Delay Table Representation and Function Minimization in Space-Time Algebra

Ido Guy
School of Electrical Engineering
Tel Aviv, Israel
idoguy@mail.tau.ac.il

Shlomo Weiss
School of Electrical Engineering
Tel Aviv, Israel
weiss@eng.tau.ac.il

ABSTRACT

Although developed for modeling spiking neurons, *space-time algebra (s-t)*, provides a general way of modeling temporal computation that extends beyond neurons. To date, neuron modeling as well as other function implementations have been primarily ad hoc in nature. This paper considers the problem of optimizing general s-t functions. Starting with function specifications in the form of "delay tables" and "delay terms", we develop and demonstrate an optimization algorithm for two-layer s-t networks, analogous to the Quine-McCluskey optimization algorithm for Boolean algebra. When applied to a simple spiking neuron model, the algorithm leads to a reduction of 21% of the components in a minterm-based design for the same model, and is on-par with existing designs for neurons in s-t algebra. The optimization algorithm is intended to be the cornerstone of optimized hardware design under the space-time algebra paradigm.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning**; **Learning paradigms**;

KEYWORDS

Artificial neural networks, Space-Time paradigm, Temporal spikes model, delay terms, delay table, function minimization

ACM Reference format:
Ido Guy and Shlomo Weiss. 2019. Delay Table Representation and Function Minimization in Space-Time Algebra. In *Proceedings of the 46 International Symposium on Computer Architecture, Phoenix, Arizona, June 2019 (ISCA 2019),* 9 pages.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

A recently developed theoretical framework for a temporal spiking computing model is the *Space-Time* (s-t) algebra [6, 8]. The Space-Time paradigm is intended for designing systems that use time as the resource of computation. While the original motivation behind such a paradigm is to provide the functionality of neurons, it is useful for other applications as well.

ISCA 2019, June 2019, Phoenix, Arizona

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

In [6, 8] it is shown that the primitive operations *min, less-than (lt) & delay* are functionally complete for functions that can be specfied with a bounded function table. This is referred to as *bounded functional completeness*. These primitives are strongly conjectured to be functionally complete for all s-t functions. Along with the *max* primitive (which can be created using only min & less-than), one can begin with any function table defining a space-time function and transform it into a "Delay Table" consisting of a "delay term" for every row of the funtion table, as illustrated in Figure 1.

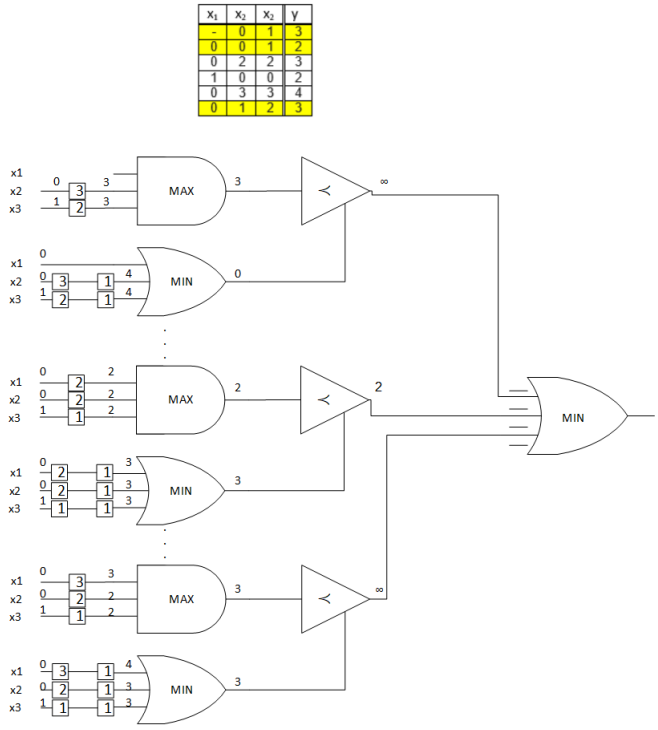

Figure 1: Top: a normalized function table having three inputs (x_i) and one output (y). Bottom: canonical form showing delay terms for the three highlighted table rows. If input $(0, 0, 1)$, corresponding to the middle highlighted term, is applied, then the resulting network values are shown in the figure near the outputs of the gates. Increments (which model delays) are shown as boxes enclosing a delay amount. The dashed lines in the input of the second-level min gate are meant to signify that there are other inputs to the min gate that were not drawn in the figure.

A *delay term* is composed of delay units, *min* and *max* operators and a *less-than* operator. It has an excitatory part (where the input vector is fed into the *max* operator through a set of delays) and an inhibitory part (where the input vector is fed into the *min* operator through a set of delays and an additional 1-unit delay). The output of the excitatory part is possibly blocked by the output of the inhibitory part by use of the *lt* operator. See Figure 2.

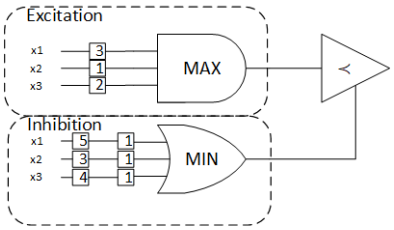

Figure 2: An example delay term. The upper half (max) is excitatory; the lower half (min) is inhibitory.

A delay term corresponding to a single function table row has the same delay values in the excitatory and inhibitory parts (excluding the additional 1-unit delay of the inhibitory part). As we will see later on, using larger delay values in the inhibitory part will implement more than one function table row.

An optimized design in s-t algebra should contain the minimum number of delay terms, where the *min* among them is taken as the output of the function. This is analogous to a two-level binary network formed as a product of sums in Boolean algebra. A worthy endeavor, in that case, would be to develop an algorithm for finding such an optimized network, equivalent to algorithms such as Quine-McCluskey [2] in Boolean algebra.

The Quine-McCluskey algorithm is performed by combining different minterms into implicants, and then further combining those implicants, until only the prime implicants are discovered. In this work, we will explore and then formalize the ways in which delay terms can be combined in s-t algebra.

1.1 Contributions

(1) Thus far, no considerations of optimization or budget constraints were taken in previous work made in the realm of space-time algebra, except for designs of specific networks such as a sorting network and a single neuron model. This work introduces the concept of general optimization into the s-t paradigm, in the same way that Karnaugh maps and the Quine-McCluskey algorithms contribute to the basic building blocks of Boolean algebra.
(2) We introduce Delay Tables. Delay tables are a new way to describe s-t functions, one which allows to see the relations between different rows of a function table, as well as describe more complex functions.
(3) We describe and formalize delay term behavior in space-time algebra. Although the structure of delay terms was already found in previous work, no formal definition of how delay terms behave and how they relate to rows in function tables was made in previous work.
(4) We articulate a minimization algorithm similar to Quine-McCluskey, with examples that show how it is used and how many resources can be saved by applying the algorithm.

1.2 Overview

In Section 1 we describe characteristics of the Space-Time algebra and provide motivation for this work. In Section 2 we summarize work that is related in some aspects of the present research. Section 3 provides additional details on the Space-Time algebra. Section 4 describes delay term behavior, which is formalized in Section 5. In section 6 we introduce Delay Tables, a new way to describe s-t functions. In Section 7 we articulate a minimization algorithm similar to the Quine-McCluskey algorithm in Boolean algebra. This is followed in Section 8 by the description of minimizing a neuron model. Finally we summarize this work in Section 9.

2 RELATED WORK

The space-time paradigm is a theoretical foundation [6, 8] for Time-based Neural Networks (TNNs). In TNNs information is encoded in the timing of voltage spikes that show up on separate interconnection lines, relative to the time of the first spike. There is at most one spike on a line. As opposed to TNNs, in Rate-based Neural Networks (RNNs) information is encoded in the rate of spikes occurring on a specific interconnection line. TNN is the model that is most relevant to the present research.

The space-time paradigm is a subtype of unary computing - all computation and communication in the paradigm is based on unary codes. This theoretical framework can be physically implemented using standard CMOS components used for binary computers by applying the principles of "Race Logic" [1]. Work by Najafi et al. [3] shows that unary computation paradigms can have an advantage for certain computation tasks (in this case, sorting applied to median filtering).

This paper is based on the work described in [6, 8] on the space-time model. We elaborate on a specific aspect of space-time functions, namely delay terms, delay tables and minimization. We develop algorithms that have a similar role in s-t algebra as the Quine-McCluskey algorithms in Boolean algebra.

3 BACKGROUND

Space-time algebra is a mathematical framework that provides a theoretical foundation for Temporal Neural Networks (TNNs), but can also be used for other, more general applications. Here we only provide a brief description necessary for the remainder of this paper. For more details the reader is referred to [8].

In space-time networks information is encoded as the precise timing of voltage spikes. An example is shown in Figure 3. At the top row time is shown in time units. A '1' on line x_i at a specific time indicates that a spike was produced at that time. The default value is ∞ for a timeline on which there is no spike. The intuitive meaning of ∞ is that one has to wait a very long (infinite) time until a spike arrives on that line.

Space-Time functions were formally defined in [9] and again in [7]. The definition from the latter is quoted here:

> Define the set N_0^∞ to consist of 0, the natural numbers, and the special element ∞, which models the situation

time	0	1	2	3	4	5	6	7	...	...	∞
x_1	1										
x_2						1					
x_3			1								
x_4											
x_5							1				

Figure 3: Space-time encoding of the sequence < 0 5 2 ∞ 6 >.

where there is no spike on a given communication line. The symbol "∞" has defined properties typically associated with infinity. That is: $\infty + n = \infty$ and $\infty > n$ for all $n \in N_0^\infty$.

Definition: A function $z = F(x_1 \ldots x_q), x_1 \ldots x_q, z \in N_0^\infty$ is a *Space-Time Function* if it satisfies the following properties:

1) *computability*: F implements a computable total function.
2) *causality*: For all $x_j > z, F(x_1 \ldots, x_j, \ldots x_q) = F(x_1 \ldots, \infty, \ldots x_q)$ and if $z \neq \infty$, then $z \geq x_{min}$.
3) *invariance*: $F(x_1 + 1, \ldots, x_q + 1) = F(x_1, \ldots x_q) + 1$.

The s-t algebra defines three basic functions [8]. Refer to Figure 4. The *increment* function produces an output spike one time unit later than its input. The *min* function (∧) produces an output spike as soon as the first spike arrives at its inputs. The *less-than (lt)* function (≺) produces an output spike at the same time as the input spike if the input spike arrives before the control spike.

These three functions make up a functionally complete set for bounded function tables, but it is worth noting that the *max* function (∨), which produces a spike only after the last spike arrives at its inputs, is convenient but not essential for completeness.

Functions in s-t algebra can be described by function tables, very similar to truth tables in Boolean algebra (see example in top of Fig 1). There is a single column for every input to the function, and the rightmost column is the output value of the function. The symbol "∞" is mostly written as "-" in such tables.

These function tables are normalized and bounded. "Normalized" means that at every row in the table, at least one input has a value of zero. This is due to the "invariance" property of s-t algebra functions - if one would like to determine the output value of an unnormalized input, one can first normalize the input, find the row corresponding to the normalized input in the function table, and then add the normalizing constant back to the output value in the function table. "Bounded" means that the table does not contain the entire unbounded set N_0^∞, but only values up to a certain finite value. It is assumed that if a possible input does not appear in the function table, then the output value corresponding to that input is ∞.

However, function tables have some significant shortcomings. Function tables can be used to show bounded functional completeness, i.e., any function described by a function table can be implemented using min, max, lt and delay (by implementing each row in the table as a delay term). Observe that this completeness result holds only for functions with bounded function tables. Consequently, this does not include all of the s-t functions, and excludes

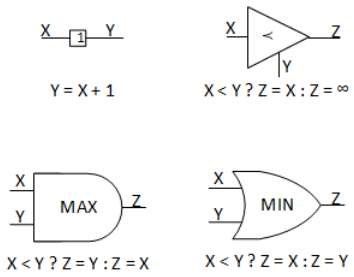

Figure 4: Basic functions of space-time algebra.

some important ones. For example, the max function $a \vee b$ cannot be specified by a bounded function table.

This apparent inadequacy raises the issue of whether a conventional function table specifying delay terms, analogous to a truth table, is the best way to specify s-t functions. To potentially resolve this issue a more complete tabular way of specifying functions is given in section 6.

4 DELAY TERM BEHAVIOR

In this section, we explore some sample delay terms and check their relation to an example single-row delay term. We will deduce from this exploration the general behavior of delay terms, and which function table rows they cover.

4.1 Combining Two Single-row Delay Terms

Let us start with an example of a simple single-row delay term, one where *f(2,1,0) = 3* (Figure 5).

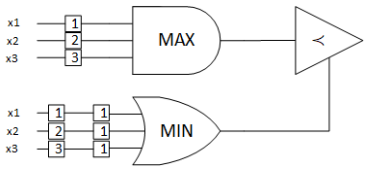

Figure 5: The delay term f(2,1,0) = 3.

It was shown [9] that a delay term where $d_i^I < d_i^E$, for any i, will have an output value of ∞ for every possible set of inputs, so it will therefore be fruitless to try and reduce the d_i^I values or increase the d_i^E values. We will examine a case where d_1^I is increased by 1, i.e., a delay term such as described in Figure 6.

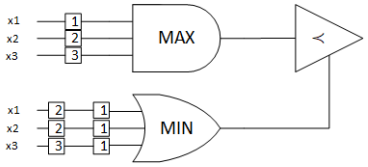

Figure 6: A delay term where d_1^I is increased by one from the delay term f(2,1,0)=3.

In such a case, if we use an input of (2, 1, 0), we will get an output of 3, just like in the original delay term. So, increasing d_1^I did not change the behavior of the design for this specific input. But, unlike the original design, there is one more input that will return an output value different than ∞; i.e. (1, 1, 0). In that case, the output value will be 3 as well. By increasing d_1^I from 1 to 2, we have created a delay term which is a combination of the delay terms $f(2,1,0) = 3$ and $f(1,1,0) = 3$.

This nice property can be reached by increasing any one value of d_i^I by one, but special care needs to be taken in cases where doing so increases the maximum value of d_i^I. In our specific example, this will happen if we increase d_3^I from 3 to 4, as illustrated in Figure 7.

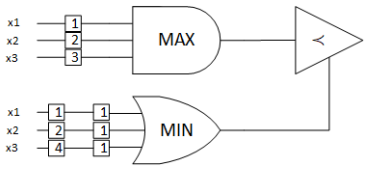

Figure 7: A delay term where d_3^I is increased by one from the delay term f(2,1,0)=3.

In this case, as before, the input (2, 1, 0) will provide an output value of 3, and the additional input that will not output a value of ∞ will be (3, 2, 0), but unlike the previous example, it will output a value of 4. So, increasing the maximum value of the delays in the inhibition section will make the resulting delay term have the ability to output values higher than the output of the original delay term.

4.2 Combining General Delay Terms

We take the delay term from Fig. 6, and increase the value of d_1^I by an additional 1, as illustrated in Fig. 8. We will get a delay term

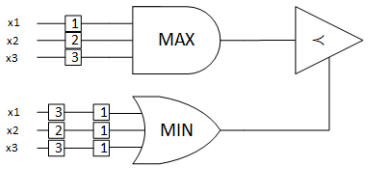

Figure 8: Delay term created by increasing the d_1^I of the delay term f(2,1,0)=3 by 2.

that outputs a value of 3 for the inputs (2, 1, 0), (1, 1, 0) and (0, 1, 0), and a value of ∞ for every other input. So, it appears that the number of inputs that do not output a value of ∞ is $d_1^I - d_1^E + 1$. But, this is only for the case where a single input gets its inhibitory delay increased. The picture becomes clearer once we examine a case like in Fig. 9.

In this case, there will be a total of 6 inputs that will provide an output different than ∞. More generally, the number of inputs will be $(d_1^I - d_1^E + 1)(d_3^I - d_3^E + 1)$.

It is important to note that this multiplication does not occur as expected when $d_i^I > d_i^E$ for every i in the same delay term. This case is explored in Fig. 10.

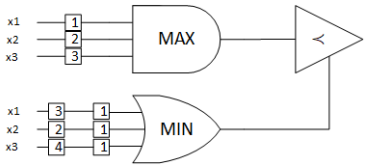

Figure 9: Delay term created by increasing the d_1^I of the delay term f(2,1,0)=3 by 2 and the d_3^I by 1.

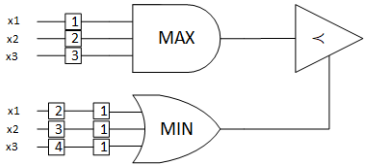

Figure 10: A delay term where all d_i^I values are increased by one from the delay term f(2,1,0)=3.

This delay term has a total of 7 inputs that will provide an output different than ∞, rather than the expected 8 that would result from the multiplication $(d_1^I - d_1^E + 1)(d_2^I - d_2^E + 1)(d_3^I - d_3^E + 1)$. It would seem that the "missing" input that we would have expected is identical to the input corresponding to the original delay term, but with an output value 1 higher than the original, i.e. f(2,1,0)=4 (which matches the delay term with delay values $d_1^E = 2, d_2^E = 3, d_3^E = 4$ - all of which higher by one than the original d_i^E values).

It would seem that the restriction is that combined delay terms cannot combine cases where the same inputs result in different outputs, which should not be regarded as an actual restriction since a valid function cannot result in more than one output for the same input.

5 FORMALIZATION OF DELAY TERM BEHAVIOR

In this section we formalize the guiding rules of delay term behavior based on the observations from the previous section. We present an algorithm to predict which function table rows are covered by a delay term.

5.1 Deduced rules of delay term behavior

If we generalize the behavior observed by the previous examples, we can come to a few rules regarding delay term behavior in s-t algebra:

a. The number of function table rows that a delay term implements is given by (1):

$$N = \prod_{i=1}^{n}(d_i^I - d_i^E + 1) - \prod_{i=1}^{n}(d_i^I - d_i^E) \quad (1)$$

Where N is the number of rows and n is the number of inputs.

b. A delay term (normalized) output value can only be in the following range, or infinity:

$$\max_i d_i^E \leq out \leq \max_i d_i^I \quad (2)$$

c. The rows covered by a delay term with 3 inputs can be found with algorithm 1, which prints all of the rows and their output value (assuming that $d_i^E < d_i^I$, of course). A similar algorithm can be written for any number of inputs.

Algorithm 1 Function table rows covered by a delay term with 3 inputs.

```
for i = d1^E; i <= d1^I; i++ do
    for (j = d2^E; j <= d2^I; j++) do
        for (k = d3^E; k <= d3^I; k++) do
            if i > d1^E && j > d2^E && k > d3^E then
                continue
            m = max(i, j, k)
            print((m − i, m − j, m − k), m)
```

5.2 Rules

Note that if rules **a** and **b** follow from rule **c**, since they can be deduced from the algorithm in rule **c**.

Rule a simply says that the number of function table rows that a delay term implements is equal to the total number of iterations, i.e. the total number of "print" commands in the algorithm on rule **c**.

Rule b is also observed from the algorithm - the minimal output value is the maximum value among all minimum values of all of the iterators (max(i, j, k) when i, j, k are minimal, i.e. when they're all equal to d_i^E), and the maximal output value is the maximum value among all maximum values of all of the iterators (max(i, j, k) when i, j, k are maximal, i.e. when they're all equal to d_i^I).

6 DELAY TABLE REPRESENTATION OF FUNCTIONS

As discussed before, representing s-t function using function tables has significant drawbacks. An additional drawback that emerges from the discussion of delay terms is that two function table rows that can be combined into a single delay term might look very different, making it difficult to accurately represent delay terms using the function table form.

In order to specify delay terms, define a Delay Table (DT) where the table rows define algebraic delay terms in the following manner.

Each input variable, x_i, has two associated delay variables, d_i^E and d_i^I, which can be assigned any member of N_0^∞. An example DT is given in Figure 11. Observe there is no specified output column in the DT; the output for a given set of inputs in not explicitly specified.

Each row in the DT defines a delay term of the specified function. For a given row, the associated delay term is:

$\bigvee_{i=1}^{n}(x_i + d_i^E) \prec \bigwedge_{j=1}^{n}(x_i + d_i^I + 1)$

The symbol "-" means that the input is not connected to the delay term at all.

Special care needs to be taken for ∞ delay. While it conceptually has the meaning of "infinity" (i.e., infinite delay, or open circuit), when added with other delays, it behaves algebraically as -1 when added to other delay units. For example: since the inhibitory part includes $(x_i + d_i^I + 1)$, if $d_i^I = \infty$, we would get $(x_i + d_i^I + 1) = (x_i - 1 + 1) = x_i$, which would mean that any value of x_i other than ∞ would inhibit the delay term.

x_1		x_2		x_3	
d_1^E	d_1^I	d_2^E	d_2^I	d_3^E	d_3^I
0	2	1	∞	-	-
1	3	0	0	2	4
1	2	0	∞	0	∞

Figure 11: Example Delay Table (DT).

In the above example, the delay term implemented in the first row is $(x_1+0)\vee(x_2+1) \prec (x_1+2+1)\wedge(x_2-1+1)$. The second row delay term: $(x_1+1)\vee(x_2+0)\vee(x_3+2) \prec (x_1+3+1)\wedge(x_2+0+1)\wedge(x_3+4+1)$; the third row delay term: $(x_1 + 1) \vee (x_2 + 0) \vee (x_3 + 0) \prec (x_1 + 2 + 1) \wedge (x_2 - 1 + 1) \wedge (x_3 - 1 + 1)$.

A function specified as rows in a function table can be re-cast into DT form. The transformation is straightforward and proceeds as follows. For a function table row containing entries for inputs x_i and output y, the DT entries are defined as: if $x_i \neq \infty$ then $d_i^E = d_i^I = y - x_i$ else if $x_i = \infty$ then $d_i^E = d_i^I = \infty$. For example, if we have a row in a function table (2,0,1) with an output value of 3, then the corresponding delay table row would have $d_1^E = d_1^I = 3 - 2 = 1, d_2^E = d_2^I = 3 - 0 = 3, d_3^E = d_3^I = 3 - 1 = 2$.

The strange behavior of ∞ delay can be intuitively explained by thinking about the causality property of s-t algebra when performing the transformation from function table to delay table: according to causality, if the input x_i is larger than the output y, then the output is the same as $x_i = \infty$. The other side of the coin is, that it's also the same as $x_i = y + 1$. If we calculate $d_i^I = y - x_i = y - (y + 1)$, we get $d_i^I = -1$. This notation will also be useful for the minimization algorithm specified in the following sections.

6.1 Minimization Theorem

Theorem 6.1. *Let f_1 and f_2 be two function table rows such that $f_1(x_1, x_2) = ((x_1 + d_1 \vee x_2 + d_2) \prec (x_1 + d_1 + 1 \wedge x_2 + d_2 + 1))$ and $f_2(x_1, x_2) = ((x_1 + d_1 \vee x_2 + d_2 + 1) \prec (x_1 + d_1 + 1 \wedge x_2 + d_2 + 2))$. The following identity holds for every integer value of x_1, d_1, x_2, d_2:*

$$f_1(x_1, x_2) \wedge f_2(x_1, x_2) = ((x_1 + d_1 \vee x_2 + d_2) \prec (x_1 + d_1 + 1 \wedge x_2 + d_2 + 2)) \quad (3)$$

This identity is the basis for the minimization algorithm that will be described in the next section. To put in simple terms, this identity means that two function table rows that have the same delay values for each input except for one where the difference between delays is exactly 1 can be replaced by a single delay term.

7 ARTICULATING A MINIMIZATION ALGORITHM SIMILAR TO QUINE-MCCLUSKEY

7.1 Explanation with an example

While input values of some function table row might not always have an obvious connection to the input values of another row that

can be combined with it to a delay term, delay values (which are use in the DT representation) for a certain function table row are always a distance of 1 from delay values of adjacent rows (i.e., all delay values are equal, except for one input where the delay values differ by 1) - this is also supported by the minimization theorem. For example, the rows $f(2,1,0) = 3$ and $f(3,2,0) = 4$ seem very different when presented in this inputs and output values way, but when calculating their "delay representation" in the DT, one can see that they are actually very similar: (1, 2, 3) and (1, 2, 4).

Therefore, the first step of an optimization algorithm would be to take an input function and represent it using a delay table. Once that is done, the following suggested optimization algorithm works very similarly to the Quine-McCluskey algorithm:

1) Write the function in delay table format, as described in section 6.

x_1	x_2	x_3	y
0	0	0	1
0	1	1	2
1	0	1	2
0	0	1	2
0	2	2	3
1	0	0	2
0	3	3	4
0	1	2	3

==>

	d_1^E	d_2^E	d_3^E
A	1	1	1
B	2	1	1
C	1	2	1
D	2	2	1
E	3	1	1
F	1	2	2
G	4	1	1
H	3	2	1

2) Like in Q-M, order all rows (analogous to "minterms" in Boolean algebra) in groups according to the sum of all values in the string. If the row is used in the next step, we put a check mark next to it.

Group	Name	Value	Used in next step
3	A	(1,1,1)	✓
4	B	(2,1,1)	✓
	C	(1,2,1)	✓
5	D	(2,2,1)	✓
	E	(3,1,1)	✓
	F	(1,2,2)	✓
6	G	(4,1,1)	✓
	H	(3,2,1)	✓

3) The next step is to find all possible combinations of two rows into a single delay term (like finding combinations of minterms into implicants in Q-M for Boolean algebra). Note that a row can only be combined with another row if they are from adjacent groups (i.e. a row from group 4 can only be combined with a row from group 3 or group 5, and not with any row from group 4). Write out all possible combinations in the table of the next step (you only need to check for each row if it can be combined with rows from the next group). If two rows are combined, write the delay value where they differed as the range of the two values (for example, (1,2,3) and (1,2,4) are combined into (1,2,3..4)). The groups in this step's table are ordered according to the range of sums of the rows (so, (1,2,3..4) is a combination of a row from group 6 and a row from group 7, so in the second step's table the delay term will be in group 6..7).

Group	Name	Value	Used in next step
3..4	AB	(1..2,1,1)	✓
	AC	(1,1..2,1)	✓
4..5	BD	(2,1..2,1)	✓
	CD	(1..2,2,1)	✓
	BE	(2..3,1,1)	✓
	CF	(1,2,1..2)	
5..6	DH	(2..3,2,1)	✓
	EG	(3..4,1,1)	✓

4) In the next step, we will combine delay terms from the second step's table with delay terms from the same table. Generally speaking, if a delay term is in group *x..x+i*, then we will look for matching delay terms to combine with it in group *x+1..x+i+1* in the same table. A delay term can be combined with another delay term if the delay representation of every input is identical for both delay terms, except for one, that should be of the form *d..d+i* in the first delay term and of the form *d+1..d+i+1* in the second delay term. For example, the delay term (1..2,1,1..3) can be combined with the delay term (2..3,1,1..3) into (1..3,1,1..3). It can also be combined with the delay term (1..2,2,1..3) into (1..2,1..2,1..3), and with the delay term (1..2,1,2..4) into (1..2,1,1..4). We continue these steps until no more combinations can be made.

Group	Name	Value	Used in next step
3..5	ABCD	(1..2,1..2,1)	✓
	ABE	(1..3,1,1)	✓
4..6	CDH	(1..3,2,1)	✓
	BEG	(2..4,1,1)	✓

Final step:

Group	Name	Value
3..6	ABCDEH	(1..3,1..2,1)
	ABEG	(1..4,1,1)

5) Any delay term that does not have a check mark next to it is a prime delay term. We will create a prime delay term chart to figure out which prime delay terms are essential (like in the Q-M algorithm). If a certain column (that represents a certain table row in the original table) only has X in one row, the delay term corresponding to that row is essential, and must be used.

	A	B	C	D	E	F	G	H
CF			X			X		
ABCDEH	X	X	X	X	X			X
ABEG	X	X			X		X	

We can see that F, G and H are such rows, so the delay terms CF, ABEG and ABCDEH are all essential, in this case. There is no problem with the fact that both CF and ABCDEH cover the row C - if the input corresponding to C will be given, they will both create the same output value, and the min gate at the second level of the optimal network will pick the correct value still. It is worth noting that G, for instance, can also be covered by EG, instead of ABEG (since A and B are covered by ABCDEH). These delay terms use very similar primitive gates, but ABEG would require less hardware to implement, since it has a lower d_1^E value, while all other attributes of these two delay terms are identical, so it would be better to use the "larger" delay term, ABEG.

7.2 Secondary optimization and using different primitive functions

While this work focused on implementation of delay terms using only the primitive s-t functions *min, lt, max & delay*, there are other primitive functions in s-t algebra, with other ways to implement delay terms. These will not be discussed here, but it is important to note that it is possible to use the minimization algorithm regardless of which primitive functions are used to implement the delay terms; Every delay term with n inputs can be described by 2n delay values - half for the inhibitory part, and half for the excitatory part. This is due to the fact that any delay term implementation must be equivalent to the following s-t function:

$$\bigvee_{i=1}^{n}(x_i + d_i) < \bigwedge_{j=n+1}^{2n}(x_j + d_j + 1) \tag{4}$$

Where x_i (for i ranging from 1 to n) are the inputs and d_i (for i ranging from 1 to 2n) are the delay values.

While the results of the algorithm are independent of the implementation, different implementations might prove to be more efficient than others. Therefore, after applying the minimization algorithm, one may consider the implementation decisions as a secondary optimization process, where the most efficient implementation is chosen for each delay term.

8 MINIMIZING A NEURON MODEL

As mentioned before, the space-time algebra paradigm is able to simulate neurons. In [6], a design was suggested for a neuron model using space-time algebra primitive components, as well as bitonic sort networks. In this design, a Type A response function ([4]) of an input signal is modeled by applying an up delay to indicate when the response function rises and a longer down delay to indicate when the response function falls. The signals with the up delay are given as inputs to an up sort network and the signals with the down delay are given as inputs to a down sort network. he amplitude of the input signal is decided by how many times the input signal is duplicated. We have applied the minimization algorithm to a simple neuron model. The model has 3 inputs, x1, x2 and x3. Each of the inputs has a rectangular response function:

- x1 has a response function that rises 1 time unit after x1, falls 4 time units after x1, and has an amplitude of 3 during this window.
- x2 has a response function that rises 1 time unit after x2, falls 3 time units after x2, and has an amplitude of 2.
- x3 has a response function that rises 1 time unit after x3, falls 2 time units after x3, and has an amplitude of 1.

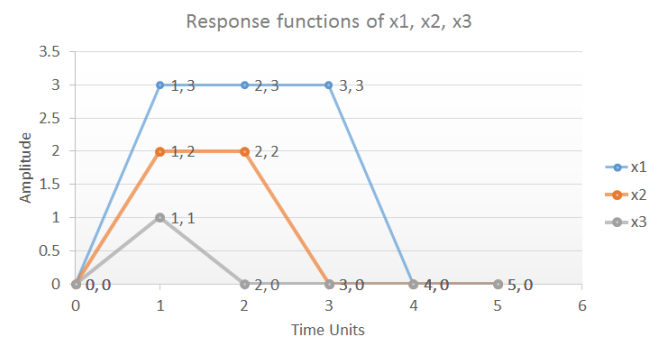

Figure 12: Response functions of x1, x2, and x3 in the neuron model, assuming that all three input spikes arrive at t=0.

The function table for this neuron model was calculated, then it was converted to a delay representation as can be seen in Table 1. This function table is bounded by 4 (i.e., we did not include cases where one of the inputs has a value greater than 4), since the response functions themselves are bounded by 4 and every behavior beyond that would just be a replication of behavior that was already covered by other rows (for example: if x3=0, x2=3 and x1=4, it would present the same behavior as x3=0, x2=4 and x1=5).

One might note that there are no input values of ∞ in this function table, but there are input values that are greater than the output values by 1 (examples: rows a, g, m). This is because input values of ∞ have the same functional meaning as any other input value that is greater than the output value, due to s-t algebra's causality principle. This observation allows us to calculate the delay representation of ∞ - it is -1 (negative one).

Every row received a name as one of the letters of the English alphabet, where using case sensitivity allowed us to give all rows a unique one-letter name. There are a total of 38 rows in this function table.

After applying the minimization algorithm, 11 prime delay terms were found, as described in table 2. Out of them, the first 8 delay terms appearing in the table were chosen as essential prime delay terms that will be used in the optimized design for the neuron model.

The optimized implementation can be viewed in Figure 13. Note that a '-1' value for an excitatory delay means that there is no connection of the corresponding input to the max component.

When compared to the naive single-row delay terms implementation, we see a reduction to about 21% of the required min, max and lt gates, and less than 50% of the required delay units (a "delay unit" is an increment by 1 time unit). When compared to the neuron design suggested by [6], we use about 33% of the required min, max and lt gates in the neuron design, but we require significantly more delay units (about 3.5 times more). According to [5], min and max each

Table 1: Function table for the neuron model, and its delay representation

x1	x2	x3	out	name	d1	d2	d3	group	used in next step
0	2	0	1	a	1	-1	1	1	V
2	0	0	1	b	-1	1	1	1	V
0	1	0	1	c	1	0	1	2	V
1	0	0	1	d	0	1	1	2	V
1	3	0	2	e	1	-1	2	2	V
3	0	1	2	f	-1	2	1	2	V
3	1	0	2	g	-1	1	2	2	V
0	0	0	1	h	1	1	1	3	V
1	2	0	2	i	1	0	2	3	V
2	0	1	2	j	0	2	1	3	V
2	1	0	2	k	0	1	2	3	V
2	4	0	3	l	1	-1	3	3	V
4	2	0	3	m	-1	1	3	3	V
0	1	1	2	n	2	1	1	4	V
1	0	1	2	o	1	2	1	4	V
1	1	0	2	p	1	1	2	4	V
2	3	0	3	q	1	0	3	4	V
3	2	0	3	r	0	1	3	4	V
0	0	1	2	s	2	2	1	5	V
0	2	2	3	t	3	1	1	5	V
2	0	2	3	u	1	3	1	5	V
2	2	0	3	v	1	1	3	5	V
0	1	2	3	w	3	2	1	6	V
0	2	1	3	x	3	1	2	6	V
0	3	3	4	y	4	1	1	6	V
3	0	3	4	z	1	4	1	6	V
0	2	3	4	A	4	2	1	7	V
0	3	2	4	B	4	1	2	7	V
0	4	4	5	C	5	1	1	7	V
3	0	2	4	D	1	4	2	7	V
4	0	4	5	E	1	5	1	7	V
0	3	1	4	F	4	1	3	8	V
0	3	4	5	G	5	2	1	8	V
0	4	3	5	H	5	1	2	8	V
4	0	3	5	I	1	5	2	8	V
0	4	2	5	J	5	1	3	9	V
4	0	2	5	K	1	5	3	9	V

require a single Boolean gate to implement using CMOS, lt requires 3 Boolean gates, and each delay unit requires a flip-flop. Therefore, while the delay term implementation of the neuron model require much less gates than the neuron design, the two implementations will require a very similar amount of resources. The comparison can be seen in Table 3. "Naive Implementation" is the naive neuron model implemented by use of only the delay terms generated from the function table, "Optimized Implementation" is the neuron implementation given by applying the optimization algorithm, and "Neuron Model" is the neuron design suggested in [6].

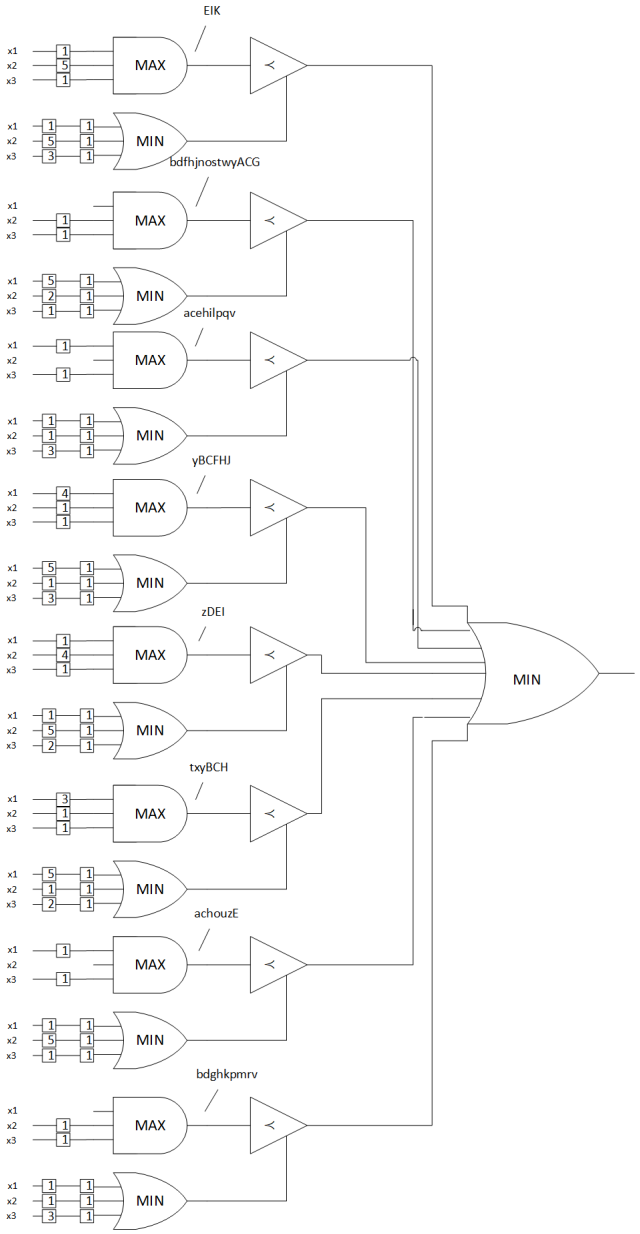

Figure 13: Optimized implementation of neuron model

9 CONCLUSIONS

So far, the novel methodology named *Space-Time algebra* was shown to be functionally complete for bounded tables, and to have the potential for creating more realistic artificial neural networks models in hardware. Motivated by these findings, this work took a step forward and characterized the behavior of delay terms in s-t algebra, and how they relate to function table rows. By exploration of delay term behavior, we have deduced 3 rules:

Table 2: Prime delay term chart for the neuron model

Prime Delay Terms	de1	di1	de2	di2	de3	di3
EIK	1	1	5	5	1	3
bdfhjnostwyACG	-1	5	1	2	1	1
acehilpqv	1	1	-1	1	1	3
yBCFHJ	4	5	1	1	1	3
zDEI	1	1	4	5	1	2
txyBCH	3	5	1	1	1	2
achouzE	1	1	-1	5	1	1
bdghkpmrv	-1	1	1	1	1	3
bdhntyC	-1	5	1	1	1	1
bgm	-1	-1	1	1	1	3
xBH	3	5	1	1	2	2

Table 3: Comparison of resources required by different neuron implementations.

	Naive Implementation	Optimized Implementation	Neuron Model
min	39	9	34
max	38	8	28
lt	38	8	12
delay	186	91	26
total non-delay	115	25	74
total	301	116	100

a. The number of function table rows that a delay term implements is given by $N = \prod_{i=1}^{n}(d_i^I - d_i^E + 1) - \prod_{i=1}^{n}(d_i^I - d_i^E)$, where N is the number of delay terms and n is the number of inputs.
b. A delay term (normalized) output value can only be in the following range, or infinity: $\max_i d_i^E \leq out \leq \max_i d_i^I$.
c. The rows covered by a delay term with 3 inputs can be found with algorithm 1. A similar algorithm can be written for any number of inputs.

From the exploration of delay term behavior, we came to a realization that in order to create an algorithm parallel to Quine-McCluskey in s-t algebra, we would first need to transform function tables into their delay representation, and then we could use the Q-M algorithm with slight modifications to create optimal two-level networks. This optimization algorithm is the first step made in s-t algebra towards practical utilization of this field, which so far has been mostly theoretical.

This optimization algorithm was then tested on a simple neuron model, showing results that could rival the current suggested neuron implementation in s-t algebra.

ACKNOWLEDGEMENT

We would like to thank Prof James E. Smith for introducing us to Space-Time Algebra and for his constant help and guidance in our research. He pointed us in the right direction and shared any new ideas he had on s-t algebra.

REFERENCES

[1] Madhavan A., T. Sherwood, and D. Strukov. 2015. Race logic: abusing hardware race conditions to perform useful computation. *IEEE Micro* 35 (2015), 48–57.
[2] Zvi Kohavi and Niraj K. Jha. 2009. *Switching and finite automata theory*. Cambridge University Press.
[3] Najafi et al. M. Hassan. 2017. Power and area efficient sorting networks using unary processing. (2017), 125–128.
[4] Wolfgang Maass. 1997. Networks of spiking neurons: the third generation of neural network models. *Neural networks* 10, 9 (1997), 1659–1671.
[5] Nasser Mehrtash, Dietmar Jung, Heik Heinrich Hellmich, Tim Schoenauer, Vi Thanh Lu, and Heinrich Klar. 2003. Synaptic plasticity in spiking neural networks (SP/sup 2/INN): a system approach. *IEEE transactions on neural networks* 14, 5 (2003), 980–992.
[6] James E Smith. 2017. Space-Time Computing with Temporal Neural Networks. *Synthesis Lectures on Computer Architecture* 12, 2 (2017).
[7] James. E. Smith. 2018. A Discrete Computation Model for Spiking (Temporal) Neural Networks. *unpublished article* (2018).
[8] J. E. Smith. 2018. Space-Time Functions: A Model for Neocortical Computation. In *Proceedings of the International Symposium on Architecture*.
[9] James. E. Smith. 2018. Unary Computers. *unpublished article* (2018).