Application Report
SLAA277–November 2005

TSC2003 WinCE 5.0 Driver

Wendy X. Fang *DAP Group, HPA*

ABSTRACT

The TSC2003 Microsoft™ Windows™ CE 5.0 touch driver has been developed, and the code has been tested on an Intel™ MainStone II development platform. This application report discusses the TSC2003 driver, including the hardware connection between the TSC2003 and the platform, the Windows CE 5.0 driver's code and structure, and the installations. Project collateral discussed in this application report can be downloaded from the following URL: www.ti.com/lit/zip/SLAA277.

Contents

List of Figures

List of Tables

Trademarks

Intel is a trademark of Intel Corporation.
Microsoft, Windows are trademarks of Microsoft Corporation.

1 Introduction

The TSC2003 WinCE 5.0 driver was developed for helping TSC2003 users to quickly set up, run, and use the device and to shorten software driver development time. The TSC2003 touch driver was coded on the standard WinCE touch device driver's PDD (platform-dependent device) layer, and the PDD layer was further split to have an additional processor-dependent layer (PDL) to make the TSC2003 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 a TSC2003EVM board (see SBAU109) and Intel MainStone II platform with the PXA270 Step B0 processor (see reference 4).

2 Connections

The TSC2003 device must be wired and connected to a host processor where the device driver code is ported and executed. The host processor controls TSC2003 through the interface that consists of three digital signals:

- The I^2C bus, two wires: SCL, and SDA
- The touch Pen-Down interrupt, $\overline{\text{PENIRQ}}$.

For developing the TSC2003 driver, the TSC2003EVM and the Intel MainStone II development platform with the PXA270 Step B0 processor were used. Figure 1 illustrates the wires and connections between PXA27x and TSC2003.

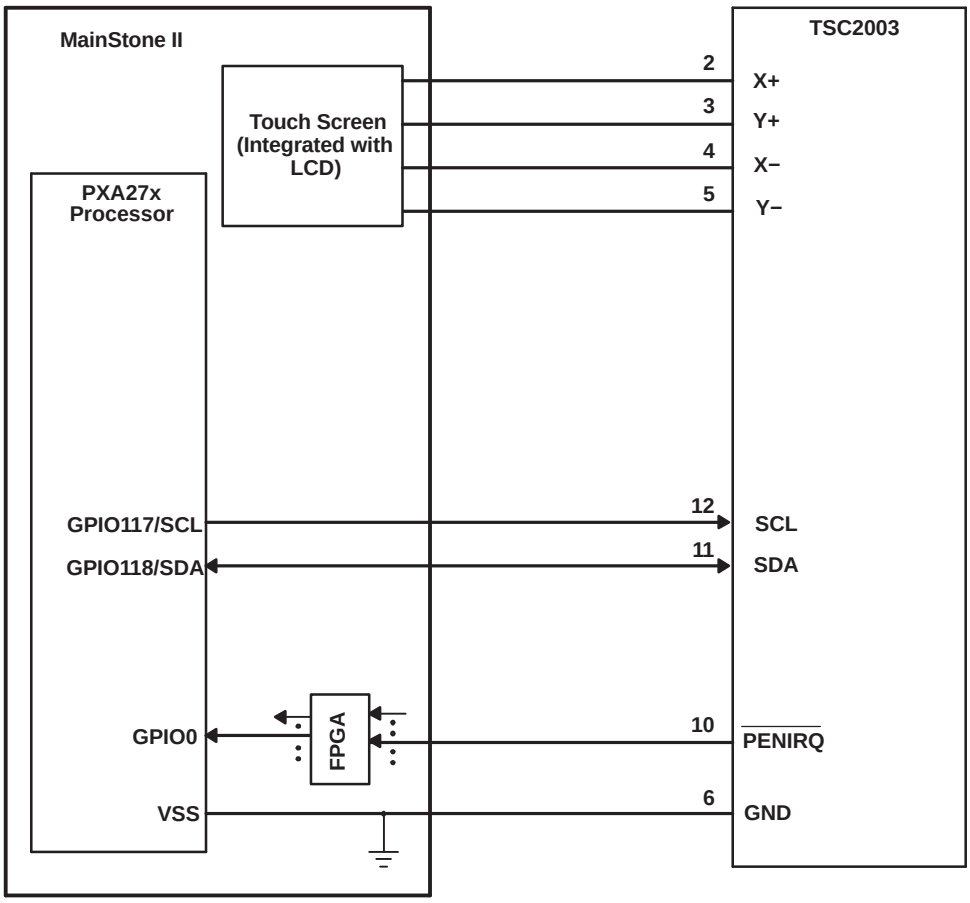

Figure 1. TSC2003 Connections to MainStone II System

On the TSC2003EVM board, a connector to J2 was made to wire the three digital signals SCL, SDA, and $\overline{\text{PENIRQ}}$. For details on the connector J2 and other aspects of the TSC2003EVM, see SBAU109.

On the MainStone II system, the original touch/audio module, connected on the MainStone II main board, was removed and replaced with the connections as shown in Figure 1. See reference 4 and other relevant Intel documentation for additional information about the MainStone II Platform.

In addition to the three digital signal pins, the TSC2003 touch panel input signals, X+, X–, Y+ and Y–, are connected to the corresponding pins on the MainStone II touch panel, as Figure 1 indicates.

3 Touch Screen Driver

Figure 2 lists the TSC2003 touch device driver's code files. The files starting with "Host…" are the processor-dependent code or PDL, such as HostTouch.CPP or HostI2CComm.H. The PDL code was developed only for the PXA27x processor.

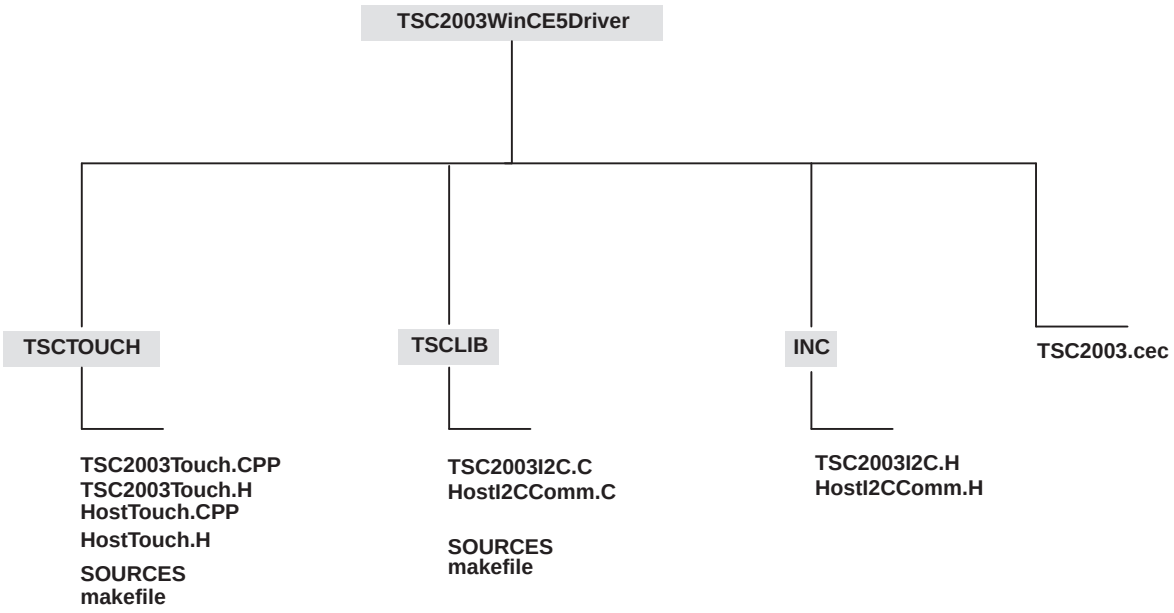

Figure 2. TSC2003 WinCE 5.0 Driver Files

3.1 I2C Interface

The I2C bus is the control and data bus, through which the host processor sends address and control commands to TSC2003 and reads the touch screen or other data back from TSC2003. The I2C communication code was developed as a library and put into the directory, **TSCLIB**.

On the hardware side, the two TSC2003 I2C bus pins, SCL and SDA, were connected to the GPIO117 and GPIO118 of the PXA27x processor, respectively.

On the software side, the PXA27x GPIO, I2C, and Clock management control registers were set up to communicate to the TSC2003 through the I2C. The setup was implemented at the routine, *HWInitI2C()*, which is inside 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 GPIO117 output 1 // SDA GPIO118] output(at init) 1
///////
BOOL HWInitI2C(BOOL InPowerHandle) { RETAILMSG(1,(TEXT("Setup Host GPIO & I2C for an I2C
Interface...\r\n")));
 // init I2C control register (disabled I2C unit) g_pI2CRegs->icr = 0;
 // enable I2C unit clock (the clock should be enabled first) g_pClockRegs-
>cken |= XLLP_CLKEN_I2C;
 // set up GPIO g_pGPIORegs->GPDR3 |= GPIO_117_SCL; g_pGPIORegs-
>GPDR3 |= GPIO_118_SDA; g_pGPIORegs->GAFR3_U &= ~GPIO_I2C_MASK; g_pGPIORegs-
>GAFR3_U |= GPIO_117_AF1_SCL; g_pGPIORegs->GAFR3_U |= GPIO_118_AF1_SDA;
 // Setup processor I2C slave address (used only when PXA27x is slave) g_pI2CRegs->isar = 0x007F;
 // Set Processor I2C as master and enable the I2C g_pI2CRegs->icr = ICR_SCLEA; g_pI2CRegs-
>icr |= ICR_IUE;
 DumpRegsGPIO(); DumpRegsI2C(); DumpRegsClock();
 return(TRUE); }
```

Two other important I2C interface routines are the *HWI2CWrite()* and *HWI2CRead()*, which allow the PXA27x to command the TSC2003, performing the touch data acquisition and to read the data back from TSC2003. The complete I2C write and read transmissions have been defined as shown by Figure 12 and Figure 13 of the TSC2003 data sheet (SBAS162).

```
/////// // Function: HWI2CWrite Routine // Purpose: This routine allows PXA27x to write command
to TSC2003 // using I2C bus. /////// BOOL HWI2CWrite(UINT8 Command, BOOL InPowerHandle) {
 UINT32 reg; if (!InPowerHandle) {
 // write (TSC2003) device address + write "0" to I2C bus g_pI2CRegs-
>idbr = I2C_WRITE; reg = g_pI2CRegs-
>icr; reg |= (ICR_START | ICR_TB); reg &= ~ICR_STOP; g_pI2CRegs-
>icr = reg; if (HWI2CTxBusy(2000)) return(FALSE);
 // write (TSC2003) command through I2C bus g_pI2CRegs->idbr = (UINT32) Command; reg = g_pI2CRegs-
>icr; reg &= ~ICR_START ; reg |= (ICR_TB | ICR_STOP); g_pI2CRegs-
>icr = reg; if (HWI2CTxBusy(2000)) return(FALSE);
 // Clear the STOP bit always g_pI2CRegs-
>icr &= ~ICR_STOP; return (TRUE); } else { RETAILMSG(1, (TEXT("HW Writing Error...\r\n")));
return(FALSE); } }
/////// // Function: HWI2CRead Routine // Purpose: This routine allows the PXA27x to read data
from TSC2003 // using I2C bus. /////// BOOL HWI2CRead(UINT8 *bytesBuf, UINT8 bitFlag, BOOL
InPowerHandle) {
 UINT32 reg; if (!InPowerHandle) { // start and device address + read "1" to I2C bus g_pI2CRegs-
>idbr = I2C_READ; reg = g_pI2CRegs-
>icr; reg |= (ICR_START | ICR_TB); reg &= ~ICR_STOP; g_pI2CRegs-
>icr = reg; if (HWI2CTxBusy(2000)) return(FALSE);
 // read the TSC2003 registers' contents reg = g_pI2CRegs-
>icr; reg &= ~ICR_START; reg |= ICR_TB; if (Flag12bit) { reg &= ~(ICR_ACKNAK | ICR_STOP); } else
{ reg |= (ICR_ACKNAK | ICR_STOP); } g_pI2CRegs-
>icr = reg; if (HWI2CRxBusy(6000)) return(FALSE); reg = (g_pI2CRegs-
>idbr) & 0xFF; bytesBuf[0] = (UINT8) reg;
 if (Flag12bit) { reg = g_pI2CRegs-
>icr; reg &= ~ICR_START; reg |= (ICR_TB | ICR_ACKNAK | ICR_STOP); g_pI2CRegs-
>icr = reg; if (HWI2CRxBusy(6000)) return(FALSE); reg = (g_pI2CRegs-
>idbr) & 0xFF; bytesBuf[1] = (UINT8) reg; }
 // Clear the STOP and ACKNAK bits always g_pI2CRegs-
>icr &= ~(ICR_ACKNAK | ICR_STOP); return(TRUE);
 } else { RETAILMSG(1, (TEXT("HW Reading Error...\r\n"))); return(FALSE); } }
```

3.2 Touch Screen Driver

In the MainStone II system, the interrupt $\overline{\text{PENIRQ}}$ pin has been connected to an FPGA, where the $\overline{\text{PENIRQ}}$ was ORed with other secondary interrupts and fed to the PXA27x GPIO0 pin (see reference 4). In this application, the TSC2003's $\overline{\text{PENIRQ}}$ pin was wired to the MainStone II's $\overline{\text{PENIRQ}}$ pin, which actually goes to the PXA27x GPIO0 pin (see Figure 1).

The touch device driver is in the directory *TSCTOUCH*, developed on the PDD layer of the standard touch device driver structure.

In the TSC2003 touch driver, the TSC2003 $\overline{\text{PENIRQ}}$ is enabled to detect any touch on the screen and the $\overline{\text{PENIRQ}}$ triggers the *DdsiTouchPanelGetPoint ()* on the PDD layer whenever the $\overline{\text{PENIRQ}}$ becomes active:

```
VOID DdsiTouchPanelGetPoint(TOUCH_PANEL_SAMPLE_FLAGS *pTipStateFlags, INT *pUncalX,INT *pUncalY)
{ static BOOL TouchIrq = TRUE; UINT32 InterruptType = SYSINTR_NOP; // RETAILMSG(1,(TEXT("Calling
DdsiTouchPanelGetPoint()\r\n"))); *pTipStateFlags = TouchSampleIgnore; if (TouchIrq) { // The pen
was previously up -
 it just transitioned to down state. TouchIrq = FALSE; InterruptType = SYSINTR_TOUCH;
*pTipStateFlags = PDDSampleTouchScreen(pUncalX, pUncalY); // The next expected interrupt will
come from sampling timer // (pen-
up doesn't cause an interrupt). g_NextExpectedInterrupt = PEN_UP_OR_TIMER; //
RETAILMSG(1,(TEXT("GetPoint: pen from up to down ...\r\n"))); } else { // The timer irq ... //
The pen could now be either up or down at this point // -
 we need to check. InterruptType = SYSINTR_TOUCH_CHANGED; // Read data if /PENIRQ is active so as
to read // the last data if available *pTipStateFlags = PDDSampleTouchScreen(pUncalX, pUncalY);
// The next expected interrupt will come from sampling time // (pen-
up doesn't cause an interrupt). g_NextExpectedInterrupt = PEN_UP_OR_TIMER;
//RETAILMSG(1,(TEXT("GetPoint: timer irq occurred ...\r\n"))); } if ((g_NextExpectedInterrupt ==
PEN_UP_OR_TIMER) && !HWGetTouchStatus()) { TouchIrq = TRUE; // the pen isn't currently down, send
```

```
the MDD a pen-
up "event". *pTipStateFlags = TouchSampleValidFlag; // but send pen up to mdd // The next
expected interrupt will come from pen-
down event. g_NextExpectedInterrupt = PEN_DOWN; // RETAILMSG(1,(TEXT("GetPoint: pen is up
...\r\n"))); } // Make sure the next expected interrupt is configured and enabled.
PrepareNextInterrupt(g_NextExpectedInterrupt); // Tell the OAL to clear and unmask interrupt just
occurred. InterruptDone(InterruptType); // RETAILMSG(1,(TEXT("END Calling
DdsiTouchPanelGetPoint()\r\n"))); }
```

In the *DdsiTouchPanelGetPoint ()* routine, the *PDDSampleTouchScreen()* is called to perform a *TSC2003Read()*:

```
// //-------------------------------------------------------------
 // General Function for TSC2003 Control Register R/W // where "source" is the configuration bits
C3- C0 ; and // return R-Adjusted 12 or 8 bit data in [pWords[0] pWord[1]] //--------------------
---------------------------------------- // // Read an 12 or 8-
bits TSC2003 Data UINT16 TSC2003Read(UINT8 source, BOOL bInPowerHandler) { UINT8 pCommand,
pWords[] = {0, 0}; if (Flag12bit) { // command byte= [C3 C2 C1 C0 0 1 0 0] pCommand = ( (source
<< 4) | 0x04 ); HWI2CWrite(pCommand, bInPowerHandler); HWI2CRead(pWords, 1, bInPowerHandler); //
adjusted data MSB=bit#11 & LSB=bit#0 // and return 10 MSBs return ( (UINT16) (pWords[0] << 2) |
(UINT16) (pWords[1] >> 6) ); } else { // command byte= [C3 C2 C1 C0 0 1 1 0] pCommand = ( (source
<< 4) | 0x06 ); HWI2CWrite(pCommand, bInPowerHandler); HWI2CRead(pWords, 0, bInPowerHandler); //
return the right adjusted data MSB=bit#7 & LSB=bit#0 return ( (UINT16) (pWords[0]<<2) ); } }
```

In the *TSC2003Read()* performs a complete Write (see Figure 12 of SBAS162) and followed by a complete Read (see Figure 13 of SBAS162) for reading a touch data (X or Y) from TSC2003.

4 Installation

This section presents the installation steps to run the TSC2003 WinCE5.0 drivers on the Intel MainStone II Platform.

Included with the Microsoft Windows CE 5.0 platform builder CD ROM is the *Board Support Package* (BSP) of the MainStone II, called \MAINSTONEII\, which may be located in your PC after installing the Platform Builder 5.0 at, for example:

C:\WinCE500\PLATFORM\.

To install the TSC2003 Windows CE 5.0 driver into one of the MainStone II WorkSpaces, execute the following steps.

4.1 Step I: Copy

1. Copy \TSC2003WinCE5Driver\TSC2003.cec file into:
   C:\WINCE500\PUBLIC\COMMON\OAK\CATALOG\CEC\
2. Copy all files inside \TSC2003WinCE5Driver\INC\ into:
   C:\WINCE500\PLATFORM\MAINSTONEII\SRC\INC\
3. Copy the directories TSCLIB and TSCTouch into:
   C:\WINCE500\PLATFORM\MAINSTONEII\SRC\DRIVERS\

4.2 Step II: Set Up

This step sets up the catalog to include the TSC2003 device driver.

1. Run **Platform Builder 5.0**, and the Platform Builder IDE appears.
2. At the Platform Builder 5.0 IDE, open **Manage Catalog Items** from the menu **File\Manage Catalog Items …\.** When the **Manage Catalog Items** window appears, click on **Import** button on the right side of the window; navigate, find, and select **TSC2003.cec** in the directory:
   C:\WINCE500\PUBLIC\COMMON\OAK\CATALOG\CEC\
   Then click on **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 ensure that the new item is loaded.
4. Close the **Manage Catalog Items** window by clicking on its **OK** button.

4.3 Step III: Open

This step, in the Platform Builder 5.0 IDE, opens a new or existing MainStone II workspace per the application. The procedure is ignored here.

4.4 Step IV: Add

This step adds the TSC2003 device driver from the **Catalog** into the existing **OS design**.

1. In the **Catalog** window of the **Platform Builder 5.0 IDE**, find **TI TSC2003 Touch Controller Driver**, right-click on it, and select **Add to OS Design** to add the touch controller driver to the OS.
2. As a result, the touch device driver appears under the **Device Drivers** section at the **OSDesignView** window of the WorkSpace.

4.5 Step V: Modify

This step modifies the building device drivers so as to include TI TSC2003 touch driver.

1. Open the **dirs** file in the directory:

   C:\WINCE500\PLATFORM\MAINSTONEII\SRC\DRIVERS\
2. Eliminate the original touch directory from the list, and add on the **TSCLIB** and **TSCTOUCH**. For example: the **dirs** file could be

```
DIRS=\ # @CESYSGEN IF CE_MODULES_POINTER TSCLIB \ TSCTOUCH \ # @CESYSGEN ENDIF
CE_MODULES_POINTER # @CESYSGEN IF CE_MODULES_DEVICE # @CESYSGEN IF CE_MODULES_USBD hcd \ #
@CESYSGEN ENDIF CE_MODULES_USBD # @CESYSGEN IF CE_MODULES_SERIAL serial \ # @CESYSGEN ENDIF
CE_MODULES_SERIAL ..... .....
```

3. Save and close the modified dirs file.

4.6 Step VI: Change

This step changes one secondary interrupt of the GPIO0 from the AC97 link to $\overline{\text{PENIRQ}}$ (TSC2003).

1. At the menu **File\Open…**, navigate, find, and open the software code file **intr.c** inside the directory:

   C:\WINCE500\PLAFORM\MAINSTONEII\SRC\KERNEL\OAL\
2. Change the line in the BSPIntrInit() routine from:

   OALIntrStaticTranslate(SYSINTR_TOUCH, IRQ_GPIO0_UCB1400);

   To:

   OALIntrStaticTranslate(SYSINTR_TOUCH, IRQ_GPIO0_PENIRQ);
3. Save and close the **intr.c** code file.

5 TSC2003 WinCE 5.0 Driver Code

To obtain the TSC2003 WinCE 5.0 driver code, contact the TI DAP Application Support Group at e-mail address *dataconvapps@list.ti.com*.

6 References

1. *TSC2301 WinCE Generic Drivers* application report (SLAA187)
2. *TSC2003EVM and TSC2003EVM-PDK User's Guide* (SBAU109)
3. *TSC2003, I^2C Touch Screen Controller* data sheet (SBAS162)
4. Intel PXA27x Processor Developer's Kit, order number 278827-005

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