Application Report
SPRA974 − November 2003

TMS320C6416 Coprocessors and Bit Error Rates

Sebastien Tomas, Mattias Ahnoff,
Patrick Geremia, Pierre Bertrand

Wireless Infrastructure

ABSTRACT

The turbo and viterbi coprocessors (TCP/VCP) are programmable peripherals used to decode IS2000/3GPP turbo/viterbi codes. They are integrated into Texas Instruments TMS320C6416 digital signal processor (DSP). Turbo and viterbi decoders lie at the heart of all of the third-generation (3G) wireless standards. Their usage in 3G systems, meets the tough bit-error-rate requirements and low signal-to-noise ratios (SNRs).

This application report describes the methodology and assumptions used to generate TMS320C6416 TCP/VCP bit error rate curves. It also gives details on the channel model, the resolution and the normalization of the soft decisions, and examples about their efficient implementation on the TMS320C6416 DSP. The resulting TCP/VCP bit error rate curves on some 3GPP frames are provided.

Contents

List of Figures

List of Tables

1 Introduction

Forward-error correction (FEC), also known as channel codeing, is used to improve the capacity of a channel by adding redundant information to the data being transmitted. Viterbi and turbo coding are FEC techniques that are used in all of the third-generation (3G) wireless standards.

This application report describes the methodology and assumptions used to generate TMS320C6416 TCP/VCP bit error rate (BER) curves. The transmitted signal is corrupted by additive white Gaussian noise (AWGN). Details are given on the channel model, the resolution and the normalization of the soft decisions, and examples about their efficient implementation on the TMS320C6416 DSP. The resulting TCP/VCP bit error rate curves on some 3GPP frames are provided.

For details on the TMS320C6416 Viterbi and Turbo coprocessors, please refer to the application notes listed in the references section.

Note that the TMS320C6416 TCP Coprocessor processing unit implements the MAX*−LOG−MAP approximation of the BCJR algorithm(6). This MAP decoder with a small lookup table gives better BER results.

2 BER Curve Methodology and Assumptions

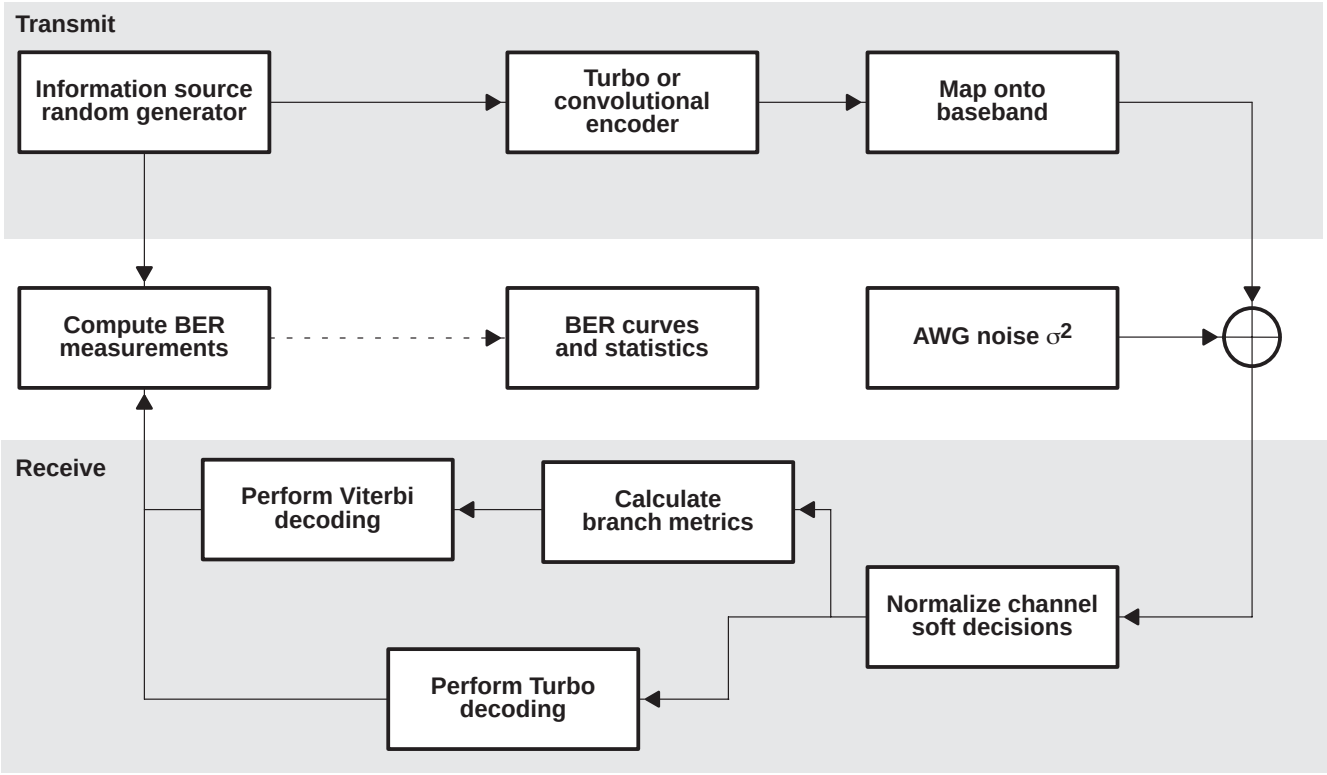


Figure 1. Communication Channel and TCP/VCP BER Computation

2.1 Simulation of a Communication Channel Using a Viterbi Decoder

The steps involved in simulating a communication channel using convolutional encoding and Viterbi decoding are as follows:

- Generate the binary data bits (information sequence) to be transmitted through the channel.
- Convolutional encode the information sequence in channel symbols.
- Map the one/zero channel symbols onto an antipodal baseband signal (0 −> a and 1 −> −a, a is the carrier amplitude), producing transmitted channel symbols.
- Add AWG noise to the transmitted channel symbols to generate received channel symbols (soft decisions).
- Normalize channel soft decisions to the resolution required by the VCP.
- Combine normalized channel soft decisions to generate branch metrics inputs and perform Viterbi decoding using the TMS320C6416 VCP coprocessor.

2.2 Simulation of a Communication Channel Using a Turbo Decoder

The steps involved in simulating a communication channel using turbo encoding and turbo decoding are as follows:

- Generate the binary data bits (information sequence) to be transmitted through the channel.
- Generate the turbo interleaver table; turbo interleave and turbo encode the information sequence in channel symbols.
- Map the one/zero channel symbols onto an antipodal baseband signal (0 −> a and 1 −> −a, a is the carrier amplitude), producing transmitted channel symbols.
- Add AWG Noise to the transmitted channel symbols to generate received channel symbols (soft decisions).
- Normalize channel soft decisions (systems and parities) to generate input buffers to the TCP coprocessor.
- Generate the turbo interleaver table and perform turbo decoding using the TMS320C6416 TCP coprocessor.

The minimum set of functions has been chosen to simulate a communication channel. In a 3G system, you may want to add more symbol rate functionalities such as interleaving or puncturing algorithms in the simulation. Such algorithms may have an influence on the bit error rate measurements as they may add extra bit errors.

2.3 Symbol Mapping at Transmission

The output of the encoder is a binary sequence …010011001b….

Assuming a binary phase shift keying (BPSK) modulation, a ‘1’ channel bit is transmitted at a level of –1V, and a ‘0’ channel bit is transmitted at a level of 1V.

The 1/−1V levels can be represented on 2nd complement signed 8 bits word with the following resolution: SIII.FFFF (1 −> 0x10 and –1 −> 0xF0) (S = sign bit, I = integer, F = fractional bit).

An efficient implementation based on the TMS320C6416 instruction set:

```
unsigned int words,i,xbits;
unsigned int inputWord;

    words = length>>5;
    xbits = length & 0x1F;

    for(i=0;i<words;i++)  {
        inputWord=~*in++;
        *out++=_sub4(_xpnd4(inputWord>>0 ) & 0x20202020 , 0x10101010);
        *out++=_sub4(_xpnd4(inputWord>>4 ) & 0x20202020 , 0x10101010);
        *out++=_sub4(_xpnd4(inputWord>>8 ) & 0x20202020 , 0x10101010);
        *out++=_sub4(_xpnd4(inputWord>>12) & 0x20202020 , 0x10101010);
        *out++=_sub4(_xpnd4(inputWord>>16) & 0x20202020 , 0x10101010);
        *out++=_sub4(_xpnd4(inputWord>>20) & 0x20202020 , 0x10101010);
        *out++=_sub4(_xpnd4(inputWord>>24) & 0x20202020 , 0x10101010);
        *out++=_sub4(_xpnd4(inputWord>>28) & 0x20202020 , 0x10101010);
    }

    {
    unsigned char *outc = (unsigned char *)out;
    inputWord=~*in++;
    for(i=0;i<xbits;i++) *outc++ = ( (_extu(inputWord,31-i,31)<<5) - 0x10);
    }
```

2.4 Signal-to-Noise Ratio (SNR)

In a AWGN channel, the signal is corrupted by additive noise, n(t), which has a variance σ^2 and a noise density ratio N_0.

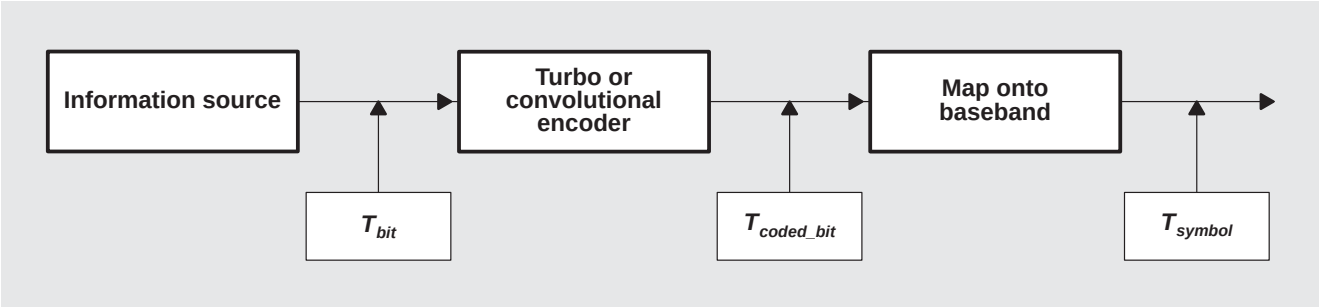


Figure 2. The Different Periods in the Transmission Chain

The information source consists of a bit sequence assumed randomly distributed amongst (0,1). That source is coded with rate $\frac{k}{n}$ and the resulting symbol sequence is mapped onto a BPSK modulation:

0 is transmitted as a and 1 as –a, a being the carrier amplitude. The resulting root-mean-square (rms) power of the transmitted signal is:

$$P_{rms} = Power(0) \times \Pr ob(0) + Power(1) \times \Pr ob(1) = a^2 \times \frac{1}{2} + (-a)^2 \times \frac{1}{2}$$

$$P_{rms} = a^2$$

If $\frac{1}{T_{bit}}$ is the bit rate of the useful information sequence before coding, then the energy per coded symbol, E_{symbol}, is:

$$E_{symbol} = a^2 \times \frac{k}{n} \times T_{bit}$$

The energy per useful bit, E_b, can be written as: $E_b = a^2 \times T_{bit}$

The noise samples (noise density ratio N_0) are added to the transmitted symbol stream, at a rate of $\frac{n}{k} \times \frac{1}{T_{bit}}$, and a variance:

$$\sigma^2 = \frac{1}{2} \times N_0 \times BandwidthOfUse = \frac{1}{2} \times N_0 \times \frac{1}{T_{symbol}}$$

$$\sigma^2 = \frac{1}{2} \times N_0 \times \frac{n}{k} \times \frac{1}{T_{bit}}$$

As a result, the signal-to-noise ratio $SNR_{bit}(dB)$, is expressed as a function of the signal over noise power, $\frac{a^2}{\sigma^2}$, assuming a is set to 1 in the scope of the study:

$$SNR_{bit}(dB) = 10 \times \log_{10}\frac{E_b}{N_0}$$

or

$$\boxed{SNR_{bit}(dB) = 10 \times \log_{10}\left(\frac{n}{2 \times k \times \sigma^2}\right)}$$

Examples: σ^2 = 0.64 Watts/Hz

- AMR 12.2 kbps Class A (frame length:51 coding rate:1/3 constraint length:9)
- SNR = 3.73 dB
- AMR 12.2 kbps Class C (frame length:60 coding rate:1/2 constraint length:9)
- SNR = 1.94 dB

2.5 Bit Error Rates Measurements and Stopping Criteria

The bit error measurements consist of:

- Comparing the decoded data bits to the transmitted data bits
- Counting the number of bit errors
- Generating enough frames and bits to assume a point on a BER curve is a valid statistical value

This is the reason a stopping criteria is needed. The number of generated bits/frames has an obvious influence on a BER curve validity.

A value is considered valid if:

- The number of corrupted frames is greater than the empirical value of 1000
- The number of corrupted bits is greater than 1000 * frame length
- The number of generated bits is great enough to have 10 corrupted frames

2.6 BER Software Framework

A BER software framework needs to:

- Implement a communication channel for the different types of frames
- Implement the BER measurements and stopping criteria
- Fix a given noise variance (σ^2) and calculate the SNR for a given type of frame

3 White Gaussian Noise Channel

3.1 Noise Generation

As just mentioned, to simulate a transmission channel, we need to distort the transmitted channel symbols that were generated through the viterbi/turbo encode process and through mapping into the antipodal base band. For the BER measurements outlined here, we assume that transmission takes place over an AWGN channel, i.e., Gaussian distributed random values are added to the original signal that was transmitted. To implement such a channel model on the DSP, we need a source of such random numbers. Because of the required formatting, we seek to implement a 8-bit, fixed-point random generator, which can be done in several ways. We must ensure that implementation is robust with regards to the limited dynamic range, and that values that fall outside the range that can be represented in 8-bit format are saturated to the most positive or negative value.

3.2 Implementation

For the BER measurements, we chose to implement a fairly straightforward, but because of many calls to library functions, a somewhat resource-consuming, method. First, we approximate the Gaussian distribution with zero mean and standard deviation sigma

$$P(x) = \frac{1}{\sigma\sqrt{2\pi}}\exp\left(-x^2/(2\sigma^2)\right) \tag{3.1}$$

by a binomial distribution given by

$$P(n) = \left(\frac{N}{n}\right)p^n q^{N-n} \tag{3.2}$$

This approximation is valid, provided that N is sufficiently large since

$$\left(\frac{N}{n}\right)p^n q^{N-n} \rightarrow \frac{1}{\sqrt{2\pi Npq}}\exp\left(\frac{-(n-Np)^2}{2Npq}\right) \tag{3.3}$$

as $N \rightarrow \infty$. We set $p = q = 1/2$ in (3.2) and chose $N = \sigma^2/(pq) = 4\sigma^2$ to obtain a binomial distribution with standard deviation σ. Then setting $n' = n - Np$, we obtain a distribution with zero mean that approximates P(x) of (3.1).

As far as the software implementation goes, we make use of the run-time support library function, rand(), to generate random bits that are simply interpreted as steps of a “random walk,” in the positive and negative direction depending on whether bits are 0 or 1. The standard deviation of the resulting distribution will then be the square root of the number of random bits that were generated. The code is outlined below. Since the result will always be even (or odd) given an even (or odd) number of steps, we may add or remove one step based on another randomly generated bit pattern.

It should be pointed out that such an 8-bit random generator will not be accurate for all conceivable values of the standard deviation σ. This is not related to the actual implementation itself, but is rather a consequence of the 8-bit format used. If sigma grows towards 0x7F (maximum number that can be represented), output will differ from the pure Gaussian distribution because of the many values being saturated. On the other hand, if σ is close to zero, output will be degenerated because of quantization. For the BER measurements performed here, this function was always called with $\sigma = 0x20$, and the results were then scaled afterwards to obtain the desired noise power.

```
/*
This function generates gaussian distributed random values in the
range [-128, 127] with standard deviation equal to sigma. The gaussian
distribution is approximated with a binomial distribution.
*/

char wgn_fixpt(unsigned char sigma)
{
       short value = 0;
       unsigned int number_of_steps;
       unsigned int number_of_iter;
       int i;
       unsigned int mask, remaining_steps;
       unsigned int positive_steps;

       number_of_steps = (short) sigma * (short) sigma;

  /* if sigma is even (odd) we always get an even (odd) number
  of steps and hence even (odd) output value. To correct that
  we make one more step (with probability=25%) or one less
  step (p=25%) or leave it as is (p=50%). */

  mask = rand()&0x3;
  number_of_steps += (mask>>1)-(mask&1);

  /* in the loop below, 30 steps are calculated at once.
  Get # of iteration in the loop */

  number_of_iter = number_of_steps/30;
  remaining_steps = number_of_steps-number_of_iter*30;

 /* rand() in rts lib is returning a value in [0, 32767],
    i.e. the rightmost bit is always zero. Hence we get
    15 random bits from each call to rand() */

  for(i=0; i<number_of_iter; i++)
  {

     mask = ((rand() & 0x7FFF)<<15) | (rand() & 0x7FFF);
     positive_steps = _dotpu4(_bitc4(mask), 0x01010101);
     value += (positive_steps<<1)-30; //i.e. bitc(mask) - (30 - bitc(mask))
  }

 /* do the remaining steps (if sigma^2 was not divisible by 30) */
  mask = ((rand() & 0x7FFF)<<15) | (rand() & 0x7FFF);;
  mask >>= (30-remaining_steps);
  positive_steps = _dotpu4(_bitc4(mask), 0x01010101);
  value += (positive_steps<<1)-remaining_steps;

 /* saturate and cast to char */
  if (value>127) value = 127;
  if (value<-128) value =-128;
  return ((char) value);
}
```

3.3 Accuracy

Let us assume the probability distribution of code outlined above is H(n). We need to compare H(n), our generated noise function, to the Gaussian distribution P(x) given by equation (3.1). Rather than comparing directly to P(x), we will compare it against its discrete version, called Q(n), given by

$$Q(n) = \int_{n-1/2}^{n+1/2} P(x)dx = \begin{cases} erf\left(\frac{\sqrt{2}}{4\sigma}\right), n = 0 \\ \frac{1}{2}\left[erf\left(\frac{|n| + \frac{1}{2}}{\sigma\sqrt{2}}\right) - erf\left(\frac{|n| - \frac{1}{2}}{\sigma\sqrt{2}}\right)\right], n \neq 0 \end{cases}$$

In other words, the result would be Q(n), that is, if we had a perfect Gaussian random generator, and quantized its output to integer format. Plots of H(n) as well as the absolute difference |(H(n) − Q(n)| are given below for $\sigma = 32$. We see that the latter curve shows a good conformity between H(n) and Q(n), thus justifying the approximations that were made.

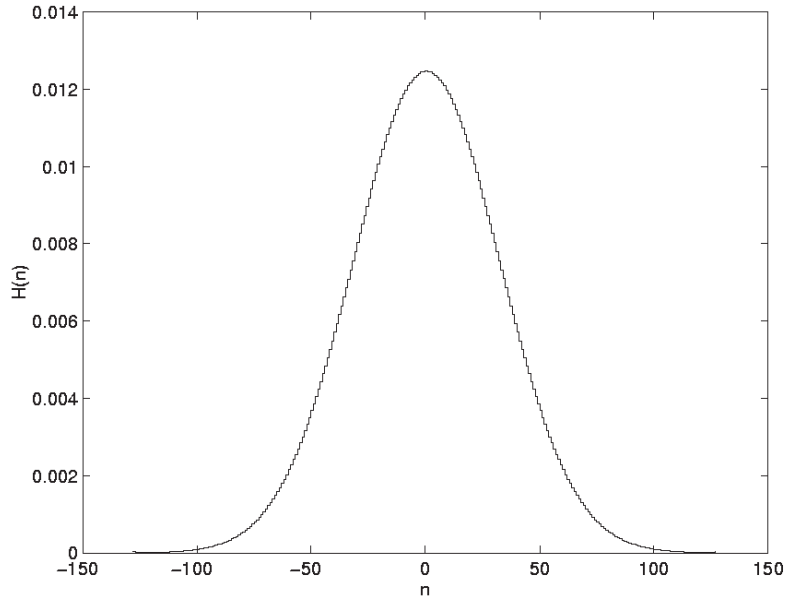


Figure 3. Probability Distribution of the Noise Generator for sigma = 32

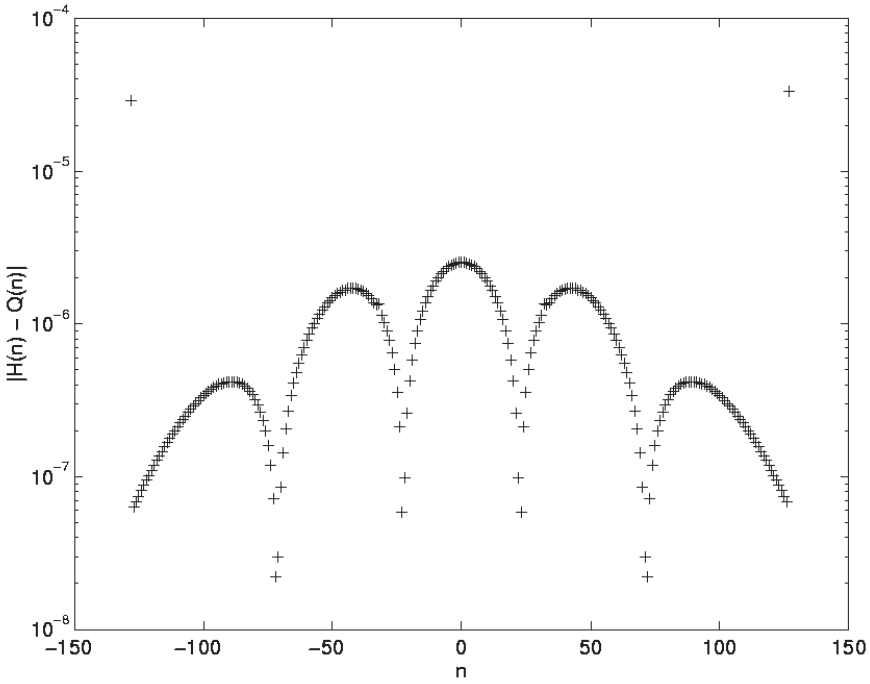


Figure 4. Deviation of H(n) Compared With Q(n) for sigma = 32

3.4 Further Optimizations

Given a source for uniformly distributed random values, there are several possibilities to generate Gaussian random numbers. One is the so-called Box-Muller transformation given by the following pseudo code:

```
do {
        x1 = 2.0 * ranf() – 1.0;
        x2 = 2.0 * ranf() – 1.0;
        w = x1 * x1 + x2 * x2;
} while ( w >= 1.0 );
w = sqrt( (–2.0 * ln( w ) ) / w );
y1 = x1 * w;
y2 = x2 * w;
```

where ranf() gives a random number uniformly distributed in [0,1], and y1, y2 are *two* independent Gaussian random numbers. A fixed-point implementation of the Box-Muller transformation has the potential to be faster than the implementation outlined here, due to far fewer calls to library functions. A prerequisite, however, is that the fixed-point (custom) implementation of the mapping $w \mapsto \sqrt{-2ln(w)/w}$ is carried out carefully with regards to its singularity at $w = 0$.

4 TMS320C6416 Coprocessors Soft-Decision Inputs and Configurations

The TMS320C6416 coprocessors produce the most likely transmitted sequence, given received noisy sequence. An ideal decoder would work with infinite precision, or at least with floating-point numbers. In practical systems, we quantize the received channel symbols with one or a few bits of precision, referred to as soft-decision input. The VCP and TCP have different soft-decision input requirements, so to decode the soft-decision inputs and create realistic BER curves, you must configure the coprocessors as they would be in a typical 3G application.

4.1 TMS320C6416 Viterbi Coprocessor

4.1.1 Input Requirements

The inputs to the VCP coprocessor (channel soft decisions/systems and parities) have to be:

- quantized with the following resolution (Table 1) depending on the rate r and constraint length K.

  The VCP implementation on TMS320C6416 implies that the soft inputs should be quantized so that the branch metrics satisfy the following bound B_1 (branch metrics upper bound – absolute value):

  $$2^{(C-1)} - 1 \geq (2 \times (K - 1) + 2) \times B_1$$

  K is the constraint length and C determines the truncation of state metrics that can be performed without loss of decoding performance.

  The VCP is designed with C = 12 and the branch metrics can have a maximum dynamic range of 6+1 sign bits do [−64;+63]. This give another branch metrics upper bound

  $$B_2 \leq 64$$

  So for a given constraint length, $\min(B_1, B_2)$ will give the final branch metrics maximum bound B.

  To satisfy B in the branch metrics calculation, the soft input resolution D is calculated with the following formula where 1/n is the rate.

  $$B = n \times 2^D$$

  Example:

  K=9 then $B_1 \leq 113.72$ and the branch metrics range B_2 is [−64;+63]. So the branch metrics need to be in [−64;+63] range.

  If rate 1/3, $\log_2\left(\frac{64}{3}\right) = 4.41$, so the soft inputs need to be quantized on 4+1sign = 5 bits.

  Calculation for the different constraint length and rate are summarized in Table 1.

Table 1. VCP Soft Input Resolution

1/Rate	K	Resolution
2	5, 6, 7, 8, 9	6
3	5, 6, 7, 8,9	5
4	5, 6, 7, 8,9	5

- sign-extended to 8 bits according to the resolution.
  Example: rate 1/3 and K=9
  The VCP input will have then to be provided with the following format:
  SSSSI.FFF (S = sign bit, I = integer, F = fractional bit).

4.1.2 Implementation

The code below shows an efficient implementation based on the TMS320C6416 instruction set.

The in[ ] buffer contains 2nd complement, signed 8-bit words with the following resolution: SIII.FFFF and is aligned on a 4-bytes boundary.

```
unsigned int i, j = 0;
char  maxpos, maxneg;
unsigned int maxpos4, maxneg4;
unsigned int rangemax4 = 0x1f1f1f1f;  // +31, max positive that won't be saturated
unsigned int rangemin4 = 0xE0E0E0E0;  //-32, most negative that won't be saturated
double temp;
unsigned int outtemp, temp1, temp2, shr_amnt;
unsigned int mask_repl_pos, mask_repl_neg, mask_not_repl, mask_neg_or_pos;
unsigned int repl_neg, repl_pos, repl;

       maxpos   = _set(0x00000000,0,8-softInputResolution);
       maxneg   = _sshvr((-128), (softInputResolution-2));
       maxpos4  = _packl4(_pack2(maxpos,maxpos),_pack2(maxpos,maxpos));
       maxneg4  = _packl4(_pack2(maxneg,maxneg),_pack2(maxneg,maxneg));
       shr_amnt = softInputResolution-4;


for (i=0; i<length; i+=4, j++)
{
   // pack outtemp with "normal" output, some may be saturated later
       temp    = _mpysu4(in[j], 0x01010101);
       temp1   = _shr2(_hi(temp), shr_amnt);
       temp2   = _shr2(_lo(temp), shr_amnt);
       outtemp = _packl4(temp1, temp2);

   // determine which bytes to saturate and which to keep
       mask_neg_or_pos = _cmpgtu4(in[j], 0x7F7F7F7F);
       mask_repl_neg   = _xpnd4(_cmpgtu4(rangemin4, in[j]) & mask_neg_or_pos);
       mask_repl_pos   = _xpnd4(_cmpgtu4(in[j], rangemax4) & ~mask_neg_or_pos);
       mask_not_repl   = ~(mask_repl_pos | mask_repl_neg);

   // clear outtemp from the bytes that are going to be saturated
       outtemp  = outtemp & mask_not_repl;
       repl_neg = mask_repl_neg & maxneg4;
       repl_pos = mask_repl_pos & maxpos4;

   // repl holds saturated bytes (saturated to positive and negative)
       repl = repl_neg | repl_pos;

   //  merge "normal" and saturated bytes
       out[j] = outtemp | repl;
}
```

4.1.3 VCP Configuration

The VCP should be serviced using the TMS320C6416 enhanced direct memory access (EDMA) module for most accesses, but you must first configure the VCP control values. The VCP control values, or input configuration (IC) values will be sent via the EDMA to program its operation. To generate the VCP BER curves, the coprocessor was configured as it would be in a typical 3G application.

Here is a description of VCP IC words in a typical case (AMR 12.2 kbps − class A in 3GPP standard):

Table 2. VCP Configuration Example

Input Configuration		
POLY0 = 0x6F	F = 93	SDHD = hard decisions
POLY1 = 0xB3	R = C = 0	OUTF = 1
POLY2 = 0xC9	IMAXS = 0x400	TB = tailed
POLY3 = 0x00	IMINS = 0x0	SYMR = 0x1
YAMEN = 1	IMAXI = 0x0	SYMX = 0x6
YAMT = 100	RATE = 1/3	

4.2 TMS320C6416 Turbo Coprocessor

4.2.1 Input Requirements

The inputs to the TCP coprocessor (channel soft decisions / systems and parities) have to be:

- scaled by a factor

  $$-\frac{2\sqrt{E_{symbol}}}{\sigma^2}$$

  In the channel simulation, the received symbols r with an energy E_{symbol} can be written as

  $$r = \pm\sqrt{E_{symbol}} \times 1, -1 + noise,$$

  assuming BPSK modulation. Both E_{symbol} and the variance σ^2 of the received frame need to be estimated.
- quantized on 8 bits as SIIII.FFF (S = sign bit, I = integer, F = fractional bit).

4.2.2 Implementation

4.2.2.1 Estimating the Scaling Factor

Consider a high speed data rate frame of N soft symbols x_i.

$$\hat{X} = \frac{1}{N} \times \sum_{i=1}^{N} |x_i| \text{ and } \hat{X^2} = \frac{1}{N} \times \sum_{i=1}^{N} x_i^2 \text{ on the frame.}$$

First, estimate the variance.

$$\sigma^2 = \frac{1}{N-1}\sum_{i=1}^{N}(|x_i| - \hat{X})^2$$

$$\sigma^2 = \frac{1}{N-1} \times \left(\sum_{i=1}^{N} x_i^2 - \sum_{i=1}^{N}(2 \times |x_i| \times \hat{X}) + \sum_{i=1}^{N}(\hat{X})^2\right)$$

$$\sigma^2 = \frac{N}{N-1} \times (\hat{X^2} - (\hat{X})^2)$$

The high speed data rate frame length are generally big enough to consider $\frac{N}{N-1} \sim 1$

Then the variance will be computed as: $\boxed{\sigma^2 = \hat{X^2} - (\hat{X})^2}$

Now estimate the energy per symbol of the received frame. The mean of the square received symbols $\hat{X^2}$ can be used to calculate the energy per symbol E_{symbol}. A received symbol can be written as:

$x_i = \sqrt{E_{symbol}} \times u_i + n_i$

where u_i is a BPSK symbol with value +1 or −1 and n_i comes from AWG noise with zero mean and variance σ^2.

$$\hat{X^2} = \sum_{i=1}^{N} \left(\sqrt{E_{symbol}} \times u_i + n_i \right)^2$$

Considering a zero mean noise, $\sum_{i=1}^{N} n_i = 0$ and $\sigma^2 = \sum_{i=1}^{N} n_i^2$

$\hat{X^2} = E_{symbol} + \sigma^2$ or $E_{symbol} = \hat{X^2} - \sigma^2$

and assuming $\frac{N}{N-1} \sim 1$, then: $\boxed{E = (\hat{X})^2}$

the scaling factor can then be estimated as:

$$scalingFactor = -2 \times \frac{\sqrt{E_{symbol}}}{\sigma^2}$$

or

$$\boxed{scalingFactor = -2 \times \frac{\hat{X}}{\hat{X^2} - (\hat{X})^2}}$$

The implementation on a fixed point DSP may require a lookup table such as

$scalingFactor = -f(sigma^2) \times \sqrt{E_{symbol}}$.

4.2.2.2 Quantization

Below is an efficient implementation based on the TMS320C6416 instruction set.

The in[ ] buffer contains a 2nd complement signed 8-bit word with the following resolution: SIII.FFFF and is aligned on a 4 bytes boundary.

```
double  temp,temp1,temp2;
S32   hi_temp1,lo_temp1,hi_temp2,lo_temp2;
U32   scale, i, j=0;
```

```
/* duplicate scale in 4 bytes */
scale = _packl4(_pack2(scale_fixpt,scale_fixpt),_pack2(scale_fixpt,scale_fixpt));

for(i=0; i < length; i+=4, j++)
{
   /* scale 4 input values by scaling factor */
      // in[j] is in signed Q4, and scale holds unsigned Q5
      // then temp is in signed Q9
       temp=_mpysu4(in[j],scale);

   /* saturate into upper byte and negate */
         temp1=_smpy2(_hi(temp),0xC000C000);
         temp2=_smpy2(_lo(temp),0xC000C000);

   // here, upper 16 bits of temp1 and temp2 will be in signed Q8, i.e.
   // upper 8 bits are in Q0. We need Q3, hence scale 3 positions left.

        hi_temp1=_sshl(_hi(temp1),3);
        lo_temp1=_sshl(_lo(temp1),3);
        hi_temp2=_sshl(_hi(temp2),3);
        lo_temp2=_sshl(_lo(temp2),3);

   /* pack 4 upper bytes of sshl's */
        out[j]=_packh4(_packh2(hi_temp1,lo_temp1),  _packh2(hi_temp2,lo_temp2));
}
```

4.2.3 TCP Configuration

The TCP should be serviced using the TMS320C6416 EDMA module for most accesses, but you must first configure the TCP control values. The TCP control values, or input configuration (IC) values will be sent via the EDMA to program its operation. To generate the TCP BER curves, the coprocessor was configured as it would be in a typical 3G application.

Here is a description of TCP IC words in a typical case (144 kbps data in 3GPP standard) :

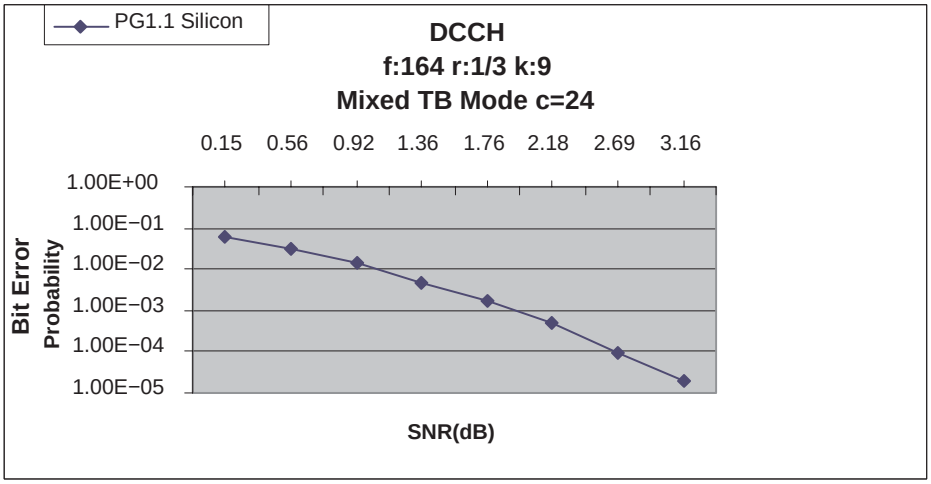
Table 3. TCP Configuration Example

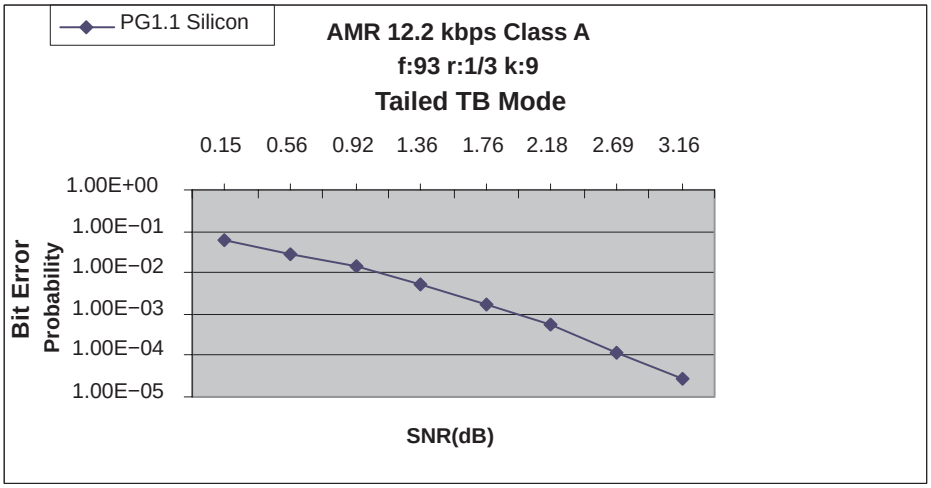
Input Configuration		
FL = 3168 OUTF = 1 INTER = 1 RATE = 1/3 OPMOD = SA R = 0x71 SFL = 0x0 SNR Threshold disabled	MAXIT = 8 LASTNSB = 0 NSB = 7 P = 32 NWORDDSP = 0x252 NWORDINTER = 0x630 NWORDHD = 0x63	TAIL1 = 0x00F5FDD3 TAIL2 = 0x00F70104 TAIL3 = 0x00000000 TAIL4 = 0x00EFFE02 TAIL5 = 0x00FDFB23 TAIL6 = 0x00000000

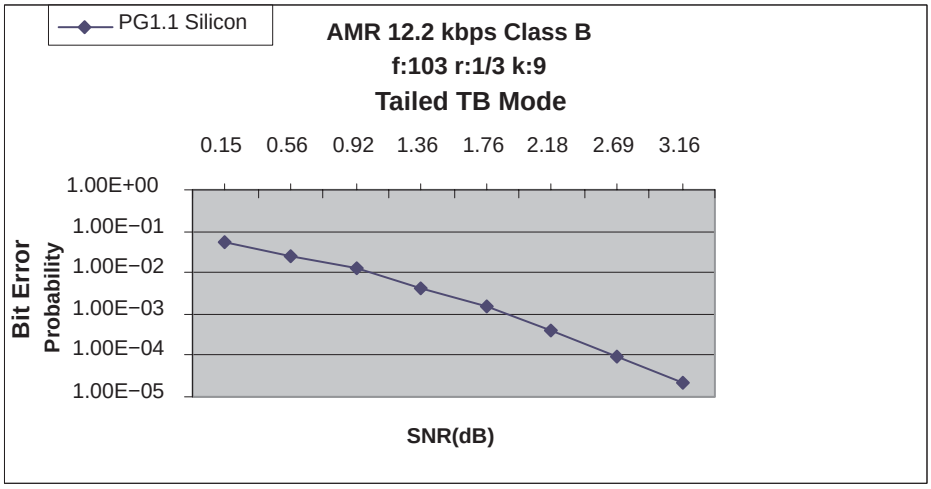
5 BER Results

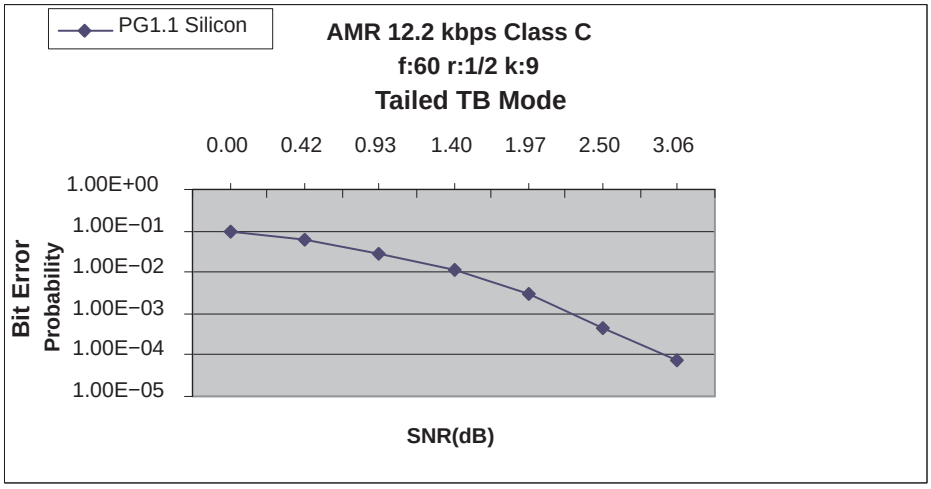
Here are some bit error rate curves performed on a TMS320C6416 PG1.1 device. The most common 3GPP rates have been chosen.

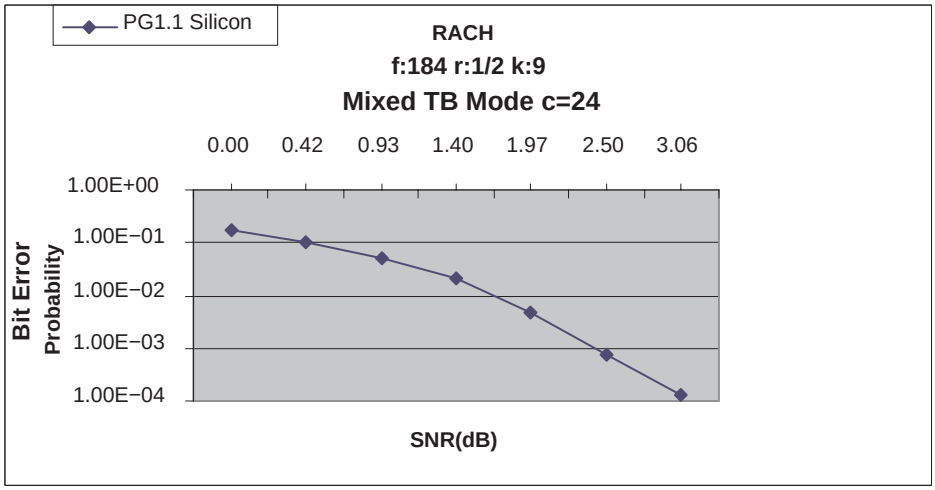
5.1 VCP: 3GPP Frames

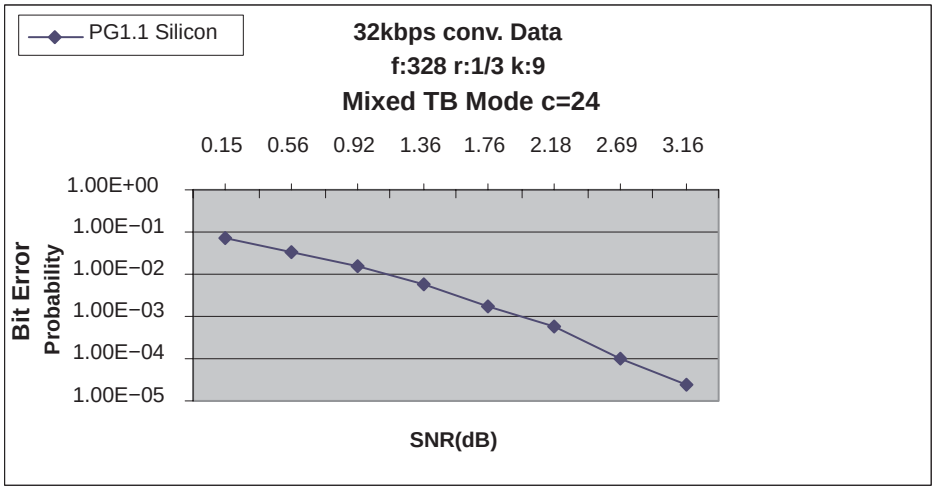




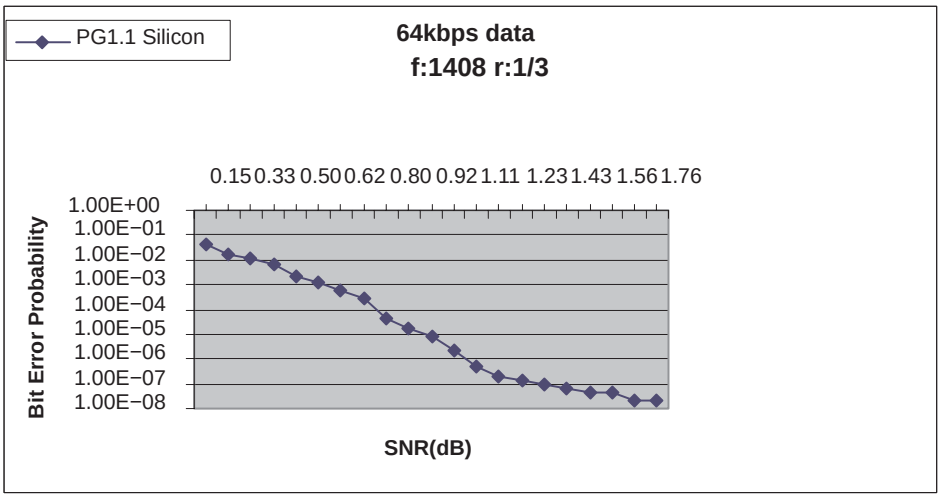

PG1.1 Silicon
AMR 12.2 kbps Class B
f:103 r:1/3 k:9
Tailed TB Mode
0.15 0.56 0.92 1.36 1.76 2.18 2.69 3.16
1.00E+00
1.00E−01
1.00E−02
1.00E−03
1.00E−04
1.00E−05
Bit Error Probability
SNR(dB)

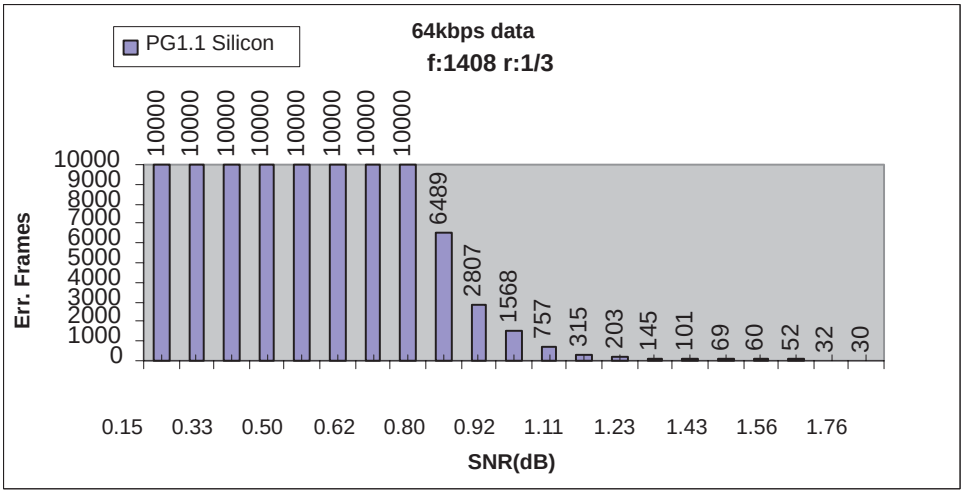

PG1.1 Silicon
AMR 12.2 kbps Class C
f:60 r:1/2 k:9
Tailed TB Mode
0.00 0.42 0.93 1.40 1.97 2.50 3.06
1.00E+00
1.00E−01
1.00E−02
1.00E−03
1.00E−04
1.00E−05
Bit Error Probability
SNR(dB)

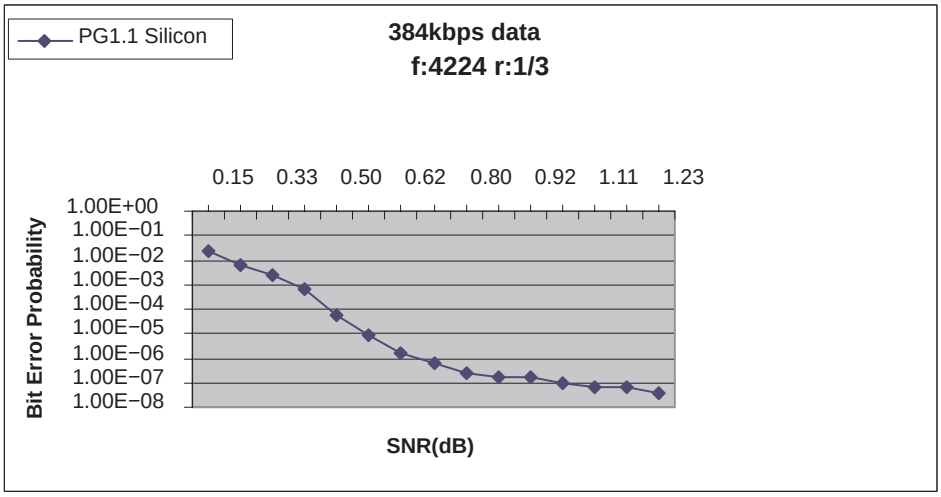

PG1.1 Silicon
RACH
f:184 r:1/2 k:9
Mixed TB Mode c=24
0.00 0.42 0.93 1.40 1.97 2.50 3.06
1.00E+00
1.00E−01
1.00E−02
1.00E−03
1.00E−04
Bit Error Probability
SNR(dB)

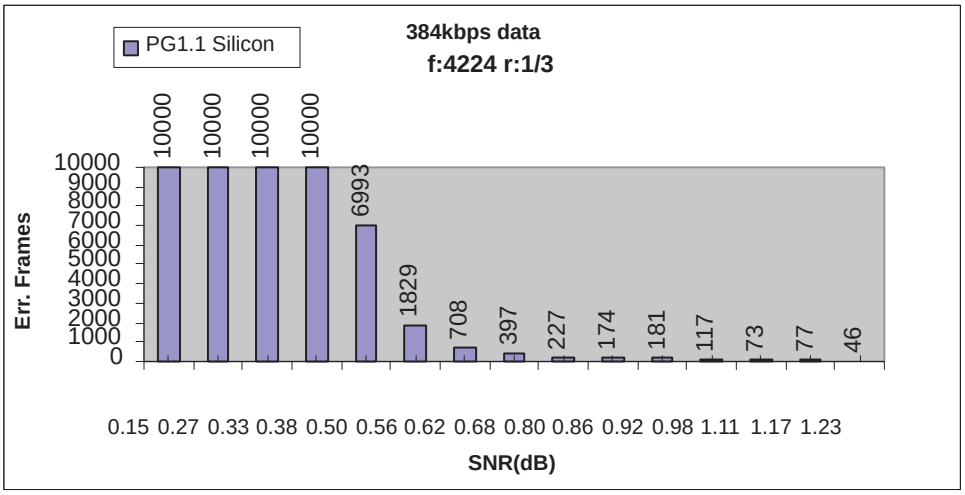


5.2 TCP: 3GPP Frames









6 References

1. *Viterbi Decoder Coprocessor User's Guide* (SPRU533)
2. *Turbo Decoder Coprocessor User's Guide* (SPRU534)
3. *TMS320C6000 Peripherals Reference Guide* (SPRU190)
4. *Using TMS320C6416 Coprocessors: Viterbi Coprocessor (VCP)* (SPRA750)
5. *Using TMS320C6416 Coprocessors: Turbo Coprocessor (TCP)* (SPRA749)
6. L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," IEEE Trans. Inform. Theory, vol. IT–20, pp. 284–287, Mar. 1974.

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
		Telephony	www.ti.com/telephony
		Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

Mailing Address: Texas Instruments
Post Office Box 655303 Dallas, Texas 75265