Implementation of Turbo Product Codes in the FEC-API

Kiran Karra

Virginia Tech

Agenda

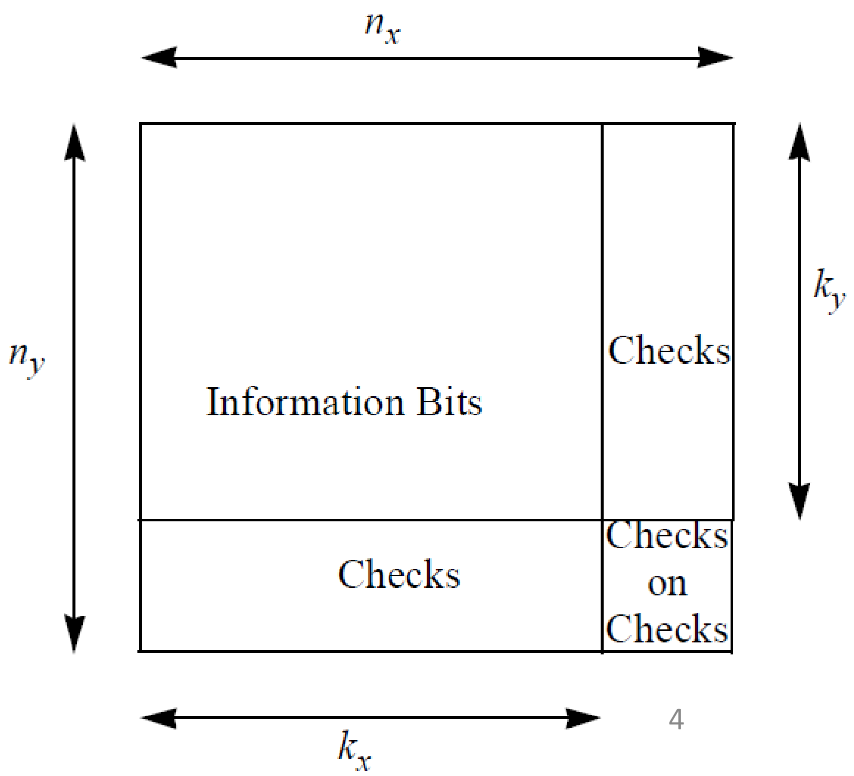
- Introduction
- Turbo Product Code – Encoding Overview
- Turbo Product Code – Decoding Overview
- Implementation in C++
- BER Performance
- Computational Performance
- Using FEC-API TPC Blocks in GNURadio

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 x k_x bits
  2.) Place data into k_y x 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 x 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_per_col = (k_y – I_y)
- Define k_per_row = (k_x – I_x)
- Algorithm to allow for shortened blocks:

  1.) READ k_per_row x k_per_col bits

  2.) PREPAD with B+Q zeros

  3.) Place data into k_per_row x k_per_col matrix

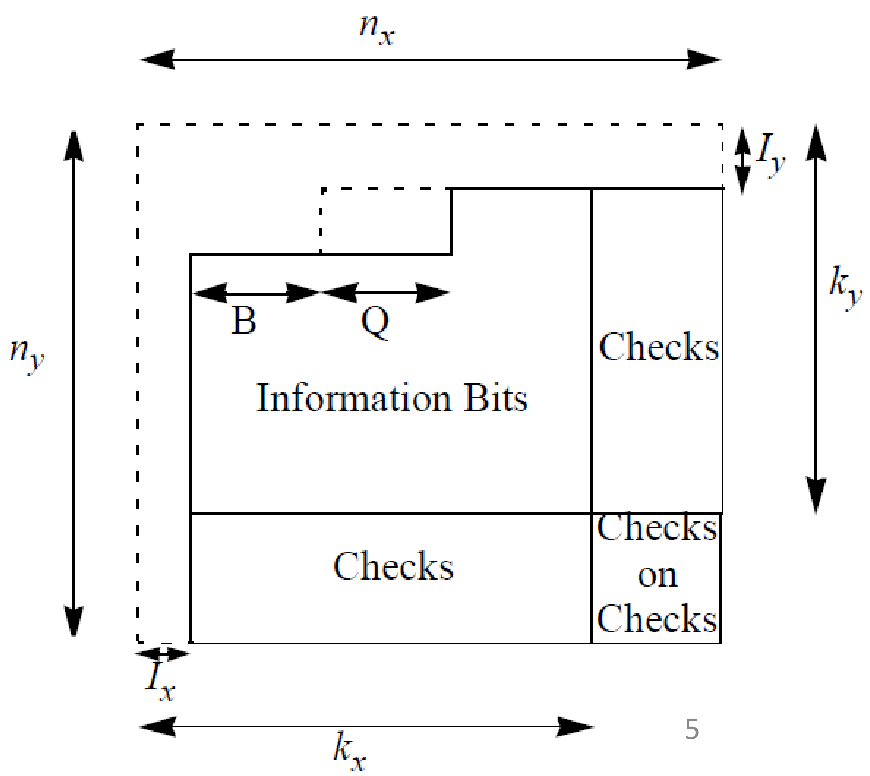
  4.) ENCODE each row by (n_x-I_x, k_per_row) systematic code

  5.) ENCODE each column by (n_y-I_y,k_per_col) systematic code

  6.) DELETE B zeros

  7.) TRANSMIT (n_x-I_x)* (n_y-I_y) – B data bits

  8.) GOTO Step 1



Turbo Product Codes – Encoding Overview (cont.)

- Parameters such as B, Q, k_x, l_x, k_y, l_y and the systematic code polynomials to encode the rows and columns all need to be picked carefully to ensure proper operation

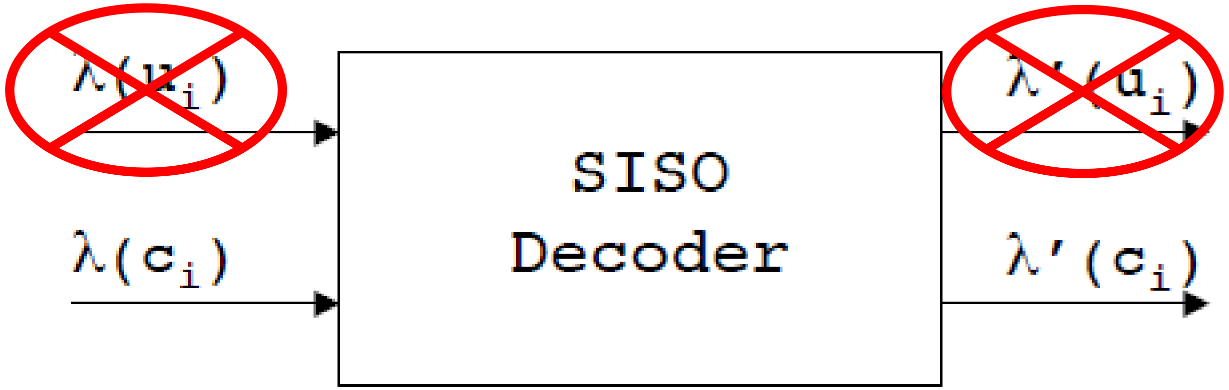
Code bytes	Data bytes	grows	gcols	k_per_row	k_per_col	B	Q
12	6	g1	g6	3	18	0	6
12	9	g1	g1	9	9	4	5
24	12	g1	g5	17	6	6	0
24	20	g1	g1	13	13	4	5
36	18	g1	g5	26	6	9	3
36	25	g1	g7	5	41	0	5
48	23	g6	g5	22	9	8	6
48	35	g6	g1	26	11	0	6
60	31	g6	g6	16	16	4	4
72	40	g6	g6	18	18	0	4

Generator polynomial	Generator vector	Shorthand
$1 + X$	$[11]$	g1
$1 + X^2 + X^4 + X^5$	$[101011]$	g5
$1 + X + X^2 + X^3 + X^5 + X^6$	$[1111011]$	g6
$1 + X^2 + X^6 + X^7$	$[10100011]$	g7

*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

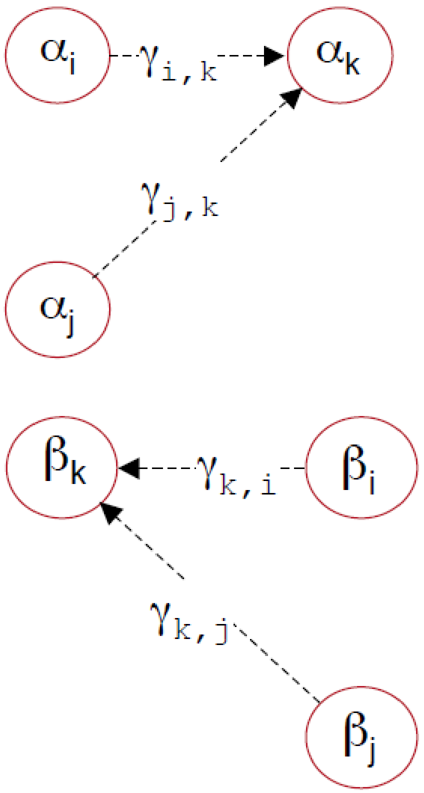
$$\alpha_k = \max * \left[\left(\alpha_i + \gamma_{i,k} \right), \left(\alpha_j + \gamma_{j,k} \right) \right]$$

  - Backward Sweep
    - Sweep through trellis. At each node, update metric instead of ADD/COMPARE/SELECT.

$$\beta_k = \max * \left[\left(\beta_i + \gamma_{k,i} \right), \left(\beta_j + \gamma_{k,j} \right) \right]$$

  - Update LLR

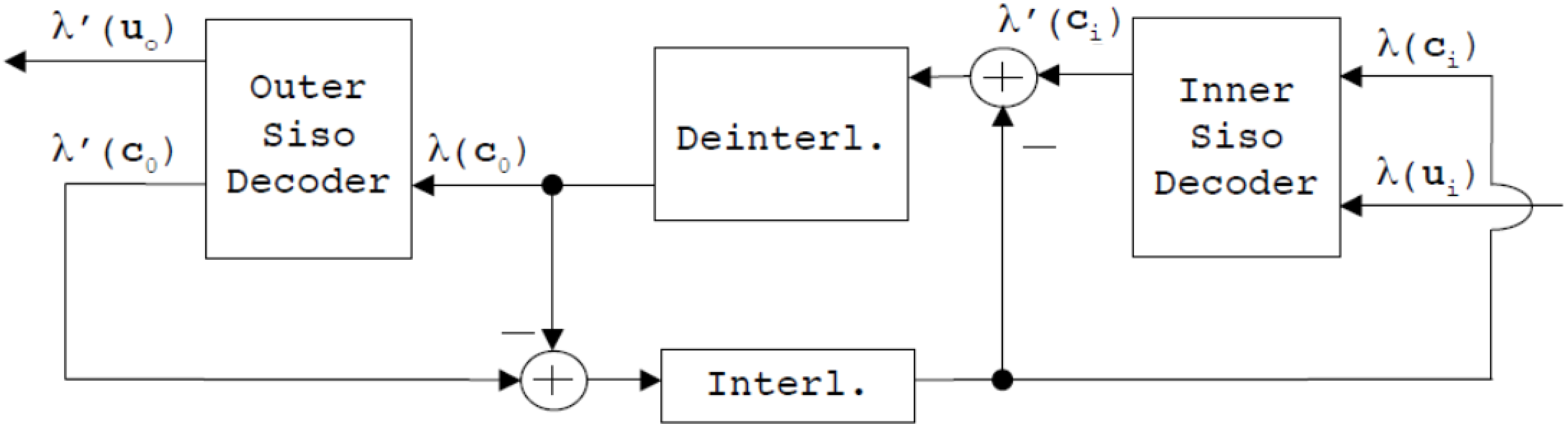
$$\lambda'(u) = \max_{i \to j : u=1} * \{ \Lambda(i \to j) \} - \max_{i \to j : u=0} * \{ \Lambda(i \to j) \}$$



*Information on this slide from Reference [1]

Turbo Product Codes – Decoding Overview

- Block Diagram of overall decoder is shown below.



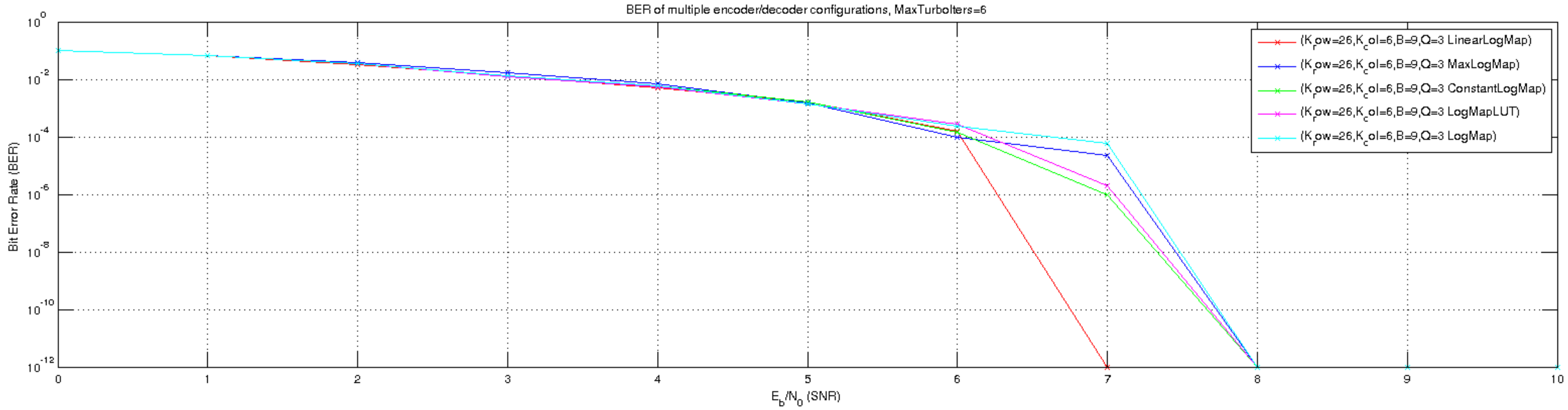
C++ Implementation

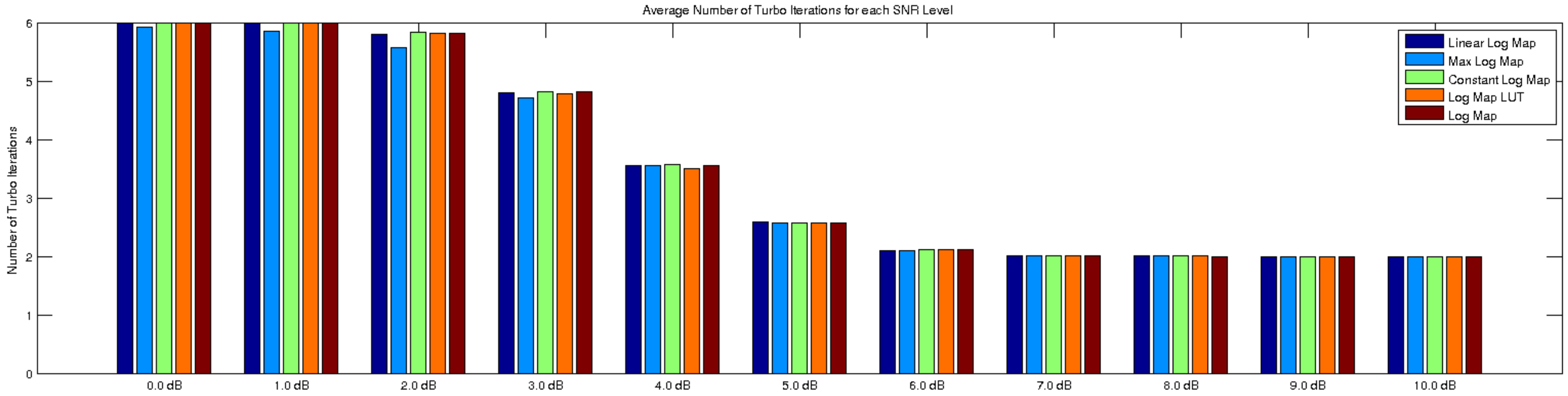
- Implementation drawn heavily from code provided by the Coded Modulation Library (CML). https://code.google.com/p/iscml/
- **RSCEncode** and **SISODecode** functions were implemented in C in the CML.
  - **RSCEncode** and **SISODecode** are the core functions which enable encoding and decoding.
  - 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.)

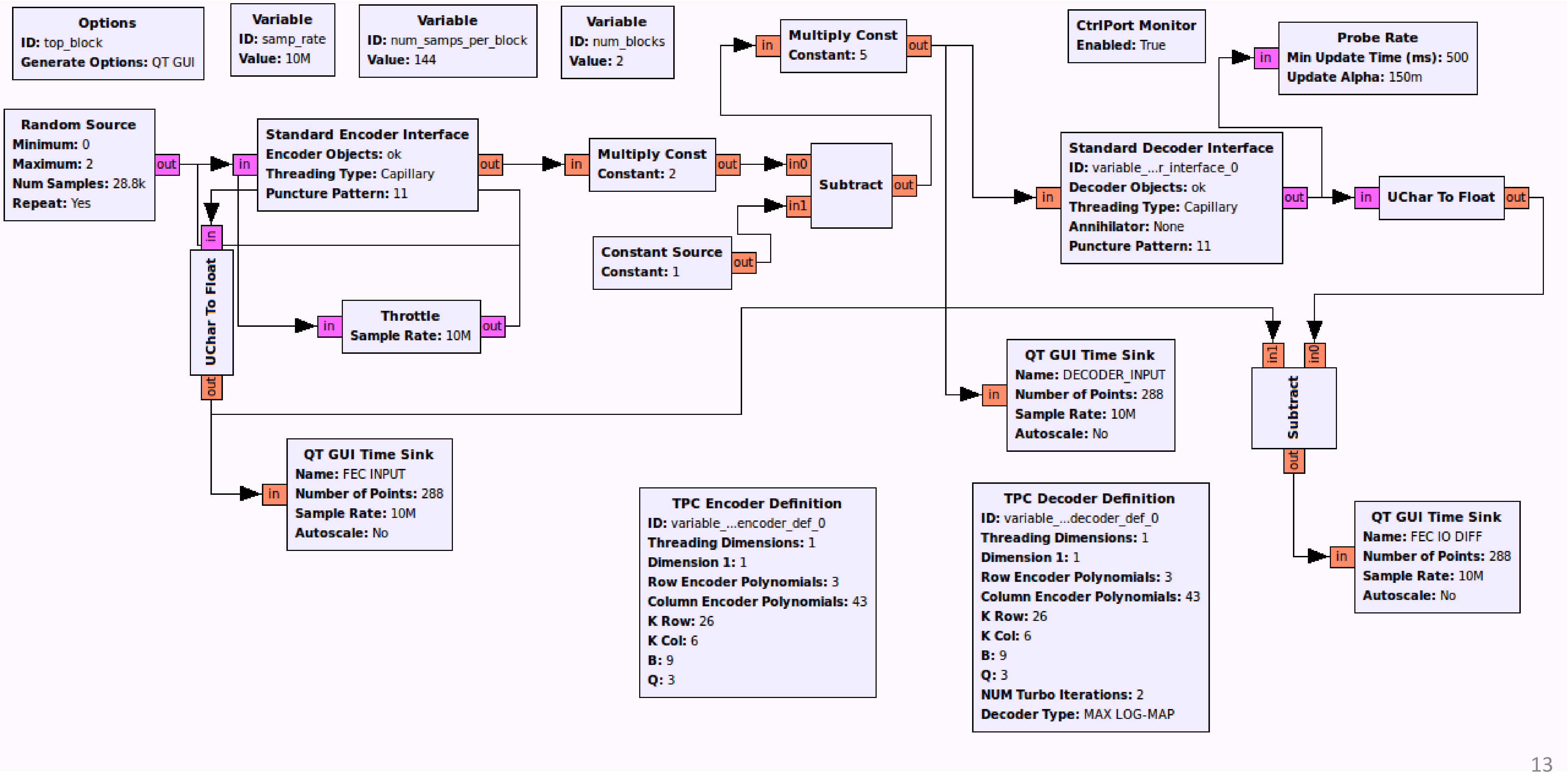
- Early Exit Algorithm
  - The signs of the LLRs at the input and the output of the SISO module are compared.
  - Decoder is stopped if signs agree.
  - Higher SNR environment leads to less iterations of decoder because there is a higher likelihood that the LLRs will agree between input and output.

BER Performance





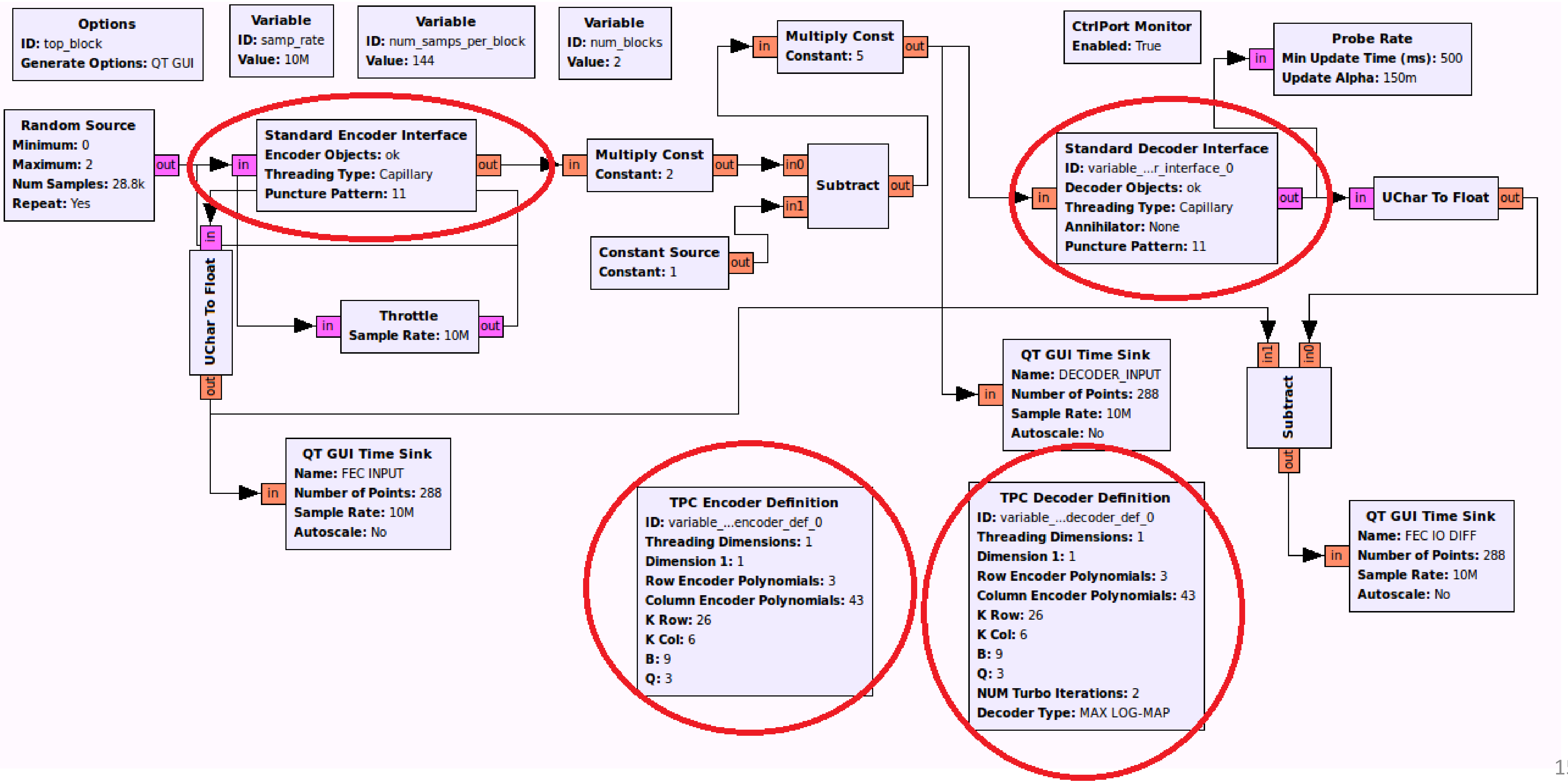
Computational Performance



Computational Performance

Number of Decoder Iterations	Max Log-MAP	Constant Log-MAP	Linear Log-MAP	Log-MAP LUT	Log-MAP C
6	72 kbits/sec	56 kbits/sec	53 kbits/sec	47 kbits/sec	8 kbits/sec
5	87 kbits/sec	67 kbits/sec	64 kbits/sec	56 kbits/sec	10 kbits/sec
4	107 kbits/sec	85 kbits/sec	80 kbits/sec	70 kbits/sec	13 kbits/sec
3	144 kbits/sec	114 kbits/sec	106 kbits/sec	91 kbits/sec	17 kbits/sec
2	216 kbits/sec	169 kbits/sec	157 kbits/sec	139 kbits/sec	26 kbits/sec

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)
- Boutillon, Emmanuel et. al, "Iterative Decoding of Concatenated Convolutional Codes: Implementation Issues," *Proceedings of the IEEE* 2007