Application Report
SBAA169–March 2009

TSC2007 WinCE® 5.0 Driver

Data Acquisition Products

ABSTRACT

The TSC2007 Microsoft® Windows® CE (WinCE) 5.0 touch driver was developed with an I^{2}C™ control interface; the code has been tested on a Samsung® SC32442 application processor. This application report discusses the TSC2007 driver, including the hardware connection between the TSC2007 and the platform, the WinCE 5.0 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/SBAA169.

Contents

1 Introduction

The TSC2007 WinCE 5.0 driver was developed to help users of the TSC2007 touch screen controller device from Texas Instruments to quickly set up, run, and use the device and to shorten software driver development time. The TSC2007 touch driver was coded on the standard WinCE touch device driver platform-dependent device (PDD) layer, and the PDD layer was further split to have an additional processor-dependent layer (PDL) to make the TSC2007 driver easy to port into different host processors. See TI application report SLAA187 for additional details on both PDD and PDL. The driver was developed and tested using a TSC2007EVM board and the Samsung SMDK platform with an SC32442 application processor.

2 Connections

The TSC2007 device must be wired and connected to a host processor to which the device driver code is ported and executed. In developing the TSC2007 drivers for this application, the TI TSC2007EVM board and the Samsung platform with the SC32442A application processor were used.

The host processor controls the TSC2007 through an interface that consists of three digital signals:

- The two-wire I^{2}C bus: SCL and SDA;
- The touch pen-down interrupt, $\overline{\text{PENIRQ}}$.

See Figure 1 for the connections between the TSC2007 device and the SMDK2442 applications processor.

On the TSC2007EVM board, a connection to J2 was made in order to wire the three digital signals SCL, SDA, and PENIRQ. For details on J2 and other aspects of the TSC2007EVM, see the TSC2007EVM 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 TSC2007 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.

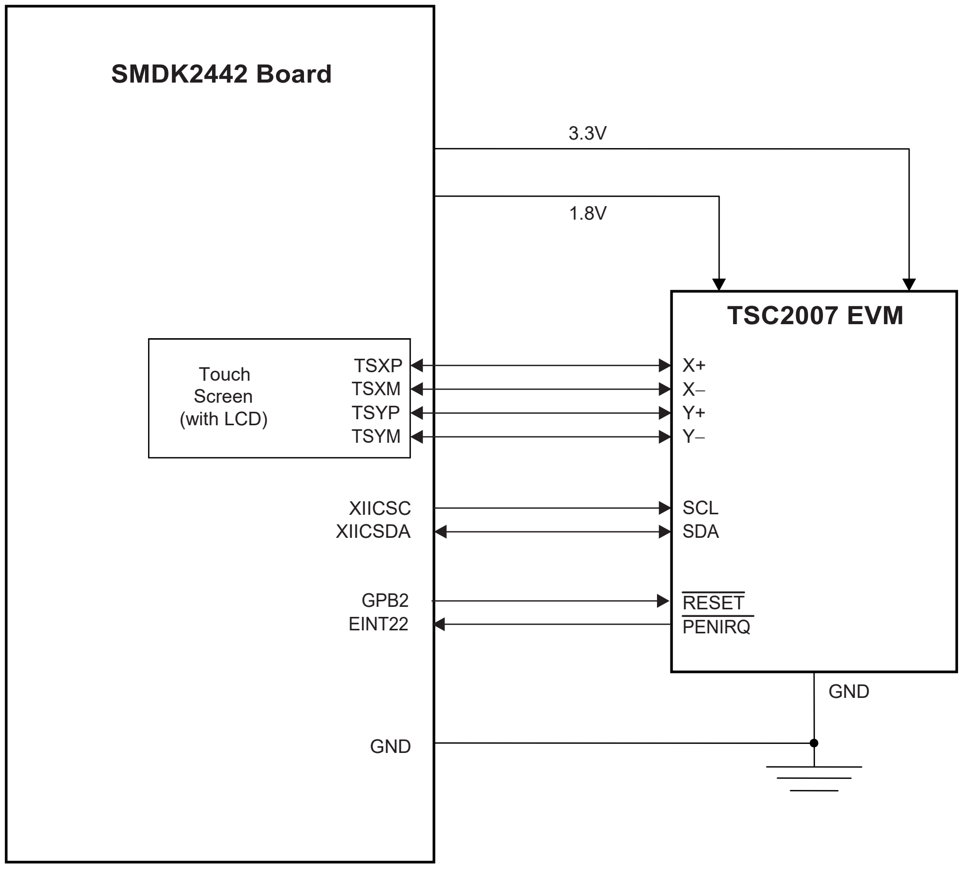

Figure 1. TSC2007 Connection to SC32442 Application Processor

3 Device Driver

Figure 2 details the TSC2007 touch device driver code file structure.

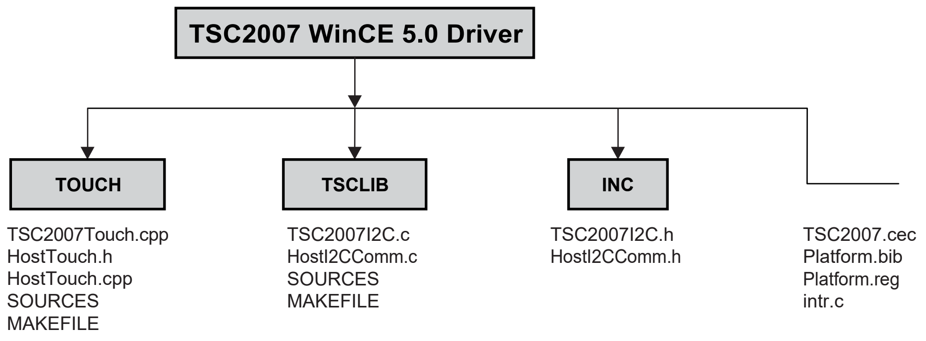

Figure 2. TSC2007 WinCE 5.0 Driver Files with I^{2}C Control Interface

3.1 I^{2}C interface

The I^{2}C bus is the control and data bus through which the host processor sends address and control commands to the TSC2007 and reads the touch screen or other data back from the device. The I^{2}C communication code was developed as a library and put into the directory, TSCLIB.

On the hardware side, the two TSC2007 I^{2}C 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^{2}C, and clock management control registers are set up to communicate to the TSC2007 through the I^{2}C 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 I^{2}C interface routines are the *HWI2CWrite( )* and *HWI2CRead( )*. These routines allow the S3C2442 to control the TSC2007, performing touch data acquisition and reading the data back from the TSC2007. The complete I^{2}C write and read transmissions are defined as shown in the TSC2007 product data sheet (Reference 1).

```
///////
// Function: HWI2CWrite Routine
// Purpose:    This routine allows the SMDK2442 to write to TSC2007
//          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 TSC2007
///////
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 TSC2007 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 generation,
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 condition 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 TSC2007
//          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 TSC2007
///////
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 TSC2007 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 TSC2007 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 $\overline{\text{PENIRQ}}$ pin has been connected to the external interrupt EINT22, where the PENIRQ 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.

In the TSC2007 touch driver, $\overline{\text{PENIRQ}}$ is enabled to detect any touch on the screen; $\overline{\text{PENIRQ}}$ then triggers the *DdsiTouchPanelGetPoint( )* routine on the PDD layer whenever PENIRQ 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;
    UINT8 pWords[] = {0,0,0,0,0};

        // EINTPEND Reg holds the raw interrupt status for the TOUCH interrupt.
        // making use of the EINTPEND reg here to know status of the TOUCH.
    g_pIORegs->EINTPEND    = (1<<22);
    if(g_pIORegs->EINTPEND & (1<<22))        //    TOUCH occured
        fPenDown = TRUE;
    else                        //    NO TOUCH
        fPenDown = FALSE;
    if (g_pINTregs->INTMSK & (1<<IRQ_TIMER3))    //gIntrTouchChaged Case.
    {    if(fPenDown)     // The TIMER irq ...
        {
            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
        {        // The pen isn't currently down, sending MDD a pen-up "event".
            *pTipStateFlags = TouchSampleValidFlag;
            *pUncalX = PrevX;    // Making use of the Prev values of X,Y
*pUncalY = PrevY;

            TouchTimerStop();
            g_pIORegs->EINTPEND    = (1<<22);
            g_pIORegs->EINTMASK &= ~(1<<22);
            InterruptDone(gIntrTouch);
            InterruptDone(gIntrTouchChanged);
// The next expected interrupt will come from pen-down event.
        RETAILMSG(TOUCH, (TEXT("***** UP: X=%d\tY=%d\r\n"),*pUncalX,*pUncalY));
        }
    }
    else    //PENIRQ From TSC200x Mapped to EINT22    //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 timer.
        RETAILMSG(TOUCH, (TEXT("@@@@@ DOWN: X=%d\tY=%d\r\n"),*pUncalX,*pUncalY));
    }
    g_pIORegs->EINTPEND    = (1<<22);
    pWords[0] = 0xD0;
    if(!HWI2CWrite(pWords, 2, FALSE))
        RETAILMSG(1,(TEXT("Device I2C Write Failed\r\n")));
}
```

4 Installation

This section presents the installation steps required to run the TSC2007 WinCE5.0 drivers on the Samsung SMDK2442 platform. The SC32442 application processor BSP can be obtained from Samsung and should 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 TSC2007 WinCE 5.0 driver into the SMDK2442 workspace, execute the following steps.

4.1 Step 1: Copy

1. Copy the \TSC2007WinCE5Driver\TSC2007.cec file into: C:\WINCE500\PUBLIC\COMMON\OAK\CATALOG\CEC\
2. Copy all files within \TSC2007WinCE5Driver\INC\ into: C:\WINCE500\PLATFORM\SMDK2442\SRC\INC\
3. Copy the \TSC2007WinCE5Driver\Intr\intr.c file into: C:\WINCE500\PLATFORM\SMDK2442\SRC\Common\Intr\
4. Copy the TSCLIB and TOUCH directories into: C:\WINCE500\PLATFORM\SMDK2442\SRC\DRIVERS\

4.2 Step 2: Set Up

This step sets up the catalog to include the TSC2007 device drivers.

1. Run Platform Builder 5.0; the Platform Builder integrated development environment (IDE) appears.
2. At the Platform Builder 5.0 IDE, open *Manage Catalog Items* from the menu File\Manage CatalogItems…\. When the Manage Catalog Items window appears, click on the *Import* button on the right side of the window; navigate, find, and select **TSC2007.cec** in the directory C:\WINCE500\PUBLIC\COMMON\OAK\CATALOG\CEC\. Then click *Open* so that the item is ported in.
3. Click and drag to select all *.cec files in the Manage Catalog Items Window, and then click on the *Refresh* button to make sure the new item is loaded.
4. Close the Manage Catalog Items window by clicking on the *OK* button.

4.3 Step 3: Open

This step, in the Platform Builder 5.0 IDE, opens a new or existing SMDK2442 workspace according to the application. This procedure is ignored here.

4.4 Step 4: Add

This step adds the TSC2007 device drivers from the Catalog into the existing OS design.

1. In the Catalog window of the Platform Builder 5.0 IDE, find the TI TSC2007 Touch Controller Driver. Right-click on the driver file, and select *Add to OS Design* to add the touch controller driver to the OS.
2. The touch controller driver should appear under the *Device Drivers* section at the OSDesignView window of the workspace.

4.5 Step 5: Modify

This step modifies the building device drivers to include TI TSC2007 drivers.

1. Open the *dirs* file in the directory: C:\WINCE500\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\
usb\
nleddrvr\
Battdrvr\
Backlight\
cs8900\
Display\
SDHC\
TSCLIB\
touch\
wavedev\
```

4.6 Step 6: Update

This step updates the hardware-specific files so that the operating system will use the TSC2007 device drivers.

1. Open the existing platform.reg file from the hardware-specific section of the ParameterView window of the workspace.
2. Edit the platform.reg file in order to delete the old touch.dll and to add the TSC2007 touch screen controller.

```
;*********************************************TI-TSC200x**************
; @CESYSGEN IF CE_MODULES_POINTER
IF BSP_NOTOUCH !
[HKEY_LOCAL_MACHINE\HARDWARE\DEVICEMAP\TOUCH]
"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; for example:

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

- TSC2007: Nano-power touch screen controller with I^{2}C serial interface. Product data sheet SBAS405A.
- Chammings, Y. and Fang, W. (2003.) TSC2301 WinCE Generic Drivers. Application report SLAA187.
- TSC2007EVM and TSC2007EVM-PDK. User's guide SLAU199.
- Samsung SC32442A Processor Developer’s Kit. User guide. Available at www.samsung.com.