}	a_i	b_i	m_i	a_{i+1}	b_{i+1}	m_{i+1}	c^1	c^2	c^3	c^4	s_i	c^1	c^2	c^3	c^4	a_{i-1}	m_{i-1}	a_i	b_i	m_i	a_{i+1}	b_{i+1}	m_{i+1}	c^1	c^2	c^3	c^4	s_i	c^1	c^2	c^3	c^4
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	1	1	1	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0
0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	1	0	0	1	1	1	0	0	0	0
0	0	0	0	0	0	0	1	0	0	1	1	0	0	0	1	1	1	0	0	0	0	0	1	0	0	0	0	0	1	0	1	0	0
0	0	0	0	0	0	0	1	1	1	0	0	1	0	0	0	0	1	0	0	0	0	0	1	0	0	0	1	1	1	0	1	0	0
0	0	0	0	0	0	1	0	0	0	0	0	1	0	1	0	0	1	0	0	0	0	0	1	1	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	1	0	0	0	1	1	0	0	1	1	1	1	0	0	0	0	0	1	1	0	0	1	1	0	0	1	0	0
0	0	0	0	0	0	1	0	1	1	0	0	1	0	1	0	0	1	0	0	0	0	1	0	0	0	0	0	0	1	0	0	1	0
0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	0	0	1	0	0	0	0	1	0	0	0	0	1	1	1	0	0	1	0
0	0	0	0	0	0	1	1	0	0	1	1	1	0	1	1	1	1	0	0	0	0	1	0	1	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	1	1	1	0	0	0	0	1	0	0	1	0	0	0	0	1	0	1	0	0	1	1	0	0	0	1	0
0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	1	0	1	0	0	0	0	1	1	0	0	0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	0	0	0	1	1	0	0	0	1	1	1	0	0	0	0	1	1	0	0	0	1	1	0	0	1	0	0
0	0	0	0	0	1	0	0	1	1	0	0	1	0	0	1	0	1	0	0	0	0	1	1	1	0	0	0	0	1	0	1	0	0
0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	1	0	1	0	0	0	0	1	1	1	0	0	1	1	1	0	1	0	0
0	0	0	0	0	1	0	1	0	0	1	1	1	0	0	1	1	1	0	0	0	1	0	0	0	0	0	0	0	0	1	1	1	0
0	0	0	0	0	1	0	1	1	1	0	0	0	0	0	1	0	1	0	0	0	1	0	0	0	0	0	1	1	1	0	0	1	0
0	0	0	0	0	1	1	0	0	0	0	0	0	0	1	0	0	1	0	0	0	1	0	0	1	0	0	0	0	1	1	1	0	0
0	0	0	0	0	1	1	0	0	0	1	1	1	0	1	1	1	1	0	0	0	1	0	0	1	0	0	1	1	0	0	0	0	0
0	0	0	0	0	1	1	0	1	1	0	0	0	0	1	0	0	1	0	0	0	1	0	1	0	0	0	0	0	1	1	1	0	0
0	0	0	0	0	1	1	1	0	0	0	0	1	0	1	0	0	1	0	0	0	1	0	1	0	0	0	1	1	0	0	1	0	0
0	0	0	0	0	1	1	1	0	0	1	1	0	0	1	1	1	1	0	0	0	1	0	1	1	0	0	0	0	0	1	1	0	0
0	0	0	0	0	1	1	1	1	1	0	0	1	0	1	0	0	1	0	0	0	1	0	1	1	0	0	1	1	1	0	1	0	0
0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	1	0	1	0	0	0	1	1	0	0	0	0	0	0	1	1	1	1	0
0	0	0	0	1	0	0	0	0	0	1	1	1	0	0	1	0	1	0	0	0	1	1	0	0	0	0	1	1	0	0	0	1	0
0	0	0	0	1	0	0	0	1	1	0	0	0	1	1	1	0	1	0	0	0	1	1	0	1	0	0	0	0	0	1	1	1	0
0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	1	1	0	1	0	0	1	1	1	0	0	1	0
0	0	0	0	1	0	0	1	0	0	1	1	0	0	0	0	0	1	0	0	0	1	1	1	0	0	0	0	0	0	1	1	0	0
0	0	0	0	1	0	0	1	1	1	0	0	1	1	1	0	0	1	0	0	0	1	1	1	0	0	0	1	1	1	0	1	0	0
0	0	0	0	1	0	1	0	0	0	0	0	0	0	1	0	0	1	0	0	0	1	1	1	1	0	0	0	0	1	1	1	0	0
0	0	0	0	1	0	1	0	0	0	1	1	0	0	1	0	0	1	0	0	0	1	1	1	1	0	0	1	1	0	0	1	0	0
0	0	0	0	1	0	1	0	1	1	0	0	1	1	1	0	0	1	0	0	1	0	0	0	0	0	1	0	0	1	1	1	0	0
0	0	0	0	1	0	1	1	0	0	0	0	1	0	1	0	0	1	0	0	1	0	0	0	0	0	1	1	1	0	0	0	0	0
0	0	0	0	1	0	1	1	0	0	1	1	1	0	1	0	0	1	0	0	1	0	0	0	1	0	1	0	0	0	1	1	0	0
0	0	0	0	1	0	1	1	1	1	0	0	0	1	1	0	0	1	0	0	1	0	0	0	1	0	1	1	1	1	0	0	0	0
0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1	0	1	0	0	1	0	0	1	0	0	1	0	0	0	1	1	0	0
0	0	0	0	1	1	0	0	0	0	1	1	0	0	0	1	0	1	0	0	1	0	0	1	0	0	1	1	1	1	0	1	0	0
0	0	0	0	1	1	0	0	1	1	0	0	1	1	1	1	0	1	0	0	1	0	0	1	1	0	1	0	0	1	1	1	0	0
0	0	0	0	1	1	0	1	0	0	0	0	1	0	0	1	0	1	0	0	1	0	0	1	1	0	1	1	1	0	0	1	0	0
0	0	0	0	1	1	0	1	0	0	1	1	1	0	0	1	0	1	0	0	1	0	1	0	0	0	1	0	0	0	1	1	1	0
0	0	0	0	1	1	0	1	1	1	0	0	0	1	1	1	0	1	0	0	1	0	1	0	0	0	1	1	1	1	0	0	1	0
0	0	0	0	1	1	1	0	0	0	0	0	1	0	1	0	0	1	0	0	1	0	1	0	1	0	1	0	0	1	1	1	1	0
0	0	0	0	1	1	1	0	0	0	1	1	1	0	1	0	0	1	0	0	1	0	1	0	1	0	1	1	1	0	0	0	1	0
0	0	0	0	1	1	1	0	1	1	0	0	0	1	1	0	0	1	0	0	1	0	1	1	0	0	1	0	0	1	1	1	0	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	1	0	0	1	0	0	1	0	1	1	0	0	1	1	1	0	0	1	0	0
0	0	0	0	1	1	1	1	0	0	1	1	0	0	1	0	0	1	0	0	1	0	1	1	1	0	1	0	0	0	1	1	0	0
0	0	0	0	1	1	1	1	1	1	0	0	1	1	1	0	0	1	0	0	1	0	1	1	1	0	1	1	1	1	0	1	0	0
0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	1	1	0	0	0	0	1	0	0	0	1	1	1	0
0	0	0	1	0	0	0	0	0	1	1	1	0	0	0	0	0	1	0	0	1	1	0	0	0	0	1	1	1	0	1	1	1	0
0	0	0	1	0	0	0	0	1	1	0	0	1	1	1	0	0	1	0	0	1	1	0	0	1	0	1	0	0	1	1	1	0	0
0	0	0	1	0	0	0	1	0	1	0	0	1	0	0	0	0	1	0	0	1	1	0	0	1	0	1	1	1	1	1	1	0	0
0	0	0	1	0	0	0	1	0	1	1	1	1	0	0	0	0	1	0	0	1	1	0	1	0	0	1	0	0	1	1	1	0	0
0	0	0	1	0	0	0	1	1	1	0	0	0	1	1	0	0	1	0	0	1	1	0	1	0	0	1	1	1	1	1	1	0	0
0	0	0	1	0	0	1	0	0	1	0	0	1	0	1	0	0	1	0	0	1	1	0	1	1	0	1	0	0	0	1	1	0	0
0	0	0	1	0	0	1	0	0	1	1	1	1	0	1	0	0	1	0	0	1	1	0	1	1	0	1	1	1	0	1	1	0	0
0	0	0	1	0	0	1	0	1	1	0	0	0	1	1	0	0	1	0	0	1	1	1	0	0	0	1	0	0	1	1	1	1	0
0	0	0	1	0	0	1	1	0	1	0	0	0	0	1	0	0	1	0	0	1	1	1	0	0	0	1	1	1	1	1	1	1	0
0	0	0	1	0	0	1	1	0	1	1	1	0	0	1	0	0	1	0	0	1	1	1	0	1	0	1	0	0	0	1	1	1	0
0	0	0	1	0	0	1	1	1	1	0	0	1	1	1	0	0	1	0	0	1	1	1	0	1	0	1	1	1	0	1	1	1	0
0	0	0	1	0	1	0	0	0	1	0	0	1	0	0	1	0	1	0	0	1	1	1	1	0	0	1	0	0	0	1	1	0	0
0	0	0	1	0	1	0	0	0	1	1	1	1	0	0	1	0	1	0	0	1	1	1	1	0	0	1	1	1	0	1	1	0	0
0	0	0	1	0	1	0	0	1	1	0	0	0	1	1	1	0	1	0	0	1	1	1	1	1	0	1	0	0	1	1	1	0	0
0	0	0	1	0	1	0	1	0	1	0	0	0	0	0	1	0	1	0	0	1	1	1	1	1	0	1	1	1	1	1	1	0	0
0	0	0	1	0	1	0	1	0	1	1	1	0	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	1	0	1	0	1	1	1	0	0	1	1	1	1	0	1	0	1	0	0	0	0	0	0	0	1	1	0	0	0	1	1
0	0	0	1	0	1	1	0	0	1	0	0	0	0	1	0	0	1	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	0	0	1	0	1	1	0	0	1	1	1	0	0	1	0	0	1	0	1	0	0	0	0	1	0	0	1	1	1	0	0	1	1
0	0	0	1	0	1	1	0	1	1	0	0	1	1	1	0	0	1	0	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0
0	0	0	1	0	1	1	1	0	1	0	0	1	0	1	0	0	1	0	1	0	0	0	1	0	0	0	1	1	1	0	1	1	1
0	0	0	1	0	1	1	1	0	1	1	1	1	0	1	0	0	1	0	1	0	0	0	1	1	0	0	0	0	1	0	1	0	0
0	0	0	1	0	1	1	1	1	1	0	0	0	1	1	0	0	1	0	1	0	0	0	1	1	0	0	1	1	0	0	1	1	1
0	0	0	1	1	0	0	0	0	1	0	0	0	1	1	1	0	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	1	0	1	0	1	0	0	1	0	0	0	0	1	1	1	0	0	1	1
0	0	0	1	1	0	0	0	1	1	0	0	0	1	1	1	0	1	0	1	0	0	1	0	1	0	0	0	0	1	0	0	0	0

Inputs												Outputs					Inputs												Outputs				
a_{i-1}	m_{i-1}	a_i	b_i	m_i	a_{i+1}	b_{i+1}	m_{i+1}	c^1	c^2	c^3	c^4	s_i	c^1	c^2	c^3	c^4	a_{i-1}	m_{i-1}	a_i	b_i	m_i	a_{i+1}	b_{i+1}	m_{i+1}	c^1	c^2	c^3	c^4	s_i	c^1	c^2	c^3	c^4
0	0	0	1	1	0	0	1	0	1	0	0	1	1	1	0	0	1	0	1	0	0	1	0	1	0	0	1	1	0	0	0	1	1
0	0	0	1	1	0	0	1	0	1	1	1	0	0	0	0	0	1	0	1	0	0	1	1	0	0	0	0	0	1	0	1	0	0
0	0	0	1	1	0	0	1	1	1	0	0	1	1	1	0	0	1	0	1	0	0	1	1	0	0	0	1	1	0	0	1	1	1
0	0	0	1	1	0	1	0	0	1	0	0	1	1	1	0	0	1	0	1	0	0	1	1	1	0	0	0	0	0	0	1	0	0
0	0	0	1	1	0	1	0	0	1	1	1	0	0	1	0	0	1	0	1	0	0	1	1	1	0	0	1	1	1	0	1	1	1
0	0	0	1	1	0	1	0	1	1	0	0	1	1	1	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	1	0
0	0	0	1	1	0	1	1	0	1	0	0	0	1	1	0	0	1	0	1	0	1	0	0	0	0	0	1	1	0	0	0	1	0
0	0	0	1	1	0	1	1	0	1	1	1	1	0	1	0	0	1	0	1	0	1	0	0	1	0	0	0	0	1	0	0	0	0
0	0	0	1	1	0	1	1	1	1	0	0	0	1	1	0	0	1	0	1	0	1	0	0	1	0	0	1	1	1	0	0	0	0
0	0	0	1	1	1	0	0	0	1	0	0	1	1	1	1	0	1	0	1	0	1	0	1	0	0	0	0	0	1	0	1	0	0
0	0	0	1	1	1	0	0	0	1	1	1	0	0	0	1	0	1	0	1	0	1	0	1	0	0	0	1	1	1	0	1	0	0
0	0	0	1	1	1	0	0	1	1	0	0	1	1	1	1	0	1	0	1	0	1	0	1	1	0	0	0	0	0	0	1	0	0
0	0	0	1	1	1	0	1	0	1	0	0	0	1	1	1	0	1	0	1	0	1	0	1	1	0	0	1	1	0	0	1	0	0
0	0	0	1	1	1	0	1	0	1	1	1	1	0	0	1	0	1	0	1	0	1	1	0	0	0	0	0	0	1	0	0	1	0
0	0	0	1	1	1	0	1	1	1	0	0	0	1	1	1	0	1	0	1	0	1	1	0	0	0	0	1	1	1	0	0	1	0
0	0	0	1	1	1	1	0	0	1	0	0	0	1	1	0	0	1	0	1	0	1	1	0	1	0	0	0	0	0	0	0	0	0
0	0	0	1	1	1	1	0	0	1	1	1	1	0	1	0	0	1	0	1	0	1	1	0	1	0	0	1	1	0	0	0	0	0
0	0	0	1	1	1	1	0	1	1	0	0	0	1	1	0	0	1	0	1	0	1	1	1	0	0	0	0	0	0	0	1	0	0
0	0	0	1	1	1	1	1	0	1	0	0	1	1	1	0	0	1	0	1	0	1	1	1	0	0	0	1	1	0	0	1	0	0
0	0	0	1	1	1	1	1	0	1	1	1	0	0	1	0	0	1	0	1	0	1	1	1	1	0	0	0	0	1	0	1	0	0
0	0	0	1	1	1	1	1	1	1	0	0	1	1	1	0	0	1	0	1	0	1	1	1	1	0	0	1	1	1	0	1	0	0
0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	1	1	1	0	1	1	0	0	0	0	0	1	0	0	1	0	0	0	0
0	0	1	0	0	0	0	0	0	0	1	1	0	0	0	1	1	1	0	1	1	0	0	0	0	0	1	1	1	1	0	0	0	0
0	0	1	0	0	0	0	0	1	1	1	0	1	0	0	0	0	1	0	1	1	0	0	0	1	0	1	0	0	0	0	0	0	0
0	0	1	0	0	0	0	1	0	0	1	0	1	0	0	1	1	1	0	1	1	0	0	0	1	0	1	1	1	0	0	0	0	0
0	0	1	0	0	0	0	1	0	0	1	1	1	0	0	1	1	1	0	1	1	0	0	1	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	1	1	1	1	0	0	0	0	0	0	1	0	1	1	0	0	1	0	0	1	1	1	0	0	1	0	0
0	0	1	0	0	0	1	0	0	0	1	0	1	0	1	1	1	1	0	1	1	0	0	1	1	0	1	0	0	1	0	1	0	0
0	0	1	0	0	0	1	0	0	0	1	1	1	0	1	1	1	1	0	1	1	0	0	1	1	0	1	1	1	1	0	1	0	0
0	0	1	0	0	0	1	0	1	1	1	0	0	0	1	0	0	1	0	1	1	0	1	0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	0	0	1	1	0	0	1	0	0	0	1	1	1	1	0	1	1	0	1	0	0	0	1	1	1	0	0	0	0	0
0	0	1	0	0	0	1	1	0	0	1	1	0	0	1	1	1	1	0	1	1	0	1	0	1	0	1	0	0	1	0	0	0	0
0	0	1	0	0	0	1	1	1	1	1	0	1	0	1	0	0	1	0	1	1	0	1	0	1	0	1	1	1	1	0	0	0	0
0	0	1	0	0	1	0	0	0	0	1	0	1	0	0	1	1	1	0	1	1	0	1	1	0	0	1	0	0	1	0	1	0	0
0	0	1	0	0	1	0	0	0	0	1	1	1	0	0	1	1	1	0	1	1	0	1	1	0	0	1	1	1	1	0	1	0	0
0	0	1	0	0	1	0	0	1	1	1	0	0	0	0	0	0	1	0	1	1	0	1	1	1	0	1	0	0	0	0	1	0	0
0	0	1	0	0	1	0	1	0	0	1	0	0	0	0	1	1	1	0	1	1	0	1	1	1	0	1	1	1	0	0	1	0	0
0	0	1	0	0	1	0	1	0	0	1	1	0	0	0	1	1	1	0	1	1	1	0	0	0	0	1	0	0	1	1	1	1	0
0	0	1	0	0	1	0	1	1	1	1	0	1	0	0	0	0	1	0	1	1	1	0	0	0	0	1	1	1	0	0	0	1	0
0	0	1	0	0	1	1	0	0	0	1	0	0	0	1	1	1	1	0	1	1	1	0	0	1	0	1	0	0	0	1	1	0	0
0	0	1	0	0	1	1	0	0	0	1	1	0	0	1	1	1	1	0	1	1	1	0	0	1	0	1	1	1	1	0	0	0	0
0	0	1	0	0	1	1	0	1	1	1	0	1	0	1	0	0	1	0	1	1	1	0	1	0	0	1	0	0	0	1	1	0	0
0	0	1	0	0	1	1	1	0	0	1	0	1	0	1	1	1	1	0	1	1	1	0	1	0	0	1	1	1	1	0	1	0	0
0	0	1	0	0	1	1	1	0	0	1	1	1	0	1	1	1	1	0	1	1	1	0	1	1	0	1	0	0	1	1	1	0	0
0	0	1	0	0	1	1	1	1	1	1	0	0	0	1	0	0	1	0	1	1	1	0	1	1	0	1	1	1	0	0	1	0	0
0	0	1	0	1	0	0	0	0	0	1	0	0	0	0	1	0	1	0	1	1	1	1	0	0	0	1	0	0	0	1	1	1	0
0	0	1	0	1	0	0	0	0	0	1	1	1	0	0	1	1	1	0	1	1	1	1	0	0	0	1	1	1	1	0	0	1	0
0	0	1	0	1	0	0	0	1	1	1	0	0	0	0	1	0	1	0	1	1	1	1	0	1	0	1	0	0	1	1	1	0	0
0	0	1	0	1	0	0	1	0	0	1	0	1	0	0	0	0	1	0	1	1	1	1	0	1	0	1	1	1	0	0	0	0	0
0	0	1	0	1	0	0	1	0	0	1	1	0	0	0	1	1	1	0	1	1	1	1	1	0	0	1	0	0	1	1	1	0	0
0	0	1	0	1	0	0	1	1	1	1	0	1	0	0	0	0	1	0	1	1	1	1	1	0	0	1	1	1	0	0	1	0	0
0	0	1	0	1	0	1	0	0	0	1	0	1	0	1	0	0	1	0	1	1	1	1	1	1	0	1	0	0	0	1	1	0	0
0	0	1	0	1	0	1	0	0	0	1	1	0	0	1	1	1	1	0	1	1	1	1	1	1	0	1	1	1	1	0	1	0	0
0	0	1	0	1	0	1	0	1	1	1	0	1	0	1	0	0	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
0	0	1	0	1	0	1	1	0	0	1	0	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	1	1	1	0	0	1	1
0	0	1	0	1	0	1	1	0	0	1	1	1	0	1	1	1	1	1	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0
0	0	1	0	1	0	1	1	1	1	1	0	0	0	1	0	0	1	1	0	0	0	0	0	1	0	0	1	0	1	0	0	0	0
0	0	1	0	1	1	0	0	0	0	1	0	1	0	0	1	0	1	1	0	0	0	0	0	1	0	0	1	1	0	0	0	1	1
0	0	1	0	1	1	0	0	0	0	1	1	0	0	0	1	1	1	1	0	0	0	0	0	1	1	1	1	0	1	0	0	0	0
0	0	1	0	1	1	0	0	1	1	1	0	1	0	0	1	0	1	1	0	0	0	0	1	0	0	0	1	0	1	0	1	0	0
0	0	1	0	1	1	0	1	0	0	1	0	0	0	0	0	0	1	1	0	0	0	0	1	0	0	0	1	1	0	0	1	1	1
0	0	1	0	1	1	0	1	0	0	1	1	1	0	0	1	1	1	1	0	0	0	0	1	0	1	1	1	0	1	0	1	0	0
0	0	1	0	1	1	0	1	1	1	1	0	0	0	0	0	0	1	1	0	0	0	0	1	1	0	0	1	0	0	0	1	0	0
0	0	1	0	1	1	1	0	0	0	1	0	0	0	1	0	0	1	1	0	0	0	0	1	1	0	0	1	1	1	0	1	1	1
0	0	1	0	1	1	1	0	0	0	1	1	1	0	1	1	1	1	1	0	0	0	0	1	1	1	1	1	0	0	0	1	0	0
0	0	1	0	1	1	1	0	1	1	1	0	0	0	1	0	0	1	1	0	0	0	1	0	0	0	0	1	0	1	0	0	1	0
0	0	1	0	1	1	1	1	0	0	1	0	1	0	1	0	0	1	1	0	0	0	1	0	0	0	0	1	1	0	0	0	1	1
0	0	1	0	1	1	1	1	0	0	1	1	0	0	1	1	1	1	1	0	0	0	1	0	0	1	1	1	0	1	0	0	1	0
0	0	1	0	1	1	1	1	1	1	1	0	1	0	1	0	0	1	1	0	0	0	1	0	1	0	0	1	0	0	0	0	1	0
0	0	1	1	0	0	0	0	0	1	0	0	1	0	0	0	0	1	1	0	0	0	1	0	1	0	0	1	1	1	0	0	1	1
0	0	1	1	0	0	0	0	0	1	1	1	0	0	0	1	1	1	1	0	0	0	1	0	1	1	1	1	0	0	0	0	1	0
0	0	1	1	0	0	0	0	1	1	0	0	1	0	0	0	0	1	1	0	0	0	1	1	0	0	0	1	0	0	0	1	0	0
0	0	1	1	0	0	0	1	0	1	0	0	0	0	0	0	0	1	1	0	0	0	1	1	0	0	0	1	1	1	0	1	1	1
0	0	1	1	0	0	0	1	0	1	1	1	1	0	0	1	1	1	1	0	0	0	1	1	0	1	1	1	0	0	0	1	0	0
0	0	1	1	0	0	0	1	1	1	0	0	0	0	0	0	0	1	1	0	0	0	1	1	1	0	0	1	0	1	0	1	0	0
0	0	1	1	0	0	1	0	0	1	0	0	0	0	1	0	0	1	1	0	0	0	1	1	1	0	0	1	1	0	0	1	1	1
0	0	1	1	0	0	1	0	0	1	1	1	1	0	1	1	1	1	1	0	0	0	1	1	1	1	1	1	0	1	0	1	0	0
0	0	1	1	0	0	1	0	1	1	0	0	0	0	1	0	0	1	1	0	0	1	0	0	0	0	0	0	0	1	0	0	1	0
0	0	1	1	0	0	1	1	0	1	0	0	1	0	1	0	0	1	1	0	0	1	0	0	0	0	0	1	1	1	0	0	1	0
0	0	1	1	0	0	1	1	0	1	1	1	0	0	1	1	1	1	1	0	0	1	0	0	0	1	1	0	0	0	1	1	1	0
0	0	1	1	0	0	1	1	1	1	0	0	1	0	1	0	0	1	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0
0	0	1	1	0	1	0	0	0	1	0	0	0	0	0	0	0	1	1	0	0	1	0	0	1	0	0	1	1	0	0	0	0	0
0	0	1	1	0	1	0	0	0	1	1	1	1	0	0	1	1	1	1	0	0	1	0	0	1	1	1	0	0	1	1	1	0	0
0	0	1	1	0	1	0	0	1	1	0	0	0	0	0	0	0	1	1	0	0	1	0	1	0	0	0	0	0	0	0	1	0	0
0	0	1	1	0	1	0	1	0	1	0	0	1	0	0	0	0	1	1	0	0	1	0	1	0	0	0	1	1	0	0	1	0	0
0	0	1	1	0	1	0	1	0	1	1	1	0	0	0	1	1	1	1	0	0	1	0	1	0	1	1	0	0	1	1	1	0	0
0	0	1	1	0	1	0	1	1	1	0	0	1	0	0	0	0	1	1	0	0	1	0	1	1	0	0	0	0	1	0	1	0	0
0	0	1	1	0	1	1	0	0	1	0	0	1	0	1	0	0	1	1	0	0	1	0	1	1	0	0	1	1	1	0	1	0	0
0	0	1	1	0	1	1	0	0	1	1	1	0	0	1	1	1	1	1	0	0	1	0	1	1	1	1	0	0	0	1	1	0	0
0	0	1	1	0	1	1	0	1	1	0	0	1	0	1	0	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	1	0
0	0	1	1	0	1	1	1	0	1	0	0	0	0	1	0	0	1	1	0	0	1	1	0	0	0	0	1	1	0	0	0	1	0
0	0	1	1	0	1	1	1	0	1	1	1	1	0	1	1	1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	1	1	0
0	0	1	1	0	1	1	1	1	1	0	0	0	0	1	0	0	1	1	0	0	1	1	0	1	0	0	0	0	1	0	0	1	0
0	0	1	1	1	0	0	0	0	1	0	0	0	0	0	1	0	1	1	0	0	1	1	0	1	0	0	1	1	1	0	0	1	0
0	0	1	1	1	0	0	0	0	1	1	1	0	0	0	1	0	1	1	0	0	1	1	0	1	1	1	0	0	0	1	1	1	0
0	0	1	1	1	0	0	0	1	1	0	0	1	1	1	1	0	1	1	0	0	1	1	1	0	0	0	0	0	1	0	1	0	0
0	0	1	1	1	0	0	1	0	1	0	0	1	0	0	0	0	1	1	0	0	1	1	1	0	0	0	1	1	1	0	1	0	0
0	0	1	1	1	0	0	1	0	1	1	1	1	0	0	0	0	1	1	0	0	1	1	1	0	1	1	0	0	0	1	1	0	0
0	0	1	1	1	0	0	1	1	1	0	0	0	1	1	0	0	1	1	0	0	1	1	1	1	0	0	0	0	0	0	1	0	0
0	0	1	1	1	0	1	0	0	1	0	0	1	0	1	0	0	1	1	0	0	1	1	1	1	0	0	1	1	0	0	1	0	0
0	0	1	1	1	0	1	0	0	1	1	1	1	0	1	0	0	1	1	0	0	1	1	1	1	1	1	0	0	1	1	1	0	0
0	0	1	1	1	0	1	0	1	1	0	0	0	1	1	0	0	1	1	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	1	1	1	0	1	1	0	1	0	0	0	0	1	0	0	1	1	0	1	0	0	0	0	0	1	1	1	0	0	0	0	0
0	0	1	1	1	0	1	1	0	1	1	1	0	0	1	0	0	1	1	0	1	0	0	0	0	1	1	0	0	1	1	1	0	0

Inputs												Outputs					Inputs												Outputs				
a_{i-1}	m_{i-1}	a_i	b_i	m_i	a_{i+1}	b_{i+1}	m_{i+1}	c^1	c^2	c^3	c^4	s_i	c^1	c^2	c^3	c^4	a_{i-1}	m_{i-1}	a_i	b_i	m_i	a_{i+1}	b_{i+1}	m_{i+1}	c^1	c^2	c^3	c^4	s_i	c^1	c^2	c^3	c^4
0	0	1	1	1	0	1	1	1	1	0	0	1	1	1	0	0	1	1	0	1	0	0	0	1	0	1	0	0	1	0	0	0	0
0	0	1	1	1	1	0	0	0	1	0	0	1	0	0	1	0	1	1	0	1	0	0	0	1	0	1	1	1	1	0	0	0	0
0	0	1	1	1	1	0	0	0	1	1	1	1	0	0	1	0	1	1	0	1	0	0	0	1	1	1	0	0	0	1	1	0	0
0	0	1	1	1	1	0	0	1	1	0	0	0	1	1	1	0	1	1	0	1	0	0	1	0	0	1	0	0	1	0	1	0	0
0	0	1	1	1	1	0	1	0	1	0	0	0	0	0	0	0	1	1	0	1	0	0	1	0	0	1	1	1	1	0	1	0	0
0	0	1	1	1	1	0	1	0	1	1	1	0	0	0	0	0	1	1	0	1	0	0	1	0	1	1	0	0	0	1	1	0	0
0	0	1	1	1	1	0	1	1	1	0	0	1	1	1	0	0	1	1	0	1	0	0	1	1	0	1	0	0	0	0	1	0	0
0	0	1	1	1	1	1	0	0	1	0	0	0	0	1	0	0	1	1	0	1	0	0	1	1	0	1	1	1	0	0	1	0	0
0	0	1	1	1	1	1	0	0	1	1	1	0	0	1	0	0	1	1	0	1	0	0	1	1	1	1	0	0	1	1	1	0	0
0	0	1	1	1	1	1	0	1	1	0	0	1	1	1	0	0	1	1	0	1	0	1	0	0	0	1	0	0	1	0	0	1	0
0	0	1	1	1	1	1	1	0	1	0	0	1	0	1	0	0	1	1	0	1	0	1	0	0	0	1	1	1	1	0	0	1	0
0	0	1	1	1	1	1	1	0	1	1	1	1	0	1	0	0	1	1	0	1	0	1	0	0	1	1	0	0	0	1	1	1	0
0	0	1	1	1	1	1	1	1	1	0	0	0	1	1	0	0	1	1	0	1	0	1	0	1	0	1	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	0	0	1	0	1	0	0	1	1	1	1	0	1	0	1	0	1	0	1	1	1	0	0	0	1	0
0	1	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	1	1	0	1	0	1	0	1	1	1	0	0	1	1	1	1	0
0	1	0	0	0	0	0	1	0	0	1	0	0	0	0	1	1	1	1	0	1	0	1	1	0	0	1	0	0	0	0	1	0	0
0	1	0	0	0	0	0	1	1	1	1	0	1	0	0	0	0	1	1	0	1	0	1	1	0	0	1	1	1	0	0	1	0	0
0	1	0	0	0	0	1	0	0	0	1	0	0	0	1	1	1	1	1	0	1	0	1	1	0	1	1	0	0	1	1	1	0	0
0	1	0	0	0	0	1	0	1	1	1	0	1	0	1	0	0	1	1	0	1	0	1	1	1	0	1	0	0	1	0	1	0	0
0	1	0	0	0	0	1	1	0	0	1	0	1	0	1	1	1	1	1	0	1	0	1	1	1	0	1	1	1	1	0	1	0	0
0	1	0	0	0	0	1	1	1	1	1	0	0	0	1	0	0	1	1	0	1	0	1	1	1	1	1	0	0	0	1	1	0	0
0	1	0	0	0	1	0	0	0	0	1	0	0	0	0	1	1	1	1	0	1	1	0	0	0	0	1	0	0	0	1	1	1	0
0	1	0	0	0	1	0	0	1	1	1	0	1	0	0	1	0	1	1	0	1	1	0	0	0	0	1	1	1	1	0	0	1	0
0	1	0	0	0	1	0	1	0	0	1	0	1	0	0	1	1	1	1	0	1	1	0	0	0	1	1	0	0	0	1	1	1	0
0	1	0	0	0	1	0	1	1	1	1	0	0	0	0	1	0	1	1	0	1	1	0	0	1	0	1	0	0	1	1	1	0	0
0	1	0	0	0	1	1	0	0	0	1	0	1	0	1	1	1	1	1	0	1	1	0	0	1	0	1	1	1	0	0	0	0	0
0	1	0	0	0	1	1	0	1	1	1	0	0	0	1	0	0	1	1	0	1	1	0	0	1	1	1	0	0	1	1	1	0	0
0	1	0	0	0	1	1	1	0	0	1	0	0	0	1	1	1	1	1	0	1	1	0	1	0	0	1	0	0	1	1	1	0	0
0	1	0	0	0	1	1	1	1	1	1	0	1	0	1	0	0	1	1	0	1	1	0	1	0	0	1	1	1	0	0	1	0	0
0	1	0	0	1	0	0	0	0	0	0	0	1	0	0	1	0	1	1	0	1	1	0	1	0	1	1	0	0	1	1	1	0	0
0	1	0	0	1	0	0	0	1	1	0	0	1	0	0	1	0	1	1	0	1	1	0	1	1	0	1	0	0	0	1	1	0	0
0	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	1	1	0	1	1	0	1	1	0	1	1	1	1	0	1	0	0
0	1	0	0	1	0	0	1	1	1	0	0	0	0	0	0	0	1	1	0	1	1	0	1	1	1	1	0	0	0	1	1	0	0
0	1	0	0	1	0	1	0	0	0	0	0	0	0	1	0	0	1	1	0	1	1	1	0	0	0	1	0	0	1	1	1	1	0
0	1	0	0	1	0	1	0	1	1	0	0	0	0	1	0	0	1	1	0	1	1	1	0	0	0	1	1	1	0	0	0	1	0
0	1	0	0	1	0	1	1	0	0	0	0	1	0	1	0	0	1	1	0	1	1	1	0	0	1	1	0	0	1	1	1	1	0
0	1	0	0	1	0	1	1	1	1	0	0	1	0	1	0	0	1	1	0	1	1	1	0	1	0	1	0	0	0	1	1	1	0
0	1	0	0	1	1	0	0	0	0	0	0	0	0	0	1	0	1	1	0	1	1	1	0	1	0	1	1	1	1	0	0	1	0
0	1	0	0	1	1	0	0	1	1	0	0	0	0	0	1	0	1	1	0	1	1	1	0	1	1	1	0	0	0	1	1	1	0
0	1	0	0	1	1	0	1	0	0	0	0	1	0	0	1	0	1	1	0	1	1	1	1	0	0	1	0	0	0	1	1	0	0
0	1	0	0	1	1	0	1	1	1	0	0	1	0	0	1	0	1	1	0	1	1	1	1	0	0	1	1	1	1	0	1	0	0
0	1	0	0	1	1	1	0	0	0	0	0	1	0	1	0	0	1	1	0	1	1	1	1	0	1	1	0	0	0	1	1	0	0
0	1	0	0	1	1	1	0	1	1	0	0	1	0	1	0	0	1	1	0	1	1	1	1	1	0	1	0	0	1	1	1	0	0
0	1	0	0	1	1	1	1	0	0	0	0	0	0	1	0	0	1	1	0	1	1	1	1	1	0	1	1	1	0	0	1	0	0
0	1	0	0	1	1	1	1	1	1	0	0	0	0	1	0	0	1	1	0	1	1	1	1	1	1	1	0	0	1	1	1	0	0
0	1	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	1	0	0	0	0	1	1
0	1	0	1	0	0	0	0	1	1	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	1	1	0	0	0	1	1
0	1	0	1	0	0	0	1	0	1	0	0	1	0	0	0	0	1	1	1	0	0	0	0	0	1	1	1	0	1	0	0	0	0
0	1	0	1	0	0	0	1	1	1	0	0	1	0	0	0	0	1	1	1	0	0	0	0	1	0	0	1	0	1	0	0	1	1
0	1	0	1	0	0	1	0	0	1	0	0	1	0	1	0	0	1	1	1	0	0	0	0	1	0	0	1	1	1	0	0	1	1
0	1	0	1	0	0	1	0	1	1	0	0	1	0	1	0	0	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	0	0
0	1	0	1	0	0	1	1	0	1	0	0	0	0	1	0	0	1	1	1	0	0	0	1	0	0	0	1	0	1	0	1	1	1
0	1	0	1	0	0	1	1	1	1	0	0	0	0	1	0	0	1	1	1	0	0	0	1	0	0	0	1	1	1	0	1	1	1
0	1	0	1	0	1	0	0	0	1	0	0	1	0	0	1	0	1	1	1	0	0	0	1	0	1	1	1	0	0	0	1	0	0
0	1	0	1	0	1	0	0	1	1	0	0	1	0	0	1	0	1	1	1	0	0	0	1	1	0	0	1	0	0	0	1	1	1
0	1	0	1	0	1	0	1	0	1	0	0	0	0	0	1	0	1	1	1	0	0	0	1	1	0	0	1	1	0	0	1	1	1
0	1	0	1	0	1	0	1	1	1	0	0	0	0	0	1	0	1	1	1	0	0	0	1	1	1	1	1	0	1	0	1	0	0
0	1	0	1	0	1	1	0	0	1	0	0	0	0	1	0	0	1	1	1	0	0	1	0	0	0	0	1	0	1	0	0	1	1
0	1	0	1	0	1	1	0	1	1	0	0	0	0	1	0	0	1	1	1	0	0	1	0	0	0	0	1	1	1	0	0	1	1
0	1	0	1	0	1	1	1	0	1	0	0	1	0	1	0	0	1	1	1	0	0	1	0	0	1	1	1	0	0	0	0	0	0
0	1	0	1	0	1	1	1	1	1	0	0	1	0	1	0	0	1	1	1	0	0	1	0	1	0	0	1	0	0	0	0	1	1
0	1	0	1	1	0	0	0	0	1	0	0	1	0	0	1	0	1	1	1	0	0	1	0	1	0	0	1	1	0	0	0	1	1
0	1	0	1	1	0	0	0	1	1	0	0	0	1	1	1	0	1	1	1	0	0	1	0	1	1	1	1	0	1	0	0	0	0
0	1	0	1	1	0	0	1	0	1	0	0	0	0	0	0	0	1	1	1	0	0	1	1	0	0	0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	1	1	1	0	0	1	1	1	0	0	1	1	1	0	0	1	1	0	0	0	1	1	0	0	1	1	1
0	1	0	1	1	0	1	0	0	1	0	0	0	0	1	0	0	1	1	1	0	0	1	1	0	1	1	1	0	1	0	1	0	0
0	1	0	1	1	0	1	0	1	1	0	0	1	1	1	0	0	1	1	1	0	0	1	1	1	0	0	1	0	1	0	1	1	1
0	1	0	1	1	0	1	1	0	1	0	0	1	0	1	0	0	1	1	1	0	0	1	1	1	0	0	1	1	1	0	1	1	1
0	1	0	1	1	0	1	1	1	1	0	0	0	1	1	0	0	1	1	1	0	0	1	1	1	1	1	1	0	0	0	1	0	0
0	1	0	1	1	1	0	0	0	1	0	0	0	0	0	1	0	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	1	0
0	1	0	1	1	1	0	0	1	1	0	0	1	1	1	1	0	1	1	1	0	1	0	0	0	0	0	1	1	1	0	0	1	1
0	1	0	1	1	1	0	1	0	1	0	0	1	0	0	1	0	1	1	1	0	1	0	0	0	1	1	0	0	0	0	0	1	0
0	1	0	1	1	1	0	1	1	1	0	0	0	1	1	1	0	1	1	1	0	1	0	0	1	0	0	0	0	1	0	0	0	0
0	1	0	1	1	1	1	0	0	1	0	0	1	0	1	0	0	1	1	1	0	1	0	0	1	0	0	1	1	0	0	0	1	1
0	1	0	1	1	1	1	0	1	1	0	0	0	1	1	0	0	1	1	1	0	1	0	0	1	1	1	0	0	1	0	0	0	0
0	1	0	1	1	1	1	1	0	1	0	0	0	0	1	0	0	1	1	1	0	1	0	1	0	0	0	0	0	1	0	1	0	0
0	1	0	1	1	1	1	1	1	1	0	0	1	1	1	0	0	1	1	1	0	1	0	1	0	0	0	1	1	0	0	1	1	1
0	1	1	0	0	0	0	0	0	0	1	0	0	0	0	1	1	1	1	1	0	1	0	1	0	1	1	0	0	1	0	1	0	0
0	1	1	0	0	0	0	0	1	1	1	0	0	0	0	1	1	1	1	1	0	1	0	1	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	1	0	1	0	0	1	1	1	1	1	0	1	0	1	1	0	0	1	1	1	0	1	1	1
0	1	1	0	0	0	0	1	1	1	1	0	1	0	0	1	1	1	1	1	0	1	0	1	1	1	1	0	0	0	0	1	0	0
0	1	1	0	0	0	1	0	0	0	1	0	1	0	1	1	1	1	1	1	0	1	1	0	0	0	0	0	0	1	0	0	1	0
0	1	1	0	0	0	1	0	1	1	1	0	1	0	1	1	1	1	1	1	0	1	1	0	0	0	0	1	1	0	0	0	1	1
0	1	1	0	0	0	1	1	0	0	1	0	0	0	1	1	1	1	1	1	0	1	1	0	0	1	1	0	0	1	0	0	1	0
0	1	1	0	0	0	1	1	1	1	1	0	0	0	1	1	1	1	1	1	0	1	1	0	1	0	0	0	0	0	0	0	0	0
0	1	1	0	0	1	0	0	0	0	1	0	1	0	0	1	1	1	1	1	0	1	1	0	1	0	0	1	1	1	0	0	1	1
0	1	1	0	0	1	0	0	1	1	1	0	1	0	0	1	1	1	1	1	0	1	1	0	1	1	1	0	0	0	0	0	0	0
0	1	1	0	0	1	0	1	0	0	1	0	0	0	0	1	1	1	1	1	0	1	1	1	0	0	0	0	0	0	0	1	0	0
0	1	1	0	0	1	0	1	1	1	1	0	0	0	0	1	1	1	1	1	0	1	1	1	0	0	0	1	1	1	0	1	1	1
0	1	1	0	0	1	1	0	0	0	1	0	0	0	1	1	1	1	1	1	0	1	1	1	0	1	1	0	0	0	0	1	0	0
0	1	1	0	0	1	1	0	1	1	1	0	0	0	1	1	1	1	1	1	0	1	1	1	1	0	0	0	0	1	0	1	0	0
0	1	1	0	0	1	1	1	0	0	1	0	1	0	1	1	1	1	1	1	0	1	1	1	1	0	0	1	1	0	0	1	1	1
0	1	1	0	0	1	1	1	1	1	1	0	1	0	1	1	1	1	1	1	0	1	1	1	1	1	1	0	0	1	0	1	0	0
0	1	1	0	1	0	0	0	0	0	1	0	1	0	0	1	1	1	1	1	1	0	0	0	0	0	1	0	0	1	0	0	0	0
0	1	1	0	1	0	0	0	1	1	1	0	0	0	0	1	0	1	1	1	1	0	0	0	0	0	1	1	1	0	0	0	1	1
0	1	1	0	1	0	0	1	0	0	1	0	0	0	0	1	1	1	1	1	1	0	0	0	0	1	1	0	0	1	0	0	0	0
0	1	1	0	1	0	0	1	1	1	1	0	1	0	0	0	0	1	1	1	1	0	0	0	1	0	1	0	0	0	0	0	0	0
0	1	1	0	1	0	1	0	0	0	1	0	0	0	1	1	1	1	1	1	1	0	0	0	1	0	1	1	1	1	0	0	1	1
0	1	1	0	1	0	1	0	1	1	1	0	1	0	1	0	0	1	1	1	1	0	0	0	1	1	1	0	0	0	0	0	0	0
0	1	1	0	1	0	1	1	0	0	1	0	1	0	1	1	1	1	1	1	1	0	0	1	0	0	1	0	0	0	0	1	0	0
0	1	1	0	1	0	1	1	1	1	1	0	0	0	1	0	0	1	1	1	1	0	0	1	0	0	1	1	1	1	0	1	1	1
0	1	1	0	1	1	0	0	0	0	1	0	0	0	0	1	1	1	1	1	1	0	0	1	0	1	1	0	0	0	0	1	0	0
0	1	1	0	1	1	0	0	1	1	1	0	1	0	0	1	0	1	1	1	1	0	0	1	1	0	1	0	0	1	0	1	0	0
0	1	1	0	1	1	0	1	0	0	1	0	1	0	0	1	1	1	1	1	1	0	0	1	1	0	1	1	1	0	0	1	1	1

Inputs												Outputs					Inputs												Outputs				
a_{i-1}	m_{i-1}	a_i	b_i	m_i	a_{i+1}	b_{i+1}	m_{i+1}	c^1	c^2	c^3	c^4	s_i	c^1	c^2	c^3	c^4	a_{i-1}	m_{i-1}	a_i	b_i	m_i	a_{i+1}	b_{i+1}	m_{i+1}	c^1	c^2	c^3	c^4	s_i	c^1	c^2	c^3	c^4
0	1	1	0	1	1	0	1	1	1	1	0	0	0	0	0	0	1	1	1	1	0	0	1	1	1	1	0	0	1	0	1	0	0
0	1	1	0	1	1	1	0	0	0	1	0	1	0	1	1	1	1	1	1	1	0	1	0	0	0	1	0	0	0	0	0	0	0
0	1	1	0	1	1	1	0	1	1	1	0	0	0	1	0	0	1	1	1	1	0	1	0	0	0	1	1	1	1	0	0	1	1
0	1	1	0	1	1	1	1	0	0	1	0	0	0	1	1	1	1	1	1	1	0	1	0	0	1	1	0	0	0	0	0	0	0
0	1	1	0	1	1	1	1	1	1	1	0	1	0	1	0	0	1	1	1	1	0	1	0	1	0	1	0	0	1	0	0	0	0
0	1	1	1	0	0	0	0	0	1	0	0	0	0	0	1	1	1	1	1	1	0	1	0	1	0	1	1	1	0	0	0	1	1
0	1	1	1	0	0	0	0	1	1	0	0	1	0	0	0	0	1	1	1	1	0	1	0	1	1	1	0	0	1	0	0	0	0
0	1	1	1	0	0	0	1	0	1	0	0	1	0	0	1	1	1	1	1	1	0	1	1	0	0	1	0	0	1	0	1	0	0
0	1	1	1	0	0	0	1	1	1	0	0	0	0	0	0	0	1	1	1	1	0	1	1	0	0	1	1	1	0	0	1	1	1
0	1	1	1	0	0	1	0	0	1	0	0	1	0	1	1	1	1	1	1	1	0	1	1	0	1	1	0	0	1	0	1	0	0
0	1	1	1	0	0	1	0	1	1	0	0	0	0	1	0	0	1	1	1	1	0	1	1	1	0	1	0	0	0	0	1	0	0
0	1	1	1	0	0	1	1	0	1	0	0	0	0	1	1	1	1	1	1	1	0	1	1	1	0	1	1	1	1	0	1	1	1
0	1	1	1	0	0	1	1	1	1	0	0	1	0	1	0	0	1	1	1	1	0	1	1	1	1	1	0	0	0	0	1	0	0
0	1	1	1	0	1	0	0	0	1	0	0	1	0	0	1	1	1	1	1	1	1	0	0	0	0	1	0	0	0	0	0	1	0
0	1	1	1	0	1	0	0	1	1	0	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	1	1	1	0	0	0	1	0
0	1	1	1	0	1	0	1	0	1	0	0	0	0	0	1	1	1	1	1	1	1	0	0	0	1	1	0	0	1	1	1	1	0
0	1	1	1	0	1	0	1	1	1	0	0	1	0	0	0	0	1	1	1	1	1	0	0	1	0	1	0	0	1	0	0	0	0
0	1	1	1	0	1	1	0	0	1	0	0	0	0	1	1	1	1	1	1	1	1	0	0	1	0	1	1	1	1	0	0	0	0
0	1	1	1	0	1	1	0	1	1	0	0	1	0	1	0	0	1	1	1	1	1	0	0	1	1	1	0	0	0	1	1	0	0
0	1	1	1	0	1	1	1	0	1	0	0	1	0	1	1	1	1	1	1	1	1	0	1	0	0	1	0	0	1	0	1	0	0
0	1	1	1	0	1	1	1	1	1	0	0	0	0	1	0	0	1	1	1	1	1	0	1	0	0	1	1	1	1	0	1	0	0
0	1	1	1	1	0	0	0	0	1	0	0	0	0	0	1	0	1	1	1	1	1	0	1	0	1	1	0	0	0	1	1	0	0
0	1	1	1	1	0	0	0	1	1	0	0	0	0	0	1	0	1	1	1	1	1	0	1	1	0	1	0	0	0	0	1	0	0
0	1	1	1	1	0	0	1	0	1	0	0	1	0	0	0	0	1	1	1	1	1	0	1	1	0	1	1	1	0	0	1	0	0
0	1	1	1	1	0	0	1	1	1	0	0	1	0	0	0	0	1	1	1	1	1	0	1	1	1	1	0	0	1	1	1	0	0
0	1	1	1	1	0	1	0	0	1	0	0	1	0	1	0	0	1	1	1	1	1	1	0	0	0	1	0	0	1	0	0	1	0
0	1	1	1	1	0	1	0	1	1	0	0	1	0	1	0	0	1	1	1	1	1	1	0	0	0	1	1	1	1	0	0	1	0
0	1	1	1	1	0	1	1	0	1	0	0	0	0	1	0	0	1	1	1	1	1	1	0	0	1	1	0	0	0	1	1	1	0
0	1	1	1	1	0	1	1	1	1	0	0	0	0	1	0	0	1	1	1	1	1	1	0	1	0	1	0	0	0	0	0	0	0
0	1	1	1	1	1	0	0	0	1	0	0	1	0	0	1	0	1	1	1	1	1	1	0	1	0	1	1	1	0	0	0	0	0
0	1	1	1	1	1	0	0	1	1	0	0	1	0	0	1	0	1	1	1	1	1	1	0	1	1	1	0	0	1	1	1	0	0
0	1	1	1	1	1	0	1	0	1	0	0	0	0	0	0	0	1	1	1	1	1	1	1	0	0	1	0	0	0	0	1	0	0
0	1	1	1	1	1	0	1	1	1	0	0	0	0	0	0	0	1	1	1	1	1	1	1	0	0	1	1	1	0	0	1	0	0
0	1	1	1	1	1	1	0	0	1	0	0	0	0	1	0	0	1	1	1	1	1	1	1	0	1	1	0	0	1	1	1	0	0
0	1	1	1	1	1	1	0	1	1	0	0	0	0	1	0	0	1	1	1	1	1	1	1	1	0	1	0	0	1	0	1	0	0
0	1	1	1	1	1	1	1	0	1	0	0	1	0	1	0	0	1	1	1	1	1	1	1	1	0	1	1	1	1	0	1	0	0
0	1	1	1	1	1	1	1	1	1	0	0	1	0	1	0	0	1	1	1	1	1	1	1	1	1	1	0	0	0	1	1	0	0

The previous tables have the exact size because the two solutions require the same types of the logic gates, and same arity. This is normal, since, in both cases, we have to recognise exactly the same cases with different algebraic contexts. Depending on the need of the operation, Booth-Karatsuba's actions are natural or inverse projection developed in **table 1**, and along with this, an element can have an additive or subtractive role. It should be remembered that we have now 3 operands simultaneously.

Once established all possible cases, we find that there are 2^9+2^7 combinations to implement the output and carry-out; the latter being denoted as c^1, c^2, c^3, and c^4 in **table 6** and **table 7**. As you would expect, we have carry input bits and carry output bits.

Due to the nature of the Booth-Karatsuba's method, there is a need to evaluate the adjacent bits. This influences to the arity of logic gates, with a significant increase. We obtain a maximum number of 9 inputs, which would be problematic: usually the number of inputs is restricted to a maximum of 5. In the discuss section we will consider the impact that this implies. The main objective, now, is to demonstrate that it is possible to calculate modular additions in complexity order of $\mathcal{O}(n)$, rather than referring to physical implementation details.

Aware, despite the above, we look for a combinational circuit, especially at the logic gate level and putting aside the technology to manufacture of diodes. In spite of this, the technology used will be decisive in the performance and for the results. But this section concerns the theoretical analysis.

On the basis of the previous tables we have everything we need to describe our modular full adder cell. As any Boolean function can be put into the CDNF, with minterms, and its dual CCNF, with maxterms, there are many ways to build a circuit that implement each possible configuration of the input variables. It is interesting to get the optimal solution applying simplification methods.

In our case, we proceed to show the best solution reached, which adjusts to the rows of the tables. Remember that we have defined two tables: one in **figure 4** and one in **figure 5** and which, as such, we are going to describe two full adder cells. We describe two cells in which each includes a context: additive or subtractive.

On the other hand, in a similar way to the classical algorithms, full cell is the generic case. But, for example, the first bit is calculated in a simpler path. The last bit is also particular case, equally, this bit requires a different calculation and consider overflow problems. But in this work, the *overflow* is interpreted like a representational difficulty, since we operate in modular arithmetic. In other words, the representation of the result will always be correct, without overflows. We understand the overflows like flags that indicates the algebraic context, due to dual symmetric.

These algebraic dual is means that we experience a change in the wiring between logic cells in **table 6** and logic cell in **table 7**, affecting the inputs denoted in the circuit, with respect to m, the field module, and the operand b, which we have chosen as the base.

There is also a change between the connections of operand a to its $i-th$ normal value and the negated value; we remember that in **table 7**, a is used in additive context, while in the other adder of **table 6**, a has a subtractive context.

It was concluded that the 2 circuits will require boolean components of the same maximum order, and levels with the same number of logic gates working in parallel.

We proceed to show the **figure 6** in which we calculate a subtraction to the modulus defined for the finite field. Similarly, we show **figure 7** in which we calculate an additive to the modulus defined for the finite field:

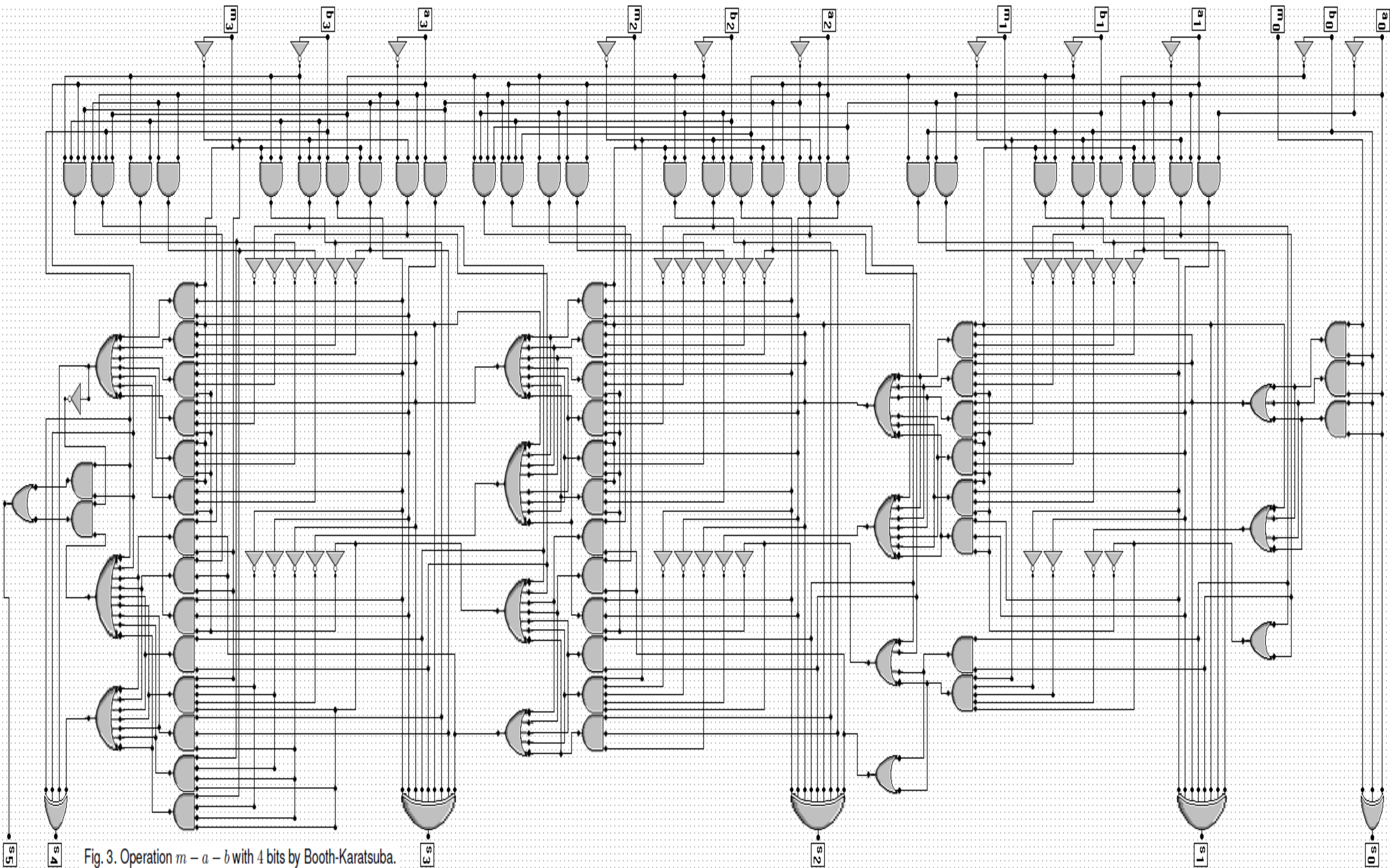

Fig. 3. Operation $m - a - b$ with 4 bits by Booth-Karatsuba.

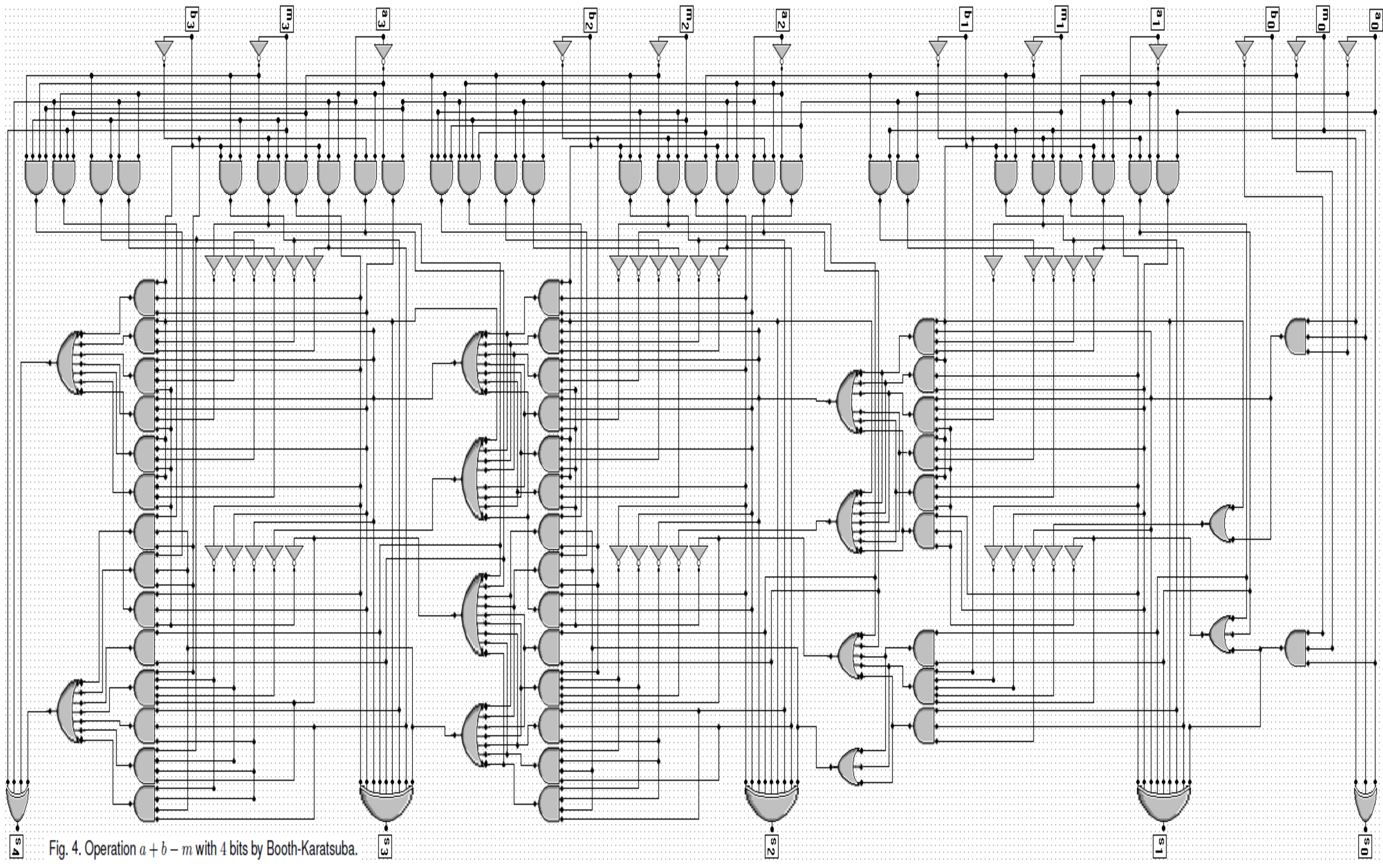

Fig. 4. Operation $a + b - m$ with 4 bits by Booth-Karatsuba.

In order to increase the understanding, and to evidence the order complexity, we show the source code for the algorithm of the operation implemented in **figure 6** for $n - bit$ operands:

```
1   ALGORITHM 1:
2
3   s[0] = XOR3(a[0], b[0], m[0]);
4
5   bit l0b1 = AND(a[0], b[0]);
6   bit l0b2 = AND(a[0], m[0]);
7   bit l0b3 = AND(b[0], m[0]);
8
9   bit l1c1b1 = AND(NOT(a[0]), a[1]);
10  bit l1c1b2 = AND3(a[0], NOT(a[1]), NOT(m[1]));
11  bit l1c1b3 = AND3(a[0], NOT(a[1]), m[1]);
12  bit l1c1b4 = AND(NOT(b[0]), b[1]);
13  bit l1c1b5 = AND3(b[0], NOT(b[1]), NOT(m[1]));
14  bit l1c1b6 = AND3(b[0], NOT(b[1]), m[1]);
15  bit l1c1b7 = AND(a[0], NOT(a[1]));
16  bit l1c1b8 = AND(b[0], NOT(b[1]));
17
18  bit l0c2b1 = OR3(l0b1, l0b2, l0b3);
19  bit l0c2b2 = OR4(l0b1, l0b2, l0b3, m[1]);
20  bit l0c2b3 = OR(l1c1b2, l1c1b5);
21
22  s[1] = XOR8(l1c1b3, l1c1b6, m[1], l1c1b2, l1c1b5, l0c2b1, l1c1b1, l1c1b4);
23
24  bit l1c2b1 = AND4(m[1], l0c2b1, NOT(l1c1b6), NOT(l1c1b3));
25  bit l1c2b2 = AND4(l0c2b1, l1c1b4, NOT(l1c1b2), NOT(l0c2b3));
26  bit l1c2b3 = AND4(l0c2b1, l1c1b1, NOT(l1c1b5), NOT(l0c2b3));
27  bit l1c2b4 = AND4(m[1], l1c1b1, NOT(l1c1b8), NOT(l0c2b3));
28  bit l1c2b5 = AND4(m[1], l1c1b4, NOT(l1c1b7), NOT(l0c2b3));
29  bit l1c2b6 = AND3(l1c1b4, l1c1b1, NOT(l0c2b3));
30  bit l1c2b7 = AND(l1c1b5, l1c1b2);
31  bit l1c2b8 = AND5(NOT(m[1]), NOT(l1c1b4), NOT(l1c1b1), NOT(l0c2b2), l0c2b3);
32
33  bit l2c1b1 = AND(NOT(a[1]), a[2]);
34  bit l2c1b2 = AND3(a[1], NOT(a[2]), NOT(m[2]));
35  bit l2c1b3 = AND3(a[1], NOT(a[2]), m[2]);
36  bit l2c1b4 = AND(NOT(b[1]), b[2]);
37  bit l2c1b5 = AND3(b[1], NOT(b[2]), NOT(m[2]));
38  bit l2c1b6 = AND3(b[1], NOT(b[2]), m[2]);
39  bit l2c1b7 = AND(a[1], NOT(a[2]));
40  bit l2c1b8 = AND(b[1], NOT(b[2]));
41  bit l2c1b9 = AND4(NOT(b[1]), b[2], a[1], NOT(a[2]));
42  bit l2c1bA = AND4(NOT(a[1]), a[2], b[1], NOT(b[2]));
43
44  bit l1c3b1 = OR6(l1c2b1, l1c2b2, l1c2b3, l1c2b4, l1c2b5, l1c2b6);
45  bit l1c3b2 = OR7(m[2], l1c2b1, l1c2b2, l1c2b3, l1c2b4, l1c2b5, l1c2b6);
46  bit l1c3b3 = OR4(l2c1b2, l2c1b5, l1c2b7, l1c2b8);
47  bit l1c3b4 = OR(l1c2b7, l1c2b8);
48
49  s[2] = XOR9(l1c3b4, l2c1b3, l2c1b6, m[2], l2c1b2, l2c1b5, l1c3b1,
50              l2c1b1, l2c1b4);
51
52  bit l2c2b0 = AND3(m[2], l2c1b4, l2c1b1);
53  bit l2c2b1 = AND4(m[2], l1c3b1, NOT(l2c1b6), NOT(l2c1b3));
54  bit l2c2b2 = AND4(l1c3b1, l2c1b4, NOT(l2c1b2), NOT(l1c3b3));
55  bit l2c2b3 = AND4(l1c3b1, l2c1b1, NOT(l2c1b5), NOT(l1c3b3));
56  bit l2c2b4 = AND4(m[2], l2c1b1, NOT(l2c1b8), NOT(l1c3b3));
57  bit l2c2b5 = AND4(m[2], l2c1b4, NOT(l2c1b7), NOT(l1c3b3));
58  bit l2c2b6 = AND(l2c1b9, l1c3b4);
59  bit l2c2b7 = AND(l2c1bA, l1c3b4);
60  bit l2c2b8 = AND3(l2c1b4, l2c1b1, NOT(l1c3b3));
61  bit l2c2b9 = AND(l2c1b5, l2c1b2);
62  bit l2c2bA = AND5(NOT(m[2]), NOT(l2c1b4), NOT(l2c1b1), NOT(l1c3b2), l1c3b3);
63  bit l2c2bB = AND3(l2c1b6, l2c1b3, NOT(l1c3b1));
64
65  bit l3c1b1 = AND(NOT(a[2]), a[3]);
66  bit l3c1b2 = AND3(a[2], NOT(a[3]), NOT(m[3]));
67  bit l3c1b3 = AND3(a[2], NOT(a[3]), m[3]);
68  bit l3c1b4 = AND(NOT(b[2]), b[3]);
69  bit l3c1b5 = AND3(b[2], NOT(b[3]), NOT(m[3]));
70  bit l3c1b6 = AND3(b[2], NOT(b[3]), m[3]);
71  bit l3c1b7 = AND(a[2], NOT(a[3]));
72  bit l3c1b8 = AND(b[2], NOT(b[3]));
73  bit l3c1b9 = AND4(NOT(b[2]), b[3], a[2], NOT(a[3]));
74  bit l3c1bA = AND4(NOT(a[2]), a[3], b[2], NOT(b[3]));
75
76  bit l2c3b1 = OR7(l2c2b0, l2c2b1, l2c2b2, l2c2b3, l2c2b4, l2c2b5, l2c2b8);
77  bit l2c3b2 = OR8(m[3], l2c2b0, l2c2b1, l2c2b2, l2c2b3, l2c2b4, l2c2b5, l2c2b8);
78  bit l2c3b3 = OR7(l3c1b2, l3c1b5, l2c2b6, l2c2b7, l2c2b9, l2c2bA, l2c2bB);
79  bit l2c3b4 = OR5(l2c2b6, l2c2b7, l2c2b9, l2c2bA, l2c2bB);
80
81  s[3] = XOR9(l2c3b4, l3c1b3, l3c1b6, m[3], l3c1b2, l3c1b5, l2c3b1,
82              l3c1b1, l3c1b4);
83
84  bit lxc2b0 = AND3(m[3], l3c1b4, l3c1b1);
85  bit lxc2b1 = AND4(m[3], l2c3b1, NOT(l3c1b6), NOT(l3c1b3));
86  bit lxc2b2 = AND4(l2c3b1, l3c1b4, NOT(l3c1b2), NOT(l2c3b3));
87  bit lxc2b3 = AND4(l2c3b1, l3c1b1, NOT(l3c1b5), NOT(l2c3b3));
88  bit lxc2b4 = AND4(m[3], l3c1b1, NOT(l3c1b8), NOT(l2c3b3));
89  bit lxc2b5 = AND4(m[3], l3c1b4, NOT(l3c1b7), NOT(l2c3b3));
90  bit lxc2b6 = AND3(l3c1b9, l2c3b4, NOT(m[3]));
91  bit lxc2b7 = AND3(l3c1bA, l2c3b4, NOT(m[3]));
92  bit lxc2b8 = AND3(l3c1b4, l3c1b1, NOT(l2c3b3));
93  bit lxc2b9 = AND(l3c1b5, l3c1b2);
94  bit lxc2bA = AND5(NOT(m[3]), NOT(l3c1b4), NOT(l3c1b1), NOT(l2c3b2), l2c3b3);
95  bit lxc2bB = AND3(l3c1b3, l3c1b6, NOT(l2c3b1));
96  bit lxc2bC = AND4(l3c1b8, NOT(l3c1b1), NOT(l2c3b1), l2c3b3);
97  bit lxc2bD = AND4(l3c1b7, NOT(l3c1b4), NOT(l2c3b1), l2c3b3);
98
99  bit lxc1b1 = AND(NOT(a[3]), a[4]);
100 bit lxc1b2 = AND3(a[3], NOT(a[4]), NOT(m[4]));
101 bit lxc1b3 = AND3(a[3], NOT(a[4]), m[4]);
102 bit lxc1b4 = AND(NOT(b[3]), b[4]);
103 bit lxc1b5 = AND3(b[3], NOT(b[4]), NOT(m[4]));
104 bit lxc1b6 = AND3(b[3], NOT(b[4]), m[4]);
105 bit lxc1b7 = AND(a[3], NOT(a[4]));
106 bit lxc1b8 = AND(b[3], NOT(b[4]));
107 bit lxc1b9 = AND4(NOT(b[3]), b[4], a[3], NOT(a[4]));
108 bit lxc1bA = AND4(NOT(a[3]), a[4], b[3], NOT(b[4]));
109
110 bit lxc3b1 = OR7(lxc2b0, lxc2b1, lxc2b2, lxc2b3, lxc2b4, lxc2b5, lxc2b8);
111 bit lxc3b2 = OR8(m[4], lxc2b0, lxc2b1, lxc2b2, lxc2b3, lxc2b4, lxc2b5, lxc2b8);
112 bit lxc3b3 = OR9(lxc1b2, lxc1b5, lxc2b6, lxc2b7, lxc2b9, lxc2bA, lxc2bB,
113                  lxc2bC, lxc2bD);
114 bit lxc3b4 = OR7(lxc2b6, lxc2b7, lxc2b9, lxc2bA, lxc2bB, lxc2bC, lxc2bD);
115
116 s[4] = XOR9(lxc3b4, lxc1b3, lxc1b6, m[4], lxc1b2, lxc1b5, lxc3b1,
117             lxc1b1, lxc1b4);
118
119 for(int i = 4; i < n - 1; i++) {
120     lxc2b0 = AND3(m[i], lxc1b4, lxc1b1);
121     lxc2b1 = AND4(m[i], lxc3b1, NOT(lxc1b6), NOT(lxc1b3));
122     lxc2b2 = AND4(lxc3b1, lxc1b4, NOT(lxc1b2), NOT(lxc3b3));
123     lxc2b3 = AND4(lxc3b1, lxc1b1, NOT(lxc1b5), NOT(lxc3b3));
124     lxc2b4 = AND4(m[i], lxc1b1, NOT(lxc1b8), NOT(lxc3b3));
125     lxc2b5 = AND4(m[i], lxc1b4, NOT(lxc1b7), NOT(lxc3b3));
126     lxc2b6 = AND3(lxc1b9, lxc3b4, NOT(m[i]));
127     lxc2b7 = AND3(lxc1bA, lxc3b4, NOT(m[i]));
128     lxc2b8 = AND3(lxc1b4, lxc1b1, NOT(lxc3b3));
129     lxc2b9 = AND(lxc1b5, lxc1b2);
130     lxc2bA = AND5(NOT(m[i]), NOT(lxc1b4), NOT(lxc1b1), NOT(lxc3b2), lxc3b3);
131     lxc2bB = AND3(lxc1b3, lxc1b6, NOT(lxc3b1));
132     lxc2bC = AND4(lxc1b8, NOT(lxc1b1), NOT(lxc3b1), lxc3b3);
133     lxc2bD = AND4(lxc1b7, NOT(lxc1b4), NOT(lxc3b1), lxc3b3);
134
135     lxc1b1 = AND(NOT(a[i]), a[i + 1]);
136     lxc1b2 = AND3(a[i], NOT(a[i + 1]), NOT(m[i + 1]));
137     lxc1b3 = AND3(a[i], NOT(a[i + 1]), m[i + 1]);
138     lxc1b4 = AND(NOT(b[i]), b[i + 1]);
139     lxc1b5 = AND3(b[i], NOT(b[i + 1]), NOT(m[i + 1]));
140     lxc1b6 = AND3(b[i], NOT(b[i + 1]), m[i + 1]);
141     lxc1b7 = AND(a[i], NOT(a[i + 1]));
142     lxc1b8 = AND(b[i], NOT(b[i + 1]));
143     lxc1b9 = AND4(NOT(b[i]), b[i + 1], a[i], NOT(a[i + 1]));
144     lxc1bA = AND4(NOT(a[i]), a[i + 1], b[i], NOT(b[i + 1]));
145
146     lxc3b1 = OR7(lxc2b0, lxc2b1, lxc2b2, lxc2b3, lxc2b4, lxc2b5, lxc2b8);
147     lxc3b2 = OR8(m[i + 1], lxc2b0, lxc2b1, lxc2b2, lxc2b3, lxc2b4, lxc2b5, lxc2b8);
148     lxc3b3 = OR9(lxc1b2, lxc1b5, lxc2b6, lxc2b7, lxc2b9, lxc2bA, lxc2bB, lxc2bC, lxc2bD);
149     lxc3b4 = OR7(lxc2b6, lxc2b7, lxc2b9, lxc2bA, lxc2bB, lxc2bC, lxc2bD);
150
151     s[i + 1] = XOR9(lxc3b4, lxc1b3, lxc1b6, m[i + 1], lxc1b2, lxc1b5, lxc3b1,
152                     lxc1b1, lxc1b4);
153 }
154
155 bit lnc2b0 = AND3(m[n - 1], lxc1b4, lxc1b1);
156 bit lnc2b1 = AND4(m[n - 1], lxc3b1, NOT(lxc1b6), NOT(lxc1b3));
157 bit lnc2b2 = AND4(lxc3b1, lxc1b4, NOT(lxc1b2), NOT(lxc3b3));
158 bit lnc2b3 = AND4(lxc3b1, lxc1b1, NOT(lxc1b5), NOT(lxc3b3));
159 bit lnc2b4 = AND4(m[n - 1], lxc1b1, NOT(lxc1b8), NOT(lxc3b3));
160 bit lnc2b5 = AND4(m[n - 1], lxc1b4, NOT(lxc1b7), NOT(lxc3b3));
161 bit lnc2b6 = AND3(lxc1b9, lxc3b4, NOT(m[n - 1]));
162 bit lnc2b7 = AND3(lxc1bA, lxc3b4, NOT(m[n - 1]));
163 bit lnc2b8 = AND3(lxc1b4, lxc1b1, NOT(lxc3b3));
164 bit lnc2b9 = AND(lxc1b5, lxc1b2);
165 bit lnc2bA = AND5(NOT(m[n - 1]), NOT(lxc1b4), NOT(lxc1b1), NOT(lxc3b2), lxc3b3);
166 bit lnc2bB = AND3(lxc1b3, lxc1b6, NOT(lxc3b1));
167 bit lnc2bC = AND4(lxc1b8, NOT(lxc1b1), NOT(lxc3b1), lxc3b3);
168 bit lnc2bD = AND4(lxc1b7, NOT(lxc1b4), NOT(lxc3b1), lxc3b3);
169
170 bit lnc3b1 = OR7(lnc2b0, lnc2b1, lnc2b2, lnc2b3, lnc2b4, lnc2b5, lnc2b8);
171 bit lnc3b2 = OR9(a[n - 1], b[n - 1], lnc2b6, lnc2b7, lnc2b9, lnc2bA, lnc2bB,
172                  lnc2bC, lnc2bD);
173 bit lnc3b3 = OR7(lnc2b6, lnc2b7, lnc2b9, lnc2bA, lnc2bB, lnc2bC, lnc2bD);
174
175 s[n] = XOR4(lnc3b3, lnc3b1, a[n - 1], b[n - 1]);
176
177 bit lnb1 = AND(a[n - 1], b[n - 1]);
178 bit lnb2 = AND(NOT(lnc3b1), lnc3b2);
179
180 s[n + 1] = OR(lnb1, lnb2);
```

Exactly like dual case, in order to increase the understanding, and to evidence the order complexity in the second solution, we show the source code for the algorithm of the operation implemented in **figure 7**:

```
1   ALGORITHM 2:
2
3   s[0] = XOR3(a[0], b[0], m[0]);
4
5   bit l0b1 = AND3(b[0], m[0], NOT(a[0]));
6   bit l0b2 = AND3(NOT(b[0]), NOT(m[0]), a[0]);
7
8   bit l1c1b1 = AND(a[0], NOT(a[1]));
9   bit l1c1b2 = AND3(NOT(a[0]), a[1], NOT(b[1]));
10  bit l1c1b3 = AND3(NOT(a[0]), a[1], b[1]);
11  bit l1c1b4 = AND(NOT(m[0]), m[1]);
12  bit l1c1b5 = AND3(m[0], NOT(m[1]), NOT(b[1]));
13  bit l1c1b6 = AND3(m[0], NOT(m[1]), b[1]);
14  bit l1c1b7 = AND(NOT(a[0]), a[1]);
15  bit l1c1b8 = AND(m[0], NOT(m[1]));
16
17  bit l0c2b1 = OR(b[1], l0b1);
18  bit l0c2b2 = OR3(l1c1b2, l1c1b5, l0b2);
19
20  s[1] = XOR9(l0b2, l1c1b3, l1c1b6, b[1], l1c1b2, l1c1b5, l0b1, l1c1b1, l1c1b4);
21
22  bit l1c2b1 = AND4(b[1], l0b1, NOT(l1c1b6), NOT(l1c1b3));
23  bit l1c2b2 = AND3(b[1], l1c1b4, l1c1b1);
24  bit l1c2b3 = AND4(l0b1, l1c1b1, NOT(l1c1b5), NOT(l0c2b2));
25  bit l1c2b4 = AND4(b[1], l1c1b1, NOT(l1c1b8), NOT(l0c2b2));
26  bit l1c2b5 = AND4(b[1], l1c1b4, NOT(l1c1b7), NOT(l0c2b2));
27  bit l1c2b6 = AND3(l1c1b4, l1c1b1, NOT(l0c2b2));
28  bit l1c2b7 = AND(l1c1b5, l1c1b2);
29  bit l1c2b8 = AND5(NOT(b[1]), NOT(l1c1b4), NOT(l1c1b1), NOT(l0c2b1), l0c2b2);
30  bit l1c2b9 = AND3(l1c1b6, l1c1b3, NOT(l0b1));
31
32  bit l2c1b1 = AND(a[1], NOT(a[2]));
33  bit l2c1b2 = AND3(NOT(a[1]), a[2], NOT(b[2]));
34  bit l2c1b3 = AND3(NOT(a[1]), a[2], b[2]);
35  bit l2c1b4 = AND(NOT(m[1]), m[2]);
```

```
36   bit l2c1b5 = AND3(m[1], NOT(m[2]), NOT(b[2]));
37   bit l2c1b6 = AND3(m[1], NOT(m[2]), b[2]);
38   bit l2c1b7 = AND(NOT(a[1]), a[2]);
39   bit l2c1b8 = AND(m[1], NOT(m[2]));
40   bit l2c1b9 = AND4(NOT(m[1]), m[2], NOT(a[1]), a[2]);
41   bit l2c1bA = AND4(a[1], NOT(a[2]), m[1], NOT(m[2]));
42
43   bit l1c3b1 = OR6(l1c2b1, l1c2b2, l1c2b3, l1c2b4, l1c2b5, l1c2b6);
44   bit l1c3b2 = OR7(b[2], l1c2b1, l1c2b2, l1c2b3, l1c2b4, l1c2b5, l1c2b6);
45   bit l1c3b3 = OR5(l2c1b2, l2c1b5, l1c2b7, l1c2b8, l1c2b9);
46   bit l1c3b4 = OR3(l1c2b7, l1c2b8, l1c2b9);
47
48   s[2] = XOR9(l1c3b4, l2c1b3, l2c1b6, b[2], l2c1b2, l2c1b5, l1c3b1,
49               l2c1b1, l2c1b4);
50
51   bit l2c2b0 = AND3(b[2], l2c1b4, l2c1b1);
52   bit l2c2b1 = AND4(b[2], l1c3b1, NOT(l2c1b6), NOT(l2c1b3));
53   bit l2c2b2 = AND4(l1c3b1, l2c1b4, NOT(l2c1b2), NOT(l1c3b3));
54   bit l2c2b3 = AND4(l1c3b1, l2c1b1, NOT(l2c1b5), NOT(l1c3b3));
55   bit l2c2b4 = AND4(b[2], l2c1b1, NOT(l2c1b8), NOT(l1c3b3));
56   bit l2c2b5 = AND4(b[2], l2c1b4, NOT(l2c1b7), NOT(l1c3b3));
57   bit l2c2b6 = AND3(l2c1b9, l1c3b4, NOT(b[2]));
58   bit l2c2b7 = AND3(l2c1bA, l1c3b4, NOT(b[2]));
59   bit l2c2b8 = AND3(l2c1b4, l2c1b1, NOT(l1c3b3));
60   bit l2c2b9 = AND(l2c1b5, l2c1b2);
61   bit l2c2bA = AND5(NOT(b[2]), NOT(l2c1b4), NOT(l2c1b1), NOT(l1c3b2), l1c3b3);
62   bit l2c2bB = AND3(l2c1b3, l2c1b6, NOT(l1c3b1));
63   bit l2c2bC = AND4(NOT(l2c1b4), NOT(l1c3b1), l1c3b4, l2c1b3);
64   bit l2c2bD = AND4(NOT(l2c1b1), NOT(l1c3b1), l1c3b4, l2c1b6);
65
66   bit l3c1b1 = AND(a[2], NOT(a[3]));
67   bit l3c1b2 = AND3(NOT(a[2]), a[3], NOT(b[3]));
68   bit l3c1b3 = AND3(NOT(a[2]), a[3], b[3]);
69   bit l3c1b4 = AND(NOT(m[2]), m[3]);
70   bit l3c1b5 = AND3(m[2], NOT(m[3]), NOT(b[3]));
71   bit l3c1b6 = AND3(m[2], NOT(m[3]), b[3]);
72   bit l3c1b7 = AND(NOT(a[2]), a[3]);
73   bit l3c1b8 = AND(m[2], NOT(m[3]));
74   bit l3c1b9 = AND4(NOT(m[2]), m[3], NOT(a[2]), a[3]);
75   bit l3c1bA = AND4(a[2], NOT(a[3]), m[2], NOT(m[3]));
76
77   bit l2c3b1 = OR7(l2c2b0, l2c2b1, l2c2b2, l2c2b3, l2c2b4, l2c2b5, l2c2b8);
78   bit l2c3b2 = OR8(b[3], l2c2b0, l2c2b1, l2c2b2, l2c2b3, l2c2b4, l2c2b5, l2c2b8);
79   bit l2c3b3 = OR9(l3c1b2, l3c1b5, l2c2b6, l2c2b7, l2c2b9, l2c2bA, l2c2bB,
80                    l2c2bC, l2c2bD);
81   bit l2c3b4 = OR7(l2c2b6, l2c2b7, l2c2b9, l2c2bA, l2c2bB, l2c2bC, l2c2bD);
82
83   s[3] = XOR9(l2c3b4, l3c1b3, l3c1b6, b[3], l3c1b2, l3c1b5, l2c3b1,
84               l3c1b1, l3c1b4);
85
86   bit lxc2b0 = AND3(b[3], l3c1b4, l3c1b1);
87   bit lxc2b1 = AND4(b[3], l2c3b1, NOT(l3c1b6), NOT(l3c1b3));
88   bit lxc2b2 = AND4(l2c3b1, l3c1b4, NOT(l3c1b2), NOT(l2c3b3));
89   bit lxc2b3 = AND4(l2c3b1, l3c1b1, NOT(l3c1b5), NOT(l2c3b3));
90   bit lxc2b4 = AND4(b[3], l3c1b1, NOT(l3c1b8), NOT(l2c3b3));
91   bit lxc2b5 = AND4(b[3], l3c1b4, NOT(l3c1b7), NOT(l2c3b3));
92   bit lxc2b6 = AND3(l3c1b9, l2c3b4, NOT(b[3]));
93   bit lxc2b7 = AND3(l3c1bA, l2c3b4, NOT(b[3]));
94   bit lxc2b8 = AND3(l3c1b4, l3c1b1, NOT(l2c3b3));
95   bit lxc2b9 = AND(l3c1b5, l3c1b2);
96   bit lxc2bA = AND5(NOT(b[3]), NOT(l3c1b4), NOT(l3c1b1), NOT(l2c3b2), l2c3b3);
97   bit lxc2bB = AND3(l3c1b3, l3c1b6, NOT(l2c3b1));
98   bit lxc2bC = AND4(l3c1b8, NOT(l3c1b1), NOT(l2c3b1), l2c3b3);
99   bit lxc2bD = AND4(NOT(l3c1b4), NOT(l2c3b1), l2c3b4, l3c1b3);
100
101  bit lxc1b1 = AND(a[3], NOT(a[4]));
102  bit lxc1b2 = AND3(NOT(a[3]), a[4], NOT(b[4]));
103  bit lxc1b3 = AND3(NOT(a[3]), a[4], b[4]);
104  bit lxc1b4 = AND(NOT(m[3]), m[4]);
105  bit lxc1b5 = AND3(m[3], NOT(m[4]), NOT(b[4]));
106  bit lxc1b6 = AND3(m[3], NOT(m[4]), b[4]);
107  bit lxc1b7 = AND(NOT(a[3]), a[4]);
108  bit lxc1b8 = AND(m[3], NOT(m[4]));
109  bit lxc1b9 = AND4(NOT(m[3]), m[4], NOT(a[3]), a[4]);
110  bit lxc1bA = AND4(a[3], NOT(a[4]), m[3], NOT(m[4]));
111
112  bit lxc3b1 = OR7(lxc2b0, lxc2b1, lxc2b2, lxc2b3, lxc2b4, lxc2b5, lxc2b8);
113  bit lxc3b2 = OR8(b[4], lxc2b0, lxc2b1, lxc2b2, lxc2b3, lxc2b4, lxc2b5, lxc2b8);
114  bit lxc3b3 = OR9(lxc1b2, lxc1b5, lxc2b6, lxc2b7, lxc2b9, lxc2bA, lxc2bB,
115                   lxc2bC, lxc2bD);
116  bit lxc3b4 = OR7(lxc2b6, lxc2b7, lxc2b9, lxc2bA, lxc2bB, lxc2bC, lxc2bD);
117
118  s[4] |= XOR9(lxc3b4, lxc1b3, lxc1b6, b[4], lxc1b2, lxc1b5, lxc3b1,
119               lxc1b1, lxc1b4);
120
121  for(int i = 4; i < n - 1; i++) {
122      lxc2b0 = AND3(b[i], lxc1b4, lxc1b1);
123      lxc2b1 = AND4(b[i], lxc3b1, NOT(lxc1b6), NOT(lxc1b3));
124      lxc2b2 = AND4(lxc3b1, lxc1b4, NOT(lxc1b2), NOT(lxc3b3));
125      lxc2b3 = AND4(lxc3b1, lxc1b1, NOT(lxc1b5), NOT(lxc3b3));
126      lxc2b4 = AND4(b[i], lxc1b1, NOT(lxc1b8), NOT(lxc3b3));
127      lxc2b5 = AND4(b[i], lxc1b4, NOT(lxc1b7), NOT(lxc3b3));
128      lxc2b6 = AND3(lxc1b9, lxc3b4, NOT(b[i]));
129      lxc2b7 = AND3(lxc1bA, lxc3b4, NOT(b[i]));
130      lxc2b8 = AND3(lxc1b4, lxc1b1, NOT(lxc3b3));
131      lxc2b9 = AND(lxc1b5, lxc1b2);
132      lxc2bA = AND5(NOT(b[i]), NOT(lxc1b4), NOT(lxc1b1), NOT(lxc3b2), lxc3b3);
133      lxc2bB = AND3(lxc1b3, lxc1b6, NOT(lxc3b1));
134      lxc2bC = AND4(lxc1b8, NOT(lxc1b1), NOT(lxc3b1), lxc3b3);
135      lxc2bD = AND4(NOT(lxc1b4), NOT(lxc3b1), lxc3b4, lxc1b3);
136
137      lxc1b1 = AND(a[i], NOT(a[i + 1]));
138      lxc1b2 = AND3(NOT(a[i]), a[i + 1], NOT(b[i + 1]));
139      lxc1b3 = AND3(NOT(a[i]), a[i + 1], b[i + 1]);
140      lxc1b4 = AND(NOT(m[i]), m[i + 1]);
141      lxc1b5 = AND3(m[i], NOT(m[i + 1]), NOT(b[i + 1]));
142      lxc1b6 = AND3(m[i], NOT(m[i + 1]), b[i + 1]);
143      lxc1b7 = AND(NOT(a[i]), a[i + 1]);
144      lxc1b8 = AND(m[i], NOT(m[i + 1]));
145      lxc1b9 = AND4(NOT(m[i]), m[i + 1], NOT(a[i]), a[i + 1]);
146      lxc1bA = AND4(a[i], NOT(a[i + 1]), m[i], NOT(m[i + 1]));
147
148      lxc3b1 = OR7(lxc2b0, lxc2b1, lxc2b2, lxc2b3, lxc2b4, lxc2b5, lxc2b8);
149      lxc3b2 = OR8(b[i + 1], lxc2b0, lxc2b1, lxc2b2, lxc2b3, lxc2b4, lxc2b5, lxc2b8);
150      lxc3b3 = OR9(lxc1b2, lxc1b5, lxc2b6, lxc2b7, lxc2b9, lxc2bA, lxc2bB, lxc2bC, lxc2bD);
151      lxc3b4 = OR7(lxc2b6, lxc2b7, lxc2b9, lxc2bA, lxc2bB, lxc2bC, lxc2bD);
152
153      s[i + 1] = XOR9(lxc3b4, lxc1b3, lxc1b6, b[i + 1], lxc1b2, lxc1b5, lxc3b1,
154                      lxc1b1, lxc1b4);
155  }
156
157  bit lnc2b0 = AND3(b[n - 1], lxc1b4, lxc1b1);
158  bit lnc2b1 = AND4(b[n - 1], lxc3b1, NOT(lxc1b6), NOT(lxc1b3));
159  bit lnc2b2 = AND4(lxc3b1, lxc1b4, NOT(lxc1b2), NOT(lxc3b3));
160  bit lnc2b3 = AND4(lxc3b1, lxc1b1, NOT(lxc1b5), NOT(lxc3b3));
161  bit lnc2b4 = AND4(b[n - 1], lxc1b1, NOT(lxc1b8), NOT(lxc3b3));
162  bit lnc2b5 = AND4(b[n - 1], lxc1b4, NOT(lxc1b7), NOT(lxc3b3));
163  bit lnc2b6 = AND3(lxc1b9, lxc3b4, NOT(b[n - 1]));
164  bit lnc2b7 = AND3(lxc1bA, lxc3b4, NOT(b[n - 1]));
165  bit lnc2b8 = AND3(lxc1b4, lxc1b1, NOT(lxc3b3));
166  bit lnc2b9 = AND(lxc1b5, lxc1b2);
167  bit lnc2bA = AND5(NOT(b[n - 1]), NOT(lxc1b4), NOT(lxc1b1), NOT(lxc3b2), lxc3b3);
168  bit lnc2bB = AND3(lxc1b3, lxc1b6, NOT(lxc3b1));
169  bit lnc2bC = AND4(lxc1b8, NOT(lxc1b1), NOT(lxc3b1), lxc3b3);
170  bit lnc2bD = AND4(NOT(lxc1b4), NOT(lxc3b1), lxc3b4, lxc1b3);
171
172  bit lnc3b1 = OR7(lnc2b0, lnc2b1, lnc2b2, lnc2b3, lnc2b4, lnc2b5, lnc2b8);
173  bit lnc3b2 = OR7(lnc2b6, lnc2b7, lnc2b9, lnc2bA, lnc2bB, lnc2bC, lnc2bD);
174
175  s[n] = XOR4(lnc3b2, lnc3b1, a[n - 1], m[n - 1]);
```

Finally, we redefine $a, b \in \mathbb{Z}/m\mathbb{Z}$ being $m \in \mathbb{N}$, and we describe:

TABLE 8
OPERATIONS IN $\mathbb{Z}/m\mathbb{Z}$

	Modular operation
Figure 6 / Algorithm 1	$-(a+b)+m \mod(m)$
Figure 7 / Algorithm 2	$(a+b)-m \mod(m)$

We combine the two designs seen in **figure 6** and **figure 7**, and we connect their outputs to 1 multiplexer, to be able to choose between one result value or another. The final output will depend on the flags generated by the cells; for example, for the value *'00'* of the cell encapsulating **figure 6** circuit, we would select its output, and in the other case, the result of its other cell: encapsulating **figure 7** circuit.

We know that that this design can return 1 congruent element, which requires additional processing or interpretation. This might be an indication of that complexity has not been narrowed down to an $\mathcal{O}(n)$. However, it is possible to use congruent elements, achieving perfect balance when that operand is on the correct side of the binary relation. For example, in [6] we do exactly this in this way with modular inverses, and in [8,12] it is used for modular exponentiations and in [3,13] they are used within an additive context.

In addition, we can enrich the hardware design, incorporating a module with traditional computation, in case we need the value with the natural algebraic representation appending the classic algorithm and using a 3-inputs multiplexer.

Assisted by the **algorithm 2**, is practically identical with **algorithm 1**, we can calculate complexity costs in a very simple way. We will use cost-equations, denoted F. We simply have:

$$F(1) = F(2) = F(3) = F(4) = 5$$
$$F(n+1) = 1 + F(n)$$

It has been established that is always possible to obtain both, the result of a modular addition or, failing that, its structural congruent, with an algebraically correct representation, without coding problems or unwanted overflows, generating in all cases a formal element of a finite field and always in order complexity of $\mathcal{O}(n)$.

3 Discussion

It is not simple to make comparisons with current adder solutions, because the present solution works on finite fields and the adders work in $\mathbb{N}$ and manage a final carry-out.

Other works about modular addition, do exist, based on QCA [18] and CMOS technology [19], with quantum computing [20]... But this work discusses the theoretical complexity order required for mentioning operation. The proposal presented in this work is basically algorithmic, not technological or non-determinism.

Consequently, and without recourse to parallel computing of multiple-value, in this section we compare 3-arity algorithm proposed in **algorithm 2** with 2-arity algorithm for calculating the modular addition. It is noted that a solution based on 3-arity function does not reduce the operational functionality of the hardware: setting module to 0 have the same behavior as original algorithm.

For the above reasons, known binary adders methods: Kogge-Stone [21], Ladner-Fischer, Brent-Kung [22], Han-Carlson and Lynch-Swartzlander, we conceive as one by their gate level depth. Furthermore, we use as criteria:

- 5-inputs maximum for logic gates
- 4-bits blocks for carry-select and multiplexers
- 0 delay for NOT gates
- equivalent delay all gates, excluding NOT
- 261ps gate delay, multiple-inputs are ignored
- subtraction adding the additive inverse
- additive inverse using 2's Complement
- modular addition, adding module in 2's Complement

Based on these basic premises, we assumed that a logic gate takes 1 delays, D, to complete and n is the bit-width binary numbers. We therefore have:

TABLE 9
TIME COMPLEXITY ADDITION

Adder	Gate delay
Ripple-carry	$(3n-1)\cdot 3D$
Carry-lookahead	$(2+\sum_{i=1}^{log_4(n)} 3\cdot(n/4^i))\cdot 3D$
Carry-select	$(12+2\cdot((n/4)-1))\cdot 3D$
Parallel Prefix	$(2+log_2(n))\cdot 3D$
Carry-save	$2n\cdot 3D$
Booth-Karatsuba	$(4n-1)\cdot D$

The above table shows that the delays multiplied by 3, except for Booth-Karatsuba, the reason for this is that known methods require three executions: addition, 2's Complement and subtraction, and the last two operations are equally additions. **Algorithm 2** implements modular arithmetic in 1 addition. Carry-save for 3 operands requires two additions with propagation: one of them for the additive inverse.

Restrictions on input gates affect to multiple methods: carry-lookahead, carry-select... and the proposed concept based on Booth-Karatsuba, as we have seen, requires logic gates of 9-arity; but in the latter case, arity increases linearly with bit-width. In this work, we substitute them for 2 components. If this is not the case, and we use technology with 9-arity gates, we will have for Booth-Karatsuba technique: $(3n-1)*D$.

Continuing the discussion, we talked about time complexity, here we talk about space complexity:

TABLE 10
SPACE COMPLEXITY ADDITION

Adder	Gate count
Ripple-carry	$5n-3$
Carry-lookahead	$12\cdot(n/4)+\sum_{i=1}^{log_4(n)} 14\cdot(n/4^i)$
Carry-select	$2\cdot 17\cdot(n/4-1)+17+3\cdot(n/4-1)$
Parallel Prefix	$3n+log_2(n)\cdot n$
Carry-save	$3n+5n-3$
Booth-Karatsuba	$34n-38$

These results will be shown graphically to establish a comparison with known methods. Area of layout and number of transistors is in relation to space complexity analysis, and power consumed depends on time and space complexity. This work focuses on the complexity theory of the algorithms, and such parameters are subject to the physical technologies used.

The results obtained, with different bit-width, for time complexity are:

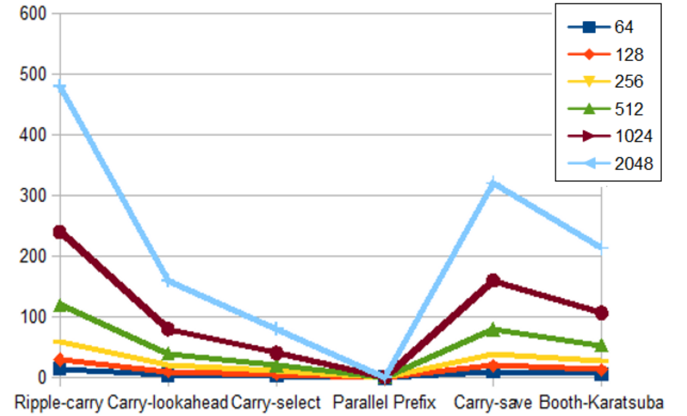

Fig. 5. Gate delay results (in ns).

And the empirical results obtained for space complexity are:

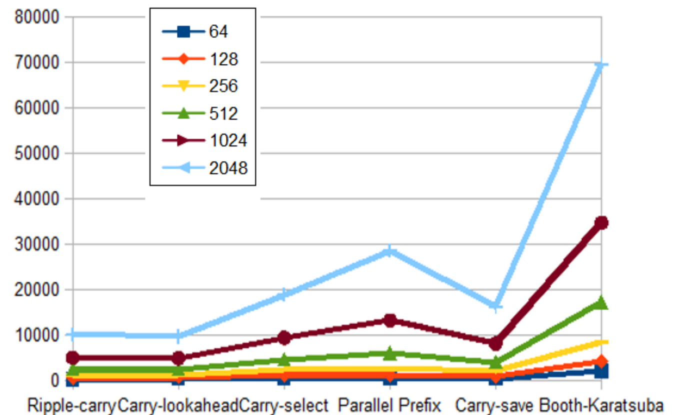

Fig. 6. Gate count results.

Firstly, it points out that the classical algorithm, Ripple-carry, is hampered by its $\mathcal{O}(n)$ in time, so that Booth-

Karatsuba substantially improves the data, because is not overloaded by a constant factor. In the same way, the other techniques perform better, since they have shorter times. The theoretical analysis is clearly illustrated by **graph 1**. While it is true that this work is not about the carry optimization, we insisted. The concepts introduced here questioning the nature of the analyzed operation. An operation that was traditionally calculated as three computations, when this can be seen as one single transformation.

We would like to emphasise, that proposal is not based on parallel computing. Order complexity is due to approach to the solution, founded on Booth-Karatsuba concepts. Thanks to its canceling capabilities and their vision based on primitives for transformation composition allows us to simultaneously calculations. The rest of adder techniques used duplicate calculations or ways to expedite carry-in.

The fact that the new concepts developed are based to more complex mathematical foundations it translates into greater impact int space requirements. **Graph 2** reveals that the increase in the number of improvements, also increases complexity of our circuit.

4 Conclusion

The designs proposed in this paper have a theoretical complexity order of $\mathcal{O}(n)$, and avoid the typical representation problems and overflow that are produced on finite arithmetics. Therefore, no reduction operations are required to accommodate the result in the module, easing the calculation burden of those extra operations.

The performance of the analyzed algorithms is conditional upon the ability to manufacture efficient logic gates with the arity used by the B-K simplification. This, of course, depends on the type of technology used in the diodes and the correct functioning at high speeds, the result of increased arity.

Conversely, the ability to calculate a modular addition in $\mathcal{O}(n)$ leads to an increase in hardware. This is normal as a result of supplementary techniques, for example, this occurs with simplifications based on Parallel Prefix. But B-K algorithm produces poor results, in this respect. Results in this work double in size to Parallel Prefix methods.

It is important to point out that if we incorporate a Parallel Prefix scheme to Booth-Karatsuba would imply that your order complexity is $\mathcal{O}(log_2(n))$. In other words, time complexity for modular addition is exactly the same as for addition, in a classical context, without implying constants in order complexity as a result of several computations.

This work shows that the algorithmic tools defined by Booth and Karatsuba are extrapolated to other scenarios and other algebraic contexts, simplifying more types of operation, and not only multiplication. In this study, we have utilized the concept in additive operations.

The new implementations proposed in this work seek to adjust their emerging methods with possible implications in Goppa Codes [23]: to extend the hardware word-length is simple. Also under study is applying the Parallel Prefix technique in calculating carry values with Booth-Karatsuba adder. Another line of research, we are working on different simplifications of truth tables, **table 6** and **table 7**, which allow a better management of space complexity.

Acknowledgments

The author would like to thank the UCM staff, for their precious time, constructive comments and recommendations, which helped me to improve the quality of this manuscript, for its facilities, support and access the tools and technical documentation. Finally, the author would like to thank the Complutense University of Madrid and its scholarship programs.

References

[1] Booth, A. D.: "A signed binary multiplication technique", The Quarterly Journal of Mechanics and Applied Math. Vol 4, No.2, 1951, pp. 236-240.
[2] Karatsuba, A.: "Multiplication of multidigit numbers on automata". Doklady Akademii Nauk SSSR. No.145, 1962, pp. 293-294.
[3] Ayuso, J.: "Booth algorithm operations addition and subtraction", 3C TIC. Vol 4, No.2, 2015, pp.113-119.
[4] Ayuso, J.: "Booth algorithm modular arithmetic operations of addition and subtraction", 3C TIC.Vol 4, No.3, 2015, pp. 222-229.
[5] Ayuso, J.: "Booth algorithm modular arithmetic operations of multiplication", 3C TIC. Vol 4, No.4,2015, pp. 255-221.
[6] Ayuso, J.: "Booth algorithm operations modular inverse", 3C TIC. Vol 5, No.2, 2016, pp. 28-41.
[7] Ayuso, J.: "Booth algorithm in signed-digit representation", 3C TIC. Vol 5, No.3, 2016, pp. 33-43.
[8] Ayuso, J.: "Booth algorithm in modular exponentiation operations", 3C TIC. Vol 6, No.2, 2017,pp. 1-12.
[9] Ayuso, J.: "Booth algorithm hardware operations addition and subtraction", 3C TIC. Vol 6, No.3,2017, pp. 1-9.
[10] Ayuso, J.: "Booth algorithm in arity with multiple operands", 3C TIC. Vol 6, No.4, 2017, pp.19-26.
[11] Ayuso, J.: "Karatsuba algorithm operations exponentiation", 3C TIC. Vol 7, 2018, pp. 13-20.
[12] Ayuso, J.: "Booth algorithm modular arithmetic for scalar multiplication operations", 3C TIC. 7(2),2018, pp. 10-35.
[13] Ayuso, J.: "Karatsuba algorithm in additive context", 3C TIC. 7(3), 2018, pp. 10-21.
[14] Ayuso, J.: "Booth-Karatsuba algorithm additive operations", 3C TIC. 7(4), 2018, pp. 30-59.
[15] Burks, A., Goldstein, H. and Von Neumann, J.: "Logical Design of an Electronic Computing Instrument". Princeton, 1946, pp. 39-48.
[16] Booth, A. D. and Britten, K. H. V.: "General Considerations in the Design of an Electronic Computer". Q.J. Mech. and Appl. Math. Vol 4, No.2, 1951, pp.236-240.
[17] Booth, A. D.: "A signed binary multiplication technique". Math. Vol 4, No.2, 1951, pp.236-240.
[18] Bilal, B.; Ahmed, S.; Kakkar, V.: "Modular Adder Designs Using Optimal Reversible and Fault Tolerant Gates in Field-Coupled QCA Nanocomputing". International Journal of Theoretical Physics volume 57, pages 13561375, 2018.
[19] Vergos, H. T.; Efstathiou, C.: "On the design of efficient modular adders". Journal of Circuits, Systems and ComputersVol. 14, No. 05, pp. 965-972, 2005.
[20] Kim, A.; Cho, S.; Seo, C.; Lee, S.; Seo, S.: "Quantum Modular Adder over $\mathbf{GF}(2^n - 1)$ without Saving the Final Carry". Appl. Sci. 11(7), 2949, 2021.
[21] Kogge, P. M.; Stone, H. S.: "A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations", IEEE Transactions on Computers. C-22 (8): 786793, 1973.
[22] Brent, R. P.; Kung, H. T.: "A Regular Layout for Parallel Adders", IEEE Transactions on Computers, Department of Computer Sciences, Carnegie-Mellon University, USA. C-31 (3): 260264, 1982.
[23] McEliece, R. J.; Rodriguez-Palanquex, M. C.: "AG Goppa Codes from Maximal Curves over determined Finite Fields of characteristic 2", IEEE International Symposium on Information Theory: 1066-1069, 2006.
[24] Sun-Mi, P.; Ku-Young, C.; Dowon, H.: "Space Efficient $\mathbf{GF}(2(m))$ Multiplier for Special Pentanomials Based on n-Term Karatsuba Algorithm", IEEE Access, Vol. 8, pp. 27342-27360, 2020.
[25] Rubinfield, L.P.: "A Proof of the Modified Booth's Algorithm for Multiplication", IEEE Transactions on Computers, Volume: C-24, Issue: 10, Oct. 1975.

[26] Balakumaran, R.; Prabhu, E: ”Design of high speed multiplier using modified Booth Algorithm with hybrid carry look-ahead adder”, 2016 International Conference on Circuit, Power and Computing Technologies, 18-19 March 2016.

[27] Takagi, N.: ”A VLSI Algorithm for Modular Division Based on the Binary GCD Algorithm”, IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, Vol.E81-A, No.5, pp.724-728, 1998.

[28] Weimerskirch, A.; Paar, C.: Generalizations of the Karatsuba Algorithm for Efficient Implementations, Technical Report, Ruhr-University Bochum, 2002.

[29] Mishra, S.; Pradhan, M.: ”Implementation of Karatsuba Algorithm using polynomial multiplication”, Indian Journal of Computer Science and Engineering 3(1),February 2012.

[30] Eyupoglu, C.: ”Performance Analysis of Karatsuba Multiplication Algorithm for Different Bit Lengths”, Procedia - Social and Behavioral Sciences Volume 195, pp. 1860-1864, July 2015.

[31] Patel, R. A.; Benaissa, M.; Powell, N.; Boussakta, S.: ”Novel Power-Delay-Area-Efficient Approach to Generic Modular Addition”, IEEE Transactions on Circuits and Systems I: Regular Papers, Volume: 54, Issue: 6, June 2007.

[32] Jaberipur, G.; Parhami, B.; Nejati, S.: ”On building general modular adders from standard binary arithmetic components, Conference Record of the Forty Fifth Asilomar Conference on Signals”, Systems and Computers, Nov. 2011.

[33] Takagi, N.: ”A Modular Inversion Hardware Algorithm with a Redundant Binary Representation”, IEICE Transactions on Information and Systems, Vol.E76-D, No.8, pp.863-869, 1993.

[34] Jarvinen, K.; Forsten, J.; Skytta, J.: ”Efficient Circuitry for Computing t-adic Non-Adjacent Form”, IEEE International Conference on Electronics, Circuits and Systems, 10-13 Dec. 2006.

[35] Ling, H.: ”High-Speed Binary Adder”, IBM Journal of Research and Development, Volume: 25, Issue: 3, March 1981.

[36] Hiasat, A.: ”General modular adder designs for residue number system applications”, IET Circuits, Devices Systems, Volume: 12, Issue: 4, 7 2018.

[37] Vergos, H.; Efstathiou, C.: ”On the Design of Efficient Modular Adders”, Journal of Circuits, Systems and Computers 14(5):965-972, October 2005.

[38] Migliore, V.; Mendez-Real, M. M.; Lapotre, V.; Tisserand, A.; Fontaine, C.; Gogniat, G.: ”Hardware/Software Co-Design of an Accelerator for FV Homomorphic Encryption Scheme Using Karatsuba Algorithm”, IEEE Transactions on Computers, 2018, Volume: 67, Issue: 3, pp. 335-347.

[39] Oseledets, I.: ”Improved n-Term Karatsuba-Like Formulas in GF(2)”, IEEE Transactions on Computers, 2011, Volume: 60, Issue: 8, pp. 1212-1216.

[40] Venkatachalam, S.; Adams, E.; Lee, H. J.; Ko, S.: ”Design and Analysis of Area and Power Efficient Approximate Booth Multipliers”, IEEE Transactions on Computers, 2019, Volume: 68, Issue: 11, pp. 1697-1703.

[41] Rafiq, A.; Chaudhry, S. M.; Awan, K. S.; Usman, M.: ”An efficient architecture of modified Booth multiplier using hybrid adder”, International Bhurban Conference on Applied Sciences and Technologies (IBCAST), 2021, pp. 12-16.

[42] Find, M. G.; Peralta, R.: ”Better Circuits for Binary Polynomial Multiplication”, IEEE Transactions on Computers, 2019, Volume: 68, Issue: 4, pp. 624-630.

[43] Rafferty, C.; O’Neill, M.; Hanley, N.: ”Evaluation of Large Integer Multiplication Methods on Hardware”, IEEE Transactions on Computers, 2017, Volume: 66, Issue: 8, pp. 1369-1382.

[44] Gu, Z.; Li, S.: ”A Novel Method of Modular Multiplication Based on Karatsuba-like Multiplication”, IEEE 27th Symposium on Computer Arithmetic (ARITH), 2020, pp. 7-10.

[45] Li, Y.; Ma, X.; Zhang, Y.; Qi, C.: ”Mastrovito Form of Non-Recursive Karatsuba Multiplier for All Trinomials”, IEEE Transactions on Computers, 2017, Volume: 66, Issue: 9, pp. 1573-1584.

J. AYUSO PÉREZ received his B.S. degree in computer science engineering from University Carlos III of Madrid, in 2006. He is currently a developer of software division in CT Engineering Group, Madrid, Spain. He was working in several development sectors: telecommunications, bank, road traffic, aeronautical.