ABSTRACT

The TSC2046 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 TSC2046 driver, including the hardware connection between the TSC2046 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/SLAA278.

1 Introduction

The TSC2046 WinCE 5.0 driver was developed for helping TSC2046 users to quickly set up, run, and use the device and to shorten software driver development time. The TSC2046 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 TSC2046 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 TSC2046EVM board (see SBAU100) and Intel MainStone II platform with the PXA270 Step B0 processor (see reference 4).

2 Connections

The TSC2046 device must be wired and connected to a host processor where the device driver code is ported and executed. The host processor controls TSC2046 through the interface that consists of five digital signals:

- The SPI bus, four wires: CS, DCLK, DIN, and DOUT
- The touch Pen-Down interrupt, PENIRQ
For developing the TSC2046 driver, the TSC2046 EVM and the Intel MainStone II platform with the PXA270 Step B0 processor were used. Figure 1 illustrates the wires and connections between the PXA27x and the TSC2046.

On the TSC2046 EVM board, a connector to J2 was made to wire the five digital signals SS, SCLK, MOSI, MISO, and PENIRQ. For information on the connector J2 and other details of the TSC2046EVM, see SBAU100.

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 five digital signal pins, the TSC2046 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 TSC2046 touch device driver’s code files. The files starting with “Host…” are the processor-dependent code or PDL, such as HostTouch.CPP or HostSPIComm.H. The PDL code was developed only for the PXA27x processor.
3.1 **SPI Interface**

The SPI bus is the control and data bus, through which the host processor sends commands to TSC2046 and reads the touch screen or other data back from TSC2046. The SPI communication code was developed as a library and put into the directory, **TSCLIB**.

On the hardware side, the four TSC2046 SPI pins, CS, DCLK, DIN, and DOUT, were connected to the GPIO23 to GPIO26 of the PXA27x processor, respectively.

On the software side, the PXA27x GPIO, SSP, and Clock management control registers were set up to communicate to the TSC2046 through the SPI. The setup was implemented at the routine, `HWInitSPI()`, which is inside file **HostSPIComm.C**:

```c
// Function: void HWInitSPI(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 // SSPSCLK GP23 output 2 // SSPSFRM GP24 output 0 // SSPTXD GP25 output 2 // SSPRXD GP26 input 1 //-------------------------------------------------------------------

BOOL HWInitSPI(BOOL InPowerHandle) { RETAILMSG(1,(TEXT("Setup Host GPIO & SSP for an SPI Interface...
\r\n"))); // disable Unit clock g_pClockRegs->cken &= ~XLLP_CLKEN_SSP1; // disable SSP1 g_pSSPRegs->sscr0 &= ~SSE_ENABLE; if (g_pGPIORegs) { // Set up the GPIO24-SFRM = 1 (GPSR0) g_pGPIORegs->GPSR0 |= ( GPIO_24_SFRM ); // Program direction of the GPIOs (GPDR0) (GPIO23/24/25 as outputs and GPIO26 as input) g_pGPIORegs->GPDR0 |= GPIO_23_SCLK; g_pGPIORegs->GPDR0 |= GPIO_24_SFRM; g_pGPIORegs->GPDR0 |= GPIO_25_MOSI; g_pGPIORegs->GPDR0 &= ~GPIO_26_MISO; // Program GPIO alternate function register (GAFR0_U) g_pGPIORegs->GAFR0_U |=( 0xFFF01FFF); g_pGPIORegs->GAFR0_U |= GPIO_23_AF2_SSPSCLK; g_pGPIORegs->GAFR0_U |= GPIO_24_AF2_SSPRXD; g_pGPIORegs->GAFR0_U |= GPIO_25_AF1_SSPRXD; }

// Program direction of the GPIOs (GPDR0) (GPIO23/24/25 as outputs and GPIO26 as input) g_pGPIORegs->GPDR0 |= GPIO_23_SCLK; g_pGPIORegs->GPDR0 |= GPIO_24_SFRM; g_pGPIORegs->GPDR0 |= GPIO_25_MOSI; g_pGPIORegs->GPDR0 &= ~GPIO_26_MISO;
```

---

---

---
// enable SPI1 Unit clock g_pClockRegs->cken |= XLLP_CLKEN_SSP1;

DmpGpio(); DmpSsp(); DmpClock(); return (TRUE); }

For the SPI interface, the routine SPITransaction() actually performs the sequence as shown in Figure 9 of the TSC2046 data sheet (SBAS265) which is:

//-- General Function for TSC2046 Control Register R/W // where "Command" is the command byte ;
// and // return R-Adjusted 12 or 8 bit data in [pWords[0] pWord[1]] //-----------------------------
UINT16 SPITransaction(UINT8 Command, BOOL bInPowerHandler) { UINT8 iTempData[3] = {0, 0, 0};
UINT16 iTempD = 0; if (!bInPowerHandler) { HWStartFrame(); while (HWSPITxBusy()) { HWWait(1); }

// command byte is written to TSC2046 by SPI HWSPIWriteWord(Command); HWWait(2); while (HWSPITxBusy()) { HWWait(1); } while (HWSPIRxBusy()) { HWWait(1); } iTempData[0] = HWSPITReadWord(); HWWait(2);

// read high byte of the ADC data HWSPIWriteWord(0); HWWait(2); while (HWSPITxBusy()) { HWWait(1); } while (HWSPIRxBusy()) { HWWait(1); } iTempData[1] = HWSPITReadWord(); HWWait(2);

// read low byte of the ADC data HWSPIWriteWord(0); HWWait(2); while (HWSPITxBusy()) { HWWait(1); } while (HWSPIRxBusy()) { HWWait(1); } iTempData[2] = HWSPITReadWord(); HWWait(2);

while (HWSPIFIFONotEmpty()) // clean up FIFO to make sure getting latest data ( iTempData[0] = iTempData[1]; iTempData[1] = iTempData[2]; iTempData[2] = HWSPITReadWord(); HWWait(2); )
HWStopFrame(); HWWait(2); iTempD = ( (0x00FF & ((UINT16)iTempData[2]) ) << 8 ) | (0x00FF & ((UINT16)iTempData[1]) ); return(iTempD >> 5); // return the 10 MSBs } else return(0); }

3.2 Touch Screen Driver

In the MainStone II system, the interrupt PENIRQ pin has been connected to an FPGA, where the PENIRQ was ORed with other secondary interrupts and fed to the PXA27x GPIO0 pin (see reference 4).

In this application, the TSC2046’s /PENIRQ pin was wired to the MainStone II’s PENIRQ pin, which would actually go 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 TSC2046 touch driver, the TSC2046 PENIRQ is enabled to detect any touch on the screen and the PENIRQ triggers the DdsiTouchPanelGetPoint () on the PDD layer whenever the 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()\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 ...
"))); } 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 timer // (pen- up doesn’t cause an interrupt). g_NextExpectedInterrupt = PEN_UP_OR_TIMER;
RETAILMSG(1,(TEXT("GetPoint: timer irq occurred ...
"))); } if (!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 ...
"))); } // 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()\n"))); }
In the \texttt{DdsiTouchPanelGetPoint()} routine, the \texttt{PDDSampleTouchScreen()} is called to perform a \texttt{SPITransaction()} as previously described.

4 Installation

This section presents the installation steps to run the TSC2046 WinCE 5.0 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 \texttt{\MainStoneII\}, which may be located in your PC after installing the Platform Builder 5.0 at, for example:

\texttt{C:\WinCE500\PLATFORM}.  

To install the TSC2046 Windows CE 5.0 driver into one of the MainStone II WorkSpaces, execute the following steps.

4.1 Step I: Copy

1. Copy \texttt{TSC2046WinCE5Driver\TSC2046.cec} file into:
   \texttt{C:\WINCE500\PUBLIC\COMMON\OAK\CATALOG\CEC}\n
2. Copy all files inside \texttt{TSC2046WinCE5Driver\INC\} into:
   \texttt{C:\WINCE500\PLATFORM\MAINSTONEII\SRC\INC}\n
3. Copy the directories \texttt{TSCLIB} and \texttt{TSCTouch} into:
   \texttt{C:\WINCE500\PLATFORM\MAINSTONEII\SRC\DRIVERS}\n
4.2 Step II: Set Up

This step sets up the catalog to include the TSC2046 device drivers.

1. Run \texttt{Platform Builder 5.0}, and the Platform Builder IDE appears.

2. At the Platform Builder 5.0 IDE, open \texttt{Manage Catalog Items} from the menu \texttt{File\Manage Catalog Items ...}. When the \texttt{Manage Catalog Items} window appears, click on \texttt{Import} button on the right side of the window, navigate, find, and select \texttt{TSC2046.cec} in the directory:

   \texttt{C:\WINCE500\PUBLIC\COMMON\OAK\CATALOG\CEC}\n
   Then click on \texttt{Open}, so that the item is ported in.

3. Click and drag to select all \texttt{*\textunderscore cec} files in the \texttt{Manage Catalog Items} window, and then click on the \texttt{Refresh} button to ensure that the new item is loaded.

4. Close the \texttt{Manage Catalog Items} window by clicking on its \texttt{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 TSC2046 device driver from the Catalog into the existing OS design.

1. In the \texttt{Catalog} window of the \texttt{Platform Builder 5.0 IDE}, find \texttt{TI TSC2046 Touch Controller Driver}, right-click on it, and select \texttt{Add to OS Design} to add the touch controller driver to the OS.

2. As the result, the touch device driver appears under the \texttt{Device Drivers} section at the \texttt{OSDesignView} window of the Workspace.

4.5 Step V: Modify

This step modifies the building device drivers so as to include TI TSC2046 touch driver.

1. Open the \texttt{dirs} file in the directory:

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

```plaintext
DIRS=-\ # @CESYSGEN IF CE_MODULES_POINTER TSCLIB \ TSCTOUCH\ # @CESYSGEN ENDIF
CE_MODULES_POINTER # @CESYSGEN IF CE_MODULES_DEVICE # @CESYSGEN IF CE_MODULES_USB hcd \ #
@CESYSGEN ENDIF CE_MODULES_USB # @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 original UCB1400 AC97 to PENIRQ (TSC2046).

1. At the menu `File\Open…`, navigate, find, and open the software code file `intr.c` inside the directory: `C:\WINCE500\PLAFORE\MAINSTONEII\SRC\KERNEL\OAL\`

2. Change the line in the `BSPIntrInit()` routine from:

   ```c
   OALIntrStaticTranslate(SYSINTR_TOUCH, IRQ_GPIO0_UCB1400);
   ```

   Into:

   ```c
   OALIntrStaticTranslate(SYSINTR_TOUCH, IRQ_GPIO0_PENIRQ);
   ```

3. Save and close the `intr.c` code file.

5 TSC2046 WinCE 5.0 Driver Code

To obtain the TSC2046 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. TSC2046EVM and TSC2046EVM-PDK User’s Guide (SBAU100)
3. TSC2046, Low Voltage I/O Touch Screen Controller data sheet (SBAS265)
4. Intel PXA27x Processor Developer’s Kit, order number 278827-005