PUM-BASED TURBO CODES

L Fagoonee*, Prof B Honary*, C Williams**

*Dept. of Communication Systems, Lancaster University, Lancaster, UK
****QinetiQ, Malvern, UK**

1. Introduction

Lauer [1] introduced Partial Unit Memory (PUM) codes in 1979. PUM codes, which can be described as multiple-input convolutional codes, are optimal in the sense of having maximum free distance for a given code rate, number of encoder inputs and number of encoder memory cells. Their advantage over standard convolutional codes is the reduced number of states in their trellis for the same number of encoder inputs because only a fraction of these are shifted to the memory cells. On the other hand, PUM codes can attain larger free distances than block codes of the same codeword size and rate. Moreover, the block structure of the code facilitates synchronization. Another advantage of PUM codes is that their block length is such that they can be chosen to agree with the byte or word length of the target microprocessor, allowing further simplification during implementation. A number of non-systematic generator matrices have been searched and published whose distance properties sometimes exceed equivalent convolutional codes of the same block size and memory complexity [1,2]. In recent years turbo codes have proved themselves when used in conjunction with iterative decoding schemes. However, to maintain a reasonably high throughput, turbo codes use systematic constituent codes and transmit the information bits only once together with a number of sets of parity bits. We have previously attempted to construct systematic PUM codes [3], but did not allow feedback in the encoder structure thereby limiting the free distance of the resulting code and hence their error correction ability when used together with maximum-likelihood decoding. In this paper, we describe the construction of minimal recursive systematic PUM generator encoder structures from equivalent non-systematic generator matrices, which generate codewords whose free distances reach or are very close to the upper bound defined in [1].

The paper is organised as follows. In section 2, we briefly describe PUM codes and their properties. In section 3, we show, by means of an example, how to construct equivalent systematic PUM generator matrices from known non-systematic generator matrices. The resulting encoder structure can sometimes be minimised to an equivalent structure with fewer delay elements. This minimalization procedure is described in section 4 by continuing the example from the previous section. In section 5, results from the simulation of a turbo code with two constituent recursive systematic PUM (RSPUM) codes in a Gaussian channel are illustrated. This turbo code is compared with an equivalent RSC turbo code. By equivalent, we mean that these component codes have the same free distance, constraint length and code rate.

Simulation results indicate that the RSPUM turbo code outperforms the equivalent RSC turbo code. In general, the performance of RSPUM turbo codes is comparable or in many cases better than that of the equivalent classical RSC turbo code.

2. PUM Codes

A codeword, $\boldsymbol{x}_t$, of the (n, k, μ, d_{free}) PUM code is a function of the current input word of k information bits, $\boldsymbol{u}_t$ and a fraction μ of the previous input word, $\boldsymbol{u}_{t-1}$. This can be expressed by the following equation: $\boldsymbol{x}_t = \boldsymbol{u}_t\mathbf{G_0} + \boldsymbol{u}_{t-1}\mathbf{G_1}$.

$\mathbf{G_0}$ and $\mathbf{G_1}$ are generator matrices of size $k \times n$, where n is the codeword length. The rank of the $\mathbf{G_1}$ matrix is μ, where $\mu < k$. μ determines the state complexity of the state diagram and trellis of the code. The addition and multiplication operations are modulo-2 for binary codes. The free distance, d_{free} is the minimum weight of a codeword that diverges from the all-zero path in the encoder standard trellis and returns to the all-zero path for the first time.

$\mathbf{G_0}$ and $\mathbf{G_1}$ can be represented by one generator matrix of the form G(D) where the contents of the matrix are a function of the time delay. The latter representation will be used throughout the rest of this paper.

A PUM code trellis can be represented by a standard trellis with 2^{μ} starting and ending states and 2^{k} branches leaving and entering each node, respectively. Since a PUM code is characterised by $k>\mu$, there are $2^{k-\mu}$ parallel branches between any

two states in the trellis. Each branch is labelled with *n* symbols. For codes encoded by non-systematic generator matrices, the trellis labels will all be parity bits. In the case of systematic code structures, the trellis labels will each be made of *k* information bits and *n-k* parity bits.

The generic max-log MAP decoding of systematic codes with multi-labelled trellis branches can be summarised by the following equations.

The forward and backward recursion node metrics, α and β, at time t are a function of the branch transition probability Γ for all possible k inputs allowing transition from *s'* to *s*. The log-likelihood Γ is the sum of the *a priori* likelihoods, *L(u)* for the *k* information bits and the likelihoods of the *n* channel-compensated received values $L_c y$.

$$\log \Gamma_t(s', s) = \frac{1}{2}\left[\sum_k u_{t,k} L(u_{t,k}) + \sum_n x_{t,n} L_c y_{t,n}\right]$$

$$\log \alpha_t(s) = \max_{s'}\left(\log \Gamma_t(s', s) + \log \alpha_{t-1}(s')\right)$$

$$\log \beta_{t-1}(s') = \max_{s}\left(\log \Gamma_t(s', s) + \log \beta_t(s)\right)$$

The a posteriori LLR for each of the *k* symbols at time t is the sum of the *a priori* LLR, the LLR of the channel-compensated received value and the extrinsic LLR of the symbol in question.

$$\hat{L}(u_{t,j}) = L(u_{t,j}) + L_c y_{t,j} + \log \gamma_{t,j}(s', s)$$

$$\log \gamma_{t,j}(s', s) = \frac{1}{2}\left[\sum_{\substack{k,\\ k \neq j}} u_{t,k} L(u_{t,k}) + \sum_{\substack{n,\\ n \neq j}} x_{t,n} L_c y_{t,n}\right]$$

3. Construction of RSPUM code structures

If an encoding matrix has some *k×k* sub matrix whose determinant is a delay free polynomial, premultiplication by such a matrix yields an equivalent systematic encoding matrix [3].

Every convolutional matrix is equivalent to a systematic rational encoding matrix. If the determinant of the leftmost *k×k* sub matrix of *G(D)* is not a delay free polynomial, then we can always, by permuting the columns of *G(D)*, find a generator matrix *G'(D)* whose leftmost *k×k* sub matrix has a determinant which is a delay free polynomial, where *G'(D)* encodes an equivalent code. Hence, without loss of generality we can write a systematic encoding matrix $G(D)=(I_k R(D))$.

As an example, consider the (4,2,1,4) PUM code with generator matrix *G(D)*.

$$G(D) = \begin{bmatrix} 1 & 1\,1\,1 \\ 1+D & 0\,1\,D \end{bmatrix}$$

Let the T(D) be the leftmost 2×2 sub matrix of *G(D)*, whose determinant is *1+D*.

$$T(D) = \begin{bmatrix} 1 & 1 \\ 1+D & 0 \end{bmatrix}$$

The equivalent systematic generator matrix $G_{sys}(D)$ is given by:

$$G_{sys}(D) = T(D)^{-1}.G(D)$$

$$G_{sys}(D) = \frac{1}{1+D}\begin{bmatrix} 0 & 1 \\ 1+D & 1 \end{bmatrix}.G(D) = \begin{bmatrix} 1\,0 & \frac{1}{1+D} & \frac{D}{1+D} \\ 0\,1 & \frac{D}{1+D} & \frac{1}{1+D} \end{bmatrix}$$

The controller canonical form of $G_{sys}(D)$ contains 2 storage registers as shown in Figure 1.

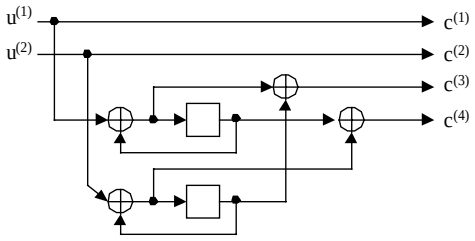

Fig 1: Controller canonical form of $G_{sys}(D)$.

However, a minimal encoder with only 1 register can be realized. This minimalization process [4] is described below.

4. Low complexity RSPUM encoder structure

Let $s^{(1)}_t$ and $s^{(2)}_t$ be the states of the first and second delay elements at time t, $u^{(1)}_t$ and $u^{(2)}_t$ be the two inputs to the encoder at time t, $c^{(3)}_t$ and $c^{(4)}_t$ be the two parity bits calculated by the encoder at time t. The two systematic outputs can be ignored for this process, as they remain unchanged.

The next states of the delay elements at time t+1 and the outputs of encoder at time t can be written as a function of the states and the inputs at time t.

$$s^{(1)}_{t+1} = s^{(1)}_t + u^{(1)}_t$$
$$c^{(3)}_t = s^{(1)}_{t+1} + s^{(2)}_t = s^{(1)}_t + s^{(2)}_t + u^{(1)}_t$$
$$s^{(2)}_{t+1} = s^{(2)}_t + u^{(2)}_t$$
$$c^{(4)}_t = s^{(1)}_t + s^{(2)}_{t+1} = s^{(1)}_t + s^{(2)}_t + u^{(2)}_t$$

These equations are used to build Table 1.

Present State $s^{(1)}_t\ s^{(2)}_t$	Inputs $u^{(1)}_t\ u^{(2)}_t$			
	00	**01**	**10**	**11**
00	00/00	01/01	10/10	11/11
01	01/11	00/10	11/01	10/00
10	10/11	11/10	00/01	01/00
11	11/00	10/01	01/10	00/11

Table 1: Successor states and outputs ($s^{(1)}_{t+1}\ s^{(2)}_{t+1}/c^{(3)}_t\ c^{(4)}_t$) as a function of the present state ($s^{(1)}_t\ s^{(2)}_t$) and the inputs ($u^{(1)}_t\ u^{(2)}_t$).

In order to find which states are equivalent, those states that correspond to the same outputs are merged into partition P_1: {00,11}, {01,10}.
The states in each set are 1-equivalent. The next stage is to determine whether this set can be further broken down into a partition P_2 whose states are 2-equivalent. Two states are 2-equivalent if they are 1-equivalent and their successor states are 1-equivalent. In this case, $P_2=P_1$. In general, partitioning is carried out till $P_{i+1}=P_i$.

In order to obtain a linear realization of the minimal encoder, we let 0 represent the state {00,11} and 1 represent {01,10}. This is shown in Table 2.

Present State s_t	Inputs $u^{(1)}_t\ u^{(2)}_t$			
	00	**01**	**10**	**11**
0	0/00	1/01	1/10	0/11
1	1/11	0/10	0/01	1/00

Table 2: Successor state and output ($s_{t+1}/c^{(3)}_t\ c^{(4)}_t$) as a function of the present state (s_t) and the inputs ($u^{(1)}_t\ u^{(2)}_t$)

From Table 2, the successor state can be represented as a function of the present state and inputs as shown in Table 3.

Present State	Inputs			
	00	**01**	**10**	**11**
0	0	1	1	0
1	1	0	0	1

Table 3: Successor state as a function of the present state (s_t) and the input ($u^{(1)}_t\ u^{(2)}_t$).

Hence, $s_{t+1} = s_t + u^{(1)}_t + u^{(2)}_t$. The same procedure is used to determine the parity bits as a function of the present state and *k* inputs. This yields for the first parity bit: $c^{(3)}_t = s_t + u^{(1)}_t$, and the second parity bit: $c^{(4)}_t = s_t + u^{(2)}_t$. Figure 2 illustrates the minimised encoder structure representing the new equations determined.

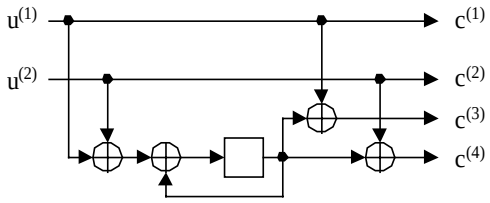

Fig 2: Minimized realization of the systematic (4,2,1,4) PUM Code.

The decoding trellis, including branch labels, of the above realization is shown in Figure 3. The labels are written in the order $c^{(1)}_t\ c^{(2)}_t\ c^{(3)}_t\ c^{(4)}_t$. The edge complexity is only 4 branches per information bit (bpi).

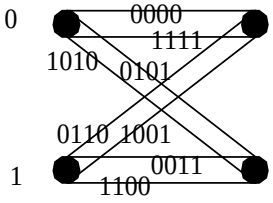

Fig 3: Low complexity 2-state trellis

5. Simulation and performance

A parallel-concatenated code with two component (4,2,1,4) RSPUM codes is constructed to yield a rate $^1/_3$ turbo code. Figure 4 shows its performance in an AWGN channel for an interleaver size of 10000.
The encoder structure of the equivalent recursive systematic convolutional code with constraint length 2 is shown in Figure 5. Its trellis has 2 states and a branch complexity of 4 bpi. The code has a free distance of 4 over two information symbols.

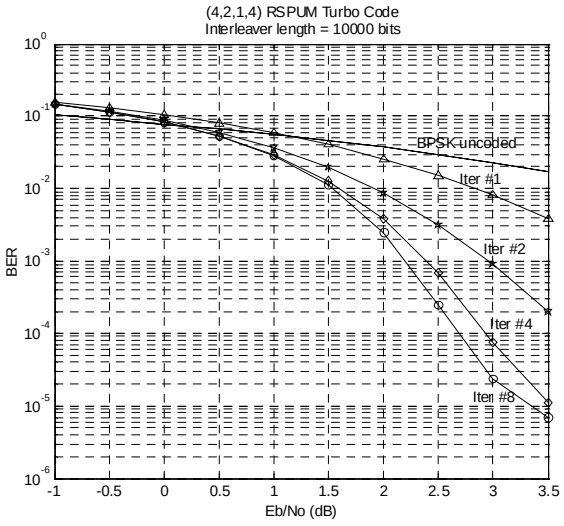

Fig 4: Performance of (4,2) RSPUM turbo code for up to 8 iterations.

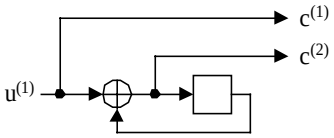

Fig 5: Equivalent RSC encoder structure.

The relative performance of the rate $^1/_3$ turbo decoders with two constituent RSC codes and RSPUM codes respectively is illustrated in Figure 6. Both turbo encoders use an interleaver size of 10000.

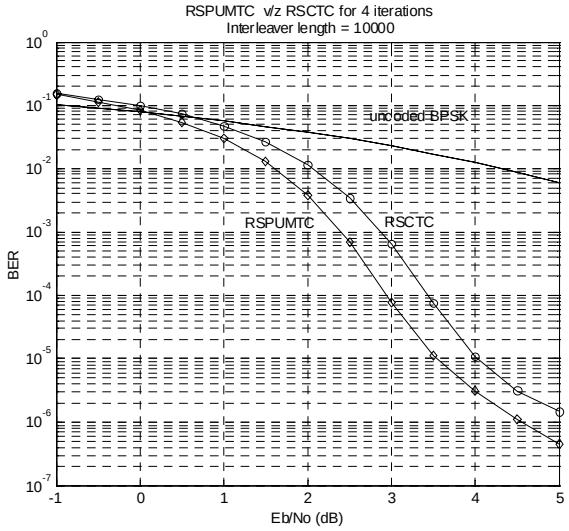

Fig 6: Relative performance of equivalent codes for 4 iterations.

Overall, the RSPUM turbo code outperforms the equivalent RSC turbo code by about 0.5 dB.

6. Conclusion

In this paper, the example of the simple (4,2,1,4) PUM code was used to demonstrate construction of the low complexity and maximum distance recursive systematic encoder structure. Hence recursive systematic PUM code structures can be efficiently constructed from known non-systematic generator matrices in order to retain their good distance properties. These encoder structures can be further minimised to yield encoder structures with fewer memory elements. The resulting code trellis has a very low decoding complexity and can be used effectively in turbo code systems. Simulation results showed that the performance of the (4,2,1,4) RSPUM as a constituent code in a rate $^1/_3$ turbo code system outperforms the equivalent RSC turbo code.

REFERENCES

[1] G S Lauer, “Some Optimal Partial Unit-Memory Codes”, IEEE Trans. Inf. Theory, Vol. IT-25, No. 2, pp. 240-243, March 1979.

[2] L N Lee, “Short, unit-memory, byte-oriented, binary convolutional codes having maximal free distance”, IEEE Trans. Inf. Theory, vol. IT-22, pp. 349-352, May 1976

[3] L Fagoonee, B Honary and C Williams. "Error Protection Using PUM Turbo Codes for Future Mobile Communication Systems", 2nd Int. Conf. on 3G Mobile Communication Technologies, London, March 2001.

[4] R Johannesson and K S Zigangirov, “Fundamentals of Convolutional Coding”, IEEE Press, 1999.

ACKNOWLEDGEMENT

This work is funded under the Ministry of Defence Applied Research Program. The authors would also like to thank Prof. Shavgulidze for his suggestions and support.