LTE-Advanced Channel Coding Generic Procedures

A High-level Model To Guide Low-Level Implementations

Abstract — **This paper presents a high-level functional model for the 3GPP LTE-Advanced channel coding generic procedures. Due to the complexity of modern wireless communication systems, it is fundamental to have a well-defined high-level model to guide low-level implementations, like C/C++ or VHDL. High-level modeling not only serves the purpose of proving the correctness of a given system, but also to present measures of its optimal performance, since we are not yet restricted by any technology. The LTE-Advanced channel coding is composed by five generic procedures: CRC calculation, code block segmentation and CRC attachment, channel coding, rate matching and code block concatenation. The aggregation of these procedures creates the great majority of the downlink and uplink transport channels. In this paper, we present a very simple way of modeling the channel coding, offering a reference for future LTE channel coding developments.**

Index Terms—**LTE; channel coding; generic procedures; high-level model.**

I. Introduction

Wireless communication systems are becoming more complex every day. Physical (PHY) layer requirements demand attention to details ranging from performance to power consumption. In order to implement such systems in any available technology, such as GPPs/DSPs, FPGAs, ASICs, etc., a well-defined high-level model is necessary. From the system-level specification to the final low-level implementation many design phases are required when dealing with such complex systems.

The first phase and the most important one, since it serves as reference to all other phases, is the design of a high-level functional model. The efficiency and performance of the system being designed is greatly influenced by the results of this phase. A poor-defined or even incorrect high-level model leads to a not-compliant implementation. Cycle and bit-accurate model phases follows, until it is possible to map each operation onto a GPP/DSP instruction set or to a hardware architecture.

Among the most recent wireless communication systems proposed today is the LTE-Advanced, a 4G wireless broadband medium access technology. The LTE-Advanced standard, defined by the 3rd Generation Partnership Project (3GPP), is composed of several documents to specify its operation. From a PHY perspective, at least 3 documents must be carefully studied, namely, the 3GPP TS 36.211, 36.212 and 36.213. From those, TS 36.212 "Channel Coding and Multiplexing" is the one that offers the most relevant information about channel coding generic procedures. Understanding such, sometimes difficult to read, standards is essential to design an efficient and effective high-level functional model.

There are five generic procedures in channel coding that need to be performed in order to implement the transport channels and control information types described in the standard: CRC calculation, code block segmentation and CRC attachment, channel coding, rate matching and code block concatenation.

The paper is organized as follows. In Section II we present the theoretical aspects of the five channel coding procedures. In Section III we briefly discuss the transport channels of the LTE-Advanced. Section IV presents the high-level models for such procedures. Section V demonstrates the encoding results for a arbitrary input bit stream. The paper is concluded in Section VI.

II. LTE Channel Coding Generic Procedures

Data and control streams from and to the Media Access and Control (MAC) layer are either encoded or decoded to offer transport and control services over the radio transmission link. Channel coding scheme is a combination of error detection, error correcting, rate matching, interleaving and information mapping onto/splitting from physical channels. This section presents generic coding procedures adopted by LTE-Advanced [1].

A. CRC Calculation

The MAC layer protocols operate on the unit of Transport Blocks (TB). In order to support forward error correction (FEC), a TB is first appended with a cyclic redundancy check (CRC) sequence computed from all input bits of the TB, using a CRC generator polynomial of length $L = 24$, $g_{CRC24A}(D)$ which is showed in eq. (1).

$$g_{CRC24A}(D) = [D^{24}+D^{23}+D^{18}+D^{17}+D^{14}+D^{11}+D^{10}+D^{7}+D^{6}+D^{5}+ +D^{4}+D^{3}+D+1] \quad (1)$$

Another CRC sequence of length $L = 24$ is also defined in [1] and used only when code block segmentation is applied, based on a second generator polynomial $g_{CRC24B}(D)$ which is showed in eq. (2).

$$g_{CRC24B}(D) = [D^{24}+D^{23}+D^{6}+D^{5}+D+1] \quad (2)$$

There are also two other CRC polynomials presented in [1] which are g_{CRC16} and g_{CRC8}. The first is attached to the Master-Information Block (MIB) and Downlink Control Information (DCI) messages and has the following polynomial:

$$g_{CRC16}(D) = [D^{16}+D^{12}+D^5+1] \quad (3)$$

The latter one is employed by some uplink channels (PUCCH and PUSCH) for transmitting Channel Quality Indicator (CQI) information and its polynomial is given by eq. (4).

$$g_{CRC8}(D) = [D^{24}+D^{23}+D^6+D^5+D+1] \quad (4)$$

An assessment of the error detection reliability of LTE CRC coding is presented in [3].

B. Code Block Segmentation and Code Block CRC Attachment

The limited number of code block sizes, K, to be used by the LTE Turbo-code internal interleaver is presented in Table 5.1.3-3 of [1]. The maximum code block size, Z, is equal to 6144 bits (including CRC) [1]. In case the block size B, which includes the TB size plus CRC, exceeds Z, code block segmentation is applied. Code block segmentation forces a TB to be segmented into smaller CBs of predefined sizes K [4, 5].

In order to ensure that the size of each CB is matched to the set of available code block sizes K, filler bits may have to be inserted at the head of the first code block [22]. Figure 1. shows the code block segmentation and CRC attachment procedure.

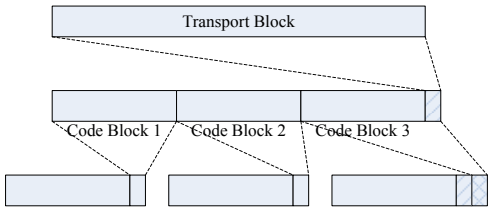

Figure 1. Code block segmentation and CRC attachment.

Code block segmentation implies that an additional CRC is calculated and appended to each new CB. Having a CRC per code block allows for early detection of correctly decoded code blocks and corresponding early termination of the iterative decoding. This reduces the terminal processing effort, thus also reducing power consumption. In case of no code block segmentation, when there is only a single CB, no additional CRC is attached. Details of the code block segmentation algorithm can be found in [1, 4, 15, 23].

C. Channel Coding

Control channels and data channels in LTE use convolutional and Turbo codes [2], respectively, where the latter is an enhanced development of convolutional codes achieving near-Shannon performance.

1) Tail biting Convolutional Coding

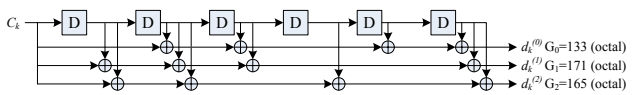

Figure 2. Rate 1/3 tail biting convolutional encoder.

Convolutional coding is a form of forward error correction that improves the channel capacity by adding carefully selected redundant information. A convolutional code is called tail-biting when its codewords are those code sequences associated with paths in the trellis that start from a state equal to the last m bits of an information vector of k data bits. Tail biting ensures that the final state of a convolutional encoder is the same as the initial state.

This is done by initializing the shift register state with the last bits of the information block. The tail-biting convolutional decoder can be implemented using the same low complexity decoding algorithms that exist to traverse the trellis, such as the Viterbi algorithm [6]. Tail biting can reduce coding overheads at the expense of an increase in complexity and slight degradation in the decoding performance.

LTE uses a 1/3 code rate tail biting encoder with a constraint length $k = 7$. This means that one in three bits of the output contain 'useful' information while the other two add redundancy. The structure of the convolutional encoder is showed in Fig. 2.

Each output stream of the encoder is obtained by convolving the input with the impulse response, called generator, of the encoder $(d_k^{(i)} = C_k * G_i)$. For LTE the three generator sequences are: G_0 = 133 (octal), G_1 = 171 (octal), and for last G_2 = 165 (octal).

2) Turbo Coding

Turbo codes were first proposed by Berrou et al. [2] in 1993 as an enhanced convolutional coding technique. Since its introduction, turbo code has become the coding technique of choice in many communication systems due to its near Shannon limit error correction capability.

The fundamental turbo code encoder is built using two identical Recursive Systematic Convolutional (RSC) codes with parallel concatenation [2]. A RSC encoder is typically a rate 1/2 encoder known as the constituent encoder. The input to the second constituent encoder is interleaved first. Only one of the systematic outputs from the two component encoders is used, once the systematic output from the other component encoder is just a permuted version of the chosen systematic output.

The turbo coding used in the LTE employs a Parallel Concatenated Convolutional Code (PCCC) with two 8-state constituent encoder as shown in Fig. 3. The coding rate of turbo encoder is 1/3. The transfer function for the constituent coders is given by (5):

$$G(D) = \left[1, \frac{g_1(D)}{g_0(D)}\right] \quad (5)$$

where, $g_0(D) = 1 + D^2 + D^3$ and $g_1(D) = 1 + D + D^3$.

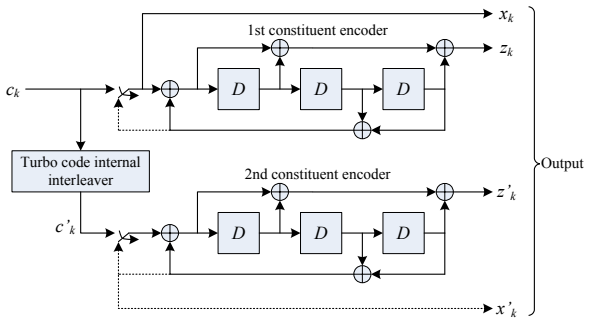

Figure 3. Rate 1/3 turbo encoder.

LTE employs a Quadrature Permutation Polynomial (QPP) based interleaver [7]. The QPP interleaver provides a mapping from the input (non-interleaved) bits to the output (interleaved) bits according to the function:

$$c(i) = (f_1 \cdot i + f_2 \cdot i^2) \bmod K \qquad (8)$$

where i is the input bit index of the interleaver, $c(i)$ is the output index of the interleaver, and K is the code block size. The values of the parameters f_1 and f_2 depend on the size of K. The LTE specification lists all supported K sizes, ranging from a minimum of 40 bits to a maximum of 6144 bits, together with the associated values of f_1 and f_2 [16, 17, 18, 19, 20, 21].

In contrast to the WCDMA/HSPA turbo-code interleaver, a QPP-based interleaver is maximum contention free [8], implying that the decoding can be parallelized without the risk for contention when different parallel processes are accessing the interleaver memory. In [20] it is presented a very efficient way to design the turbo coder in hardware.

D. Rate Matching

The rate matching procedures is composed of sub-block interleaver, bit collection and bit selection. The purpose of these procedures is to create different code rates according to some PHY parameters, such as modulation order, channel quality, etc. [1, 9, 10].

There are two types of rate matching: one specified for turbo and one for convolutional encoded data streams. As mentioned before, the code rate for both is the same. The structure of the rate matching for both encoder types is depicted in Fig. 4.

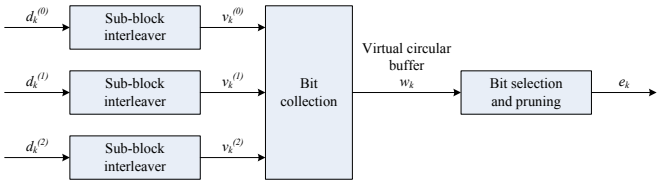

Figure 4. Rate matching generic procedure.

In order to obtain other code rates, repetition or puncturing has to be performed. The rate matching module consists of three so-called sub-block interleavers for the three parallel output streams of the encoder and a bit selection stage, which is realized by a circular buffer (bit collection).

The sub-block interleaver is based on the classic row-column interleaver with 32 columns and intra-column permutation. The bits of each of the three streams are written row-by-row into this R x 32 matrix, where R is the number of rows and its value depends on the coded block size. Dummy bits are padded to the first row of each stream to fill the matrix, if necessary. After a column-based permutation, bits are read out from the matrix column-by-column. An optimized rate-matching architecture is proposed in [14].

1) Circular-buffer rate matching for turbo code

The circular buffer is the most important part of the rate matching, making possible to change the code rate through puncturing and repetition of the mother code. The structure of the Turbo encoder with rate matching is shown in Fig. 5. The interleaved systematic bits are written into the circular buffer in sequence, with the first bit of the interleaved systematic bit stream at the beginning of the buffer. The interleaved and interlaced parity bit streams are written second to the buffer, with the first bit of the stream next to the last bit of the interleaved systematic bit stream.

A number of coded bits (depending on the code rate) are read out serially from a certain starting point, k_0, specified by Redundancy Version (RV), in the buffer. If the end of the buffer is reached and more coded bits are needed for the transmission, the transmitter wraps around and continues at the beginning of the buffer [11].

The advantages of circular buffer are its flexibility in the code rates achieved and granularity in streams sizes and performance [12, 13]. Moreover, circular buffer is well suited to HARQ operation as different redundancy versions (RV) can be specified by simply defining different starting points in circular buffer. HARQ technique is used to control the retransmission of packets, so that, if there is no error in the decoded packet, there is no retransmission [24].

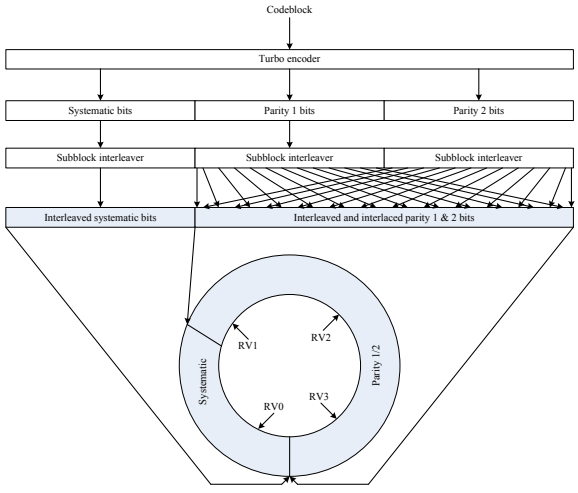

Figure 5. Circular buffer rate matching for turbo code.

2) Circular-buffer rate matching for convolutional code

The circular buffer rate matching scheme for convolutional code is illustrated in Fig. 6. Note that here there are no systematic bits as outputs of the convolutional code, but three parity streams $d_k^{(0)}$, $d_k^{(1)}$ and $d_k^{(2)}$. Similar to turbo code, sub-block interleaving is performed separately on each of the output streams. Another difference is that interleaved parity bits are not interlaced before storing to the circular buffer. Also, there are no redundancy version starting points, since hybrid ARQ operation is not used for convolutionally coded transport channels.

E. Code Block Concatenation

The code block concatenation is the gathering of different coded CBs into a single bit stream. It is only performed when the number of CBs is greater than one for the turbo coding channels. This generic procedure does not add, remove or alter any bits.

III. LTE Transport Channels and Control Information

By combining these generic procedures, we establish the channel coding processing chain for most of the downlink and uplink LTE transport channels. Transport channels provide the interface between the MAC and PHY layer. Each channel has a different purpose that will be briefly discussed. From data to control information, all of them will use some or all of the generic procedures presented before. The only transport channels that do not use these procedures are the HARQ Indicator (HI) and Control Format Indicator (CFI) channels, for downlink and the Uplink Control Information (UCI) and Random Access Channel (RACH) for uplink.

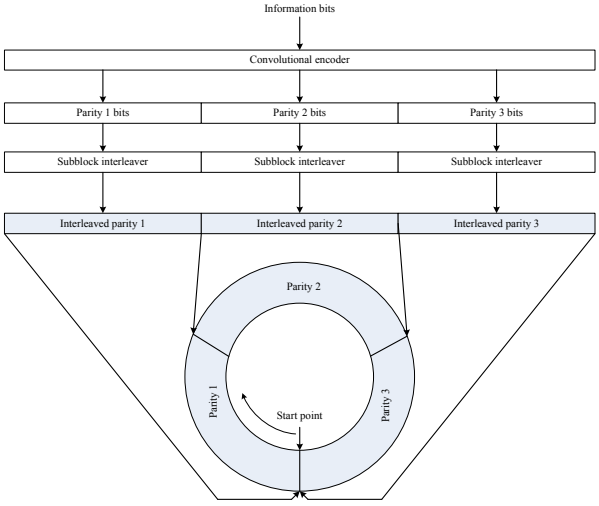

Figure 6. Circular buffer rate matching for convolutional code.

A. Broadcast Channel (BCH)

BCH is the first channel UE receives after acquiring synchronization to the cell. The primary purpose of the BCH is to broadcast a certain set of cell and/or MIB which is needed for UE's to access the network.

The transmission chain processing for the BCH first attaches a CRC16 to the MIB message (14 information bits + 10 spare bits). After CRC attachment, the information bits are coded using a rate 1/3 tail-biting convolutional coding. The coded bits are then rate matched using the circular buffer approach. Next, the BCH is mapped to a physical channel referred to as physical broadcast channel (PBCH), which transmits a TB, corresponding to the MIB, over four sub-frames with 40 ms timing interval, providing a data rate of just 350 bps.

B. Downlink Shared Channels (DL-SCH), Paging Channels (PCH) and Multicast Channel (MCH)

The DL-SCH is the main downlink transport channel type in LTE and is used for both user data and control information as well as part of downlink system information that is not mapped to the BCH.

PCH conveys the PCCH, which is used for paging information when searching a UE on a network. This channel is also used to inform UE's about updates of the system information. MCH is used to transport user data or control messages that require MBSFN (Multicast-Broadcast Single Frequency Network) combining.

The processing chain for all the three transport channels is the same. A 24-bit CRC is calculated and appended to each TB. After that, code block segmentation and CRC attachment follows. Large TBs are segmented into multiple CBs and a new CRC24B is calculated and appended to each one. The resulting CBs are then delivered to the turbo encoder, which feeds the rate matching with its encoded streams. At last, the concatenation block combines the rate matched turbo coded blocks into a single codeword, which is then sent for downstream physical channel processing.

C. Downlink Control Information (DCI)

The DCI carries downlink or uplink scheduling information as well as uplink power control commands. The DCI is mapped on the Physical Downlink Control Channel (PDCCH). It has four different formats and further different variations for each format. Depending on the size DCI is transmitted using one or more Control Channel Elements (CCEs). The DCI may provide the control information for the following cases:

- PUSCH allocation information (DCI Format 0);
- PDSCH information with one codeword (DCI Format 1 and its variants);
- PDSCH information with two codewords (DCI Format 2 and its variants);
- Uplink power control information (DCI Format 3 and its variants).

The processing chain for the DCI is as follows. Initially, a CRC16 is appended to the end of each PDCCH payload. Next, the DCI message with the CRC attached undergoes convolutional coding. Then, the rate matching procedure creates an output bit stream with a desired code rate.

D. Uplink Shared Channel (UL-SCH)

This transport channel is used to transport uplink user data and/or control messages. The UL-SCH carries all the information from all the uplink logical channels. The UL-SCH supports:

- Dynamic link adaptation by varying the transmit power and potentially modulation and coding;
- Hybrid HARQ;
- Dynamic and semi-static resource allocation;
- Power control;
- Beam forming (optional).

The UL-SCH transport channel is mapped to physical uplink shared channel (PUSCH). To create the PUSCH payload, a TB undergoes the encoding process, which includes CRC24A calculation, code block segmentation and CRC24B attachment, turbo coding, rate matching and code block concatenation. This processing is described in [1], Sections 5.2.2.1 to 5.2.2.5 and 5.2.2.8.

Additionally coding of control information is carried out with the resulting codewords time multiplexed and interleaved with the UL-SCH data. The control information includes the CQI, Precoder Matrix Indication (PMI), Rank Indication (RI) and HI. Together these three fields form UCI. The additional processing for the UCI is described in [1], Sections 5.2.2.6 to 5.2.2.8.

It is possible for the PUSCH to carry only control information and no data. In this case only the control information coding and multiplexing chain are followed.

IV. High-Level Models

The models shown here were developed in Matlab using scripting language. It was assumed that all data inputs are bit serial row vectors. Most of the input parameters to each generic procedure can be inferred from the size of the input block. For the turbo encoding generic procedure a copy of the turbo code interleaver parameters is also stored to support its operation. Regarding the rate matching procedure, which needs some information about the sub-frame configuration, it is necessary to explicitly pass such parameters.

A. CRC Calculation

For the calculation of the CRC, the following code is used to compute and append a CRC to an input bit stream. For each CRC type, a different polynomial should be passed to the class constructor. Here is an example of the construction of a CRC16 polynomial generator.

Listing (a). CRC generator handler.

```
hCRC16 = comm.CRCGenerator('Polynomial', [16 12 5 0]);
TB_plus_CRC = step(hCRC16, TB);
```

To append a CRC to an input bit stream, a call to the step function should be used. The step function has two input parameters, one for the handler and one for the data vector. It executes one iteration of the method specified by the handler.

B. Code Block Segmentation and Code Block CRC Attachment

The code block segmentation has a straightforward coding, since it is already specified as a pseudo-code in the 3GPP 36.212 Std. [1]. An optimized algorithm for code block segmentation can be found in [15].

C. Channel Coding

As mentioned on section II, there are two types of encoders. For both of them, a system object is created for modeling the channel coding. On listing (b) we present a way on how to create handlers for the channel encoding procedures.

Listing (b). Turbo and convolutional encoder handlers.

```
hTEnc = comm.TurboEncoder('TrellisStructure', poly2trellis(4, ...
[13 15], 13), 'InterleaverIndices', indexes );
ECB = step(hTEnc, CB);

hCEnc = comm.ConvolutionalEncoder('TrellisStructure', ...
poly2trellis(7, [133 171 165]),'TerminationMethod','Terminated');
ETB = step(hCEnc, TB_plus_CRC);
```

For the turbo encoder it is necessary to specify the interleaver indexes. In this case, the indexes must follow the pattern expressed by (8).

The values of K, f_1 and f_2 must be fetched from table 5.1.3-3 [1]. The input bit stream is encoded by using the step function, which receives as parameters the encoder handler and the input bit stream.

D. Rate Matching

There are three sub-procedures in rate matching:

1) Sub-block interleaver

To compute the sub-block interleaver, first we need to find the number of rows, padding bits and matrix size. The patterns for intra-column permutation are presented in table 5.1.4-1 [1]. The sub-block interleaver can be modeled as the following:

Listing (c). Sub interleaver indexes generation.

```
Nrows = ceil(block_size / 32);          % row size
Msize = Nrows * Ncols;                  % matrix size
Npadd = Msize - Ncode;                  % zero pad (at begin)
idx = 0 : Msize-1;
a = pattern(floor(idx ./ Nrows) + 1);
b = Ncols * mod(idx, Nrows);
pa = mod(a + b, Msize);                 % pattern d0(k) and d1(k)
pi = mod(a + b + 1, Msize);             % pattern  d2(k)
```

In this code fragment, 'pattern' is a row-vector containing the permutation pattern of the encoder, 'pa' should be used to index all convolutional encoded channels, while 'pi' is used only in turbo codes on parity $d_2(k)$.

2) Bit collection

Bit collection is modeled as a circular buffer for the rate matching procedure. For turbo encoded channels, parity bits should be interlaced before storing, while convolutional channels store their parity bits in sequence.

Listing (d). Bit collection circular buffer.

```
if encoder_type == 'turbo'
  buffer = [vk0 reshape([vk1 vk2], 1, 2 * Msize)];
else
  buffer = [vk0 vk1 vk2];
end
```

3) Bit selection

Bit selection is a simple read out procedure that retrieve bits from the circular buffer, starting from k_0 index and removing any null (invalid) bits, if any, inserted by the sub-block interleaver procedure, until E valid bits were read. If the procedure reaches the maximum buffer size without completing the amount of valid bits needed, it starts over from the beginning of the memory.

Listing (e). Rate matching bit selection.

```
k =0; j = 0;
while k < E
   if buffer(mod(j + k0, buf_size) + 1) ~= <INVALID>
      e(k) = buffer(mod(j + k0, buf_size) + 1);
      k= k + 1;
   end;
   j = j + 1;
end;
```

E. Code Block Concatenation

This generic procedure is simply modeled as the concatenation of all the encoded bit streams at the bit selection output resulting from the segmentation of a single input TB.

V. Discussion And Results

In this section we present the simulation results of a DL-SCH transport channel, based on the models presented before. Since convolutional encoded channels less complex than turbo encoded channels, because there is no need to perform code block segmentation and concatenation and because it has a simplified rate-matching structure, we limit this discussion only to turbo coded channels.

We assume the following system configuration for a LTE eNodeB used to demonstrate the generic procedures: frequency division duplex (FDD) mode, UE category 3, one transmission layer, RV0, single input - single output (SISO) mode, QPSK modulation and a total available bits for the transmission of one TB (G) of 8000 bits. These configuration parameters are only relevant for computing of k_0 and E, related to the rate-matching procedure [1].

Assuming a discrete impulse response (only the MSB bit of the codeword is on) of length 6200 bits, representing an arbitrary TB delivered by the MAC layer, the first procedure of the DL-SCH channel coding is to calculate the CRC. The hexadecimal result of the CRC24A calculated for this specific input is 0xEC739E.

After calculating and attaching the CRC24A to the TB data, the block size B is now 6224 bits. The segmentation procedure follows and after evaluating B, two CBs of size 3136 bits are selected with no filler bits. A new CRC is calculated and appended to each new CB. The CRC24B calculated for both CB1 and CB2 are 0x2B182F and 0x440E69, respectively.

CB1 have 389 bytes, which are fetched from the original TB plus CRC24A, and the remaining 3 bytes are the CRC24B. The hexadecimal value of CB1 is 0x8000 ...(zeros)... 2B182F. CB2 has the same size and contains the remaining bits of the TB, resulting in the hexadecimal value of 0xEC739E440E69 (zeros to the left are not presented).

Once the TB is segmented into two CBs, the turbo encoder procedure is performed. To initialize the interleaver, from K = 3136, we have f_1 = 13 and f_2 = 28. Three parallel output streams of size 3140 (3136 plus 4 bits from the trellis termination procedure of the encoder) are generated for CB1 and CB2, respectively.

$d_k^{(0)}$ = 0x8000...2B182F4 and $d_k^{(0)}$ = 0x0000...440E695
$d_k^{(1)}$ = 0xF2E5...1FF48F4 and $d_k^{(1)}$ = 0x0000...9B25656
$d_k^{(2)}$ = 0xF2E5...0000004 and $d_k^{(2)}$ = 0x0000...972E5C6

After turbo coding, the three parallel bit streams are interleaved by the sub-block interleaver before bit collection and selection are carried out, all of them being sub-procedures of the rate matching procedure (valid bits only).

$v_k^{(0)}$ = 0x0000...0000000 and $v_k^{(0)}$ = 0x0000...0000003
$v_k^{(1)}$ = 0x5CB9...2E5DB96 and $v_k^{(1)}$ = 0x0000...0000002
$v_k^{(2)}$ = 0x76E5...CB96D70 and $v_k^{(2)}$ = 0x16E5...F7E5D86

Since we are using turbo coding, the parity bits must be interlaced before storing in the rate-matching circular buffer. The final output of the DL-SCH will be generated by the bit selection, where k_0 = 198 and E = 4000 bits for each CB, based on system's configuration. The final channel coding result for the arbitrary impulse response TB with 775 bytes of length will be the concatenation of both bit streams generated for each CB by the bit selection. The head and tail of this bit stream of total length 8000 bits (1000 bytes) is presented below:

e_k = 0x0000...4544005

VI. Conclusion

In this paper we present high-level models for the LTE-Advanced channel coding generic procedures. The majority of downlink and uplink channels presented in the LTE-Advanced standard are encoded based on these procedures. Serving as a guide to low-level implementations, these models can help explore the design space of those implementations as well as to assure compliance. Since any mistake in the high-level model will be propagated through all design phases, this paper also offers a reference example for a DL-SCH channel to demonstrate all generic procedures.

Acknowledgment

The authors thanks the support given to this work, developed as part of the RASFA project, financed by the Fundo de Desenvolvimento das Telecomunicações - FUNTTEL, from the Brazilian Department of Communication,

through a partnership no. 01.09.0631.00 with FINEP – Finaciadora de Estudos e Projetos.

REFERENCES

[1] 3GPP Technical Specification 36.212, "Multiplexing and channel coding (Release 10)", ww.3gpp.org.

[2] C. Berrou, A. Glavieux, and P. Thitimajshima, “Near Shannon Limit Error-Correcting, Coding, and Decoding: Turbo Codes”, in Proceedings of the IEEE International Conference on Communications, 1993, pp. 1064-1070.

[3] Jung-Fu Cheng and Havish Koorapaty, “Error Detection Reliability of LTE CRC Coding”, Vehicular Technology Conference, 2008. VTC 2008-Fall. IEEE 68th.

[4] Karlo G. Lenzi, José A. B. Filho and Felipe A. P. Figueiredo, “Code Block Segmentation Hardware Architecture for LTE-Advanced”, to be published.

[5] R1-072673, “RE sizing for turbo code block segments”, Motorola, 3GPP TSG RAN1 #49bis, Orlando, USA, June 25-29, 2007.

[6] Rami A. Abdallah, Seok-Jun Leet, Manish Goel and Naresh R. Shanbhag, “Low-power pre-decoding based viterbi decoder for tail-biting convolutional codes”, SiPS 2009 - IEEE Workshop on Signal Processing Systems, 2009.

[7] J. Sun and O. Y. Takeshita, “Interleavers for Turbo Codes Using Permutation Polynomials Over Integer Rings”, IEEE Transactions on Information Theory, Vol. 51, No. 1, January 2005, pp. 101 - 119.

[8] O. Y. Takeshita, “On Maximum Contention-free Interleavers and Permutation Polynomials Over Integer Rings”, IEEE Transactions on Information Theory, Vol. 52, No. 3, March 2006, pp. 1249 – 1253.

[9] Long Yu, et. Al, “An Improved Rate Matching Algorithm for 3GPP LTE Turbo Code”, Third International Conference on Communications and Mobile Computing, IEEE Computer Society, 2011.

[10] Jung-Fu (Thomas) Cheng, et. Al, “Analysis of Circular Buffer Rate Matching for LTE Turbo Code”, Vehicular Technology Conference, 2008. VTC 2008-Fall. IEEE 68th.

[11] 3GPP Technical Specification 36.211, “Physical Channels and Modulation (Release 10)”, www.3gpp.org.

[12] R1-061050, "EUTRA FEC Enhancement", Motorola, France Telecom, GET, Orange, 3GPP TSG RAN WG1#44bis, Athens, Greece. 27-31 March 2006.

[13] R1-072137, "Turbo rate-matching in LTE", Motorola, 3GPP TSG RAN1 #49, Kobe, Japan, May 07 - 11, 2007.

[14] Karlo G. Lenzi, José A. B. Filho, Felipe A. Figueiredo, “Optimized Rate Matching Architecture for a LTE-Advanced FPGA-based PHY”, to be published.

[15] Karlo G. Lenzi, José A. B. Filho, Felipe A. Figueiredo, “On The Performance of Code block Segmentation for LTE-Advanced”, to be published.

[16] Y. Sun and J. Cavallaro, “Efficient hardware implementation of a highly parallel 3GPP LTE/LTE-advance turbo decoder”, Integration VLSI Journal, July - 2010.

[17] Shuenn-Gi Lee, Chung-Hsuan Wang, and Wern-Ho Sheen, “Architecture Design of QPP Interleaver for Parallel Turbo Decoding”, 71st IEEE Vehicular Technology Conference (VTC 2010-Spring), 2010.

[18] Christoph Studer, Christian Benkeser, Sandro Belfanti and Qiuting Huang , “Design and Implementation of a Parallel Turbo-Decoder ASIC for 3GPP-LTE”, IEEE Journal of Solid-State Circuits, Jan. 2011.

[19] Guohui Wang, Yang Sun, Joseph R. Cavallaro and Yuanbin Guo, “High-Throughput Contention-Free Concurrent Interleaver Architecture for Multi-Standard Turbo Decoder”, Proceedings of the ASAP 2011.

[20] Ardimas Andi Purwita, Arnaud Setio and Trio Adiono, “Optimized 8-Level Turbo Encoder Algorithm and VLSI Architecture for LTE”, International Conference on Electrical Engineering and Informatics, July 2011.

[21] Soo Yun Hwang, Dae Ho Kim and Kyoung Son Jhang, “Implementation of an Encoder based on Parallel Structure for LTE Systems”, IEEE Wireless Communications and Networking Conference (WCNC), April 2010.

[22] 3GPP Technical Specification 36.213, “Physical layer procedures (Release 10)”, www.3gpp.org.

[23] Motorola, "Code Block Segmentation for LTE Channel Coding", R1-071059, 3GPP TSG RAN WG1 #48, Feb. 2007; XP-050105053.

[24] Antonio Maria Cipriano, Paul Gagneur, Guillaume Vivier and Serdarn Sezginer, “Overview of ARQ and HARQ in Beyond 3G systems”, IEEE 21st International Symposium on Personal, Indoor and Mobile Radio Communications Workshops, Sept. 2010.