Quest Journals
Journal of Electronics and Communication Engineering Research
ISSN:2321-5941 Volume1 ~ Issue 4 (2013) pp: 01-14
www.questjournals.org

Research Paper

Flexible Decoder Based on MAP-Algorithm for Two Different Families of Codes

Ahmed Refaey Hussein and Weikun Hou

Department of Electrical and Computer Engineering University of Western Ontario, N6A 5B9, London, Canada.

Received 16 September, 2013; **A**ccepted 30 December, 2013© The author(s) 2013. Published with open access at **www.questjournal.org**

Many wireless communication systems such as IS-54, EDGE, WiMAX, and LTE have adopted turbo codes and tail-biting convolutional codes as the forward error correcting codes (FEC) scheme for data and overhead channels. However, some releases propose LDPC codes for error-corrections due to the relative complexity of turbo codes decoder implementations as well as the success of LDPC codes in achieving the same performance as turbo codes and in some cases surpassing it with low decoding complexity. As a matter of fact, these new standard releases will work side by side in actual devices with older ones which are based on turbo and convolutional codes. Indeed, these two families of codes are both very good in performance. Consequently, it is mandatory to relate them so as to enhance technology transfer and hybridization between the two methodologies.

Moreover, in wireless systems, power consumption as well as die area must be minimized, because of battery life for wireless mobile devices, maximum allowable heat dissipation and available packaging space due to small-size hand-helds. Thus, efficient design of flexible convolutional, turbo and LDPC codes decoder with respect to power, area, and throughput is critical for future wireless system implementations. In the literature, many efficient turbo decoder architectures based on the MAP algorithm have been extensively investigated by many researchers. However, designing a flexible decoder to support LDPC, convolutional and turbo codes based on a unified MAP algorithm is very challenging. Henceforth,, in this paper we aim to provide an alternative solution proposing a combined decoder supporting the three types of codes with a small additional overhead based on a unified MAP algorithm.

I. INTRODUCTION

As a matter of fact, a few researchers have tried to treat LDPC codes as turbo codes and apply a turbo-like message-passing algorithm to LDPC codes. For example, in [1], Mansour and Shanbhag introduced an efficient turbo message passing algorithm for architecture-aware LDPC codes. One year later in [2], Hocevar was able to propose a layered decoding algorithm which treats the parity-check matrix as horizontal layers and passes the soft information between layers to improve the performance. Subsequently, Chakrabarti and Zhu demonstrated the super-code-based LDPC construction and decoding [3]. Furthermore, Zhang and Fossorier in [4] evoked an attention to the shuffled BP algorithm to achieve faster decoding convergence. As well, Lu and Moura in [5] sectionalized the Tanner graph into several trees, then they were able to apply the turbo-like decoding algorithm in each tree for faster convergence rate. What is more, Dai et al. introduced a turbo-sum-product hybrid decoding algorithm for quasi-cyclic (QC) LDPC codes by splitting the parity-check matrix into two submatrices where the information is exchanged [6].

Eventually, in [7], they have extended a super-code-based decoding algorithm for LDPC codes and presented a more generic message passing algorithm for LDPC and turbo decoding. Therein, a code is divided into multiple super-codes and then the decoding operation is performed by iteratively exchanging the soft information between super codes. Moreover, they exploit commonalities between LDPC and turbo decoders.

Herein, we propose two approaches to decode the two families of codes by means of the MAP algorithm. Given that architecturally the MAP algorithm is a trellis-based algorithm, we first propose the regular trellis diagram representation for the **H** matrix of LDPC codes, thus exhibiting the complexity problem and proposing a solution. Second, we consider a super-code based decoding algorithm for LDPC codes as an alternative solution, showing the applicability of this approach to introduce a new turbo decoder. Finally, we propose a general

Department of Electrical and Computer Engineering University of Western Ontario, N6A 5B9, London, Canada.

architecture for a combined decoder of the three mentioned codes based on the approach previously introduced for turbo and convolutional codes in UMTS.

1. Trellis Representation of LDPC Codes

In 1974, Bahl, Coke and Raviv described a technique for constructing trellises for block codes, uncovering an important relation between block codes and convolutional codes [8]. In fact, this work did not present an algorithm, but merely showed that a trellis can be constructed. In 1978, Wolf developed a similar trellis [9], which provides a basis for the maximum likelihood (ML) soft-decision decoding of block codes using the MAP algorithm [8]. Wolf's method yields a trellis which represents a superset of codewords which must be modified by deletion of branches and states in order to obtain the trellis for the code.
A trellis definition and method for constructing a trellis for any LDPC code, which is conceptually the same as linear block codes, from its parity-check matrix will be presented in the following subsections. Once the trellis has been constructed, the well-known MAP algorithm can be applied. The trellis complexity of the decoding algorithm will then be evaluated.

II. TRELLIS CONSTRUCTION

There are several ways a generic (n,k) linear block code C can be represented by a trellis structure (whereas a binary LDPC code is a linear block code specified by a very sparse binary parity check matrix). However, the original construction introduced by Bahl et al. [8] yields a trellis which uniquely minimizes both the edge count and the vertex count at each depth. Because of this dual minimal characteristic, the so-called BCJR trellis offers significant advantages over other trellises for the use of the Viterbi or MAP algorithms [10].

We introduce the construction of the BCJR trellis through a simplified example as **follows:**

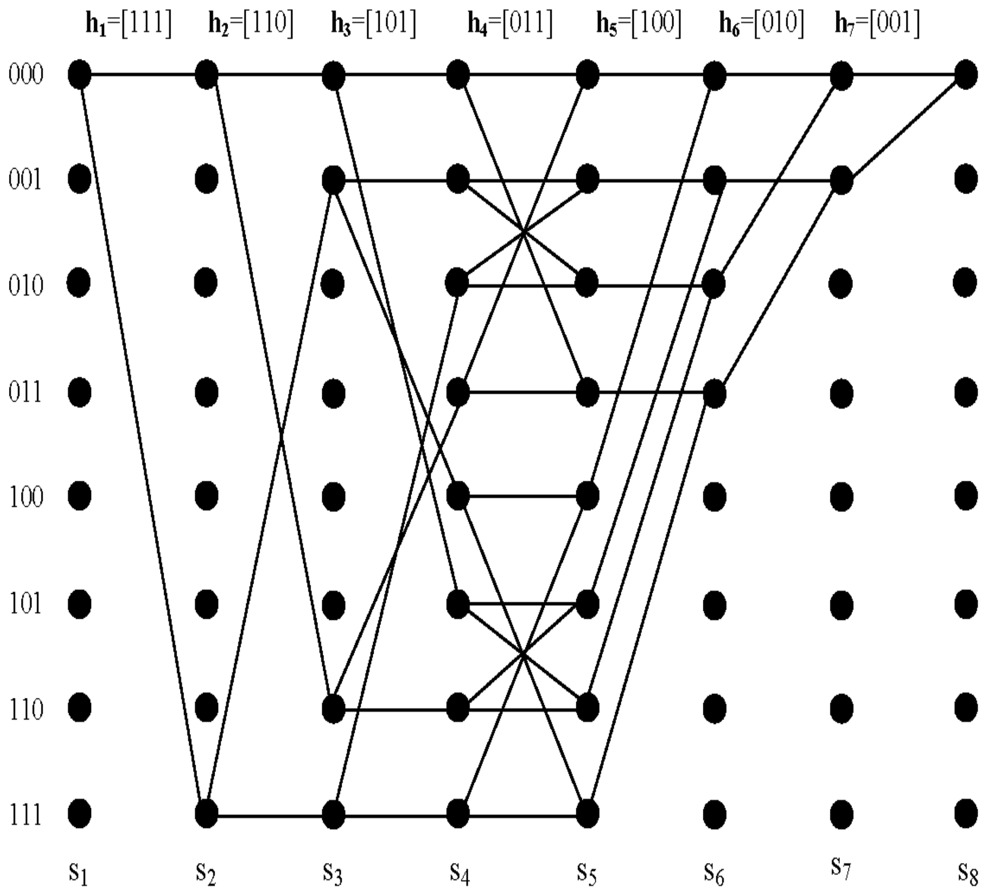

Figure 1: The trellis of a (7, 3) binary linear block code.

Example 1: We demonstrate the trellis idea with a (7, 3) binary linear block code. Let

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \mathbf{h}_1 & \mathbf{h}_2 & \mathbf{h}_3 & \mathbf{h}_4 & \mathbf{h}_5 & \mathbf{h}_6 & \mathbf{h}_7 \end{bmatrix}, \qquad (1)$$

be the parity-check matrix of the code. Accordingly, a column vector $\mathbf{x}$ is a codeword if and only if $\mathbf{s} = \mathbf{Hx} = \mathbf{0}$; that is, the syndrome $\mathbf{s}$ must satisfy

$$\mathbf{s} = \begin{bmatrix} \mathbf{h}_1 & \mathbf{h}_2 & \mathbf{h}_3 & \mathbf{h}_4 & \mathbf{h}_5 & \mathbf{h}_6 & \mathbf{h}_7 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix}$$

$$= \sum_{t=1}^{7} \mathbf{h}_t x_i = \mathbf{0}. \qquad (2)$$

We define the partial syndrome by

$$\mathbf{s}_{r+1} = \sum_{t=1}^{r} \mathbf{h}_t x_t = \mathbf{s}_r + \mathbf{h}_r x_r, \qquad (3)$$

with $\mathbf{s}_1 = 0$. As a consequence $\mathbf{s}_{n+1} = \mathbf{s}$.

A trellis representation of a code is obtained by using $\mathbf{s}_r$ as the state, with an edge between a state $\mathbf{s}_r$ and $\mathbf{s}_{r+1}$ if $\mathbf{s}_{r+1} = \mathbf{s}_r$ (corresponding to $x_r = 0$) or if $\mathbf{s}_{r+l} = \mathbf{s}_r + \mathbf{h}_r$ (corresponding to $x_r = 1$). Furthermore, the trellis is terminated at state $\mathbf{s}_{n+1} = \mathbf{0}$, corresponding to the fact that a valid codeword has a syndrome of zero. The trellis has at most 2^{n-k} states in it. Figure 1 shows the trellis for the above parity-check matrix. Horizontal transitions correspond to $x_t = 0$ and diagonal transitions correspond to $x_t = 1$. Only those paths which end up at $\mathbf{s}_8 = \mathbf{0}$ are retained [8].

2.2 Trellis Complexity

The complexity of a trellis-based decoder for a general block code C follows almost immediately from the complexity of the trellis representation upon which it is based. In [9], the trellis complexity grows exponentially with min $(k, n-k)$. In general, trellis complexity can be quantified using a number of different metrics, the dimension profile of the vertex space and the number of edges in the trellis being particularly popular in the literature [10]. Termed edge complexity, the latter has been defined as:

$$E(C) = \sum_{t} q^{b_t} \qquad (4)$$

where b_t is the dimension of the branch space at time t, and q is the size of the code alphabet. McEliece in [10] has shown that the total number of edges in the trellis is a better reflection of the number of operations required to execute a software-based implementation of the Viterbi and MAP algorithms than any other known measure of trellis complexity.

2. Problem and Proposed Solution

Unfortunately, the implementation of the MAP algorithm requires large computation and storage. Furthermore, its forward and backward recursions result in long decoding delays. Practically, this decoding algorithm must be simplified and its decoding complexity and delay must be reduced. As a consequence, for a general LDPC code, the trellis structure is sufficiently complicated that it may be very difficult to represent it efficiently in hardware as its very large generator matrix requires large computation and storage. There has been recent work on the MAP algorithm and its variations, such as Log-MAP and Max-Log-MAP algorithms based on sectionalized trellises for linear block codes [11]. Using the structural properties of properly sectionalized trellises, the decoding complexity and delay of the MAP algorithm can be reduced. Therefore, first we propose to apply the MAP algorithm based on bit-level trellises (minimal trellis) [12], also called the recursive MAP algorithm. In addition, several works have discussed the MAP algorithm based on sectionalized trellises which may also reduce the implementation complexity. As a consequence, in the following sections we show the two directions in details to find an optimal trade-off between performance and complexity.

III. BIT-LEVEL TRELLIS

As mentioned before, we will use a binary (n,k) linear block code C to be our simplified model as our way to generalize this work to LDPC codes. This linear block code can be represented by an n-section trellis diagram (bit-level trellis), denoted T, over the time span $\Re = \{0,1,2, \ldots, n\}$ which displays the dynamic behavior of its encoder.

The trellis T is a directed graph consisting of $n+1$ levels of nodes (called states) and edges (called branches) such that:

• State space at time t, denoted $\sum_t(C)$, is the set of states of the trellis T at time t (i.e., at the end of the t-th level). For $0 \le t \le n$, the nodes at the t-th level represent the states in the state-space $\sum_t(C)$. At time 0 (or 0-th level), there is only one node, denoted σ_0, called the initial node (or state). At time n (or ($n+1$)-th level), there is only one node, denoted σ_f, called the final node (or state).

• For $0 \le t \le n$, a branch in the t-th section of the trellis T connects a state $\sigma_{t-1} \in \sum_{t-1}(C)$ to a state $\sigma_t \in \sum_t(C)$ and has a label y_{t-1} that represents the encoder output code bit in the interval from time $(t-1)$ to time t. A branch represents a state transition.

• Except for the initial and final states, every state has at least one, but no more than two, incoming branches and at least one, but no more than two, outgoing branches. The initial state has two outgoing branches but no incoming branches. The final state has two incoming branches but no outgoing branches. Two branches diverging from the same state have different labels.

• There is a directed path from the initial state σ_0 to the final state σ_f with a label sequence $(y_0, y_2, \ldots, y_{n-1})$ if and only if $(y_0, y_2, \ldots, y_{n-1})$ is a codeword in C.

A trellis diagram for C is said to be minimal if the number of states at each time stage of the trellis (or at the end of each section) is minimum [13].

Example 2: We demonstrate the idea here by considering the (8, 4) binary linear block code. The 8-section trellis diagram is shown in Figure 2.

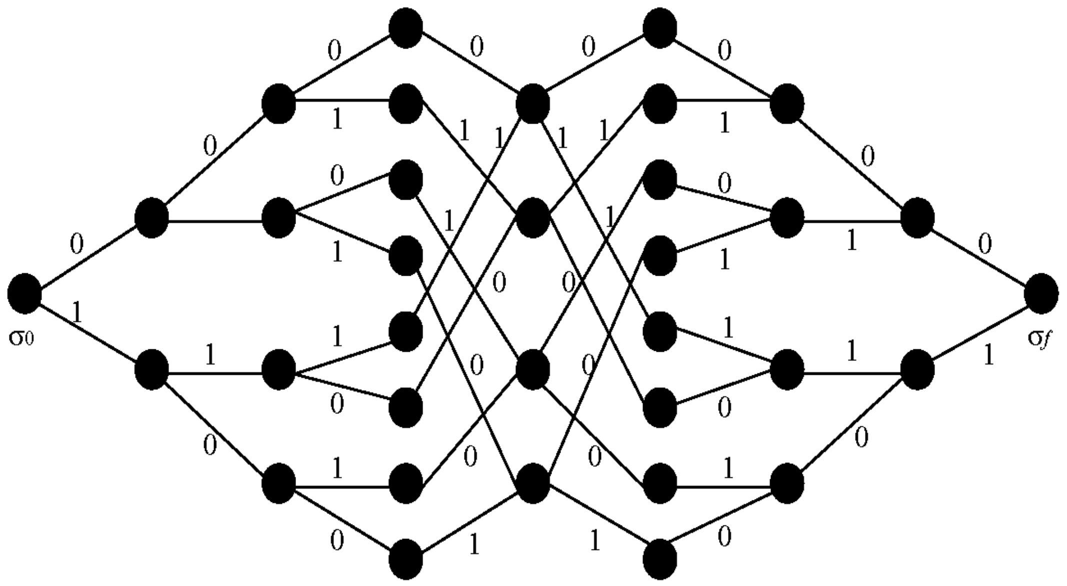

Figure 2: 8-section minimal trellis diagram for the (8, 4) binary linear block code.

For $0 \le t \le n$, let $\left|\sum_t\right|$ denote the cardinality of the state-space $\sum_t$. Then, the sequence, $\left(\left|\sum_0\right|, \left|\sum_1\right|, \ldots, \left|\sum_{N-1}\right|, \left|\sum_N\right|\right)$, is called the state-space profile which is a measure of state complexity of the n -section code trellis T. Define

$$\sigma_t = \log_2 \left|\sum_t (C)\right|, \qquad (5)$$

which is called the state-space dimension at time t. The sequence $(\sigma_0, \sigma_1, \ldots, \sigma_n)$ is called the state-space dimension profile.

We see that the state-space and state-space dimension profiles are $(1,2,4,8,4,8,4,2,1)$ and $(0,1,2,3,2,3,2,1,0)$, respectively. To facilitate the code trellis construction, we arrange the parity-check matrix H in a special form. Let $\mathbf{h} = (h_0, h_2, \ldots, h_{n-1})$ be a nonzero binary n -tuple. The first nonzero component of $\mathbf{h}$ is called the leading '1' of $\mathbf{h}$ and the last nonzero component of $\mathbf{h}$ is called the trailing '1' of $\mathbf{h}$. For example, the leading and trailing ones of the 8-tuple $(0,1,0,1,0,1,0,1,0)$ are h_1 and h_6 respectively. A parity-check matrix H for C is said to be in trellis oriented form (TOF) if the following two conditions hold:

1. The leading "1" of each row of H appears in a column before the leading "1"of any row below it.
2. No two rows have their trailing "ones" in the same column.

Any parity-check matrix for C can be put in TOF by two steps of Gaussian elimination (or elementary row operations). It should be noted that a parity-check matrix in TOF is not necessarily in systematic form and that fact will prove useful when we use different LDPC codes.

Example 3: Considering the same (8, 4) binary linear block code with the following parity check matrix,

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}. \qquad (6)$$

To obtain the **H** matrix in TOF, we interchange the second and fourth rows as follows:

$$\mathbf{H}' = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}. \qquad (7)$$

Add the fourth row of the above matrix to the first, second and third rows. These additions result in the following matrix in TOF:

$$\mathbf{H}_{TOHM} = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}, \qquad (8)$$

Let $\mathbf{h} = (\mathbf{h}_0, \mathbf{h}_1, \ldots, \mathbf{h}_{n-1})$ be a row in the trellis-oriented parity-check matrix (TOHM) $\mathbf{H}_{TOHM}$ for code C. Let $\omega(h) = \{t, t+1, \ldots, f\}$ denote the smallest index interval which contains all the nonzero components of $\mathbf{h}$. This says that $\mathbf{h}_t = 1$ and $\mathbf{h}_f = 1$ and they are leading and trailing "ones" of $\mathbf{h}$, respectively. This interval $\omega(\mathbf{h}) = \{t, t+1, \ldots, f\} \cong [t, f]$, called the bit span of $\mathbf{h}\omega(\mathbf{h})$, occupies the time span from time t to time $(f+1)$, i.e., $\{t, t+1, \ldots, f+1\}$. In terms of time, we define the time span of $\mathbf{h}$, denoted $\tau(\mathbf{h})$, as the following time interval,

$$\tau(\mathbf{h}) = t, t+1, \ldots, f+1 \cong [t, f+1]. \qquad (9)$$

Then, we define the active time span of h, denoted $\tau_\alpha(h)$, as the time interval

$$\tau_\alpha(h) = \begin{cases} [t+1, f], & \text{for } f >> t, \\ \omega(\text{empty set}), & \text{for } f = t. \end{cases} \qquad (10)$$

Let $(\mathbf{h}_0, \mathbf{h}_2, \ldots, \mathbf{h}_{k-1})$ be the k rows of a TOHM $\mathbf{H}_{TOHM}$ for an (n,k) linear code C with

$$\mathbf{h}_l = (\mathbf{h}_{l,0}, \mathbf{h}_{l,1}, \ldots, \mathbf{h}_{l,n-1}), \qquad (11)$$

for $0 \le l \le k$. Let $x = (x_0, x_1, \ldots, x_{k-1})$ be the message of k information bits, encoded to the codeword $y = (y_0, y_1, \ldots, y_{n-1})$. Whereas $y = xh$, then we can rewrite y as $y = (x_0.\mathbf{h}_0, y_1.\mathbf{h}_1, \ldots)$. Based on the dimensions of **H**, there will be k parity-check constraints for the length-n codeword.

We see that the l-th information bit x_l affects the output codeword y of the encoder over the bit-span $\omega(h)$ of the l-th row $\mathbf{h}_1$ of the TOHM $\mathbf{H}_{TOHM}$, where $\omega(h) = [t, f]$. Then, the information x_l affects the output code bits $(y_t, y_{t+1}, \ldots, y_f)$ from time t to time $(f+1)$. At time t, the input information bit x_l is regarded as the current input. At time $(t+1)$, x_l is shifted into the encoder memory and remains in the memory for $f-t$ units of time. At time $(f+1)$, it will have no effect on future output code bits of the encoder, and it is shifted out of the encoder memory. At time t, $0 \le t \le n$, we divide the rows of $\mathbf{H}_{TOHM}$ into three disjoint subsets:

1. $\mathbf{H}_t^p$ consists of those rows of $\mathbf{H}_{TOHM}$ whose bit spans are contained in the interval $[0, t-1]$.
2. $\mathbf{H}_t^f$ consists of those rows of $\mathbf{H}_{TOHM}$ whose bit spans are contained in the interval $[t, n-1]$.
3. $\mathbf{H}_t^s$ consists of those rows of $\mathbf{H}_{TOHM}$ whose active time spans contain time t.

Let $\mathbf{X}_t^p$, $\mathbf{X}_t^f$, and $\mathbf{X}_t^s$ denote the subsets of information bits that correspond to the rows of $\mathbf{H}_t^f$, $\mathbf{H}_t^s$, and $\mathbf{H}_t^p$, respectively. The information bits in $\mathbf{X}_t^p$ do not affect the encoder outputs after time t, and hence they become the past with respect to time t. The information bits in $\mathbf{X}_t^f$ only affect the encoder outputs after time t. Since the active time spans of the rows in $\mathbf{H}_t^s$ contain the time instant t, the information bits in $\mathbf{H}_t^s$ affect not only the past encoder outputs up to time t, but also the future encoder outputs beyond time t. Therefore, the bits in $\mathbf{H}_t^s$ are the information bits stored in the encoder memory that affect the current output code bit y_t and the future output code bits beyond time t. These information bits in $\mathbf{H}_t^s$ hence define a state of the encoder for the code C at time t. Considering $\sigma_t = \left|\mathbf{X}_t^s\right| = \left|\mathbf{H}_t^s\right|$, then there are 2^{σ_t} distinct states that the encoder can reside in at time t. Each state is defined by a specific combination of the σ_t information bits in $\mathbf{X}_t^s$. These states form the state-space $\sum_t(C)$ of the encoder (or simply of the code C).

Hence, the parameter σ_t is the dimension of the state-space $\sum_t(C)$. In the trellis representation of C, the states in $\sum_t(C)$ are represented by 2^{σ_t} nodes at the t-th level of the trellis. The set $\mathbf{X}_t^s$ is called the state-defining information set at time t. From the above analysis, we see that at time t for $0 \le t \le n$, the dimension σ_t of the state-space $\sum_t(C)$ is simply equal to the number of rows in the TOHM $\mathbf{H}_{TOHM}$ of C whose active time spans contains time t. These rows will henceforth be designated as active at time t. Therefore, from the TOHM $\mathbf{H}_{TOHM}$, we can easily determine the state-space dimension profile $(\sigma_0, \sigma_1, \ldots, \sigma_n)$ by simply counting the number of rows in $\mathbf{H}_{TOHM}$ which are active at time t for $0 \le t \le n$. For $0 \le t \le n$, suppose the encoder is in state $\sigma_t \in \sum_t(C)$. From time t to time $t+1$, the encoder generates a code bit y_t and moves from state σ_t to a state $\sigma_{t+1} \in \sum_t(C)$. Let

$$\mathbf{H}_t^s = h_1^t, h_2^t, \ldots, h_{\sigma_t}^t, \qquad (12)$$

and

$$\mathbf{X}_t^s = x_1^t, x_2^t, \ldots, x_{\sigma_t}^t, \qquad (13)$$

where $\sigma_t = \left|\mathbf{X}_t^s\right| = \left|\mathbf{H}_t^s\right|$. The current state σ_t of the encoder is defined by a specific combination of the information bits in $\mathbf{X}_t^s$. Let h^* denote the row in $\mathbf{H}_t^f$ whose leading "1" is a bit position t. The uniqueness of this row $\mathbf{h}^*$ (if it exists) is guaranteed by the first condition in the definition of a generator matrix in TOF. Let $\mathbf{h}_i^*$ denote the i-th component of $\mathbf{h}^*$. Then $h_i^* = 1$. Let x^* denote the information bit that corresponds to row $\mathbf{h}^*$. Given that $y = (x_0.h_0, y_1.h_1, \ldots)$, the bit interval between time t and time $t+1$ is given by

$$\mathbf{y}_t = x^* + \sum_{l=1}^{\sigma_t} x_l^{(t)}.h_{l,t}^{(t)}, \qquad (14)$$

where $h_{l,t}^{(t)}$ is the t-th component of $h_l^{(t)}$ in $\mathbf{H}_t^s$. It should be noted that the information bit x^* begins to affect the output of the encoder at time t. For this reason, bit x^* is regarded as the current input information bit. The second term in (14) is the contribution from the state σ_t defined by the information bits in $\mathbf{X}_t^s = \left\{x_1^t, x_2^t, \ldots, x_{\sigma_t}^t\right\}$, which are stored in memory. From (14), we see that the current output $\mathbf{y}_t$ is uniquely determined by the current state σ_t of the encoder and the current input information bit x^*. The output code bit $\mathbf{y}_t$ can have two possible values depending on the current input information bit x^*. Each value takes the encoder to a different state at time $t+1$. That is, there are two possible transitions from the current state σ_t to two states in $\sum_{t+1}(C)$ at time $t+1$. In the code trellis T for C, there are two branches diverging from the node σ_t labeled with "0" and "1", respectively. Suppose there is no such row h^* in $\mathbf{H}_t^s$. Then, the output code bit $\mathbf{y}_t$ at time t is given by:

$$\mathbf{y}_t = \sum_{l=1}^{\sigma_t} x_l^{(t)}.h_{l,t}^{(t)}. \qquad (15)$$

In this case, we may regard the current input information x^* as being set to "0", i.e. $x^* = 0$ (this is called a dummy information bit). The output code bit $\mathbf{y}_t$ can take only one value given by (15) and there is only one possible transition from the current state σ_t to a state in $\sum_{t+1}(C)$. In the trellis T, there is only one branch diverging from the node σ_t. Thus for, we have formulated the state-space and determined the output function of a linear block code encoder. Next, we need to determine the state transition of the encoder from one time instant to another. Let $\mathbf{h}^0$ denote the row in $\mathbf{H}_t^s$ whose trailing "1" is at bit position t, i.e. the t-th component h_t^0 of $\mathbf{h}^0$ is the last nonzero component of $\mathbf{h}^0$. Note that this row $\mathbf{h}^0$ may not exist. The uniqueness of the row $\mathbf{h}^0$ (if it exists) is guaranteed by the second condition of a parity-check matrix in TOF. Let x^* be the information bit in $\mathbf{H}_t^s$ that corresponds to row $\mathbf{h}^0$. Then at time $t+1$,

$$\mathbf{H}_{t+1}^s = \left(\mathbf{H}_t^s \ \{h^0\}\right) \bigcup \{h^*\}, \qquad (16)$$

and

$$\mathbf{X}_{t+1}^s = \left(\mathbf{X}_t^s \ \{x^0\}\right) \bigcup \{a^*\}. \qquad (17)$$

The information bits in $\mathbf{X}_{t+1}^s$ define the state-space $\sum_{t+1}(C)$ at time $t+1$. The change from $\mathbf{X}_t^s$ to $\mathbf{X}_{t+1}^s$ defines the state transition of the encoder from the current state σ_t defined by $\mathbf{X}_t^s$ to the next state σ_{t+1} defined by $\mathbf{X}_{t+1}^s$. We consider the information bit x^0 as the oldest information bit stored in the encoder memory at time t. As the encoder moves from time t to time $t+1$, x^0 is shifted out of the memory and x^* (if it exists) is shifted into the memory. The state defining sets $\mathbf{X}_t^s$, $\mathbf{X}_{t+1}^s$ and the output functions given by (14) and (15) provide us all the information needed to construct the n-section bit-level trellis diagram T for the (n,k) code C.

The construction of the n-section trellis T is carried out serially, section by section. Suppose the trellis has been constructed up to section t (i.e. from time 0 to time t). Now, we want to construct the $(t+1)$-th section from time t to time $(t+1)$. The state-space $\sum_{t+1}(C)$ is known. The $(t+1)$-th section is constructed by taking the following steps:

1. Determine $\mathbf{H}_{t+1}^s$ and $\mathbf{X}_{t+1}^s$ from (16) and (17), from the state-space $\sum_{t+1}(C)$ at time $(t+1)$.
2. For each state $\sigma_t \in \sum_t(C)$, determine its state transition(s) based on the change from $\mathbf{X}_t^s$ to $\mathbf{X}_{t+1}^s$. Connect σ_t to its adjacent state(s) in $\sum_{t+1}(C)$ by branches.
3. For each state transition, determine the output code bit $\mathbf{y}_t$ from the output function of (14) or (15), and label the corresponding branch in the trellis with $\mathbf{y}_t$.

The bit-level trellis T for C constructed based on TOHM, $\mathbf{H}_{TOHM}$ is a minimal trellis [14].

3.2 Sectionalized Trellises

For any positive integer v such that $1 \le v \le n$, the bit-level trellis T can be sectionalized into v sections with section boundary locations in $U = \{u_0, u_1, ..., u_v\}$ where $0 = u_0 < u_1 < \ldots < u_v = n$. The sectionalization can be done by:

1. Deleting every state in $\sum_t(C)$ for $t \in \{0,1,\ldots,n\}/U$ and every branch to or from a deleted state.
2. For $1 \le v \le n$, connecting a state $\sigma' \in \sum_{u_{f-1}}(C)$ to a state $\sigma \in \sum_{u_f}(C)$ by a branch with label $l(\sigma', \sigma)$ IFF there is a path with label $l(\sigma', \sigma)$ from σ' to σ in T.

This sectionalization results in a v-section trellis $T(U)$. At boundary location u_f, the state-space (the set of states) is denoted $\sum_{u_f}(C)$. The b-th section of $T(U)$, denoted T_f, consists of the state-space $\sum_{u_{f-1}}(C)$ at time u_{f-1}, the state $\sum_{u_f}(C)$ at time u_b and the branches that connect the states in $\sum_{u_{f-1}}(C)$ and the states in $\sum_{u_f}(C)$. A branch in this section represents $m_f = u_f - u_{f-1}$ code bits. The length of the b-th section is m_b. If the lengths of all the sections of an v-section code trellis $T(U)$ are the same, $T(U)$ is said to be uniformly sectionalized. Two adjacent states σ' and σ with $\sigma' \in \sum_{u_{f-1}}(C)$ and $\sigma \in \sum_{u_f}(C)$ may be connected by multiple branches, called parallel branches. For convenience, we say that these parallel branches form a composite branch, denoted $L(\sigma', \sigma)$. Each branch $F(\sigma', \sigma) \in L(\sigma', \sigma)$ is labeled by an m_f-tuple, $l(\sigma', \sigma) = (v_{u_{f-1}+1}, v_{u_{f-1}+2}, \ldots, v_{u_f})$, where $v_{u_{f-1}+t} = \pm 1$ for BPSK signaling with unit energy. Let $\sigma_u = \log_2 \left| \sum_{u_f} \right|$ be the dimension of the state-space $\sum_{u_f}$ at time u_f. Then, $(\sigma_0, \sigma_{u_1}, \sigma_{u_2}, \ldots; \sigma_{u_{v-1}}, \sigma_n)$, is the state-space dimension profile of the v-section code trellis $T(U)$ with section boundary set $U = \{0, u_1, u_2, ..., u_{v-1}, n\}$. If we choose the section boundaries, $U = \{u_0, u_1, u_2, ..., u_v\}$ at the places where $(\sigma_0, \sigma_{u_1}, \sigma_{u_2}, \ldots; \sigma_{u_{v-1}}, \sigma_n)$ are small, then the resultant v-section code trellis $T(U)$ has a small state-space dimension profile. The maximum state complexity is

$$\sigma_{v,\max}(C) = \max_{0 \le f \le v} \sigma_{u_f} \qquad (18)$$

In the implementation of a trellis-based decoding algorithm, such as the MAP algorithm, a proper choice of the section boundary locations results in a significant reduction in decoding complexity.

Example 4: Considering the same (8, 4) binary linear block code whose 8-section code trellis is shown in Figure 2, suppose we choose $v = 4$ and the section boundary set $U = \{0,2,4,6,8\}$.
We obtain a uniform 4-section code trellis as shown in Figure 3.

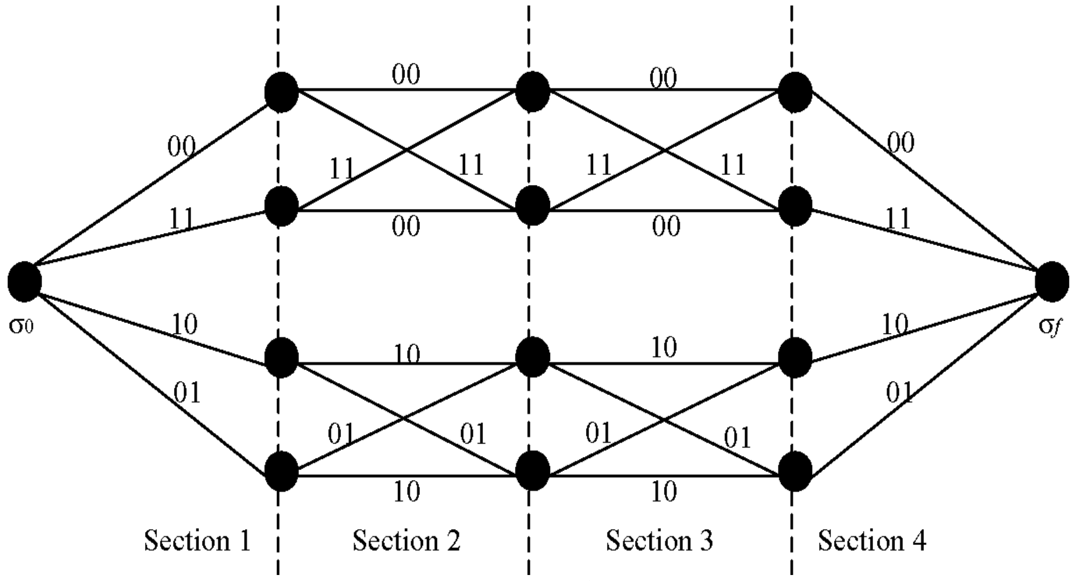

Figure 3: 4-section minimal trellis diagram for the (8, 4) binary linear block code.

The state-space dimension profile for this 4-section code trellis is (0,2,2,2,0) and the maximum state-space dimension is $\sigma_{4,\max} = 2$. It is a 4-section 4-state code trellis.

3.3 MAP Algorithm Based on Mackay's Approach

As suggested by Mackay in [15], the check-to-variable message can also be computed by using the forward-backward algorithm [8]. The detailed computation approaches using the MAP algorithm were elaborated by Zhang and Mansour [16] [1]. To illustrate how the MAP algorithm works, a pseudo-random 12×16 LDPC **H** matrix is defined as follows:

$$H = \begin{bmatrix} 1111000000000000 \\ 0000111100000000 \\ 0000000011110000 \\ 0000000000001111 \\ \\ 0100000110000001 \\ 1000001000101000 \\ 0000110001000010 \\ 0011000000010100 \\ \\ 0100000010010010 \\ 0000001100000101 \\ 1010100001000000 \\ 0001010000101000 \end{bmatrix}.$$

This matrix is divided into 3 sub-matrices $\mathbf{H}_1$, $\mathbf{H}_2$, and $\mathbf{H}_3$, in which $\mathbf{H}_2$ and $\mathbf{H}_3$ are pseudo-random permutations of $\mathbf{H}_1$. As a consequence, each round of message updating across the whole matrix is considered a full decoding iteration, and each round of iteration is divided into 3 sub-iterations corresponding to

*Corresponding Author: Ahmed Refaey Hussein
Department of Electrical and Computer Engineering University of Western Ontario,
N6A 5B9, London, Canada.

the message computation of $\mathbf{H}_1$, $\mathbf{H}_2$, and $\mathbf{H}_3$. First, the extrinsic information is computed in the MAP component decoder of $\mathbf{H}_1$ and the results are permuted to the MAP component decoder of $\mathbf{H}_2$ through interleaver $\mathbf{M}_1$. Similarly, the computed messages in $\mathbf{H}_2$ are fed to the MAP component decoder of $\mathbf{H}_3$ through interleaver $\mathbf{M}_2$. This implies a complete round of iteration and the next iteration begins with the message passing from $\mathbf{H}_3$ to $\mathbf{H}_1$ through interleaver $\mathbf{M}_3$ [17]. The trellis diagram of the above **H** matrix is shown in Figure 4. The MAP component decoders of the **H** matrix are comprised of several independent simple MAP decoders associated with the sub-matrices. The detailed architecture of the MAP decoder is found in [16] and [18]. In fact, the architecture in [18] is claimed to achieve low interconnect complexity and low memory access in addition to faster convergence [17].

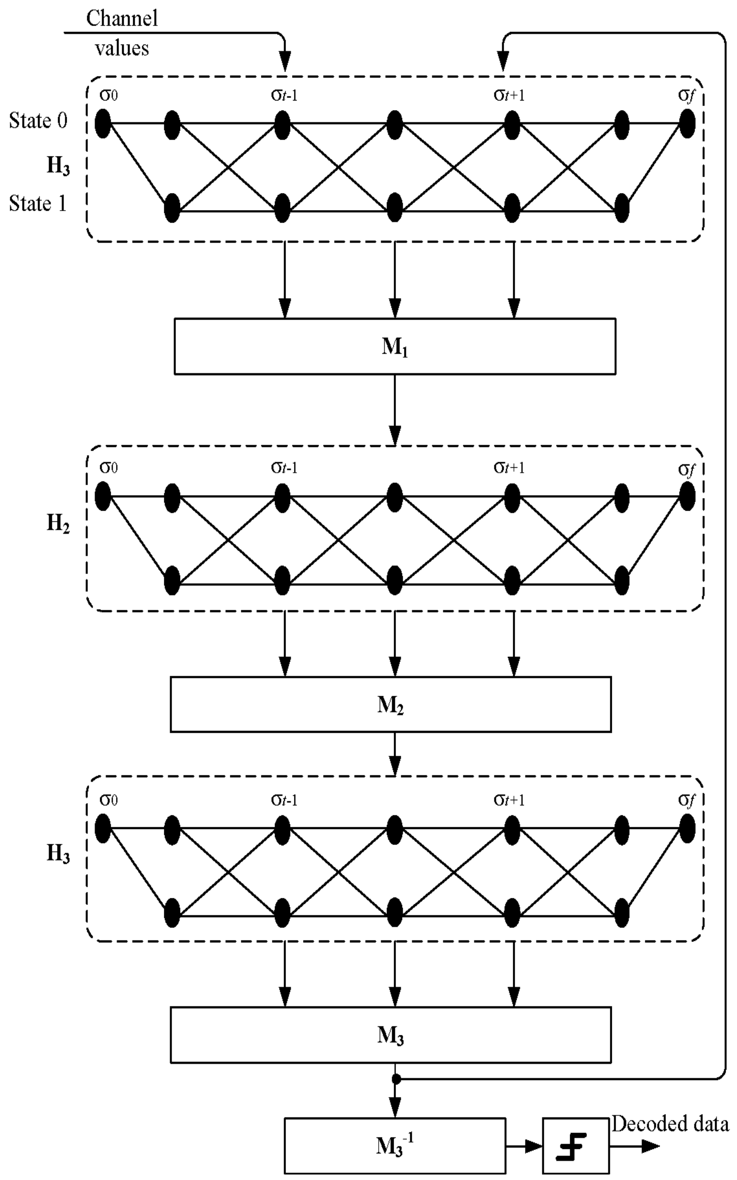

Figure 4: Trellis diagram of LDPC decoder based on single parity-check code.

Eventually, the parity-check matrix for the turbo code can be expressed by [19]:

$$H = \begin{bmatrix} 10100000 & 11100000 & 0000000 \\ 01010000 & 01110000 & 0000000 \\ 00101000 & 00111000 & 0000000 \\ 00010100 & 00011100 & 0000000 \\ 00001010 & 00001110 & 0000000 \\ 00000101 & 00000111 & 0000000 \\ 00000010 & 00000011 & 0000000 \\ 00000001 & 00000001 & 0000000 \\ & & \\ 00001001 & 0000000 & 11100000 \\ 01010000 & 0000000 & 01110000 \\ 00000101 & 0000000 & 00111000 \\ 10010000 & 0000000 & 00011100 \\ 00000110 & 0000000 & 00001110 \\ 10100000 & 0000000 & 00000111 \\ 00000010 & 0000000 & 00000011 \\ 00100000 & 0000000 & 00000001 \end{bmatrix}.$$

This **H** matrix is similar to the matrix given in Example 4. Furthermore, we can treat the turbo code as a concatenation of n super-codes, just like LDPC codes [1]. Consequently, it is natural to obtain a unified MAP algorithm to produce a combined decoder. It should be noted that each super-code has a simpler trellis structure so that the MAP algorithm can be efficiently executed.

3. Proposed Combined Decoder Architecture

In this section, we propose a combined convolutional, turbo, and LDPC decoder architecture based on a MAP component decoder.

IV. COMBINED DECODER ARCHITECTURE BASED ON MAP ALGORITHM

In this subsection, we present an optimized combined compliant decoder for the two families of codes considered in this paper. As a matter of fact, the turbo decoder and convolutional decoder have similar structures based on a MAP component decoder [20]. In the previous section, we introduced the MAP algorithm for LDPC codes. This opens the possibility of a single MAP-based unified decoder for both families of codes by merging the different MAP decoders mentioned above.

However, these decoders differ substantially in their memory requirements and branch-metric calculations (due to the different code structures). Since the branch-metric calculation units (BMU) are of less complexity, and thus area, three separate BMUs can be implemented. For moderate throughput requirements, forward and backward recursions can be calculated serially on the same hardware unit (SMU). As a consequence, one SMU can be used for the three decoders with only few modifications. In practice, only a single set of multiplexers has to be added to the SMU due to the efficient data ordering and the optimized memory organization. Hence, the critical path is not significant altered. The LLR values are calculated in a dedicated LLR unit (LLRU) which consists of two pipelined trees which perform additions, comparisons and subtractions. The LLR calculation is performed in the same loop as the backward recursion (MAP Algorithm, Appendix 3). Thus, only the α -values have to be stored in memory (α -RAM in Figure 5), whereas the β -values are directly consumed after calculation. The size of the α -RAM is determined by the block size. Furthermore, the well-known sliding window approach [21] allows reducing this memory significantly at the cost of some additional computations (acquisition phase). Precisely, a window size of 64 was found to be a good trade-off between computational overhead and memory size. Parts of the pipeline tree of the turbo code LLRU can be used for the convolutional and LDPC codes as well. Only the feedback loop of the convolutional code has to be added in the case of recursive code classes. Accordingly, most of the computational hardware can be reused for the three decoders. Nevertheless, the whole architecture is dominated by memory. Therefore, an efficient memory partitioning is essential for architectural efficiency. Following the architecture given in [20], the turbo decoder is dominated by the I/O memories due to the large block sizes. The α -RAM is of negligible size. In the case of the convolutional decoder, it is the opposite: the α -RAM is significantly larger than the I/O memories. Therefore, it is not possible to use the same I/O RAMs for turbo, convolutional, and LDPC codes. The same is true for the α -RAMs. In fact, we propose here the architecture in general cases. However, for a specific case, we can merge the memories which store the same amount of data. As in [20], the turbo code I/O-RAMs and the convolutional code α -RAM store about the same amount of data and are therefore merged.

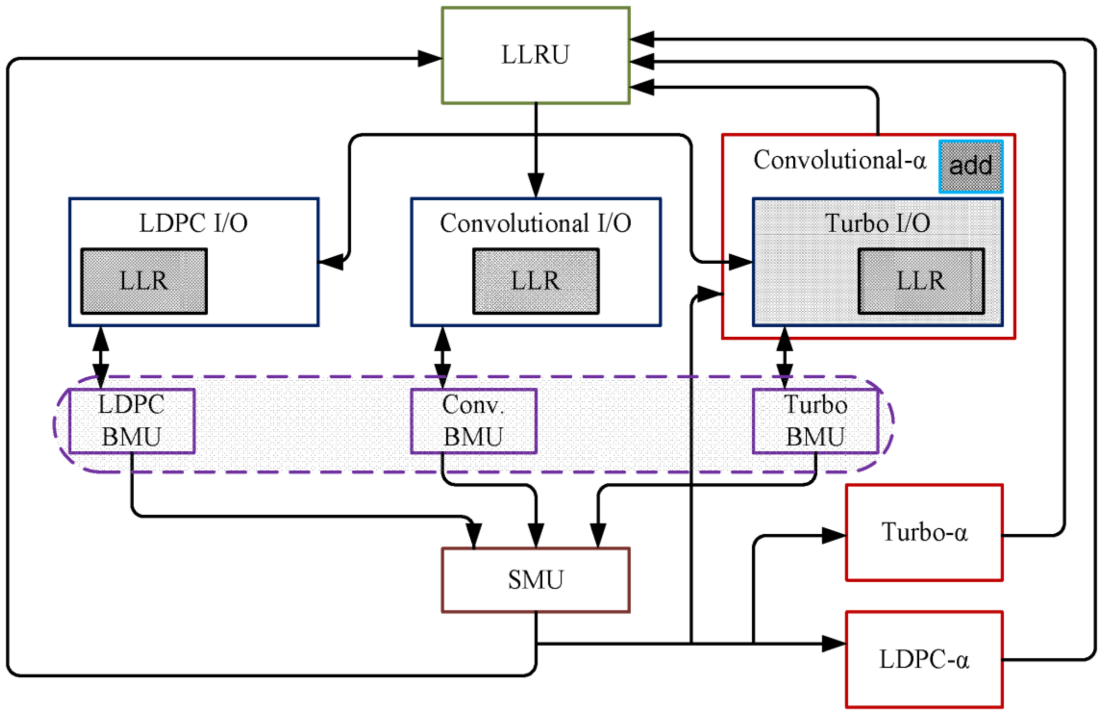

Figure 5: Proposed combined decoder scheme.

Considering the UMTS combined architecture example [20], the turbo code I/O memories can be partitioned in such a way that it is possible to use them as connvolutional α -RAM. The RAM has to be 88 bit wide because 8 state-metric values, each 11 bit wide, are calculated at every time step. To build this memory, each of the turbo code I/O RAMs is split into three separate RAMs resulting in a total of 12 RAMs, each of size 1728×6. These RAMs are then concatenated together with an additional 1728×16 RAM, forming the required bit-width for the storage of 8 state metrics in parallel. This memory sharing enables a window size of 54 for convolutional code decoding.

V. CONCLUSION

The application of the MAP algorithm to decode the regular LDPC code has been investigated. As a matter of fact, for a general LDPC code, the trellis structure is sufficiently complicated that it may be difficult to efficiently represent it in hardware as its very large generator matrix requires large computation and storage. Therefore, a new solution has been proposed which consists in using a sectionalized trellises as in a previous work on linear block codes. Indeed, using the structural properties of properly sectionalized trellises for linear block codes reduces the decoding complexity and the latency of the MAP algorithm can be reduced. Consequently, we showed that it is possible to simplify the complexity of LDPC decoders using the sectionalized trellises. In parallel, we considered a super-code based decoding algorithm for LDPC codes as an alternative solution, and we showed the applicability of this approach to introduce a new turbo decoder. Finally, we were able to propose a combined decoder architecture for the convolutional, turbo, and LDPC codes based on the unified MAP algorithm.

*Corresponding Author: Ahmed Refaey Hussein
Department of Electrical and Computer Engineering University of Western Ontario, N6A 5B9, London, Canada.

REFERENCES

[1]. M. M. Mansour and N. R. Shanbhag, "High-Throughput LDPC Decoders," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 11, pp. 976-996, December 2003.

[2]. D. Hocevar, "A Reduced Complexity Decoder Architecture Via Layered Decoding of LDPC Codes," in IEEE Workshop on Signal Processing Systems Design and Implementation, pp. 107-112, October 2004.

[3]. Y. Zhu and C. Chakrabarti, "Architecture-Aware LDPC Code Design for Multiprocessor Software Defined Radio Systems," IEEE Transactions on Signal Processing, vol. 57, pp. 3679 -3692, September 2009.

[4]. J. Zhang and M. Fossorier, "Shuffled Belief Propagation Decoding," Asilomar Conference on Signals, Systems and Computers, November 2002.

[5]. L. Jin and J.M.F. Moura, "Turbo Like Decoding of LDPC Codes," Digests of INTERMAG 2003. International Magnetics Conference, pp. DT–11, March 2003,

[6]. Y. Dai, Z. Yan, and N. Chen, "High-Throughput Turbo Sum-Product Decoding of QC LDPC Codes," 40th Annual Conference on Information Sciences and Systems, vol. 11, pp. 839- 844. March 2006.

[7]. Y. Sun and J. R. Cavallaro, "Unified Decoder Architecture For LDPC/Turbo Codes," in IEEE Workshop on Signal Processing Systems, SiPS, vol. 11, p. 1318, March 2008.

[8]. L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate," IEEE Transactions on Information Theory, vol. 20, pp. 284–287, March 1974.

[9]. J. K. Wolf, "Efficient Maximum Likelihood Decoding of Linear Block Codes Using a Trellis," IEEE Transactions on Information Theory, vol. 20, pp. 76–80, June 1978.

[10]. R. J. McEliece, T. Fujiwara, and S. Lin, "On the BCJR Trellis for Linear Block Codes," IEEE Transactions on Information Theory, vol. 42, pp. 1072–1092, July 1996.

[11]. A. Lafourcade and A. Vardy, "Optimal Sectionalization of a Trellis," IEEE Transactions on Information Theory, vol. 42, pp. 689–703, May 1996.

[12]. T. Kasami, T. Takata, T. Fujiwara, and S. Lin, "On Branch Labels of Parallel Components of the L-section Minimal Trellis Diagrams for Binary Linear Block codes," IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, vol. E77-A, pp. 1058–1068, June 1994.

[13]. D. J. Muder, "Minimal Trellises for Block Codes," IEEE Transactions on Information Theory, vol. 34, no. 5, pp. 1049–1053, September 1988.

[14]. S. Lin and M. P. C. Fossorier, "Trellises and new trellis-based decoding algorithms for codes," in Information Theory Workshop, June 1998, pp. 79–80.

[15]. D. J. C. MacKay, "Good Error-Correcting Codes Based on Very Sparse Matrices," IEEE Transactions on Information Theory, vol. 45, pp. 399– 431, June 1999.

[16]. M. M. Mansour and N. R. Shanbhag, "Turbo Decoder Architectures for Low-Density Parity-Check Codes," IEEE Global Telecommunications Conference, vol. 2, pp. 1383–1388, November 2002.

[17]. L. Ye, L. Shu, and M.P.C. Fossorier, "MAP Algorithms for Decoding Linear Block Codes Based on Sectionalized Trellis Diagrams," IEEE Transactions on Communication, vol. 48, pp. 577–587, April 2000.

[18]. Y. Zhang, Z. Wang, and K. K. Parhi, "Efficient High-Speed Quasi-Cyclic LDPC Decoder Architecture," in Asilomar Conference on Signals, Systems and Computers, vol. 1, pp. 540–544, November 2004.

[19]. Ahmed Refaey, Sebastien Roy, and Paul Fortier, "On the application of BP decoding to convolutional and turbo codes," Asilomar Conference on Signals, Systems Computers, pp. 996–1001, November 2009.

[20]. F. Berens and G. Kreiselmaier, "Combined Turbo-Code/Convolutional Code Decoder, In Particular for Mobile Radio Systems," US Patent 7191377, March 2007.

[21]. M. Marandian, J. Fridman, Z. Zvonar, and M. Salehi1, "Performance Analysis of Sliding Window Turbo Decoding Algorithms for 3GPP FDD Mode," International Journal of Wireless Information Networks, vol. 9, pp. 39–54, January 2002.