Application Report
SBAA167–March 2009

TSC2006 WinCE® 6.0 Driver

Data Acquisition Products

ABSTRACT

The TSC2006 Microsoft® Windows® CE (WinCE) 6.0 touch driver has been developed with an SPI™ control interface; the code has been tested on a Samsung® SC32442 application processor. This application report discusses the TSC2006 driver, including the hardware connection between the TSC2006 and the platform, the WinCE 6.0 driver code structure, and the installation. Project collateral discussed in this application report can be downloaded from the following URL: http://www.ti.com/lit/zip/SBAA167.

Contents

1 Introduction

The TSC2006 WinCE 6.0 driver was developed for helping users of the TSC2006 touch screen controller device from Texas Instruments to quickly set up, run, and use the device and to shorten software driver development time. The TSC2006 touch driver was coded on the standard WinCE touch device driver platform-dependent device (PDD) layer; the PDD layer was further split to have an additional processor-dependent layer (PDL) to make the TSC2006 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 TSC2006EVM board and the Samsung SMDK platform with an SC32442 application processor.

2 Connections

The TSC2006 device must be wired and connected to a host processor to which the device driver code is ported and executed. In developing the TSC2006 drivers for this application, the TI TSC2006EVM board and the Samsung platform with the SC32442A application processor were used.

The host processor controls the TSC2006 through an interface that consists of five digital signals:

- The four-wire SPI bus: MISO, MOSI, $\overline{SS}$, and SCLK;
- The touch pen-down and/or data ready interrupt, $\overline{PINTDAV}$.

See Figure 1 for the connections between the TSC2006 device and the SMDK2442 applications processor.

On the TSC2006EVM board, a connector to J2 was made in order to wire the five digital signals MISO, MOSI, $\overline{SS}$, SCLK, and $\overline{PINTDAV}$. For details on J2 and other aspects of the TSC2006EVM, see the TSC2006EVM 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 five digital signal pins, the TSC2006 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.

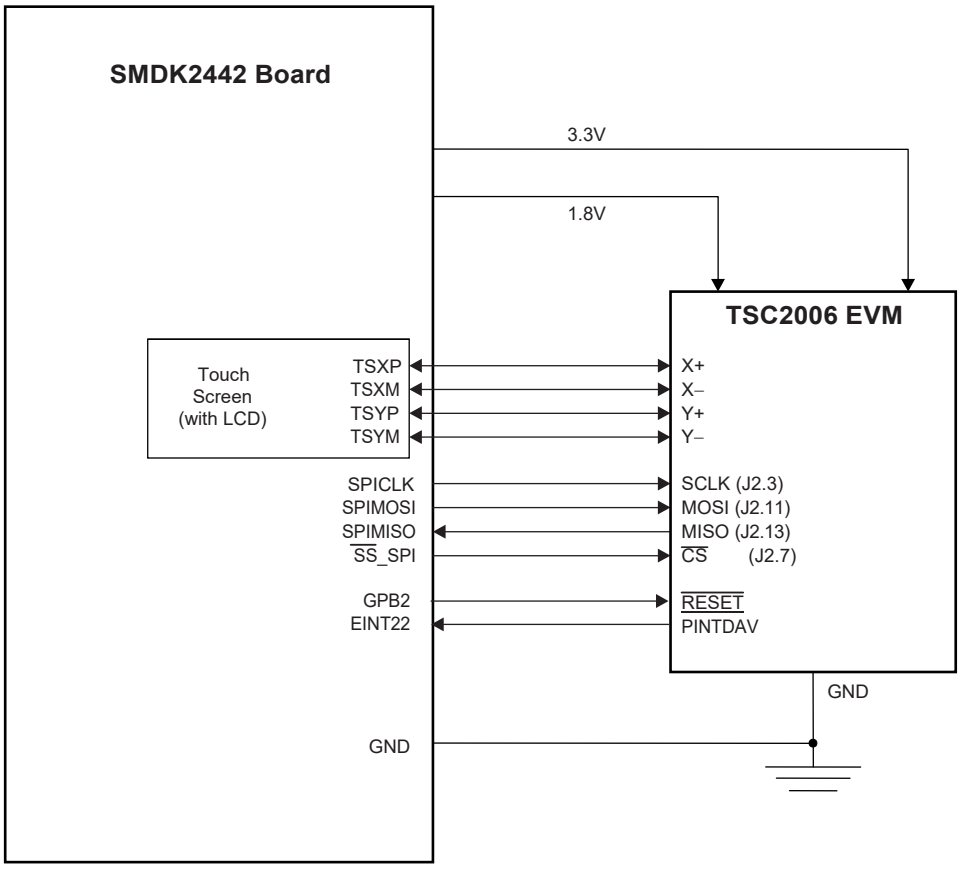

Figure 1. TSC2006 Connection to SC32442 Application Processor

3 Device Driver

Figure 2 shows the details of the TSC2006 touch device driver code file structure.

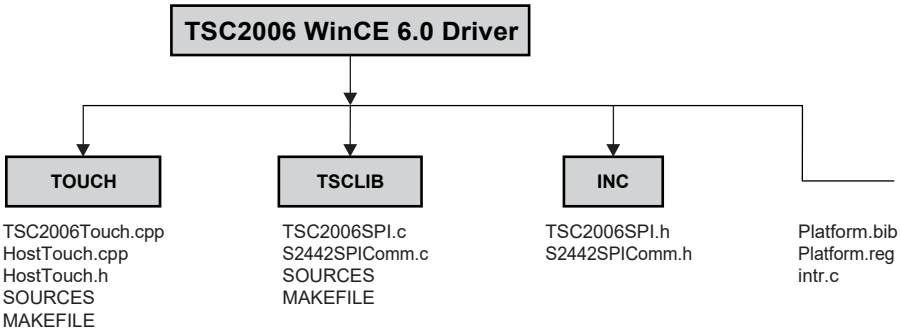

Figure 2. TSC2006 WinCE 6.0 Driver Files with SPI Control Interface

3.1 Serial Peripheral Interface

The SPI bus is the control and data bus through which the host processor sends address and control commands to the TSC2006 and reads the touch screen coordinates or other data back from the device. The SPI communication code was developed as a library and is available in the directory TSCLIB. The host configuration for the SPI is summarized in Table 1.

Table 1. Host Configuration for SPI

Signal/Input	GPIO Pin	SPI Function
1	GPE13	SPICLK
2	GPE12	SPIMOSI0
3	GPE11	SPIMISO0
4	GPG2	$\overline{SS}$_SPI

On the hardware side, there are four TSC2006 SPI bus pins. On the software side, the Samsung SC32442, SPI, and clock management control registers are set up to communicate with the TSC2006 through the SPI. This setup is implemented at the routine, *HWInitSPI( )*. This method also includes the configuration of the TSC2006 configuration registers and the initialization routine for the TSC2006. The routine *HWInitSPI( )* is available in the file *S2442SPIComm.C*.

```
///////
// Function: HWInitSPI
// Purpose: Setup and Initialize S3C2442 SPI Channel0 Module
///////
void HWInitSPI(BOOL InPowerHandle)
{
int i=0,Value=0;
// GPIO PIN Configuration //
// enable SPI unit clock (the clock should be enabled first)
//g_pClockRegs->CLKCON |= S3C_CLK_EN_SPI;
g_pClockRegs->CLKCON |= (0x1 << 18);
// set up GPE
g_pGPIORegs->GPEDN = (g_pGPIORegs->GPEDN & ~(7<<11))|(1<<13)|(0x1 << 12) | (0x1 << 11);
g_pGPIORegs->GPECON = ((g_pGPIORegs->GPECON&0xf03fffff)
| (0x2 << 26) //Configure GPE13=> SPICLK ,
| (0x2 << 24) //GPE12=>SPIMOSI0
| (0x2 << 22) ); //GPE11=>SPIMISO0 */
// set up GPG
g_pGPIORegs->GPGCON=((g_pGPIORegs->GPGCON&0xfffffffcf)|0x10); // Master(GPIO_Output)
g_pGPIORegs->GPGDAT &= ~(0x1<<2); // De-Activate nSS
// SPI Module Configuration //
//Configure Rate Prescaler Register (SPPRE0).
g_pSPIRegs->SPPRE0 = 0x00; //if PCLK=50Mhz,SPICLK=25Mhz
//Configure SPI CONTROL REGISTER (SPCON0)
g_pSPIRegs->SPCON0 = (0<<6)|(0<<5) //SMOD = 00 (SPI Mode Select=Polling)
|(1<<4) //ENSCK = 1 (Enable SPI Clock)
|(1<<3) //MSTR = 1 (Master Mode)
|(0<<2) //CPOL =0 (Clock Polarity Select )
|(1<<1) //CPHA=0 (Clock Phase Select )
|(0<<0); //TAGD=0 (normal)*??????*
//SPI PIN CONTROL REGISTER (SPPIN0)
g_pSPIRegs->SPPIN0 = (0<<2)
//ENMUL =0 (Disable Multi Master error detect )
|(0<<1) //CSn=0 (Deactivate CSn)
|(0<<0); //KEEP=0
HWStartFrame();
//Writing 0xff 10 times
for(i=0;i<10;i++)
{
while(!(g_pSPIRegs->SPSTA0 & 0x1)); //Check for Rx Ready
g_pSPIRegs->SPTDAT0=0xff;
Value=g_pSPIRegs->SPRDAT0;
HWSPIWait(1);
}
HWStopFrame();
return;
}
```

TSC device-specific initialization is present in the *TSC2006SPI.C* routine. Here is a snippet of the initialization code.

```
///////
// Function: BOOL InitSPI(BOOL bInPowerHandler)
// Purpose: Initialize Processor for SPI Interface.
/////////
BOOL InitSPI(BOOL bInPowerHandler)
{
int i=0,Value=0,V[3];
// Allocate SPI Control Resources.
if (!HWAllocateSPIResources( ))
return(FALSE);
// Setup SPI Interface at Host
HWInitSPI(bInPowerHandler);
//Software Reset
HWSPIConvertFunction(CON_FN_SW_RESET);
HWSPIConvertFunction(CON_FN_STOP);
if(TSC_Controlled_Mode == 1 )
{
HWSPIWrite(CTRL_BYTE_WRITE|CFR0_ADDR,
(CFR0_TSC_CONV | CFR0_STS_NRML |CFR0_RM_12BIT | CFR0_CLK_2MHz |
CFR0_PVS_5mS| CFR0_PRE_1044|CFR0_SNS_2080| CFR0_LSM_ON),
0); //Configure CFR0
HWSPIWrite(CTRL_BYTE_WRITE|CFR1_ADDR, (CFR1_BTD_2mS),0);//Configure CFR1
HWSPIWrite(CTRL_BYTE_WRITE|CFR2_ADDR,
(CFR2_DAV1 | CFR2_ZONE_DIS | CFR2_MAVE_DIS)
,0); //Configure CFR2
//PSM=1 start conversion function 0001 (read xy ).
HWSPIConvertFunction(CON_FN_12_BIT | XY_SCAN_FN);
}
else
{
//Configure CFR0 Register
HWSPIWrite(CTRL_BYTE_WRITE | CFR0_ADDR,
(CFR0_HOST_CONV| CFR0_STS_NRML |CFR0_RM_12BIT | CFR0_CLK_2MHz |
CFR0_PVS_1mS | CFR0_DTW_ON), 0);
//Configure CFR1 Register
HWSPIWrite(CTRL_BYTE_WRITE | CFR1_ADDR, CFR1_BTD_2mS, 0);
//Configure CFR2 Register
HWSPIWrite(CTRL_BYTE_WRITE | CFR2_ADDR, (CFR2_PENIRQ | CFR2_MAVE_XYZ), 0);
}
return(TRUE);
}
```

Two other important SPI interface routines are the *HWSPIWrite( )* and *HWSPIRead( )*. These routines allow the S3C2442 to control the TSC2006, performing touch data acquisition and reading the data back from the TSC2006. The complete SPI write and read transmission routines are defined in the TSC2006 product data sheet (Reference 1).

```
///////
// Function: HWSPIWrite Routine
// Purpose: This routine allows the SMDK2442 to write to TSC2006
// control register(s) using SPI bus.
///////
BOOL HWSPIWrite(UINT8 CtrlByte, UINT16 RegValue, BOOL InPowerHandle)
{
UINT8 TxByte, MSB, LSB, i;
UINT16 temp;
if (!InPowerHandle)
{
HWStartFrame();
while(!(g_pSPIRegs->SPSTA0 & 0x1)); //Send the Register Address
g_pSPIRegs->SPTDAT0=CtrlByte;
Delay(10);
temp = RegValue >> 8 ;
MSB = (UINT8) temp;
while(!(g_pSPIRegs->SPSTA0 & 0x1)); //Send the MSB
g_pSPIRegs->SPTDAT0=MSB;
Delay(10);
LSB = 0x00; //clear the variable
LSB = (UINT8)( RegValue & 0x00ff);
while(!(g_pSPIRegs->SPSTA0 & 0x1)); //Send the LSB
g_pSPIRegs->SPTDAT0=LSB;
HWStopFrame();
}
return (TRUE);
}

///////
// Function: HWSPIRead Routine
// Purpose: This routine allows the SMDK2442 to read from TSC2006
// control register(s) using SPI bus.
///////
UINT16 HWSPIRead(UINT8 CtrlByte,BOOL InPowerHandle)
{
UINT8 MSB, LSB, Dummy;
UINT16 finalValue;
if (!InPowerHandle)
{
HWStartFrame();
while(!(g_pSPIRegs->SPSTA0 & 0x1));
//Send the Control Byte to read the register
g_pSPIRegs->SPTDAT0=CtrlByte;
Delay(1);
while(!(g_pSPIRegs->SPSTA0 & 0x1));
//Read Dummy data
g_pSPIRegs->SPTDAT0=0x00;
Dummy=g_pSPIRegs->SPRDAT0;
Delay(1);
while(!(g_pSPIRegs->SPSTA0 & 0x1));
//Read LSB
g_pSPIRegs->SPTDAT0=0x00;
MSB=g_pSPIRegs->SPRDAT0;
Delay(1);
while(!(g_pSPIRegs->SPSTA0 & 0x1));
//Read LSB
g_pSPIRegs->SPTDAT0=0x00;
LSB=g_pSPIRegs->SPRDAT0;
Delay(1);
finalValue= (UINT16) MSB<<8;
finalValue |= (UINT16) LSB;
HWStopFrame();
return finalValue;
}
else
{
RETAILMSG(DebugMsg, (TEXT("HW Tx Error...\r\n")));
return(FALSE);
}
}
```

3.2 Touch Screen Driver

In the Samsung SMDK2442 system, the interrupt $\overline{\text{PINTDAV}}$ pin has been 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 screen device driver structure.

In the TSC2006 touch driver, the TSC2006 $\overline{\text{PINTDAV}}$ is enabled to detect any touch on the screen. $\overline{\text{PINTDAV}}$ triggers the *DdsiTouchPanelGetPoint( )* routine on the PDD layer whenever $\overline{\text{PINTDAV}}$ becomes active, as shown here:

```
///////
// 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
//Implemented 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;
if(g_pIORegs->EINTPEND & (1<<22))
{
g_pIORegs->EINTPEND = (1<<22);
fPenDown = TRUE;
}
else
fPenDown = FALSE;
if (g_pINTregs->INTMSK & (1<<IRQ_TIMER3))
{
if(fPenDown)
{
if (!GetCoordinate(&PrevX, &PrevY))
*pTipStateFlags = TouchSampleIgnore;
else
{
TransCoordinate(&PrevX, &PrevY);
*pTipStateFlags = TouchSampleValidFlag | TouchSampleDownFlag;
*pTipStateFlags &= ~TouchSampleIgnore;
*pUncalX = PrevX;
*pUncalY = PrevY;
TouchTimerStart();
}
InterruptDone(gIntrTouchChanged);
}
else
{
*pTipStateFlags = TouchSampleValidFlag;
*pUncalX = PrevX;
*pUncalY = PrevY;
TouchTimerStop();
g_pIORegs->EINTPEND = (1<<22);
g_pIORegs->EINTMASK &= ~(1<<22);
InterruptDone(gIntrTouch);
InterruptDone(gIntrTouchChanged);
}
}
else
{
```

```
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);
}
}
```

4 Installation

This section presents the installation steps required to run the TSC2006 WinCE 6.0 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:\WinCE600\PLATFORM\ as *SMDK2442*.

To install the TSC2006 WinCE 6.0 driver into one of the SMDK2442 workspace, execute the following steps.

4.1 Step 1: Copy

Copy the following files and directories:

1. Copy all files inside \TSC2006WinCE6Driver\INC\ into this directory: C:\WINCE600\PLATFORM\SMDK2442\SRC\INC\
2. Copy \TSC2006WinCE6Driver\Intr\intr.c file to this location: 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 TSC2006 drivers.

1. Open the **dirs** file in the directory: C:\WINCE600\PLATFORM\SMDK2442\SRC\DRIVERS\.
2. Add in 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 TSC2006 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 such as to delete the old Touch.dll file, and to add in the TSC2006 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 the platform.reg file, by replacing the existing entry of the touch related .dll files into 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.

- TSC2006: Nano-power touch screen controller with SPI interface. Product data sheet SBAS415.
- Chammings, Y. and Fang, W. (2003.) TSC2301 WinCE Generic Drivers. Application report SLAA187.
- TSC2006EVM and TSC2006EVM-PDK. User's guide SLAU200.
- 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