ARCHITECTURAL EFFECTUATION OF CONVOLUTIONAL HOMEOMORPHIC ERROR-CORRECTING CODES USING NANO CONTROLLED-SELECTORS AND LATTICE NETWORKS

Anas N. Al-Rabadi

Department of Computer Engineering, School of Engineering, The University of Jordan
&
Department of Artificial Intelligence, Faculty of Information Technology, Zaytoonah University of Jordan

ABSTRACT

A new nano-based architectural design of multiple-stream convolutional homeomorphic error-control coding will be conducted, and a corresponding hierarchical implementation of important class of the homeomorphic Viterbi algorithm within convolutional homeomorphic error-control coding using lattice networks via nano carbon-based field emission controlled-switching will be achieved. Error-correction coding is highly important in modern wireless networking and digital data transaction since channel coding is required to counteract data errors that are encountered in wireless data networking due to the corresponding existence of unavoidable channel noise. Further, the new lattice nano-based implementation will be useful for enhancing the error-control system performance such as the corresponding enhancements within error-correcting capability, regular synthesis, speed improvement and the minimization of power consumption. Logic homeomorphism describes properties-preserving logic mapping that is intrinsically bijective. Homeomorphic property in error-control coding is important since it is shown that the homeomorphism relationship between multiple-streams of data can be used for further correction of errors that are uncorrectable using the implemented decoding algorithm such as in the case of triple-errors that are uncorrectable using the classical Viterbi algorithm. Applications of the new regular nano-based homeomorphic architecture include low-power design of circuits and systems for enhanced and more reliable wireless data networking and transaction.

KEYWORDS

Error-Control Coding, Homeomorphic Logic, Lattice Networks, Low-Power Computing, Regular Circuits and Systems.

1. INTRODUCTION

Nano computing will play an increasingly crucial role in building more compact and less power consuming computers in current and future technologies [2,4-6,8,9]. Due to this fact, and because several nano-scale computer gates should be homeomorphic such as in nano-scale quantum computing systems [2,3,7], homeomorphic computing will have an increasingly more existence in the future design of regular, compact, and universal circuits and systems. In general, (k, k) homeomorphic circuits are circuits that have the same number of inputs (k) and outputs (k) and are

one-to-one mappings between vectors of inputs and outputs, thus the vector of input states can be always uniquely reconstructed from the vector of output states [2,7].

Other motivations for pursuing the possibility of implementing circuits and systems using homeomorphic logic would include items such as: (1) *power*: the fact that, theoretically, the internal computations in homeomorphic systems consume no power; the amount of energy (heat) dissipated for every non-homeomorphic bit operation is given by $K \cdot T \cdot \ln(2)$, where K is the Boltzmann constant and T is the operating temperature, and that a necessary (but not sufficient) condition for not dissipating power in any physical circuit is that all system circuits must be built using fully homeomorphic logical components. Thus, *homeomorphic logic circuits are information-lossless*.

Acknowledgement: This research was performed during sabbatical leave in 2024-2025 granted to the author from The University of Jordan and spent at Zaytoonah University of Jordan.

For this reason, different technologies have been studied to implement homeomorphic logic in hardware such as in bioinformatics, nanotechnology-based circuits and systems, adiabatic CMOS VLSI circuit design, optical systems, and quantum circuits [2,3,5-7]. Fully homeomorphic digital systems will greatly reduce the power consumption (theoretically eliminate) through three conditions: (i) *logical homeomorphism*: the vector of input states can always be uniquely reconstructed from the vector of output states, (ii) *physical homeomorphism*: the physical switch operates backwards as well as forwards, and (iii) the use of *"ideal-like" switches* that have no parasitic resistances; (2) *size*: the current trends related to more dense hardware implementations are heading towards 1 Angstrom (atomic size), at which nano mechanical effects have to be accounted for; and (3) *speed* (*performance*): significant nano-scale computational speed improvements can be expected.

In general, in data communications between two communicating systems (nodes), noise exists and corrupts the sent data messages, and thus noisy corrupted messages will be received. The corrupting noise is usually sourced from the communication channel. Therefore, error correction of communicated data and homeomorphic error correction of communicated batch of data (i.e., parallel data streams) are highly important tasks in situations where noise occurs [1,3,10,11,13]. Many solutions have been classically implemented to solve for the classical error detection and correction problems: (1) one solution to solve for error-control is *parity checking* [11] which is one of the most widely used methods for error detection in digital logic circuits and systems, in which re-sending data is performed in case error is detected in the transmitted data. This error is detected by the parity checker in the receiver side. Various parity-preserving circuits have been implemented in which the parity of the outputs matches that of the inputs, and such circuits can be fault-tolerant since a circuit output can detect a single error; (2) another solution to solve this highly important problem, that is to extract the correct data message from the noisy erroneous counterpart, is by using various coding schemes that work optimally for specific types of noise [1,10,11,13].

The main contribution of this paper is the introduction of new nano-based implementation of convolution-based error-control coding that apply the homeomorphism property in both the convolution-based encoder for multiple-stream error-control encoding and in the new homeomorphic Viterbi decoding algorithm for multiple-stream error-control decoding. Figure 1 shows the introduced new system hierarchy implementation in this article, where the first bottom level represents the applied mathematics of Galois algebra which will be used in the upper levels, the second middle level represents the used applied physics which is comprised of three sub-levels of field-emission physics and carbon-based field-emission devices and field-emission circuits, and the third upper level represents the implemented computation which is comprised of two sub-

levels of logic function implementation using regular two-dimensional lattice networks and system implementation of the homeomorphic Viterbi algorithm.

Homeomorphic Viterbi Implementation
Lattice Network Realizations
Field Emission-Based Circuits
Carbon-Based Field Emission Devices
Field-Emission Physics
Galois Algebra

Figure 1. The introduced and implemented system design hierarchy.

This article is organized as follows: Basic background in error-control coding, homeomorphic logic and homemorphic error-control coding is presented in Section 2. Fundamentals of lattice networks is presented in Section 3. Basics of computing using carbon nano devices is presented in Section 4. The new nano circuit design of the homeomorphic Viterbi algorithm is introduced in Section 5. Conclusions are presented in Section 6.

2. HOMEOMORPHIC ERROR CORRECTION

This Section presents basic background in the topics of error-correction coding, homeomorphic logic and homeomorphic error-control coding. The fundamentals presented in this section will be utilized in the development of the new results which will be introduced in Section 5.

2.1. Error-Control Coding

In data communication, noise usually is generated from the channel in which transmitted data is communicated. Such noise corrupts sent messages from one end and thus noisy corrupted messages are received on the other end. To solve the problem of extracting a correct message from its corrupted counterpart, noise must be modeled and accordingly an appropriate encoding / decoding communication schemes must be implemented. For this reason, various coding schemes have been proposed and one very important family is the convolutional codes [1,10,11,13].

Each of the two nodes' sides in a networked system consists of three major parts: (1) *encoding* (e.g., generating a convolutional code using a convolutional encoder) to generate an encoded transmitted message, (2) *channel noise*, and (3) *decoding* (e.g., generating the correct convolution code using the corresponding decoding algorithm such as the Viterbi algorithm) to generate the decoded correct received data message. In general, in block coding, the encoder receives a k-bit message block and generates an n-bit code word, and therefore code words are generated on a block-by-block basis, and the whole message block must be buffered before the generation of the associated code word. On the other hand, message bits are received serially rather than in blocks where it is undesirable to use a buffer. In such case, one uses convloutional coding, in which a convolutional coder generates redundant bits by using modulo-2 convolutions. The binary convolutional encoder can be seen as a finite state machine consisting of an M-stage shift register with interconnections to n modulo-2 adders and a multiplexer to serialize the outputs of the adders, in which an L-bit message sequence generates a coded output sequence of length $n(L + M)$ bits.

Definition 1. For an L-bit message sequence, M-stage shift register, n modulo-2 adders, and a generated coded output sequence of length $n(L + M)$ bits, the code rate r is calculated as $r = \frac{L}{n(L+M)}$ bits / symbol.

Definition 2. The constraint length K of a convolutional code is the number of shifts over which a single message bit can influence the encoder output. Thus, for an encoder with an M-stage shift register, the number of shifts required for a message bit to enter the shift register and then come out of it is equal to $K = M + 1$.

A binary convolutional code can be generated with code rate $r = \frac{k}{n}$ by using k shift registers, an n modulo-2 adders, an input multiplexer, and an output multiplexer. An example of a convolutional encoder with constraint length = 3 and rate = ½ is the one shown in Figure 2.

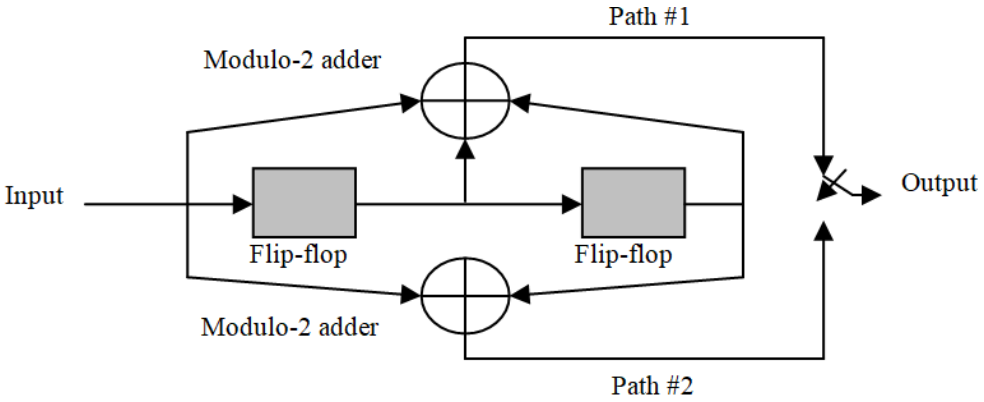


Figure 2. Convolutional encoder with constraint length = 3 and rate = ½ . The flip-flop is a unit-delay element, and the modulo-2 adder is the logic XOR operation.

The convolutional codes generated by the encoder in Figure 2 are part of what is generally called *nonsystematic codes*. Each path connecting the output to the input of a convolutional encoder can be characterized in terms of the impulse response which is defined as the response of that path to "1" applied to its input, with each flip-flop of the encoder set initially to "0". Equivalently, we can characterize each path in terms of a generator polynomial defined as the unit-delay transform of the impulse response. More specifically, the generator polynomial is defined as:

$$g(D) = \sum_{i=0}^{M} g_i D^i \qquad (1)$$

where g_i is the generator coefficients $\in \{0, 1\}$, and the generator sequence $\{g_0, g_1, \ldots, g_M\}$ composed of generator coefficients is the impulse response of the corresponding path in the convolutional encoder, and D is the unit-delay variable.

Example 1. For the convolutional encoder in Figure 2, path #1 impulse response is (1, 1, 1), and path #2 impulse response is (1, 0, 1). Thus, according to Equation (1), the following are the corresponding generating polynomials, respectively, where addition is performed in modulo-2 addition arithmetic:

$g_1(D) = 1 \cdot D^0 + 1 \cdot D^1 + 1 \cdot D^2 = 1 + D + D^2$
$g_2(D) = 1 \cdot D^0 + 0 \cdot D^1 + 1 \cdot D^2 = 1 + D^2$

For a message sequence (10011), the following is the D-domain polynomial representation:

$$m(D) = 1 \cdot D^0 + 0 \cdot D^1 + 0 \cdot D^2 + 1 \cdot D^3 + 1 \cdot D^4 = 1 + D^3 + D^4$$

As convolution in time domain is transformed into multiplication in the D-domain, path #1 output polynomial and path #2 output polynomial are as follows, respectively:

$$c_1(D) = g_1(D)m(D) = (1 + D + D^2)(1 + D^3 + D^4) = 1 + D + D^2 + D^3 + D^6$$
$$c_2(D) = g_2(D)m(D) = (1 + D^2)(1 + D^3 + D^4) = 1 + D^2 + D^3 + D^4 + D^5 + D^6$$

Therefore, the output sequences of paths #1 and #2 are as follows, respectively:
Output sequence of path #1: (1111001)
Output sequence of path #2: (1011111)

The resulting encoded sequence from the convolutional encoder in Figure 2 is obtained by multiplexing the two output sequences of paths #1 and #2 as follows:
c = (11, 10, 11, 11, 01, 01, 11)

Example 2. For the convolutional encoder in Figure 2, the following are examples of encoded data messages:

m_1=(11011)$\rightarrow c_1$=(11010100010111), m_2=(00011) $\rightarrow$ c_2=(00000011010111), m_3=(01001) $\rightarrow$ c_3=(00111011111011)

In general, a data message sequence of length L bits results in an encoded sequence of length equals to $n(L + K - 1)$ bits. Usually a terminating sequence of $(K – 1)$ zeros called the tail of the message is appended to the last input bit of the message sequence in order for the shift register to be restored to its zero initial state.

The structural properties of the convolutional encoder (cf. Figure 2) can be represented graphically in several equivalent representations using: (1) code tree, (2) trellis, and (3) state diagram [11,13]. The trellis contains $(L + K)$ levels where L is the length of the incoming message sequence and K is the constraint length of the code. Therefore, the trellis form is preferred over the code tree form because the number of nodes at any level of the trellis does not continue to grow as the number of incoming message bits increases, but rather it remains constant at 2^{K-1}, where K is the constraint length of the code. An important decoder that uses the trellis representation to correct received erroneous messages is the Viterbi decoding algorithm [10,11,13]. The Viterbi algorithm is a dynamic programming algorithm used to find the maximum-likelihood sequence of hidden states, which results in a sequence of observed events particularly in the context of hidden Markov models, that forms a subset of information theory with wide range of applications including speech recognition, keyword spotting, computational linguistics, and modern digital communications.

The Viterbi algorithm is a maximum-likelihood decoder which is optimum for a noise type which is statistically characterized as an Additive White Gaussian Noise. This algorithm operates by computing a metric for every possible path in the trellis representation. The metric for a specific path is computed as the Hamming distance between the coded sequence represented by that path and the received sequence. For a pair of code vectors c_1 and c_2 that have the same number of elements, the Hamming distance $d(c_1, c_2)$ between such a pair of code vectors is defined as the number of locations in which their respective elements differ. In the Viterbi algorithm context, the Hamming distance is computed by counting how many bits are different between the received channel symbol pair and the possible channel symbol pairs, in which the results can only be “0”,

"1" or "2". Therefore, for each node (i.e., state) in the trellis the Viterbi algorithm compares the two paths entering the node. The path with the lower metric is retained and the other path is discarded. This computation is repeated for every level j of the trellis in the range $M \leq j \leq L$, where $M = K - 1$ is the encoder's memory and L is the length of the incoming message sequence. The paths that are retained are called survivor or active paths. In some cases, applying the Viterbi algorithm leads to the following difficulty: when the paths entering a node (state) are compared and their metrics are found to be identical then a choice is made by making a guess (i.e., flipping a fair coin). The Viterbi algorithm is a maximum likelihood sequence estimator [10,11,13], where the following procedure and Example 3 illustrate the detailed steps for the implementation of this algorithm.

Algorithm Viterbi

---

1. *Initialization step*: Label the left-most state of the trellis (i.e., all zero state at level 0) as 0
2. *Computation step*: Let $j = 0, 1, 2, \ldots$, and assume at the previous j the following is performed:
    a. All survivor paths are identified
    b. The survivor paths and its metric for each state of the trellis are stored

   Then, at level (clock time) $(j+1)$ and **for** all the paths entering each state of the trellis, compute the metric by adding the metric of the incoming branches to the metric of the connecting survivor path from level j. Thus, for each state, identify the path with the lowest metric as the survivor of step $(j + 1)$, therefore updating the computation
3. *Final step*: **Continue** the computation until the algorithm completes the forward search through the trellis and thus reaches the terminating node (i.e., all zero state), at which time it makes a decision on the maximum-likelihood path. **Then**, the sequence of symbols associated with that path is released to the destination as the decoded version of the received sequence

---

Example 3. For the convolutional encoder in Figure 2, path #1 impulse response is (1, 1, 1), and path #2 impulse response is (1, 0, 1). Thus, the following are the corresponding generating polynomials, respectively:

$$g_1(D) = 1 \cdot D^0 + 1 \cdot D^1 + 1 \cdot D^2 = 1 + D + D^2,\ g_2(D) = 1 \cdot D^0 + 0 \cdot D^1 + 1 \cdot D^2 = 1 + D^2$$

For a message sequence (101), the following is the D-domain polynomial representation:

$$m(D) = 1 \cdot D^0 + 0 \cdot D^1 + 1 \cdot D^2 = 1 + D^2$$

As convolution in time domain is transformed into multiplication in the D-domain, the path #1 output polynomial and path #2 output polynomial are as follows, respectively, where addition is performed in modulo-2 arithmetic:

$$c_1(D) = g_1(D)m(D) = (1 + D + D^2)(1 + D^2) = 1 + D + D^3 + D^4,\ c_2(D) = g_2(D)m(D) = (1 + D^2)(1 + D^2) = 1 + D^4$$

Therefore, the output sequences of paths #1 and #2 are as follows, respectively:

Output sequence of path #1: (11011)
Output sequence of path #2: (10001)

The resulting encoded sequence from the convolutional encoder in Figure 2 is obtained by multiplexing the two output sequences of paths #1 and #2 as follows:

c = (11, 10, 00, 10, 11)

Now suppose a noise corrupts this sequence and the noisy received sequence is as follows: c' = (01, 10, 10, 10, 11).

Using the Viterbi algorithm, Figure 3 shows the resulting survivor path which generates the correct sent message c = (11, 10, 00, 10, 11).

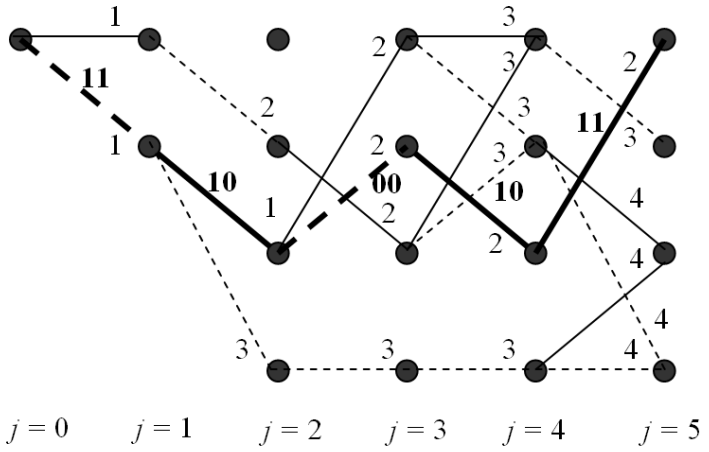


Figure 3. The resulting survivors of the Viterbi algorithm for Example 3, where the bold path is the survivor path.

A difficulty with the application of the Viterbi algorithm occurs when the received sequence is very long. In this case the Viterbi algorithm is applied to a truncated path memory using a decoding window of length greater or equal five times the convolutional code constraint length K, in which the algorithm operates on a frame-by-frame of the received sequence each of length $l \geq 5K$. The decoding decisions made in this way are not a truly maximum likelihood, but they can be made almost as good provided that the decoding window is long enough. Another difficulty is the number of errors; for example, in case of three errors, the Viterbi algorithm when applied to a convolutional code of r = ½ and K = 3 cannot produce a correctable decoded message from the incoming erroneous message. Exceptions are triple-error patterns that spread over a time span > K.

2.2. Homeomorphic Digital Logic

In general, an (n, k) homeomorphic circuit is a circuit that has n number of inputs and k number of outputs and is one-to-one mapping between vectors of inputs and outputs, thus the vector of input states can be always uniquely reconstructed from the vector of output states [2,3,7]. Thus, a (k, k) homeomorphic map is a bijective function which is both injective (one-to-one) and surjective (onto). (Such bijective systems are also known as one-to-one correspondence.) The auxiliary outputs that are needed only for the purpose of homeomorphism are called “garbage” outputs. These are auxiliary outputs from which a homeomorphic map is constructed (cf. Example 4). Therefore, homeomorphic systems are information-lossless.

Geometrically, achieving homeomorphism leads to value space-partitioning that leads to spatial partitions of unique values. Algebraically and in terms of systems representation, homeomorphism leads to multi-input multi-output (MIMO) bijective maps (i.e., bijective functions). An

algorithm called homeomorphic Boolean function (HomBF) that produces a homeomorphic form from a non-homeomorphic Boolean function is as follows [2].

Algorithm HomBF

1. To achieve (k, k) homeomorphism, add sufficient number of auxiliary output variables such that the number of outputs equals the number of inputs. Allocate a new column in the mapping table for each auxiliary variable
2. For construction of the first auxiliary output, assign a constant C_1 to half of the cells in the corresponding table column (e.g., zeros), and the second half as another constant C_2 (e.g., ones). For convenience, one may assign C_1 to the first half of the column, and C_2 to the second half of the column (cf. Table 1a, column W_1)
3. For the next auxiliary output, **If** non-homeomorphism still exists, **Then** assign for *identical* output tuples (non-homeomorphic map entries) values which are half zeros and half ones, and then assign a constant for the remainder that are already homeomorphic
4. **Do** step 3 until all map entries are homeomorphic

Example 4. The standard two-variable Boolean equivalence (XNOR): $W = c \otimes d$ is non-homeomorphic. The following table lists the mapping components of this XNOR function:

c	*d*	W
0	0	1
0	1	0
1	0	0
1	1	1

Applying the above HomBF algorithm, the following are four possible homeomorphic two-variable Boolean maps for the XNOR function:

Table 1. Four possible (2, 2) homeomorphic maps for the Boolean XNOR (Boolean equivalence).

c	*d*	W	W_1
0	0	1	**0**
0	1	0	**0**
1	0	0	**1**
1	1	1	**1**

(a)

c	*d*	W	W_1
0	0	1	**1**
0	1	0	**1**
1	0	0	**0**
1	1	1	**0**

(b)

c	*d*	W	W_1
0	0	1	**0**
0	1	0	**1**
1	0	0	**0**
1	1	1	**1**

(c)

c	*d*	W	W_1
0	0	1	**1**
0	1	0	**0**
1	0	0	**1**
1	1	1	**0**

(d)

For example, using the HomBF algorithm, the construction of the homeomorphic map in Table 1a is obtained as follows: since W is non-homeomorphic, assign auxiliary output W_1 and assign the first half of its values the constant "0" and the second half another constant "1". The new XNOR map is now homeomorphic. This gate is also called the inverted Controlled-NOT (inverted C-NOT) gate) in which: $W_1 = c$ and $W = c \otimes d = (c \oplus d)'$.

2.3. Error Correction Via the Homeomorphic Viterbi Algorithm

While in subsection 2.1 the error correction of communicated data was done for the case of single-input single-output (SISO) systems, this section presents homeomorphic error correction of communicated batch (parallel) of data in multiple-input multiple-output (MIMO) systems. Homeomorphism in parallel-based data communication is directly observed [3] since:

$$\vec{O}_1 = \vec{I}_2 \qquad (2)$$

where $\vec{O}_1$ is the *unique* output transmitted data from node #1 and $\vec{I}_2$ is the *unique* input received data to node #2.

In MIMO systems, the existence of noise will cause an error that may lead to non-homeomorphism in data communication (i.e., non-homeomorphism in data mapping) since $\vec{O}_1 \neq \vec{I}_2$. The implementation of homeomorphic error correction can be performed (1) in software using the homeomorphic error-correction algorithm and (2) in hardware using nano error correction hardware. The following algorithm, which is called Homeomorphic Viterbi (HV) Algorithm, presents the implementation of homeomorphic error correction [3].

Algorithm HV

1. Use the HomBF Algorithm to reversibly encode the communicated batch of data
2. Given a specific convolutional encoder circuit, determine the generator polynomials for all paths
3. **For** each communicated message within the batch, determine the encoded message sequence
4. **For** each received message, use the Viterbi Algorithm to decode the received erroneous message
5. Generate the total maximum-likelihood trellis resulting from the iterative application of the Viterbi decoding algorithm
6. Generate the corrected communicated batch of data messages
7. **End**

The convolutional encoding for the HV algorithm can be performed *serially* using a single convolutional encoder from Figure 2, or in *parallel* using the general parallel convolutional encoder circuit shown in Figure 4 in which several *s* convolutional encoders operate in parallel for encoding *s* number of simultaneously submitted messages (i.e., data message set of cardinality (size) equal to *s*) generated from *s* nodes.

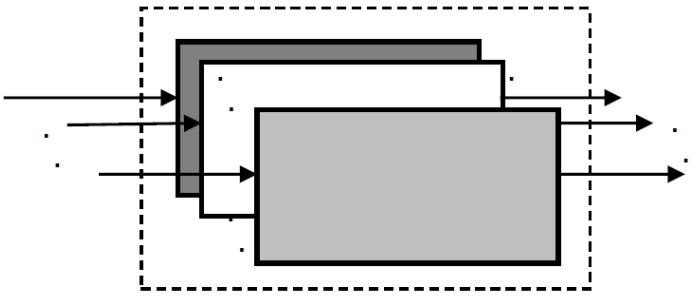


Figure 4. General MIMO encoder circuit for the parallel generation of convolutional codes where each box represents a single SISO convolutional encoder such as the one shown in Figure 2.

Example 5. The homeomorphism implementation (e.g., HomBF Algorithm) upon the following input bit stream $\{m_1 = 1, m_2 = 1, m_3 = 1\}$ produces the homeomorphic set of messages $\{m_1 = (101), m_2 = (001), m_3 = (011)\}$.

For the convolutional encoder in Figure 4, the following is the D-domain polynomial representations, respectively:

$m_1(D) = 1{\cdot}D^0 + 0{\cdot}D^1 + 1{\cdot}D^2 = 1 + D^2$, $m_2(D) = 0{\cdot}D^0 + 0{\cdot}D^1 + 1{\cdot}D^2 = D^2$, $m_3(D) = 0{\cdot}D^0 + 1{\cdot}D^1 + 1{\cdot}D^2 = D + D^2$

The resulting encoded sequences are generated in parallel as follows, respectively:

$$c_1 = (1110001011),\ c_2 = (0000111011),\ c_3 = (0011010111)$$

Now suppose noise sources corrupt these sequences, and the noisy received sequences are as follows:

$$c'_1 = (1111001001),\ c'_2 = (0100101011),\ c'_3 = (0010011111)$$

Using the HV algorithm, Figure 5 shows the resulting survivor paths which generate the correct sent messages:

$$\{c_1 = (1110001011),\ c_2 = (0000111011),\ c_3 = (0011010111)\}.$$

As in the non-homeomorphic Viterbi Algorithm, in some cases, applying the homeomorphic Viterbi (HV) algorithm leads to the following difficulties: (1) when the paths entering a node (state) are compared and their metrics are found to be identical then a choice is made by making a guess; (2) when the received sequence is very long and in this case the homeomorphic Viterbi algorithm is applied to a truncated path memory using a decoding window of length greater or equal five times the convolutional code constraint length K, in which the algorithm operates on a frame-by-frame of the received sequence each of length $l \geq 5K$, and the decoding decisions made in this way are not a truly maximum likelihood, but they can be made almost as good provided that the decoding window is long enough; (3) correctable decoded message for high number of errors.

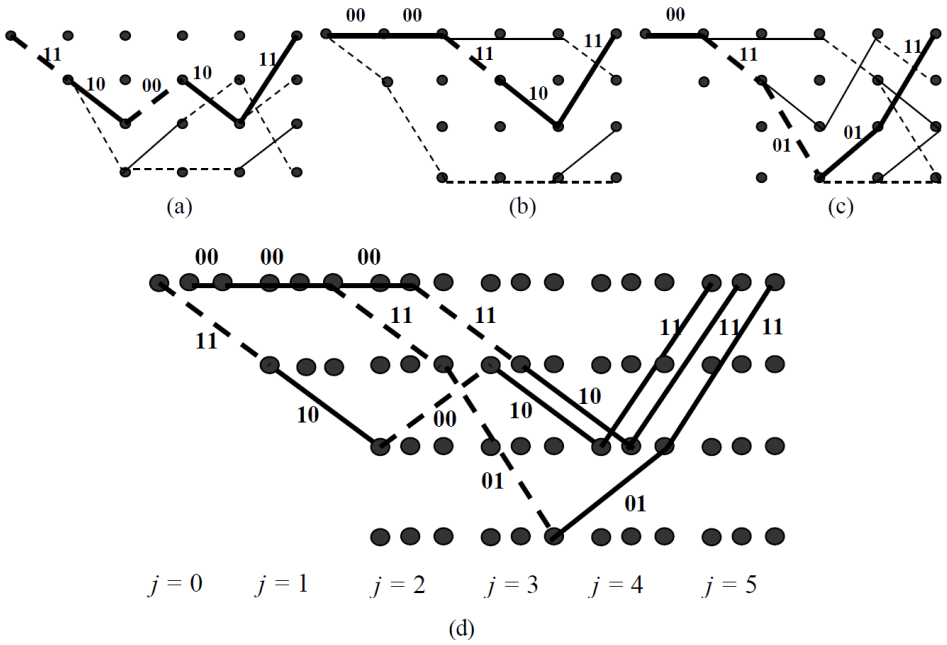


Figure 5. The resulting survivor paths of the HV algorithm when applied to Example 5.

Yet, parallelism in multi-stream data submission (transmission) allows for the possible existence of extra relationship(s) between the submitted data-streams that can be used for (1) detection of error existence and (2) further correction after HV algorithm in case the HV algorithm fails to correct for the occurring errors. Examples of such inter-stream relationships are: (1) parity (even and odd) relationship between the corresponding bits within the inter-stream submitted data, (2) homeomorphism relationship between the parallel submitted data streams and this relationship exists from applying a known homeomorphic mapping such as the HomBF algorithm, or (3) combination of parity and homeomorphism properties [3]. The homeomorphism property in the HV algorithm produces a homeomorphism relationship between the sent parallel streams of data, and this known homeomorphism mapping can be used to correct the uncorrectable errors (e.g., triple errors) which the HV algorithm fails to correct.

Example 6. The following is a version of the HomBF algorithm that produces homeomorphism as follows:

Algorithm HomBF (Version 1)

1. To achieve (k, k) homeomorphism, add sufficient number of auxiliary output variables (starting from right to left) such that the number of outputs equals the number of inputs. Allocate a new column in the mapping's table for each auxiliary variable
2. For construction of the first auxiliary output, assign a constant C_1 = "0" to half of the cells in the corresponding table column, and the second half as another constant C_2 = "1". Assign C_1 to the first half of the column, and C_2 to the second half of the column
3. For the next auxiliary output, **If** non-homeomorphism still exists, **Then** assign for *identical* output tuples (non-homeomorphic map entries) values which are half ones and half zeros, and then assign a constant for the remainder that are already homeomorphic which is the one's complement (NOT; inversion) of the previously assigned constant to that remainder
4. **Do** step 3 until all map entries are homeomorphic

For the parallel sent bit stream {1, 1, 1}in Example 5 in which the homeomorphism implementation (using Version 1 of the HomBF Algorithm) produces the following homeomorphic sent set of data sequences: {m_1 = (101), m_2 = (001), m_3 = (011)}. Suppose that m_1 and m_2 are decoded correctly and m_3 is still erroneous due to submission errors. Figure 6 shows possible tables in which erroneous m_3 exist:

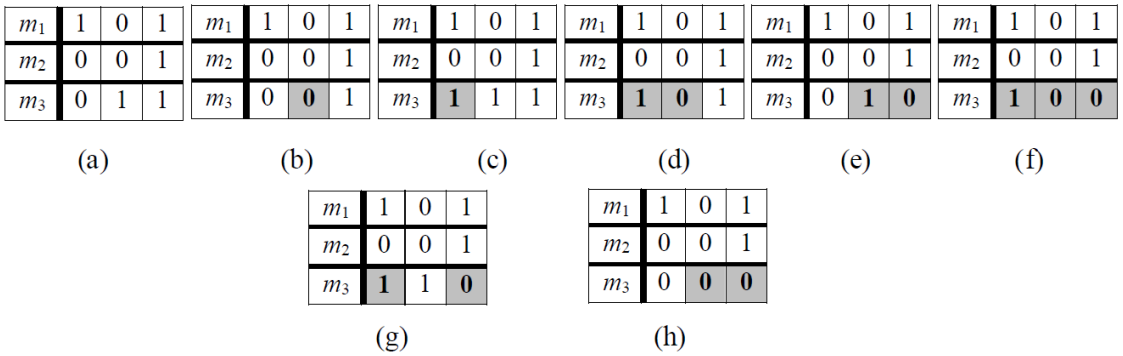


Figure 6. Tables for possible errors in data stream m_3 that is generated by the HomBF Algorithm V1: (a) original sent correct m_3 that resulted from the application of the HomBF Algorithm V1, and (b) – (h) possibilities of the erroneous received m_3.

Note that the erroneous m_3 is Figures 6b-6e and 6g-6h are correctable using the HV algorithm since less than triple-errors exits, but the triple error as in Figure 6f is (usually) uncorrectable using the HV algorithm. Yet, the existence of the homeomorphism property using the HomBF algorithm adds information that can be used to correct m_3 as follows: By applying the HomBF Algorithm (Version 1) from right-to-left in Figure 6f one notes that in the second column (from right) two "0" cells are added in the top in the correctly received m_1 and m_2 messages, which means that in the most right column the last cell must be "1" since otherwise the top two cells in the correctly received m_1 and m_2 messages should have been "0" and "1" respectively to achieve value space-partitioning. Now, since the 3rd cell of the most right column must be "1" then the last cell of the 2nd column from the right must be "1" also because of the uniqueness requirement according to the HomBF algorithm (Version 1) for value space-partitioning between the first two messages {m_1, m_2} and the 3rd message m_3. Then, and according to the HomBF algorithm (Version 1) the 3rd cell of the last column from right must have the value "0" which is the one's complement (NOT) of the previously assigned constant "1" to the 3rd cell of the 2nd column from the right. Consequently, the correct message m_3 = (011) is obtained.

3. LATTICE NETWORKS

It is well-known in logic synthesis that certain classes of logic functions exhibit specific types of symmetries [2]. One method to characterize a symmetry that might exist in a function is done using symmetry indices.

Definition 3. A single index symmetric function, denoted as $S^k(x_1, x_2,\ldots, x_n)$ has value 1 when exactly k of its n inputs are equal to 1, and exactly $(n - k)$ of its remaining inputs are 0.

Definition 4. The elementary symmetric functions of n variables are:
$S^0 = \bar{x}_1\bar{x}_2..\bar{x}_n$, $S^1 = x_1\bar{x}_2..\bar{x}_n + \bar{x}_1x_2\bar{x}_3..\bar{x}_n + ... + \bar{x}_1\bar{x}_2..\bar{x}_{n-1}x_n$, …, $S^n = x_1x_2..x_n$.

Thus, for a Boolean function of three variables, one obtains the following sets of symmetry indices: $S^0 = \{\bar{a}\bar{b}\bar{c}\}$, $S^1 = \{\bar{a}\bar{b}c, \bar{a}b\bar{c}, a\bar{b}\bar{c}\}$, $S^2 = \{\bar{a}bc, ab\bar{c}, a\bar{b}c\}$, and $S^3 = \{abc\}$. It has been shown [2] that a function which is not symmetric can be symmetrized by repeating its variables.

Example 7. The following Karnaugh map in Figure 7 demonstrates the symmetrization by repeating the variables of non-symmetric Boolean function $F = \bar{a} + b$.

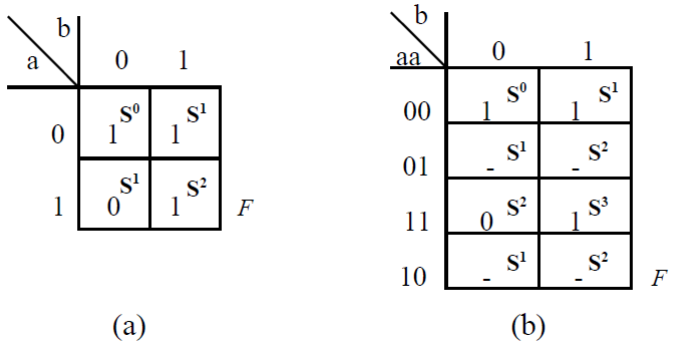


Figure 7. Symmetrization by repeating variables: (a) non-symmetric function F, and (b) symmetric function F.

One notes that while in Figure 7(a) conflicting values occur for symmetry index S^1 in minterms $\bar{a}b$ and $a\bar{b}$ and thus producing non-symmetric function, non-conflicting values are produced for the same non-symmetric function in Figure 7(b) by repeating variable $\{a\}$ two times. As stated previously, various applications of symmetry indices for the synthesis of logic functions have been previously shown. This includes symmetric networks, Akers arrays and lattice networks [2] among other several implementations. The concept of lattice networks for switching functions involves three components: (1) *expansion* of a function that corresponds to the root in the lattice which creates several successor nodes of the expanded node, (2) *joining* of several nodes to a single node, and (3) *geometry* to which the nodes are mapped. Figure 8(a) shows an example of a four-variable (i.e., four-level) lattice network and Figs. 8(b) and 8(c) show the relationship between Figure 8(a) and symmetry indices.

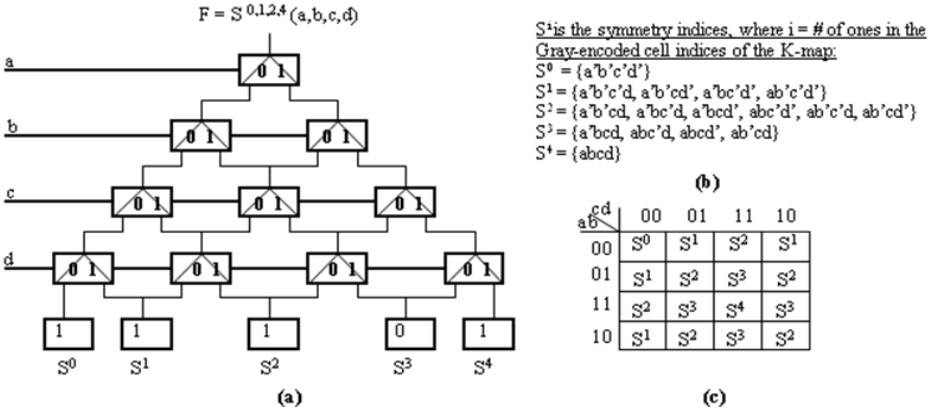


Figure 8. The relation between a lattice and symmetry indices: (a) lattice network for a four-variable function, (b) sets of binary symmetry indices, and (c) Karnaugh map interpretation of the binary symmetry indices.

It has been shown that every non-symmetric function can be symmetrized by repeating its variables [2]. Also, it has been shown that the realization of non-symmetric functions using lattice networks demands the repetition of variables [2].Therefore, since a single variable corresponds to a single level in the lattice network, repeating variables produces a repetition of levels in a lattice network. In general, three main factors control the size of a lattice network that realizes non-symmetric functions: (1) *expansion types* that are used in the internal nodes, (2) *order of variables* upon which functions are expanded in each level of the lattice, and (3) *choice of repeated variables*. Consequently, various optimization methods have been addressed for an optimal choice of the three factors that are mentioned above in order to minimize the size of the corresponding lattice network.

Figure 9 illustrates, as an example, the geometry of a four-neighbor lattice network and joining operations on the nodes where each cell has two inputs and two outputs (i.e., four neighbors). The construction of the lattice network in Figure 9 implements the following one possible convention of top-to-bottom expansion and left-to-right joining (i.e., left-to-right propagation of the corresponding correction functions in Figs. 9(c) and 9(d), respectively). The function that is generated by joining two nodes (sub-functions) in a lattice network is called the joined function. The function that is generated in nodes other than the joining nodes, to preserve the functionality within the lattice network, is called the correction function. Note that the lattices shown in Figure 9 preserve the functionality of the corresponding sub-functions f and g. This can be observed, for instance, in Figure 9(b) as the negated variable $\{a'\}$ will cancel the un-complemented variable $\{a\}$, when propagating the cofactors from the lower levels to the upper levels or vice versa, without the need for any correction functions to preserve the output functionality of the corresponding lattice network. This simple observation cannot be seen directly in Figures 9(c) and 9(d), as the correction functions are involved to cancel the effect of the new joining nodes for the preservation of the functionality of the new lattice networks (these correction functions are shown in the extreme right leaves of the second level in Figures 9(c) and 9(d), respectively).

Example 8. For the following non-symmetric three-variable Boolean function $F = a \cdot b + a' \cdot c$, by utilizing the joining rule that was presented in Figure 9(b) for two-dimensional lattice network with binary Shannon nodes, one obtains the lattice network shown in Figure 10. One can note that without the repetition of variable(s) (e.g., variable b in Figure 10) F cannot be produced by any lattice network. It is also to be noted that all internal nodes in Figure 10 are two-to-one multiplexers (i.e., selectors). In Figure 10, if one multiplies each leaf value, from left to right, with *all* possible bottom-up paths (from the leaves to the root F) and add them over Boolean algebra then one obtains the function F (i.e., the root) as follows:

$$F = (0 \cdot c' \cdot b' \cdot a') + (1 \cdot c \cdot b' \cdot a') + (0 \cdot c' \cdot b \cdot a') + (0 \cdot b' \cdot c \cdot b \cdot a') + (1 \cdot b \cdot c \cdot b \cdot a') + (1 \cdot c' \cdot b \cdot a) + (1 \cdot c \cdot b \cdot a)$$
$$= (1 \cdot c \cdot b' \cdot a') + (1 \cdot b \cdot c \cdot b \cdot a') + (1 \cdot c' \cdot b \cdot a) + (1 \cdot c \cdot b \cdot a) = a' \cdot c + a \cdot b.$$

One can observe that in order to represent the non-symmetric function in Example 8 in the lattice network, variable b is repeated, where the nodes in Figure 10 are Shannon nodes, which are merely two-input one-output multiplexers, whose output goes in two directions, with the set of variables $\{a, b, c\}$ operate as control signals.

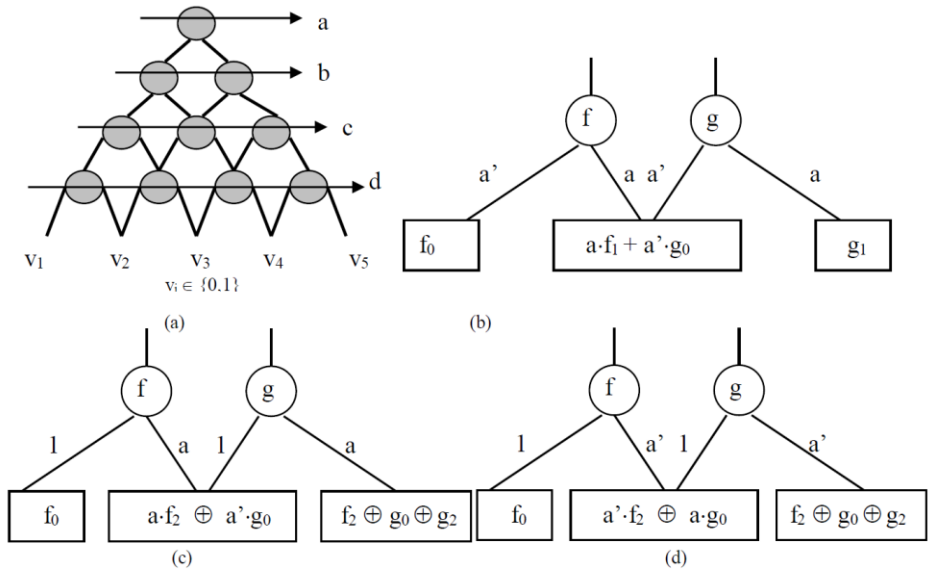


Figure 9. Lattice networks: (a) A two-dimensional 4-neighbor lattice network, (b) Shannon lattice network, (c) positive Davio lattice network, and (d) negative Davio lattice network.

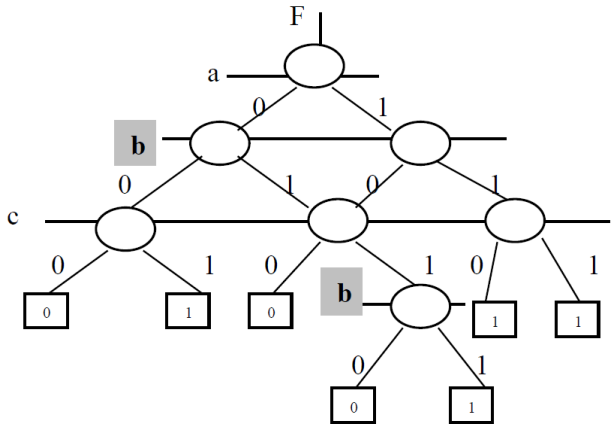


Figure 10. Shannon lattice network for the corresponding non-symmetric function $F = a \cdot b + a' \cdot c$.

Again, it is to be noted that all internal nodes in Figure 10 are merely two-to-one multiplexers. Regular lattice networks that were presented in this section will be used to synthesize the corresponding internal functions within the homeomorphic Viterbi algorithm which will be introduced in Section 5.

4. Carbon Field Emission – Based Controlled Switching and Computing

This section presents important and needed background on carbon nanotubes (CNT) where the presented carbon-based nanotubes will be utilized in this section within building controlled-switching devices and circuits that will be used later in Section 5 for the construction of nano-based lattice homeomorphic Viterbi circuits.

4.1. Carbon Nanotubes Characteristics and Properties

The CNT, which is a cylindrical sheet of graphite, is formed geometrically in two distinct forms which affect CNT properties: (a) straight CNT in which a CNT is formed as a straight cut from graphite sheet and rolled into a carbon nanotube and (b) twisted CNT in which a CNT is formed as a cut at an angle from graphite sheet and rolled into a carbon nanotube. The newly emerging CNT technology has been implemented in many new promising applications [4-6,8]. This includes TVs that are based on field emission of CNTs that consume much less power, thinner and have much higher resolution than the best plasma-based TVs available, nano circuits based on CNTs such as CNT-based FETs that consume less power and are much faster than the available silicon-based FETs, carbon nanocoils that can be used as inductors in nanofilters and as nanosprings in nano dynamic systems, and CNT rings. In addition, the CNT has also been demonstrated for exciting applications such as in CNT probes, new composite materials, and CNT data storage devices capable of storing more than 10^{15} bytes/cm^2.

4.2. Carbon Nanotubes-Based Field Emitters

Field electron emission is the emission of electrons from the surface of a cathode under the influence of the applied electric field (of 3 x 10^9 V/ m) which is strongly dependent upon the work function of the emitting material. The general form of Fowler-Nordheim-type (FN-type) equation can be produced as follows [9]:

$$J = \lambda_L a \varphi^{-1} F^2 EXP[\, -V_F\, b\, \varphi^{3/2} / F\,] \tag{3}$$

Equation (3) is used in all cases of field emission processes, where J is the local emission current density, a and b are the first and the second Fowler – Nordheim constants, respectively, v_F is the barrier form correction factor and it accounts for the particular shape of the potential barrier model, and λ_L is the local pre-exponential correction factor where it takes into account all of the other factors that influence the emission. The factors v_F and λ_L depend on the applied field F. In order to perform the static modeling of CNT field emitters, four field emission carbon nanotubes, which are shown in Figures 11(b)-11(e), were manufactured by Xintek, Inc. The copper anode is at the right and the CNT emitter is mounted on a tungsten wire attached to the copper cylinder at the left as shown in Figure 11(a). Figures 11(b)-11(e) show the images of CNT emitters for each carbon nanotube that were taken with a JEOL Ltd. model JEM 6300 SEM [5]. Carbon nanotubes M-1 and M-4 have a single MWCNT as the emitter, and carbon nanotubes C-3 and C-6 have a single SWCNT as the emitter. The used CNTs were formed in bundles with diameters of 10-30 nm, but in each carbon nanotube the field emission is from one CNT at the end of the bundle where the electric field is most intense.

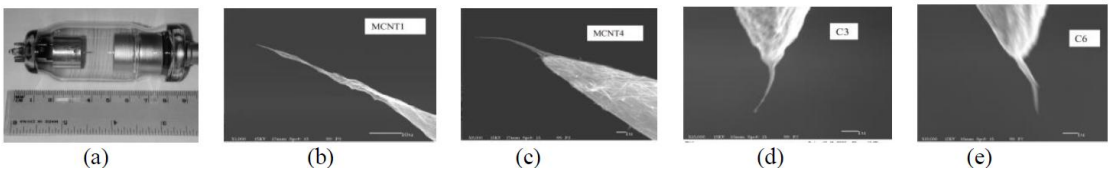


Figure 11. Carbon nanotube field emission: (a) structure of the field emission carbon nanotubes, and (b) - (e) Scanning Electron Microscopy images of the CNT emitters in the four utilized carbon nanotubes.

The DC current-voltage characteristics were measured for these four carbon nanotubes, as well as a field emitter tube from Leybold Didactic GmbH, which has an etched single crystal of tungsten as the emitter. All of the measurements that were made with the five tubes were performed at room temperature. The tungsten tip is mounted on a filament so that this tip is heated for cleaning

shortly before each session of measurements, and the data from the DC measurements were reduced by the Fowler-Nordheim analysis [5].

4.3. Carbon Field Emission-Based Two-to-One Controlled Switching

Multiplexing also known as controlled switching is considered as a highly fundamental operation in digital systems [12]. This sub-section presents carbon field emission – based controlled switching. By the utilization of the previously experimented and observed characterizations and operations of carbon field-emission from the corresponding nano-apex CNTs that were previously presented in sub-section 4.2, Figure 12 presents the carbon field emission – based primitive that realizes the two-to-one controlled switching. In Figure 12, the input control signal that is used to control the electric conduct of the device is implemented using the imposed electric field intensity (E) or equivalently the work function (Φ) or voltage (V).

The description of the operation of the carbon field emission – based device shown in Figure 12(b) is as follows: by imposing the control signal of high voltage (HV), the voltage difference between the carbon cathodes and the facing anode is varied. This change will make the carbon cathode with control signal (HV) to be field emitting while the other carbon cathode with the complementary control signal ($\overline{\text{HV}}$) to be without field emission. When the voltage difference is reversed, the carbon cathode with the complementary control signal ($\overline{\text{HV}}$) will be field emitting while the other carbon cathode with the control signal (HV) will be without field emission. Thus, this device implements the 2-to-1 controlled switching ($G = ac + bc'$) which is shown in Figure 12(a).

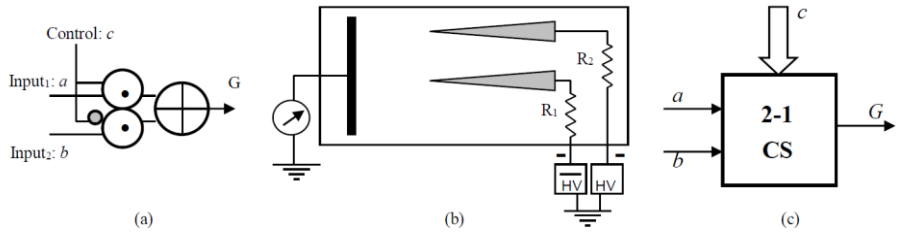


Figure 12. The carbon field emission – based device implementing the operation of the two-to-one controlled switching (CS): (a) two-to-one multiplexer ($G = ac + b\bar{c}$), (b) the carbon field emission–based two-to-one CS, and (c) block diagram for the two-to-one multiplexer.

5. NANO CIRCUIT DESIGN OF THE HOMEOMORPHIC VITERIBI ALGORITHM

This section introduces the new implementation of nano circuit design of the homeomorphic Viterbi algorithm. The functions inside the design are implemented regularly using the regular lattice networks as was shown and presented in Section 3. The multiplexing nodes in the lattice networks are then practically achieved utilizing carbon field emission-based controlled switching that was presented in Section 4. For the purpose of the design, Table 2 shows the truth tables of an non-homeomorphic half-adder, non-homeomorphic subtractor and non-homeomorphic full-adder, and Figure 13 shows the various nano circuits for the nano realization of each nano Viterbi cell in the corresponding (homeomorphic) Viterbi algorithm.

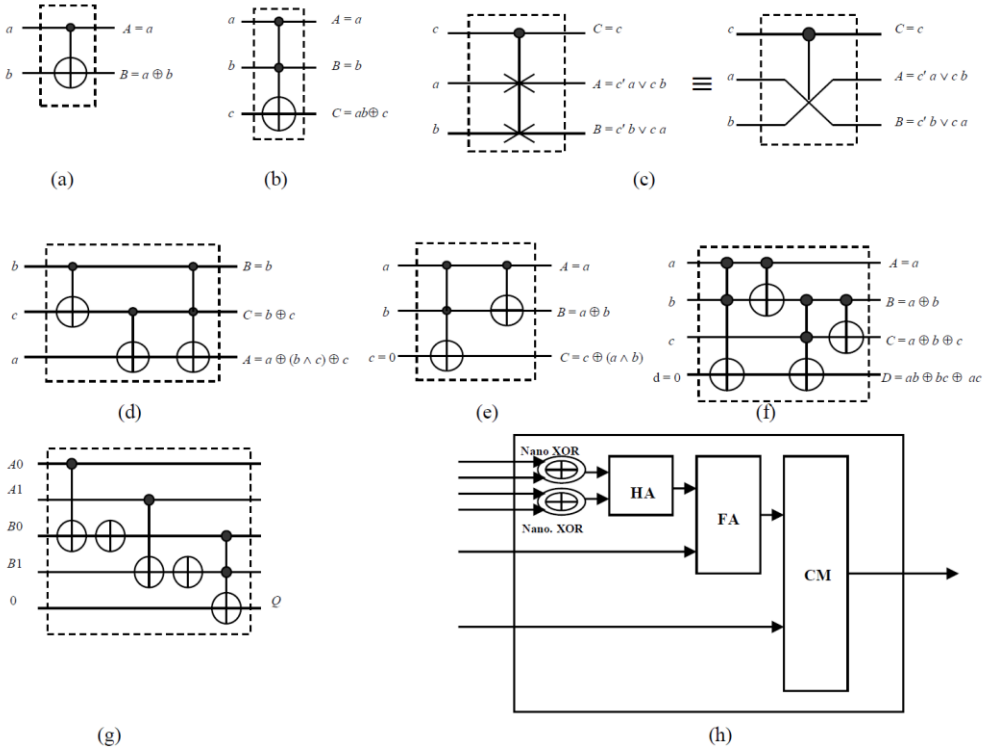
Table 2. Truth tables: (a) non-homeomorphic half-adder and subtractor and (b) non-homeomorphic full-adder.

Inputs		Half-Adder		Subtractor	
		Outputs		Outputs	
a	b	$a + b$	Carry	a - b	Borrow
0	0	0	0	0	0
0	1	1	0	1	1
1	0	1	0	1	0
1	1	0	1	0	0

(a)

Inputs			Outputs	
a	b	c_i	s	c_o
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

(b)



(a) (b) (c) (d) (e) (f) (g) (h)

Figure 13. Nano homeomorphic circuits for the nano realization of each Viterbi cell: (a) nano XOR gate (Controlled-NOT gate), (b) nano Controlled-Controlled-NOT gate, (c) nano multiplexer (Controlled-Swap gate), (d) nano subtractor, (e) nano half-adder (HA), (f) nano full-adder (FA), (g) nano equality-based comparator that compares two 2-bit numbers where an isolated XOR symbol means a nano NOT gate, and (h) basic nano homeomorphic Viterbi cell which is made of two Controlled-NOT gates, one nano HA, one nano FA and one nano comparator with multiplexing. The nano comparator can be synthesized using a nano subtractor and a multiplexer gate. The symbol $\oplus$ is logic XOR, $\wedge$ is logic AND, $\vee$ is logic OR, and $'$ is logic NOT.

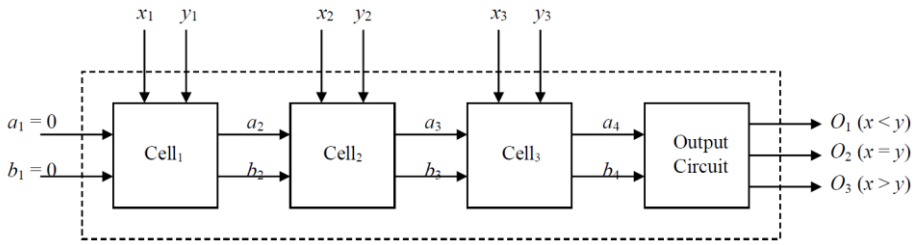


Figure 14. An iterative network to compare two 3-digit binary numbers: $X = x_1\ x_2\ x_3$ and $Y = y_1\ y_2\ y_3$.

Figure 14 shows the logic circuit design of an iterative network to compare corresponding two 3-digit binary numbers $X = x_1\ x_2\ x_3$ and $Y = y_1\ y_2\ y_3$, and Figure 15 presents the detailed synthesis of a comparator circuit which is made of a comparator cell (Figure 15a) and a comparator output circuit (Figure 15b). The extension of the circuit in Figure 14 to compare two *n*-digit binary numbers is done by utilizing *n*-cells and the same output circuit.

Figure 16 illustrates the nano circuit synthesis for the comparator cell and the output circuit (which were shown in Figure 15), and Figure 17 shows the design of a nano comparator with multiplexing where Figure 17a shows an iterative nano network to compare two 3-digit binary numbers and Figure 17c shows the complete design of the nano comparator with multiplexing. The extension of the nano circuit in Figure 17a to compare two *n*-digit binary numbers is done by utilizing *n* nano cells (from Figure 16a) and the output nano circuit (in Figure 16b).

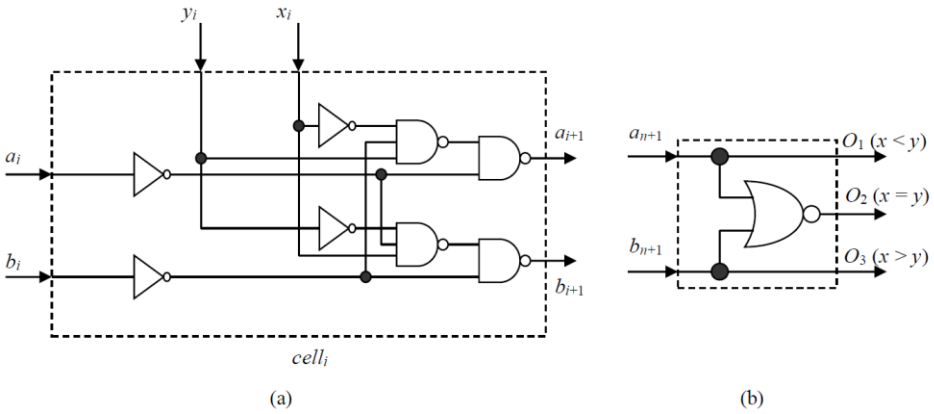


Figure 15. Designing a comparator circuit: (a) comparator cell and (b) comparator output circuit.

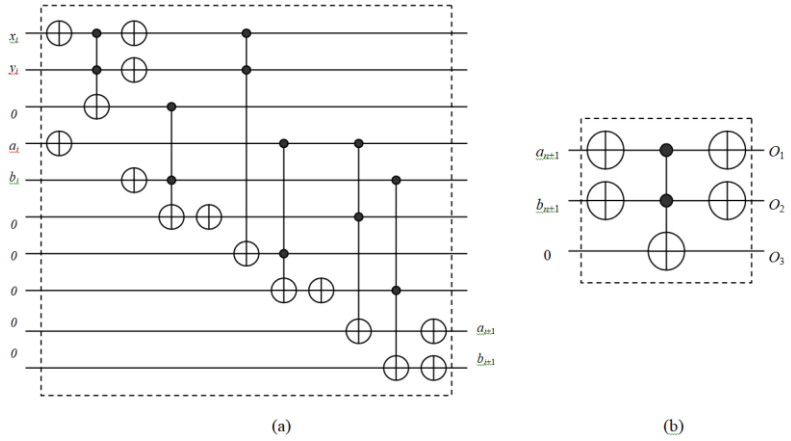


Figure 16. Nano circuit synthesis for the comparator cell and output circuit in Figure 15: (a) nano comparator cell and (b) nano comparator output circuit.

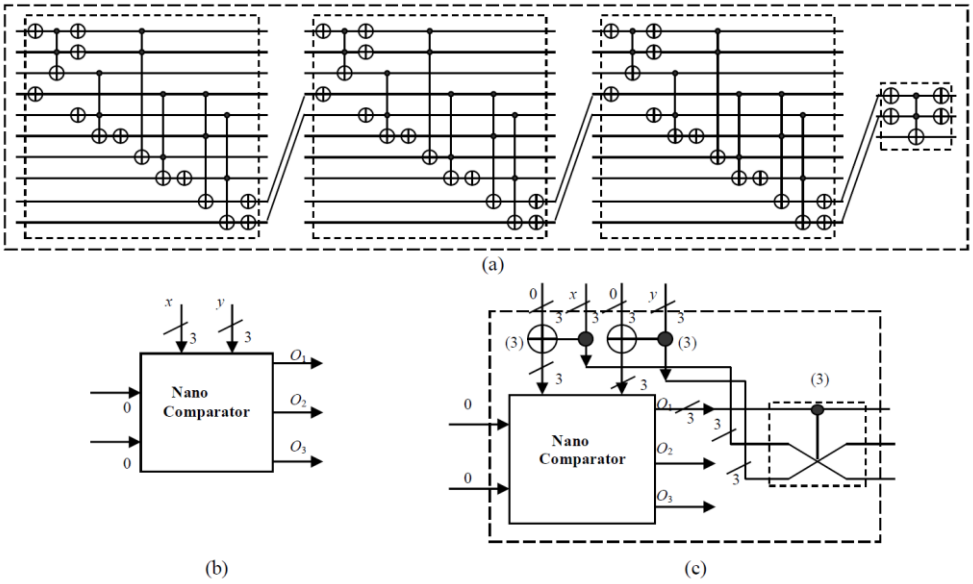


Figure 17. Designing a nano comparator with multiplexing: (a) an iterative nano network to compare two 3-digit binary numbers, (b) symbol of the nano comparator circuit in (a), and (c) complete design of nano comparator with multiplexing where the number 3 on lines indicates triple lines and (3) beside sub-circuits indicates triple circuits (i.e., three copies of each sub-circuit for the processing of the triple-input triple-output lines.)

Figure 18 shows the complete design of a nano Viterbi cell in the Viterbi algorithm that was shown in Figure 13h. The design of the nano Viterbi cell shown in Figure 18f proceeds as follows: (1) two nano circuits for the first and second lines entering the Viterbi cell each is made of two nano XORs to produce the difference between incoming received bits and trellis bits followed by nano half-adder to produce the corresponding sum (which is the Hamming distance) are shown in Figures 18a and 18b, (2) logic circuit composed of a nano half-adder and a nano full-adder that adds the current Hamming distance to the previous Hamming distance is shown in Figure 18c, (3) two nano circuits for the first and second lines entering the Viterbi cell each is synthesized according to the logic circuit in Figure 18c (which is made of a half-adder followed by a full-adder) are shown in Figures 18d and 18e, (4) nano comparator with multiplexing in the

Viterbi cell that compares the two entering metric numbers (i.e., two entering Hamming distances) and selects using the control line O_1 the path that produces the minimum entering metric (i.e., minimum entering Hamming distance) is shown in Figure 18f.

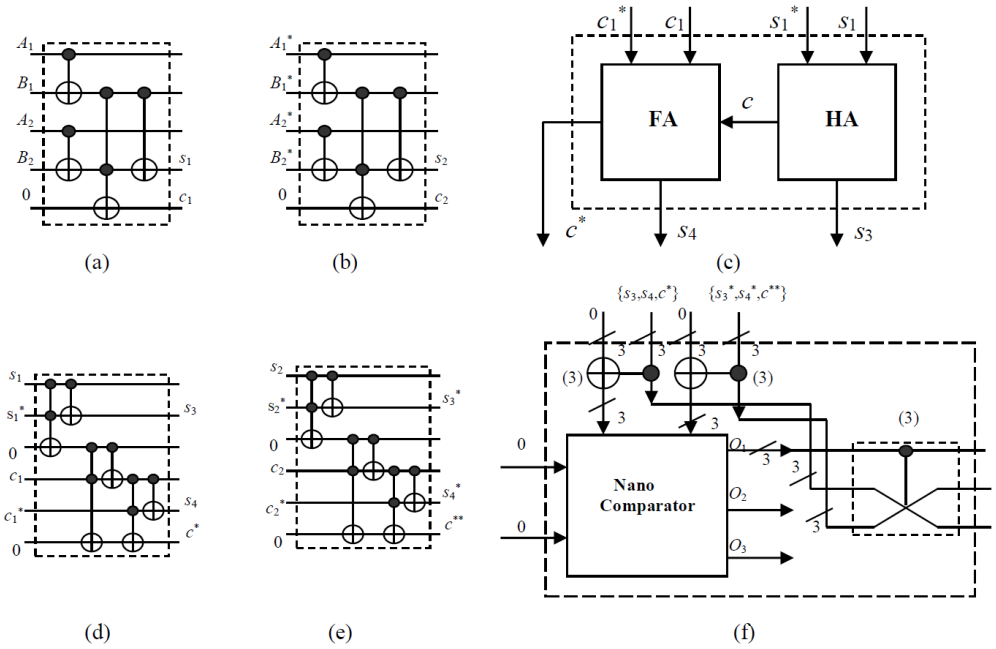


Figure 18. The complete design of a nano Viterbi cell in the Viterbi algorithm that was shown in Figure 13h: (a) nano circuit that is made of two nano XORs to produce the difference between incoming received bits ($A_1\,A_2$) and trellis bits ($B_1\,B_2$) followed by nano half-adder to produce the corresponding sum ($s_1\,c_1$) which is the Hamming distance for the first line entering the Viterbi cell, (b) nano circuit that is made of two nano XORs to produce the difference between incoming received bits ($A_1^*\,A_2^*$) and trellis bits ($B_1^*\,B_2^*$) followed by nano half-adder to produce the corresponding sum ($s_2\,c_2$) which is the Hamming distance for the second line entering the Viterbi cell, (c) logic circuit composed of nano half-adder and nano full-adder that adds the current Hamming distance to the previous Hamming distance, (d) nano circuit in the first line entering the Viterbi cell for the logic circuit in (c) that is made of a nano half-adder followed by a nano full-adder, (e) nano circuit in the second line entering the Viterbi cell for the logic circuit in (c) that is made of a nano half-adder followed by a nano full-adder, and (f) nano comparator with multiplexing in the Viterbi cell that compares the two entering metric numbers: $X = s_3\,s_4\,c^*$ and $Y = s_3^*\,s_4^*\,c^{**}$ and selects using control line O_1 the path producing the minimum metric (i.e., $X < Y$).

In Figures 18c-18e, the current Hamming metric $\{s_1, c_1\}$ for the first entering path of the Viterbi cell and the current Hamming metric $\{s_2, c_2\}$ for the second entering path of the Viterbi cell is always made of two bits (00, 01, or 10). If more than two digits (two bits) is needed to represent the previous Hamming metric for the first or second entering paths of the Viterbi cell (e.g., $(5)_{10} = (101)_2$), then extra nano full-adders are added in the logic circuit in Figure 18c and consequently in the nano circuits shown in Figures 18d-18e. Also, in the case that when the paths entering a nano Viterbi cell (state) are compared and their metrics are found to be identical then a choice is made by making a guess to choose any of the two entering paths, and this is done in the nano circuit in Figure 18f since if ($\{s_3, s_4, c^*\} < \{s_3^*, s_4^*, c^{**}\}$) then O_1 = "1" and thus chooses $X = \{s_3, s_4, c^*\}$, else O_1 = "0" and then it chooses $Y = \{s_3^*, s_4^*, c^{**}\}$ in both cases of ($\{s_3, s_4, c^*\} > \{s_3^*, s_4^*, c^{**}\}$) or ($\{s_3, s_4, c^*\} = \{s_3^*, s_4^*, c^{**}\}$).

As stated previously, this section introduced the new implementation of nano circuit design of the homeomorphic Viterbi algorithm. Each function inside the introduced design is implemented us-

ing a regular lattice network as was illustrated and presented in Section 3. This is done in either one of two ways: (1) implementing each function separately using a lattice network and then connecting single lattice networks to produce the total function, or (2) producing the total function from sub-functions utilizing logic synthesis methods [12] and then implementing the resulting total function at once using the corresponding lattice network. In either way, the multiplexing nodes in the resulting lattice network(s) are then practically achieved and implemented utilizing carbon field emission-based controlled switching that was demonstrated and presented from Section 4.

6. CONCLUSIONS

The homeomorphic design of multiple-stream Viterbi error-correcting codes using regular lattice networks and utilizing carbon field emission – based nano switching devices is introduced in this article. This is performed through the realization of operations using lattice networks that utilize the two-to-one basic controlled-switching elements, where each of these basic elements can be directly implemented using the presented carbon-based field emission devices. It has been shown that the property of homeomorphism in multiple-streams of communicated parallel data can be used for further correction of errors that are uncorrectable using the implemented decoding algorithm such as in the cases of the failure of the classical Viterbi algorithm in correcting for more than two errors. The introduced hierarchical design using the introduced nano-based lattice homeomorphic Viterbi architecture can be utilized within several applications where higher reliability, more speed and minimal power consumption are required such as in high-performance reliable low-power wireless data communications.

REFERENCES

[1] J. J. Adamék (1991) *Foundations of Coding*, Wiley, New York.

[2] A. N. Al-Rabadi (2004) *Reversible Logic Synthesis*: *From Fundamentals to Quantum Computing*, Springer-Verlag.

[3] A. N. Al-Rabadi (2009) "Reversible Viterbi algorithm and its closed-system Q-domain circuit design and computation," *J. Circuits, Systems, and Computers*, World Scientific, Singapore, Vol. 18, No. 8, pp. 1627 – 1649.

[4] A. N. Al-Rabadi (2009) *Carbon NanoTube* (*CNT*) *Multiplexers, Circuits, and Actuators*, United States Patent and Trademark Office, Patent No. US 7,508,039 B2, U.S.A.

[5] A. N. Al-Rabadi (2020) "Concurrency within ternary Galois processing of highly-regular 3D networks via controlled nano switching," *Int. Journal of Computer Science & Information Technology*, Vol 12, No. 1, pp. 1-23.

[6] G. Amaratunga (2003) "Watching the nanotube," *IEEE Spectrum*, pp. 28-32.

[7] C. Bennett (1973) "Logical reversibility of computation," *IBM J. of Research and Development,* 17, pp. 525 - 532.

[8] V. Derycke, R. Martel, J. Appenzeller, and P. Avouris (2001) "Carbon nanotube inter- and intramolecular logic gates," *Nano Letters*, Vol. 0, No. 0, A – D.

[9] R. G. Forbes (2012) "Extraction of emission parameters for large-area field emitters using a technically complete Fowler–Nordheim-type equation," *Nanotechnology*, IOP Publishing, 23(9).

[10] G. D. Forney, Jr. (1973) "The Viterbi algorithm," *Proceedings of the IEEE,* 61 (3), pp. 268—278.

[11] F. J. MacWilliams and N. J. A. Sloane (1977) *The Theory of Error-Correcting Codes*, North-Holland, Amsterdam.

[12] M. M. Mano and C. R. Kime (2008) *Logic and Computer Design Fundamentals*, 4th edition, Prentice-Hall.

[13] T. S. Rappaport (1996) *Wireless Communications*: *Principles and Practice*, IEEE Press, Piscataway, N. J.