ABSTRACT

TVP5146 VBI SLICER EXAMPLE: EXTRACTING PDC AND VPS DATA

PDC (Program Delivery Control) is a signal intended for controlling VCRs. It is used only with the PAL standard and it is received as part of WST (World Standard Teletext). PDC is used to control VCR recording by informing a PDC-equipped VCR when a particular program starts and ends.

VPS (Video Program System) is an earlier scheme for controlling VCR recording. VPS is used only with the PAL standard and it is received during line 16 of the VBI (Vertical Blanking Interval).

In this application note, signals which model actual PDC and VPS signals are inserted into the VBI of a PAL composite video signal which is input to the TVP5146 Video Decoder. The VBI slicer data filter feature is used to extract only the desired PDC data packets. Complete register settings are presented, as well as C source code for programming the device and extracting the PDC and VPS data using status polling.
1  **I2C Register Settings**

This section lists the register values to setup the TVP5146 for extraction of PDC and VPS data.

1.1  **Direct Registers**

Table 1 - Direct Register Settings lists the register values to setup the direct I2C registers of the TVP5146 for extraction of PDC and VPS data. The address, data and description are given for each register.

<table>
<thead>
<tr>
<th>TVP5146 DIRECT ADDRESS (HEX)</th>
<th>DATA (HEX)</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0C</td>
<td>Video Input = CVBS CH4</td>
</tr>
<tr>
<td>02</td>
<td>00</td>
<td>Standard = Auto-switch</td>
</tr>
<tr>
<td>06</td>
<td>40</td>
<td>Pedestal off</td>
</tr>
<tr>
<td>31</td>
<td>05</td>
<td>GLCO in ADI mode</td>
</tr>
<tr>
<td>34</td>
<td>11</td>
<td>Outputs active</td>
</tr>
<tr>
<td>35</td>
<td>FA</td>
<td>GLCO and FID active</td>
</tr>
<tr>
<td>B1</td>
<td>F0</td>
<td>Magazine + Row LS Bit = 000 0 ( Magazine 8 )</td>
</tr>
<tr>
<td>B2</td>
<td>FF</td>
<td>Row 4 MS Bits = 1111 ( Row = 11110 = 30 dec )</td>
</tr>
<tr>
<td>B3</td>
<td>F2</td>
<td>Type = 0010 ( Type 2 )</td>
</tr>
<tr>
<td>B4</td>
<td>00</td>
<td>Filter1 Mask4</td>
</tr>
<tr>
<td>B5</td>
<td>00</td>
<td>Filter1 Mask5</td>
</tr>
<tr>
<td>B6</td>
<td>00</td>
<td>Filter2 Mask1</td>
</tr>
<tr>
<td>B7</td>
<td>00</td>
<td>Filter2 Mask2</td>
</tr>
<tr>
<td>B8</td>
<td>00</td>
<td>Filter2 Mask3</td>
</tr>
<tr>
<td>B9</td>
<td>00</td>
<td>Filter2 Mask4</td>
</tr>
<tr>
<td>BA</td>
<td>00</td>
<td>Filter2 Mask5</td>
</tr>
<tr>
<td>BB</td>
<td>0D</td>
<td>NAND fcn, 5-byte header, Enable Filter 1</td>
</tr>
<tr>
<td>C0</td>
<td>01</td>
<td>VBI FIFO access via I2C</td>
</tr>
<tr>
<td>C1</td>
<td>00</td>
<td>Interrupt Line Number (not used)</td>
</tr>
<tr>
<td>D6</td>
<td>07</td>
<td>VDP line start</td>
</tr>
<tr>
<td>D7</td>
<td>16</td>
<td>VDP line stop</td>
</tr>
<tr>
<td>D8</td>
<td>F0</td>
<td>Global Line Mode, Teletext, F1, Filter ON</td>
</tr>
<tr>
<td>D9</td>
<td>00</td>
<td>Disable full field</td>
</tr>
<tr>
<td>DA</td>
<td>FF</td>
<td>Full field mode</td>
</tr>
</tbody>
</table>

**NOTE:** Setting to 5-byte header allows filtering on the “Type 2” byte.
1.2 Indirect Registers

Table 2 - Indirect I2C Register Settings - lists the register values to setup the indirect I2C registers of the TVP5146 for extraction of PDC and VPS data. This sets up a line mode register which specifies the line number, data type, field and options for the VPS data.

Table 2 - Indirect I2C Register Settings

<table>
<thead>
<tr>
<th>TVP5146 INDIRECT ADDRESS (HEX)</th>
<th>DATA (HEX)</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x800600</td>
<td>10</td>
<td>Line 16</td>
</tr>
<tr>
<td>0x800601</td>
<td>34</td>
<td>VPS, Sliced data to registers only, Field 1,</td>
</tr>
</tbody>
</table>

1.2.1 Writing to Indirect Registers

The following example, illustrates how to write to an indirect register. To load the previous data to the line address and line mode indirect registers:

1) Set the VBUS address to 0x800600.
   - Write 0x80 to I2C address 0xEA.
   - Write 0x06 to I2C address 0xE9.
   - Write 0x00 to I2C address 0xE8.

2) Write the data:
   - Write 0x10 to I2C address 0xE1. (Note register 0xE1 is the auto incrementing VBUS register.
     After writing to this register the internal VBUS address automatically increments to 0x800601).
   - Write 0x34 to I2C address 0xE1.
2 Program Delivery Control

2.1 PDC Input Data:

Since a PDC (Program Delivery Control) packet is sent on Teletext magazine 8, row number 30, type code 2, the data filter is setup to accept only teletext lines matching these three parameters.

The magazine 8 is encoded as 000. Row 30 in binary is 11110. After bit reversal, the magazine and row become 0000 1111. The type code is 0010. After Hamming encoding, the resulting teletext bytes are 15h, EAh and 49h. TVP5146 is normally set to filter on up to two header bytes. In this example, the data filter is set to allow filtering on up to 5 header bytes. This allows the third byte (Type Code) to be used for filtering also.

The input data is four lines of teletext, which have been arbitrarily placed on odd field lines 7-10. Only the teletext line on line 8 should pass thru the data filter.

The 40 bytes of teletext data are the same for each line. (The individual teletext lines in the output data can be distinguished by the line number, which is inserted by TVP5146.) The PDS data was sourced by a Norpak TES3.
Line 7: Magazine 8, Row 30, Type Code 1  => Discarded (wrong type code)

15 EA
02 15 15 EA EA EA 5E 15
49 64 A1 49 2F 5E FD B6
D0 D0 15 15 43 68 61 6E
6E E5 EC 20 34 20 54 E5
EC E5 76 E9 73 E9 EF 6E

Line 8: Magazine 8, Row 30, Type Code 2  => Accepted

15 EA
49 15 15 EA EA EA 5E 15
49 64 A1 49 2F 5E FD B6
D0 D0 15 15 43 68 61 6E
6E E5 EC 20 34 20 54 E5
EC E5 76 E9 73 E9 EF 6E

Line 9: Magazine 8, Row 29, Type Code 2  => Discarded (wrong row #)

02 2F
49 15 15 EA EA EA 5E 15
49 64 A1 49 2F 5E FD B6
D0 D0 15 15 43 68 61 6E
6E E5 EC 20 34 20 54 E5
EC E5 76 E9 73 E9 EF 6E

Line 10: Magazine 3, Row 30, Type Code 2  => Discarded (wrong magazine)

38 EA
49 15 15 EA EA EA 5E 15
49 64 A1 49 2F 5E FD B6
D0 D0 15 15 43 68 61 6E
6E E5 EC 20 34 20 54 E5
EC E5 76 E9 73 E9 EF 6E
2.2 PDC Output Data

Below is the output data which was read from the VBI FIFO. The total number of bytes received for each line of teletext was 52. The data shown was repeated many times. The following table describes the output data:

<table>
<thead>
<tr>
<th>DATA ITEM</th>
<th>VALUE(S) (HEX)</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>PREAMBLE</td>
<td>00, FF, FF</td>
<td>Preamble</td>
</tr>
<tr>
<td>DID</td>
<td>91</td>
<td>Sliced data of VBI lines of first field</td>
</tr>
<tr>
<td>SDID</td>
<td>78</td>
<td>From the Global Line Mode register: Teletext mode</td>
</tr>
<tr>
<td>NN</td>
<td>4B</td>
<td>0x0B = 11 x 4 = 44 bytes (counting from the 9th byte to the end)</td>
</tr>
<tr>
<td>IDID0</td>
<td>08</td>
<td>Line number 8</td>
</tr>
<tr>
<td>IDID1</td>
<td>0C</td>
<td>Status – Match#1, Match#2</td>
</tr>
<tr>
<td>SYNC</td>
<td>27</td>
<td>Sync Pattern</td>
</tr>
<tr>
<td>MAGAZINE &amp; ROW</td>
<td>15, EA</td>
<td>DeHams to 0000, 1111 =&gt; Magazine 8, Row 30</td>
</tr>
<tr>
<td>DATA</td>
<td>49, 15, 15.....6E</td>
<td>40 bytes. First byte is type code 2.</td>
</tr>
<tr>
<td>CHECKSUM</td>
<td>BA</td>
<td>Checksum</td>
</tr>
</tbody>
</table>

3 Video Program System

3.1 VPS Input Data:
VPS input data is from the Fluke 54200 TV Signal Generator, VPS data set #1.

3.2 VPS Output Data:

3.2.1 Via VPS Registers:

The data captured from TVP5146 in VPS mode using the VPS registers and status polling is shown in the following.

This data was repeated many times.

FF FF 42 FF FF FF FF FF 71 8E 7B 41 00

3.2.2 Via VBI FIFO

VPS byte number -> 3 4 5 6 7 8 9 10 11 12 13 14 15 CS PAD PAD

00 FF FF 91 BC 44 10 0C FF FF 42 FF FF FF FF FF 71 8E 7B 41 00 F6 00 00

<- TVP5146 Header ->

NOTE: For VPS, the run-in clock and sync bytes are counted as bytes 1 & 2.
### Table 4 - VPS Data via the VBI FIFO

<table>
<thead>
<tr>
<th>DATA ITEM</th>
<th>VALUE(S) (HEX)</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>PREAMBLE</td>
<td>00, FF, FF</td>
<td>Preamble</td>
</tr>
<tr>
<td>DID</td>
<td>91</td>
<td>Sliced data of VBI lines of first field</td>
</tr>
<tr>
<td>SDID</td>
<td>BC</td>
<td>from the Line Mode Register of line 16: VPS mode.</td>
</tr>
<tr>
<td>NN</td>
<td>44</td>
<td>NN = 4. 4 x 4 = 16 bytes (counting from the VPS byte #3 to the end). Two fill bytes are inserted at the end to make a multiple of 4 bytes.</td>
</tr>
<tr>
<td>IDID0</td>
<td>10</td>
<td>Line number 16 decimal</td>
</tr>
<tr>
<td>IDID1</td>
<td>0C</td>
<td>Status – Match#1, Match#2</td>
</tr>
<tr>
<td>BYTES 3-4 AND 6-10</td>
<td>FF,FF,FF,FF,FF</td>
<td>Not used by VPS</td>
</tr>
<tr>
<td>BYTES 5 AND 11-15</td>
<td>42,71,8E,7B,41,00</td>
<td>VPS Data (see analysis below)</td>
</tr>
<tr>
<td>CHECKSUM</td>
<td>F6</td>
<td>Checksum</td>
</tr>
<tr>
<td>FILL BYTES</td>
<td>00,00</td>
<td>Inserted to make a multiple of 4 bytes.</td>
</tr>
</tbody>
</table>
Figure 1 VPS Data Format from ETS 300 231
3.3 Analysis of VPS Data

From the document ETS 300 231 subtitled Television systems; Specification of the domestic video Programme Delivery Control system (PDC),

In that document, Figure 9, “Data format of the programme delivery data in the dedicated TV line”, Describes the format of the VPS data.

The analysis below shows how this data is decoded to generate the correct time stamp: Dec 24 14:30

<table>
<thead>
<tr>
<th>Byte</th>
<th>Value (Hex)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>0100 0010</td>
<td>PCS = 0100 = 4 CNI[1-4] = 0010 = 2</td>
</tr>
<tr>
<td>11</td>
<td>0111 0001</td>
<td>CNI[9-10] = 01 DAY = 11000 = 24</td>
</tr>
<tr>
<td>12</td>
<td>1000 1110</td>
<td>MONTH = 1100 = DEC HOUR = 01110 = 14:00</td>
</tr>
<tr>
<td>13</td>
<td>0111 1011</td>
<td>MINUTE = 01 1110 = 30</td>
</tr>
<tr>
<td>14</td>
<td>0100 0001</td>
<td>COUNTRY = 1101 NETWORK = 000001 CNI[5-8] = 0100 CNI[11-16] = 00 0001</td>
</tr>
<tr>
<td>15</td>
<td>0000 0000</td>
<td>PTY = 0000 0000</td>
</tr>
</tbody>
</table>

CNI = 0010 0100 0100 0001 = 2441h

Result: Dec 24 14:30

This is the correct result and matches the description given by the Fluke 54200 for this data set.
//////////////////////////////////////////////////////////////////////////////////////////////////////////
// PDC_VPS.H    TEXAS INSTRUMENTS    BY: MICHAEL A. TADYSHAK
// Function prototypes
////////////////////////////////////////////////////////////////////////////////////////////////////////////
void CmdStartVBIFIFO( BYTE *pBuf );
void InitializePDCVPS();
void TVP5146GeneralSetup();
void ResetLineModeRegisters();
void LoadLineModeRegisters();
void SetupDataFilter();
void SetupVDP();

void PollVBIFIFO( void );
void PollPDCVPS( int* pPDCAvail, int* pVPSAvail );
void HandleFIFO();
void HandleVFS();
void BlockReadVPSRegisters( unsigned char* p );
void PollFIFOThresh( int* pFIFOThresh );
void DoHouseKeeping();
void DoVPSHouseKeeping();
Retrieving the PDC and VPS data using status polling (source code)

#include "PDC_VPS.h"

#define NUM_MODE_REGISTERS   9
#define NUM_VPS_REGISTERS   13
#define NUM_PDC_BYTES       52

#define MASK_FIFO_THRESH  0x80
#define MASK_PDC_AVAIL    0x40
#define MASK_VPS_AVAIL    0x10

extern  EXTRACT_TTX_PARAMS g_stTTX;     // Structure containing teletext controls and buffer management variables
extern  BYTE g_pComBuf[];               // Serial port communication buffer
extern  BYTE g_pVBIBuffer[];    // 32KB buffer for PDC data
extern  BYTE g_pVPSBuffer[];    // 32KB buffer for VPS data
// Initialize for PDC and VPS data
InitializePDCVPS();
// Fill the buffer with TTX transfers
PollVBIFIFO();

// Return to polling for serial port commands
return;
void InitializePDCVPS()
{
    // Load general setup
    TVP5146GeneralSetup();

    // Reset line mode registers
    ResetLineModeRegisters();

    // Reset VBI status bits
    WriteTVP5146( 0xF6, 0x50 );

    // Reset the FIFO to empty
    WriteTVP5146( 0xBF, 0x00 );

    // Clear interrupt masks
    WriteTVP5146( 0xF4, 0x00 );
    WriteTVP5146( 0xF5, 0x00 );

    SetupDataFilter();
    SetupVDP();
    LoadLineModeRegisters();
}

void TVP5146GeneralSetup()
{
    WriteTVP5146( 0x00, 0x0C );    // Video Input = CVBS CH4
    WriteTVP5146( 0x02, 0x00 );    // Standard = Auto-switch
    WriteTVP5146( 0x06, 0x40 );    // Pedestal off
    WriteTVP5146( 0x34, 0x11 );    // Outputs active
    WriteTVP5146( 0x35, 0xFA );    // GLCO and FID active
    WriteTVP5146( 0x31, 0x05 );    // GLCO in ADI mode
}
void ResetLineModeRegisters()
{
    int i = 0;
    int nNumBytes = NUM_MODE_REGISTERS << 1;

    // Set the VBUS address to 0x800600.
    WriteTVP5146( 0xEA, 0x80 );
    WriteTVP5146( 0xE9, 0x06 );
    WriteTVP5146( 0xE8, 0x00 );

    // Set all line addresses to 0x00
    // Set all line mode registers to unused (0xFF)
    for( i=0; i<nNumBytes; i+=2 )
    {
        // Write to auto-incrementing data register
        WriteTVP5146( 0xE1, 0x00 );
        WriteTVP5146( 0xE1, 0xFF );
    }
}

void SetupDataFilter()
{
    // Data filter #1 - Mask and Pattern settings
    WriteTVP5146( 0xB1, 0xF0 ); // Magazine + Row LS Bit = 0000 (Magazine 8)
    WriteTVP5146( 0xB2, 0xFF ); // Row 4 MS Bits         = 1111 (Row 30)
    WriteTVP5146( 0xB3, 0xF2 ); // Type code 2
    WriteTVP5146( 0xB4, 0x00 );
    WriteTVP5146( 0xB5, 0x00 );

    // Data filter #2 - Mask and Pattern settings
    WriteTVP5146( 0xB6, 0x00 );
    WriteTVP5146( 0xB7, 0x00 );
    WriteTVP5146( 0xB8, 0x00 );
    WriteTVP5146( 0xB9, 0x00 );
    WriteTVP5146( 0xBA, 0x00 );

    // Filter control register
    // NAND function
    // NABTS 5-byte header
    // Enable filter #1
    // NOTE: Setting to 5-byte header allows filtering on the Type Code byte
    WriteTVP5146( 0xBB, 0x0D );
}
void SetupVDP()
{
    WriteTVP5146(0xD6, 0x07); // VDP line start at line 7
    WriteTVP5146(0xD7, 0x16); // VDP line stop at line 22
    WriteTVP5146(0xD8, 0xF0); // Global Line Mode:
                               // WST teletext (PDC), Field 1, FIFO, Filter on
    WriteTVP5146(0xC0, 0x01); // VBI FIFO access via I2C
    WriteTVP5146(0xD9, 0x00); // Disable full field
    WriteTVP5146(0xDA, 0xFF); // Full field mode
}

void LoadLineModeRegisters()
{
    // Set the VBUS address to 0x800600.
    WriteTVP5146(0xEA, 0x80);
    WriteTVP5146(0xE9, 0x06);
    WriteTVP5146(0xE8, 0x00);

    // VPS: Line 16, Field 1, Registers only, Filter disabled
    // Write to auto-incrementing data register
    WriteTVP5146(0xE1, 0x10);
    WriteTVP5146(0xE1, 0x34);
}
void PollVBIFIFO( void )
{
    int bPDC = FALSE;
    int bVPS = FALSE;
    int bThresh = FALSE;

    while( !g_stTTX.bBufferFull && !g_stTTX.bVPSBufferFull )
    {
        // Test if PC has requested to abort data extraction
        if( CTS == 0 )
        {
            // Set BUFFER_FULL_FLAG
            g_stTTX.bBufferFull = TRUE;
            g_stTTX.bVPSBufferFull = TRUE;
        }

        // Poll for PDC and VPS data
        PollPDCVPS( &bPDC, &bVPS );

        // Call this instead to read VPS data from the FIFO
        // PollFIFOThresh( &bThresh );
    }

    // Buffer is now full or stop was hit
    g_stTTX.bBufferFull = FALSE;
    g_stTTX.bVPSBufferFull = FALSE;

    // Reset buffer pointer to beginning
    g_stTTX.pBuffer          = &(g_pVBIBuffer[0]);
    g_stTTX.pVPSBuffer       = &(g_pVPSBuffer[0]);

    // Signal to PC that data extraction to buffer is complete
    RTS = 0;

    return;
}
void PollPDCVPS( int* pPDCAvail, int* pVPSAvail )
{
    int nStatus = ReadTVP5146( 0xF0 );

    int nPDC = nStatus & MASK_PDC_AVAIL;
    int nVPS = nStatus & MASK_VPS_AVAIL;

    *pPDCAvail = *pVPSAvail = FALSE;

    if( !nPDC && !nVPS )
    {
        return;
    }

    if( nPDC )
    {
        *pPDCAvail = TRUE;
        // Reset PDC available status
        WriteTVP5146( 0xF6, MASK_PDC_AVAIL );
        HandleFIFO();
    }

    if( nVPS )
    {
        *pVPSAvail = TRUE;
        // Reset VPS available status
        WriteTVP5146( 0xF6, MASK_VPS_AVAIL );
        HandleVPS();
    }

    return;
}
void PollFIFOThresh( int* pFIFOThresh )
{
    int nStatus = ReadTVP5146( 0xF0 );

    *pFIFOThresh = FALSE;

    if( nStatus & MASK_FIFO_THRESH )
    {
        *pFIFOThresh = TRUE;
        // Reset status
        WriteTVP5146( 0xF6, MASK_FIFO_THRESH );
        HandleFIFO();
    }

    return;
}
JNIEXPORT jint JNICALL Java_com_example_TvxDriver__c1TvxDriver_HandleFIFO(JNIEnv *env, jobject obj)
{
    jint i = 0;
    jint wByteCnt = ReadTVP5146(0xBC) << 1;
    if(wByteCnt == 0)
    {
        return;
    }
    if((WORD)wByteCnt > wBytesAvail)
    {
        wByteCnt--; // Decrease number of bytes to read until it fits in buffer
        bBufferFull = TRUE;
    }
    if(wByteCnt == 0)
    {
        bBufferFull = TRUE;
    }
    else
    {
        BlockReadTVP5146(0xE2, wByteCnt, pBuffer);
        DoHouseKeeping();
    }
void HandleVPS()
{
    g_stTTX.wVPSByteCnt = NUM_VPS_REGISTERS;

    // If bytes to read from VPS registers is more than available space in buffer
    // Decrease number of bytes to read until it fits in buffer
    // Set buffer full flag
    while( (WORD)g_stTTX.wVPSByteCnt > g_stTTX.wVPSBytesAvail )
    {
        g_stTTX.wVPSByteCnt --;
        g_stTTX.bVPSBufferFull = TRUE;
    }

    if( g_stTTX.wVPSByteCnt == 0 )
    {
        g_stTTX.bVPSBufferFull = TRUE;
    }
    else
    {
        // Read from VPS registers
        BlockReadVPSRegisters( g_stTTX.pVPSBuffer );
        DoVPSHouseKeeping();
    }
}
void BlockReadVPSRegisters( unsigned char* pBuf )
{
    // Set the VBUS address to 0x800700.
    WriteTVP5146( 0xEA, 0x80 );
    WriteTVP5146( 0xE9, 0x07 );
    WriteTVP5146( 0xE8, 0x00 );

    // Read the data
    // Read auto-incrementing data register
    BlockReadTVP5146( 0xE1, g_stTTX.wVPSByteCnt, pBuf );
}

void DoHouseKeeping()
{
    WORD w = 0;

    // Increase count of bytes in buffer
    g_stTTX.wBytesInBuffer += g_stTTX.wByteCnt;
    // Advance buffer pointer
    g_stTTX.pBuffer        += g_stTTX.wByteCnt;
    // Decrease bytes available in buffer
    g_stTTX.wBytesAvail    -= g_stTTX.wByteCnt;
}

void DoVPSHouseKeeping()
{
    // Increase count of bytes in buffer
    g_stTTX.wVPSBytesInBuffer += g_stTTX.wVPSByteCnt;
    // Advance buffer pointer
    g_stTTX.pVPSBuffer       += g_stTTX.wVPSByteCnt;
    // Decrease bytes available in buffer
    g_stTTX.wVPSBytesAvail   -= g_stTTX.wVPSByteCnt;
}
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:

<table>
<thead>
<tr>
<th>Products</th>
<th>Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amplifiers</td>
<td>amplifier.ti.com</td>
</tr>
<tr>
<td>Data Converters</td>
<td>dataconverter.ti.com</td>
</tr>
<tr>
<td>DSP</td>
<td>dsp.ti.com</td>
</tr>
<tr>
<td>Interface</td>
<td>interface.ti.com</td>
</tr>
<tr>
<td>Logic</td>
<td>logic.ti.com</td>
</tr>
<tr>
<td>Power Mgmt</td>
<td>power.ti.com</td>
</tr>
<tr>
<td>Microcontrollers</td>
<td>microcontroller.ti.com</td>
</tr>
<tr>
<td>Audio</td>
<td><a href="http://www.ti.com/audio">www.ti.com/audio</a></td>
</tr>
<tr>
<td>Automotive</td>
<td><a href="http://www.ti.com/automotive">www.ti.com/automotive</a></td>
</tr>
<tr>
<td>Broadband</td>
<td><a href="http://www.ti.com/broadband">www.ti.com/broadband</a></td>
</tr>
<tr>
<td>Digital Control</td>
<td><a href="http://www.ti.com/digitalcontrol">www.ti.com/digitalcontrol</a></td>
</tr>
<tr>
<td>Military</td>
<td><a href="http://www.ti.com/military">www.ti.com/military</a></td>
</tr>
<tr>
<td>Optical Networking</td>
<td><a href="http://www.ti.com/opticalnetwork">www.ti.com/opticalnetwork</a></td>
</tr>
<tr>
<td>Security</td>
<td><a href="http://www.ti.com/security">www.ti.com/security</a></td>
</tr>
<tr>
<td>Telephony</td>
<td><a href="http://www.ti.com/telephony">www.ti.com/telephony</a></td>
</tr>
<tr>
<td>Video &amp; Imaging</td>
<td><a href="http://www.ti.com/video">www.ti.com/video</a></td>
</tr>
<tr>
<td>Wireless</td>
<td><a href="http://www.ti.com/wireless">www.ti.com/wireless</a></td>
</tr>
</tbody>
</table>

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

Copyright © 2003, Texas Instruments Incorporated