Application Report
SPRAAG3−December 2006

TMS320TCI648x TCP2 Channel Density

Brighton Feng

ABSTRACT

Turbo decoder lies at the heart of all of the third-generation (3G) wireless standards. The turbo coprocessors (TCP2) are programmable peripherals used to decode turbo codes. It is integrated into Texas Instruments TMS320TCI648x Digital Signal Processor (DSP). This application report gives the channel density data for TCP2 under various conditions, the CPU load for pre-processing TCP2 input data is also provided.

This application report contains project code that can be downloaded from this link.

http://www-s.ti.com/sc/techlit/spraag3.zip

Contents

List of Figures

List of Tables

1 Introduction

Forward-error correction (FEC), also known as channel coding, is used to improve the reliability of a channel by adding redundant information to the data being transmitted. Turbo coding techniques are used in all third-generation (3G) wireless standards. The turbo coprocessors (TCP) are programmable peripherals used to decode turbo codes. It is integrated into some Texas Instruments digital signal processors (DSP). The turbo coprocessor in the TMS320TCI100/TMS320C6416 is called TCP1; the turbo coprocessor in the TMS320TCI648x is an enhanced version of TCP1, and is called TCP2.

The channel density of TCP2 is related to several parameters. The main parameters are frame length, number of iterations, and sliding window prolog length. This application report gives channel density data for different scenarios and discusses the effect of these parameters.

Pre-processing are required before data can be decoded in TCP2, include scaling, de-interleaving interleaved parities, un-puncturing data to 1/5 rate, configuring TCP2/EDMA3, etc. The MIPS consumption of pre-processing is also provided in this application report.

This document gives designers a basis for estimating system performance. Most of the tests operate under best-case situations to estimate maximum throughput that can be obtained. Most of the following performance data is examined on the 1 GHz TCI6482 EVM.

2 Channel Density Measurement Methodology and Assumptions

To measure the channel density, communication channel are simulated. The communication simulation module and channel density measurement block diagram are shown in Figure 1.

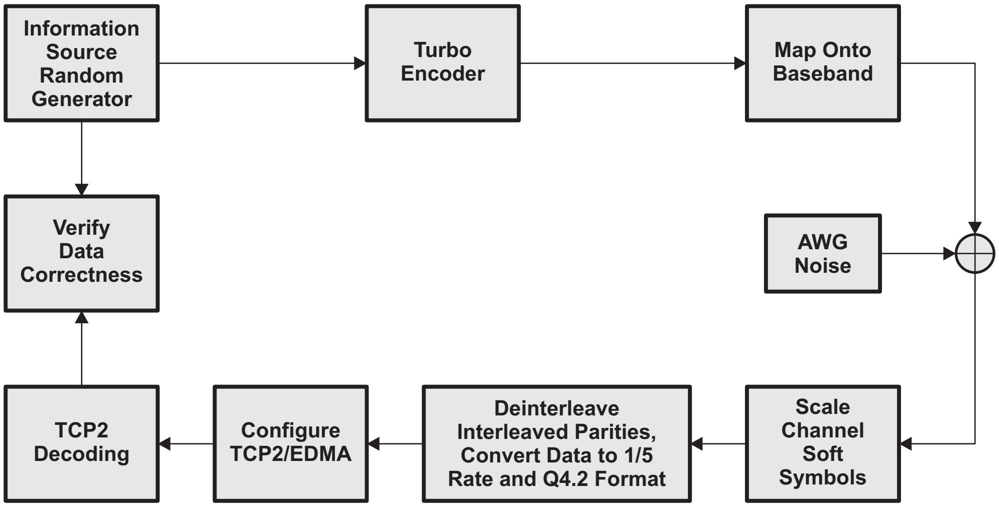
Information Source Random Generator
Turbo Encoder
Map Onto Baseband
Verify Data Correctness
AWG Noise
TCP2 Decoding
Configure TCP2/EDMA
Deinterleave Interleaved Parities, Convert Data to 1/5 Rate and Q4.2 Format
Scale Channel Soft Symbols

Figure 1. Communication Channel Simulation for TCP2

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

1. Generate the binary data bits (information sequence) to be transmitted through the channel.
2. Generate the turbo interleaver table, turbo interleave, and turbo encode the information sequence in channel symbols.
3. 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.
4. Add AWG Noise to the transmitted channel symbols to generate received channel symbols (soft symbol).

Pre-processing are required before data can be decoded in TCP2, the steps are as following:

1. Scale channel soft symbols by (2*a)/(Sigma^2)
2. De-interleave Interleaved Parities
3. De-puncture data to 1/5 rate
4. Quantize data on 6 bits as Q(4,2) bits signed fix-point data
5. Calculate TCP2 parameters
6. Configure EDMA3 PaRAM

Pre-processing can be overlapped with TCP2 decoding. Typical scheduling for TCP2 decoding and CPU pre-processing are shown in Figure 2.

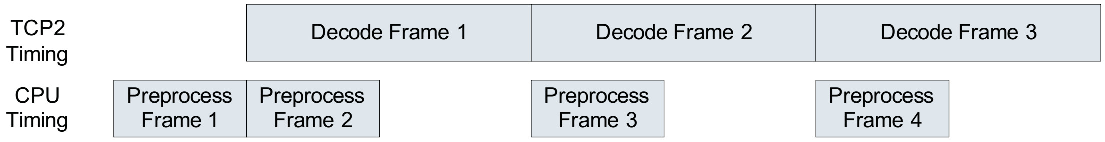

Figure 2. TCP2 Decoding and Pre-Processing Scheduling

Pre-processing time should be always less than decoding time, and channel density is directly determined by the decoding time. Though pre-processing doesn't directly determine the channel density, it may affect overall system channel density because CPU consumes some MIPS for TCP2 pre-processing, thus has less MIPS for other processing of the system. So, CPU load for pre-processing are also measured and provided in this document.

The last step for channel density measurement is to start the TCP2 and wait for completion. The decoding time is measured from TCP2 starting until all output data are read out from TCP2. Please note, the time for EDMA transferring data from/to DSP memory to/from TCP2 internal memory are treated as a part of the decoding time in this document.

All data is measured on 1GHz TCI6482 EVM. All time is originally measured in cycle and transformed to nanosecond (ns). One cycle is just one nanosecond (ns) for 1GHz DSP. Except for special notes, the default conditions for the measurement are:

- Frame Length = 2576 (3GPP 64kbps)
- Code Rate = 1/3
- Iteration Number = 8 (MaxIter = MinIter = 8, early stopping criteria disabled)
- Prolog Length = 12, Prolog reduction enabled
- Standalone operation mode, shared processing mode is not covered in document since the maximum standalone frame length, 20730, is enough for almost all applications.
- max*-log-MAP
- Interleaver table is transferred into TCP2 for every frame
- Output parameters are transferred out from TCP2 for every frame
- All data in internal memory, 32KB L1D and 32KB L1P
- Build options: -o3, no debug

3 Channel Density for Typical Scenarios

The following sections give channel density for some typical scenarios in WCDMA, CDMA2000, and TD-SCDMA system.

The formulas used to calculate channel density are as following:

- (Channel Density) = (Frame Interval)/(Decoding Time per Channel)
- (Overall Throughput) = (Channel Density)*(Information Bit Rate per Channel)
- (CPU Load for Pre-Processing) = (Pre-Processing Time)/(Decoding Time)*100%

The frame length in the following tables is the number of input bits to Turbo encoder or output bits from Turbo decoder. These bits are composed of original information bits, CRC bits or other control bits depend on the standard.

3.1 WCDMA

Table 1 gives the channel density data of TCP2 for the reference measurement channels listed in 3GPP TS 25.104 V7.4.0, Base Station (BS) radio transmission and reception (FDD).

Table 1. Typical Channel Density for WCDMA[(1)]

Information Rate (kbps)	Frame Length (bits)	Frame Interval (ms)	Decoding Time (ns)	Channel Density	Throughput (Mbps)	Pre-Processing Time (ns)	CPU Load for Pre-Processing
64	2576	40	161510	247.7	15.850	17504	10.84%
144	2896	20	178742	111.9	16.113	19402	10.85%
384	3856	10	233870	42.8	16.419	25823	11.04%
1353	2730	2	169330	11.8	15.981	18438	10.89%
2706	2718	1	168278	5.9	16.081	18366	10.91%
4050	4062	1	243482	4.1	16.634	27483	11.29%
507.6	5076	10	300926	33.2	16.868	35739	11.88%
978	4902	5	292906	17.1	16.695	34349	11.73%
1927.8	4826	2.5	292906	8.7	16.818	33545	11.71%
69	714	10	53914	185.5	12.798	5769	10.70%

[(1)] Code Rate = 1/3, Iteration Number = 8, Prolog Length = 12

3.2 TD-SCDMA

Table 2 gives the channel density data of TCP2 for the reference measurement channels listed in 3GPP TS 25.105 V7.2.0, Base Station (BS) radio transmission and reception (TDD).

Table 2. Typical Channel Density for TD-SCDMA[(1)]

Information Rate (kbps)	Frame Length (bits)	Frame Interval (ms)	Decoding Time (ns)	Channel Density	Throughput (Mbps)	Pre-Processing Time (ns)	CPU Load for Pre-Processing
64	1296	20	87458	228.7	14.636	9323	10.66%
144	2896	20	178742	111.9	16.113	19402	10.85%
384	3856	10	233870	42.8	16.419	25823	11.04%

[(1)] Code Rate = 1/3, Iteration Number = 8, Prolog Length = 12

3.3 CDMA2000

Table 3 gives the channel density data of TCP2 for reverse channels listed in 3GPP2 C.S0002-D_v2.0_ Physical Layer Standard for cdma2000 Spread Spectrum Systems.

Table 3. Typical Channel Density for CDMA2000

Information Rate (kbps)	Frame Length (bits)	Frame Interval (ms)	Code Rate	Decoding Time (ns)	Channel Density	Throughput (Mbps)	Pre-Processing Time (ns)	CPU Load for Pre-Processing
Reverse Fundamental Channel and Reverse Supplemental Channel Structure for Radio Configuration 5								
28.8	570	20	1/4	44703	447.4	12.885	5918	13.24%
57.6	1146	20	1/4	78695	254.1	14.639	10141	12.89%
115.2	2298	20	1/4	143831	139.1	16.019	19295	13.42%
115.2	4602	20	1/4	273875	73.0	16.825	50777	18.54%
460.8	9210	20	1/4	533967	37.5	17.259	165027	30.91%

Table 3. Typical Channel Density for CDMA2000 (continued)

Information Rate (kbps)	Frame Length (bits)	Frame Interval (ms)	Code Rate	Decoding Time (ns)	Channel Density	Throughput (Mbps)	Pre-Processing Time (ns)	CPU Load for Pre-Processing
1036.8	20730	20	½	1184195	16.9	17.511	210372	17.76%
Reverse Dedicated Control Channel Structure for Radio Configuration 6								
19.2	378	20	1/4	33339	599.9	11.518	4726	14.18%
38.4	762	20	1/4	56787	352.2	13.524	7392	13.02%
76.8	1530	20	1/4	100479	199.0	15.287	13149	13.09%
153.6	3066	20	1/4	187179	106.8	16.412	26443	14.13%
307.2	6138	20	1/3	360575	55.5	17.039	57207	15.87%
614.4	12282	20	1/3	707363	28.3	17.372	186985	26.43%

The CPU load for pre-processing in above tables is the maximum CPU load for maximum channel number. Please note, the actual CPU load is proportional to actual channel number.

4 The Effects of Different Parameters on TCP2 Decoding Time

The channel density of TCP2 is related to several parameters. The main parameters are frame length, number of iterations, and sliding window prolog length, etc. This section describes the effect of these parameters on channel density. The system designer can estimate channel density for non-typical scenarios according these information.

4.1 Effects of Frame Length

The following measurement result shows the effect of frame length on decoding time.

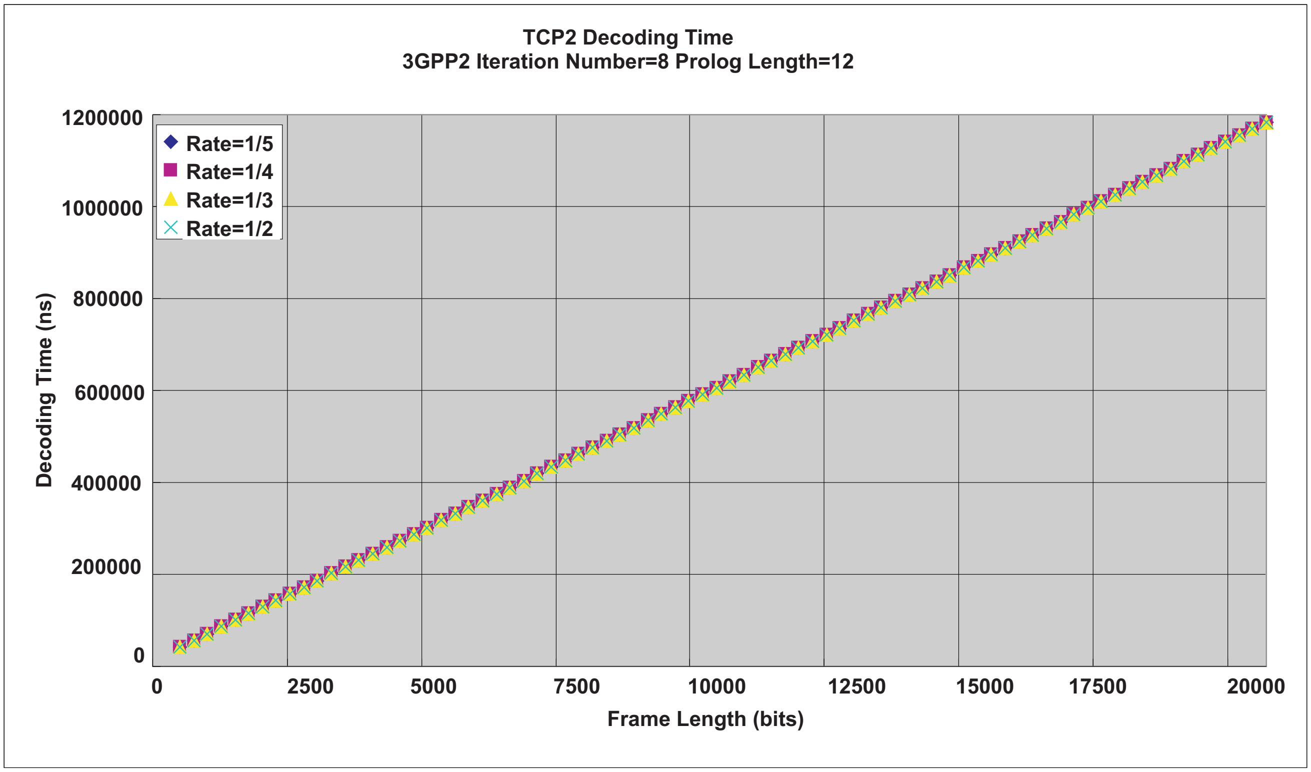

Figure 3. Effect of Frame Length and Rate on TCP2 Decoding Time

Figure 3 shows the TCP2 decoding time is proportional to frame length, i.e. the channel density is inversely proportional to frame length.

Figure 3 also shows that the code rate has no effect on the TCP2 decoding time or channel density, this is because all data are un-punctured to 1/5 rate before send to the TCP2 for decoding, and the TCP2 treats all input as 1/5 rate data.

4.2 Effects of Iteration Number

The following measurement result shows the effect of iteration number on decoding time.

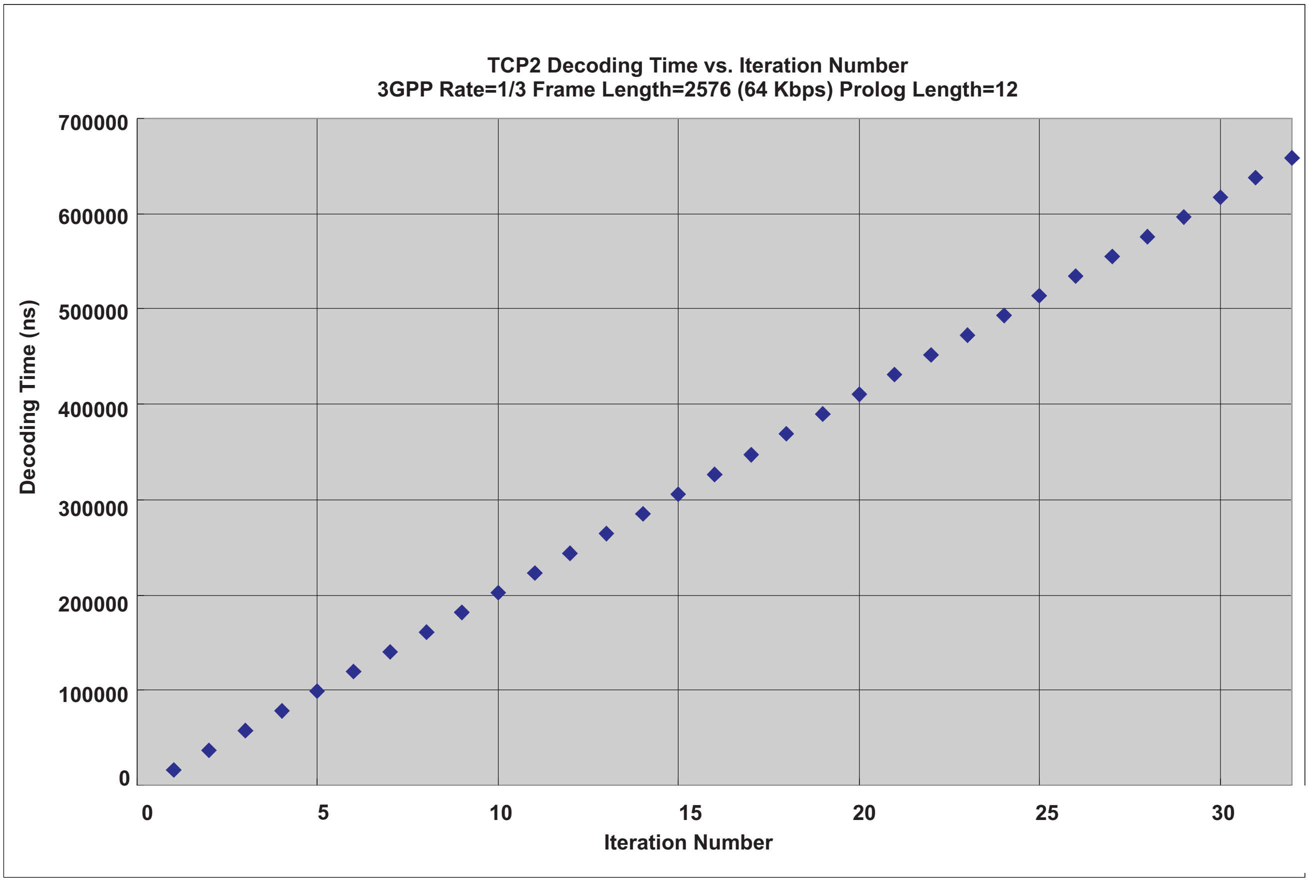

Figure 4. Effect of Iteration Number on TCP2 Decoding Time

Figure 4 shows the decoding time is proportional to iteration number, i.e. the channel density is inversely proportional to iteration number.

Iteration number is configurable, it is chosen based on the channel quality, i.e. the Signal Nosie Ratio (SNR) of input data. Increasing the iteration number reduces the Bit Error Rate (BER). System designer needs to run its BER measurements and adjust this parameter for his own system needs. Generally speaking, Iteration number = 8 is enough for most cases. More iteration than 8 can not improve BER.

Two stopping criteria can be used to expedite the TCP2 processing and stop TCP2 before the maximum iteration number reaches:

- Comparing the extrinsic SNR estimate to a SNR threshold (user defined)
- CRC pattern match

So, if these criteria are applied, the actual iteration number decreases if the channel has good quality, that is, the TCP2 can support higher channel density if the channel quality is better. For more information about stopping criteria, please refer to *TMS320TCI648x DSP Turbo-Decoder Coprocessor 2 (TCP2) Reference Guide* (SPRUE10).

4.3 Effects of Prolog Length

The following measurement result shows the effect of prolog length on decoding time.

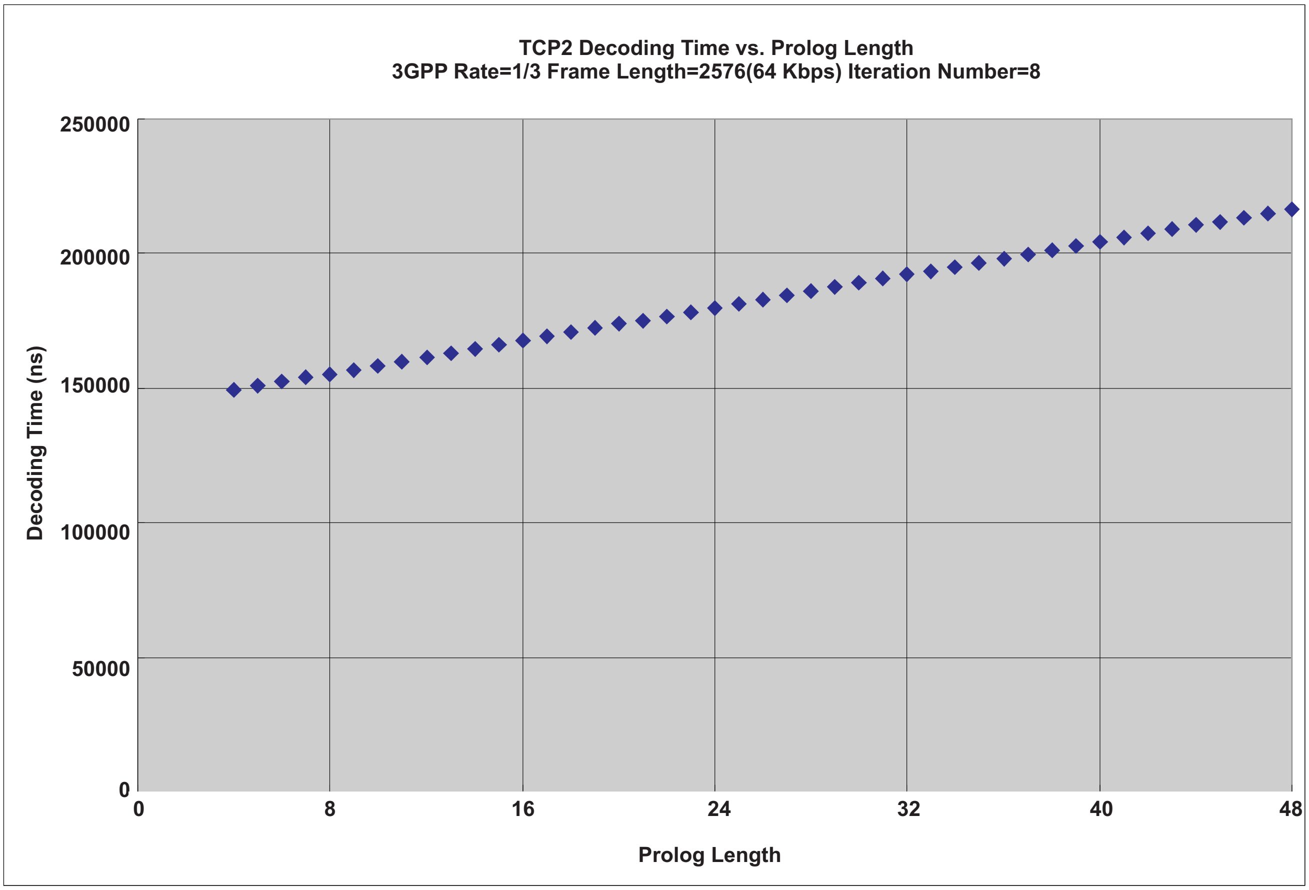

Figure 5. Effect of Prolog Length on TCP2 Decoding Time

Figure 5 shows the prolog length affects the decoding time or channel density obviously. The decoding time is proportional to (Prolog Length) + (slide windows reliability length) in theory.

Prolog length is configurable; generally speaking, the prolog should be equal to 3× the constraint length, i.e. 12 for 3GPP and 3GPP2. Increasing the prolog length can reduce the BER, but the effect on BER is not that obvious. System designer should run its BER measurements and adjust this parameter for his own system needs.

The prolog section of the sliding window is required to properly initialize the trellis for the reliability section of the sliding window. On the C6416 TCP, it is assumed that all states are equally probable at the beginning of the prolog. As the state metrics are accumulated over the length of the prolog, some states become more probable than others. The longer the prolog, the more reliable the initialization values, but also the higher the processing time. The decoding time for prolog length = 48 is about 45% more then prolog length = 4 is shown in Figure 5.

A new trellis initialization technique is introduced on the TCI6482 TCP2, which enables shorter prolog sections (typically between 4 and 16), without impacting the BER performance. This is achieved by using the results from the previous iteration to initialize the trellis at the beginning of the prolog, rather than just assuming that all states are equally probable. This is called a prolog reduction technique. Its main benefit is in reduced processing delay.

For backward compatibility reasons, the prolog reduction feature needs to be explicitly enabled, for each frame, through TCP input parameters. If this feature is enabled, a smaller prolog length can be chosen. It is not recommended to use a prolog length smaller than 24 when the prolog reduction feature is turned off, since that could result in poor BER performance.

4.4 Effects of Other Parameters

Code rate has no effect on the TCP2 decoding time since TCP2 treats all input data as 1/5 rate.

The choosing of max*-log-MAP or max-log-MAP algorithm has no effect on the TCP2 decoding time. The choosing criteria for them are introduced in the following section about scaling.

Decoding time for 3GPP frame and IS2000 frame are same as long as other parameters are same.

Interleaver table transfer can be omitted if the following frame uses same interleaver table as the last one, this saves the EDMA transfer time for interleaver table.

For some applications, the output parameter transfers can be omitted saving EDMA transfer time.

The EDMA can transfer 64 bits (8 bytes) every EDMA cycle (3 CPU cycles), so, the EDMA transfer time for interleaver and output parameter is only a small part of overall TCP2 decoding time. All tests for this document transfer interleaver table and output parameters.

5 MIPS Consumption of Data Pre-Processing for TCP2

Pre-processing is required before data can be decoded in TCP2. This section gives detailed data about the MIPS consumption of these pre-processing.

Figure 6 shows the test result of pre-processing time and the pre-processing time partition for 3GPP.

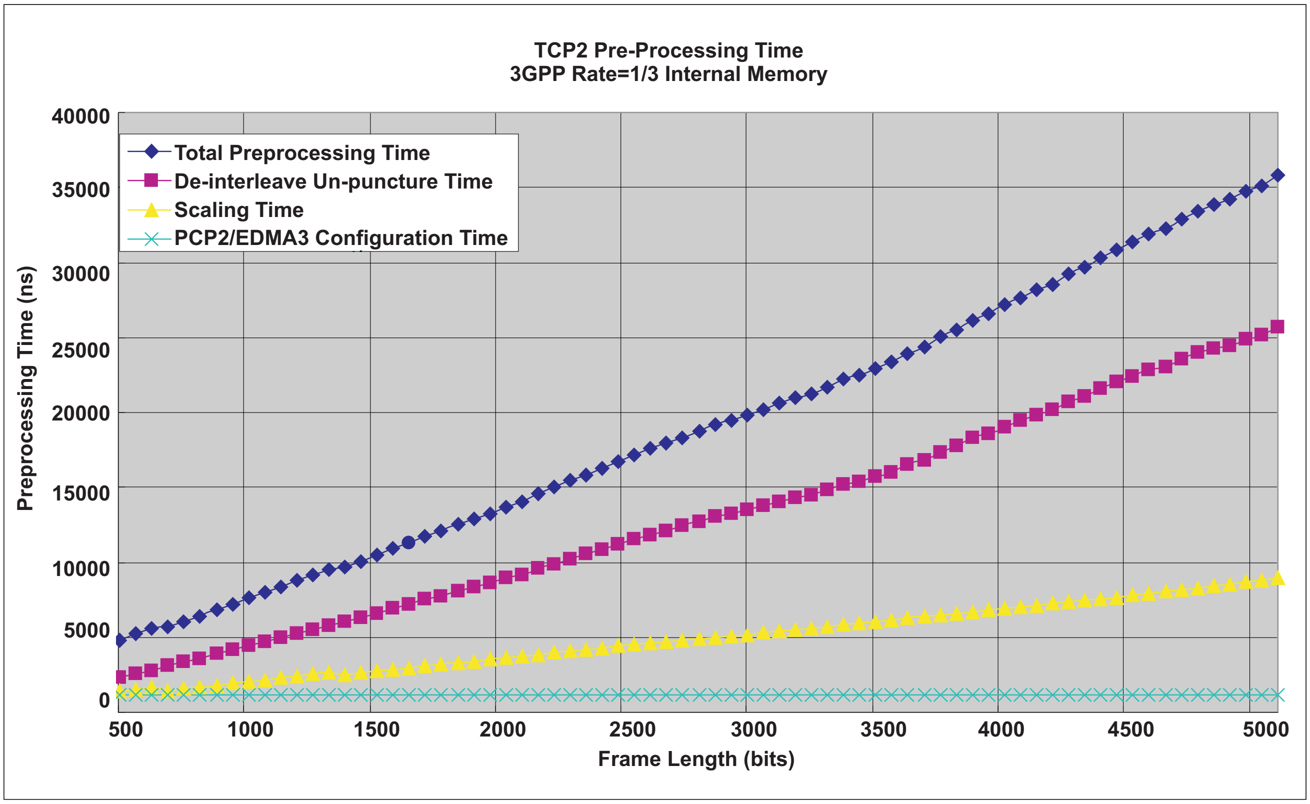

Figure 6. TCP2 Pre-Processing Time for 3GPP and Its Partition

The maximum frame length for 3GPP is 5114. Figure 6 shows the pre-processing time is almost proportional to frame length. TCP2 and EDMA3 configuration time is almost constant, totally about 1200 cycles. The following sections discuss processing in more detail.

Figure 7 shows the test result of pre-processing time and the pre-processing time partition for 3GPP2.

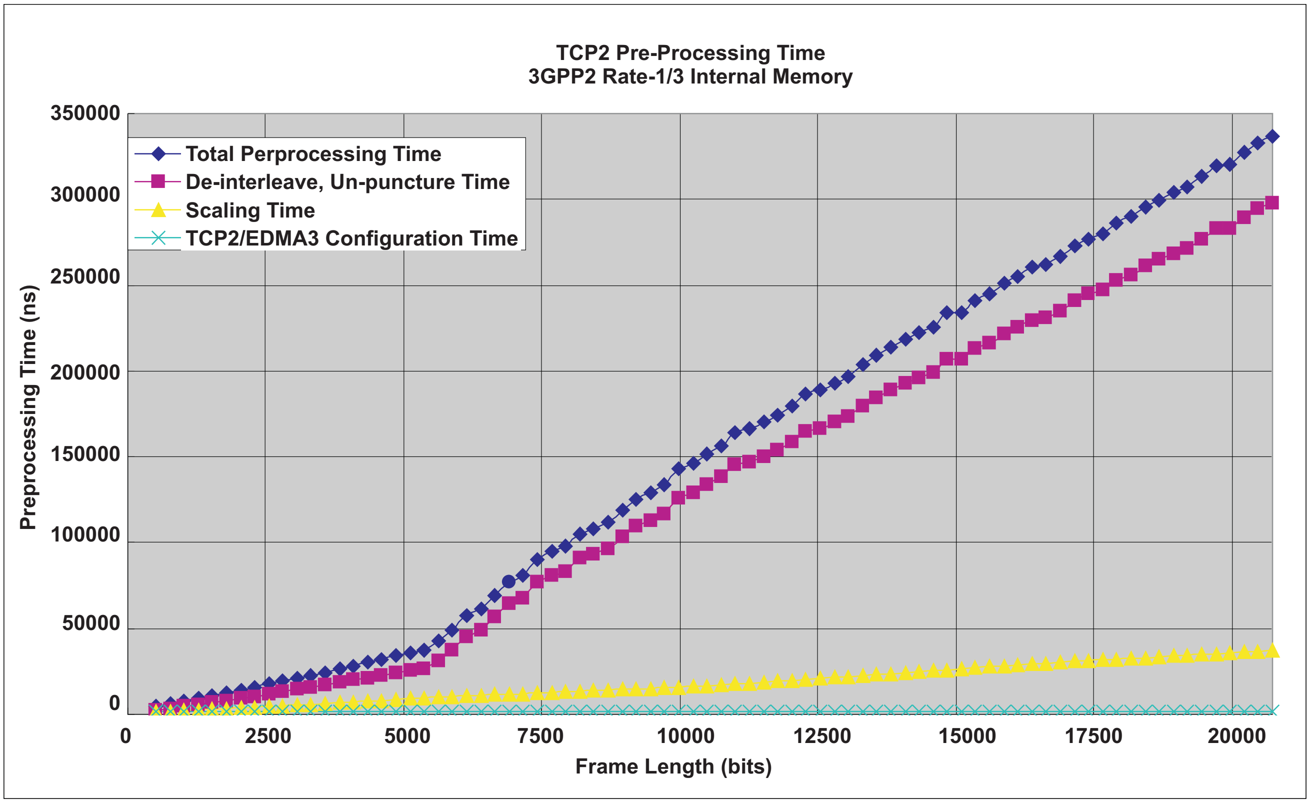

Figure 7. TCP2 Pre-Processing Time for 3GPP2 and Its Partition

The max frame length for 3GPP2 is 20730. Compared to 3GPP, the characteristic of pre-processing time for code rate = 1/3 and frame length < 5114 is the same as 3GPP.

For frame length larger than 5461, the de-interleaving, un-puncturing time increases much more quickly as frame length increases. The key reason is the 32KB L1D cache can't hold all frame data, therefore cache conflict happens much more frequently.

The scaling isn't affected by the cache size because the data is accessed linearly in that function, but for de-interleaving, the data is accessed randomly according de-interleaver table.

For different code rate, the turning point for de-interleaving time should be different because the same cache can hold different length of data for different code rate. The frame length of the data that can be held in cache is calculated as following formula:

(Frame Length) = 32KB/(2 bytes per data)/(Code Rate)

For ½ rate, it is about 32*1024/2/2 = 8192.

For ½ rate, it is about 32*1024/2/2 = 8192.

For 1/4 rate, it is about 32*1024/2/4 = 4096.

For 1/5 rate, it is about 32*1024/2/5 = 3276.

Please note the input data to the de-interleaving and data formatting function is in 16-bit format, 2 byte for every coding bit. More details will be described in the following sections.

Figure 8 compares the pre-processing time vs. code rate. It proves the above analysis.

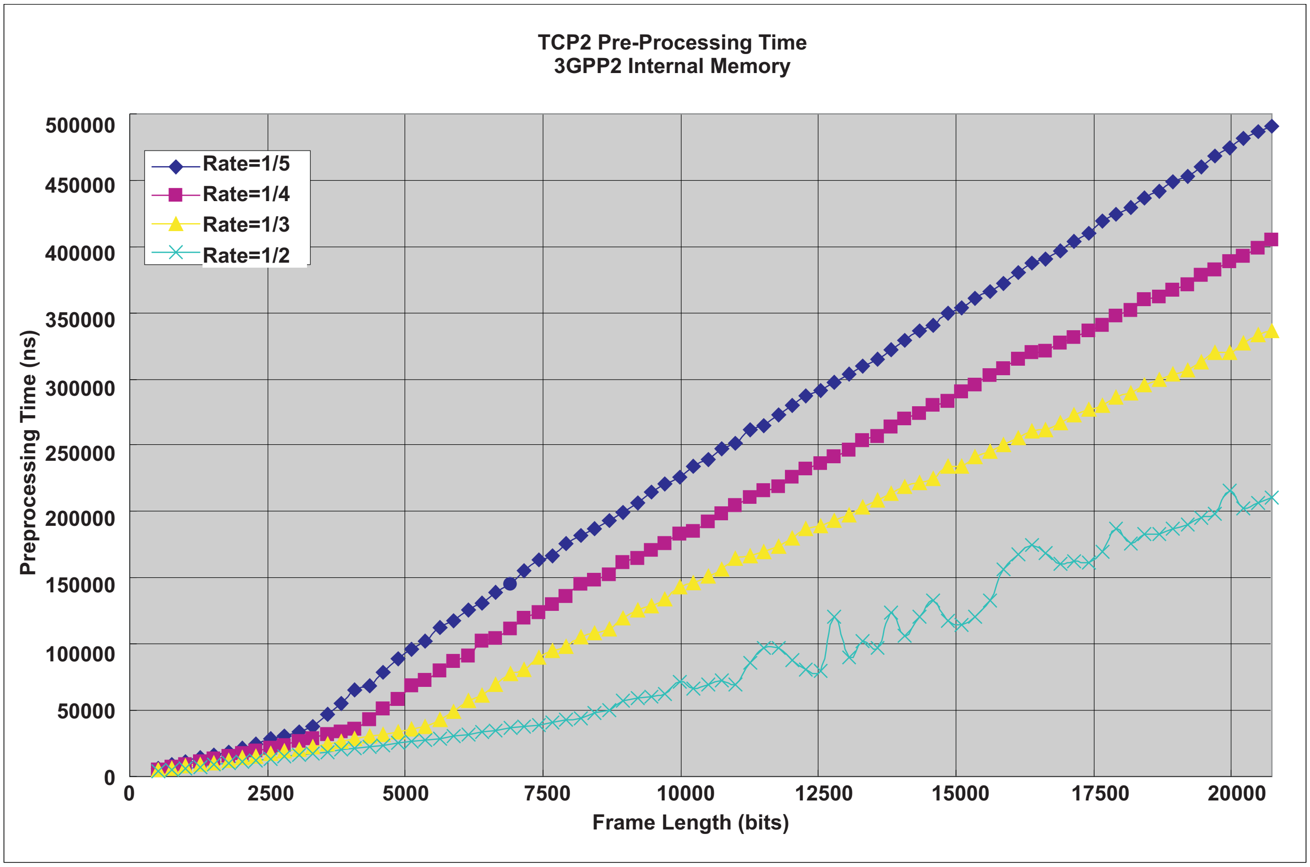

Figure 8. Effect of Code Rate on 3GPP2 Pre-Processing Time

Figure 8 shows the pre-processing for 3GPP2 is almost proportional to code rate, because more data need to be processed for smaller code rate. Please note, 3GPP only uses 1/3 rate.

Whereas, the TCP2 decoding time has nothing to do with the code rate because the TCP2 always treats all input data as 1/5 rate.

The following sections introduce more details about the pre-processing.

5.1 Scale Input Data

The inputs to the TCP coprocessor (channel soft symbols/systems and parities) have to be scaled by a factor:

$$\text{Scaling factor} = \frac{2a}{\sigma^2} \tag{1}$$

a is the channel fading/amplifying factor, σ^2 is the variance of received data. That these can be estimated in different ways depends on system implementation. The straightforward way is to estimate it according the input soft symbol data.

Assume 0 → a and 1→ -a, denote the received soft symbol as X, then:

$$X = \begin{cases} a + noise\left(OriginalBit = 0\right) \\ -a + noise\left(OriginalBit = 1\right) \end{cases} \tag{2}$$

If the original bit = 0, the fading/amplifying factor can be estimated as:

$a = \overline{X - noise} = \overline{X} - \overline{noise} = \overline{X}$

If original bit =1, the fading/amplifying factor can be estimated as:

$a = \overline{-X + noise} = \overline{-X} + \overline{noise} = \overline{-X}$

Since the original bits are not known for receiver, the approximate estimation is:

$a = |\overline{X}|$

This transformation is correct only if following conditions are met.

$$X = \begin{cases} (a + noise) \geq 0 \\ (-a + noise) \leq 0 \end{cases} \qquad (3)$$

The variance can be estimate in same way as:

$\sigma^2 = \overline{(X)}^2 - \overline{(|X|)}^2$

So, the scaling factor can be computed as:

$$Scaling\ Factor = \frac{2 \times |\overline{x}|}{\overline{x^2} - |\overline{x}|^2} = \frac{2 \times \frac{\sum |X|}{Len}}{\frac{\sum x^2}{Len} - \left(\frac{\sum |X|}{Len}\right)^2} = \frac{2 \times \left(\sum |X| \times Len\right)}{\left(\sum x^2\right) \times Len - \left(\sum |X|^2\right)} \qquad (4)$$

In this test, scaling factor is denoted as unsigned Q2.6. The data after scaling is represented as Q14.2 format. The following is the codes for scaling.

```
void TCP2_ScaleData(Int8 * restrict InData, Int16 * restrict OutData, Uint32  Length)
{
Uint32 I, uiAbsSum=0, uiSquareSum=0, uiAbsData0, uiAbsData1;
Uint32 uiEstLength, uiStep, uiRightShiftBits;
double *dpInData= (double *)InData, *dpOutData=(double *)OutData;
double dInput, dTemp0, dTemp1;
unsigned long long llSigma, llMean;
Uint32 uiSigma, uiMean;
Uint32 uiScaleFactor, uiFixScaleFactor_4;

/*Here only use less than 1<<TCP2_SCALE_EST_LEN_BITS (2048 by default)
data to estimate the scale factor. To avoid overflow of 32bit intermedial uiSquareSum,
TCP2_SCALE_EST_LEN_BITS must be no more 16. This should be acceptable since the
scale factor is a statistic value, and 2048 samples should be enough for statistics.
Another benefit is save MIPS. To save more MIPS, this number can be decreased
further according your application */
if(Length<(1<<TCP2_SCALE_EST_LEN_BITS))
{
        uiEstLength= Length;
        uiStep=1;
}
else
{
        uiRightShiftBits= 32-_lmbd(1, Length)-TCP2_SCALE_EST_LEN_BITS;
        //Right shift to get a length less than (1<<TCP2_SCALE_EST_LEN_BITS)
        uiEstLength=Length>>(uiRightShiftBits);
        uiStep= 1<<(uiRightShiftBits);
}
uiEstLength= (uiEstLength/8)*8; //round to multiple of 8
for(I=0; I< uiEstLength/8; I++)
{
        dInput= *dpInData;
        dpInData+=uiStep;
        //Absolute four data
        uiAbsData0=_minu4(_lo(dInput), _sub4(0x00000000,_lo(dInput)));
        uiAbsData1=_minu4(_hi(dInput), _sub4(0x00000000,_hi(dInput)));
        //Sum of four data
        uiAbsSum+=_dotpu4(uiAbsData0, 0x01010101);
        uiAbsSum+=_dotpu4(uiAbsData1, 0x01010101)
        //Sum of the square of four data
        uiSquareSum+=_dotpu4(uiAbsData0, uiAbsData0);
        uiSquareSum+=_dotpu4(uiAbsData1, uiAbsData1);
}
llSigma= _mpy32u(uiSquareSum, uiEstLength)- _mpy32u(uiAbsSum, uiAbsSum);


/* 2*uiAbsSum* uiEstLength is the original numerator,
multiple 64 to get Q6 result, so it is 64 * 2*uiAbsSum* uiEstLength
                                       = uiAbsSum* uiEstLength<<(7)  */
llMean= _mpy32u(uiAbsSum, uiEstLength<<(7) );

//Shift both numerator and denominator to less than 32 bit for division
uiRightShiftBits= 32- _min2(_lmbd(1, _hill(llSigma)), _lmbd(1, _hill(llMean)));
uiSigma= (_hill(llSigma)<<(32-uiRightShiftBits))|(_loll(llSigma)>>uiRightShiftBits);
uiMean= (_hill(llMean)<<(32-uiRightShiftBits))|(_loll(llMean)>>uiRightShiftBits);

//Calculate the scale factor
uiScaleFactor= uiMean/uiSigma;
if(uiScaleFactor>0xff)
          uiScaleFactor=0xff;
else if(uiScaleFactor==0)
          uiScaleFactor=1;
//        printf("ScaleFactor=%3d  ", uiScaleFactor);

//Pack 4 scale factors into one 32 bit scale factor
uiFixScaleFactor_4=_packl4(_pack2(uiScaleFactor, uiScaleFactor),
                           _pack2(uiScaleFactor, uiScaleFactor));

//Scale the data, Length/8+1 is for speculative optimization
dpInData= (double *)InData;
for(I=0; I< (Length/8+1); I++)

{
        dInput= *dpInData++;


        //8bit x 8bit generates 16bit data, Q6
        dTemp0=_mpysu4(_lo(dInput), uiFixScaleFactor_4);
        dTemp1=_mpysu4(_hi(dInput), uiFixScaleFactor_4);

        //Shift right by 4 to get Q2 format
        *dpOutData++=_itod(_shr2(_hi(dTemp0), 4), _shr2(_lo(dTemp0), 4));
        *dpOutData++=_itod(_shr2(_hi(dTemp1), 4), _shr2(_lo(dTemp1), 4));
 }
}
```

Since the original bits are not known for receiver, if noise is strong enough to make the signal across the zero, the above estimation is not accurate. Another way to avoid this is to use pilot bits to estimate the scaling factor because the sign of pilots are known for receiver.

The TCI648x TCP2 offers both max*-log-MAP and max-log-MAP implementations of Turbo decoder, which can be selected on a frame-by-frame basis. The max*-log-MAP should get a little bit better BER performance, but the latter one does not require the input data to be scaled and is therefore more robust in situations where SNR can not be accurately estimated.

5.2 De-Interleave Interleaved Parities, Un-Puncture and Convert to Q4.2 Format

For optimization reason, the de-interleaving interleaved parities, un-puncturing to 1/5 rate and converting to Q4.2 format are combined into one function.

The input data of this function is the output of scaling, signed Q14.2, denoted as ***InputData***.

The interleaved parities must be de-interleaved prior to being sent to TCP2, the purpose of this requirement is to simplify the implementation of TCP2 and reduce the internal memory of TCP2. Assume the Turbo coder output denote as X, A, B, A’, B’, the interleaved parities are the A’ and B’. The de-interleaver table is generated as following:

```
for (i=0;i<tcpParameters->frameLen;i++)
           deintTable[interleaverTable[i]]=I;
```

To de-interleave interleaved parities, the A’ or B’ parities in position I are read according the de-interleaver table, for example, following code get the A’ of position i:

```
A’[I]= A’[DeintTable[i]]= InputData[DeintTable[i]*3+2]
```

The purpose for un-puncturing input data to 1/5 rate is to make TCP2 independent of coding rate, therefore support any rates derived from 1/5. The TCP2 requires the input data to be converted as Q4.2 format, so, the input precision is 6 bits. The benefit of the reduced input precision is reduced internal input memory of TCP2, therefore TCP2 can support longer frame length.

The data is read into 32-bit register, so it becomes Q30.2, to get Q4.2 format, the data are **saturate** left shift 26 bit first:

```
_sshl(InputData[i*3], 26)
```

The 6 most significant bits are the right data we need, the rest of the bits should be cleared:

```
_sshl(InputData [I*3+1], 26)&0xfc000000
```

Then extract or rotate the data to the right position according to the data format required by TCP2:

```
_extu(_sshl(InputData [I*3], 26), 0, 26)
_rotl(_sshl(InputData [I*3+1], 26)&0xfc000000, 12)
```

Following are the codes for these pre-processing:

```
void TCP2_deintUnpunctData(Int16   *restrict InputData,  Uint32 *restrict OutputData,
Uint16 *restrict DeintTable, TCP2_BaseParams *tcpParameters)
{
Uint32 I, uiTemp0, uiTemp1;
Uint32 iFrameLen=tcpParameters->frameLen;
Uint32 iRate=tcpParameters->rate;
double * dOutData= (double *)OutputData;
Uint32 * uipDeIntTable= (Uint32 *)DeintTable;
double dParityIndex;
if(iRate==2)
{
     //iFrameLen+2 is speculative optimization
     for(I=0; I< iFrameLen+2; I+=2)
     {
             //_sshl is for saturate, rotl to specified position
             uiTemp0 = (_extu(_sshl(InputData[i*2], 26), 0, 26))             //X
                     (_rotl(_sshl(InputData[i*2+1], 26)&0xfc000000, 12));    //A
             uiTemp1 = (_extu(_sshl(InputData[i*2+2], 26), 0, 26))|          //X
                     //A', Deinterleave the parity
                     (_rotl(_sshl(InputData[DeintTable[i+1]*2+1], 26)&0xfc000000, 24));
             *dOutData++= _itod(uiTemp1, uiTemp0);
             }
}
else if(iRate==3)

{
/*          #pragma MUST_ITERATE(40, 20730)
     for(I=0; I< iFrameLen; I++)
     {
            //_sshl is for saturate
            uiTemp = (_rotl(_sshl(InputData[i*3], 26), 6))|
                   (_rotl(_sshl(InputData[i*3+1], 26)&0xfc000000, 12))|
                   //Deinterleave the last parities
                   (_rotl(_sshl(InputData[DeintTable[i]*3+2], 26)&0xfc000000, 24));
            OutputData[i] = uiTemp;
     }*/
     for(I=0; I< iFrameLen+2; I+=2)
     {
            //Get Index of Interleaved Parities
            dParityIndex=_mpy2(*uipDeIntTable++, 0x00030003);
            //_sshl is for saturate, rotl to specified position
            uiTemp0 = (_extu(_sshl(InputData[i*3], 26), 0, 26))|               //X
                    (_rotl(_sshl(InputData[i*3+1], 26)&0xfc000000, 12))|        //A
                    //A', Deinterleave the  parities
                    (_rotl(_sshl(InputData[_lo(dParityIndex)+2], 26)&0xfc000000, 24));
            uiTemp1 = (_extu(_sshl(InputData[i*3+3], 26), 0, 26))|             //X
                    (_rotl(_sshl(InputData[i*3+4], 26)&0xfc000000, 12))|        //A
                    //A', Deinterleave the  parities
                    (_rotl(_sshl(InputData[_hi(dParityIndex)+2], 26)&0xfc000000, 24));
            *dOutData++= _itod(uiTemp1, uiTemp0);
     }

}
else if(iRate==4)
{
     for(I=0; I< iFrameLen+2; I+=2)
     {
            //Get Index of Interleaved Parities
            dParityIndex=_mpy2(*uipDeIntTable++, 0x00040004);
            //_sshl is for saturate, rotl to specified position
            uiTemp0 = (_extu(_sshl(InputData[i*4], 26), 0, 26))|               //X
                    (_rotl(_sshl(InputData[i*4+1], 26)&0xfc000000, 12))|        //A
                    (_rotl(_sshl(InputData[i*4+2], 26)&0xfc000000, 18))|        //B
                    //B', Deinterleave the  parities
                    (_rotl(_sshl(InputData[_lo(dParityIndex)+3], 26)&0xfc000000, 30));
            uiTemp1 = (_extu(_sshl(InputData[i*4+4], 26), 0, 26))|             //X
                    (_rotl(_sshl(InputData[i*4+5], 26)&0xfc000000, 12))|        //A
                    //A', Deinterleave the  parities
                    (_rotl(_sshl(InputData[_hi(dParityIndex)+2], 26)&0xfc000000, 24))|
                    //B', Deinterleave the  parities
                    (_rotl(_sshl(InputData[_hi(dParityIndex)+3], 26)&0xfc000000, 30));
            *dOutData++= _itod(uiTemp1, uiTemp0);
     }
}
else if(iRate==5)
{
     for(I=0; I< iFrameLen+2; I+=2)
     {
           //Get Index of Interleaved Parities
           dParityIndex=_mpy2(*uipDeIntTable++, 0x00050005);
           //_sshl is for saturate, rotl to specified position
           uiTemp0 = (_extu(_sshl(InputData[i*5], 26), 0, 26))|                 //X
                   (_rotl(_sshl(InputData[i*5+1], 26)&0xfc000000, 12))|          //A
                   (_rotl(_sshl(InputData[i*5+2], 26)&0xfc000000, 18))|          //B
                   //A', Deinterleave the  parities
                   (_rotl(_sshl(InputData[_lo(dParityIndex)+3], 26)&0xfc000000, 24))|
                   //B', Deinterleave the  parities
                   (_rotl(_sshl(InputData[_lo(dParityIndex)+4], 26)&0xfc000000, 30));
           uiTemp1 = (_extu(_sshl(InputData[i*5+5], 26), 0, 26))|               //X
                   (_rotl(_sshl(InputData[i*5+6], 26)&0xfc000000, 12))|          //A
                   (_rotl(_sshl(InputData[i*5+7], 26)&0xfc000000, 18))|          //B
                   //A', Deinterleave the  parities
                   (_rotl(_sshl(InputData[_hi(dParityIndex)+3], 26)&0xfc000000, 24))|
                   //B', Deinterleave the  parities
                   (_rotl(_sshl(InputData[_hi(dParityIndex)+4], 26)&0xfc000000, 30));
           *dOutData++= _itod(uiTemp1, uiTemp0);
      }
 }
}
```

5.3 Configurations for TCP2 and EDMA3

TI provides chip support library (CSL) for TCP2 and EDMA3 PaRAM parameters calculation/configuration. The TCP2/EDMA3 configuration time is almost constant. The measurement result shows TCP2 parameter calculation and configuration consumes about 700 cycles, and EDMA3 PaRAM configuration consumes about 500 cycles.

EDMA3 PaRAM can also be configured by IDMA, it will reduce the EDMA3 PaRAM configuration time to about 200 cycles, but it increases the complexity of configuration codes.

TCP2 supports chaining the decoding of multiple user channels at a time, to reduce the overhead of TCP2/EDMA3 configuration, multiple user channels should be configured at a time and chain them together. For more details about chaining multiple user channels, please refer to *TMS320TCI648x DSP Turbo-Decoder Coprocessor 2 (TCP2) Reference Guide* (SPRUE10).

6 External Memory vs Internal Memory

All of the above data is measured under the condition that all data buffer for TCP2 are in internal memory. But in the real system, internal memory may not be enough for all data. So, the channel density and CPU load is also measured under the condition that all the data is in external memory. The test is done on TCI6482 EVM with 250 MHz 32-bit DDR2 memory (data rate is 500 M), 256KB L2 cache are enabled.

Figure 9 shows the effect of external memory vs internal memory on TCP2 decoding time.

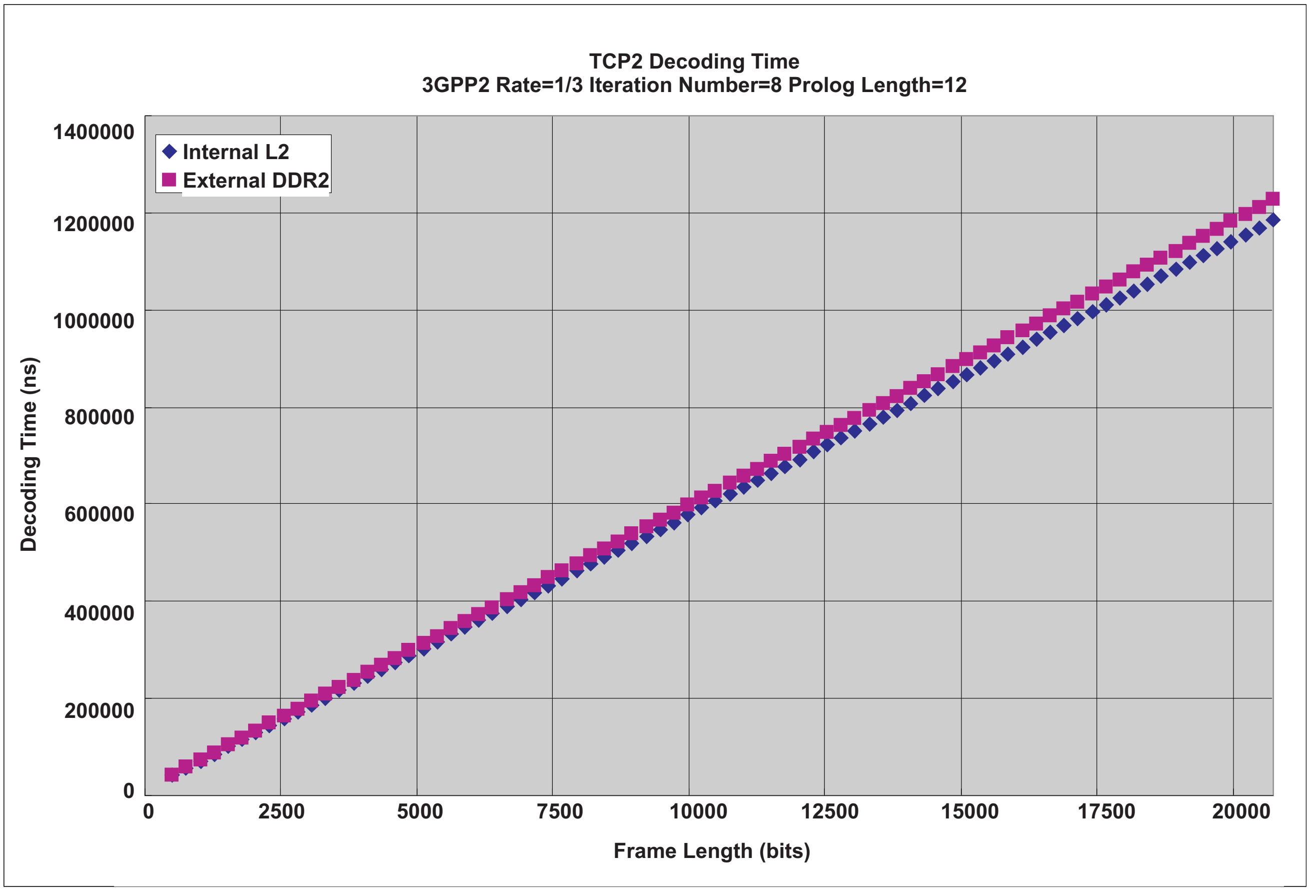

Figure 9. External Memory vs Internal Memory on TCP2 Decoding Time

Figure 9 shows the memory location has little effect on the TCP2 decoding time. Actually, the little effect comes from the EDMA transfer from DSP memory to TCP2 internal memory. Once the data are in the TCP2 internal memory, the decoding time is fixed. Please note, the decoding time in this document includes the EDMA transfer time for TCP2.

Figure 10 shows the effect of external memory vs internal memory on TCP2 pre-processing time.

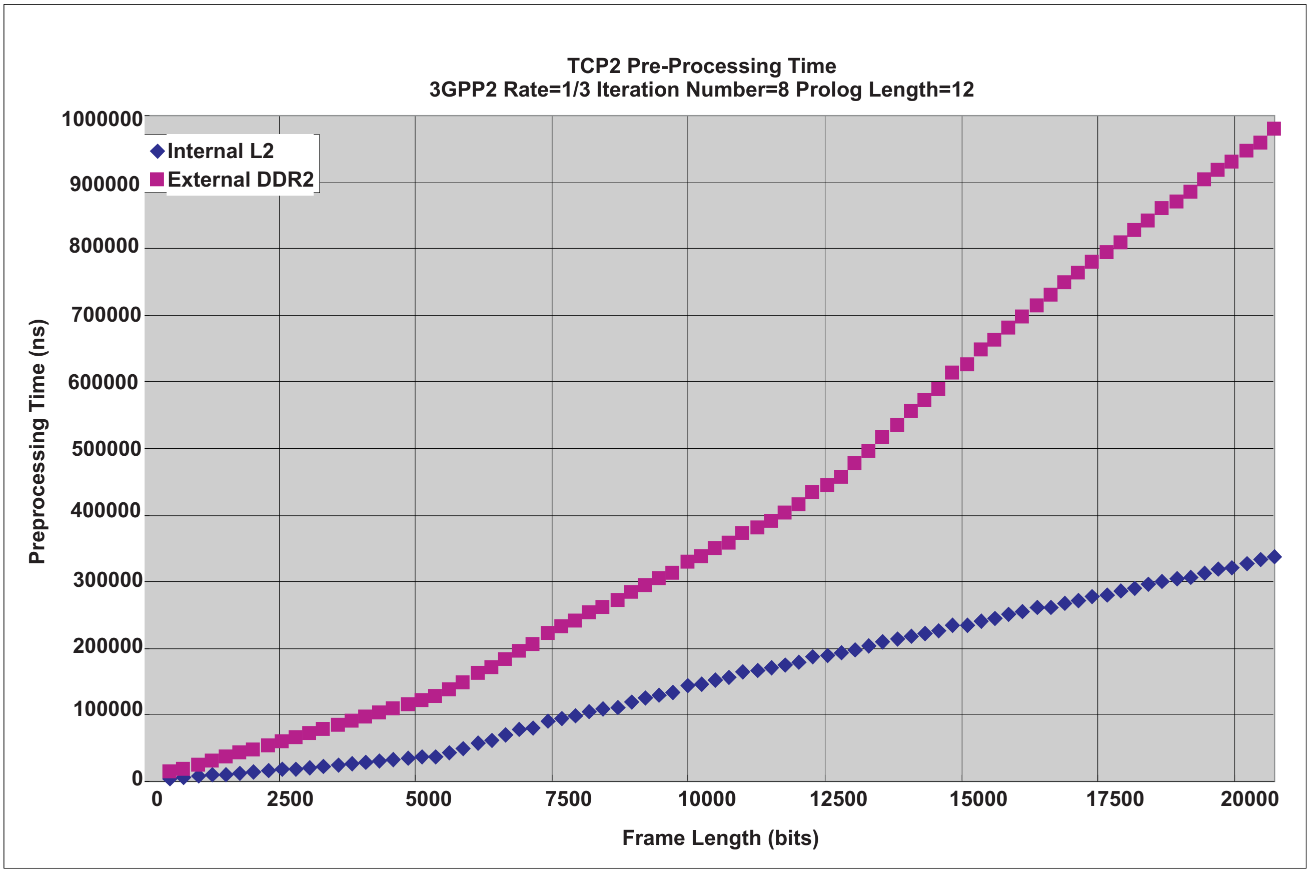

Figure 10. External Memory vs Internal Memory on TCP2 Pre-Processing Time

Figure 10 shows the data memory location has obvious effect on pre-processing. The pre-processing time for the data in external memory is more than three times as the pre-processing time for the data in internal memory.

7 Summary

The TCP2 channel density varies for different channel parameters, but the overall throughput is about 16 Mbps, so the approximate formula for channel density estimation is:

Channel numbers = 16 Mbps/(bit rate per channel)

For examples, 384 K 3GPP channel numbers = 16 Mbps/(384 kbps) =41.7, this is very close to the test result.

The CPU load for pre-processing to support max channel density is about 10% to 20% for all data in internal memory, and the MIPS consumption for data in external memory is about 3 times as data in internal memory.

8 References

- *TMS320TCI648x DSP Turbo-Decoder Coprocessor 2 (TCP2) Reference Guide* (SPRUE10)
- *TMS320C6416 Coprocessors and Bit Error Rates Application Report* (SPRA974)
- 3GPP TS 25.104 V7.4.0, Base Station (BS) Radio Transmission and Reception (FDD)
- 3GPP TS 25.105 V7.2.0, Base Station (BS) Radio Transmission and Reception (TDD)
- 3GPP2 C.S0002-D_v2.0, Physical Layer Standard for cdma2000 Spread Spectrum Systems

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
Low Power Wireless	www.ti.com/lpw	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