Turbo code

in

IEEE 802.20

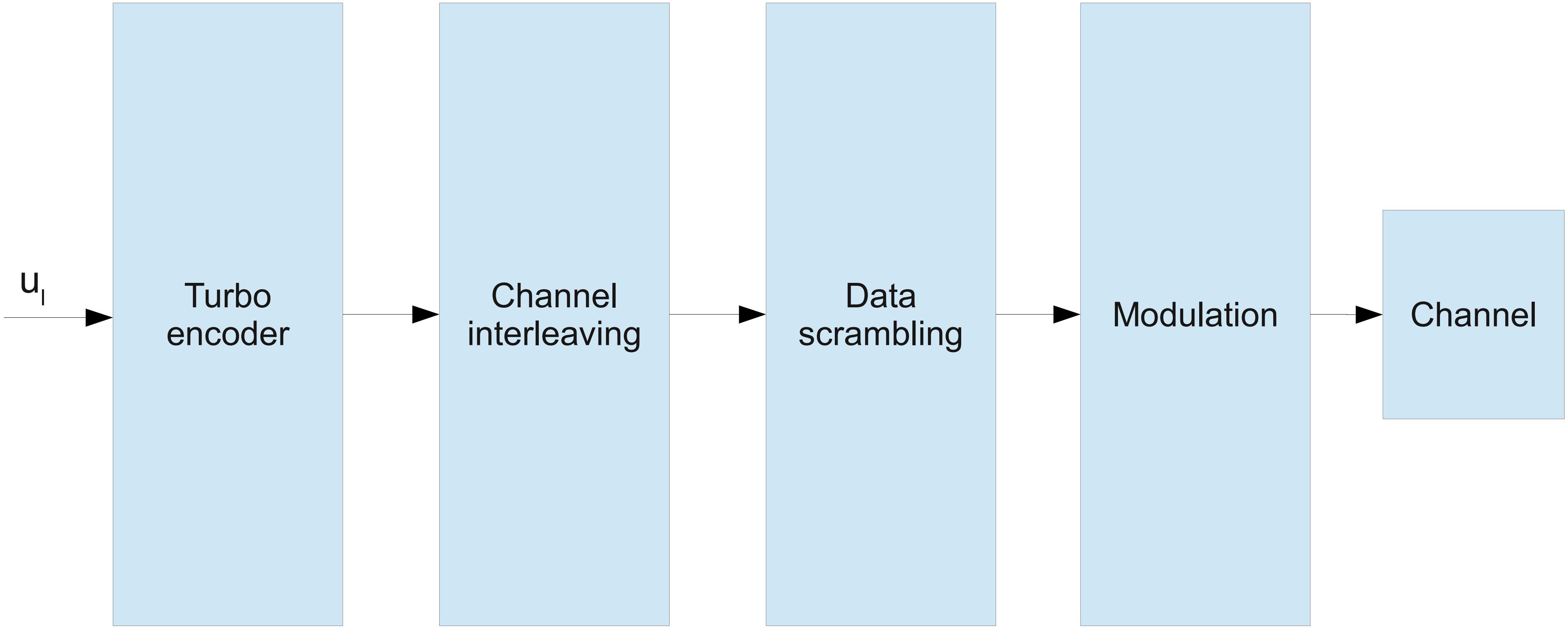
Mobile Broadband Wireless Access

Presented by Daniel Zucchetto

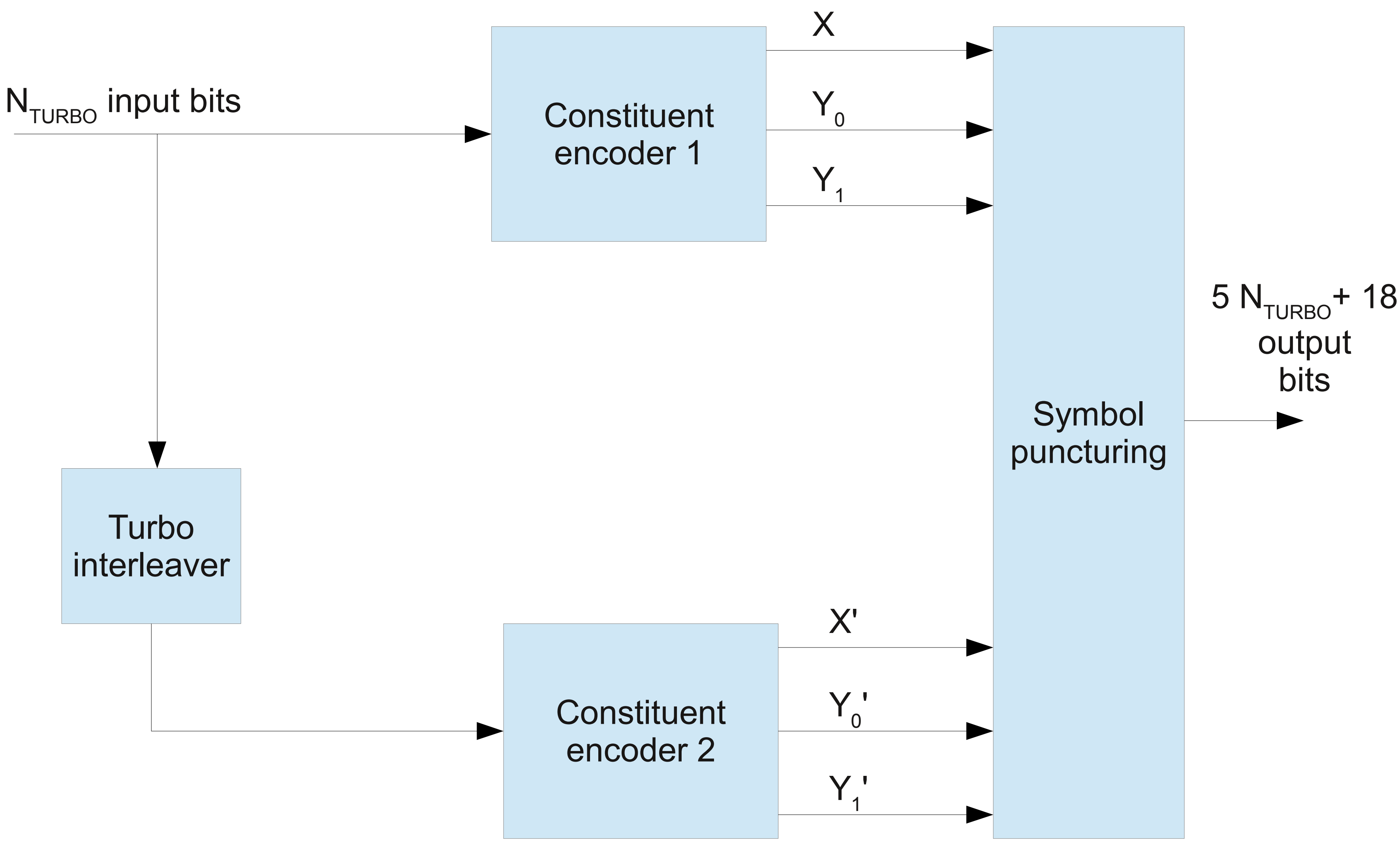
IEEE 802.20

- Boost real-time data transmission rates in wireless metropolitan area networks to speeds that rival DSL and cable connections
- Cell ranges between 1 and 15 km
- Uses licensed frequency bands
- Fully mobile standard
  - supports moving terminals with speed up to 250 km/h
  - handoff support

Transmitter



Turbo encoder



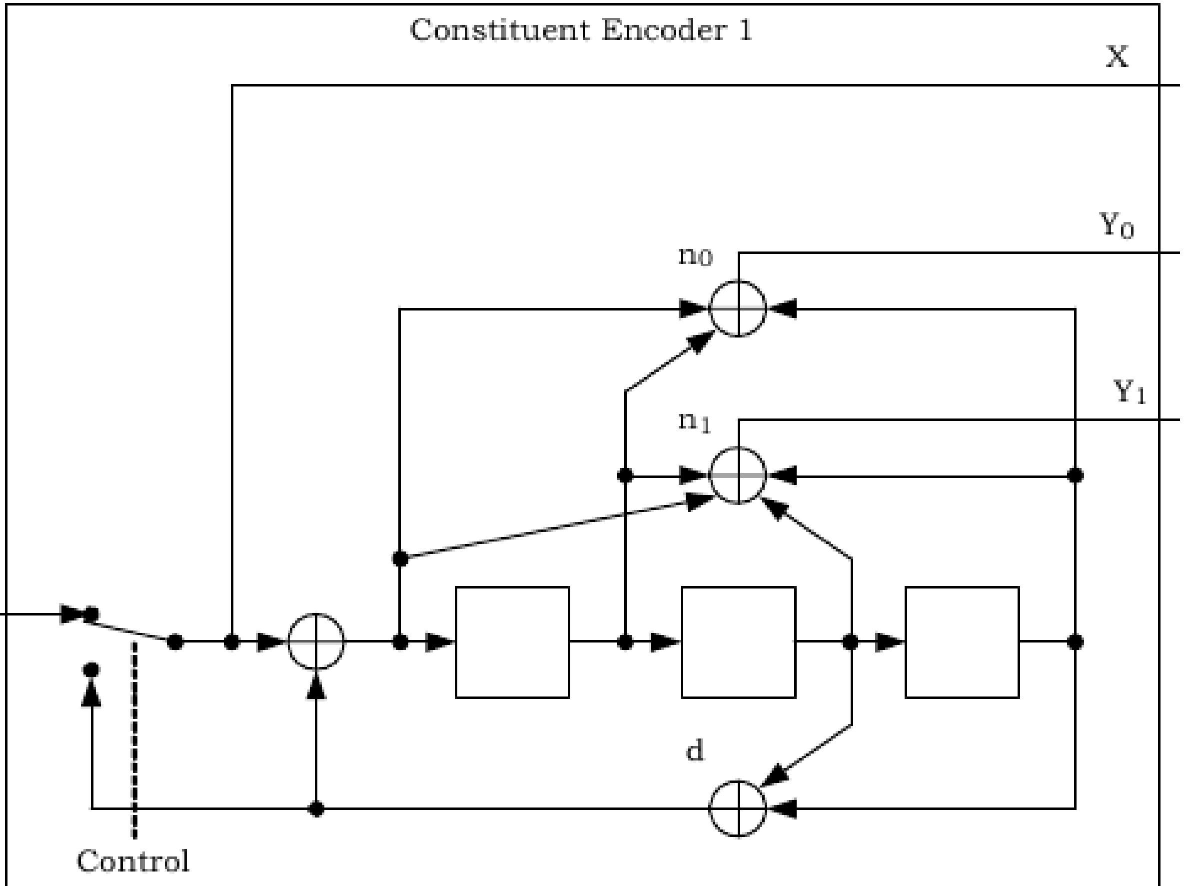
Constituent encoder

$$G(D) = \begin{bmatrix} 1 & \frac{n_0(D)}{d(D)} & \frac{n_1(D)}{d(D)} \end{bmatrix}$$

$$d(D) = 1 + D^2 + D^3$$

$$n_0(D) = 1 + D + D^3$$

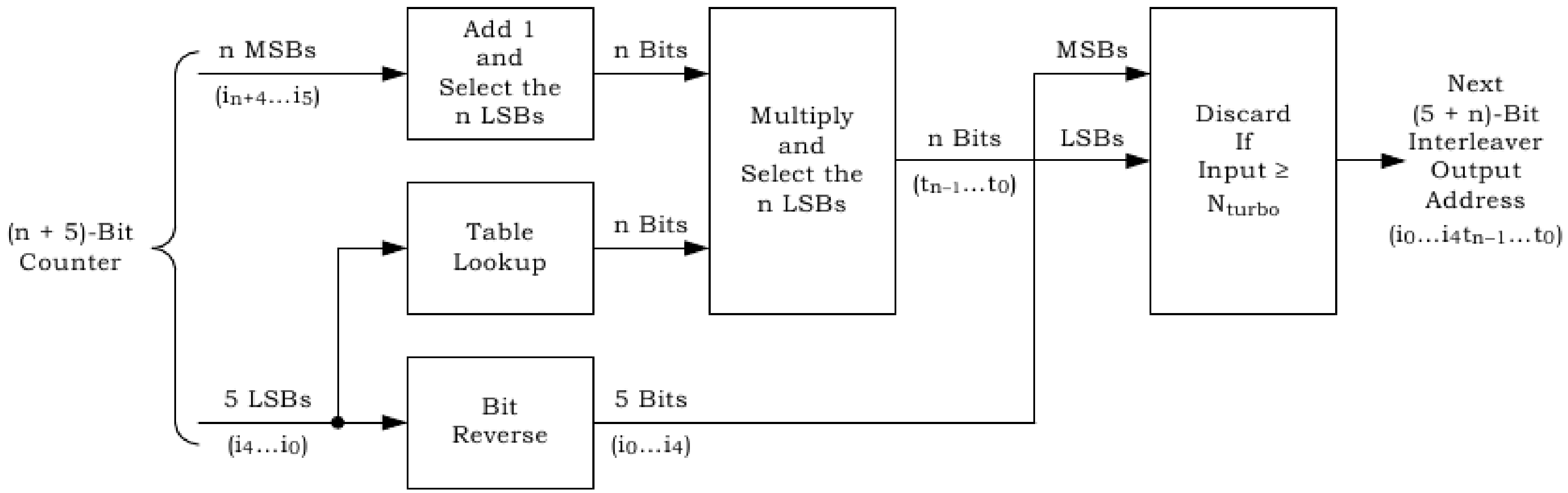
$$n_1(D) = 1 + D + D^2 + D^3$$



Clocked once for each of the N_{turbo} input bit periods with the switch up; then, clocked once for each of the three Constituent Encoder 1 tail bit periods with the switch down; then, not clocked for the three Constituent Encoder 2 tail bit periods.

Turbo interleaver

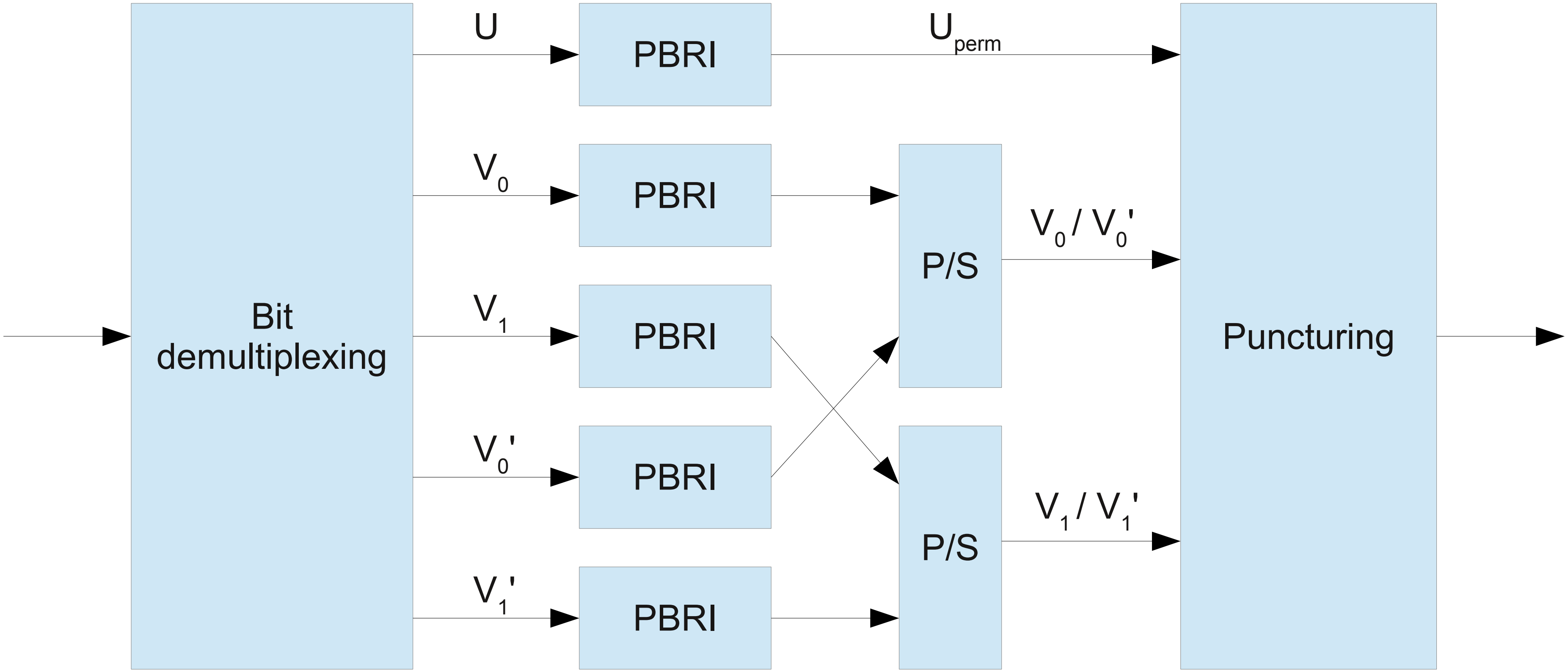
To generate `interleaver_array[]`:



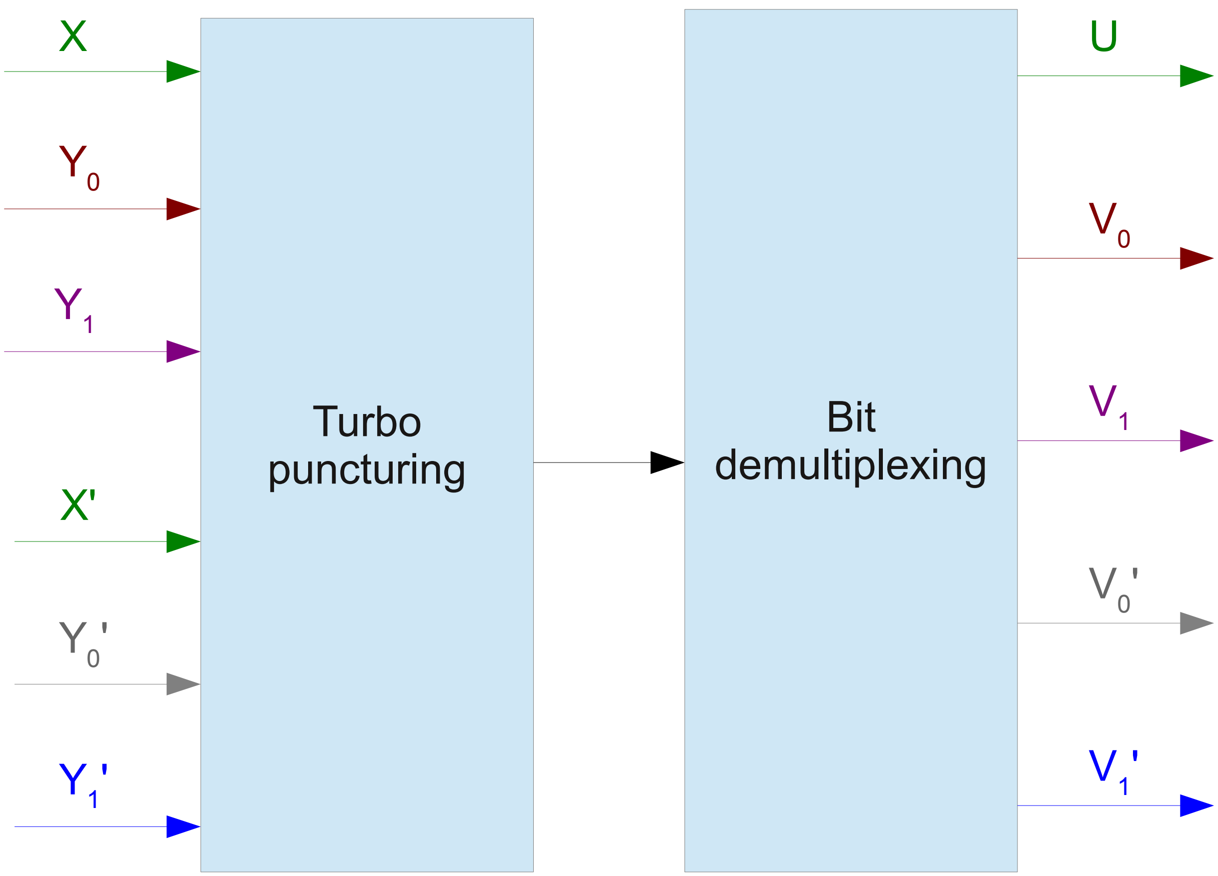
To execute interleaving:

```
output[i]=input[interleaver_array[i]]
```

Channel interleaving



Bit demultiplexing



N.B.: Non-tail bits of X' are discarded

Pruned bit reversal interleaver

The **pruned bit reversal interleaver** generates a permutation y = PBRI(i, M) of the sequence of {0, 1, ..., M – 1} of size M, where y is the output value corresponding to the input value i. The pruned bit reversal interleaver is defined as follows:

1) Determine the pruned bit-reversal interleaver parameter, n, where n is the smallest integer such that $M \leq 2^n$.

2) Initialize counters i and j to 0.

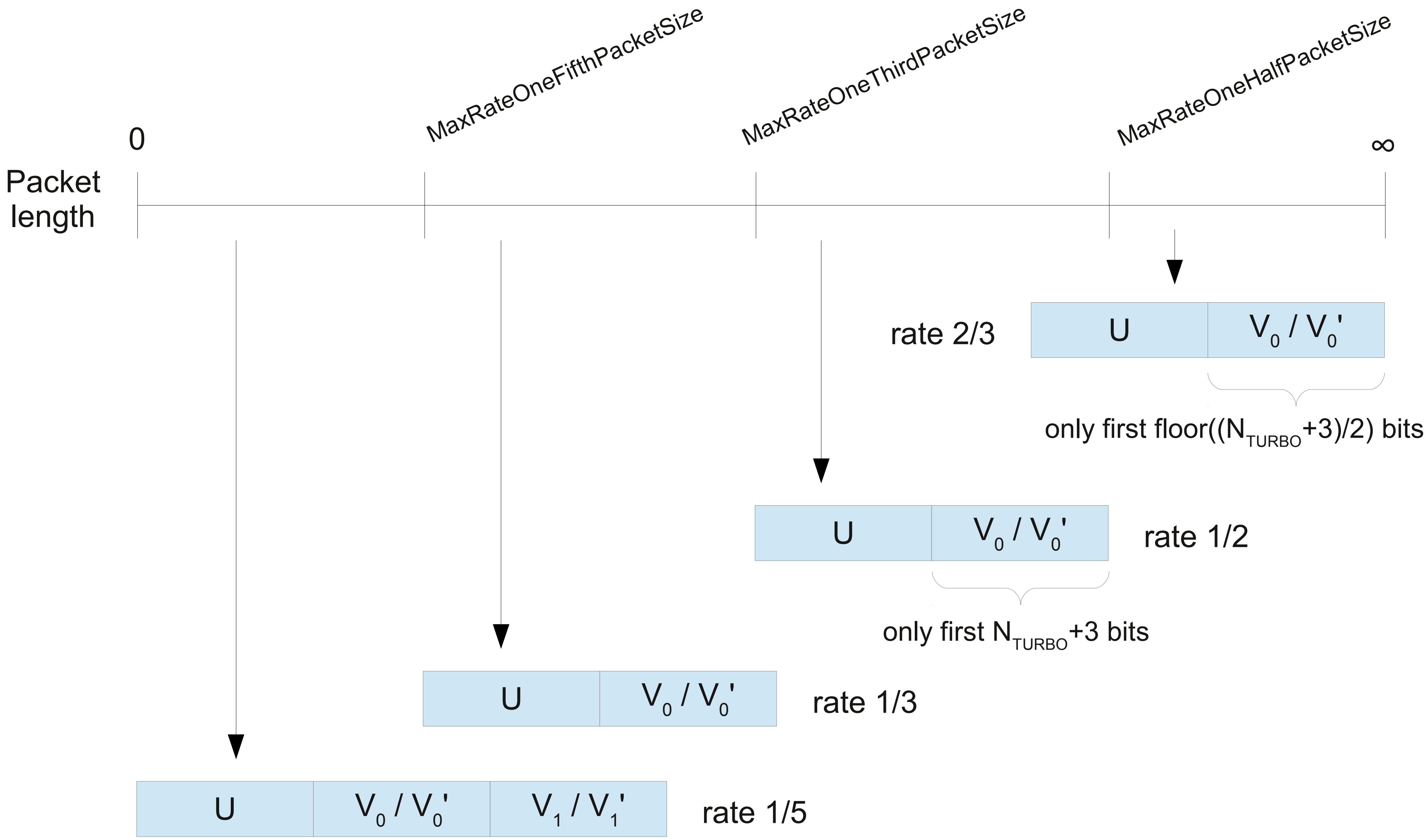
3) Define x as the bit-reversed value of j using an n-bit binary representation. For example, if n = 4 and j = 3, then x = 12.

4) If x < M, set PBRI(i,M) to x and increment the counter i by 1.

5) Increment the counter j by 1.

6) If (i < M) go to 3.

Puncturing

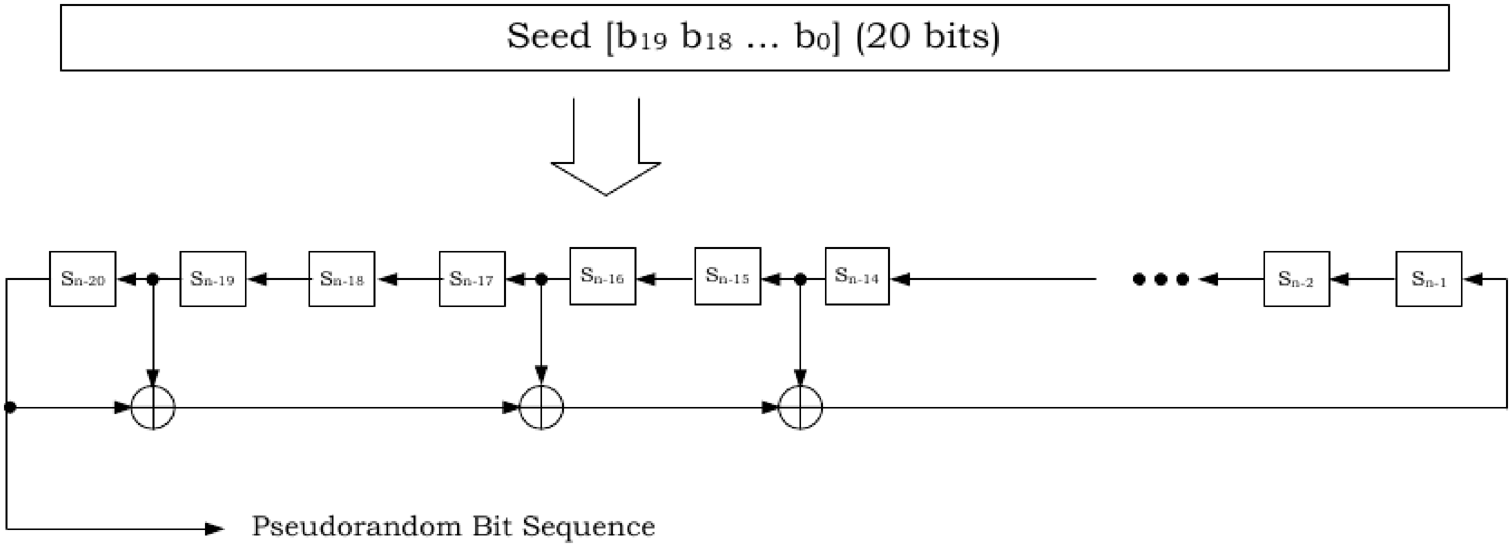


MaxRateOneFifthPacketSize
MaxRateOneThirdPacketSize
MaxRateOneHalfPacketSize
0
∞
Packet length
rate 2/3
U
V_0 / V_0'
only first floor((N_{TURBO}+3)/2) bits
U
V_0 / V_0'
rate 1/2
only first N_{TURBO}+3 bits
U
V_0 / V_0'
rate 1/3
U
V_0 / V_0'
V_1 / V_1'
rate 1/5

Data scrambling

using the *Common real scrambling algorithm*

- Generate a pseudorandom bit sequence of length equal to the packet to send using an algorithm equivalent to the following figure:

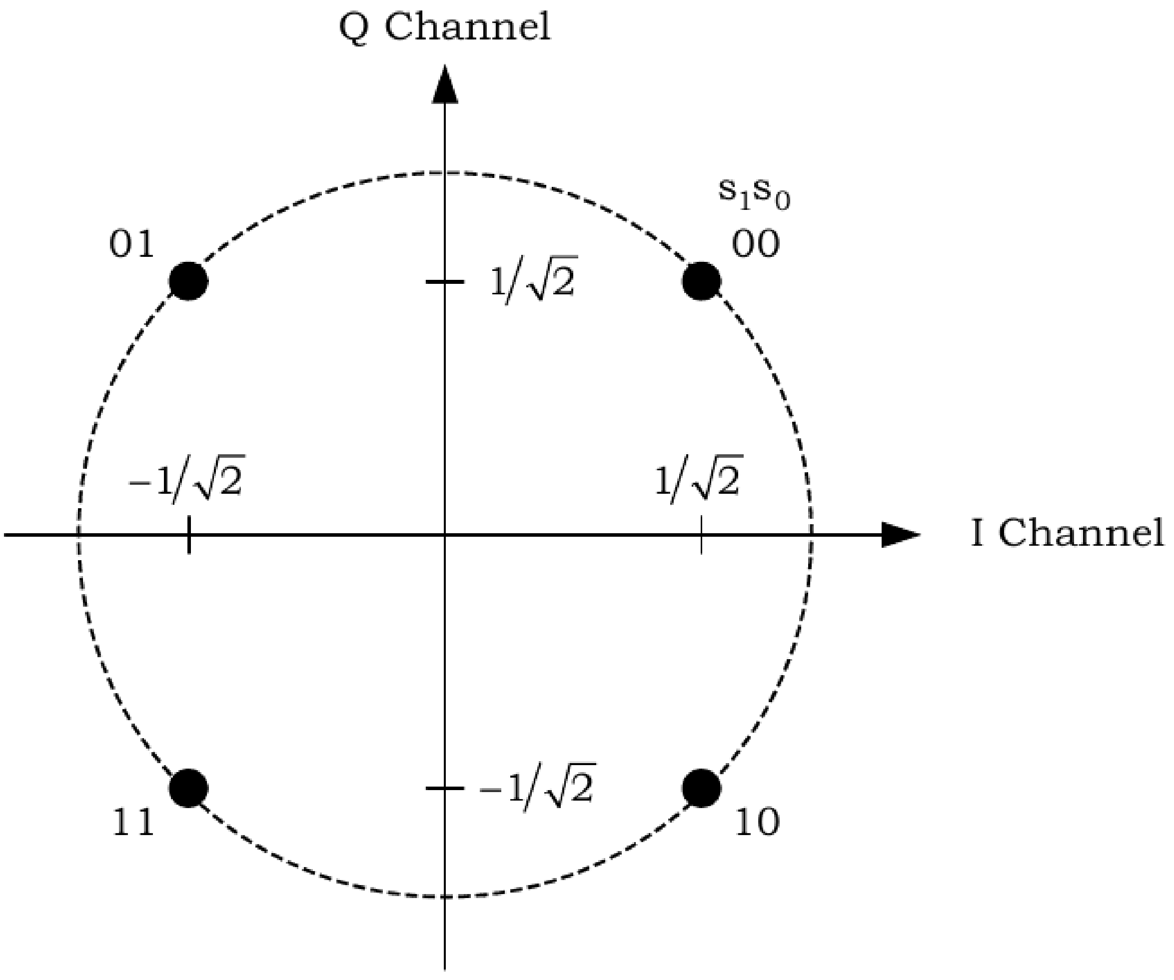


- The i-th entry r(i) in the real scrambling sequence shall be generated from a bit denoted by $b_r(i)$, using the mapping $r(i) = (1 – 2\, b_r(i))$.
- Flip the bit in the input sequence if the corrsponding value in the pseudorandom bit sequence is equal to -1

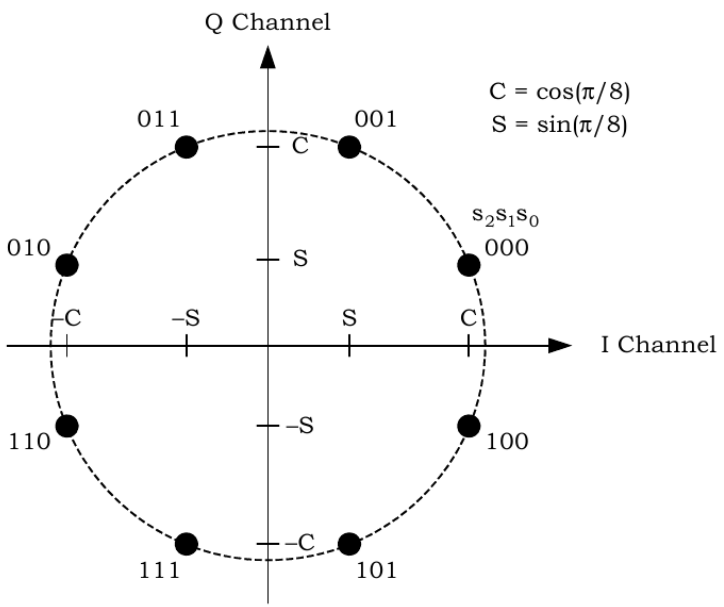
4-QAM Modulation

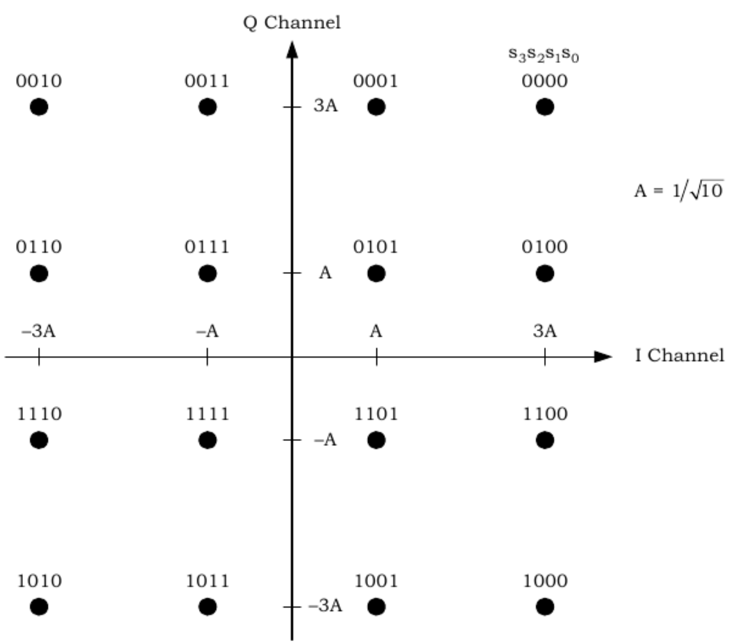
$$\Gamma = R\log_2(M)\frac{E_b}{N_0}$$

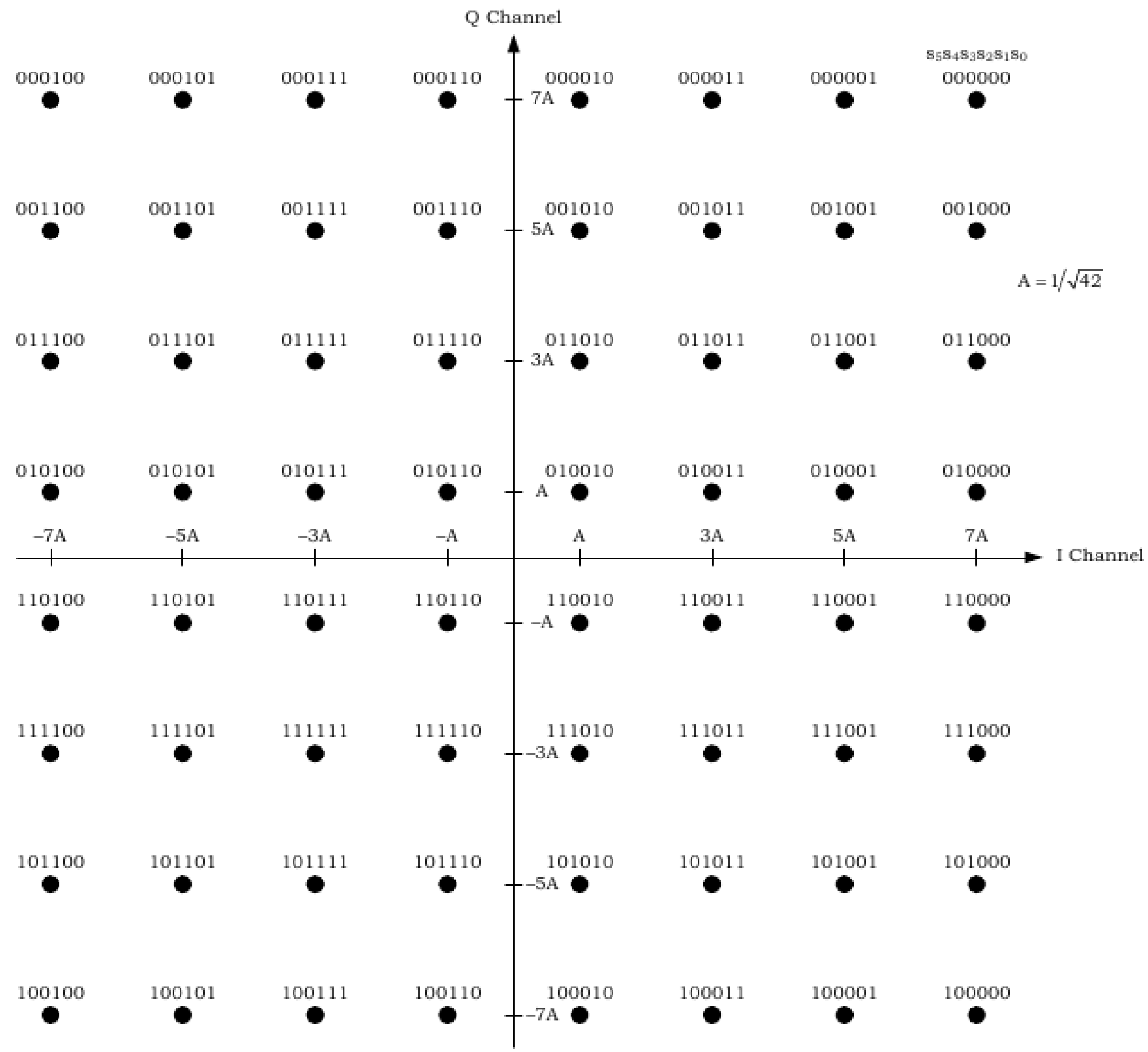
$$P_{bit} = Q(\sqrt{\Gamma}) = Q\left(\sqrt{2\frac{E_b}{N_0}}\right)$$



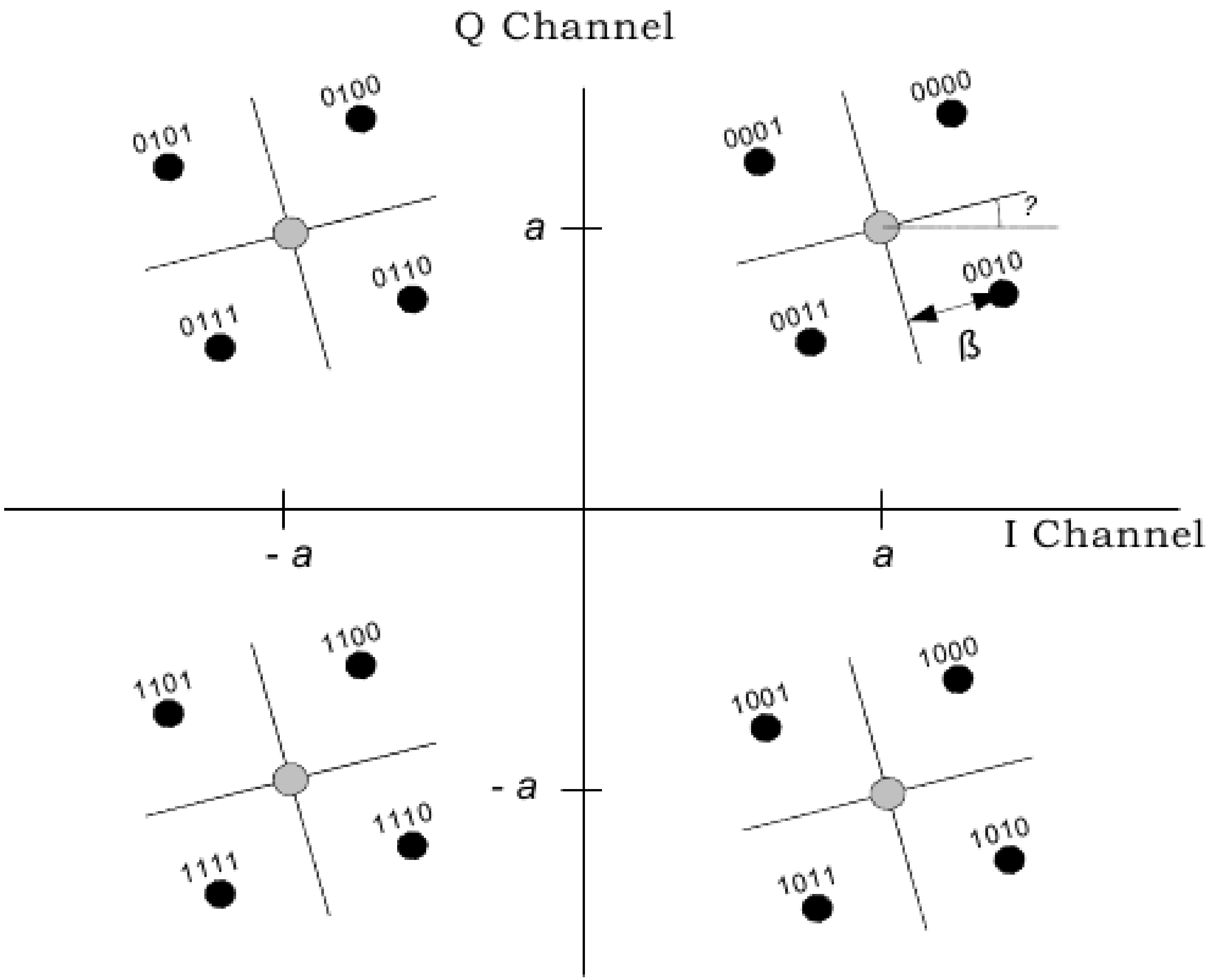
Other modulations (8-PSK, 16-QAM, 64-QAM)





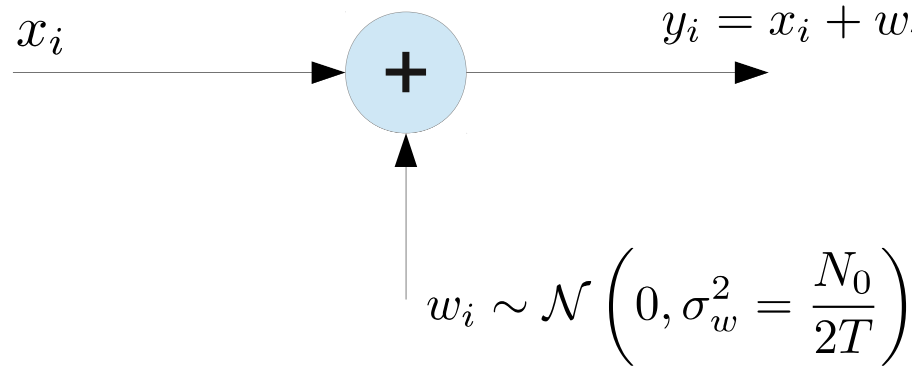


Hierarchical modulation

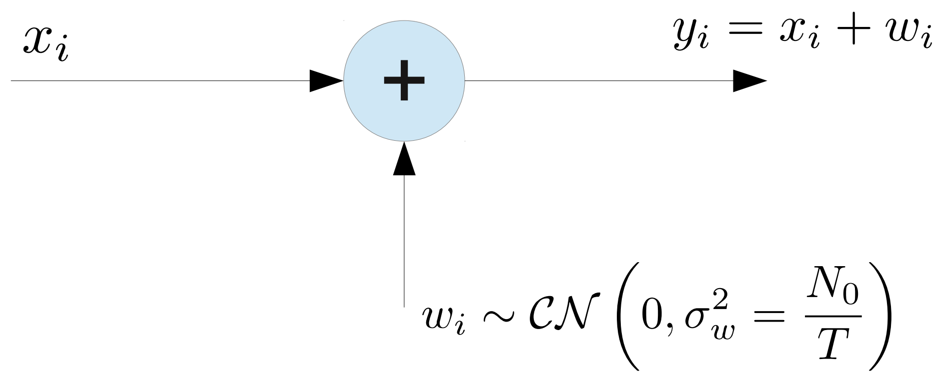


Memoryless AWGN Channel

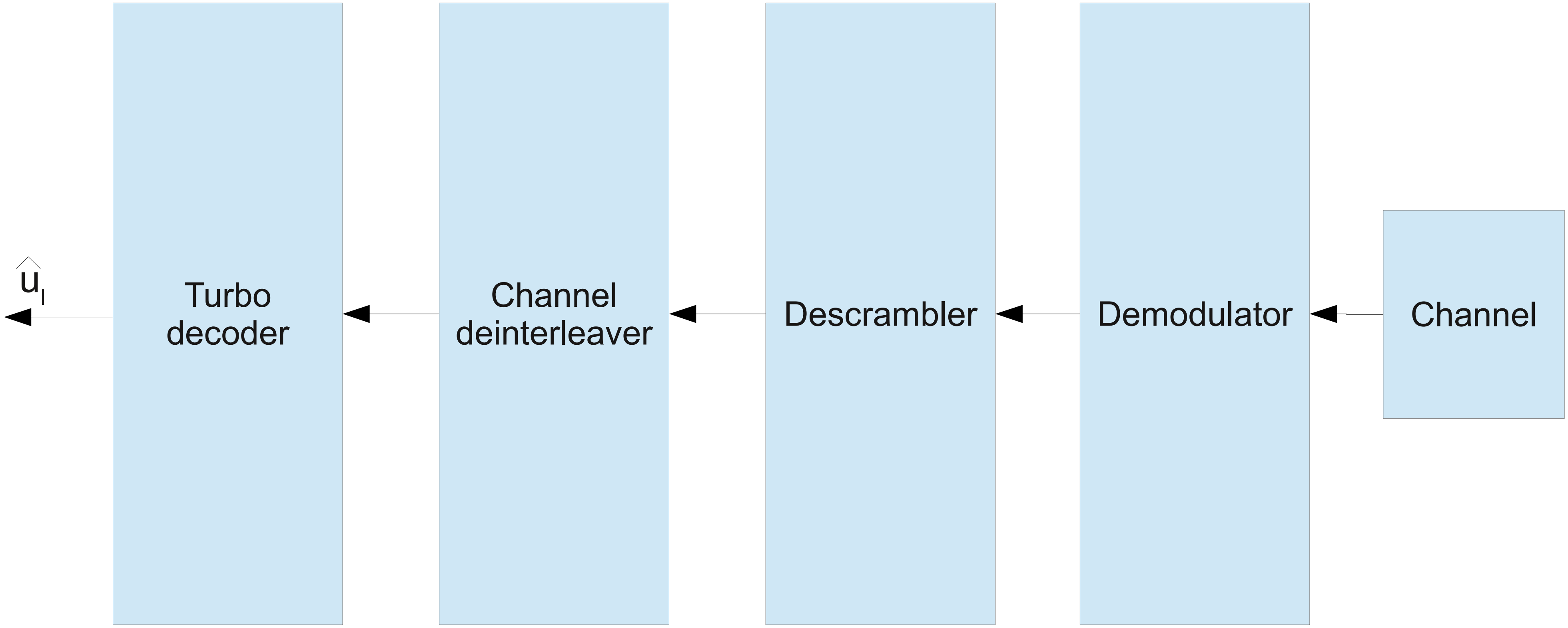
Baseband



Passband (QAM)



Receiver



$\hat{u}_l$
Turbo decoder
Channel deinterleaver
Descrambler
Demodulator
Channel

Demodulator

input symbol to the modulator $\nu_i = [\nu_i^0 \;\; \nu_i^1], \nu_i^k \in \{0, 1\}$

modulated symbol $x_i = \mu(\nu_i), x_i \in \mathcal{X}$

received symbol y_i

LLR metrics (with the assumption of equally likely input symbols)

$$\lambda_i^k = \log \frac{P(\nu_i^k = 1 | y_i)}{P(\nu_i^k = 0 | y_i)} = \log \frac{\sum_{x_i \in \mathcal{X}_1^k} P(y_i | x_i) P(x_i)}{\sum_{x_i \in \mathcal{X}_0^k} P(y_i | x_i) P(x_i)} =$$

$$= \log \left(\frac{\sum_{x_i \in \mathcal{X}_1^k} \frac{1}{\sqrt{2\pi\sigma_w^2}} \exp\left(-\frac{|y_i - x_i|^2}{2\sigma_w^2}\right) P(x_i)}{\sum_{x_i \in \mathcal{X}_0^k} \frac{1}{\sqrt{2\pi\sigma_w^2}} \exp\left(-\frac{|y_i - x_i|^2}{2\sigma_w^2}\right) P(x_i)} \right) = \log \left(\frac{\sum_{x_i \in \mathcal{X}_1^k} \exp\left(-\frac{|y_i - x_i|^2}{2\sigma_w^2}\right)}{\sum_{x_i \in \mathcal{X}_0^k} \exp\left(-\frac{|y_i - x_i|^2}{2\sigma_w^2}\right)} \right)$$

with $\mathcal{X}_b^k = \{x \in \mathcal{X} | \nu_i^k = b\}$

Demodulator (C++ code)

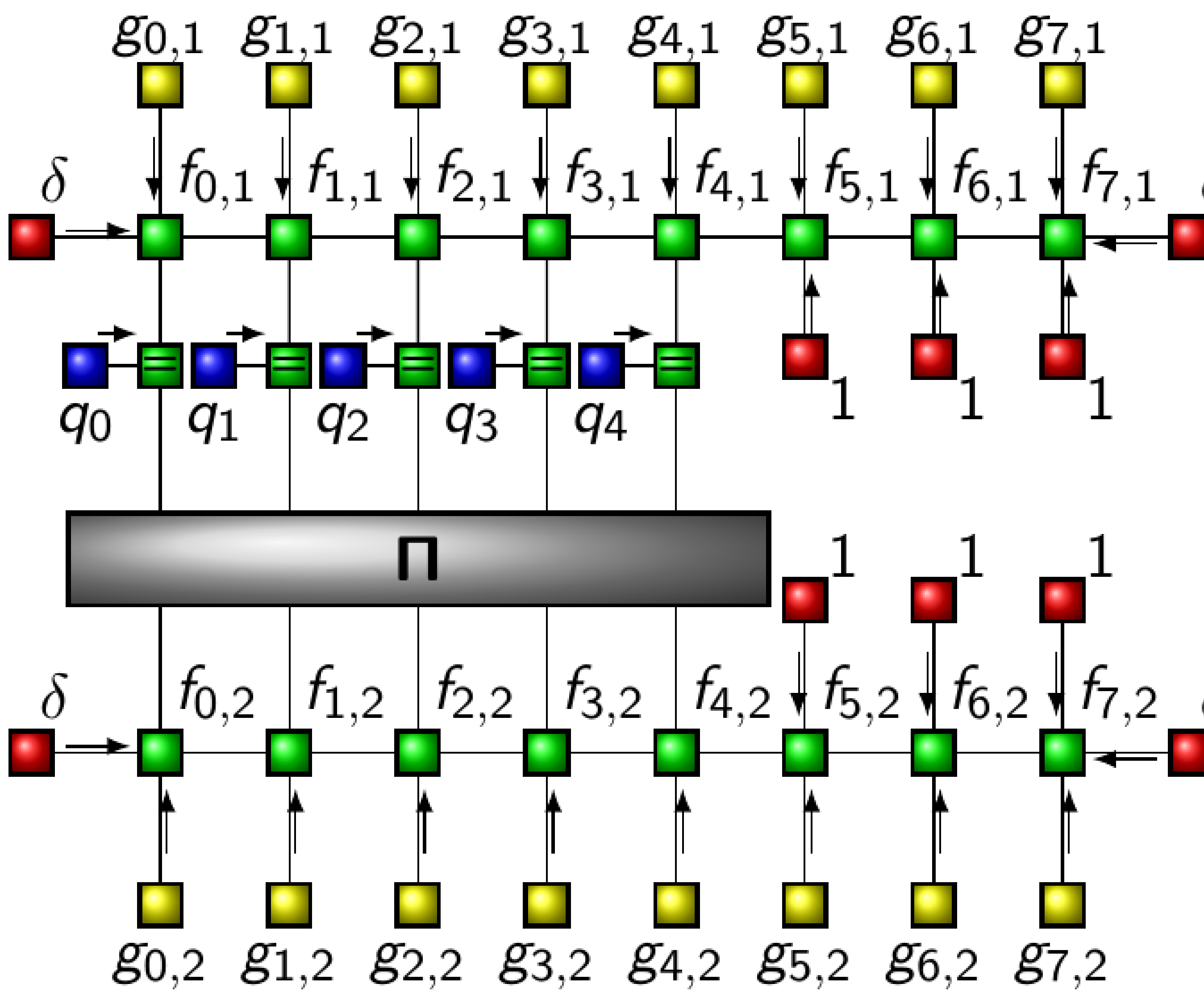
```
int totBit=0;
for(int sym=0;sym<symbolLength && totBit<usefulLength;sym++) // symbols index
   for(int bit=0;bit<2 && totBit<usefulLength;bit++){ // bits in symbol index
      double loglike0=-infinity;
      double loglike1=-infinity;

      // span every possible symbol
      for(unsigned char i=0;i<2;i++)
         for(unsigned char j=0;j<2;j++){
            if(bit==0&&i==0 || bit==1&&j==0)
              loglike0=log_sum(loglike0,distance(sigma_sq,inI[sym],inQ[sym],i,j));
            else if(bit==0&&i==1 || bit==1&&j==1)
              loglike1=log_sum(loglike1,distance(sigma_sq,inI[sym],inQ[sym],i,j));
         }

      out[totBit++]=loglike1-loglike0;
   }
```

Turbo decoder

Constituent convolutional code 1

Turbo interleaver

Constituent convolutional code 2



BCJR

Sum-product decoder $$\hat{u}_l = \arg\max_{u_l} \sum_{\mathbf{c},\mathbf{s},\mathbf{u}\sim u_l} \mathcal{B}(\mathbf{c},\mathbf{u},\mathbf{s}) \prod_i p(r_i|c_i) \prod_j p(u_j)$$

$$q_l(u_l) = \begin{cases} p(u_l) & l < \mu \\ 0.5 & l \geq \mu \end{cases}$$

$$f_l(\mathbf{s}_{l+1}, \mathbf{s}_l, \mathbf{y}_l, u_l) = \delta_{\mathbf{s}_{l+1}, \mathcal{S}(\mathbf{s}_l, u_l)} \delta_{\mathbf{y}_l, \mathcal{O}(\mathbf{s}_l, u_l)}$$

$$g_l(\mathbf{y}_l) = p(\mathbf{r}_l|\mathbf{y}_l) = \exp\left(-\frac{||\mathbf{r}_l - L(\mathbf{y}_l)||^2}{2\sigma_w^2}\right)$$

Backward messages $$B_{l-1}(\mathbf{s}_l) = \sum_{u_l} q_l(u_l) B_l(\mathcal{S}(\mathbf{s}_l, u_l)) g_l(\mathcal{O}(\mathbf{s}_l, u_l))$$

intialization: $$B_{\mu+\nu-1}(\mathbf{s}_{\mu+\nu}) = \delta_{\mathbf{s}_{\mu+\nu}, \mathbf{0}}$$

BCJR

Forward messages $$F_{l+1}(\mathbf{s}_{l+1}) = \sum_{\mathbf{s}_l, u_l} q_l(u_l) F_l(\mathbf{s}_l) g_l(\mathcal{O}(\mathbf{s}_l, u_l)) \delta_{\mathbf{s}_{l+1}, \mathcal{S}(u_l, \mathbf{s}_l)}$$

initialization: $F_0(\mathbf{s}_0) = \delta_{\mathbf{s}_0, \mathbf{0}}$

Extrinsic information $$E_l(u_l) = \sum_{\mathbf{s}_l} F_l(\mathbf{s}_l) B_l(\mathcal{S}(\mathbf{s}_l, u_l)) g_l(\mathcal{O}(\mathbf{s}_l, u_l))$$

Marginalization $$\hat{u}_l = \arg\max_{u_l} q_l(u_l) E_l(u_l)$$

Turbo decoder (C code)

```
initialize_q_array(q1);
initialize_q_array(q2);
int* interleaver_array=new int[mu];
interleaving_procedure(interleaver_array,mu);
depuncture(r_serial,r_l,r1,r2,u_l);

for(int iteration=0;iteration<MAX_ITERATIONS;iteration++){
   q_array=q1;
   bcjr(r1,E_matrix1);
   marginalization(q1,E_matrix1,u_hat);
   propagate(interleaver_array,E_matrix1,q2,true);
   q_array=q2;
   bcjr(r2,E_matrix2);
   propagate(interleaver_array,q1,E_matrix2,false);
}
```

BCJR decoder (C code)

```
for(int l=1;l<mu+NU;l++) {
     double tot=-infinity;
     for(unsigned char s=0;s<NUM_STATES;s++) {
         F_matrix[l][s]=F(l,s,r);
         tot=log_sum(tot,F_matrix[l][s]);
     }
     for(unsigned char s=0;s<NUM_STATES;s++) F_matrix[l][s]-=tot;
}
for(int l=mu+NU-2;l>=0;l--){
     double tot=-infinity;
     for(unsigned char s=0;s<NUM_STATES;s++) {
         B_matrix[l][s]=B(l,s,r);
         tot=log_sum(tot,B_matrix[l][s]);
     }
     for(unsigned char s=0;s<NUM_STATES;s++) B_matrix[l][s]-=tot;
}
double epsilon;
for(int l=0;l<mu;l++){
   E_matrix[l][0]=E(l,0,r);
   E_matrix[l][1]=E(l,1,r);
   epsilon=log_sum(E_matrix[l][0],E_matrix[l][1]);
   E_matrix[l][0]-=epsilon;
   E_matrix[l][1]-=epsilon;
}
```

Implementation aspects

The use of the logaritmic version provides simplifications and less problems due to finite numeric precision.

But the sum of two values in linear domain corresponds to

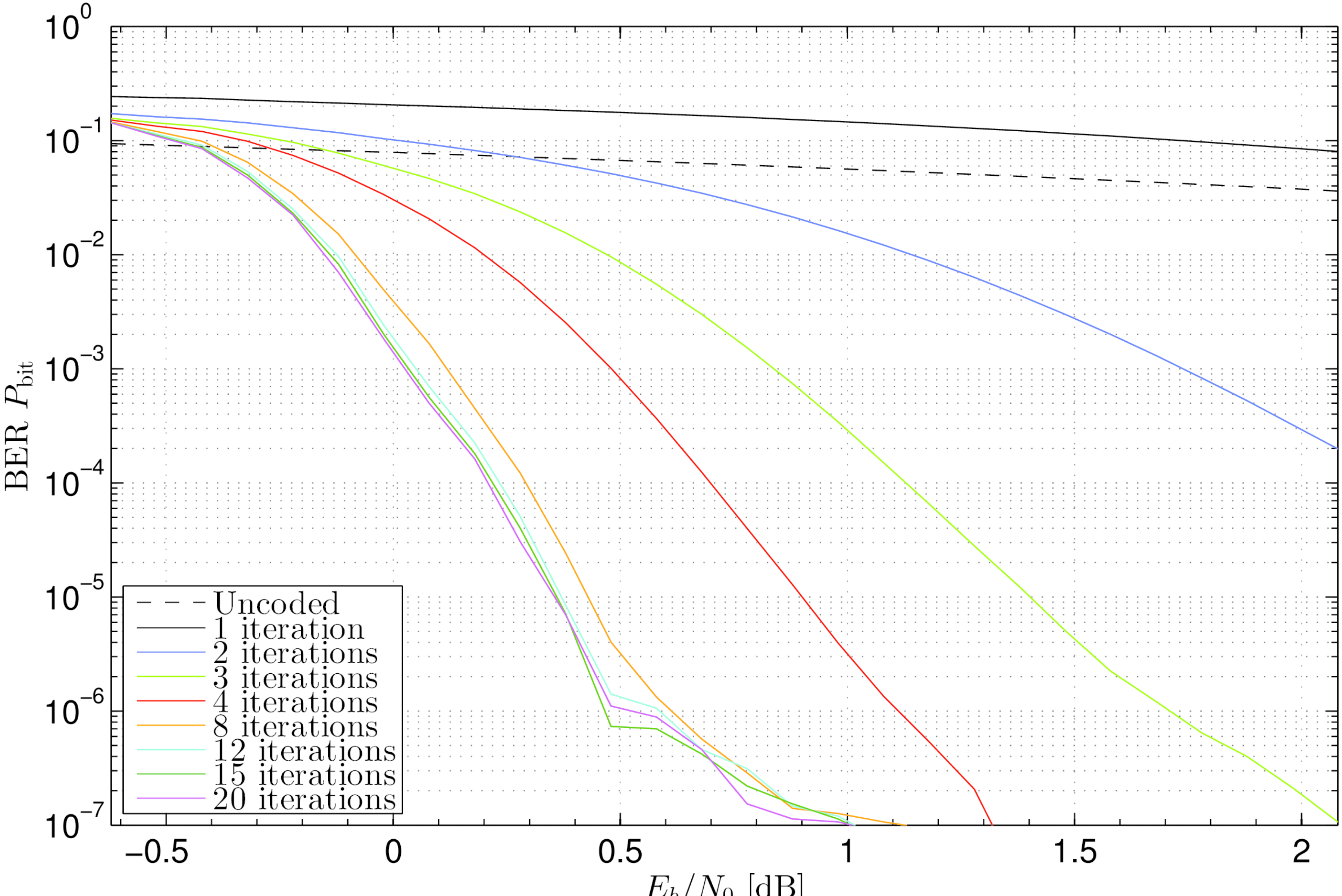
$$a + b \rightarrow \log(e^{a'} + e^{b'}) = \max(a', b') + \log(1 + e^{-|a'-b'|})$$

with $a' = \log(a), b' = \log(b)$

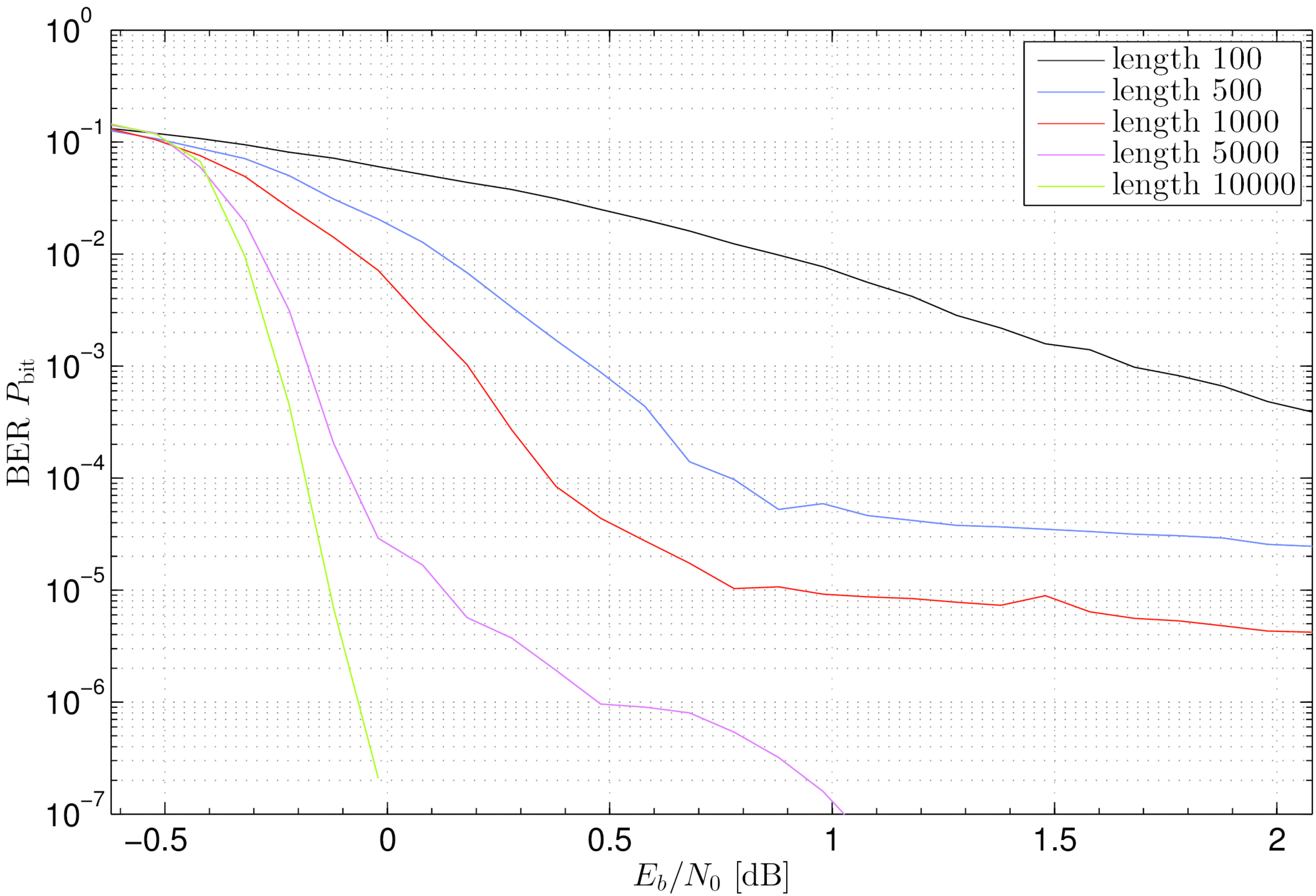
It's possible to approximate $\log(1+e^{-x})$ via a lookup table or a piecewise linear approximation. Using only max(a',b') gives suboptimal results. The approximation used in the implementation is

$$\log(1 + e^{-x}) \simeq \begin{cases} -0.28311x + 0.69314 & x < 2 \\ -0.05439x + 0.23571 & 2 \leq x < 4.3337 \\ 0 & x \geq 4.3337 \end{cases}$$

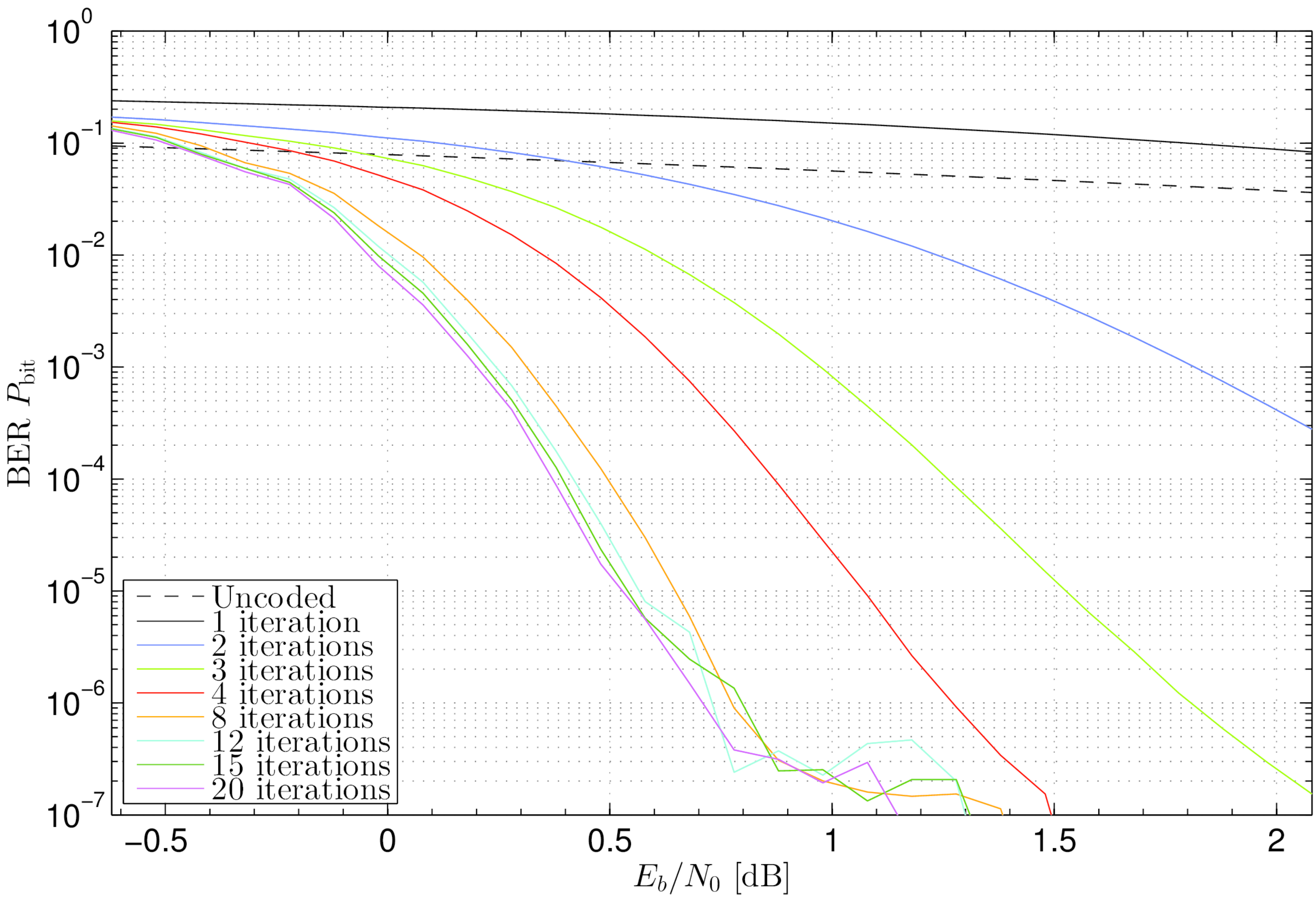
Turbo code with 2-PAM modulation
Impact of number of iterations



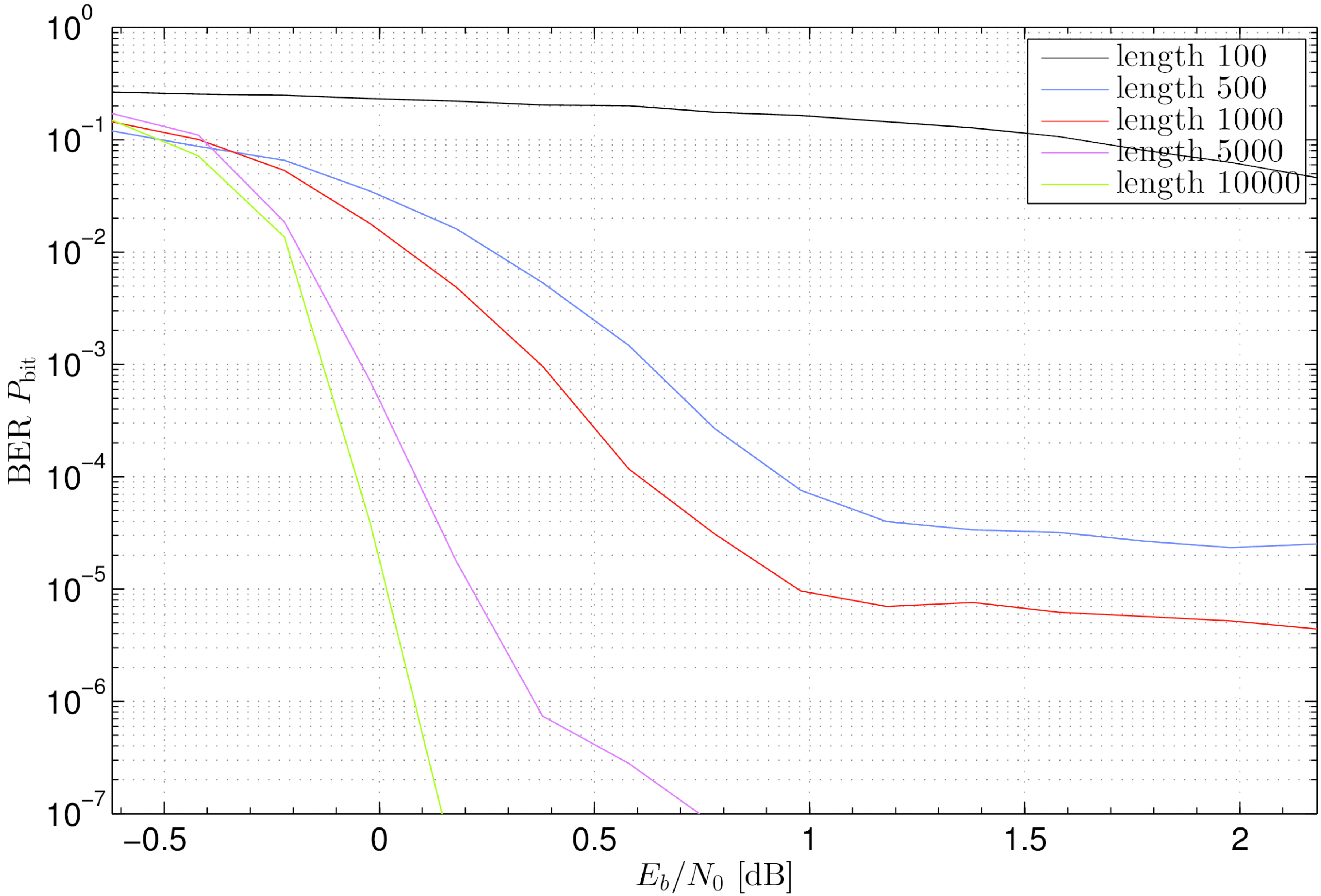
Turbo code with 2-PAM modulation
Impact of word length



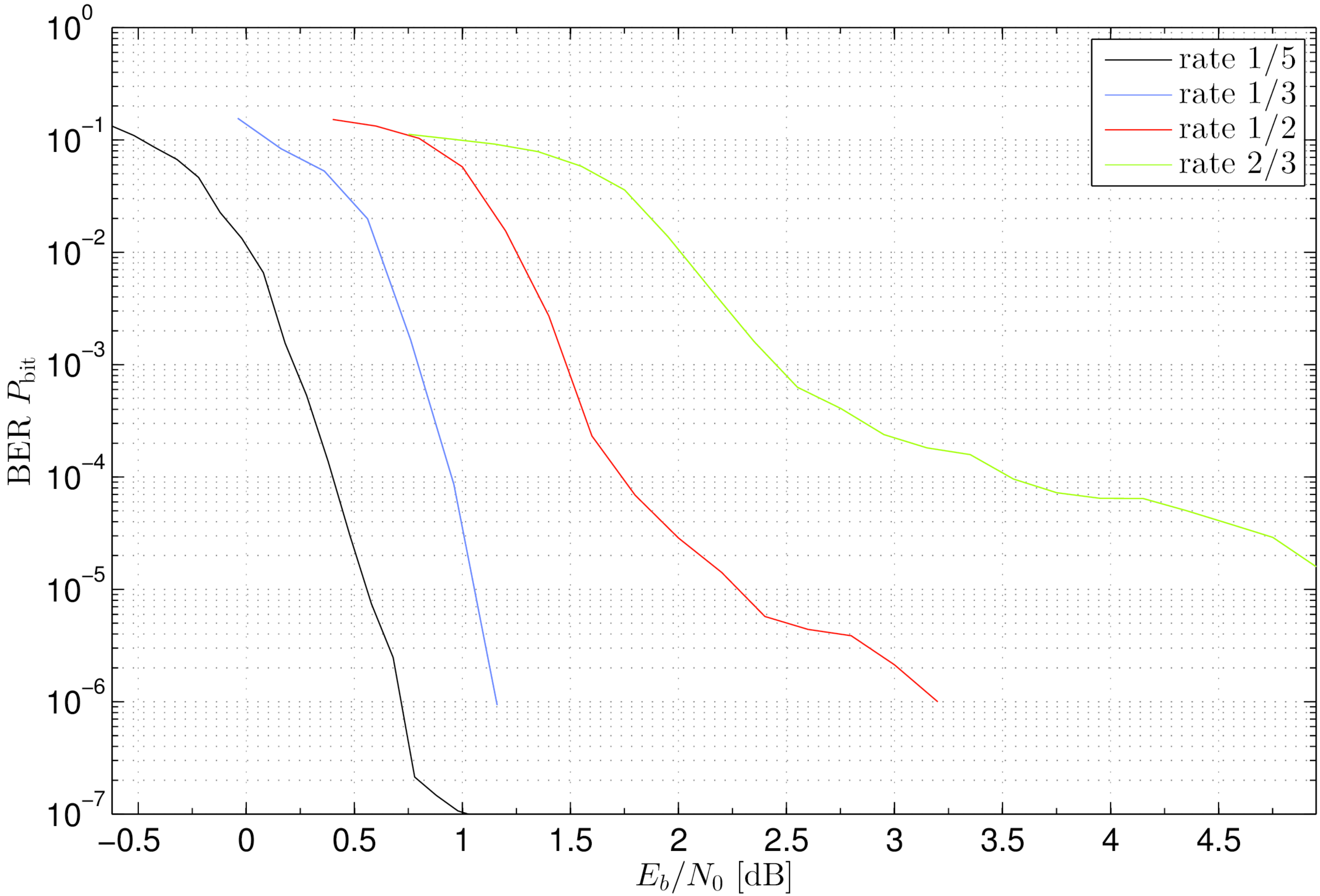
BICM (turbo code) with 4-QAM modulation
Impact of number of iterations



BICM (turbo code) with 4-QAM modulation
Impact of word length



BICM (turbo code) with 4-QAM modulation
Impact of puncturing



Hard decoding vs Soft decoding