Application Report
SBAA163–March 2009

TSC2004 WinCE® 6.0 Driver

Data Acquisition Products

ABSTRACT

The TSC2004 Microsoft® Windows® CE (WinCE) 6.0 touch driver has been developed with an I²C™ control interface; the code has been tested on a Samsung® SC32442 application processor. This application report discusses the TSC2004 driver, including the hardware connection between the TSC2004 and the platform, the WinCE 6.0 driver code structure, and the installations. Project collateral discussed in this application report can be downloaded from the following URL: http://www.ti.com/lit/zip/SBAA163.

Contents

1 Introduction

The TSC2004 WinCE 6.0 driver was developed for helping TSC2004 users to quickly set up, run, and use the device and to shorten software driver development time. The TSC2004 touch driver was coded on the standard WinCE touch device driver PDD (platform-dependent device) layer; the PDD layer was further split to have an additional processor-dependent layer (PDL) to make the TSC2004 driver easy to port into different host processors. See TI application report SLAA187 for details on PDD and PDL. The driver was developed and tested using TSC2004EVM board (see SLAU215) and Samsung SMDK platform with an SC32442 application processor.

2 Connections

The TSC2004 device must be wired and connected to a host processor, where the device driver code is ported and executed. In developing the TSC2004 drivers for this application, the TI TSC2004EVM board and the Samsung platform with the SC32442A application processor were employed.

The host processor controls TSC2004 through the interface that consists of three digital signals:

- The two-wire I²C bus: SCL and SDA;
- The touch pen-down and/or data ready interrupt, $\overline{\text{PINTDAV}}$.

See Figure 1 for the connections between the TSC2004 device and the SMDK2442 applications processor.

On the TSC2004EVM board, a connection to J2 is made in order to wire the three digital signals SCL, SDA, and $\overline{\text{PINTDAV}}$. For details on J2 and other aspects of the TSC2004EVM, see the TSC2004EVM User Guide.

On the Samsung SMDK2442 platform, the original touch module connected on the SMDK2442 main board was removed and replaced with the connections as shown in Figure 1. See Reference 4 and other relevant Samsung documentation for additional information about the Samsung SMDK2442 platform.

In addition to the three digital signal pins, the TSC2004 touch panel input signals, X+, X–, Y+ and Y–, are connected to the corresponding pins on the Samsung SMDK2442 touch panel, as Figure 1 indicates.

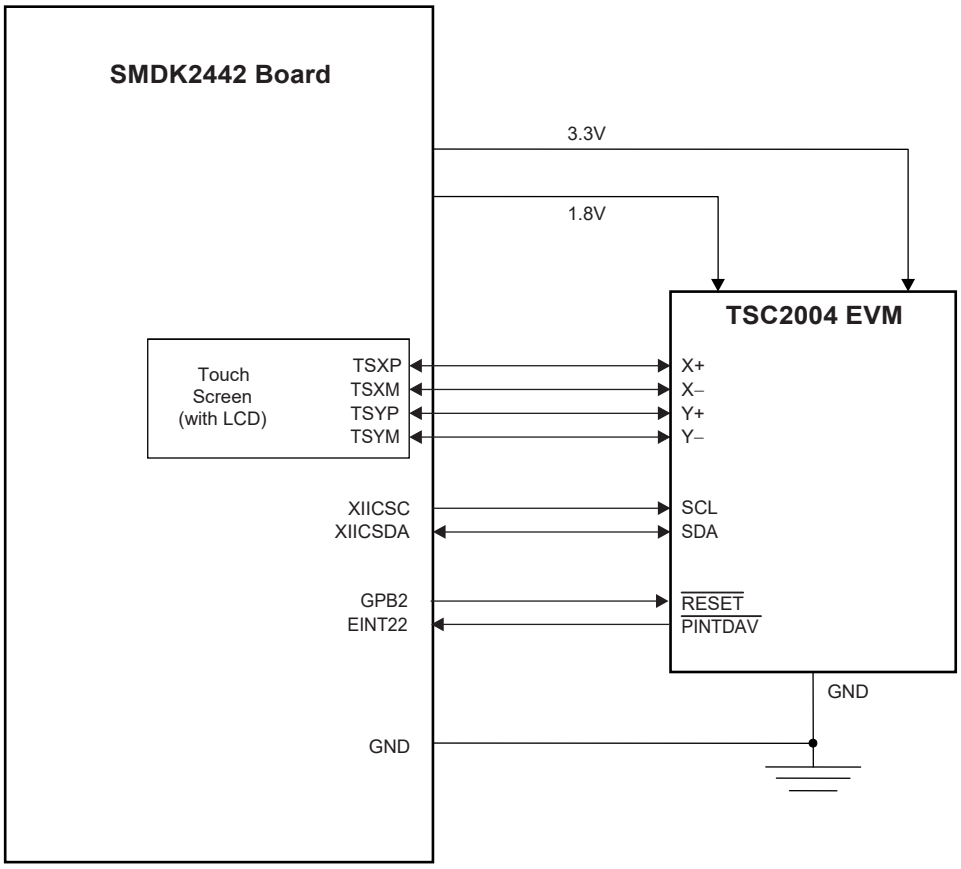

Figure 1. TSC2004 Connection to SC32442 Application Processor

3 Device Driver

Figure 2 details the TSC2004 touch device driver code file structure.

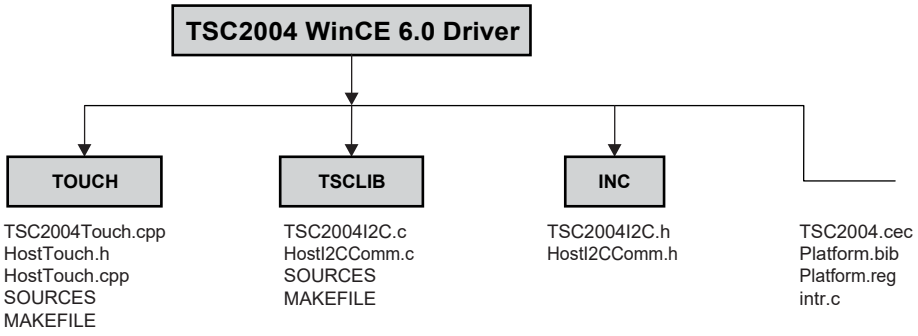

Figure 2. TSC2004 WinCE 6.0 Driver Files with I^2C Control Interface

3.1 I^2C interface

The I^2C bus is the control and data bus through which the host processor sends address and control commands to the TSC2004 and reads the touch screen or other data back from the device. The I^2C communication code was developed as a library and put into the directory, TSCLIB.

On the hardware side, the two TSC2004 I^2C bus pins (SCL and SDA) are connected to pins GPE14 and GPE15 of the Samsung SC32442 processor, respectively.

On the software side, the Samsung SC32442, I^2C, and clock management control registers are set up to communicate to the TSC2004 through the I^2C interface. The setup is implemented at the routine, *HWInitI2C( )*, which is inside the file *HostI2CComm.C*.

```
///////
// Function: void HWInitI2C (BOOL InPowerHandle)
// Purpose: This function must be called from the power handler
// of the respective drivers using this library. This
// function will configure the GPIO pins according to
// the functionality shown in the table below
// Signals Pin# Direction Alternate Function
// ---------------------------------------------------------------
// SCL GPE14 output 1
// SDA GPE15 output (at init) 1
///////
BOOL HWInitI2C(BOOL InPowerHandle)
{
UINT8 reg = 0x00;
RETAILMSG(TOUCH,(TEXT("Setup Host GPIO & I2C for an I2C Interface...\r\n")));
// init I2C control register (disabled I2C unit)
// enable I2C unit clock (the clock should be enabled first)
g_pClockRegs->CLKCON |= S3C_CLKEN_I2C;
// set up GPE
g_pGPIORegs->GPEDN |= GPE_DN;
//0xc000, Pull-up disable
//Making GPE15=>IICSDA, GPE14=>IICSCL
g_pGPIORegs->GPECON |= (GPE14_IIC_SCL | GPE15_IIC_SDA);
//Enable ACK, Prescalar IICCLK=PCLK/16, Enable interrupt, Transmit clock
//value Tx clock=IICCLK/16
//e.g. If PCLK 50.7MHz, IICCLK = 3.17 MHz, TX Clock = 0.198MHz
reg = ICR_ACK | ICR_INTR;
reg &= ~(ICR_TXCLK);
reg |= ICR_TXCLKVAL;
g_pI2CRegs->IICCON = reg;
g_pI2CRegs->IICADD = 0x10; //2442 slave address [7:1]
g_pI2CRegs->IICSTAT |= ISR_ENOP; //IIC bus data output enable(Rx/Tx)
//Filter enable and
g_pI2CRegs->IICLC = ILCR_FEN | ILCR_SDADLY; //15 clocks SDA output delay
DumpRegsI2C();
return(TRUE);
}
```

Two other important I²C interface routines are the *HWI2CWrite( )* and *HWI2CRead( )*. These routines allow the S3C2442 to control the TSC2004, performing touch data acquisition and reading the data back from the TSC2004. The complete I²C write and read transmissions are defined as shown in the TSC2004 product data sheet (Reference 1).

```
///////
// Function: HWI2CWrite Routine
// Purpose:    This routine allows the SMDK2442 to write to TSC2004
//         control register(s) using I2C bus.
// Note: The first byte in bytesBuf is the starting address
//    for writing; and the 2nd and on are bytes/contents
//    writing to TSC2004
///////
BOOL HWI2CWrite(UINT8 *bytesBuf, int bytesCount, BOOL InPowerHandle)
{
    int i;
    RETAILMSG(TOUCH, (TEXT("HWI2CWrite.... \r\n")));
    if (!InPowerHandle)
    {
        UINT8 reg,time = 100;
        iicMod    = WR_DATA;
        iicPtr    = 0;
for(i = 0; i < bytesCount; i++)    //Putting 1st byte i.e register addr iicDat[i] =
*bytesBuf++;     //2nd byte onwards actual data

        iicDCount    = bytesCount;

        g_pI2CRegs->IICDS = I2C_WRITE;    //I2C_WRITE;
//Putting TSC2004 slave address (7bit address + 0 'write bit')
        reg = g_pI2CRegs->IICSTAT;
            //Master transmit mode, START signal genration, Enable output
reg = (ISR_MTX | ISR_START | ISR_ENOP);

        g_pI2CRegs->IICSTAT = reg;

        while(iicDCount != -1)
            Run_Iic_Poll();

        iicMod = POLL_ACK;
        while(time--)
        {
            g_pI2CRegs->IICDS = I2C_WRITE;
            iicStat    = 0x100;

            reg = g_pI2CRegs->IICSTAT;
            reg = (ISR_MTX | ISR_START | ISR_ENOP);                                    //Master tx
mode, START signal genration, Enable output
            g_pI2CRegs->IICSTAT = reg;
            reg = g_pI2CRegs->IICCON;
            reg = ICR_ACK | ICR_INTR | ICR_TXCLKVAL;
            reg &= ~(ICR_PENITR);                                                 //Resumes IIC
operation.
            g_pI2CRegs->IICCON = reg;

            while(iicStat==0x100)
                Run_Iic_Poll();

            if(!(iicStat & 0x1))
                break;                                                    //When ACK is received
        }

        g_pI2CRegs->IICSTAT = ~(ISR_STOP);    //Stop MasTx condition
        reg = g_pI2CRegs->IICCON;                                          //Resumes IIC
operation.
        reg = ICR_ACK | ICR_INTR | ICR_TXCLKVAL;
        reg &= ~(ICR_PENITR);
        g_pI2CRegs->IICCON = reg;
        Delay(3);                                                 //Wait until stop condtion is
in effect.
            //Write is completed.
        return(TRUE);
    }
    else
```

```
    {
        RETAILMSG(TOUCH, (TEXT("HW Tx Error...\r\n")));
        return(FALSE);
    }
}



///////
// Function: HWI2CRead Routine
// Purpose:    This routine allows the SMDK2442 to read from TSC2004
//        control register(s) using I2C bus.
// Note: The first byte in bytesBuf is the starting address for
//    reading; and the 2nd and on are values reading from TSC2004
///////
UINT8 HWI2CRead(UINT8 *bytesBuf, INT bytesCount, BOOL InPowerHandle)
{
    RETAILMSG(TOUCH, (TEXT("HWI2CRead.... \r\n")));
    if (!InPowerHandle)
    {
        UINT8 reg;
        iicMod      = SETRD_ADDR;
        iicPtr        = 0;
        iicDat[0]   = *bytesBuf++;    //Putting 1st byte i.e. register address
        iicDCount    = 1;

//Putting slave address of TSC2004 for write mode [7:0]
        g_pI2CRegs->IICDS = I2C_WRITE;
        Delay(1);

        reg = g_pI2CRegs->IICSTAT;
            //Master transmit mode, START signal genration, Enable output
        reg = (ISR_MTX | ISR_START | ISR_ENOP);                          g_pI2CRegs->IICSTAT
= reg;

        while(iicDCount!=-1)
            Run_Iic_Poll();

        iicMod      = RD_DATA;
        iicPtr      = 0;
        iicDCount    = bytesCount;

//Putting slave address of TSC2004 for read mode[7:1]
        g_pI2CRegs->IICDS = I2C_READ;
        Delay(1);

        reg = g_pI2CRegs->IICSTAT;
        reg = (ISR_MRX | ISR_START | ISR_ENOP);
        g_pI2CRegs->IICSTAT = reg;        //Mater Rx, Start signal

        reg = g_pI2CRegs->IICCON;
        reg = ICR_ACK | ICR_INTR | ICR_TXCLKVAL ;
        reg &= ~(ICR_PENITR);
        g_pI2CRegs->IICCON = reg;        //Resumes IIC operation.

        while(iicDCount!=-1)
            Run_Iic_Poll();
        reg = g_pI2CRegs->IICCON;
        reg = ICR_ACK | ICR_INTR | ICR_TXCLKVAL;
        reg &= ~(ICR_PENITR);
        g_pI2CRegs->IICCON = reg;
        *bytesBuf++ = (UINT8) iicDat[1];
        return(1);
    }
    else
    {
        RETAILMSG(TOUCH, (TEXT("HW Rx Error...\r\n")));
        return(0);
    }
}
```

3.2 Touch Device Driver

In the Samsung SMDK2442 system, the interrupt $\overline{\text{PINTDAV}}$ pin is connected to the external interrupt EINT22, where the PINTDAV is fed to the S3C2442 GPG14 (J2.2) pin. The touch device driver is in the directory TOUCH, developed on the PDD layer of the standard touch device driver structure.

The TSC2004 touch configuration registers are set up or initialized in the routine *InitTSC2004Touch()* and called by the touch PDD routine *DdsiTouchPanelEnable( )*, which is shown here.

```
///////
// Function: void InitTSC2004Touch (BOOL bInPowerHandler)
// Purpose: Initalize TSC2004 Touch Screen Registers for Normal
//        X/Y TouchScreen Operation by writing to CFR0, CFR1
//        and CFR2 registers over I2C.
/////////
void InitTSC2004Touch(BOOL bInPowerHandler)
{
    UINT8 pWords[] = {0, 0, 0, 0, 0};

        //Software Reset of TSC2004.
    pWords[0] = (0x87);
    if(!HWI2CWrite(pWords, 2, FALSE))
        RETAILMSG(1,(TEXT("Device I2C Write Failed Start X Y conv\r\n")));

        //CFR2 Reg-    PINTDAV interrupt configured as PENIRQ,
        //            Enable MAVE filter for XYZ data.
    pWords[0] = 0x72; pWords[1] = 0x80; pWords[2] = 0x1C;
    if(!HWI2CWrite(pWords, 4, FALSE))
        RETAILMSG(1,(TEXT("Device I2C Write Failed CFR2\r\n")));

        //CFR0 Reg-    Converter functions initiated and controlled by the Host,
        //            Normal Operation, 12 Bit Resolution,
        //            fADC = fOSC/2. (= 2MHz A/D converter clock rate),
        //            Panel voltage stabilization time control 1ms,
        //            Detection of pen touch in wait.
    pWords[0] = 0x62; pWords[1] = 0x2B; pWords[2] = 0x02;
    if(!HWI2CWrite(pWords, 4, FALSE))
        RETAILMSG(1,(TEXT("Device I2C Write Failed CFR0\r\n")));
}
```

In the TSC2004 touch driver, the TSC2004 $\overline{\text{PINTDAV}}$ signal is enabled to detect any touch on the screen; $\overline{\text{PINTDAV}}$ triggers the *DdsiTouchPanelGetPoint( )* routine on the PDD layer whenever the $\overline{\text{PINTDAV}}$ becomes active.

```
///////
// DDSI Implementation
//
// @func void | DdsiTouchPanelGetPoint |
//        Returns the most recently acquired point and its associated tip state
//        information.
//
// @parm PDDSI_TOUCHPANEL_TIPSTATE | pTipState |
//        Pointer to where the tip state information will be returned.
// @parm PLONG | pUnCalX |
//        Pointer to where the x coordinate will be returned.
// @parm PLONG | pUnCalY |
//        Pointer to where the y coordinate will be returned.
// @comm
//        Implmented in the PDD.
///////
void DdsiTouchPanelGetPoint(TOUCH_PANEL_SAMPLE_FLAGS *pTipStateFlags,
                        INT *pUncalX, INT *pUncalY)
{
    static INT PrevX=0;
    static INT PrevY=0;
    static bool fPenDown = TRUE;

        // EINTPEND Reg holds the raw interrupt status for the TOUCH interrupt.
        // making use of the EINTPEND reg here to know status of the TOUCH.
    if(g_pIORegs->EINTPEND & (1<<22))        //    TOUCH occured
    {
        g_pIORegs->EINTPEND    = (1<<22);
        fPenDown = TRUE;
    }
    else                          //    NO TOUCH
        fPenDown = FALSE;

    if (g_pINTregs->INTMSK & (1<<IRQ_TIMER3))
    {        // The TIMER irq ...
        if(fPenDown)
        {
            if (!GetCoordinate(&PrevX, &PrevY))
                *pTipStateFlags = TouchSampleIgnore;
            else
            {
                TransCoordinate(&PrevX, &PrevY);

*pTipStateFlags = TouchSampleValidFlag | TouchSampleDownFlag;
                *pTipStateFlags &= ~TouchSampleIgnore;

                *pUncalX = PrevX;
                *pUncalY = PrevY;
                TouchTimerStart();
            }
            InterruptDone(gIntrTouchChanged);
                // The next expected interrupt will come from timer
RETAILMSG(TOUCH, (TEXT("$$$$$ TIMER: X=%d\tY=%d\r\n"),*pUncalX,*pUncalY));
        }
        else
        {        // Pen isn't currently down, sending MDD a pen-up "event".
            *pTipStateFlags = TouchSampleValidFlag;
                // Making use of the Prev values of X,Y when PEN is DOWN.
            *pUncalX = PrevX;
            *pUncalY = PrevY;

            TouchTimerStop();
            g_pIORegs->EINTPEND    = (1<<22);
            g_pIORegs->EINTMASK &= ~(1<<22);
            InterruptDone(gIntrTouch);
            InterruptDone(gIntrTouchChanged);
```

```
                // Next expected interrupt will come from pen-down event.
        RETAILMSG(TOUCH, (TEXT("***** UP: X=%d\tY=%d\r\n"),*pUncalX,*pUncalY));
        }
    }
    else            //gIntrTouch Case.
    {         //PEN transition from UP to DOWN
        if (!GetCoordinate(&PrevX, &PrevY))
            *pTipStateFlags = TouchSampleIgnore;
        else
            TransCoordinate(&PrevX, &PrevY);

        *pTipStateFlags = TouchSampleValidFlag;
        *pTipStateFlags |= TouchSampleIgnore;

        *pUncalX = PrevX;
        *pUncalY = PrevY;

        *pTipStateFlags |= TouchSampleDownFlag;

        if (g_pINTregs->INTMSK & (1<<IRQ_TIMER3))
            InterruptDone(gIntrTouchChanged);
        TouchTimerStart();
        InterruptDone(gIntrTouch);

            // The next expected interrupt will come from        RETAILMSG(TOUCH, (TEXT("@@@@@
DOWN: X=%d\tY=%d\r\n"),*pUncalX,*pUncalY));
    }
}
```

4 Installation

This section presents the installation steps required to run the TSC2004 WinCE 6.0 drivers on the Samsung SMDK2442 platform. The SC32442 application processor BSP can be obtained from Samsung and must be installed. After the application processor BSP installation, it will be located on your PC in a standard location within the WinCE directory; for example, at C:\WinCE500\PLATFORM\ as *SMDK2442*.

To install the TSC2004 WinCE 6.0 driver into one of the SMDK2442 workspace, execute the following steps.

4.1 Step 1: Copy

1. Copy all files inside \TSC2004WinCE6Driver\INC\ into: C:\WINCE600\PLATFORM\SMDK2442\SRC\INC\
2. Copy \TSC2004WinCE6Driver\Intr\intr.c file into: C:\WINCE600\PLATFORM\SMDK2442\SRC\Common\Intr\
3. Copy the directories TSCLIB and TOUCH into: C:\WINCE600\PLATFORM\SMDK2442\SRC\DRIVERS\

4.2 Step 2: Open

This step, in the Microsoft Visual Studio® 2005 Platform Builder IDE for CE 6.0, opens a new or existing SMDK2442 workspace. This procedure is ignored here.

4.3 Step 3: Modify

This step modifies the building device drivers to include the TI TSC2004 drivers.

1. Open the *dirs* file in the directory: C:\WINCE600\PLATFORM\SMDK2442\SRC\DRIVERS\

2. Add on the TSCLIB just before the TOUCH.
   For example, the dirs file could be:

```
DIRS=\
ceddk\
keybd\
PowerButton\
pccard\
serial\
nleddrvr\
Battdrvr\
cs8900\
Display\
Backlight\
TSCLIB\
Touch
```

4.4 Step 4: Update

This step updates the hardware-specific files so that the operating system will use the TSC2004 device drivers.

1. Open the existing platform.reg file from the *Parameter files* Section under the SMDK2442 in the PLATFORM window of the workspace.
2. Edit the platform.reg file in order to delete the old touch.dll and to add the TSC2004 touch screen controller.

```
;*************************************************TI-TSC200x************
; @CESYSGEN IF CE_MODULES_POINTER
IF BSP_NOTOUCH !
[HKEY_LOCAL_MACHINE\HARDWARE\DEVICEMAP\TOUCH]
"DriverName"="touch.dll"
"MaxCalError"=dword:10
; portrait
"CalibrationData"="491,632 115,124 110,1136 848,1136 852,132 "
ENDIF BSP_NOTOUCH !
; @CESYSGEN ENDIF CE_MODULES_POINTER
;*********************************************************************
```

3. Save and close the updated platform.reg file.
4. Similarly, edit the platform.bib file located in the same path as Platform.reg by replacing the existing inclusion of touch-related dll files in the NK.bin image with the following entry in the platform.bib file:

```
;*************************************************TI-TSC200x**************
; @CESYSGEN IF CE_MODULES_POINTER
IF BSP_NOTOUCH !
touch.dll $(_FLATRELEASEDIR)\touch.dll NK SHK
ENDIF BSP_NOTOUCH !
; @CESYSGEN ENDIF CE_MODULES_POINTER
;*********************************************************************
```

5. Save and close the updated platform.bib file.

5 References

The following documents are available for download through the Texas Instruments web site (www.ti.com), except where noted.

- TSC2004: Nano-power touch screen controller with I^2C serial interface. Product data sheet SBAS408A.
- Chammings, Y. and Fang, W. (2003.) TSC2301 WinCE Generic Drivers. Application report SLAA187.
- TSC2004EVM and TSC2004EVM-PDK. User's guide SLAU215.
- Samsung SC32442A Processor Developer's Kit. User guide. Available at www.samsung.com.

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale (www.ti.com/legal/termsofsale.html) or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

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