

Architecture for Managing a 2^7 -Node Network Using a Threshold-Based Combinational Circuit Designed Specifically to Bypass Routing Conflicts and Prevent Systemic Deadlocks

Anup Kumar Biswas

Department of Computer Science and Engineering, Kalyani Govt. Engineering College, Kalyani, Nadia-741235, West Bengal, India
Email: akbese456[at]gmail.com

Abstract: Physically constructing geometric figures is bound to three spatial dimensions; structures existing in four or more dimensions can typically only be conceptualized or simulated virtually. This work presents the spatial realization of a 7-dimensional (7D) network topology mapped into a 3-dimensional (3D) physical space. The network consists of $2^7 = 128$ nodes, where each node is uniquely identified by a 7-bit binary address. To ensure a collision-free and deadlock-free environment, the 128 nodes are arranged in a deterministic geometric pattern. Routing paths within this network are predefined using a deterministic algorithm. The exact packet travel paths between any given source and destination node are computed using sequential Exclusive-OR (XOR) operations applied to their binary addresses. Rather than relying on classical CMOS-based logic, this routing architecture utilizes a novel XOR gate model built upon Multiple-Input Threshold Logic (MITL). While this MITL framework is versatile enough to synthesize fundamental logic primitives (AND, OR, NOT) and complex sequential circuits, this specific implementation leverages a specialized arrangement of 56 MITL-based XOR gates. This circuit configuration successfully computes deadlock-free routing paths dynamically, ensuring that data packets transit through a bounded path consisting of at most 8 nodes. Based on the experimental input-output values, the network's truth tables demonstrate robust topological constraints. When applying the routing algorithm, the maximum path length is strictly confined to 7. Specifically, evaluating any source-destination pair generates a path sequence of exactly 8 node positions (inclusive of the source and destination). Because certain nodes within this sequence may share identical values, the number of unique nodes involved in any given path is at most 8, ensuring the actual path length remains less than 8. Furthermore, the implemented network design is verified to be entirely collision-free. This is structurally validated by the truth tables, which exhibit no duplicate node identifiers within any single column.

Keywords: conflict, collision free, deadlock, combinational, Threshold, Routing-path

1. Introduction

This text explains a routing method for a 7-hypercube network. Using a specific algorithm, it maps multiple paths from a starting node to a destination node, detailing each intermediate link. The distance between any two connected nodes is calculated using the Hamming weight, which counts the number of 1s resulting from an Exclusive-OR (XOR) operation between their binary representations. For example, the distance between Node X (1101001) and Node Y (0001110) is 5, because $1101001 \oplus 0001110 = 1100111$ (which contains five 1s).

The routing algorithm can be built as a combinational circuit using an emerging technology known as multi-input threshold logic. This technology is first used to design basic OR, AND, and XOR gates. By leveraging the XOR gate, a streamlined yet highly capable circuit is built. This circuit automatically determines the routing path, calculates the overall distance, and predicts the sequential node numbers along the path. Compared to traditional CMOS circuits, this threshold-logic design processes data much faster while consuming significantly less energy.

2. An n-dimensional hypercube for 2^n nodes

An n-dimensional hypercube consists of 2^n nodes (vertices). Each node is identified by an n-bit binary string, represented

as $p = p_{n-1}p_{n-2}p_{n-3} \dots p_1p_0$. The routing path starts at a source node ($s = s_{n-1}s_{n-2}s_{n-3} \dots s_1s_0$) and ends at a destination node ($d = d_{n-1}d_{n-2}d_{n-3} \dots d_1d_0$). To evaluate the routing path, we must determine the total path length as well as the exact sequence of visited nodes, identified by their node numbers in either binary or decimal format.

Generally, the n dimensions are indexed as $i = 0, 1, 2, 3, \dots, n-1$, where the i^{th} dimension corresponds to the i -th bit position in a node's address $= q_{n-1}q_{n-2}q_{n-3} \dots q_1q_0$. The routing path from the source node (s) to the destination node (d) is determined by the following algorithm:

Algorithm 1: Dimension-by-Dimension Hypercube Routing

- The direction bit k_i is the XOR ($\oplus$) combination of bits from source s and destination d. $k_i = s_{i-1} \oplus d_{i-1}$. (where $i = 1, 2, 3, \dots, n$). Initialization: Set the value $q = s$.
- If $k_i = 1$: Update the path from node q to node $(q \oplus 2^{i-1})$.
if $k_i = 0$: Skip to the next step.
- Advance to the next dimension is $i \leftarrow i + 1$. If $i \leq n$, loop back to step (b); else, terminate.

In this work, we consider a network of dimension $n = 7$, resulting in a total of $2^7 = 128$ nodes. These nodes are

represented by 7-bit binary strings and their corresponding decimal values, ranging from 0000000 [0] to 1111111 [127]. To illustrate the routing process, let the source node be $S = 0000100$ and the destination node be $D = 0011100$. The directional routing vector k is determined by the bitwise XOR operation between the source and destination:
 $k = k_7k_6k_5k_4k_3k_2k_1 = (S \oplus D) = 0011000$.

Since the bits for dimensions $i = 1, 2, 3, 6$ and 7 are all zero $k_1=0 \oplus 0=0, k_2=0 \oplus 0=0, k_3=1 \oplus 1=0, k_6=0 \oplus 0=0$ and $k_7=0 \oplus 0=0$, these dimensions are skipped during routing. The path only modifies the dimensions where $k_i = 1$.

The routing begins at the source node $S = 0000100$. Because $k_4=0 \oplus 1=1$, the next hop is calculated by flipping the fourth bit:
 $S \oplus 0001000 = 0000100 \oplus 0001000 = 0001100$

From this intermediate node, $q = 0001100$, the routing proceeds to the next required dimension. Because $k_5=0 \oplus 1=1$, flipping the fifth bit yields the destination node: $q \oplus 0010000 = 0001100 \oplus 0010000 = 0011100$. Consequently, the ordered sequential routing path is: $0000100 \rightarrow 0001100 \rightarrow 0011100$

Example 1: Standard Route with Minimal Changes

Given a source node $S = 0000100$ and destination node $D = 0011100$:

- 1) Calculate the Direction Vector (k):
 $k = S \oplus D = 0000100 \oplus 0011100 = 0011000$
 Here, $k_4 = 1$ and $k_5 = 1$. The dimensions where $k_i = 0$ ($i = 1, 2, 3, 6, 7$) are skipped because the bits match.
- 2) Route Hop-by-Hop (Right to Left):
 - Hop 1 (Resolve k_4): Flip the 4th bit from the right.
 $0000100 \oplus 0001000 = 0001100$
 - Hop 2 (Resolve k_5): Flip the 5th bit from the right.
 $0001100 \oplus 0010000 = 0011100$
 - Final Routing Path: $0000100 \rightarrow 0001100 \rightarrow 0011100$

In the same way, if we select another pair of source and destination nodes, $S = 0100110$ and $D = 0001001$, we can find the Multi-Hop Route across Diverse Dimensions:

Given a source node $S = 0100110$ [38] and destination node $D = 0001001$ [9]:

- 1) Calculate the Direction Vector k :
 $k = k_7k_6k_5k_4k_3k_2k_1 = S \oplus D = 0100110 \oplus 0001001 = 0101111$
 The mismatch bits are evaluated sequentially from right to left (k_1, k_2, k_3, k_4, k_6). Dimensions k_5 and k_7 are equal to 0 and are skipped.
- 2) Route Hop-by-Hop (Right to Left):
 - Hop 1 (Resolve k_1): $0100110 \oplus 0000001 = 0100111$ [39]
 - Hop 2 (Resolve k_2): $0100111 \oplus 0000010 = 0100101$ [37]
 - Hop 3 (Resolve k_3): $0100101 \oplus 0000004 = 0100001$ [33]
 - Hop 4 (Resolve k_4): $0100001 \oplus 0001000 = 0101001$ [41]
 - Hop 5 (Resolve k_6): $0101001 \oplus 0100000 = 0001001$ [9]

Final Routing Path: 0100110 [38] $\rightarrow$ 0100111 [39] $\rightarrow$ 0100101 [37] $\rightarrow$ 0100001 [33] $\rightarrow$ 0101001 [41] $\rightarrow$ 0001001 [9]

3) Path node calculation and Network representation

The total number of bidirectional source-destination pairs across the 128 nodes in Fig. 1 is calculated as $(127 + 126 + \dots + 1) \times 2 = 127 \times 128 = 16,256$. A multiplication factor of 2 is included to account for bidirectional communication.

Key Observations & Routing Logic

- Path Evaluation: A sample subset of these source-destination pairs was tested against the circuit in Fig. 1 to determine the total number of nodes involved in each routing path. These values are recorded in the final column of Table 1.
- Maximum Path Length: According to the routing algorithm, the maximum number of nodes connected in any single routing path is 8.
- Node Parity & Traversal: The selected source and destination nodes can be both even, both odd, or a mixed pair. If both points are exclusively even or exclusively odd, the algorithm restricts the routing path to traverse only even or only odd nodes, respectively.

According to the proposed algorithm, routing paths strictly depend on the parity (even or odd nature) of the source and destination nodes. If both the source and destination share the same parity, every intermediate node along the path must match that parity. For example, as shown in serial numbers 1 and 8 of Table-1, identical parity at both endpoints ensures a uniform path: an even-to-even route consists entirely of even nodes (Sl. No. 1), while an odd-to-odd route consists entirely of odd nodes (Sl. No. 8). Conversely, if the source and destination have opposite parities, the parity of the route shifts immediately after the source. Specifically, the second node and all subsequent nodes up to the destination must adopt the destination's parity. This rule is evident in Sl. No. 16 (source 46, destination 35), where the path shifts to all-odd nodes starting from the second point, and in Sl. No. 12, where the path shifts to all-even nodes.

A defining characteristic of this algorithm is its collision-free design. Even if the routing matrix is expanded up to Sl. No. 15,120,256, no column in the entire table will ever contain a duplicate node number. Because every node assignment within a column remains completely unique, routing conflicts are inherently prevented, resulting in a fundamentally deadlock-free network system [19].

Based on Table 1, the maximum routing path length is 7. A path length represents the total number of segments between consecutive nodes. In the 33rd row, the path length is 7 because every node in the path is distinct. In other rows, some node identification numbers repeat, causing the path lengths to vary.

Path lengths can be calculated in two ways:

- Node Count: Total number of nodes (involved in the path) - 1)

- Vector Weight: The total number of 1s in the binary vector = $\{k_7k_6k_5k_4k_3k_2k_1\}$

For example, in row 6, the source is 76 and the destination is 9.

- Method 1: The path visits 4 nodes (76, 77, 73, and 9). The path length is $(4 - 1 = 3)$.
- Method 2: The vector is $k = 1, 0, 0, 0, 1, 0, 1$. There are three 1s, so the path length is 3.

Table 1: Routing path(s)

Sl. No.	Source(s) node number	Destination node number	Original Source 1	Source 2	Source 3	Source 4	Source 5	Source 6	Source 7	goal	Routing Path Length = No. of 1s in $\{k_i\} = \{k_7k_6k_5k_4k_3k_2k_1\}$
1	10	92	0001010 = 10	0001010 = 10	0001000 = 8	0001100 = 12	0001100 = 12	0011100 = 28	0011100 = 28	1011100 = 92	$\{k_i\} = 1010110$ 4
2	14	75	0001110 = 14	0001111 = 15	0001111 = 15	0001011 = 11	0001011 = 11	0001011 = 11	0001011 = 11	1001011 = 75	$\{k_i\} = 1000101$ 3
3	32	127	0100000 = 32	0100001 = 33	0100011 = 35	0100111 = 39	0101111 = 47	0111111 = 63	0111111 = 63	1111111 = 127	$\{k_i\} = 1011111$ 6
4	32	126	0100000 = 32	0100000 = 32	0100010 = 34	0100110 = 38	0101110 = 46	0111110 = 62	0111110 = 62	1111110 = 126	$\{k_i\} = 1011110$ 5
5	126	32	1111110 = 126	1111110 = 126	1111100 = 124	1111000 = 120	1110000 = 112	1100000 = 96	1100000 = 96	0100000 = 32	$\{k_i\} = 1011110$ 5
6	76	9	1001100 = 76	1001101 = 77	1001101 = 77	1001001 = 73	1001001 = 73	1001001 = 73	1001001 = 73	0001001 = 9	$\{k_i\} = 1000101$ 3
7	52	74	0110100 = 52	0110100 = 52	0110110 = 54	0110010 = 50	0111010 = 58	0101010 = 42	0001010 = 10	1001010 = 74	$\{k_i\} = 1111110$ 6
8	21	41	0010101 = 21	0010101 = 21	0010101 = 21	0010001 = 17	0011001 = 25	0001001 = 9	0101001 = 41	0101001 = 41	$\{k_i\} = 0111100$ 4
9	0	64	0000000 = 0	0000000 = 0	0000000 = 0	0000000 = 0	0000000 = 0	0000000 = 0	0000000 = 0	1000000 = 64	$\{k_i\} = 1000000$ 1
10	54	118	0110110 = 54	0110110 = 54	0110110 = 54	0110110 = 54	0110110 = 54	0110110 = 54	0110110 = 54	1110110 = 118	$\{k_i\} = 1000000$ 1
11	57	121	0111001 = 57	0111001 = 57	0111001 = 57	0111001 = 57	0111001 = 57	0111001 = 57	0111001 = 57	1111001 = 121	$\{k_i\} = 1000000$ 1
12	3	80	0000011 = 3	0000010 = 2	0000000 = 0	0000000 = 0	0000000 = 0	0010000 = 16	0010000 = 16	1010000 = 80	$\{k_i\} = 1010011$ 4
13	22	100	0010110 = 22	0010110 = 22	0010100 = 20	0010100 = 20	0010100 = 20	0000100 = 4	0100100 = 36	1100100 = 100	$\{k_i\} = 1110010$ 4
14	80	102	1010000 = 80	1010000 = 80	1010010 = 82	1010110 = 86	1010110 = 86	1000110 = 70	1100110 = 102	1100110 = 102	$\{k_i\} = 0110110$ 4
15	92	29	1011100 = 92	1011101 = 93	1011101 = 93	1011101 = 93	1011101 = 93	1011101 = 93	1011101 = 93	0011101 = 29	$\{k_i\} = 1000001$ 2
16	46	35	0101110 = 46	0101111 = 47	0101111 = 47	0101011 = 43	0100011 = 35	0100011 = 35	0100011 = 35	0100011 = 35	$\{k_i\} = 0001101$ 3
17	9	109	0001001 = 9	0001001 = 9	0001001 = 9	0001101 = 13	0001101 = 13	0001101 = 13	0101101 = 45	1101101 = 109	$\{k_i\} = 1100100$ 3
18	91	19	1011011 = 91	1011011 = 91	1011011 = 91	1011011 = 91	1010011 = 83	1010011 = 83	1010011 = 83	0010011 = 19	$\{k_i\} = 1001000$ 2
19	125	25	1111101 = 125	1111101 = 125	1111101 = 125	1111001 = 121	1111001 = 121	1111001 = 121	1011001 = 89	0011001 = 25	$\{k_i\} = 1100100$ 3
20	7	57	0000111 = 7	0000111 = 7	0000101 = 5	0000001 = 1	0001001 = 9	0011001 = 25	0111001 = 57	0111001 = 57	$\{k_i\} = 0111110$ 5
21	1	65	0000001 = 1	0000001 = 1	0000001 = 1	0000001 = 1	0000001 = 1	0000001 = 1	0000001 = 1	1000001 = 65	$\{k_i\} = 1000000$ 1
22	65	1	1000001 = 65	1000001 = 65	1000001 = 65	1000001 = 65	1000001 = 65	1000001 = 65	1000001 = 65	0000001 = 1	$\{k_i\} = 1000000$ 1
23	88	66	1011000 = 88	1011000 = 88	1011010 = 90	1011010 = 90	1010010 = 82	1000010 = 66	1000010 = 66	1000010 = 66	$\{k_i\} = 0011010$ 3
24	72	12	1001000 = 72	1001000 = 72	1001000 = 72	1001100 = 76	1001100 = 76	1001100 = 76	1001100 = 76	0001100 = 12	$\{k_i\} = 1000100$ 2
25	100	124	1100100 = 100	1100100 = 100	1100100 = 100	1100100 = 100	1101100 = 108	1111100 = 124	1111100 = 124	1111100 = 124	$\{k_i\} = 0011000$ 2
26	89	119	1011001 = 89	1011001 = 89	1011011 = 91	1011111 = 95	1010111 = 87	1010111 = 87	1110111 = 119	1110111 = 119	$\{k_i\} = 0101110$ 4
27	31	111	0011111 = 31	0011111 = 31	0011111 = 31	0011111 = 31	0011111 = 31	0001111 = 31	0101111 = 111	1101111 = 111	$\{k_i\} = 1110000$ 3

			31	31	31	31	31	15	47	111	3
28	99	55	1100011 = 99	1100011 = 99	1100011 = 99	1100111 = 103	1100111 = 103	1110111 = 119	1110111 = 119	0110111 = 55	{ki}=1010100 3
29	11	111	0001011 = 11	0001011 = 11	0001011 = 11	0001111 = 15	0001111 = 15	0001111 = 15	0101111 = 47	1101111 = 111	{ki}=1100100 3
30	69	89	1000101 = 69	1000101 = 69	1000101 = 69	1000001 = 65	1001001 = 73	1011001 = 89	1011001 = 89	1011001 = 89	{ki}=0011100 3
31	2	122	0000010 = 2	0000010 = 2	0000010 = 2	0000010 = 2	0001010 = 10	0011010 = 26	0111010 = 58	1111010 = 122	{ki}=1111000 4
32	127	27	1111111 = 127	1111111 = 127	1111111 = 127	1111011 = 123	1111011 = 123	1111011 = 123	1011011 = 91	0011011 = 27	{ki}=1100100 3
33	95	32	1011111 = 95	1011110 = 94	1011100 = 92	1011000 = 88	1010000 = 80	1000000 = 64	1100000 = 96	0100000 = 32	{ki}=1111111 7

Following Algorithm 1, the network of 128 nodes ($= 2^7$) is organized into a $4 \times 8 \times 4$ matrix, as illustrated in Figure 1 [1-2]. Table 2 outlines the traveling paths for specific source

and destination pairs. As a representative example, the routing path from source node 95 to destination node 32 is traced and highlighted with a thick green line in Fig.1.

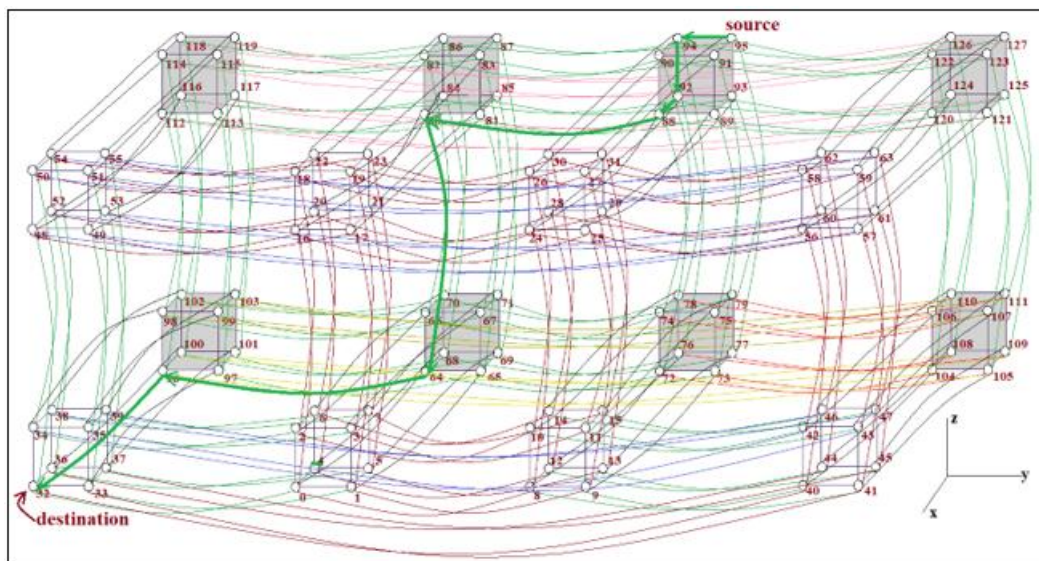


Figure 1: Illustrates a 128-node network connection generated using Algorithm 1, highlighting the routing path between node 95 and node 32 in a thick green color.

Table 2

Sl. No.	Source Node number	Destination Node number	Travelling or Routing path	Path length
1	109	54	109 → 108 → 110 → 102 → 118 → 54	5
2	95	32	95 → 94 → 92 → 88 → 80 → 64 → 96 → 32	7
3	51	8	51 → 50 → 48 → 56 → 40 → 8	5
4	48	50	48 → 48 → 50 (48 → 50)	1
5	18	68	18 → 16 → 20 → 4 → 68	4
6	39	63	39 → 47 → 63	3
7	37	124	37 → 36 → 44 → 60 → 124	4
8	32	95	32 → 33 → 35 → 39 → 47 → 63 → 31 → 95	7
9	48	105	48 → 49 → 57 → 41 → 105	4
10	83	33	83 → 81 → 65 → 97 → 33	4
11	1	122	1 → 0 → 2 → 10 → 26 → 58 → 122	6
12	83	14	83 → 82 → 86 → 94 → 78 → 14	5

Generic Multiple-Input Threshold Logic Gate (TLG)

A generic, multi-input single-output threshold logic gate (LTG) [3–8, 14–16] is illustrated in Figure 2. The core architecture consists of a single tunnel junction—defined by its capacitance C_j and resistance R_j —connected between node p and node q.

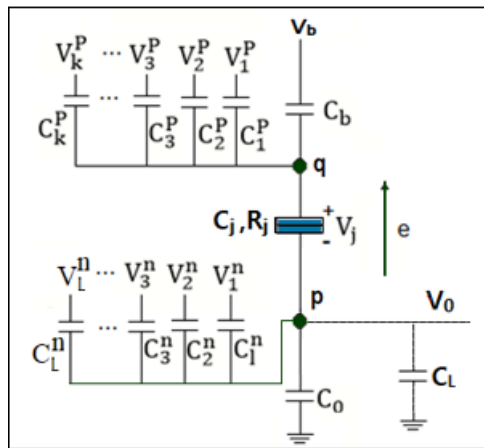


Figure 2: multi-input one-output TLG

The circuit is driven by two distinct sets of multi-input combination signals:

- Upper-side inputs (V_k^p): Connected to node q through their respective coupling capacitances C_k^p .
- Lower-side inputs (V_l^n): Connected to node p through their respective coupling capacitances C_k^p .

To assist the electron tunneling process, a constant bias voltage V_b is applied to node q via a dedicated biasing capacitor C_b . Capacitor C_0 is connected between node P and ground. Together, the tunnel junction capacitance C_j and the grounding capacitance C_0 serve as the critical decision-making elements that govern the switching behavior of the circuit.

The functional operation of this multi-input single-output TLG is mathematically evaluated using the signum function. The behavior of this signum function, denoted as $\text{sgn}[h(x)]$, is defined by equations (1) and (2) below [9–14]:

Equations and System Model

The behavior of a multi-input, single-output Linear Threshold Gate (LTG) is governed by the following system of equations:

$$g(x) = \text{sgn}\{h(x)\} = \begin{cases} 0, & \text{if } h(x) < 0 \\ 1, & \text{if } h(x) \geq 0 \end{cases} \quad (1)$$

$$h(x) = \sum_{k=1}^n (w_k \times x_k) - \theta \quad (2)$$

Where x_k represents the n distinct inputs $k=1, 2, 3 \dots n$, and w_k denotes their respective integer weights.

Functional Description:

In practice, the multi-input single-output LTG operates essentially as a comparator. The gate evaluates two distinct quantities: the pre-defined threshold value θ and the weighted sum of the inputs $\sum_{k=1}^n (w_k \times x_k)$.

When the weighted sum equals or exceeds the threshold value θ , the circuit triggers a high logical state, yielding an output of "1" at the output terminal V_0 . Conversely, if the weighted sum falls below the threshold, the gate remains inactive, resulting in a low logical output of "0".

4) 2-input AND gate having one complement input

To implement the circuit shown in Fig. 1, a two-input XOR gate for variables A and B must be constructed. The gate is defined by the logical function $F(A, B) = \text{XOR}(A, B) = \bar{A}B + A\bar{B}$, which performs an OR operation on the two individual AND terms, $\bar{A}B$ and $A\bar{B}$ [2-5]. To realize this, we first analyze the term $\text{AND}(A, \bar{B}) = A\bar{B}$ using threshold logic. The threshold logic equation for a two-input AND gate with one complemented input is expressed as:

$$\text{AND}(A, \bar{B}) = \text{sgn}\{w_A A + w_B \bar{B} - \theta\} \dots (3)$$

To determine the specific coupling weights $w_A A$ and $w_B \bar{B}$, we evaluate the condition where $A = 1$ and $\bar{B} = 1$. Under this assumption, equation (3) simplifies to:

$$\text{AND}(A, \bar{B}) = \text{sgn}\{w_A + w_B - \theta\} \dots (4)$$

To establish the final threshold logic gate parameters, Truth Table 3 is constructed for the variables A and $\bar{B}$. The combined sum of the weights, $w_A + w_B$ is then systematically compared against the threshold value θ across all logic states to ensure correct gate operation [1-5,7,8].

Table 3: Truth table of $F(A, \bar{B}) = A\bar{B}$

A	$\bar{B}$	$F(A, \bar{B}) = A\bar{B}$	θ	inequality
0	0	0	$0 < \theta$	(1)
0	1	0	$w_B < \theta$	(2)
1	0	0	$w_A < \theta$	(3)
1	1	1	$w_B + w_A \geq \theta$	(4)

For a positive-logic 2-input AND gate, let the weights for A and $\bar{B}$, i.e. w_A and w_B be 1. Setting the threshold to $\theta = 2$ satisfies all four inequalities in Table-3. This defines the linear threshold logic equation:

$$\text{AND}(A, \bar{B}) = \text{sgn}\{A + \bar{B} - 2\} \dots (5)$$

Algebraically, a logic variable and its complement always sum to 1 $B + \bar{B} = 1$. Substituting $\bar{B} = -B + 1$ into Equation (5) yields the final simplified expression:

$$F(A, B) = \text{AND}(A, \bar{B}) = \text{sgn}\{A - B - 1\} \dots (6)$$

Following Equation (6), the input capacitances for the positive and negative logic inputs C_1^p and C_1^n , respectively are incorporated into the generic multiple-input threshold logic gate shown in Fig. 1. For this specific $\text{AND}(A, \bar{B})$ gate configuration, the measured and assigned parameter values are defined as follows: logic input "0" = 0V, logic "1" = 16mV, $C = 1\text{aF}$, $C_1^p = C_1^n = \frac{1}{2}C = 0.5\text{aF}$, $C_b = 9.5\text{aF}$, $C_j = 0.25\text{aF}$, $C_l = 9\text{aF}$, $C_0 = 9.5\text{aF}$, $R_j = 10^5\Omega$, $V_b = V_s = 16\text{mV}$. The resulting two-input threshold logic gate architecture is illustrated in Fig. 3(b), while its corresponding simulation setup and simulated output results are presented in Fig. 3(c) and Fig. 3(d), respectively.

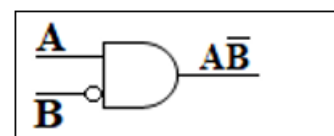


Figure 3: (a) symbol of $A\bar{B}$

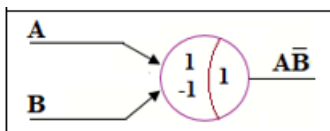


Figure 3: (b) TLG based node of AB

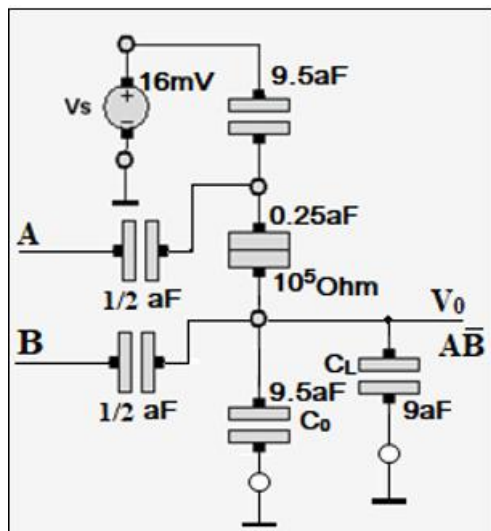


Figure 4 (c): Simulation set of AB

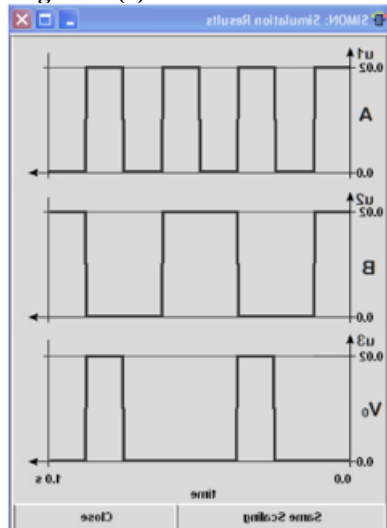


Figure 4 (d): Simulated result

5) Threshold logic equation for OR gate

Expression of an OR gate of two variables A and B is, where "+" indicates OR operation. If we wish to make the threshold logic gate of an OR gate, first the truth table of an OR gate is to be drawn. In Table-4, truth table of OR gate drawn and compare the weights of variables of two variables A and B respectively with the threshold or intrinsic value.

The Boolean expression for a two-variable OR gate is $F(A, B) = A + B$, where the "+" symbol denotes the logical OR operation. To implement this function as a threshold logic gate, we first construct the OR gate truth table (Table 4). By analyzing the inputs and outputs, we can then determine and compare the variable weights w_A and w_B against the threshold or intrinsic value θ .

Table 4

A	B	F (A, B)	θ
0	0	0	$0 < \theta$

0	1	1	$w_B \geq \theta$
1	0	1	$w_A \geq \theta$
1	1	1	$w_B + w_A \geq \theta$

Under positive logic assumptions, assigning weights $w_B=1$, and $w_A=1$ yields the characteristic inequalities derived from the fourth column of Table 4:

$$0 < \theta \dots\dots\dots (7)$$

$$w_B > \theta \dots\dots\dots (8)$$

$$w_A > \theta \dots\dots\dots (9)$$

$$w_B + w_A > \theta \dots\dots\dots (10)$$

Since condition (7) establishes that the threshold θ must be positive, setting ($\theta = 0.5$) successfully satisfies inequalities (8), (9), and (10). Consequently, the threshold logic equation for the OR gate is expressed in equation (11), and its corresponding logic gate configuration is illustrated in Fig. 4(a):

$$OR(A, B) = \text{sgn}\{A + B - 0.5\} \dots\dots\dots (11)$$

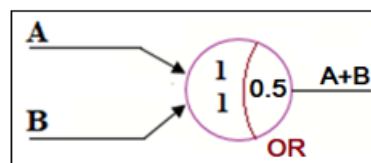


Figure 4 (a): Threshold logic OR gate

The OR gate was simulated using these specific circuit parameters:

Gate Capacitances: $C_1^p = C_2^p = 0.5 \text{ aF}$

Coupling & Load Capacitances: $C_b = 9.1 \text{ aF}, C_L = 9 \text{ aF}, C_0 = 10 \text{ aF}$

Junction Resistance $R_j = 10^5 \Omega$

Supply Voltage $V_s = 16 \text{ mV}$

The finalized simulation response is plotted in Figure 4(c).

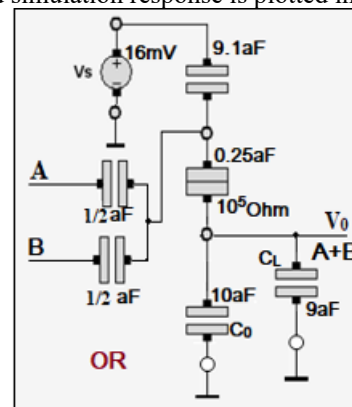


Figure 4 (b): OR gate

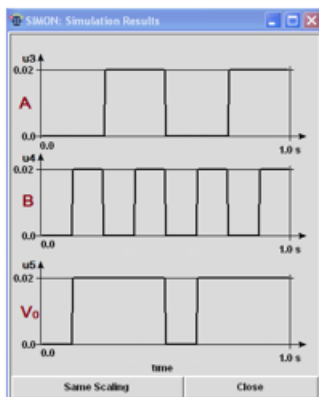


Figure 4 (C): Simulation result of OR gate

6) Threshold Logic Representation of an XOR Gate

The two-variable **Exclusive-OR (XOR)** logic function for inputs A and B is defined by the Boolean expression: $XOR(A, B) = \bar{A}B + A\bar{B}$. This function is not linearly separable because any Boolean expression containing both a variable and its complement cannot be separated by a single hyperplane. Consequently, a standard two-input XOR gate cannot be implemented using a single Linear Threshold Gate (LTG).

- For $P = A\bar{B}$: logic input "0" = 0V, logic "1" = 16mV, $C = 1\text{aF}$, $C_1^p = C_1^n = \frac{1}{2}C = 0.5\text{aF}$, $C_b = 9.5\text{aF}$, $C_j = 0.25\text{aF}$, $C_L = 9\text{aF}$, $C_0 = 9.5\text{aF}$, $R_j = 10^5\Omega$, $V_b = V_s = 16\text{mV}$

To realize $P = (A\bar{B})$ using threshold logic, the expression must be broken down into linearly separable sub-functions.

First, let the term $P = (A\bar{B})$ be represented by a threshold equation:

$$P = \text{sgn}\{A + \bar{B} - 2\} \quad (12)$$

Using the Boolean identity $\bar{B} = 1 - B$, we substitute this into equation (12):

$$P = \text{sgn}(A + (1 - B) - 2) \quad (13)$$

Finally, the complete XOR function can be reconstructed by combining the sub-functions, allowing it to be mapped to a network of threshold gates:

$$Y = XOR(A, B) = P + \bar{A}B \quad (14)$$

To determine the threshold logic gate (TLG) configuration for equation (14), we construct the truth table shown in Table-5.

The fifth column of this table defines four distinct mathematical inequalities. While equation (14) includes the term P, it is important to note that P is not an independent variable. Because $P = (A\bar{B})$, its value depends entirely on the two primary variables, A and B.

Using these variables, the TLG expression for the output Y is defined as:

$$Y = \text{sgn}\{W_1A + W_2B + W_3P - (\theta)\} \quad (15)$$

Where W_1, W_2, W_3 represent the weights assigned to each input, and θ represents the gate threshold. The

corresponding logical transitions and conditions are detailed in Table-5 below.

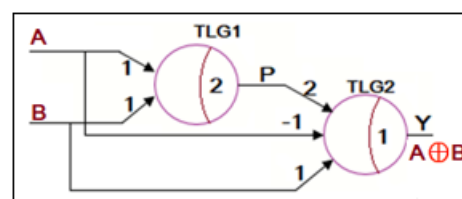
Table 5

A	B	P	Y	θ
0	0	0	0	$0 < \theta$
0	1	0	1	$W_2 \geq \theta$
1	0	1	1	$W_1 + W_3 \geq \theta$
1	1	0	0	$W_1 + W_2 < \theta$

Solving the inequalities in the fifth column of Table 5 yields the solution set $W_1, W_2, W_3; \theta = \{-1, 1, 2 : 1\}$. Consequently, the threshold equation for Y is expressed as:

$$Y = \text{sgn}\{-A + B + 2P - (1)\} \quad (16)$$

Applying the Threshold Logic Gate (TLG) expression from Equation (16) enables the construction of the depth-2 threshold logic circuit illustrated in Fig. 5(a).

Figure 5 (a): TLG implementation of $XOR(A, B) = \bar{A}B + A\bar{B}$

Simulation parameters were empirically measured to implement the routing path node framework from source to goal. The specific component values configured for the design are detailed below:

- For $Y = XOR(A, B) = P + \bar{A}B$: 0" = 0V, logic "1" = 16mV, $C = 1\text{aF}$, $C_1^p = 0.5C_2^p = C_1^n = \frac{1}{2}C = 0.5\text{aF}$, $C_b = 8.5\text{aF}$, $C_j = 0.25\text{aF}$, $C_L = 9\text{aF}$, $C_0 = 9.5\text{aF}$, $R_j = 10^5\Omega$, $V_b = V_s = 16\text{mV}$.

Applying these operational parameters, the E-cube routing circuit was constructed and is depicted in Fig. 8(a).

To evaluate whether the expression $Y = XOR(A, B) = P + \bar{A}B$ (Equation 14) is linearly separable, we analyze its three constituent variables: A, B, and P. Because no single variable appears in both its true and complemented forms simultaneously within this expression, it satisfies the algebraic condition for linear separability. Because a single surface in Fig. 7 isolates the five red bubbles from the colorless ones, the function $Y = XOR(A, B) = P + \bar{A}B$ is confirmed to be linearly separable.

This characteristic is confirmed visually and logically through Table 5 and Figure 7. The truth table for $P + \bar{A}B$ mapped onto the three-dimensional space in Figure 7, where logical outcomes of 1 are marked with solid red circles and outcomes of 0 with open, colorless circles. As demonstrated in the plot, a single decision plane can perfectly segregate the red circles from the colorless ones. This geometric separation proves that the expression is linearly separable, enabling its implementation as a linear threshold logic gate with a depth of two, as illustrated in Figures 6(a) and 8(a).

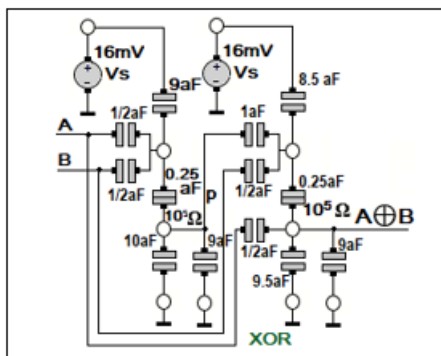
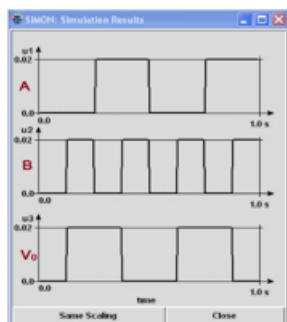
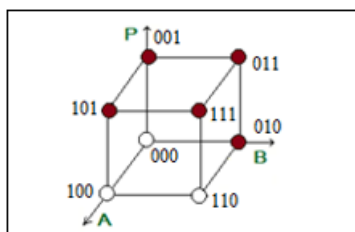
Figure 6 (a): Set of $A \oplus B = P + \bar{A}B$ Figure 6 (b): Simulation result of $A \oplus B$ 

Figure 7: Solution space Separation

Table 6: Truth table of $Y = (P + \bar{A}B)$

A	B	P	Y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Based on Algorithm 1, a Threshold Logic Gate (TLG)-based circuit can be constructed to predict the binary node numbers along a given routing path. As shown in Fig. 8(a), this architecture integrates 56 XOR gates designed using the implementation from Fig. 6(a). The circuit initiates execution immediately upon receiving the 7-bit binary inputs for the source ($S_6S_5S_4S_3S_2S_1S_0$) and destination ($d_6d_5d_4d_3d_2d_1d_0$).

The operation proceeds in two primary stages:

- 1) Intermediate Value Generation: The circuit first computes an intermediate value, $k = k_7k_6k_5k_4k_3k_2k_1$
- 2) Node Identity Prediction: Leveraging the initial source signals, destination signals, and the calculated bits k_1 through k_7 , the system determines the explicit identities of the sequential routing nodes: *source2*, *source3*, *source4*, *source5*, *source6*, *source7*, and the final destination (*goal*).

Consequently, this circuit effectively tracks real-time data/message movement and forecasts the precise nodes required to complete the routing path. The corresponding input-output relationships, verified via SIMON simulation, are illustrated in Fig. 8(b).

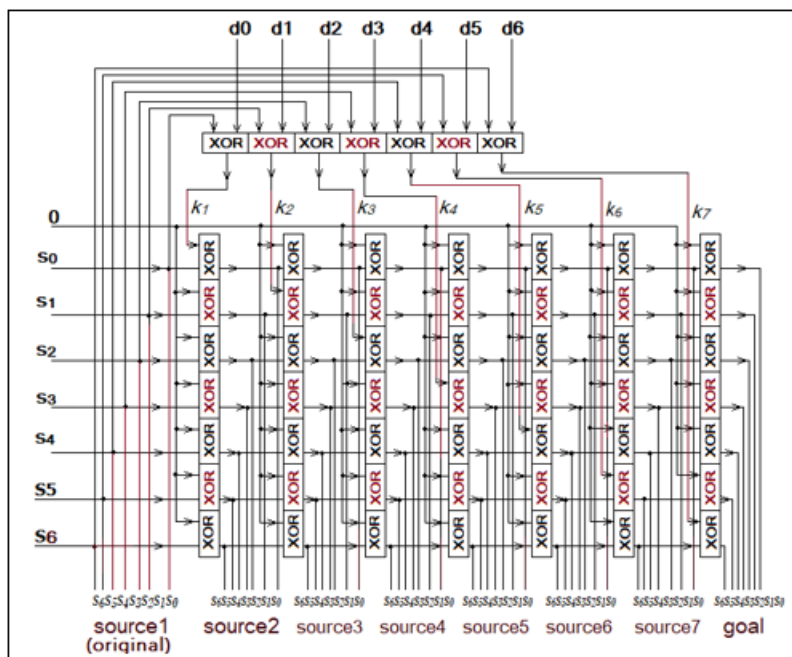


Figure 8 (a): Shows the circuit used to calculate the path metrics $\{k_1k_2k_3k_4k_5k_6k_7\}$ and all intermediate node values ($S_6S_5S_4S_3S_2S_1S_0$) that connect the source and destination points.

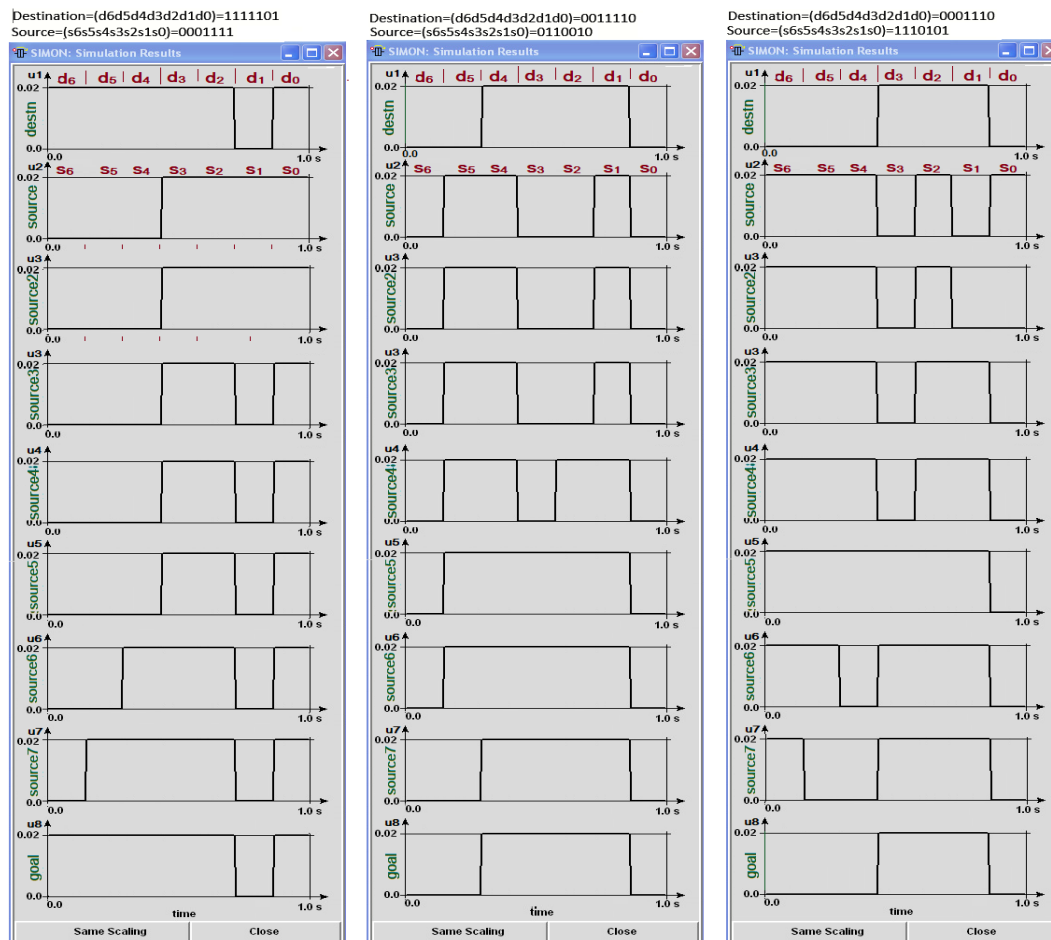


Figure 8 (b): Simulation results for three source-destination pairs, based on the configuration shown in Fig. 8(a)

7) Delay, Switching Energy, and Operating Speed

Every electronic circuit exhibits a characteristic processing delay that varies based on its architecture. The proposed circuit in Fig. 8(a) inevitably relies on specific threshold gates, namely AND, OR, and XOR. Each of these logic gates possesses a distinct propagation delay. To evaluate and compare the performance and efficiency of any circuit—whether conventional, nano-scale, or pico-scale—several critical figures of merit must be analyzed:

- Component count and overall circuit complexity
- Execution time and operating speed
- Power consumption and switching energy
- Fan-out capabilities
- Material costs and manufacturing constraints
- Controlling charge dynamics
- Environmental stability, including operating temperature limits

To calculate the processing or switching delay in a multi-input threshold logic gate, we must consider the tunnel junction capacitance C_j and its internal critical voltage V_c . Based on the established logarithmic expression for probability error [3-7, 13-14], the switching delay is defined by the following equation:

$$\text{Delay} = -(e|\ln(P_{\text{error}})|R_t) / (|V_j| - V_c) \dots\dots\dots (17)$$

In this expression, V_c represents the critical voltage, V_j is the tunnel junction voltage, and V_j is the internal resistance of the tunnel junction. A tunneling event—and thus the execution of the logic gate—occurs when the tunnel junction voltage V_j , which acts as a fraction of the bias voltage V_b , meets or exceeds the internal critical voltage V_c such that $|V_j| \geq V_c$.

For a 2-input AND gate meeting the condition $|V_j| \geq V_c$ the junction voltage is $V_j = 10.80$, and the critical voltage is $V_c = 11.58$ mV [4–7]. Using Equation (17) with an error $P_{\text{error}} = 10^{-12}$ and tunnel resistance $R_t = 10^5 \Omega$, the gate delay computes to $0.062|\ln(P_{\text{error}})| = 1.71$ ns. Table 7 lists the evaluated delays for the remaining circuits.

When an electron tunnels through the junction barrier, it changes the total circuit energy. This energy shift ΔE is expressed as:

$$\Delta E = E_{\text{before tunnel}} - E_{\text{after tunnel}} = -e(V_c - |V_j|) \dots\dots\dots (18)$$

This represents the active switching energy dissipated during a tunneling event. Individual energy consumption values for the TLG circuits are provided in Table 7.

Table 7

Gate/Device	SET-based delay ns	LTG-based delay ns	Speed-up (times)	Switching Energy meV
2-input NAND	4	$0.062 \ln(P_{error}) = 1.71$	2.33	10.80
2-input XOR	4	$0.080 \ln(P_{error}) = 2.21$	1.81	21.20
3-input OR	8	$0.073 \ln(P_{error}) = 2.01$	3.98	11.58
2-input NOR	4	$0.060 \ln(P_{error}) = 1.65$	2.42	10.70
2-input OR	4	$0.062 \ln(P_{error}) = 1.71$	2.33	10.80
2 ⁷ node network	224	$4.48 \ln(P_{error}) = 123.76$	1.81	1187.2

To evaluate performance, the processing delays of CMOS, single-electron transistor (SET), and linear threshold gate (LTG)-based designs were compared. For standard CMOS logic gates—including NAND, NOR, AND, and XOR- the processing delay was determined to be 12 ns [5–11, 17–18]. In comparison, the electron tunneling delay in a SET is approximately 4 ns [4–10, 17–18]. The relative speed-up of LTG-based delays compared to SET-based delays is detailed in Table 7.

Figure 9(a) illustrates the switching delay as a function of the switching error probability, while Figure 9(b) presents the switching delay versus the unit capacitance C .

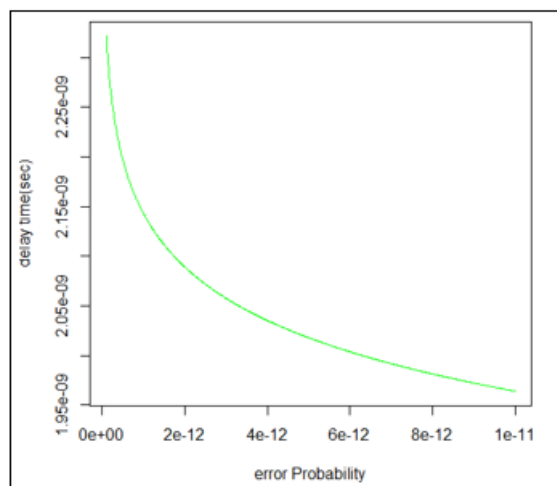


Figure 9 (a): Delay vs. Error Probability

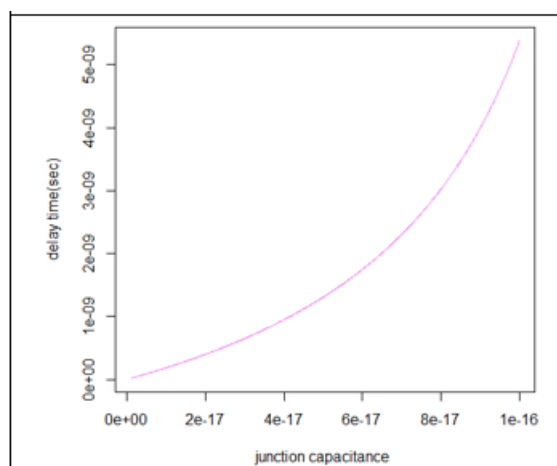


Figure 9 (b): Delay Vs. capacitance

3. Conclusion

Efficient packet routing is critical for ensuring deadlock-free data transmission across a network. When using deterministic routing, the path from the source node to the

destination node is strictly predefined by their respective addresses, inherently preventing routing conflicts. This study introduces an E-cube routing algorithm on a 7-dimensional hypercube implemented via an emerging nanotechnology based on threshold logic gates (TLGs). By utilizing a multiple-input single-output (MISO) TLG architecture, complex combinational and sequential logic circuits can be efficiently constructed. Due to the ultra-small footprint of the tunnel junctions (diameter approx. 10 nm), these TLG-based circuits achieve exceptionally high integration and functional density. Furthermore, the 7-dimensional hypercube routing architecture demonstrates ultra-low power consumption, requiring just $1187.2 \text{ meV} = 1187.2 \times 10^{-3} \times 1.602 \times 10^{-19} = 190.18 \times 10^{-24} \text{ Joule}$. In terms of processing delay, the multi-input TLG circuit outperforms traditional paradigms—operating over two times faster than Single Electron Transistor (SET) circuits and seven times faster than conventional CMOS-based equivalents. Simulation data reveals a maximum network simulator circuit delay of 123.76 ns, translating to a worst-case throughput performance of: $f = \frac{1}{123.76 \text{ ns/bit}} = 0.00808 \times 10^9 \text{ bit/sec} = 8.08 \text{ Mbit/sec}$. Consequently, a valid output is fanned out from the 7-dimensional hypercube network terminal every 123.76 ns. Crucially, an analysis of Table 1 confirms that no two points in any column are identical, proving that the network remains entirely conflict-free. This uniqueness verifies that the 7-dimensional hypercube routing process is completely immune to deadlocks. This indicates that the network is conflict-free, ensuring deadlock-free routing within the 7-dimensional hypercube. To validate this, nearly all of the designed circuits were tested using the SIMON simulator. The simulation data completely aligns with the theoretical results.

References

- [1] H. Sullivan and T. R. Bashkow, "A large scale homogeneous, fully distributed parallel machine," in *Proceedings of the 4th Annual Symposium on Computer Architecture*, vol. 5, Mar. 1977, pp. 105–125.
- [2] K. Hwang and N. Jotwani, *Advanced Computer Architecture: Parallelism, Scalability, Programmability*, 2nd ed. New York, NY, USA: McGraw-Hill.
- [3] C.-H. Liu, C.-C. Lin, *et al.*, "Threshold function identification by redundancy removal and comprehensive weight assignments," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 12, Dec. 2019.
- [4] A. K. Biswas, "Integrated-circuit random access memory based on an emerging technology—electron tunneling through tunnel junction," *International Journal of Scientific Research in Science, Engineering*

- and Technology (IJSRSET)*, vol. 8, no. 4, pp. 409–424, Jul.–Aug. 2021.
- [5] A. K. Biswas, “An accumulator using electron tunneling through tunnel junction,” *International Journal of Scientific Research in Science, Engineering and Technology (IJSRSET)*, vol. 8, no. 5, pp. 163–183, Sep.–Oct. 2021
- [6] Anup Kumar Biswas, “An Accumulator using Electron Tunneling Through Tunnel Junction” *International Journal of Scientific Research in Science, Engineering and Technology* Print ISSN: 2395-1990 | Online ISSN : 2394-4099 (www.ijsrset.com) Volume 8, Issue 5 Page Number : 163-183, September-October-2021
- [7] A.K. Biswas, “Measuring of an unknown voltage by using single electron transistor based voltmeter” , *Semiconductor Physics, Quantum Electronics & Optoelectronics(SPQEO)*, 24 (3), P. 277-287 (2021) August 2021; ISSN 1605-6582 (On-line)
- [8] Anup Kumar Biswas, “A High-Speed Bidirectional Register with Parallel Loading using single electron Threshold Logic Technology” ,*International Journal of Scientific Research in Science, Engineering and Technology (IJSRSET)*, Vol. 8 (4) : pp394-408; July-August-2021, Print ISSN: 2395-1990 | Online ISSN : 2394-4099 (www.ijsrset.com)
- [9] Anup Kumar Biswas, “Integrated-Circuit Random Access Memory based on an emerging Technology—electron tunneling through Tunnel Junction”, *International Journal of Scientific Research in Science, Engineering and Technology (IJSRSET)*, Vol. 8 (4) pp.409-424 July-August-2021, Print ISSN: 2395-1990 | Online ISSN: 2394-4099
- [10] Anup Kumar Biswas, “A High-Speed Bidirectional Register with Parallel Loading using single electron Threshold Logic Technology”, *International Journal of Scientific Research in Science, Engineering and Technology* Print ISSN: 2395-1990 | Online ISSN: 2394-4099 (www.ijsrset.com), July-August-2021, 8 (4): pp394-408
- [11] Anup Kumar Biswas, “Application of single electron threshold logic gates and memory elements to an up-down Counter” *International Journal of Creative Research Thoughts (IJCRT)* | Volume 9, Issue 6 June 2021 | ISSN: 2320-2882
- [12] Anup Kumar Biswas, -“Implementation of A 4n-Bit Comparator based on IC Type 74L85 using Linear Threshold Gate Tunneling Technology” *International Journal of Engineering Research & Technology* ISSN: 2278-0181, Vol. 10 Issue 05, May-2021 pp.299-310,
- [13] Anup Kumar Biswas, “State Transition Diagram for A Pipeline Unit based on Single Electron Tunneling” *International Journal of Engineering Research & Technology (IJERT)* Vol. 10 Issue 04, April-2021pp.325-336, ISSN: 2278-0181
- [14] Anup Kumar Biswas, “Design of A Pipeline for A Fixed-Point Multiplication using Single Electron Tunneling Technology”, *International Journal of Engineering Research & Technology* ISSN: 2278-0181, Vol. 10 Issue 04, April-2021 pp. 86-98,
- [15] A. K. Biswas and S. K. Sarkar: “An arithmetic logic unit of a computer based on single electron transport system” (SPQEO) *Semiconductor Physics, Quantum Electronics & Opt-Electronics*. 2003. Vol 6. pp 91-96 No.1,
- [16] A.K. Biswas and S. K. Sarkar: -“Error Detection and Debugging on Information in Communication System Using Single Electron Circuit Based Binary Decision Diagram.” (SPQEO) *Semiconductor Physics Quantum electronics and opt electronics*, Vol. 6, pp.1-8, 2003
- [17] C. Lageweg, S. Cotfaña, and S. Vassiliadis, “A linear threshold gate implementation in single electron technology,” *IEEE Computer Society VLSI Workshop*, Apr. 2001, pp. 93
- [18] Jacob. Millman and C. C. Halkias; “Integrated Electronics- Analog and Digital Circuits and Systems - second edition” McGraw Hill Education;
- [19] Millman's *Electronic Devices & Circuits* 4th Edition (English, Paperback, Jacob Millman)
- [20] Xiaola Lin; P.K. McKinley; L.M. Ni, “ Deadlock-free multicast wormhole routing in 2-D mesh multicomputers” *IEEE Transactions on Parallel and Distributed Systems* , Volume: 5, Issue: 8, Aug 1994 Page(s): 793 - 804

Author Profile



Dr. Anup Kumar Biswas is an Associate Professor in the Department of Computer Science and Engineering at Kalyani Government Engineering College in West Bengal, India. With over 21 years of experience in teaching and academic research, his core scientific interests include computer arithmetic, parallel architectures, nanotechnology, and computer-aided design. Dr. Biswas earned his Ph.D. in Electronics and Telecommunication Engineering from Jadavpur University in 2005. During his doctoral studies, he served as a Senior Research Fellow (SRF) within the Faculty of Engineering and Technology (FET) from 2002 to 2005. An active researcher, he has contributed 30 papers to various respected national and international journals and conferences.