Implementation of Turbo Product Codes in the FEC-API

Kiran Karra

Virginia Tech
Agenda

• Introduction
• Turbo Product Code – Encoding Overview
• Turbo Product Code – Decoding Overview
• Implementation in C++
• BER Performance
• Computational Performance
• Using FEC-API TPC Blocks in GNURadio
Introduction to Turbo Product Codes

• Turbo Codes are a class of high-performance forward error correcting (FEC) codes.
• First practical codes to reach Shannon channel capacity limit.
• Different flavors of Turbo Codes exist:
  • Turbo Convolutional Code
  • Turbo Product Code
  • Enhanced Turbo Product Code
• Now used in multiple commercial standards including:
  • UMTS, CDMA2000, LTE, DVB-RCS, WiMAX
• Turbo Product Codes (a form of parallel concatenated codes) are the focus of this talk.
Turbo Product Codes – Encoding Overview

• Algorithm for encoding:
  1.) READ $k_y \times k_x$ bits
  2.) Place data into $k_y \times k_x$ matrix
  3.) ENCODE each row by $(n_x, k_x)$ systematic code
  4.) ENCODE each column by $(n_y, k_y)$ systematic code
  5.) TRANSMIT $n_x \times n_y$ code bits
  6.) GOTO Step 1

*Information on this slide copied Reference [1]
Turbo Product Codes – Encoding Overview (cont.)

• Block shortening is also sometimes performed, to conform to other PHY layer constraints, such as OFDM symbol size.

• Define \( k_{\text{per\_col}} = (k_y - l_y) \)

• Define \( k_{\text{per\_row}} = (k_x - l_x) \)

• Algorithm to allow for shortened blocks:
  1.) READ \( k_{\text{per\_row}} \times k_{\text{per\_col}} \) bits
  2.) PREPAD with \( B + Q \) zeros
  3.) Place data into \( k_{\text{per\_row}} \times k_{\text{per\_col}} \) matrix
  4.) ENCODE each row by \((n_x - l_x, k_{\text{per\_row}})\) systematic code
  5.) ENCODE each column by \((n_y - l_y, k_{\text{per\_col}})\) systematic code
  6.) DELETE \( B \) zeros
  7.) TRANSMIT \((n_x - l_x) \times (n_y - l_y) - B\) data bits
  8.) GOTO Step 1
Turbo Product Codes – Encoding Overview (cont.)

• Parameters such as $B$, $Q$, $k_x$, $I_x$, $k_y$, $I_y$ and the systematic code polynomials to encode the rows and columns all need to be picked carefully to ensure proper operation.

<table>
<thead>
<tr>
<th>Code bytes</th>
<th>Data bytes</th>
<th>grows</th>
<th>gcols</th>
<th>k_per_row</th>
<th>k_per_col</th>
<th>B</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>6</td>
<td>g1</td>
<td>g6</td>
<td>3</td>
<td>18</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>12</td>
<td>9</td>
<td>g1</td>
<td>g1</td>
<td>9</td>
<td>9</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>24</td>
<td>12</td>
<td>g1</td>
<td>g5</td>
<td>17</td>
<td>6</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>24</td>
<td>20</td>
<td>g1</td>
<td>g1</td>
<td>13</td>
<td>13</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>36</td>
<td>18</td>
<td>g1</td>
<td>g5</td>
<td>26</td>
<td>6</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>36</td>
<td>25</td>
<td>g1</td>
<td>g7</td>
<td>5</td>
<td>41</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>48</td>
<td>23</td>
<td>g6</td>
<td>g5</td>
<td>22</td>
<td>9</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>48</td>
<td>35</td>
<td>g6</td>
<td>g1</td>
<td>26</td>
<td>11</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>60</td>
<td>31</td>
<td>g6</td>
<td>g6</td>
<td>16</td>
<td>16</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>72</td>
<td>40</td>
<td>g6</td>
<td>g6</td>
<td>18</td>
<td>18</td>
<td>0</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Generator polynomial</th>
<th>Generator vector</th>
<th>Shorthand</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1 + X$</td>
<td>[11]</td>
<td>g1</td>
</tr>
<tr>
<td>$1 + X^2 + X^4 + X^5$</td>
<td>[101011]</td>
<td>g5</td>
</tr>
<tr>
<td>$1 + X + X^2 + X^3 + X^5 + X^6$</td>
<td>[1111011]</td>
<td>g6</td>
</tr>
<tr>
<td>$1 + X^2 + X^6 + X^7$</td>
<td>[10100011]</td>
<td>g7</td>
</tr>
</tbody>
</table>

*Information on this slide from Reference [1]*
Turbo Product Codes – Decoding Overview

• Turbo decoding is an iterative process
• Each encoder must be “undone” using the appropriate decoder.
• Decoders exchange information, and hence are required to operate on soft inputs and produce soft outputs.
  • Different than Viterbi decoding
  • MAP algorithm makes decisions based on most likely bit, rather than most likely sequence as in Viterbi decoding
• Basic building block of Turbo decoding is the SISO (Soft Input Soft Output) decoder
Turbo Product Codes – Decoding Overview

• SISO Decoding Overview
  • Forward Sweep
    • Sweep through trellis. At each node, update metric instead of ADD/COMPARE/SELECT
    \[
    \alpha_k = \max \ast [(\alpha_i + \gamma_{i,k}), (\alpha_j + \gamma_{j,k})]
    \]
  • Backward Sweep
    • Sweep through trellis. At each node, update metric instead of ADD/COMPARE/SELECT.
    \[
    \beta_k = \max \ast [(\beta_i + \gamma_{k,i}), (\beta_j + \gamma_{k,j})]
    \]
  • Update LLR
    \[
    \lambda'(u) = \max_{i \rightarrow j; u=1} \{\Lambda(i \rightarrow j)\} - \max_{i \rightarrow j; u=0} \{\Lambda(i \rightarrow j)\}
    \]

*Information on this slide from Reference [1]
Turbo Product Codes – Decoding Overview

• Block Diagram of overall decoder is shown below.
C++ Implementation

• Implementation drawn heavily from code provided by the Coded Modulation Library (CML). [https://code.google.com/p/iscml/](https://code.google.com/p/iscml/)

• RSCEncode and SISODecode functions were implemented in C in the CML.
  • RSCEncode and SISODecode are the core functions which enable encoding and decoding.
  • They were copied directly from CML into GNURadio as static functions (with small function argument changes, and some code optimizations).

• BTCEncode and BTCDecode functions were implemented in Matlab in the CML.
  • The BTCEncode and BTCDecode handled the row/column processing of the block’s, essentially repeatedly calling the RSCEncode or SISODecode functions described above on the appropriate row/col.
  • Matlab code was ported to separate GNURadio blocks.
    • TPCEncoder encompasses functionality of BTCEncode Matlab function.
    • TPCDecoder encompasses functionality of BTCDecode Matlab function.
    • GNU Scientific Library’s (GSL) matrix functionality was NOT used in implementation.
      • Perhaps one path to optimizing the code even further?

• In order to increase performance, early exit algorithm was added to Turbo Decoder
C++ Implementation (cont.)

• Early Exit Algorithm
  • The signs of the LLRs at the input and the output of the SISO module are compared.
  • Decoder is stopped if signs agree.
  • Higher SNR environment leads to less iterations of decoder because there is a higher likelihood that the LLRs will agree between input and output.
BER Performance
## Computational Performance

<table>
<thead>
<tr>
<th>Number of Decoder Iterations</th>
<th>Max Log-MAP</th>
<th>Constant Log-MAP</th>
<th>Linear Log-MAP</th>
<th>Log-MAP LUT</th>
<th>Log-MAP C</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>72 kbits/sec</td>
<td>56 kbits/sec</td>
<td>53 kbits/sec</td>
<td>47 kbits/sec</td>
<td>8 kbits/sec</td>
</tr>
<tr>
<td>5</td>
<td>87 kbits/sec</td>
<td>67 kbits/sec</td>
<td>64 kbits/sec</td>
<td>56 kbits/sec</td>
<td>10 kbits/sec</td>
</tr>
<tr>
<td>4</td>
<td>107 kbits/sec</td>
<td>85 kbits/sec</td>
<td>80 kbits/sec</td>
<td>70 kbits/sec</td>
<td>13 kbits/sec</td>
</tr>
<tr>
<td>3</td>
<td>144 kbits/sec</td>
<td>114 kbits/sec</td>
<td>106 kbits/sec</td>
<td>91 kbits/sec</td>
<td>17 kbits/sec</td>
</tr>
<tr>
<td>2</td>
<td>216 kbits/sec</td>
<td>169 kbits/sec</td>
<td>157 kbits/sec</td>
<td>139 kbits/sec</td>
<td>26 kbits/sec</td>
</tr>
</tbody>
</table>
Using FEC-API TPC Blocks
References

• M.C. Valenti, “Channel coding for IEEE 802.16e mobile WiMAX,” a tutorial presented at International Conference on Communications (ICC) (Dresden, Germany), June 18, 2009.

• Iterative Solutions Coded Modulation Library (https://code.google.com/p/iscml)