TSC2101 Touch Screen, Battery, and Audio WinCE Drivers

Dan Fahrion, HandEra Inc *Wendy X. Fang, DAP*

ABSTRACT

This application report describes the TSC2101 touch screen, battery, and audio drivers for the WinCE operating system to help customers to implement designs using the TSC2101 audio CODEC with integrated headphone/speaker amplifier, and touch-screen controller. The drivers and associated code are discussed. The WinCE driver code can be integrated into the customer's software system under different host processors and have been tested and used on Intel's Lubbock and MainStone platforms.

Contents

Description **2**
SPI Interface 3
Hardware Interface 3
TSC2101 Control Registers 3
SPI Driver 4
TSC2101 Touch-Screen Driver 5
Touch-Screen Driver Initialization 5
Reading Touch Data 6
TSC2101 Battery Driver 6
TSC2101 Audio Driver 6
Audio Initialization 7
Audio Data Transformation 8
Audio Messages 8
Driver Summary 8
Installation **9**
1. $(BASE)\Kernel\HAL\cfwxsc1.c 10
2. $(BASE)\Kernel\HAL\ARM\intxsc1.c 10
3. $(BASE)\dirs 10
4. $(BASE)\DRIVERS\WAVEDEV\wavemain.cpp 11
5. $(BASE)\DRIVERS\WAVEDEV\sources 11
References **11**

Figures

Figure 1. Layered TSC2101 WinCE Driver **2**
Figure 2. TSC2101 Touch-Screen Driver Initialization - *InitTSC2101Touch( )* **5**

Tables

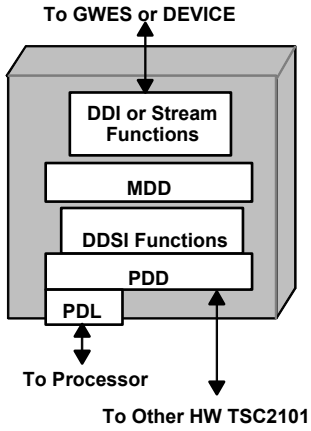
Table 1. PXA250 (Also Called Cotulla) and TSC2101 SPI HW Interface **3**
Table 2. SPI Driver Routines **4**
Table 3. TSC2101 WinCE Drivers and Processor Requirements **8**

Description

WinCE drivers were developed for the touch screen, the battery monitor, and the audio functions of the TSC2101 device. In the Windows CE device driver model, these are:

- TouchP: a standard touch panel/screen driver for the TSC2101 touch screen feature; and
- WaveDev or WaveDev2: a standard audio driver for the TSC2101 audio feature.

The touch-screen driver is a typical built-in or native device driver; and the audio driver can be classified as a hybrid (both native and stream) driver. The touch screen and audio drivers have layered architectures, as shown in 0.



Layered TSC2101 WinCE Driver

Generally, layered-driver development requires changes only to the platform-dependent device (PDD) layer. For more information on WinCE driver architecture and classification, see the application report *TSC2301 WinCE Generic Drivers*, (SLAA187) or related Microsoft documentation.

As shown in 0, the TSC2101 has an additional layer below the standard PDD layer called the processor-dependent layer (PDL). The PDL layer facilitates adaptation of the drivers to various platforms. Adapting the TSC2101 drivers to a new platform typically requires modifications only to the PDL layer.

Developing the WinCE drivers for the TSC2101 involves the following tasks:

- Developing a common library that implements SPI data communications with the TSC2101
- Developing the PDD (and the PDL) layer of the TSC2101 touch-screen driver (native driver mounted by the Graphics, Windowing, and Events Subsystem (GWES) module)
- Developing the TSC2101 battery-monitor driver to report the battery status to GWES
- Developing the PDD (and the PDL) layer of the TSC2101 audio driver (hybrid driver mounted by device).

SPI Interface

The SPI bus on the TSC2101 is the primary hardware interface to the host processor. The host processor accesses the TSC2101 registers through the SPI interface. The SPI driver is the most important software interface between the host processor and the TSC2101.

Hardware Interface

A standard SPI hardware interface consists of four signals, normally named SCK (clock), SS (slave select), MOSI (master-out, slave-in data), and MISO (master-in, slave-out data). As an example, Table 1 lists the SPI connection between the Lubbock (Cotulla) platform and the TSC2101.

Table 1. PXA250 (Also Called Cotulla) and TSC2101 SPI HW Interface

	Host Processor Pin Name	TSC2101 Pin Name
SPI Clock	GPIO 23 (Pin-F9)	SCLK (QFN Pin 4 or TSSOP Pin 8)
SPI Slave Select	GPIO 24 (Pin-E9)	/SS (QFN Pin 7 or TQFP Pin 11)
SPI MOSI Data	GPIO 25 (Pin-D9)	MOSI (QFN Pin 6 or TQFP Pin 10)
SPI MISO Data	GPIO 26 (Pin-A9)	MISO (QFN Pin 5 or TQFP Pin 9)

In the SPI interface, the TSC2101 is always the slave device, and the host processor is the master.

TSC2101 Control Registers

The TSC2101 registers are organized as three memory pages that contain data, status, all programmable controls, variables, and parameters. These TSC2101 registers are accessed and controlled by the host processor through the SPI interface. For a full description of the registers, see the TSC2101 data sheet.

The TSC2101 register definitions have been coded in a header file, *TSC2101Regs.H*, available for download from the TI website.

SPI Driver

The TSC2101 SPI driver establishes the software interface between the processor and TSC2101. It contains 4 files, *TSC2101SPI.C*, *TSC2101SPI.H*, *XXXXXSPIComm.C*, and *XXXXXSPIComm.H*, where the *XXXXX* stands for the name of the processor. The latter two files are processor dependent and part of the PDL. For example, for the PXA25x XScale processor, the two processor-dependent files may be named *XScaleSSPComm.C* and *XscaleSSPComm.H*; for the Bulverde processor, they are named *BulverdeSSPComm.C* and *BulverdeSSPComm.H*.

The fundamental routines for the SPI driver are summarized in Table 2. The processor-related PDL routines in Table 2 have *HW* as the first two letters of the names.

Table 2. SPI Driver Routines

Item	Routine Name	Involved Processor-Dependent Routines	Function
1	SetupSPIController( )	HWInitializeSPIDriver( ) HWSetupSPIController( )	Configure the host-processor SPI port
2	StopSPIController( )	HWDeinitializeSPIDriver( ) HWStopSPIController( )	Stop the host-processor SPI port and interface
3	SPITransaction( )		Write/read one or more TSC2101 registers
		HWStartFrame( )	Assert /SS (goes low)
		HWStopFrame( )	De-assert /SS (goes high)
		HWSPIWriteWord( )	Write a word to MOSI
		HWSPIReadWord( )	Read a word from MISO
		HWSPITxBusy( )	To check if an SPI transmit has completed
		HWSPIRxBusy()	To check if an SPI receive has completed
		HWSPIFIFONotEmpty( )	To ensure the SPI FIFO has been properly read so that the read data are the latest
4	TSC2101ReadReg( )		To read the content from a TSC2101 register
5	TSC2101WriteReg( )		To write a 16-bit value to a TSC2101 register

In Table 2, the routines in items 1 to 3 closely relate to the processor and its SPI configuration. Items 4 and 5 allow the host processor to access the TSC2101 control registers.

TSC2101 Touch-Screen Driver

The TSC2101 touch-screen driver normally resides in the standard WinCE TouchP directory. The driver includes the file: *TSC2101Touch.CPP*, together with the PDL files *XXXXXTouch.CPP* and *XXXXXTouch.H*, where the *XXXXX* represents the specific processor being used.

The TSC2101 touch-screen driver architecture conforms to that of the WinCE OS, with modifications to the standard DDSI functions. See the application report *TSC2301 WinCE Generic Drivers* (SLAA187) or Microsoft WinCE driver documentation for more details on the standard touch-screen DDSI functions and routines.

Touch-Screen Driver Initialization

The touch-screen driver initializes when the subroutine *InitTSC2101Touch( )* is called in the WinCE touch-screen DDSI routine *DdsiTouchPanelEnable( )*. Even though the TSC2101 does not require a strict initialization sequence, the order shown in Figure 1 is recommended.

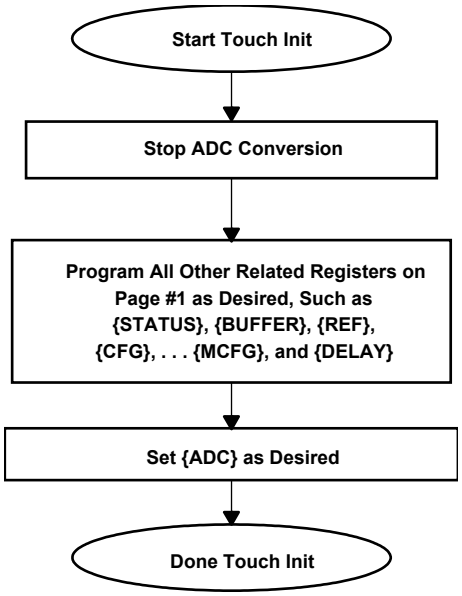


Figure 1. TSC2101 Touch-Screen Driver Initialization - *InitTSC2101Touch( )*

In Figure 1, a TSC2101 register is denoted by {register name}. For example, the *ADC* register (at page 1 and address 0x00) is denoted by {ADC}; the configuration register (at page 1 and address 0x05) is by {CFG}.

The following code segment is an example of TSC2101 touch-screen function initialization.

```
TSC2101WriteReg( {ADC},                0xC4FF);     // Stop ADC
TSC2101WriteReg( {STATUS},             0x4000 );    // Set /DAV interrupt
TSC2101WriteReg( {REF},                0x001C);     // Set Internal Reference
TSC2101WriteReg( {ADC},                0x8477);     // Set to TSC & XY mode
```

Other registers on page 1 remain at their power up defaults, and therefore do not need initialization.

Reading Touch Data

In the preceding initialization example, the TSC2101 is set to the touch-screen controlled mode (not the host-controlled mode); the /PINTDAC pin is set as the /DAV hardware interrupt (not the PENIRQ). In this condition, the /DAV interrupt asserts when the screen has been touched and the touch data has been sampled, converted, and ready to be read. At this point, the driver reads all the touch data, resetting the /DAV interrupt and readying the system for the next data acquisition.

To read data from the TSC2101 touch-data registers, the routine *ReadTouchScreenTSC2101(*X, *Y)* is called in the DDSI function *DdsiTouchPanelGetPoint()*.

The touch initialization and XY data-reading code is found in the file *TSC2101Touch.CPP* in the *DdsiTouchPanelEnable( )* and *DdsiTouchPanelGetPoint()* routines, respectively.

TSC2101 Battery Driver

The battery driver reads the battery level and reports this information to the OS. If this driver is used, it must share the analog-to-digital converter (ADC) with the touch-screen interface; so, care is taken so that battery readings do not occur while processing touch-screen inputs. This is done with an additional *mutex object*[1] that controls access to the ADC function of the TSC2101. The touch-screen driver owns this mutex as long as the pen is down. When the pen is not down, the battery driver can own the mutex, sample the battery, and release the mutex back to the touch-screen driver in case it needs the ADC.

The *BatteryDrvrGetStatus()* routine in the battery driver returns the battery status. The example TSC2101 driver only monitors the backup battery, but it can be easily modified to monitor the main battery by changing the parts of the status structure that are updated.

TSC2101 Audio Driver

Several standard WinCE audio-driver templates are available. This application report considers two of them, WaveDev and WaveDev2.

Functionally, the main difference between the WaveDev and WaveDev2 audio drivers is that WaveDev2 allows playback of multiple streams of sound simultaneously. The WaveDev driver only allows one sound to be played at a time.

While the WaveDev driver is a formal layered driver model, the WaveDev2 driver follows the new Unified Audio Model (UAM). This model moves all the code into the driver, and adds DirectSound and better streaming support.

The WaveDev driver consists of the file *TSC2101Audio.C*, together with the header file TSC2101Audio.H and the PDL files *XXXXXAudio.C* and *XXXXXAudio.H*.

[1] "A mutex object is a synchronization object whose state is set to *signaled* when it is not owned by any thread, and *nonsignaled* when it is owned. Only one thread at a time can own a mutex object, whose name comes from the fact that it is useful in coordinating mutually exclusive access to a shared resource. For example, to prevent two threads from writing to shared memory at the same time, each thread waits for ownership of a mutex object before executing the code that accesses the memory. After writing to the shared memory, the thread releases the mutex object." – From *http://msdn.microsoft.com/*

The WaveDev2 driver consists of the file *hwctxt.cpp* together with the header files *hwctxt.h* and *Vhwctxt.h* and the PDL files *XXXXXctxt.C* and *XXXXXctxt.H*.

The TSC2101 WaveDev audio driver was developed within the PDD layer of the OS, with no changes to the upper layers of the audio architecture. See the application report *TSC2301 WinCE Generic Drivers* (SLAA187) or Microsoft WinCE audio driver documentation for more details on standard audio PDD functions.

The PDD layer, via the SPI interface, controls the TSC2101 audio functions by writing to the page 2 control registers of the TSC2101. Data flow between the PDD layer and the I^2S bus is handled via direct memory access (DMA).

The TSC2101 WaveDev2 audio driver was developed within the new UAM with no changes to the upper layers of the audio architecture. In this model, the upper layers are compiled with the hardware-dependent layers; so it is not a formal layered driver. However, the hardware-dependent code is contained in separate cpp files for ease of porting between platforms. See the Microsoft WinCE audio-driver documentation for more details on the UAM and differences between the UAM and layered audio drivers.

Audio Initialization

Audio initialization has three main tasks: (1) set up SPI interface; (2) set up TSC2101 audio control registers; and (3) set up the DMAC structure for audio data (I^2S) transformations. The three tasks are done at the audio DDSI routine *PDD_AudioInitialize( )* under the WaveDev template and at *VHardwareContext::Init()* under the WaveDev2 template.

The TSC2101 audio control registers are initialized in the InitTSC2101Audio( ) subroutine. The following code fragment is an example initialization of the TSC2101 audio function.

```
TSC2101WriteReg( { AUDCTL1 },          0x0000 );
TSC2101WriteReg( { HEDVOL },           0x8000 );
TSC2101WriteReg( { DACVOL },           0x8080 );
TSC2101WriteReg( { MIXER },            0xC530 );
TSC2101WriteReg( { AUDCTL2},           0x44F0 );
TSC2101WriteReg( { AUDCTL3},           0x2000 );
TSC2101WriteReg( { PLL1},              0x1120 );
TSC2101WriteReg( { PLL2},              0x0000 );
TSC2101WriteReg( { AUDCTL4},           0x1800 );
TSC2101WriteReg( { HNDVOL },           0x8000 );
TSC2101WriteReg( { CELLVOL },          0xC57C );
TSC2101WriteReg( { AUDCTL5},           0x2100 );
TSC2101WriteReg( { AUDCTL6},           0x00C0 );
TSC2101WriteReg( { AUDCTL7},           0x0000 );
TSC2101WriteReg( { GPIO},              0x0000 );
TSC2101WriteReg( { CELLAGC},           0x0000 );
TSC2101WriteReg( { DRVPD},             0x0200 );
TSC2101WriteReg( { MICAGC},            0xFE00 );
TSC2101WriteReg( { CELLAGC2},          0xFE00 );
TSC2101WriteReg( { AUDPD},             0xFFFC );
```

Audio Data Transformation

In the audio driver, the DMAC function of the processor is used to move audio data over the I^2S bus between the processor and the TSC2101. The processor DMA function is initialized, as previously stated, in the audio initialization PDD routine *PDD_AudioInitialize( )*.

Also, another PDD routine *PDD_AudioGetInterruptType( )* determines the source of an audio interrupt and then tells the MDD layer the current audio status: input or output play, input or output record, stopped, or other status.

Audio Messages

The two audio message-sending PDD routines in the WaveDev architecture are:— *PDD_WavProc()* sends standard audio control messages from the MDD layer to the PDD layer and *PDD_AudioMessage()* sends custom messages from user applications to the PDD layer. The later can be used to modify or update the TSC2101 control registers and can be accessed by a user application.

In the WaveDev2 architecture, custom messages are passed by the *wavemain.cpp* function *WAV_IOControl()* through *HandleWaveMessage()* to *VHardwareContext::AudioMessage()* in *hwctxt.cpp*. These custom messages can be generated with the *waveOutMessage()* API function by a user application.

Driver Summary

Table 3 summarizes the TSC2101 driver files and the processor hardware requirements to support the corresponding TSC2101 drivers.

Table 3. TSC2101 WinCE Drivers and Processor Requirements

Function	Touch Screen	Audio	Battery
SW Driver Required	Touch Driver TSC2101Touch.CPP XXXXXTouch.CPP XXXXXTouch.H SPI Driver TSC2101SPI.C TSC2101SPI.H XXXXXSPIComm.C XXXXXSPIComm.H TSC2101REG.H	Audio Driver (WAVEDEV) TSC2101Audio.C TSC2101Audio.H XXXXXAudio.C XXXXXAudio.H Or (WAVEDEV2) hwctxt.CPP hwctxt.H Vhwctxt.H XXXXXctxt.CPP XXXXXctxt.H SPI Driver TSC2101SPI.C TSC2101SPI.H XXXXXSPIComm.C XXXXXSPIComm.H TSC2101REG.H	Battery Driver Battery.c SPI Driver TSC2101SPI.C TSC2101SPI.H XXXXXSPIComm.C XXXXXSPIComm.H TSC2101REG.H
Processor HW Required	External HW Interrupt for /DAV SPI Port	DMA I2S Port SPI Port	SPI Port Analog connection to battery voltage

Installation

These instructions are for WinCE, where $(BASE) is the BSP base directory. For example, the Intel DBPXA250 BSP base directory is Platform\XSC1BD\ under the WinCE410 or the WINCE420 directory.

The instructions for installing the WAVEDEV2 driver assume that a wavedev2 driver is already in the platform in use. If not, copy a generic wavedev2 driver into the build before adding the TSC2101 driver. A typical wavedev2 driver has the following additional files.

```
$(BASE)\DRIVERS\devctxt.cpp
$(BASE)\DRIVERS\devctxt.h
$(BASE)\DRIVERS\input.cpp
$(BASE)\DRIVERS\midinotecpp
$(BASE)\DRIVERS\midistrm.cpp
$(BASE)\DRIVERS\midistrm.h
$(BASE)\DRIVERS\output.cpp
$(BASE)\DRIVERS\strmctxt.cpp
$(BASE)\DRIVERS\strmctxt.h
$(BASE)\DRIVERS\wavemain.cpp
$(BASE)\DRIVERS\wavemain.h
$(BASE)\DRIVERS\wavepdd.h
```

Copy the following files into the Intel BSP.

```
$(BASE)\TSC2101\INC\TSC2101Regs.h
$(BASE)\TSC2101\INC\TSC2101SPI.h
$(BASE)\TSC2101\INC\XscaleSSPComm.h (or BulverdeSSPComm.h)
$(BASE)\TSC2101\TSCLib\makefile
$(BASE)\TSC2101\TSCLib\sources (or sources-bulverde rename to sources)
$(BASE)\TSC2101\TSCLib\TSC2101SPI.c
$(BASE)\TSC2101\TSCLib\XscaleSSPComm.c (or BulverdeSSPComm.c)
$(BASE)\TSC2101\TSCTouch\makefile
$(BASE)\TSC2101\TSCTouch\sources (or sources-bulverde renamed sources)
$(BASE)\TSC2101\TSCTouch\TSC2101Touch.cpp
$(BASE)\TSC2101\TSCTouch\XscaleTouch.cpp (or BulverdeXscaleTouch.cpp)
$(BASE)\TSC2101\TSCTouch\XscaleTouch.h (or BulverdeXscaleTouch.h)
$(BASE)\TSC2101\dirs
$(BASE)\GWE\Battery\battery.c (replace existing file)
```

Copy the following files to use the WaveDev2 Template.

```
$(BASE)\DRIVERS\WAVEDEV\hwctxt.cpp (replace existing file)
$(BASE)\DRIVERS\WAVEDEV\hwctxt.h (replace existing file)
$(BASE)\DRIVERS\WAVEDEV\Vhwctxt.h
$(BASE)\DRIVERS\WAVEDEV\ XScalectxt.cpp (or BVDctxt.cpp)
$(BASE)\DRIVERS\WAVEDEV\ XScalectxt.h (or BVDctxt.h)
```

Copy the following files to use the WaveDev Template.

```
$(BASE)\TSC2101\TSCWAVEDEV\makefile (replace existing file)
$(BASE)\TSC2101\TSCWAVEDEV\sources
$(BASE)\TSC2101\TSCWAVEDEV\TSC2101Audio.c
$(BASE)\TSC2101\TSCWAVEDEV\TSC2101Audio.h
$(BASE)\TSC2101\TSCWAVEDEV\XScaleAudio.c
$(BASE)\TSC2101\TSCWAVEDEV\XscaleAudio.h
```

Modify the following Intel BSP files according to the instructions given in subsequent sections.

```
$(BASE)\Kernel\HAL\cfwxsc1.c
$(BASE)\Kernel\HAL\ARM\intxsc1.c
$(BASE)\dirs
$(BASE)\DRIVERS\WAVEDEV\wavemain.cpp
$(BASE)\DRIVERS\WAVEDEV\sources
```

1. $(BASE)\Kernel\HAL\cfwxsc1.c

In the function OEMInterruptEnable() under the SYSINTR_TOUCH case, replace the exising code 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 the exising code 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 the exising code with the following:

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

In the function OEMInterruptDone () under the SYSINTR_TOUCH case, replace the exising code 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 the exising code with the following:

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

2. $(BASE)\Kernel\HAL\ARM\intxsc1.c

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

```
// 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;
        return SYSINTR_TOUCH;
}
```

3. $(BASE)\dirs

In the DIRS= section add a `tsc2101` entry above the `gwe` entry.

4. $(BASE)\DRIVERS\WAVEDEV\wavemain.cpp

In the function HandleWaveMessage, change the default case in the switch statement from:

```
case WIDM_UNPREPARE:
    default:
    dwRet  = MMSYSERR_NOTSUPPORTED;
```

To:

```
case WIDM_UNPREPARE:
    dwRet  = MMSYSERR_NOTSUPPORTED;
    break;

default:
    dwRet = g_pHWContext->AudioMessage(uMsg, dwParam1, dwParam2);
    break;
```

5. $(BASE)\DRIVERS\WAVEDEV\sources

Add `$(_TARGETPLATROOT)\lib\$(_CPUINDPATH)\tsclib.lib` to the TARGETLIBS line.

Add `;..\..\TSC2101\INC` to the INCLUDES line.

If using XScale (Lubbock), use the XScalectxt.* files, make sure hwctxt.h includes XScalectxt.h instead of BVDctxt.h, and add `XScalectxt.cpp` to the SOURCES section of sources.

If using Bulverde (MainStone II), use the BVDctxt.* files, make sure hwctxt.h includes BVDctxt.h instead of XScalectxt.h, and add `BVDctxt.cpp` to the SOURCES section of sources.

References

TSC2101 Audio CODEC with Integrated Headphone, Speaker Amplifier and Touch Screen Controller, (SLAS392A)

TSC2301 WinCE Generic Drivers (SLAA187)

Windows CE .Net Touch Screen, Keypad, and Audio Device Driver for the TSC2301, (SLAA169)

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.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
		Telephony	www.ti.com/telephony
		Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

Mailing Address: Texas Instruments
Post Office Box 655303 Dallas, Texas 75265