Three-Dimensional Inverted Threshold Logic Circuits

Anas N. Al-Rabadi
Portland State University
alrabadi@yahoo.com

Abstract

New Three-dimensional paradigm of neural logic circuits called Three-Dimensional Inverted Threshold Logic (3DITL) circuits is introduced. The new 3D architecture inverts the inputs and weights in the standard Threshold Logic (TL) architecture: inputs become bases on internal interconnects, and weights become leaves of the network. The new 3DITL can be useful as a possible design choice for compacting a learning (trainable) circuit in 3D space.

1 Introduction

With future logic realization in technologies that are scaled down rapidly in size, the emphasis will be increasingly on the mutually linked issues of regularity, predictable timing, high testability, and self-repair [7,9,10,11].

For the current leading technologies with the active device count reaching the hundreds of millions, and more than 80% of circuit areas are occupied by local and global interconnects, the delay of interconnects is responsible for about 40-50% or more of the total delay associated with a circuit [12]. In future technologies, interconnects will take even higher percent of area and delay which creates interest in regular cellular structures, especially for deep sub-micron technologies. For example, Figure 1 illustrates trends for electrical signal delays for global interconnects with repeaters and without repeaters versus the local interconnects [12].

Realization of regular logic circuits in three-dimensional space can be very important for future technologies, as it shows that the best way is to place combinational logic functions in a three-dimensional space, where all local interconnections are of the same length and global interconnections are only inputs on parallel oblique planes (cf. dotted triangular oblique planes in Figure 3b) [2,7].

Three-dimensional circuits have been realized using Single Electron Transistor (SET) technology. For example, 3D multiple-valued networks have been implemented using SET devices [14]. Moreover, three-dimensional cubical lattice circuits have special importance since three-dimensional Crystal Lattices exist where inter-related atoms, that lay in a potential field, are spaced on the corners of three-dimensional cubes [4], and thus the potential physical implementation of the 3D circuits using 3D Crystal Lattices [4,5].

The main result of this paper is the application of logic expansions into regular trainable three-dimensional logic circuits called three-dimensional Inverted Threshold Logic (3DITL) circuits, and demonstrating how the new 3DITL is related to the classical Neural Network (NN) paradigm.

The remainder of this paper is organized as follows. Background on 3D circuits and Threshold Logic (TL) are presented in Section 2. New families of 3DITL circuits are presented in Section 3. Conclusions and future work are presented in Section 4.

2 Background

This Section presents related material on 3D lattice circuits and TL. Although the approach in this paper is shown for ternary Galois logic, similar approaches can be performed for higher radices of Galois logic as well.

2.1 Three-Dimensional Lattice Circuits

Three-Dimensional circuits presented in this section are implemented using ternary radix of Galois field which is shown in Table 1.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

(a) Galois field tables: (a) GF₃(+), and (b) GF₃(*).

Lattice circuits [2,10] generalize the ideas from the well-known regular structures: Fat trees, Generalized PLAs, Maitra cascades, Akers Arrays [1], and decision diagrams [3] into a more systematic framework which is closely related to the symmetry of functions and symmetric networks [9].
Three-dimensional circuits called 3D lattice circuits have been developed [2]. Because three-dimensional lattice circuits exist in a three-dimensional space, a geometrical reference of coordinate system is used in order to be systematic in the realizations of the corresponding logic circuits. The right-hand rule of the Cartesian coordinate system is used for this purpose. (Examples 1 and 2 will illustrate lattice realizations using such coordinate system.)

A literal is a single-variable function. One type of literals is the 1-Reduced Post literal (1-RPL) $i'x$ [2] defined as:

$$i'x = 1 \text{ iff } x = i \text{ else } i'x = 0$$  \hspace{1cm} (1)

For example $x'c$, $x'b$, $x'a$ are the zero, first, and second polarities of the 1-RPL, and $x$ can take any value in the set $\{0, 1, 2\}$. Figure 2 shows 1-RPL for variable $x$.

![Figure 2. 1-Reduced Post Literal (1-RPL).](image)

As will be demonstrated later, a ternary symmetric function is a function that is realized in a 3D lattice circuit without the need to repeat input variables, otherwise the function is non-symmetric [2]. Example 1 demonstrates a ternary symmetric function realized in a 3D lattice circuit, and Example 2 demonstrates a ternary non-symmetric function realized in a 3D lattice circuit.

**Example 1.** Using ternary Galois addition and multiplication (cf. Table 1), 1-RPL (cf. Figure 2), and by adopting the right-hand rule of the Cartesian coordinate system, Figure 3b shows three-dimensional logic circuit realization of the three-variable ternary-input ternary-output symmetric function $F = ab + ac + bc$ represented as a ternary map in Table 2.

<table>
<thead>
<tr>
<th>$ab$</th>
<th>$c$</th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>02</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>21</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>22</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 2. Ternary map for the function: $F = ab + ac + bc$.

One can observe that Figure 3 represents a fully regular lattice circuit in three-dimensions; it uses the same 3-to-1 selector (multiplexer) as internal nodes. Each dimension in Figure 3 corresponds to a value of the corresponding control variable: value zero of the control variable propagates along the x-axis, value one of the control variable propagates along the y-axis, and value two of the control variable propagates along the z-axis.

In three-dimensional space, each control variable spreads in a plane to control the corresponding nodes. These parallel planes are represented using the dotted triangles in Figure 3b. Each node in Figure 3b represents a three-input one-output multiplexer, whose output goes in three directions. The right-hand rule of the Cartesian coordinate system is used in order to be systematic in the realizations of the corresponding logic circuits. The right-hand rule of the Cartesian coordinate system is used for this purpose. (Examples 1 and 2 will illustrate lattice realizations using such coordinate system.)

In Figure 3, if one multiplies each leaf value, going counter clockwise (CCW), with all possible outside-to-inside paths (from the leaves to the root $F$) and add them over 1-RPL Galois algebra (using Table 1 and Equation (1)) then one will obtain the function $F = ab + ac + bc$ in the root.

![Figure 3. (a) 3D node, and (b) three-dimensional logic circuit realization for ternary symmetric function: $F = ab + ac + bc$.](image)
2.2 THRESHOLD LOGIC

Threshold Logic (TL) within Neural Network (NN) context [6,8,15] has been proven to be a powerful computing paradigm for the following properties: (1) **speed**: the computations in NN are relatively speedy due to the fact that a NN is a massively parallel and distributed processing structure; (2) **graceful degradation**: this property is due to the fact that NN memory is distributed and thus if any NN element is destroyed, the whole NN structure will still function appropriately; and (3) **Learning**: NN can learn by updating its weight values by the interaction with its environment as means to get outside knowledge to NN circuit.

NN is a parallel and distributed computing methodology that implements complex function expansions to realize discrete and continuous logic functions [6,8,15]. In such methodology, the weights on the interconnects may be thought of as representing the varying expansion coefficients, and the activation functions (AF) (or Transfer Function (TF)) in the nodes represent the basis functions for such function expansions. This can be written for a single hidden layer NN with M inputs and N hidden nodes as follows [6,8,15]:

\[
y = \sum_{i=1}^{N} a_i \phi_i \left( \sum_{j=1}^{M} w_{ij} x_j \right)
\]

where the Activation Function (AF) \( \phi(.) \) can be any appropriate nonlinear mapping such as sigmoid, etc [6,8,15].

Figure 5a illustrates, as an example, a simple single-node feedforward NN circuit implementation of Equation (2) with supervised Hebbian (Delta rule) learning (which emulates the chemical synaptic “strength” in a natural NN [6,8,15]). Hebbian learning for a NN can be written as follows:

\[
\Delta w_{ij} = \beta (\text{desired function} - \text{actual function}) x_j
\]

Using Equation (2), the output \( y \) in Figure 5b is a nested function that can be obtained as:

\[
y = w_{50} \cdot 1 + w_{53} \cdot \phi(w_{50} \cdot 1 + w_{31} \cdot x_1 + w_{32} \cdot x_2) + w_{54} \cdot \phi(w_{40} \cdot 1 + w_{41} \cdot x_1 + w_{42} \cdot x_2)
\]

In the NN context, an AF for supervised learning paradigm as in Equation (3) must satisfy the following two constraints [6,8,15]: (1) continuously differentiable function, and (2) non-decreasing function. For instance, a simple non-decreasing and continuously differentiable map that is commonly used in perceptron NN is a threshold function [6,8,15] (thereafter the name of threshold logic) defined as follows:

\[
y = \begin{cases} 
+1, & \text{if} \left( \sum_{k} w_k x_k + \theta \geq 0 \right) \\
-1, & \text{if} \left( \sum_{k} w_k x_k + \theta < 0 \right)
\end{cases}
\]

It can be noted that the 1-RPL function in Figure 2 is a kind of a threshold function: multiply two threshold (step) functions (each of two levels “0” and “1”) with even symmetry around a shared \( x = i \), and then one obtains Figure 2.

In a NN context, a paradigm includes (1) selection of NN nodes, (2) NN architecture, and (3) NN training (learning) algorithm. The learning occurs in a NN through changing the values of weights on the interconnects. The learning process in a NN circuit can be represented equivalently as: (1) a learning trajectory in the weight space, and (2) error function (criterion function) trajectory in the augmented weight space, i.e., weight space that has an extra dimension for the error function. Figure 6 illustrates these two equivalent representations of a NN learning process.
Figure 5. (a) A nonlinear model of a neuron N, where \( \{1, x_1, \ldots, x_{N-1}\} \) is set of inputs, \( w_i \) are synaptic weights, \( \Sigma \) is a summing junction, \( \varphi(.) \) is an Activation Function (AF) (or Transfer Function (TF)), and \( y \) is the output, and (b) a single hidden layer feedforward NN, where \( w_{ji} \) is the weight from node \( i \) to node \( j \), \( w_{k0} \) is a bias input \( b_k \), the AF in the hidden nodes is a threshold function, and the AF in the output node is a linear function.

Figure 6. Representations for NN learning: (a) weight trajectory in weight space, and (b) error function (criterion function) trajectory in the augmented weight space.

Figure 7. A possible three-dimensional classical TL circuit: (a) general architecture, and (b) one way of representing 2DTL circuit in Figure 5b in 3D space.

A discrete logic function \( Y \) can be expanded using a linear spectral transform \([M]\) as follows \([2,13]\):

\[
Y = \hat{x}^T \tilde{y} = \left[i \tilde{x}^T [M]^{-1} [M] \tilde{y}\right] = \bar{B}^T \bar{S}
\]

(6)

where \(i \tilde{x}^T\) is a row vector of the 1-RPL, \(\tilde{y}\) is the truth vector of the logic function \(Y\), \(\bar{B}^T\) is the row vector of the set of basis functions, and \(\bar{S}\) is the vector of spectral coefficients. Learning is achieved by changing the value of the spectral coefficients \(\bar{S}\) with fixed basis functions \(\bar{B}\). In a NN/TL context, \(\bar{S}\) can be thought of as weights and \(\bar{B}\) as the internal nodes. This can be represented using Equation (6) as follows:

\[
Y = \left[i \tilde{x}^T [M]^{-1} [N] \tilde{w}_i\right] = \bar{B}^T \bar{S}
\]

(7)
where $[N]$ is a varying linear transformation that will result in the varying spectral coefficients $S^* = [M]([N] \overrightarrow{w})$. The final trained circuit will have $[N] \overrightarrow{w}_j = \overrightarrow{y}$ and therefore:

$$\text{Error} = \left| S^* - \overrightarrow{y} \right| = \left| [M]([N] \overrightarrow{w}_j - [M] \overrightarrow{y}) \right| = 0$$

As $S^*$ can be thought of as training weights and $\overrightarrow{y}$ as the internal nodes, Figure 8 shows a 3DTL paradigm that uses: (1) architecture from the 3D lattice circuits in Section 2.1, and (2) learning governed by Equation (7) such that the Error = |Actual Output $Y_a$ – Desired Output $Y_d$|, and one would want Error $\rightarrow$ 0. We call the circuit in Figure 8 by three-dimensional Inverted Threshold Logic (3DITL) circuit since the positions of weights and internal nodes are inverted (or turned inside out) as compared to the classical TL/NN architecture (cf. Figure 7).

As an example, a 3DITL circuit for two variables \{x$_1$,x$_2$\}, where (1) weights occupy the leaves and (2) inputs are used in the internal nodes, is shown in Figure 9.

As stated previously in Section 2.1 (cf. Figures 3 and 4), a 3DITL circuit (such as in Figures 8 and 9) does not need variable repetition for implementing a ternary symmetric function, but needs variable repetition for implementing a ternary non-symmetric function.

A learning algorithm (strategy; method) has to be provided for the 3DITL circuit in order for the actual output $Y_a$ to converge into the desired output $Y_d$. A simple exhaustive search algorithm called 3DITL to obtain the correct weight values for functions with small number of variables (e.g., ≤ 4) can be implemented as follows:

<table>
<thead>
<tr>
<th>Algorithm 3DITL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Let Error =</td>
</tr>
<tr>
<td>2 For one level 3DITL, Search for all possible combinations of $w_{ji}$</td>
</tr>
<tr>
<td>3 Symmetric case: If Error = 0, Then goto 7</td>
</tr>
<tr>
<td>4 Else goto 5</td>
</tr>
<tr>
<td>5 Non-Symmetric case: Expand 3DITL by one level (i.e., repeat one variable)</td>
</tr>
<tr>
<td>6 Goto 2</td>
</tr>
<tr>
<td>7 End</td>
</tr>
</tbody>
</table>

In order to guarantee a 3DITL circuit learning convergence (i.e., termination when Error = |$Y_a$ – $Y_d$| = 0), it is important to prove that the repetition of variables in 3DITL circuit will have an end in the process of symmetrization of non-symmetric functions. An intuitive proof is as follows [2]: for totally symmetric functions the number of variables is equal to the number of levels of the lattice circuit, as there is no need to repeat variables, and as it is known that by the repetition of variables every non-symmetric function is symmetrized [1], then this must result in a definite number of levels in the corresponding 3DITL circuit and as a consequence in a certain number of total variables, repeated and non-repeated, that will result in the termination of the process of symmetrization. Since the weight values in the case of 3DITL circuit presented in this Section are always integers $\in \mathbb{Z}$ of ternary value (i.e., \{0,1,2\}), then the algorithm 3DITL is guaranteed to ultimately converge (terminate).

**Example 3.** For the following ternary function $Y = a \cdot b = 1a^1b + 2^2a^1b + 2^2a^0b + 2^3a^0b$ which can be represented by the following ternary map:

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
The ternary function $Y_a$, realized in Figure 10, can be represented as follows:

$$Y_a = \sum_{i=0}^{2} \sum_{j=0}^{2} w_{ij} a^i b^j \quad (8)$$

One can use, as data structure representation of Equation (8), a decision tree (DT) and the corresponding decision diagram (DD) data structures [3], as representations that can be used for the exhaustive or heuristic search in Figure 11.

In a DT and DD basis functions (literals in this case) are located on the edges, and leaves contain the function truth values. The rules for obtaining a DD from a DT are: (1) join isomorphic nodes in the DT, and (2) remove redundant nodes from the DT [3]. Figure 12 shows the DT and DD for the 3DITL circuit in Figure 10, where (by referring to the notation in Equation (8)): $w_1 = w_{00}$, $w_2 = w_{01}$, $w_3 = w_{02}$, $w_4 = w_{10}$, $w_5 = w_{11}$, $w_6 = w_{12}$, $w_7 = w_{20}$, $w_8 = w_{21}$, and $w_9 = w_{22}$.

Figures 11 and 12 illustrate one major disadvantage of performing exact search: one more level in the 3DITL circuit (cf. Figure 8) and there will be 10 three-valued leaves, and thus the total number of all possible weight updates will be $3^{10} = 59,049$ which is a big number of combinations to search within! The problem gets even worse for larger number of inputs and/or large number of input repetitions. Consequently, one can immediately note the need for a practical either (1) formal training strategy/rule or (2) heuristic search for the new 3DITL circuits instead of the impractical exhaustive search method (to be addressed further in a future paper).
4 CONCLUSIONS AND FUTURE WORK

Novel three-dimensional architecture of Threshold Logic within the context of neural circuits called Three-Dimensional Inverted Threshold Logic (3DITL) circuits is introduced. The new 3D architecture inverts the inputs and weights in the standard neural network (NN) architecture: inputs become bases on the internal interconnects, and weights become leaves of the network.

Future work will involve the following items for the new 3DITL circuits:

1. Since in classical NN paradigm, converging to the correct output implies separability in the decision space [6,8,15], future work will investigate the type of separability that the new 3DITL circuits achieve (such as in Figure 10), i.e., linearly separable versus nonlinearly separable 3DITL circuits.

2. Finding practical (1) formal and/or (2) heuristic training rules (learning algorithms) that guarantee to converge (terminate) within finite number of steps (e.g., Evolutionary Computation (EC), Simulated Annealing (SA), etc).

3. The extension of the 3DITL family introduced in Section 3 to other types of basis functions in the internal nodes.

4. The investigation of the application of the 3DITL circuits using 3D Crystal Lattices from [4] (cf. Figure 13) by using a physically trainable property in such crystals.

![Figure 13](image)

**Figure 13.** Implementing 3DITL circuits in 3-D Crystal Lattices: (a) 3D Simple Cubic (SC) Crystal Lattices, (b) Body Centered Cubic (BCC) unit cell of a Crystal Lattice, and (c) Face Centered Cubic (FCC) unit cell of a Crystal Lattice.

5 REFERENCES


