Application Report
SLAA187 - December 2003

TSC2301 WinCE Generic Drivers

Yannick Chammings—WinCE Consultant, ADESET Inc., France (ychammings@adeset.com)
Wendy X. Fang —Application Engineer, DAP Group, Texas Instruments, Dallas, TX, USA

ABSTRACT

This application report presents the basic concept and methodology to develop the TSC2301 touch screen, keypad, audio, and GPIO drivers on the Windows CE (WinCE) operating system (OS). The drivers are not tied to any specific processor platform. They therefore can be called *generic* and are reusable for any host processor or platform with minimal additional coding.

Contents

List of Figures

List of Tables

1 Introduction

Texas Instruments (TI) produces a series of touch screen control devices that have been widely used in the wireless and communications industry. The applications include, but are not limited to, cellular phones, smart phones, MP3 players, internet appliances, and personal digital assistants (PDA). On the product tree of the following WEB site, a broad range of touch screen controllers can be found.

http://focus.ti.com/analog/docs/analogprodhome.tsp?templateId=4&familyId=82

The TSC2301 is one touch screen device from TI that integrates a touch screen controller, a keypad controller, and a full-duplex stereo audio codec in a single chip. Additionally, six GPIO pins on the silicon can be used as extra keypad inputs, power button, and/or other digital IO functions.

To apply the TSC2301 device, it is essential to develop and write the software drivers to control the touch screen, the keypad, the audio, and/or other functions. Driver development consumes time and requires in-depth knowledge on the TSC2301 device. This application report provides TSC2301 users with generic and reusable TSC2301 drivers, reducing software development time.

Different operating systems (OS) and host processor/microprocessor platforms require different drivers for the TSC2301. Hence, it is impractical to develop TSC2301 drivers that can be directly ported and used by all OS and processors. This application report uses the WinCE OS platform and *generic* TSC2301 drivers that are modified and restructured from the *nongeneric* drivers, reference [2]. The processor-related/dependent code, and processor-related routines were separated from the other parts of the drivers. The processor-dependent routines are *called* by the TSC2301 driver routines. Therefore, changes to the main drivers are not needed when changing from one host processor to another. As a result, only the code that is related to the particular processor needs to be added, or modified when a particular processor is applied. Please refer to separate application reports, references [4] and [5], for examples. For consultation and service regarding integrating and interfacing this or other TI TSC drivers to other processors, contact the following TI third-party contractor:

http://www.adeset.com

or search at

http://www.ti.com and click on the *Analog and Mixed Signal* link under the *Products* banner, then click on the *3rd Party Developers Network* link under the *Support* banner.

2 Principles

Since the TSC2301 implements three types of devices (touchscreen, keypad, and audio) a driver is required for each feature.

In the Windows CE device drivers model, implement the drivers as follows:

- A touchscreen driver for the touchscreen feature
- A keyboard driver for the keypad feature
- An audio driver (WaveDev or WavClick) for the audio feature

The Windows CE device drivers model takes into consideration three types of drivers:

- Built−in drivers, also known as native device drivers. These drivers implement a custom interface (known as the device driver interface, or DDI functions) specific to the feature it implements. They are loaded by the graphical windowing events subsystem (GWES) process, which is part of the Windows CE kernel. They are either dynamic library linked (DLL) or libraries directly linked to the GWES. GWES manages all MMI subsystems (graphical, user inputs, etc.). The typical built-in drivers are:
  - Display driver
  - Touchscreen driver
  - Keyboard driver
  - Mouse Driver
  - LEDs drivers
- Installable drivers, also known as stream drivers. These drivers implement a standard IO interface to be used for all types of IO devices. These drivers are mounted by device manager either statically or dynamically. They are DLLs which must provide a ten-function interface (Init, Deinit, Open, Close, Read, Write, Seek, IOControl, PowerUp, PowerDown) and are configured in the registry. Consequently, a new stream driver can be easily added. Examples of typical stream drivers are:
  - Serial drivers
  - Bus drivers
  - Block drivers (for storage devices)
  - Specific stream drivers
- The hybrid drivers are stream drivers that additionally implement a specific interface dedicated to their feature. They are mounted by device manager stream drivers, but are used by other processes through their specific interfaces. Some sample hybrid drivers are:
  - Audio drivers
  - PCMCIA controller drivers
  - USB drivers

Beyond this classification, the Windows CE device driver model offers two types of architecture:

- Layered drivers (split into two layers):
  - The model device driver (MDD) layer implements the functional treatment of the drivers that are independent of the device that is managed by the driver. For standard drivers in most cases, the MDD layer need not be modified when developing a driver for a new device.
  - The platform dependent device (PDD) layer implements treatments that are specific to the device being managed. This layer is adapted when a driver is developed for a new device.

    For all standard drivers, the Windows CE device driver model defines the interface between the MDD and the PDD layers, allowing development of the PDD layer without any information on the behavior of the MDD. The PDD layer interface is standardized, but specific for each type of driver. It is called the device driver service provider interface (DDSI) function.

- Monolithic drivers: Monolithic drivers meld the MDD and PDD layers into one layer. The functional and device-dependent codes are melded in the same module. These are mainly used for the simple driver that requires no independent functional layer, or specific drivers for which the reuse of an independent functional layer is not pertinent.

The functional diagram (Figure 1) summarizes Windows CE device drivers model:

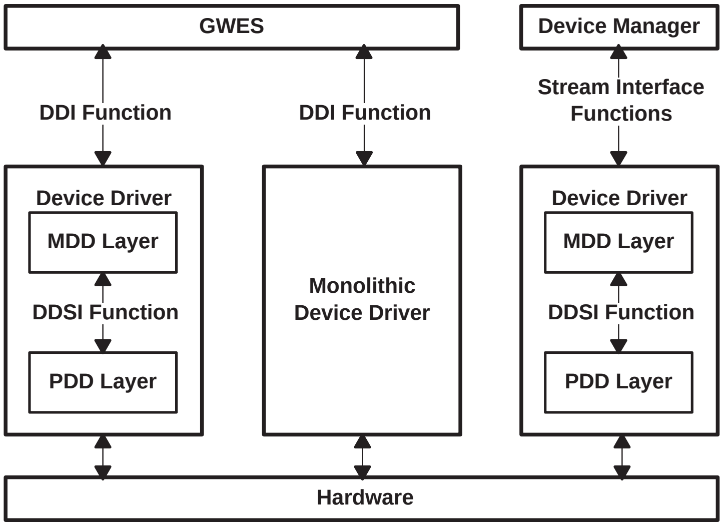

Figure 1. WinCE Device Driver Model [3]

According to the model described above, developing Windows CE drivers for the TSC2301 implies:

- Development of a common library that implements all the SPI exchange with the TSC2301
- Development of the PDD layer of a touchscreen driver (native driver mounted by GWES)
- Development of the PDD layer of a keyboard driver (native driver mounted by GWES)
- Development of the PDD layer of an audio driver (hybrid driver mounted by device manager)
- Development of a monolithic stream driver for power button management (stream driver mounted by device manager). This one is monolithic, since the functional treatments are basic enough to allow the melding of MDD and PDD layer.

To develop the TSC2301 drivers, this application report describes an additional *lower* layer from the device driver. This layer is called the processor-dependent layer (PDL). As the result, a monolithic TSC2301 driver has the structure shown in Figure 2−a; and a layered one becomes that shown in Figure 2−b.

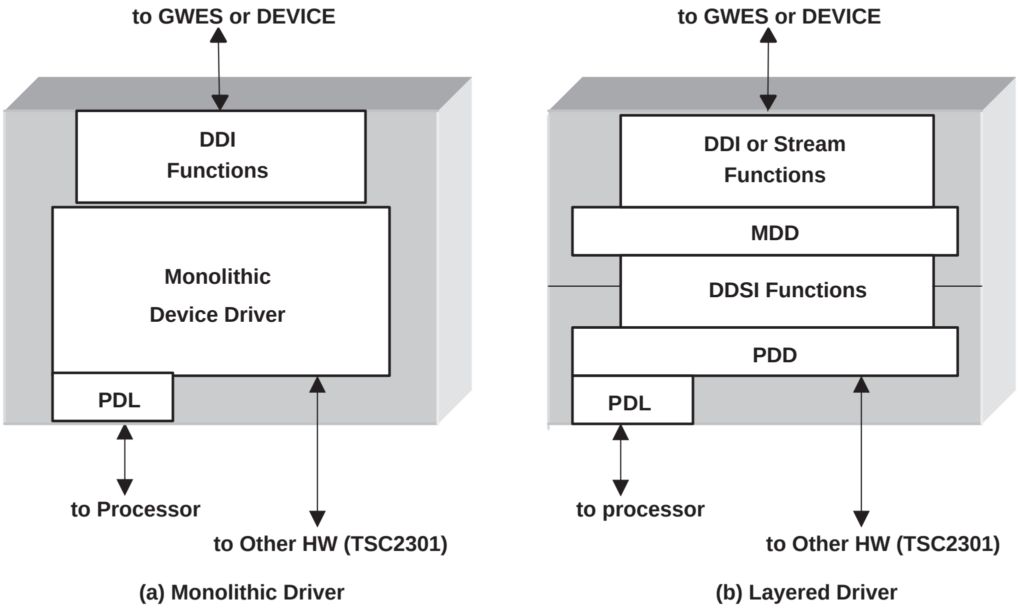

Figure 2. Monolithic and Layered TSC2301 Driver

This additional layer extends the standard Windows CE device driver model to offer the more generic and flexible drivers:

- The MDD layer implements the functional layer that is both device-independent (i.e., independent from the device managed—here the TSC2301) and platform-independent (i.e., independent from the hardware architecture in which the device is used, such as the Lubbock platform (XScale based) and the NOAH platform (AT91RM9200 based)).
- The PDD layer implements device management. It is device-dependent (it implements the TSC2301 specific code) but platform-independent (the code that implements how the driver exchanges data with the processor is not at this layer).
- The PDL layer implements platform-specific code. It is platform-dependent as it implements how the data are exchanged with the processor.

The purpose of the foregoing driver model is to ease the adaptation of the drivers to various platforms that interface with the TSC2301. Adapting the drivers to a new platform requires only modifications in the PDL layer, a sublayer of the PDD layer.

3 TSC2301 Registers

A series of registers inside the TSC2301 containing the device data, status, and all programmable controls, variables and parameters, are accessible or controllable by the host processor by SPI interface. Three pages of memory space hold the registers. Refer to Table 1 for a view of the registers and to the TSC2301 data sheet [1] for the register definitions and memory map. The registers in page #0 are data registers; those in page #1 are the touch screen and keypad control registers; and those in page #2 are audio and GPIO control registers.

Table 1. TSC2301 Registers

Page #0: (Page Address = 0000b†) DATA REGISTERS			**Page #1: (Page Address = 0001b†) CONTROL REGISTERS**			**Page #2: (Page Address = 0010b†) AUDIO CONTROL REGISTERS**		
COMMAND WORD‡	**REGISTER ADDRESS**	**REGISTER NAME**	**COMMAND WORD‡**	**REGISTER ADDRESS**	**REGISTER NAME**	**COMMAND WORD‡**	**REGISTER ADDRESS**	**REGISTER NAME**
0000h§	00h	X	0800h	00h	ADC	1000h	00h	AUDCTRL
0020h	01h	Y	0820h	01h	KEY	1020h	01h	ADCVOL
0040h	02h	Z1	0840h	02h	DACCTL	1040h	02h	DACVOL
0060h	03h	Z2	0860h	03h	REF	1060h	03h	BPVOL
0080h	04h	KPDATA	0880h	04h	RESET	1080h	04h	KEYCLICK
00A0h	05h	BAT1	08A0h	05h	CFG	10A0h	05h	AUDPD
00C0h	06h	BAT2	08C0h	06h	CFG2	10C0h	06h	GPIO
00E0h	07h	AUX1		07~ 0Fh	Reserved	10E0h ~ 1340h	07h ~ 1Ah	DAC Bass-Boost Filter
0100h	08h	AUX2	0A00h	10h	KPMASK			
0120h	09h	TEMP1						
0140h	0Ah	TEMP2						
0160h	0Bh	DACDATA						

† The b here indicates a binary number

‡ The 16-bit words here are the command words when WRITE; and the command words when READ are the same except that the MSB bit 15 = 1b, that is: Command-Word | 8000h (bit−wise OR).

§ The h here indicates a hex number.

The TSC2301 registers are defined and coded in the header file ***TSC2301Regs.H***.

4 SPI Interface

The processor controls and accesses the TSC2301 control registers described in the previous section through a standard SPI interface. Therefore, the 4-wire synchronous serial or SPI bus is the most important interface. The SPI driver is the core for the TSC2301 drivers of the touch screen, the keypad, the audio function, and any other functions.

The SPI standard and protocol are beyond the scope of this application report and are not discussed in detail; only the interface to TSC2301 is addressed here.

4.1 Processor HW Requirement

Use the built-in standard SPI port on the host processor, if one is available. Otherwise, with any four GPIO pins on the processor and a little extra coding, an SPI bus and interface can be created.

4.2 Processor/TSC2301 HW Connection

A standard SPI bus consists of four pins/lines, normally named as: SCK (clock), SS (slave select), MOSI (master-out slave-in data), and MISO (master-in slave-out data). Table 2 lists the SPI connections between the processor and the TSC2301.

Table 2. Host Processor and TSC2301 SPI HW Interface

	HOST PROCESSOR PIN NAME	TSC2301 PIN NAME
SPI Clock	SCK	SCLK (VFBGA BALL B1 or TQFP Pin17)
SPI Slave Select	$\overline{\text{SS}}$	$\overline{\text{SS}}$ (VFBGA BALL B2 or TQFP Pin16)
SPI MOSI Data	MOSI	MOSI (VFBGA BALL C2 or TQFP Pin18)
SPI MISO Data	MISO	MISO (VFBGA BALL C1 or TQFP Pin19)

In the SPI interface, the TSC2301 is always the slave device; the host processor is the master of the interface.

4.3 SPI Driver

The TSC2301 SPI driver contains four files: ***TSC2301SPI.C***, ***TSC2301SPI.H***, ***XXXXXSPIComm.****C, and* ***XXXXXSPIComm.H***, where *XXXXX* stands for the name of the utilized processor. Therefore, the two files are processor-dependent and in the PDL. For example, for an XScale processor, the two processor-dependent files should be ***XScaleSPIComm.C*** and ***XScaleSPIComm.H***; and for an AT91RM processor, the two files could be *AT91SPIComm.C* and *AT91SPIComm.H*.

Among many of the routines in the above four files, the fundamental routines for the SPI driver are summarized in Table 3. For the write and read operation timing to interface to TSC2301, refer to Figure 50 in the TSC2301 data sheet [1]. The processor-related or PDL routines, listed in Table 3, have *HW* as the first two letters of the name. Using different processors requires changing the routines in the PDL layer.

Table 3. Fundamental Routines for SPI Driver

ROUTINE NAME	INVOLVED PROCESSOR-DEPENDENT ROUTINES	FUNCTION

InitializeSPIDriver( );
SetupSPIController( )
	HWInitializeSPIDriver( );
	HWSetupSPIController( )
		Sets up the host processor SPI port so as to be ready for interface

DeIntializeSPIDriver( );
StopSPIController( )
	HWDeIntializeSPIDriver( );
	HWStopSPIController( )
		Stop the host processor SPI port and interface

SPITransaction( )		Write/read one or more TSC2301 registers
	HWStartFrame( )	Actives $\overline{\text{SS}}$ (goes low)
	HWStopFrame( )	De−actives $\overline{\text{SS}}$ (goes high)
	HWSPIWriteWord( )	Writes a word to MOSI
	HWSPIReadWord( )	Reads a word from MISO
	HWSPITxBusy( )	Checks for SPI transmitting finished
	HWSPIRxBusy()	Checks for SPI receiving finished
	HWSPIFIFONotEmpty( )	Ensures the SPI FIFO has been properly read so that the read data are the latest

All other SPI interface routines are built from the above basic routines for *write* and *read* operations, to/from TSC2301 registers. Table 4 describes the driver layers.

Table 4. TSC2301 SPI Driver Layers

Layer	Routine Name

Top (1st) Layer: Routines used by other OS routines/drivers	***InitializeSPIDriver( );***
	DeIntializeSPIDriver( );
	SetupSPIController( );
	StopSPIController( );
	And all theTSC2301-register write/read routines, TSC2301-driver function initialization routines, such as:
	TSC2301WriteADCReg( ), …, …, or ***TSC2301ReadXY( )***

Middle (2nd) Layer: Routines called by routines in the top layer	***WordSPITransaction( );***

Bottom (3rd) Layer (PDL): Processor-dependent routines, called by the top and 2nd layer routines	***HWInitializeSPIDriver( );***
	HWDeIntializeSPIDriver( );
	HWSetupSPIController( ); HWStopSPIController( );
	HWStartFrame( );HWStopFrame( )
	HWSPIWriteWord( ); HWSPIReadWord( )
	HWSPITxBusy( ); HWSPIRxBusy( );
	HWSPIFIFONotEmpty( )

5 TSC2301 Touch Screen Driver

The TSC2301 touch screen driver is in the file ***TSC2301Touch.CPP****, together with the PDL files* ***XXXXXTouch.CPP*** and ***XXXXXTouch.H,*** where *XXXXX* stands for the name of the utilized processor.

The touch screen driver can be classified as *built-in* (vs. stream) and *layered* (vs. monolithic). The TSC2301 driver was developed based on the generic touch panel structure given by WinCE OS. The DDSI functions in Table 5 were developed for driving TSC2301 touch function.

Table 5. TSC2301 Touch Screen Driver DDSI Routine List

FUNCTION	DESCRIPTION
DdsiTouchPanelAttach( )	Called when the MDD’s DLL entry point gets a DLL_PROCESS_ATTACH message.
DdsiTouchPanelDetach( )	Called when the MDD’s DLL entry point gets a DLL_PROCESS_DETACH message.
DdsiTouchPanelDisable( )	Disables the touch screen device.
DdsiTouchPanelEnable( )	Applies power to the touch screen device and initializes it for operation.
DdsiTouchPanelGetDeviceCaps( )	Queries for capabilities of the touch screen device.
DdsiTouchPanelGetPoint( )	Returns the most recently acquired point and its associated tip-state information.
DdsiTouchPanelPowerHandler( )	Indicates to the driver that the system is entering or leaving the suspend state.
DdsiTouchPanelSetMode( )	Sets information about the touch screen device.

The important functions that control the TSC2301 touch screen function are the two DDSI routines, ***DdsiTouchPanelEnable( )*** and ***DdsiTouchPanelGetPoint( ).***

To setup the TSC2301 touch screen function, the subroutine ***InitTSC2301Touch( )*** is called in the touch DDSI routine ***DdsiTouchPanelEnable( )***. It is crucial to closely follow the initialization sequence implemented in this stage. Figure 3 illustrates the TSC2301 touch function initialization, where the corresponding TSC2301 control register is indicated in { }. For example: the ADC register is shown as {ADC}.

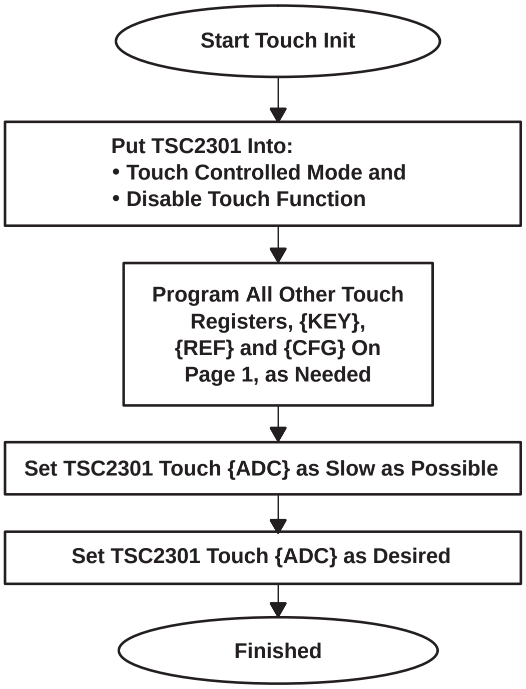
Start Touch Init
Put TSC2301 Into:
• Touch Controlled Mode and
• Disable Touch Function
Program All Other Touch Registers, {KEY}, {REF} and {CFG} On Page 1, as Needed
Set TSC2301 Touch {ADC} as Slow as Possible
Set TSC2301 Touch {ADC} as Desired
Finished

Figure 3. TSC2301 Touch Screen Driver Initialization—*InitTSC2301Touch()*

Refer to the attached code for the example settings of the TSC2301 touch screen registers. After the above initialization, the touch function is ready, and the touch data (for example, X and Y) are available as soon as the screen/panel is touched (touch screen mode) or when required by the host processor (host mode).

To read the touch screen data from the TSC2301 touch data registers, the routine ***SampleTouchScreenTSC2301(*X, *Y)*** is called in the DDSI function ***DdsiTouchPanelGetPoint***. This application report uses the TSC2301 as its touch screen control (vs. host control) mode. Refer to the initialization code attached. The $\overline{\text{DAV}}$ (not the PENIRQ) is utilized as the hardware interrupt for the driver. The $\overline{\text{DAV}}$ interrupt occurs (going *low*) whenever the screen/panel has been touched, and the touch data has been sampled, converted, and ready to be read.

At this point, the driver reads back all of the touch data so as to reset the $\overline{\text{DAV}}$ interrupt. Note that under this mode, the only way to reset the $\overline{\text{DAV}}$ is to properly read all touch screen data. Due to a silicon bug, all touch data, including the X, Y, Z1, and Z2, have to be read back no matter whether the data is of interest/needed or not. For example, even if only X and Y were required to be read from the TSC2301, the reading procedure in the touch driver must read back all of the X, Y, Z1, and Z2. Otherwise, the $\overline{\text{DAV}}$ may not be able to return to a logic *high*. Such a touch screen reading routine, named as ***TSC2301ReadXY (*X, *Y)***. can be found in the SPI driver routine, ***TSC2301SPI.C***. Refer to the previous section on SPI.

6 TSC2301 Keypad Driver

The TSC2301 keypad driver is in the file ***TSC2301Key.CPP***, together with its header file ***TSC2301Key.HPP***, and the PDL files ***XXXXXKey.CPP*** and ***XXXXXKey.H***.

The keypad driver can also be classified as *built-in* and *layered*, similar to the TSC2301 touch screen driver, and was developed based on the common keyboard structure given by WinCE OS. Table 6 lists the related driver routines on the PDD layer.

Table 6. TSC2301 Keypad Driver Routine List

FUNCTION	DESCRIPTION
KeybdPdd_InitializeDriverEx( )	Called by upper layer to initialize the keyboard hardware, including the processor and the TSC2301
KeybdPdd_GetEventEx( )	Called by upper layer to retrieve keyboard (keypad) events after the SYSINTR_KEYBD signal is sent.
KeybdPdd_PowerHandler( )	Called by the upper layer on system power state changes

The main functions for the TSC2301 keypad driver are in the two keyboard PDD routines, named ***KeybdPdd_InitializeDriverEx( )*** and ***KeybdPdd_GetEventEx( )***. The TSC2301 keypad control registers were setup or initialized in the ***KeybdPdd_InitializeDriverEx( )***; the keypad data is read in the routine ***KeybdPdd_GetEventEx( )***.

The initialization of the TSC2301 keypad is implemented by calling the subroutine ***InitTSC2301Key( )***, which is called at the PDD layer ***KeybdPdd_InitializeDriverEx( )***. The keypad initialization involves only two SPI writings to TSC2301 registers **Key** and **KeyMask**, as shown in Figure 4.

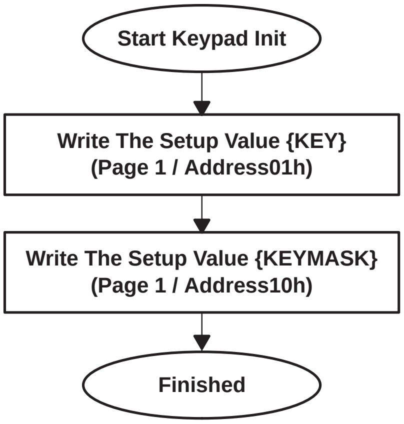

Figure 4. TSC2301 Keypad Driver Initialization—*InitTSC2301Key()*

Compared to the initialization the TSC2301 touch screen function, the TSC2301 Keypad initialization is simple and straightforward.

After the initialization, the keypad function is ready for application. At this point, whenever one or more keys in the keypad are pressed, the KEYIRQ interrupt is active after the keypad data has been sampled and converted, and the data is ready to be read. After the keypad data is read from KPDATA (in TSC2301 page#0, register # 04h), the KEYIRQ interrupt is reset or cleared (back to logic high) and the TSC2301 keypad function is ready for the next cycle. Refer to the routine ***KeybdPdd_GetEventEx**( )* for the coding details.

In this application report, note that three among the six GPIO pins in the TSC2301 are used as three extra keys of the 16-key keypad. To do so, the key *F* (last key on the 16-key pad) is programmed as a function key. If the key *F* is pressed, GPIO1 to GPIO3 become the extended keys on the keypad. Otherwise, they are normal GPIO pins.

7 TSC2301 Audio Driver

The TSC2301 audio driver is in the file ***TSC2301Audio.C***, together with the header file ***TSC2301Audio.H***, and the PDL files ***XXXXXAudio.C*** and ***XXXXXAudio.H.***

As stated previously, the audio driver is classified as the hybrid driver, with the same characteristics as the stream driver, and the specific interface dedicated to the audio functions. Figure 5 shows the interaction of the audio driver using the MDD library in WinCE OS.

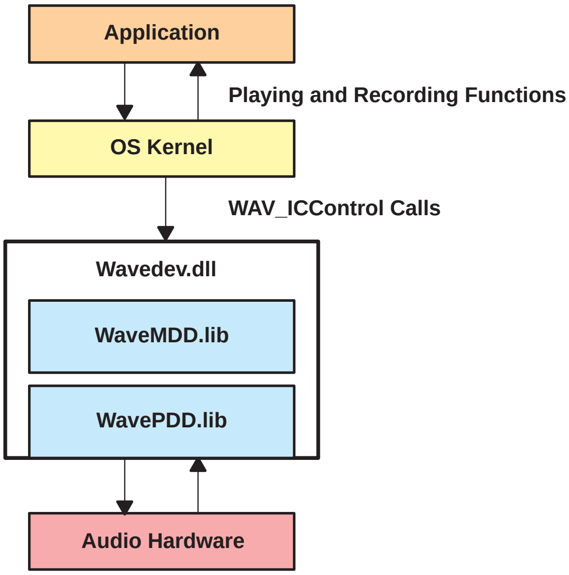

Figure 5. Audio Driver Using MDD Library [3]

The TSC2301 audio driver was developed within the PDD layer of the OS with no changes to the other layers of the audio architecture shown in Figure 5. The audio driver DDSI routines are listed in Table 7.

Table 7. TSC2301 Audio Driver PDD Routine List

FUNCTION	DESCRIPTION
PDD_AudioGetInterruptType( )	This function determines the cause of the audio interrupt and returns the current device status.
PDD_AudioInitialize()	This function initializes the audio device for operation.
PDD_AudioDeinitialize ()	This function turns off and disconnects the audio device.
PDD_AudioPowerHandler()	This function is responsible for managing the audio hardware during POWER_UP and POWER_DOWN notifications.
PDD_AudioMessage()	This function sends messages from user applications to the audio driver’s platform−dependent driver (PDD) layer.
PDD_WaveProc()	This function sends messages to the audio driver’s PDD layer.

The initialization of the TSC2301 audio function is performed through the ***InitTSC2301Audio( )*** subroutine, which is called at the audio DDSI routine ***PDD_AudioInitialize( )***. Figure 6 illustrates the audio initialization flowchart. Note that it is recommended not to power up the audio by software until all other audio control registers have been configured or initialized.

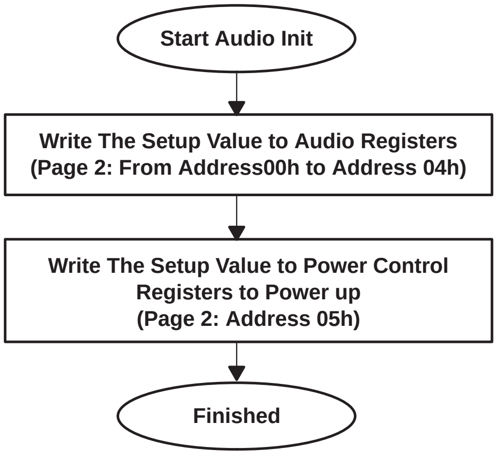
Start Audio Init
Write The Setup Value to Audio Registers
(Page 2: From Address00h to Address 04h)
Write The Setup Value to Power Control
Registers to Power up
(Page 2: Address 05h)
Finished

Figure 6. TSC2301 Audio Driver—*InitTSC2301Audio()*

The settings and mappings for the DMAC function are called in ***PDD_AudioInitialize( )*** so as to handle the audio wave stream data. See the DDSI routine for the details.

The majority of the application message handlings for the PDD is in the ***PDD_WaveProc( )*** which processes the WinCE OS standard audio driver's WPDM_xxx messages from the MDD.

The custom audio messages, which can be specified to set or check the TSC2301 programmable audio functions (the contents in TSC2301 control registers), are processed in the routine ***PDD_AudioMessage( ).***

8 Conclusion

Table 8 summarizes the TSC2301 drivers, and the hardware requirements for a processor to implement the corresponding TSC2301 drivers and functions.

Table 8. TSC2301 WinCE Drivers and Processor Requirements

FUNCTION	TOUCH SCREEN	KEYPAD	AUDIO
SW Driver Required	● Touch Driver ***TSC2301Touch.CPP*** ● SPI Driver ***TSC2301SPI.C***	● Keypad Driver ***TSC2301Key.CPP*** ● SPI Driver ***TSC2301SPI.C***	● Audio Driver ***TSC2301Audio.C*** ● SPI Driver ***TSC2301SPI.C***
Processor HW Required	● An external HW Interrupt for $\overline{\text{DAV}}$ ● SPI Port	● An external HW Interrupt for $\overline{\text{KEYIRQ}}$ ● SPI Port	● DMA ● I2S Port ● SPI Port

Generally, to use the TSC2301 drivers, simply copy the corresponding driver into the proper directory on the applied processor's WinCE development platform. For example, copy the ***TSC2301Touch.CPP*** for the touch screen function (and the processor-dependent routines as well) to the *Touch* driver directory. Also, update the source file and include the ***TSC2301Touch.CPP*** and ***XXXXXTouch.CPP*** to its file list. Refer to references [4] and [5] for more details.

The TSC2301 generic driver code referenced in this application note is available for download. Visit the TI website at www.ti.com and follow the links. The system has been tested using Intel's PXA250 processor (Lubbock platform) and on the AT91RM9200 processor (NOAH platform).

9 References

1. TSC2301 data sheet – *Programmable Touch Screen Controller with Stereo Audio CODEC*. (SLAS371)
2. *Windows CE .Net Touch Screen, Keypad and Audio Device Driver for TSC2301* (SLAA169)
3. Related documentation for Windows CE from Microsoft or other sources
4. *Installing TSC2301 WinCE Generic Drivers on an XScale Platform,* SLAA188
5. *Integrate TI Touch Screen Control Generic WinCE Drivers to AT91RM9200 Processors,* (In development)

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