

Windows CE .NET Touch Screen, Keypad, and Audio Device Drivers for the TSC2301

Bob Green and Wendy X. Fang
Data Acquisition Products

ABSTRACT

This application report describes a Windows CE .NET driver for the TSC2301, a Texas Instruments touch-screen/keypad controller, and a stereo audio codec device. Intel® PXA250 and PXA210 microprocessors are used as master processors for interfacing with the TSC2301. The driver was built using the board support package (BSP) of Intel's PCA development kit for the HCDDDBCTA1/Cotulla development platform.

Contents

| | |
|---|-----------|
| Introduction | 1 |
| Hardware Considerations | 2 |
| Software Architecture | 3 |
| Functional Description..... | 3 |
| System Setup | 3 |
| TSC2301 Touch panel Driver Overview | 4 |
| TSC2301 Keypad Driver Overview | 6 |
| SSP Overview..... | 7 |
| TSC2301 Audio Driver Overview | 7 |
| Source Code | 10 |
| Source Files..... | 10 |
| Installation Instructions | 11 |
| References..... | 15 |

Figures

| | |
|--|---|
| Figure 1. Integration of Monolithic Device Drivers Within the Windows CE System [1]..... | 3 |
|--|---|

Tables

| | |
|--|----|
| Table 1. Hardware Interface..... | 2 |
| Table 2. KeyPad Mapping..... | 6 |
| Table 3. Source Code for TSC2301 Driver..... | 11 |

Introduction

Texas Instrument's TSC2301 is a highly integrated analog interface device with touch screen and keypad controller and stereo audio codec, an ideal selection for use in many hand-held computation and communication applications.

As a TSC2301 driver design example, this application report presents a Windows CE .NET driver that was built using the board support package (BSP) of the Intel's personal client architecture (PCA) development kit with Intel's PXA250 microprocessor. On the hardware side, Intel's HCDDDBCTA1/Cotulla development platform was used and slightly modified to interface with TI's TSC2301 evaluation module (EVM). The driver described here is directly applied and tested only with the same hardware structure and connections.

The methods described herein are some of the possible implementations and are intended to serve as a guideline or example for hardware and software developers creating their own solutions. The sample setup and drivers provided demonstrate only the basic functionality of the TSC2301 with the PXA250, have not been extensively tested, and are not intended for use as is in production systems.

This application report assumes that the reader is familiar with hardware, touch screen, keyboard, and audio device driver development in general, and specifically the Microsoft Windows Platform Builder 4.0 for Windows CE .NET development environment, the Texas Instruments TSC2301 PDA analog interfaces, and the Intel® PXA250 microprocessor and development board.

Hardware Considerations

The development system used for this application report consists of a Texas Instruments TSC2301EVM board, a slightly modified Intel HCDDDBCTA1/Cotulla development platform, and some additional cabling. The modifications made to the HCDDDBCTA1 are minor and involve the routing of the /PENIRQ signal to an unused external connection point. This facilitates the connection of the /DAV signal from the TSC2301 to the HCDDDBCTA1. The HCDDDBCTA1 incorporates a four-wire resistive touch screen that is compatible with the TSC2301. The touch screen is connected directly to the TSC2301EVM. In order to test the system, several additional connections from the HCDDDBCTA1 to the TI board have been added and are shown in Table 1.

Table 1. Hardware Interface

| TSC2301 | HCDDDBCTA1 | Description |
|---------|----------------|--------------------------------|
| /SS | GP24, SSP_SFRM | SPI communications interface |
| SCLK | GP23, SSP_SCLK | SPI communications interface |
| MOSI | GP25, SSP_TXD | SPI communications interface |
| MISO | GP26, SSP_RXD | SPI communications interface |
| LRCLK | AC_SYNC | Audio interface |
| I2SDIN | AC_DOUT | Audio interface |
| I2SDOUT | AC_DIN | Audio interface |
| BCLK | AC_BITCLK | Audio interface |
| MCLK | GP32 | Audio interface (system clock) |
| /KBIRQ | GP4 | Keypad interrupt |
| /DAV | /PENIRQ | Touch screen interrupt |

The touch screen controller installed at the factory on the HCDDDBCTA1 is the TI ADS7846 touch screen controller. For this application report, the ADS7846 is disabled and the TSC2301 is substituted. The /DAV signal from the TSC2301 is connected to the /PENIRQ signal (used by the ADS7846) on the HCDDDBCTA1. Ultimately, this signal becomes an interrupt to the PXA250.

Software Architecture

Both the touch panel and keypad drivers have been developed as *layered* drivers using the Intel/Microsoft sample drivers as starting points. In addition, the touch panel driver relies upon the driver design for the ADS7846. This approach helps reduce overall project risk, since the model device driver (MDD) layer (supplied by Microsoft in the sample drivers) requires no change. Only the DDSI functions in the PDD (platform-dependent driver) layer are modified for this application. Refer to Figure 1.

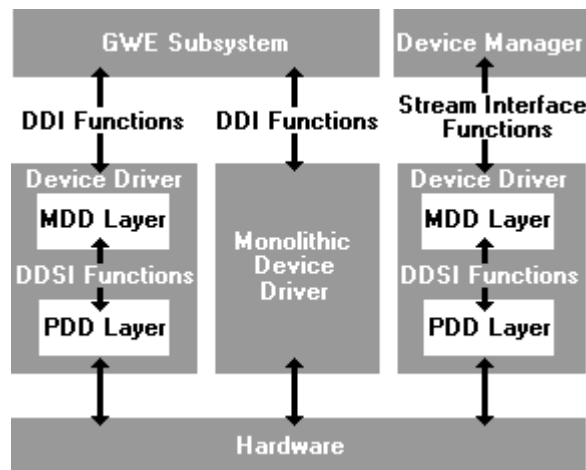


Figure 1. Integration of Monolithic Device Drivers Within the Windows CE System [1]

Functional Description

System Setup

1. Set up the HCDDDBCTA1/Cotulla development board in stand-alone configuration according to the manufacturers' documentation.
2. Update the platform files following the instructions included with Intel's board support package (BSP_WinCENet_DBPXA250_V3_00_008.exe).
3. Rebuild the platform following the instructions included with the Intel board support package (BSP_WinCENet_DBPXA250_V3_00_008.exe).
4. Connect the TSC2301EVM board to HCDDDBCTA1 development board. This requires connections to be made that are listed in Table 1.
5. Download the image to the target (see Microsoft documentation).
6. Use the keyboard to run the calibration program from the *Stylus* application in the Windows CE.NET control panel.

TSC2301 Touch Panel Driver Overview

The TSC2301 is configured to perform automatic X and Y conversions when the touch panel is pressed. The TSC2301 asserts (active low) the DAV line after the X and Y data are ready for reading. The DAV signal is shared among several ADC functional parts of the TSC2301. This requires that the driver be written to support sharing this signal. Due to the operation of the DAV logic and the timing of the touch panel driver, a pen up operation (pressure on the touch panel released) may be missed. In order to deal with that issue, a timer in the TSC2301 is set to cause an interrupt after a certain amount of time after a pen down operation (pressure put on the touch panel). This timer acts as a watchdog and ensures that the pen up event is not missed. The overall operation of the touch panel driver is discussed in the following paragraphs.

1. The driver loads. The touch panel driver is loaded into the system. This causes the TSC2301 to be configured for continuous scanning. The GPIO line connected to DAV is configured to detect the falling edge of the signal and then to generate an interrupt.
2. The touch panel is pressed. This causes the TSC2301 to perform X and Y conversions and to assert DAV. The assertion of DAV in turn causes an interrupt to occur.
3. The touch panel driver interrupt handler is called. The interrupt handler `DdsiTouchPanelGetPoint` has been constructed to deal with several different situations that can occur due to the multiple uses of the DAV line with respect to the touch panel driver. Protection against possible spurious interrupt conditions has also been built into the driver as added protection. Close inspection of the source code for the driver is important in order to gain a full operational understanding. The important decision points and actions of the handler are summarized below:
 - a. The current state of the DAV line is checked.
 - b. If no other TSC2301 related driver is expecting DAV to be asserted, then this interrupt is intended for the touch panel driver. Otherwise, an ignore message is returned to the operating system and execution of the interrupt handler ends. *A mutex is used to indicate which functional portion of the TSC2301 is currently using DAV.*
 - c. TSC2301 touch panel timer interrupts are disabled.
 - d. TSC2301 conversions are disabled so that the data is stable while being read.
 - e. The current state of the touch panel is determined (pressed—*pen down*, or not pressed—*pen up*).
 - f. If this interrupt is in response to an assertion of the DAV line:
 - 1) When the touch panel is pressed (*pen down*) and the DAV line is asserted, the following occurs:
 - *This is an actual interrupt in response to a pen down condition.*
 - The X and Y value of the *pen down* is recorded.
 - The touch panel timer interrupt is enabled (to catch the *pen up* event).
 - TSC2301 conversions are enabled.
 - A flag is set to indicate that the next type of interrupt expected is in response to a *pen up* event or the touch timer.
 - The recorded touch panel data is returned to the operating system and execution of the interrupt handler ends.

- 2) Otherwise, if a *pen up* or touch timer interrupt is expected, the following occur:
 - *This is an actual interrupt in response to a pen up condition.*
 - TSC2301 conversions are enabled.
 - A flag is set to indicate that the next type of interrupt expected is in response to a *pen down* event.
 - The recorded touch panel data is returned to the operating system and execution of the interrupt handler ends.
- 3) Otherwise:
 - *This is an unexpected pen up event.*
 - The touch panel timer interrupt is enabled.
 - An ignore message is returned to the operating system, and execution of the interrupt handler ends.
- g. Otherwise, if this interrupt is in response to a TSC2301 touch timer interrupt, the following occur:
 - 1) If the pen is up, and a *pen up* or timer interrupt is expected, the following occurs:
 - *This is a situation where the pen up event occurred but the DAV interrupt associated with that event was missed.*
 - TSC2301 conversions are enabled.
 - A flag is set to indicate that the next type of interrupt expected is in response to a *pen down* event.
 - The recorded touch panel data is returned to the operating system and execution of the interrupt handler ends.
 - 2) Otherwise, if DAV is asserted and the next expected interrupt is for a *pen down* event, the following occur:
 - *This is a situation where the state of DAV is out of sync with what is expected. The driver cleans up the current state.*
 - All registers in the TSC2301 are read, causing DAV to be not asserted.
 - The touch panel timer interrupt is enabled.
 - An ignore message is returned to the operating system, and execution of the interrupt handler ends.
 - 3) Otherwise the following occur:
 - *This is a spurious timer interrupt and no action is required.*
 - TSC2301 conversions are enabled.
 - An ignore message is returned to the operating system, and execution of the interrupt handler ends.
- h. Otherwise the following occur:
 - 1) This is an unknown interrupt.
 - 2) TSC2301 conversions are enabled.
 - 3) An ignore message is returned to the operating system and execution of the interrupt handler ends.
4. The touch panel driver is unloaded. During the system shutdown procedure, the touch panel driver is unloaded from the system. All memory allocated to the driver is recovered.

TSC2301 Keypad Driver Overview

The keypad driver works very similar to the touch panel driver. The KBIRQ is asserted (low) when a key is pressed. As with the touch panel driver's DAV line, KBIRQ also pulses high and then low again as the TSC2301 continues scanning the keypad while a button is still down. Again, this causes problems with recognizing the actual button up event. The keypad driver uses an approach similar to that used in the touch panel driver. When KBIRQ interrupt fires, we set a timer long enough to allow the TSC2301 to complete at least one more keypad scan. When a timer interrupt occurs before the next KBIRQ interrupt, and the TSC2301 says that no keys are pressed, it is assumed an actual button release occurs and acts accordingly. This timer is set in the `KeybdIstLoop` function in `KB_DIST.CPP` and setting it to 1.5 times the TSC2301's debounce time works well.

The standard Windows AutoRepeat key functionality does not currently work with this driver due to the fact that KBIRQ pulses high between each scan of the keypad. This pulse causes another KBIRQ falling edge interrupt to occur before the autorepeat minimum timeout can occur. Therefore no autorepeating occurs in this driver.

Two arrays (`arrVkeys` and `arrScanCodes`) at the top of `TSC_KEYPD.CPP` are used to map the keypad keys to Windows Virtual Keys. Entry 0 in these arrays matches to Bit0 in the `KEYDATA` register of the TSC2301. The current mapping of the keypad keys is shown in Table 2.

Table 2. Keypad Mapping

| TSC2301 Key | Windows Virtual Key |
|-------------|---------------------|
| 0 | '0' |
| 1 | '1' |
| 2 | '2' |
| 3 | '3' |
| 4 | '4' |
| 5 | '5' |
| 6 | '6' |
| 7 | '7' |
| 8 | '8' |
| 9 | '9' |
| A | VK_LEFT |
| B | VK_UP |
| C | VK_RIGHT |
| D | VK_DOWN |
| E | VK_ESCAPE |
| F | VK_RETURN |

There are also two `#defines` (`KEYREG_SETUP_VALUE` and `KEYMASKREG_SETUP_VALUE`) in `TSC_SSP.H` that are used to set up the TSC2301 `KEY` and `KPMASK` registers.

The keypad driver's source code implements a system where the keypad is a stand-alone driver. In addition, source code has been included which demonstrates how to combine the TSC2301 keypad driver with an existing PS2 keyboard driver. Even in the combined version, the TSC2301 keypad still requires a dedicated GPIO input line for KBIRQ.

SSP Overview

In order to share the single SPI connection with the TSC2301 between both the touch panel and the keypad drivers, a new module (`tsc_ssp.cpp`) was added to the DRVLIB library in the XSC1BD platform. This module sets up, configures, and uses the PXA250's SPI interface to communicate with the TSC2301 and uses a named mutex to prevent collision between the two drivers. Also, since the keyboard driver loads first, it initially resets and configures the TSC2301 with the touch driver, making any additional configuration changes required when it loads.

The PXA250's SSP port is configured to use the Motorola SPI format with the SPH and SPO registers configured to match the requirements of the TSC2301 as follows:

```

RIE_DISABLE,
TIE_DISABLE,
LBM_DISABLE,
SPO_IDLE_LOW,
SPH_HALF_DELAY,
MWDS_16_BIT,
TFT_ZERO,
RFT_SEVEN,
DSS_16_BIT,
FRF_MOTOROLA,
ECS_INTERNAL,
SSE_ENABLE,
SCR_600_KHZ,
GAFR1, GPIO_23_AF2_SCLK,
GAFR1, GPIO_24_AF2_SFRM,
GAFR1, GPIO_25_AF2_TXD,
GAFR1, GPIO_26_AF1_RXD,
GPDR, GPIO_23_OUTPUT,
GPDR, GPIO_24_OUTPUT,
GPDR, GPIO_25_OUUPUT,
GPDR, GPIO_26_INPUT,
  
```

TSC2301 Audio Driver Overview

This audio driver is implemented as a PDD layer using the model device driver (MDD) library — `Wavemdd.lib` — supplied by Microsoft. See the following website for more details regarding the MDD and PDD audio driver model.

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/wceddk40/hm/cxconmddpdd.asp>

This PDD layer uses the TI SSP interface to control the audio capabilities of the TSC2301 while using DMA transfers to the PXA250 I2S interface to actually send and receive audio data. Refer to the appropriate chapters in the PXA250 and PXA210 Application Processors Developer's Manual for more information on DMA and I2S. A comprehensive description of these two technologies is beyond the scope of this document.

This driver contains a few interesting features not commonly found in audio drivers.

In order to simultaneously play and record .WAV files of different sample rates, this driver implements real time resampling while the TSC2301 hardware is always running at 44.1 kHz. So, even though the TSC2301 records audio data at 44.1kHz, we can end up with a recorded .WAV file at several lower sample rates. Conversely, .WAV files of lower sample rates are up sampled before being output at 44.1kHz.

Automatic gain control has been implemented by examining the highest amplitude in each block of recorded audio data. Users can specify a gain target and tolerance.

In addition, through PDD_AudioMessage (described below) two functions are opened to allow user applications to directly set and read any TSC2301 register. This allows users with specific needs to directly monitor or control unusual conditions with the TSC2301.

In general, the overall flow of the audio driver is well documented in Microsoft's MSDN library. However, a simplified description is offered here. Once the driver is initialized by the system, user applications use the waveOut and waveIn functions to begin the audio data transfer. These functions (after passing through the MDD driver layer) eventually call the PDD_WaveProc function. These messages instruct the driver to open, start, and stop waveform playback and record. PDD_WaveProc then calls various internal functions (described below) to actually control the audio playback or record. The TI driver then sets up and controls the I2S interface and the DMA required to support it, in addition to controlling the TSC2301 through the SSP interface (described above).

The main functions of the PDD controlling the TSC2301 audio functionality are:

PDD_AudioInitialize performs any one-time initializations for the audio driver. In addition to setting up the necessary DMA structures, the TI driver uses this function to initialize the SSP interface to communicate with the TSC2301, and to modify default TSC2301 registers settings if desired. The current driver uses the TSC2301 defaults as is because all relevant audio settings are configured when a .WAV file is output or recorded.

PDD_AudioGetInterruptType determines the cause of the audio interrupt and returns current device status. The TI driver examines the DMA structures to determine the cause of the interrupt and returns an appropriate status.

PDD_AudioMessage sends messages from user applications to the audio driver's PDD layer. Custom messages can be accessed through this function. It is ultimately called in response to the waveOutMessage and waveInMessage called by the user application. Fourteen custom audio messages are set up for user applications:

WPDN_PRIVATE_TSC_AUTO_GAIN_CONTROL: Allows users to configure the automatic gain control by setting both the target gain (dwParam1) and the gain tolerance (dwParam2). Setting dwParam1 to 0 turns off automatic gain control.

WPDN_PRIVATE_WRITE_TSC: Allows users to set any TSC2301 register. This passes dwParam1 and dwParam2 along in a call to TSC2301GenericWriteReg(dwParam1, dwParam2).

WPDN_PRIVATE_READ_TSC: Allows users to read any TSC2301 register. This passes dwParam1 and dwParam2 along in a call to TSC2301GenericReadReg(dwParam1, dwParam2).

WPDN_PRIVATE_TSC_SetInput_MIC: Sets the TSC2301 audio control register to have both the left and right input set to the microphone input.

WPDN_PRIVATE_TSC_SetInput_AUX: Sets the TSC2301 audio control register to have both the left and right input set to the corresponding auxiliary inputs.

WPDN_PRIVATE_TSC_SetInput_AUX_MIX: Sets the TSC2301 audio control register to have both the left and right input set to the *mono mix* of the left and right auxiliary inputs.

WPDN_PRIVATE_TSC_SetInput_AUX_SWAP: Sets the TSC2301 audio control register to have both the left and right input set to the opposite auxiliary inputs (left input is set to right auxiliary and vice versa).

WPDN_PRIVATE_TSC_SET_MIC_GAIN_0dB: Sets the microphone gain in TSC2301 audio control register to 0 dB.

WPDN_PRIVATE_TSC_SET_MIC_GAIN_6dB: Sets the microphone gain in TSC2301 audio control register to 6 dB.

WPDN_PRIVATE_TSC_SET_MIC_GAIN_12dB: Sets the microphone gain in TSC2301 audio control register to 12 dB.

WPDN_PRIVATE_TSC_SET_HPF_DISABLE: Disables the high pass filter in the TSC2301 audio control register.

WPDN_PRIVATE_TSC_SET_HPF_1fs: Sets the high pass filter in the TSC2301 audio control register to 0.1xFs.

WPDN_PRIVATE_TSC_SET_HPF_78fs: Sets the high pass filter in the TSC2301 audio control register to 0.000078xFs.

WPDN_PRIVATE_TSC_SET_HPF_19fs: Sets the high pass filter in the TSC2301 audio control register to 0.000019xFs.

PDD_WaveProc sends standard audio control messages to the audio driver's PDD layer. This function is used to implement all the waveOut and waveIn functions at the PDD level. The WPDM messages passed to PDD_WaveProc are:

WPDM_CLOSE, WPDM_CONTINUE, WPDM_ENDOFDATA, WPDM_GETDEVCAPS, WPDM_GETVOLUME, WPDM_OPEN, WPDM_PAUSE, WPDM_RESTART, WPDM_SETVOLUME, WPDM_STANDBY, WPDM_START, WPDM_STOP.

Refer to the relevant Microsoft documentation for information regarding how these messages are initiated by the user and passed through the MDD driver layer.

As stated above, PDD_WaveProc either handles the message itself or calls various internal functions to handle each of these messages.

Below is a more detailed description of how the TI driver handles each of the WPDM message in PDD_WaveProc.

WPDM_CLOSE: updates the appropriate internal reference count variable to signal the DMA handling routines that the .WAV file has been closed.

WPDM_CONTINUE: calls either `private_WaveInContinue()` or `private_WaveOutContinue()`.

WPDM_ENDOFDATA: calls `private_WaveOutEndOfData()` for output .WAV files, and returns `MMSYSERR_NOTSUPPORTED` for input .WAV files.

WPDM_GETDEVCAPS: calls `private_WaveGetDevCaps()`.

WPDM_GETVOLUME: returns internally managed variable `m_nVolume`.

WPDM_OPEN: calls `private_WaveOpen()`

WPDM_PAUSE: calls `private_waveOutPause()` for output .WAV files, and returns `MMSYSERR_NOTSUPPORTED` for input .WAV files.

WPDM_RESTART: calls `private_waveOutRestart()` for output .WAV files, and returns `MMSYSERR_NOTSUPPORTED` for input .WAV files.

WPDM_SETVOLUME: sets `m_nVolume` and calls `TSCSetVolume()`.

WPDM_STANDBY: calls `private_WaveStandby()`;

WPDM_START: calls either `private_WaveInStart()` or `private_WaveOutStart()`.

WPDM_STOP: calls either `private_WaveInStop` or `private_WaveOutStop()`.

Source Code

Source Files

Table 3 lists the files that were added in or modified from Intel's BSP, for developing this application report. The TSC2301 driver files are grouped according to their basic functions. The main Windows CE .NET platform directory is: **C:\WINCE400\PLATFORM\XSC1BD**.

The files given in Table 3 contain the individual functions used in these drivers. This application report assumes the reader understands Windows CE, touch screen, keyboard, and audio drivers in general; therefore it does not cover every function needed for a working driver. These functions are required to be added or modified in order to work with the TSC2301. Other BSP files from Microsoft or Intel are also needed but not listed here.

Table 3. Source Code for TSC2301 Driver

| Group | Sub-Directory | File Name | Basic Function | Added/Modified |
|---------|--------------------------------|--------------|--|----------------|
| General | (in the main) | xsc1bd.bat | Set no USB client for BSP | Modified |
| General | \drivers\drvlib\ | SOURCES | Add tsc_ssp.c to sources | Modified |
| General | \incl\ | tsc_reg.h | Define TSC2301 registers | Added |
| General | \drivers\drvlib\ | tsc_ssp.c | Handle TSC and SSP communication | Added |
| General | \incl\ | tsc_ssp.h | Contain TSC and SSP communication prototypes and configuration defines | Added |
| General | \kernel\hal\ | cfwXcs1.c | Configure interrupts | Modified |
| General | \kernel\hal\arm\ | intxcs1.c | Interrupt service routines | Modified |
| Touch | \drivers\touchp\ | tchpdd.cpp | Touch screen functions | Modified |
| KeyPad | \drivers\kbdmouse\KBDMSCOMMON\ | ps2keybd.cpp | Keypad functions | Modified |
| Audio | \drivers\WAVEDEV\ | wavepdd.c | Audio functions | Modified |
| Audio | \drivers\WAVEDEV\ | wavepdd.h | Define audio parameters | Modified |
| Audio | \drivers\WAVEDEV\ | DMAC.h | Redefine DMAC In/Out | Modified |
| Audio | \incl\ | DMACbits.h | | Modified |

Installation Instructions

After the system setup, described in the previous section, the following three new files need to be copied in their entirety over the files from the Intel BSP:

C:\WINCE400\platform\xsc1bd\incl\tsc_reg.h

C:\WINCE400\platform\xsc1bd\drivers\tsc_ssp.c

C:\WINCE400\platform\xsc1bd\incl\tsc_ssp.h

Due to the large number of changes, the following three files need to be modified with code patches. The TI website has associated zip files containing the necessary code patches:

C:\WINCE400\platform\xsc1bd\drivers\touchp\tchpdd.cpp

C:\WINCE400\platform\xsc1bd\drivers\kbdmouse\KBDMSCOMMON\ps2keybd.cpp

C:\WINCE400\platform\xsc1bd\drivers\WAVEDEV\wavepdd.c

All other modified files, listed in Table 3, must be updated according to the following descriptions in order to work correctly with the TSC2301.

1. C:\WINCE400\platform\xsc1bd\XSC1BD.BAT

Under the Lubbock configuration, make sure that BSP_NOUSBCLIENT is set to 1 as shown below.

Set BSP_NOUSBCLIENT=1

2. C:\WINCE400\platform\xsc1bd\Drivers\drvlib\sources:

* Add TSC_SSP.C to end of SOURCES entry.

3. C:\WINCE400\platform\xsc1bd\kernel\hal\cfwXcs1.c

* In the function OEMInterruptEnable() under the SYSINTR_TOUCH case replace what is there with the following:

```
v_pBLReg->int_set_clr &=~BB_TS_PEN;
v_pBLReg->int_msk_en |= BB_TS_PEN_EN;
```

* In the function OEMInterruptEnable() under the SYSINTR_TOUCH_CHANGED case replace what is there with the following:

```
v_pBLReg->int_set_clr &=~BB_TS_PEN;
v_pBLReg->int_msk_en |= BB_TS_PEN_EN;
```

* In the function OEMInterruptDisable () under the SYSINTR_TOUCH case replace what is there with the following:

```
v_pBLReg->int_msk_en &= ~BB_TS_PEN_EN;
```

* In the function OEMInterruptDone () under the SYSINTR_TOUCH case replace what is there with the following:

```
v_pBLReg->int_set_clr &= ~BB_TS_PEN;
v_pBLReg->int_msk_en |= BB_TS_PEN_EN;
```

* In the function OEMInterruptDone () under the SYSINTR_TOUCH_CHANGED case replace what is there with the following:

```
v_pBLReg->int_set_clr &= ~BB_TS_PEN;
v_pBLReg->int_msk_en |= BB_TS_PEN_EN;
```

4. C:\WINCE400\platform\xsc1bd\kernel\hal\arm\intxsc1.c

* In the function OEMInterruptHandler(), add the following just before the first #ifdef PLAT_LUBBOCK statement:

```
// Bg 16SEP02 - copying this variable from function below
// to allow us to fix a problem where the touch driver's interrupt gets missed.
unsigned int InterestingInterrupts;
```

* Several lines down, add the two lines designated by ">" between the lines without the ">" designation:

```
v_pDrvGlob
(PDRIVER_GLOBALS)DRIVER_GLOBALS_PHYSICAL_MEMORY_START;

> // Bg 16SEP02 - copying this variable from function below
> InterestingInterrupts = v_pBLReg->int_set_clr & v_pBLReg->int_msk_en;
```

```
//Read the interrupt pending register
ipreg_copy = v_pICReg->icip;
```

* Also in the function OEMInterruptHandler(), add the following lines designated by the ">" just before return SYSINTR_TOUCH_CHANGED;

```
INTC_M1_INT_DIS(v_pICReg->icmr);
TIMER_M1_INT_CLR(v_pOSTReg->ossr);
v_pDrvGlob->tch.timerIrq=1;
> v_pDrvGlob->tch.touchIrq = 0;
```

* In the section under else if(ipreg_copy & INTC_GPIO80_2), add the following before the #ifdef PLAT_SANDGATE

```
RETAILMSG(1, (TEXT("OEMInterruptHandler - INTC_GPIO80_2\r\n")));
// Bg 22OCT02 - see if GPIO_4 (which we are using for the keypad interrupt has
gone off).
if (v_pGPIOReg->GEDR_x & GPIO_4) // now we have a pen down interrupt
{
```

```
RETAILMSG(1, (TEXT("OEMInterruptHandler - GPIO_4 - GEDR_x: 0x%X, GPLR_x:
0x%X\r\n"), v_pGPIOReg->GEDR_x, v_pGPIOReg->GPLR_x));
```

```
// clear the interrupt
//v_pGPIOReg->GFER_x &= ~GPIO_4;
//v_pGPIOReg->GEDR_x |= GPIO_4;
```

```
return SYSINTR_KEYBOARD;
```

```
}
```

* In the function FPGAInterruptHandler(), add the following else section to the return SYSINTR_TOUCH;

```
// Bg 06JUL02 - adding support for BB PEN IRQ
// this is critical - otherwise we'll never get the PEN interrupt
else if (InterestingInterrupts & BB_TS_PEN) // PENIRQ WENT LOW
{
```

```
v_pBLReg->int_msk_en &= ~BB_TS_PEN;
//Disable interrupt
v_pDrvGlob->tch.touchIrq=1;
v_pDrvGlob->tch.timerIrq=0;
```

```
// BgBgBg
//RETAILMSG(1, (TEXT("FPGAInterruptHandler -- PENIRQ SYSINTR_TOUCH!\r\n")));
```

```
return SYSINTR_TOUCH;
}
```

5. C:\WINCE400\platform\xsc1bd\drivers\wavedev\wavepdd.h

* Replace and/or redefine the currently defined private messages with the following:

```
//Private Messages
#define WPDM_PRIVATE_RET_NOTSUPPORTED MMSYSERR_NOTSUPPORTED

#define WPDM_PRIVATE 10000
#define WPDM_PRIVATE_WRITE_TSC (WPDM_PRIVATE+1)
#define WPDM_PRIVATE_READ_TSC (WPDM_PRIVATE+2)
#define WPDM_PRIVATE_TSC_SetInput_MIC (WPDM_PRIVATE+3)
#define WPDM_PRIVATE_TSC_SetInput_AUX (WPDM_PRIVATE+4)
#define WPDM_PRIVATE_TSC_SetInput_AUX_MIX (WPDM_PRIVATE+5)
#define WPDM_PRIVATE_TSC_SetInput_AUS_SWAP (WPDM_PRIVATE+6)
#define WPDM_PRIVATE_TSC_SET_MIC_GAIN_0dB (WPDM_PRIVATE+7)
#define WPDM_PRIVATE_TSC_SET_MIC_GAIN_6dB (WPDM_PRIVATE+8)
#define WPDM_PRIVATE_TSC_SET_MIC_GAIN_12dB (WPDM_PRIVATE+9)
#define WPDM_PRIVATE_TSC_SET_HPF_DISABLE (WPDM_PRIVATE+10)
#define WPDM_PRIVATE_TSC_SET_HPF_1fs (WPDM_PRIVATE+11)
#define WPDM_PRIVATE_TSC_SET_HPF_78fs (WPDM_PRIVATE+12)
#define WPDM_PRIVATE_TSC_SET_HPF_19fs (WPDM_PRIVATE+13)
#define WPDM_PRIVATE_TSC_SET_DAC_GAIN (WPDM_PRIVATE+14)
#define WPDM_PRIVATE_TSC_SET_BYPASSSS_VOLUME (WPDM_PRIVATE+15)
#define WPDM_PRIVATE_TSC_AUTO_GAIN_CONTROL (WPDM_PRIVATE+16)

#define WPDN_PRIVATE_DEBUG_PLAYBACK_RATE (WPDM_PRIVATE+100)
```

6. C:\WINCE400\platform\xsc1bd\drivers\wavedev\dmac.h

* Add the following between the definitions of MAC_AC97_MICAB_CMD_MASK and DMA_RCV_A_DESCRIPTOR_BASE_PHYSICAL:

```
// 24JUL02 - Bg adding DMA support for I2S
#define DMAC_I2S_RCV_FIFO 0x40400080
#define DMAC_I2S_XMIT_FIFO 0x40400080

#define DMAC_I2S_RCVAB_CMD_MASK 0x4023C000 //0100 0000 01 000 11 10 0 000000000000
#define DMAC_I2S_XMITAB_CMD_MASK 0x8043C000 //1000 0000 10 000 11 10 0 000000000000
#define DMAC_I2S_MICAB_CMD_MASK 0x4023C000 //0100 0000 01 000 11 10 0 000000000000
```

7. C:\WINCE400\platform\xsc1bd\inc\dmacbits

Comment out the definitions of DMAC_AC97AUDIOXMIT, DMAC_AC97AUDIORCV, and DMAC_AC97MIC, and redefine the following:

```
#define DMAC_I2SXMIT (0x1 << DMA_CH_OUT)
#define DMAC_I2SRCV (0x1 << DMA_CH_RCV)
#define DMA_AUDIO_INTR (DMAC_I2SXMIT | DMAC_I2SRCV)
```

References

1. "Writing Device Drivers from Microsoft's CE 3.0", Microsoft
2. TSC2301 data sheet, Texas Instruments (SLAS371)
3. HCDDDBCTA1/Cotulla Development Platform - User's Guide, Intel
4. HCDDDBCTA1 I/O Baseboard – Schematic Diagram, Intel
5. Intel® PCA Development Kit – Installing Microsoft Windows CE.NET, Intel
6. Intel® Cotulla Microprocessor Development Board Support Package for Microsoft Windows CE .NET and Pocket PC2002 – User's Guide, Intel
7. Intel® PXA250 and PXA210 Application Processors Developer's Manual, Intel
8. Related documentation for Windows CE Platform Builder 4.0

Many of these documents, as well as other related documentation, can be downloaded from the respective websites, or found on CDs provided by manufacturers:

<http://www.ti.com>

<http://developer.intel.com>

<http://www.microsoft.com/windows/embedded/ce/tools/default.asp>

Windows is a trademark of Microsoft

Intel® is a registered trademark of Intel Corporation

SPI is a trademark of Motorola

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Mailing Address:

Texas Instruments
Post Office Box 655303
Dallas, Texas 75265