Texas Instruments

Application Report
SPRAAG4–November 2006

TMS320TCI648x VCP2 Channel Density

Brighton Feng

ABSTRACT

Viterbi decoder lies at the heart of all of the wireless standards. Viterbi coprocessors (VCP2) are programmable peripherals used to decode convolutional codes. It is integrated into Texas Instruments TMS320TCI648x digital signal processor (DSP). This application report gives the channel density for VCP2 under various conditions, the CPU load for pre-processing VCP2 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/SPRAAG4.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. Convolutional coding techniques are used in all wireless standards. The Viterbi coprocessors (VCP) are programmable peripherals used to decode convolutional codes. It is integrated into some Texas Instruments digital signal processors (DSP). The Viterbi coprocessor in the TMS320TCI100/TMS320C6416 is called VCP1; the Viterbi coprocessor in the TMS320TCI648x is an enhanced revision of VCP1, and is called VCP2.

The channel density of VCP2 is related to several parameters. The main parameters are frame length, rate, and constraint 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 VCP2, include quantization, calculating branch metrics, configuring VCP2/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 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.

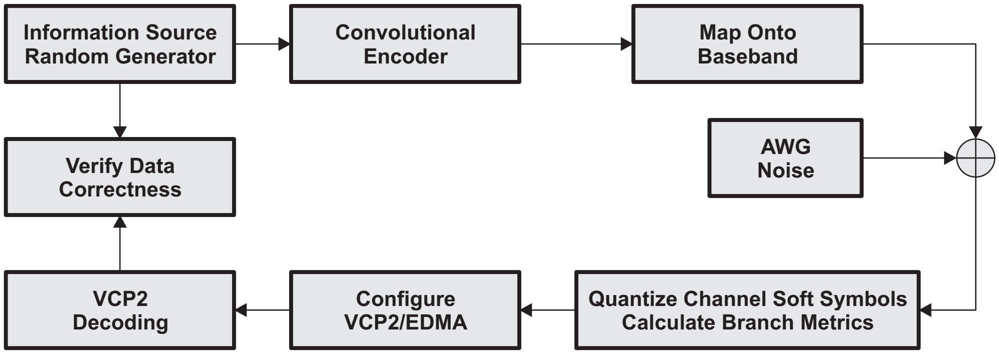

Figure 1. Communication Channel Simulation for VCP2

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

1. Generate the binary data bits (information sequence) to be transmitted through the channel.
2. Generate the binary data bits (information sequence) to be transmitted through the channel.
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 is required before data can be decoded in VCP2, the steps are as following:

1. Quantize channel soft symbols to avoid overflow
2. Combine normalized channel soft decisions to generate branch metrics inputs
3. Calculate VCP2 parameters
4. Configure EDMA3 PaRAM

Pre-processing can be overlapped with VCP2 decoding. Typical scheduling for VCP2 decoding and CPU pre-processing are as following:

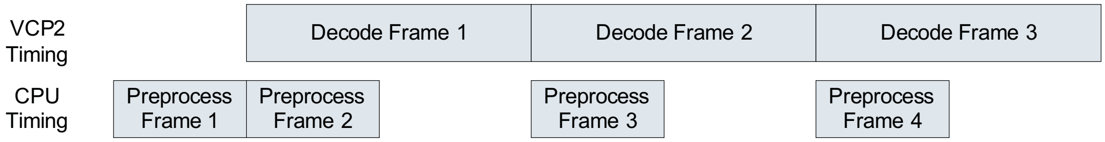

Figure 2. VCP2 Decoding and Pre-Processing Scheduling

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 VCP2 pre-processing, therefore it 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 VCP2 and wait for completion. The decoding time is measured from starting VCP2 until all output data that are read out from VCP2. Please note, the time for EDMA transferring data from/to DSP memory to/from VCP2 internal memory are treated as a part of the decoding time in this document.

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

- Code Rate = 1/3
- Constraint Length = 9 (3G)
- C = 6 × 8 = 48
- Tailed traceback mode or mixed traceback mode are used. Convergent traceback mode is not covered in document since it is rarely used.
- Hard decision is used. Channel density for soft decision is not covered in this document since it is rarely used.
- Output parameters are transferred out from VCP2 for every frame
- All data in internal memory, 32KB L1D and 32KB L1P
- Build options: -o3, no debug

3 Channel Density Data for Typical Scenarios

The following sections give channel density data for some typical scenarios in 3GPP and 3GPP2 system.

The formulas used to calculate channel density data 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 convolutional encoder or output bits from convolutional decoder. These bits are composed of original information bits, CRC bits or other control bits depend on the standard.

3.1 3GPP

Table 1 gives the channel density data of VCP2 for the reference measurement channels listed in 3GPP TS 25.104 V7.4.0, Base Station (BS) radio transmission and reception (FDD), and examples of transport format attributes for AMR speech codec in 3GPP TS 25.302 V7.1.0, services provided by the physical layer.

Table 1. Typical Channel Density for 3GPP

Channel Type	Information Rate (kbps)	Frame Length (bits)	Frame Interval (ms)	Rate	Decoding Time (ns)	Channel Density	Throughput (Mbps)	Pre-Processing Time (ns)	CPU load for Pre-Processing
DCCH	2.4	112	40	1/3	12368	3234.2	7.762	1472	11.90%
DCCH	3.4	164	40	1/3	17816	2245.2	7.762	1862	10.45%
DTCH	12.2	260	20	1/3	27422	729.3	8.898	2208	8.05%
AMR	12.2	89	20	1/3	10400	684.3	8.349	1389	13.61%
		103		1/3	11582			1442	
		60		1/3	7244			1148	
AMR	10.2	73	20	1/3	8828	790.6	8.065	1327	15.28%
		99		1/3	11186			1435	
		40		1/3	5282			1104	
AMR	7.95	83	20	1/3	9614	1040.1	8.269	1369	14.24%
		84		1/3	9614			1369	
AMR	7.4	69	20	1/3	8438	1084.1	8.023	1312	14.59%
		87		1/3	10010			1380	
AMR	6.7	66	20	1/3	8042	1185.5	7.943	1300	15.62%
		76		1/3	8828			1335	
AMR	5.9	63	20	1/3	7640	1308.9	7.723	1292	16.91%
		63		1/3	7640			1292	
AMR	5.15	57	20	1/3	7244	1418.6	7.306	1279	17.97%
		54		1/3	6854			1255	
AMR	4.75	50	20	1/3	6458	1502.4	7.136	1240	18.70%
		50		1/3	6854			1250	
AMR	1.8	47	20	1/3	6068	3296.0	5.933	1232	20.30%

3.2 3GPP2

Table 2 gives the channel density data of VCP2 for reverse channels listed in 3GPP2 C.S0002-D_v2.0_ Physical Layer Standard for cdma2000 Spread Spectrum Systems.

Table 2. Typical Channel Density for 3GPP2

Information Rate (kbps)	Frame Length (bits)	Frame Interval (ms)	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 1								
1.2	16	20	1/3	2924	6839.9	8.208	1122	38.37%
2.4	40	20	1/3	5282	3786.4	9.087	1208	22.87%
4.8	88	20	1/3	10010	1998.0	9.590	1382	13.81%
9.6	184	20	1/3	19778	1011.2	9.708	1939	9.80%
Reverse Dedicated Control Channel Structure for Radio Configuration 2								
1.8	28	20	1/2	4100	4878.0	8.780	1088	26.54%
3.6	64	20	1/2	7640	2617.8	9.424	1150	15.05%
7.2	136	20	1/2	14894	1342.8	9.424	1477	9.92%
14.4	280	20	1/2	29390	680.5	9.799	1739	5.92%
Reverse Dedicated Control Channel Structure for Radio Configuration 3/5								
1.5	22	20	1/4	3710	5390.8	8.086	1218	32.83%
2.7	46	20	1/4	6068	3296.0	8.899	1362	22.45%
4.8	88	20	1/4	10010	1998.0	9.590	1594	15.92%
9.6	184	20	1/4	19778	1011.2	9.708	2339	11.83%
19.2	376	20	1/4	38996	512.9	9.847	3443	8.83%
38.4	760	20	1/4	77606	257.7	9.896	5651	7.28%

Table 2. Typical Channel Density for 3GPP2 (continued)

Information Rate (kbps)	Frame Length (bits)	Frame Interval (ms)	Rate	Decoding Time (ns)	Channel Density	Throughput (Mbps)	Pre-Processing Time (ns)	CPU load for Pre-Processing
76.8	1528	20	1/4	154826	129.2	9.921	10067	6.50%
153.6	3064	20	1/4	309200	64.7	9.935	18899	6.11%
Reverse Dedicated Control Channel Structure for Radio Configuration 4/6								
1.8	28	20	1/4	4100	4878.0	8.780	1256	30.63%
3.6	64	20	1/4	7640	2617.8	9.424	1462	19.14%
7.2	136	20	1/4	14894	1342.8	9.668	2063	13.85%
14.4	280	20	1/4	29390	680.5	9.799	2891	9.84%
28.8	568	20	1/4	58388	342.5	9.865	4547	7.79%
57.6	1144	20	1/4	116390	171.8	9.898	7859	6.75%
115.2	2296	20	1/4	232034	86.2	9.930	14483	6.24%
230.4	4600	20	1/4	463532	43.1	9.941	27731	5.98%
460.8	9208	20	v	926696	21.6	9.945	57688	6.23%

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

4 The Effects of Different Parameters on VCP2 Decoding Time

The channel density of VCP2 is related to several parameters. The main parameters are frame length, rate, constraint length and convergent length. 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 and Code Rate

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

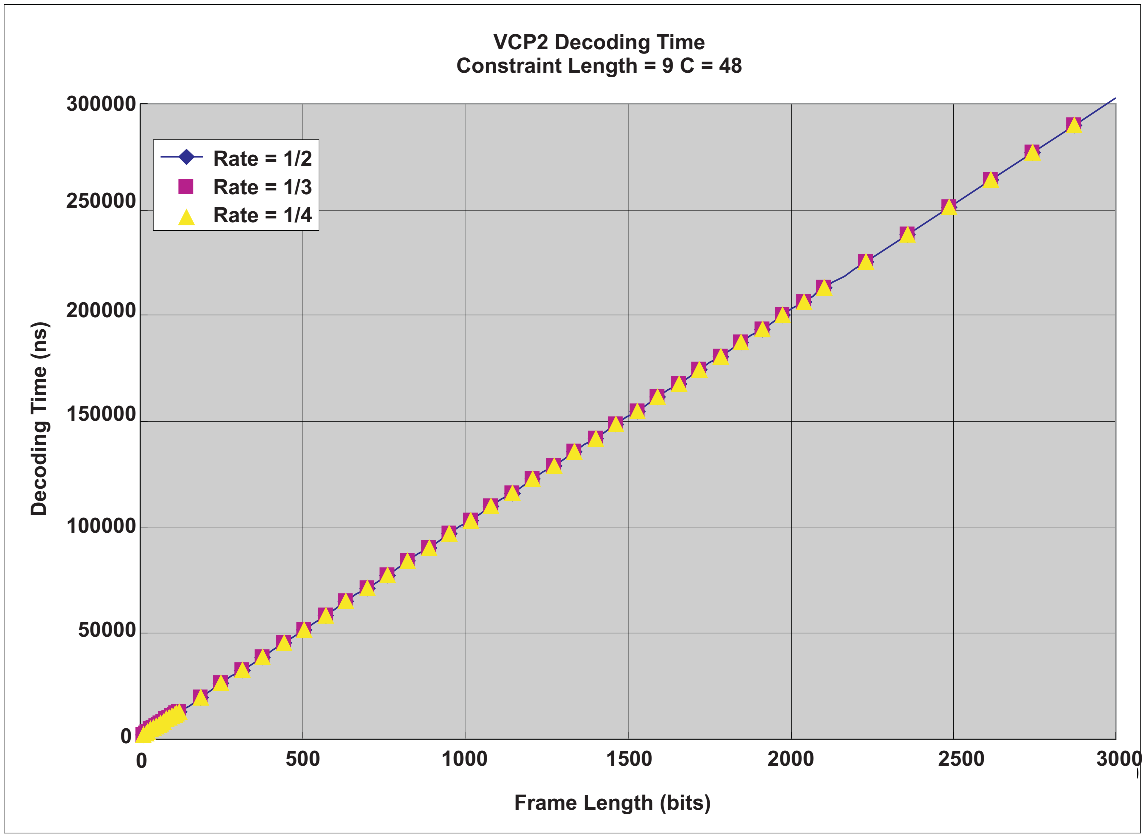

Figure 3. Effects of Frame Length and Rate on VCP2 Decoding Time

Figure 3 shows the VCP2 decoding time is proportional to frame length, i.e. the channel density is inversely proportional to frame length.

Figure 3 shows that the code rate has no effect on VCP2 decoding time or channel density.

4.2 Effects of Constraint Length

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

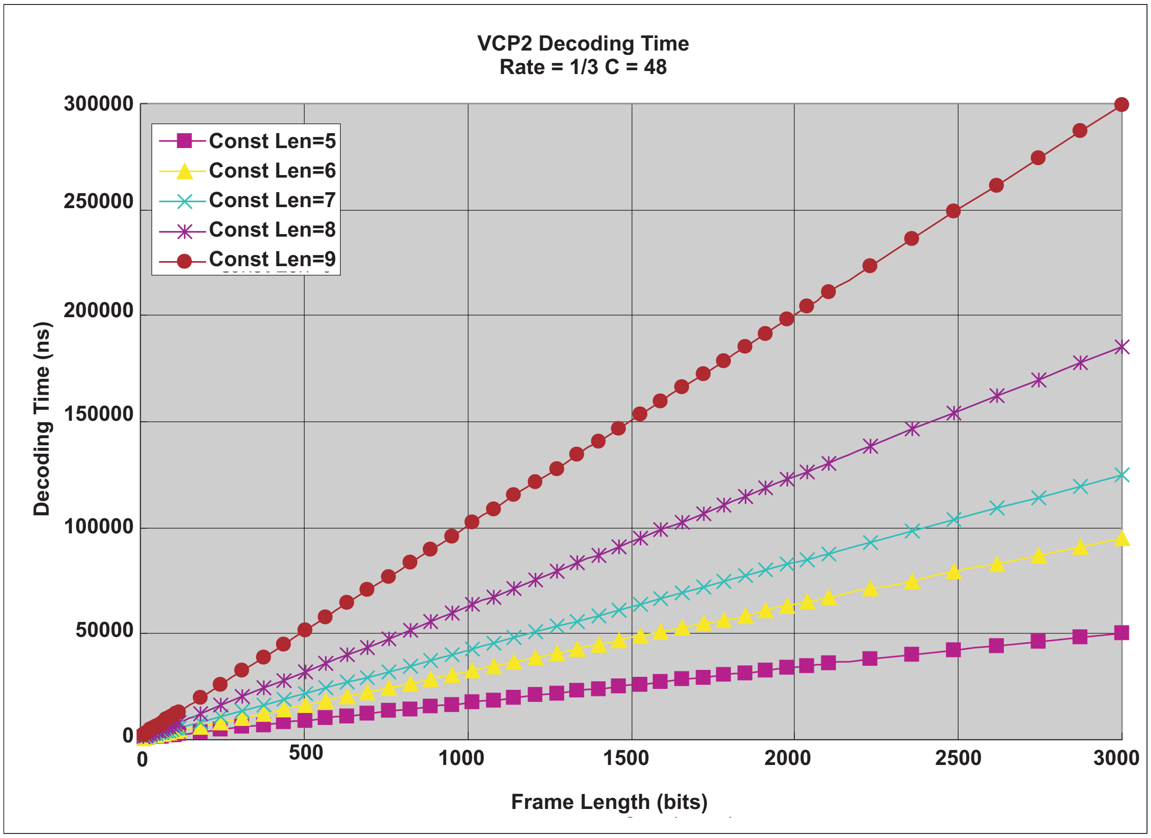

Figure 4. Effect of Constraint Length on VCP2 Decoding Time

Figure 4 shows the decoding time increase with constraint length, i.e. the channel density decreases with constraint length.

4.3 Effects of C (convergent length)

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

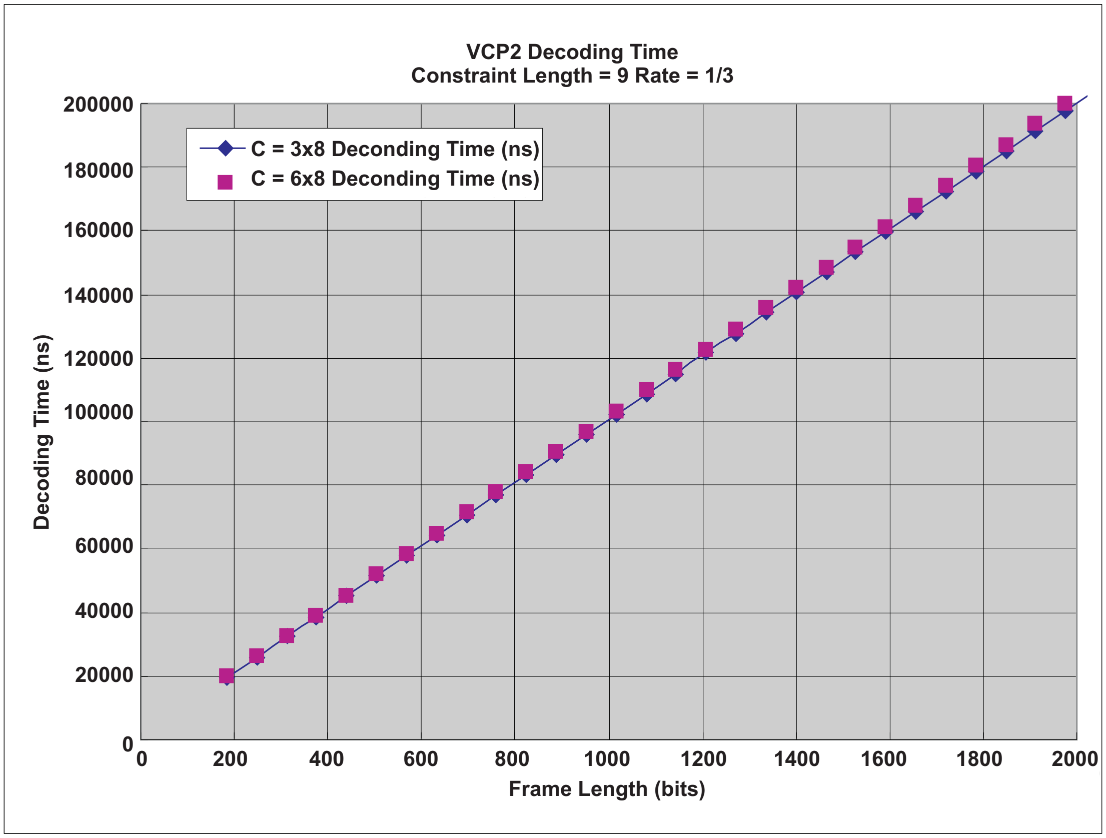

Figure 5. Effect of Convergent Length on VCP2 Decoding Time

Figure 5 shows the convergent length has little affect on the decoding time or channel density.

The reason for the small difference in decoding time is due to the traceback processing. Smaller convergent lengths have a smaller decoding time. But, generally the state metric processing time will be larger than the traceback processing time. Therefore, the overall VCP decoding time has minimal difference due to convergent lengths.

Longer convergent length may improve BER performance and it has little effect on channel density. In addition, convergent length more than 5×(constraint length) can not improve BER more, so C = 48 is recommended for constraint length = 9.

4.4 Effects of Other Parameters

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 output parameter is only a small part of overall VCP2 decoding time. All measurements for this document always transfer output parameters.

5 MIPS Consumption of Data Pre-Processing for VCP2

Pre-processing is required before data can be decoded in VCP2. 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.

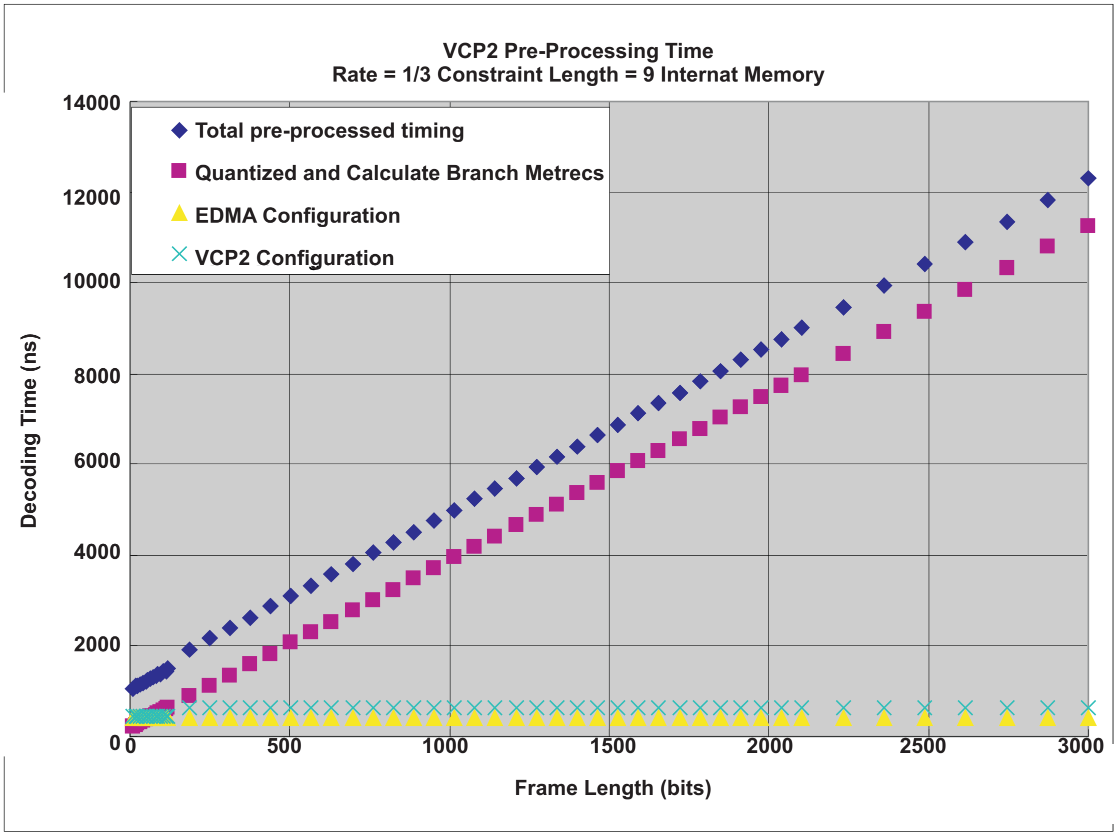

Figure 6. VCP2 Pre-Processing Time for 3GPP and Its Partition

Figure 6 shows that the quantization and calculating branch metrics time is almost proportional to frame length. VCP2 and EDMA3 configuration time is almost constant, totally about 1000 cycles. The following sections discuss processing in more detail.

Figure 7 shows the pre-processing time vs. code rate.

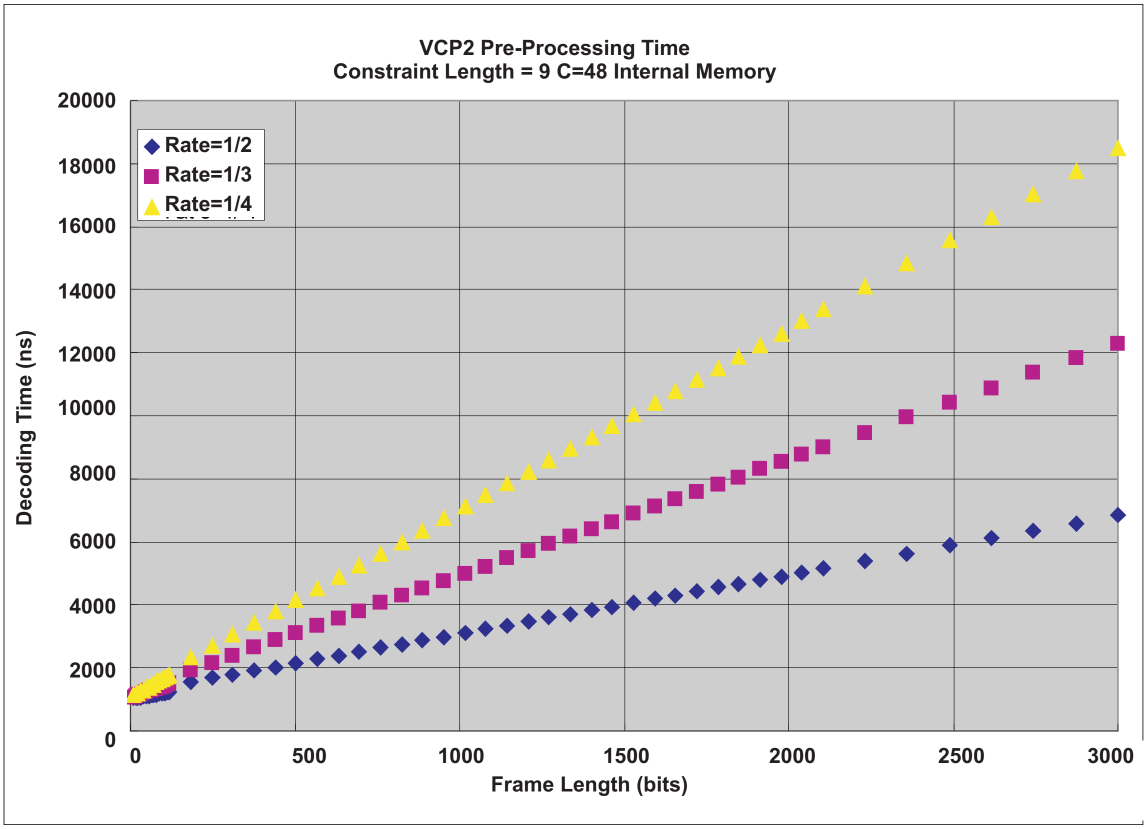

Figure 7. Effect of Code Rate on Pre-Processing Time

Figure 7 shows the pre-processing time is almost proportional to code rate, because more data needs to be processed for smaller code rates. Whereas, the VCP2 decoding time isn't affected by the code rate.

The following sections introduce details about pre-processing.

5.1 Quantize Input Data and Calculate Branch Metrics

Input data should be quantized to avoid overflow according to the following requirement.

Table 3. VCP2 Input Data Quantization

Rate	Range
1/2	[-64;+63]
1/3	[-42;+42]
1/4	[-32;+31]

In this test, we scale the input data to the above range, the

ScalingFactor = (MaxRange)/(MaxInputValue)

System designer may use other method to quantize input data.

Following are the codes for quantization.

```
if(Rate==3)
         uiRange=42;
else if(Rate==4)
         uiRange=31;
else if(Rate==2)
         uiRange=63;
else
         return;

for(i=0; i<Length*Rate/8; i++)
{
         dInput=*dpInData++;
         //Get absolute value
         uiAbsDataLo=  _minu4(_lo(dInput), _sub4(0x00000000,_lo(dInput)));
         uiAbsDataHi= _minu4(_hi(dInput), _sub4(0x00000000,_hi(dInput)));
         uiMaxLo=_maxu4(uiMaxLo,uiAbsDataLo);
         uiMaxHi=_maxu4(uiMaxHi, uiAbsDataHi);
}
uiMaxLo= _maxu4(uiMaxLo, uiMaxHi);
uiMaxLo= _maxu4(uiMaxLo, _packlh2(uiMaxLo, uiMaxLo));
uiMaxLo= _maxu4(uiMaxLo, _swap4(uiMaxLo))&0xff;

//Compute scaling factor
uiScaleFactor= 128*uiRange/uiMaxLo; //x128 to get Q7 formart
if(uiScaleFactor>128)
         uiScaleFactor=128;
```

The key operation of branch metrics calculation can be treated as complex multiplication. For example, for rate=1/2, the

BM0 = $r_0 + r_1$

BM1 = $r_0 - r_1$

These operations are transformed as

$(r_0 + r_{1j})*(1 + j) = (r_0 - r_1) + (r_0 + r_1) = \text{BM1} + \text{BM0}j$

The TCI6482 introduces new complex multiplication instruction, so we utilize them to optimize the branch metrics calculation. Following are the codes for it.

```
//Pack 4 scale factors into one 32 bit scale factor
uiFixScaleFactor_4=_packl4(_pack2(uiScaleFactor, uiScaleFactor),
                   _pack2(uiScaleFactor, uiScaleFactor));
if(Rate==3)
{
        uipOutData= (Uint32 *)BranchMetrics;
        //make length to be multiple of 2, output buffer maybe speculative overwrite for
optimization
```

```
        Length=(Length+1)/2*2;
        #pragma UNROLL(2)
        for(i=0; i<Length; i++)
        {
                //Read 4 bytes per time, only use 3 bytes, which denote one encoded bit
                uiInput3_0=  _mem4(&softInput[i*3]);

                //Scale
                //8bit x 8bit generat 16bit data, Q0 x Q7 generate Q7
                dTemp3_0=_mpysu4(uiInput3_0, uiFixScaleFactor_4);
                //shift r2 to Q5 format
                uiInput2= _shr2(_hi(dTemp3_0),2);

                //Calculate branch metrics for fist bit
                //swap (r1,r0) to (r0, r1)
                uiTemp1_0=_packlh2(_lo(dTemp3_0), _lo(dTemp3_0));
                //(r0+r1j)*(1+j)=(r0-r1)+(r0+r1)j
                /*0x40004000 denotes (1+1j) with Q14 format, so Q7 x Q14= Q21,
                cmpyr only save higher 16 bits, so the result is Q5*/
                uiTemp1_0=_cmpyr(uiTemp1_0, 0x40004000);

                //pack to get (r0+r1, r2)
                uiBM1_0= _pack2(uiTemp1_0, uiInput2);
                //((r0+r1)+r2j)*(1+j)=(r0+r1-r2)+(r0+r1+r2)j
                /*0x08000800 denotes (1+1j) with Q11 format, so Q5 x Q11= Q16,
                cmpyr only save higher 16 bits, so the result is Q0, just is the result we need*/
                uiBM1_0=_cmpyr(uiBM1_0, 0x08000800);

                //pack to get (r0-r1, r2)
                uiBM3_2= _packhl2(uiTemp1_0, uiInput2);
                //((r0-r1)+r2j)*(1+j)=(r0-r1-r2)+(r0-r1+r2)j
                uiBM3_2=_cmpyr(uiBM3_2, 0x08000800);

                //Pack the lower bytes as output
                *uipOutData++=_packl4(uiBM3_2, uiBM1_0);
        }
}
else if(Rate==4)
{
        Uint32 uiTemp2_1, uiInput3, uiInput1_0, uiInput1, uiInput0, uiBM7_4;
        dpOutData= (double *)BranchMetrics;
        //make length to be multiple of 2, output buffer may be speculative overwrite for
optimization
        Length=(Length+1)/2*2;
        #pragma UNROLL(2)
        for(i=0; i<Length; i++)
        {
                //Read 4 bytes, which denote one encoded bits
                uiInput3_0=  _amem4(&softInput[i*4]);

                //Scale
                //8bit x 8bit generat 16bit data, Q0 x Q7 generate Q7
                dTemp3_0=_mpysu4(uiInput3_0, uiFixScaleFactor_4);

                //Calculate branch metrics for fist bit
                //pack to get (r1, r2)
                uiTemp2_1=_packhl2(_lo(dTemp3_0), _hi(dTemp3_0));
                //(r1+r2j)*(1+j)=(r1-r2)+(r1+r2)j
                /*0x40004000 denotes (1+1j) with Q14 format, so Q7 x Q14= Q21,
                cmpyr only save higher 16 bits, so the result is Q5*/
                uiTemp2_1=_cmpyr(uiTemp2_1, 0x40004000);

                //shift r3 to Q5 format
                uiInput3= _shr2(_hi(dTemp3_0),2);
                //pack to get (r1+r2, r3)
                uiBM1_0= _packlh2(uiTemp2_1, uiInput3);
                //((r1+r2)+r3j)*(1+j)=(r1+r2-r3)+(r1+r2+r3)j
                /*0x08000800 denotes (1+1j) with Q11 format, so Q5 x Q11= Q16,
                cmpyr only save higher 16 bits, so the result is Q0, just is the result we need*/
                uiBM1_0=_cmpyr(uiBM1_0, 0x08000800);

                //pack to get (r1-r2, r3)
                uiBM3_2= _packh2(uiTemp2_1, uiInput3);
                //((r1-r2)+r3j)*(1+j)=(r1-r2-r3)+(r1-r2+r3)j
                uiBM3_2=_cmpyr(uiBM3_2, 0x08000800);
                //Pack the lower bytes
                uiBM3_0=_packl4(uiBM3_2, uiBM1_0);

                //shift r1, r0 7 bits to get Q0
                uiInput1_0=_shr2(_lo(dTemp3_0),7);
                uiInput1_0=_packl4(uiInput1_0, uiInput1_0);
                //pack to get packed 4 byte (r0,r0,r0,r0)
                uiInput0=_packl4(uiInput1_0, uiInput1_0);
                uiBM3_0= _add4(uiInput0, uiBM3_0);

                //pack to get packed 4 byte (r1,r1,r1,r1)
                uiInput1=_packh4(uiInput1_0, uiInput1_0);
                uiBM7_4= _sub4(_sub4(uiBM3_0, uiInput1), uiInput1);

                *dpOutData++= _itod(uiBM7_4, uiBM3_0);
        }
}
else    //Rate=2
{
        Uint32 uiBM7_6, uiBM5_4;
        dpOutData= (double *)BranchMetrics;
        //make length to be multiple of 8, output buffer maybe speculative overwrite for
optimization
        Length=(Length+7)/8*8;
        #pragma UNROLL(2)
        for(i=0; i<(Length)/4; i++)
        {
                //Read 8 bytes, which denote 4 encoded bits
                dInput=  _amemd8(&softInput[i*8]);

                //Scale
                //8bit x 8bit generat 16bit data, Q0 x Q7 generate Q7
                dTemp3_0=_mpysu4(_lo(dInput), uiFixScaleFactor_4);

                //Calculate branch metrics for fist bit
                //swap (r1,r0) to (r0, r1)
                uiBM1_0=_packlh2(_lo(dTemp3_0), _lo(dTemp3_0));
                //(r0+r1j)*(1+j)=(r0-r1)+(r0+r1)j
                /*0x02000200 denotes (1+1j) with Q9 format, so Q7 x Q9= Q16,
                cmpyr only save higher 16 bits, so the result is Q0*/
                uiBM1_0=_cmpyr(uiBM1_0, 0x02000200);

                uiBM3_2=_packlh2(_hi(dTemp3_0), _hi(dTemp3_0));
                uiBM3_2=_cmpyr(uiBM3_2, 0x02000200);

                //8bit x 8bit generat 16bit data, Q0 x Q7 generate Q7
                dTemp7_4=_mpysu4(_hi(dInput), uiFixScaleFactor_4);

                uiBM5_4=_packlh2(_lo(dTemp7_4), _lo(dTemp7_4));
                uiBM5_4=_cmpyr(uiBM5_4, 0x02000200);
                uiBM7_6=_packlh2(_hi(dTemp7_4), _hi(dTemp7_4));
                uiBM7_6=_cmpyr(uiBM7_6, 0x02000200);

                //Pack the lower bytes as output
                *dpOutData++=_itod(_packl4(uiBM7_6, uiBM5_4),_packl4(uiBM3_2, uiBM1_0));
        }
}
```

5.2 Configurations for VCP2 and EDMA3

TI provides chip support library (CSL) for VCP2 and EDMA3 PaRAM parameters calculation/configuration. The VCP2/EDMA3 configuration time is almost constant, the measurement result shows VCP2 parameter calculation and configuration consumes about 600 cycles, and EDMA3 PaRAM configuration consumes about 400 cycles.

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

VCP2 supports chaining the decoding of multiple user channels at a time, to reduce the overhead of VCP2/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 Viterbi-Decoder Coprocessor 2 (VCP2) Reference Guide* (SPRUE09).

6 External Memory vs Internal Memory

All of the above data is measured under the condition that all data buffer for VCP2 are in internal memory. But in 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 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 8 shows the effect of external memory vs internal memory on VCP2 decoding time.

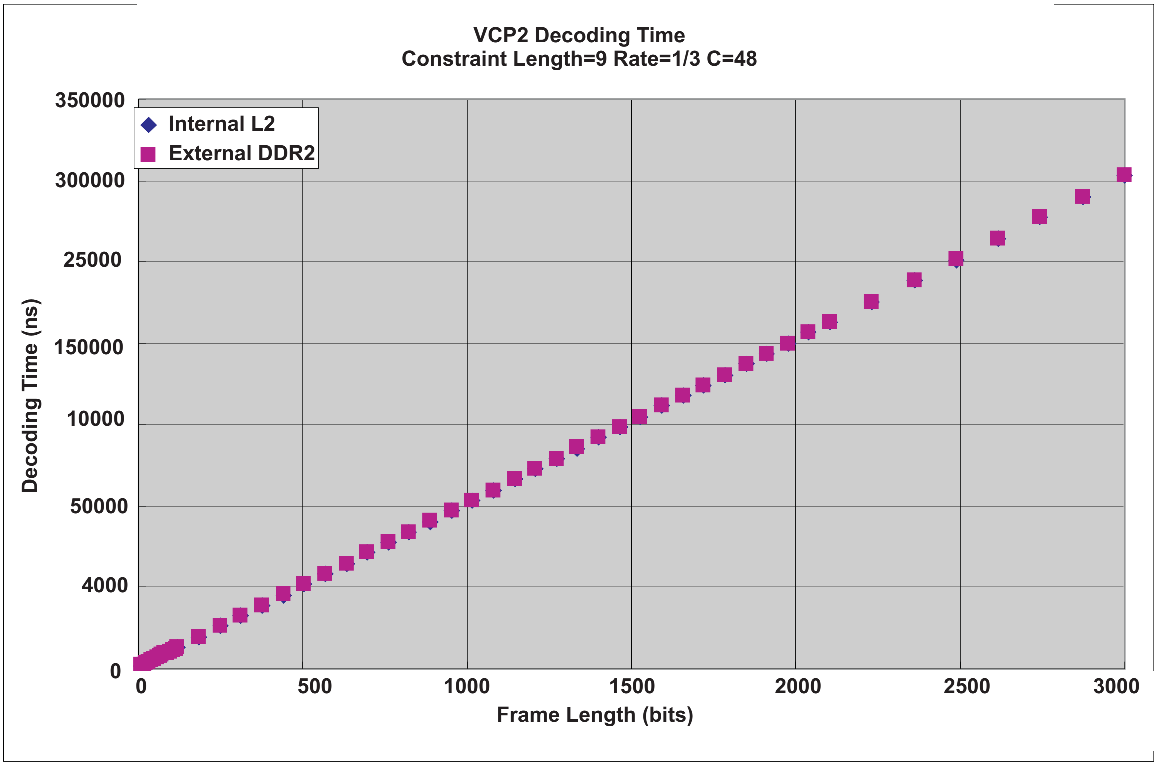

Figure 8. External Memory vs Internal Memory on VCP2 Decoding Time

Figure 8 shows the memory location has little effect on VCP2 decoding time. Actually, little effect comes from the EDMA transfer from DSP memory to VCP2 internal memory. Once the data are in the VCP2 internal memory, the decoding time is fixed. Please note, the decoding time in this document includes the EDMA transfer time for VCP2.

Figure 9 shows the effect of external memory vs internal memory on VCP2 pre-processing time.

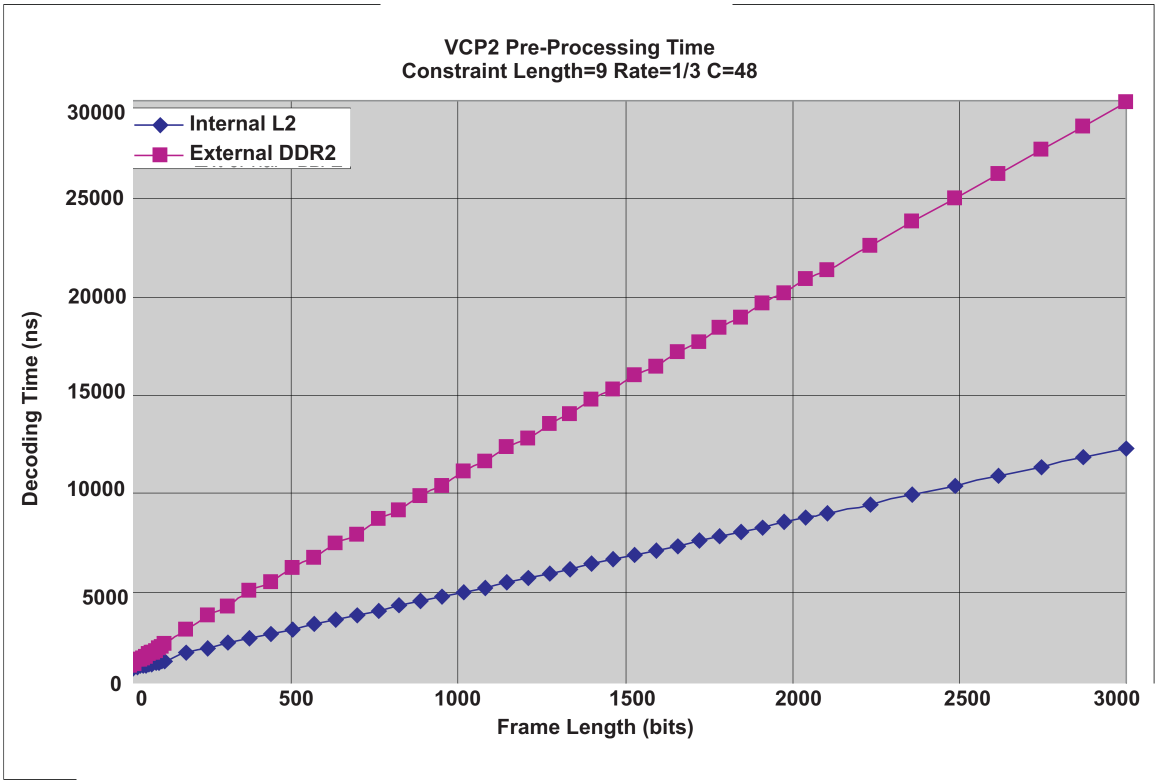

Figure 9. External Memory vs Internal Memory on VCP2 Pre-Processing Time

Figure 9 shows the data memory location has obvious effect on pre-processing. The pre-processing time for the data in external memory is more than two times as much as the pre-processing time for the data in internal memory.

7 Summary

The VCP2 channel density varies for different channel parameters, but the overall throughput is about 8 Mbps, so the approximate formula for channel density estimation is:

Channel Numbers= 8 Mbps/(bit rate per channel)

For examples, 12.2K 3GPP channel numbers = 8 Mbps/(12.2kbps) = 655, this is approximate to the test result.

The CPU load for pre-processing to support max channel density is about 10 % to 20 % for typical scenarios if all data is in internal memory, and the MIPS consumption for data in external memory is about 2 times as much as data in internal memory.

8 References

- *TMS320TCI648x DSP Viterbi-Decoder Coprocessor 2 (VCP2) Reference Guide* (SPRUE09)
- *TMS320C6416 Coprocessors and Bit Error Rates* (SPRA974)
- 3. 3GPP TS 25.104 V7.4.0, Base Station (BS) Radio Transmission and Reception (FDD)
- 3GPP TS 25.302 V7.1.0, Services provided by the Physical Layer
- 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