Application Report
SLAA198 – April 2004

TSC2100 WinCE Generic Drivers

Wendy X. Fang DAP Group

ABSTRACT

This application report develops and presents the TSC2100 touchscreen and audio WinCE drivers. These generic drivers can be used on or adapted to different processors or processor platforms. The associated driver code was tested with the Intel™ Lubbock and MainStone platforms. The sample code described in this application report can be downloaded from http://www.ti.com/lit/zip/SLAA198.

Introduction

The TS2100 programmable touchscreen controller, one of the Texas Instruments (TI™) high-performance touchscreen devices, includes a touchscreen controller and a mono input and stereo output audio codec. For detailed specification of the TSC2100 device, see the relevant data sheet (Reference [1]) or the product folder at the following TI Web location:

http://focus.ti.com/docs/prod/folders/print/tsc2100.html

To review other TI touchscreen controllers, browse the analog product tree at the following TI Web location:

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

To assist TI customers in using the TSC2100 device in their application, this application report developed the Microsoft™ Windows CE™ (WinCE) drivers on its touchscreen and audio functions. The TSC2100 drivers can be implemented on any processor; the drivers are built and arranged in such a way that the processor-related code is separated out and put on a processor-dependent layer (PDL). By changing only the PDL, these WinCE drivers can be easily reused and adapted to different host processors. Because of this, the TSC2100 WinCE drivers are considered generic.

If users wish to do no coding with the drivers, additional consultation and service on integrating and interfacing these and other TI TSC drivers to many different processors are available through TI's third party contractors by searching the following URL:

http://www.ti.com/analog3p

The TSC2100 drivers discussed in this application report were tested on Intel XScale™ processors with the Lubbock (WinCE 4.0 and the Intel PXA250 microprocessor) and the MainStone (WinCE 4.2 and Intel's Bulverde microprocessor) platforms. To obtain the driver code, contact TI data acquisition application support at:

dataconvapps@list.ti.com

Principles

Two TSC2100 WinCE drivers have been developed, the touchscreen driver and the audio driver. In the Windows CE device driver model, the two TSC2100 drivers fit into:

- TouchP: a standard touch panel/screen driver for the touchscreen feature
- WaveDev: a standard audio driver for the audio feature.

The touchscreen driver is a typical built-in or native device driver; the audio driver can be classified as a hybrid (both native and stream) driver. Both the touchscreen and audio drivers have a layered model structure (see Figure 1).

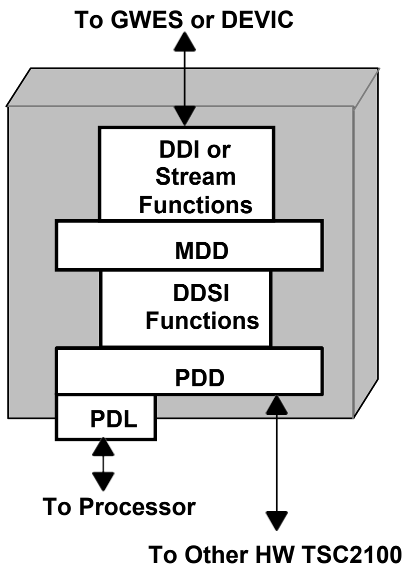

Figure 1. Layered TSC2100 WinCE Driver

Basically, the layered driver development requests changes only at its platform-dependent-device (PDD) layer. For more details on WinCE driver structure, model, and classification, see Reference [2] or related documentation from Microsoft.

Note that when compared with the standard WinCE driver structure, Figure 1 shows an additional lower or sub-layer below the PDD, called the processor-dependent layer (PDL). The purpose of the PDL is to make the adaptation of the drivers to various platforms or processors easy. Adapting the TSC2100 drivers to a new and different platform requires modifying only the code in the PDL.

Thus, developing the WinCE drivers for the TSC2100 requires the following tasks:

- Development of a common library that implements all the SPI exchange with the TSC2100
- Development of the PDD (and the PDL) layer of the TSC2100 touchscreen driver (native driver mounted by GWES)
- Development of the PDD (and the PDL) layer of the TSC2100 audio driver (hybrid driver mounted by DEVICE)

SPI Interface

The SPI bus on the TSC2100 device is the most important hardware interface to the host processor. Because the host processor controls and accesses the TSC2100 registers through the SPI interface, the SPI driver is the most important software interface between the host processor and the TSC2100.

Hardware Interface

A standard SPI hardware interface consists of 4 pins/lines, typically 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 connections between the Intel PXA250 processor and the TI TSC2100.

Table 1. PXA250 and TSC2100 SPI HW Interface

	Host Processor Pin Name	TSC2100 Pin Name
SPI Clock	GPIO 23/SSP_SCLK (Pin F9)	SCLK (QFN Pin 4 or TSSOP Pin 8)
SPI Slave Select	GPIO 24/SSP_SFRM (Pin E9)	/SS (QFN Pin 7 or TQFP Pin 11)
SPI MOSI Data	GPIO 25/SSP_TXD (Pin D9)	MOSI (QFN Pin 6 or TQFP Pin 10)
SPI MISO Data	GPIO 26/SSP_RXD (Pin A9)	MISO (QFN Pin 5 or TQFP Pin 9)

Note: For the TSC2100 SPI driver, the PXA250 processor's GPIO 23, 25, and 26 were programmed as SSP function pins, but the GPIO 24 should be used as a GPO pin, so as to interface the 16-bit SPI protocol in the TSC2100 device.

In the SPI interface, the TSC2100 is always the slave device; the host processor is the master.

TSC2100 Control Registers

A series of registers reside in the three memory pages of the TSC2100 device, containing the device's data, status, all programmable controls, variables, and parameters. These TSC2100 registers are accessible or controllable by the host processor through the SPI interface.

For the definition of the registers and the assignment of the register bits, see the TSC2100 data sheet, Reference [1].

For users' convenience, the definitions and assignments of the TSC2100 registers have been coded in a header file: *TSC2100Regs.H*. This file has been associated with this application report and can be downloaded from TI Web site.

SPI Driver

The TSC2100 SPI driver establishes the software interface between the processor and TSC2100, which contains 4 files: *TSC2100SPI.C, TSC2100SPI.H, XXXXXSPIComm.C and XXXXXSPIComm.H. T*he *XXXXX* stands for the name or type of the utilized processor. Therefore, the two files are processor-dependent and are located in the PDL. For example, for an XScale processor, the two processor-dependent files may be named as *XScaleSPIComm.C* and *XScaleSPIComm.H*.

Among the many routines in the aforementioned four files, the fundamental routines for the SPI driver are summarized in Table 2. The processor-related or the PDL routines in Table 2 were named with the *HW* in the first two letters of its name. As previously noted, different processors require different routines in the PDL layer.

Table 2. Fundamental Routines for SPI Driver

Item	**Routine Name**	**Involved Processor-Dependent Routines**	**Function**

1	*SetupSPIController( )*	*HWInitializeSPIDriver()*
		HWSetupSPIController()
			Set up the host processor's SPI port so as to be ready for interface

2	*StopSPIController( )*	*HWDeinitializeSPIDriver()*
		HWStopSPIController( )
			Stop the host processor's SPI port and interface

3	*SPITransaction( )*		Write/read one or more TSC2100 registers
		HWStartFrame( )	/SS goes low to activate
		HWStopFrame( )	/SS goes high to de-activate
		HWSPIWriteWord( )	Write a word to MOSI
		HWSPIReadWord( )	Read a word from MISO
		HWSPITxBusy( )	Check if an SPI read has completed
		HWSPIRxBusy()	Check if an SPI write has completed
		HWSPIFIFONotEmpty( )	Ensure the SPI FIFO has been properly read so that the read data are the latest
4	*TSC2100ReadReg( )*		Read the content from a TSC2100 register
5	*TSC2100WriteReg( )*		Write a 16-bit value to a TSC2100 register

In Table 2, the routines in items 1, 2, and 3 are related to the processor and its SPI configuration. The routines in items 4 and 5 are the bridges for the host processor to access all control registers of the TSC2100.

TSC2100 Touchscreen Driver

The touchscreen driver handles and controls the TSC2100 touchscreen function, which is normally in the TouchP directory. The driver includes the file: *TSC2100Touch.CPP,* together with the PDL files *XXXXXTouch.CPP* and *XXXXXTouch.H.* Again, the *XXXXX* stands for the name of the processor used.

The TSC2100 touchscreen driver was developed under the touch panel driver structure of the WinCE operating system (OS) by updating the standard DDSI functions. See Reference [2] or WinCE driver documentations for details of the standard touch DDSI functions or routines.

Touch Initialization

The TSC2100 touchscreen function initialization or setup is implemented by the subroutine *InitTSC2100Touch( ),* called in the WinCE touch DDSI routine *DdsiTouchPanelEnable( )*. Even though the TSC2100 does not require a strict programming order, the following sequence shown in Figure 2 is recommended.

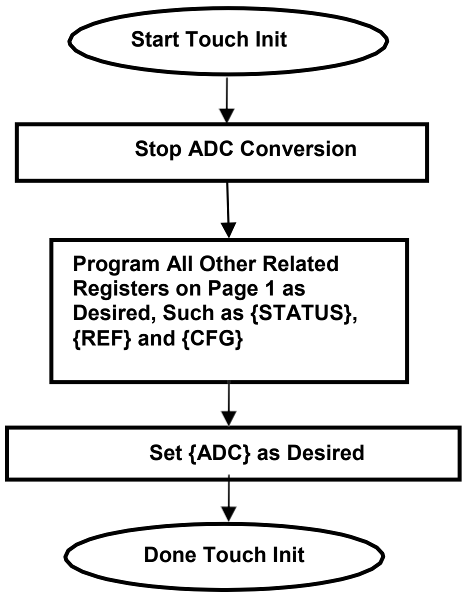
Start Touch Init
Stop ADC Conversion
Program All Other Related Registers on Page 1 as Desired, Such as {STATUS}, {REF} and {CFG}
Set {ADC} as Desired
Done Touch Init

Figure 2. **TSC2100 Touchscreen Driver Initialization - *InitTSC2100Touch( )***

In Figure 2, a TSC2100 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 denoted by {CFG}; and so on.

The following is an example of the TSC2100 touchscreen function initialization:

```
TSC2100WriteReg( {ADC},       0xC4FE);    // Stop ADC
TSC2100WriteReg( {STATUS},    0x4000 );   // Set /DAV interrupt
TSC2100WriteReg( {REF},       0x0017);    // Set Internal Reference
TSC2100WriteReg( {CFG},       0x000B);    // Set pre-charge & sense times
TSC2100WriteReg( {ADC},       0x84E6);    // Set to TSC & XY mode
```

Touch Data Reading

Under the preceding initialization example, the TSC2100 is in the touchscreen-controlled (vs. host-controlled) mode. The /PINTDAV pin is set as the /DAV (not the PENIRQ) hardware interrupt. The /DAV interrupt occurs (goes low) whenever the screen/panel is touched and the touch data has been sampled, converted, and is ready to be read. At this point, the driver reads back all the touch data, so as to reset the /DAV interrupt (goes high), and is ready for the next data acquisition.

To read the touchscreen data from the TSC2100 touch data registers, the routine *SampleTouchScreenTSC2100*(*X, *Y) is called in the DDSI function *DdsiTouchPanelGetPoint*().

The above touch initialization and XY data reading code can be found in the file TSC2100Touch.CPP by looking into the *DdsiTouchPanelEnable*( ) and *DdsiTouchPanelGetPoint*() routines, respectively.

TSC2100 Audio Driver

The TSC2100 audio driver consists of the file *TSC2100Audio.C* and the header file *TSC2100Audio.H*, as well as the PDL files *XXXXXAudio.C* and *XXXXXAudio.H.*

The TSC2100 audio driver was developed within the OS PDD layer, with no changes to the upper layers of the audio architecture. See Reference [2] or WinCE audio driver documentations for more details of the standard audio PDD functions.

The PDD layer uses the SPI interface to control the TSC2100 audio functions by writing to the page 2 control registers of the TSC2100 and uses DMA to send and receive audio data from the I2S bus.

Audio Initialization

Audio initialization has three main tasks: (1) set up SPI interface, (2) set up TSC2100 audio control registers, and (3) set up the DMAC structure for audio data (I2S) transformations. The three tasks are performed in the audio DDSI routine *PDD_AudioInitialize( ).*

The TSC2100 audio control registers are initialized in the *TSC2100Audio( )* subroutine. The following is an example for initializing the TSC2100 audio function:

```
TSC2100WriteReg( { AUDCTRL }, 0x0000 );
TSC2100WriteReg( { ADCVOL }, 0x8000 );
TSC2100WriteReg( { DACVOL }, 0x8080 );
TSC2100WriteReg( { BPVOL }, 0xC580 );
TSC2100WriteReg( { KEYCLICK}, 0x44F0 );
TSC2100WriteReg( {AUDCTRL3}, 0x3000 );
TSC2100WriteReg( {PLL1},          0x1120 );
TSC2100WriteReg( {PLL2},          0x0000 );
TSC2100WriteReg( {AUDCTRL4}, 0x0030 );
TSC2100WriteReg( {AUDCTRL5}, 0xFE00 );
TSC2100WriteReg( {AUDPD},         0xBFC0 );
```

Audio Power-Up Sequence

When the output of the audio codec is powered up, there is usually a weak but audible pop noise from the audio output, which is due to the slight change of the DAC and/or the driver output offset. In the TSC2100 device, the DAC converter and the output driver both feature the on-chip pop noise reduction schemes.

Depending on the connection to the TSC2100 audio left and right channel outputs, HPL and HPR, the TSC2100 can be called either under the *cap* (capacitor) mode (if there are AC coupling capacitors in between) or under the *capless* mode (if the HPL and HPR are directly connected to a headphone and have no capacitor in between). Table 3 lists the power-up sequence under the *cap* and *capless* modes.

Table 3. An Optimum Power-Up Sequence for TSC2100

Step 1	If in cap mode, make sure that VGND has been powered down If in capless mode, make sure to power up the VGND first
Step 2	Keep Driver Pop Reduction settings at the default (i.e., it is enabled with long duration) and power up headphone driver
Step 3	Power up Side-Tone; but keep it muted
Step 4	Enable DAC pop reduction features, and set it to slow and long options
Step 5	Power up DAC and then un-mute both channels

Audio Data Transformation

In the audio driver, the DMAC function of the processor moves the audio data on the I2S bus between the processor and the TSC2100. The processor DMA function should be initialized and set up, as previously noted, in the audio initialization PDD routine *PDD_AudioInitialize( )*.

Also, another PDD routine *PDD_AudioGetInterruptType( )* determines the cause of the audio interrupt and then tells the MDD layer the standard audio status: input or output playing, input or output recording, stopped, or other status.

Audio Messages

In WinCE audio device driver WaveDev structure, the two audio message-sending PDD routines are: the routine *PDD_WavProc()* that sends standard audio control messages from MDD layer to PDD layer, and the routine *PDD_AudioMessage()* that sends custom messages from the user applications to the PDD layer. The later can be used to set or update the TSC2100 control registers and can be accessed by a user application.

Conclusion

Table IV summarizes the TSC2100 drivers, together with the hardware requirement for a processor to implement the corresponding TSC2100 drivers and functions.

Table 4. TSC2100 WinCE Drivers and Processor Requirements

Function	Touchscreen	Audio
SW Driver Required	Touch Driver *TSC2100Touch.CPP* *XXXXXTouch.CPP* *XXXXXTouch.H* SPI Driver *TSC2100SPI.C* *TSC2100SPI.H* *XXXXXSSPComm.C* *XXXXXSSPComm.H* *TSC2100REG.H*	Audio Driver *TSC2100Audio.C* *TSC2100Audio.H* *XXXXXAudio.C* *XXXXXAudio.H* SPI Driver *TSC2100SPI.C* *TSC2100SPI.H* *XXXXXSSPComm.C* *XXXXXSSPComm.H* *TSC2100REG.H*
Processor HW Required	An external HW Interrupt for /DAV SPI Port	DMA I2S Port SPI Port

To use the TSC2100 drivers, simply copy the corresponding driver into the proper directory on the applied processor's WinCE development platform. As an example, the installation procedure for the TSC2100 drivers on the Lubbock platform follows.

- To install the header files for TSC2100 control registers and SPI driver, copy files in **\TSC2100Driver\Inc\** to **C:\WINCE400\PLATFORM\XSC1BD\inc\**
- To install the SPI driver, copy files in \TSC2100Driver\Drivers\SPILIB\ to C:\WINCE400\PLATFORM\XSC1BD\drivers\DRVLIB\

- To install the touchscreen driver, copy files in \TSC2100Driver\drivers\Touch\ to C:\WINCE400\PLATFORM\XSC1BD\drivers\TouchP\
- To install the audio driver, copy files in \TSC2100Driver\Drivers\Audio\ to C:\WINCE400\PLATFORM\XSC1BD\drivers\WAVEDEV\
- Additionally, to set up the interrupts for the touchscreen, including the /DAV hardware interrupt and a software timer interrupt, the files ***cfwxsc1.c*** and ***intxsc1.***c in **C:\WINCE400\PLATFORM\XSC1BD\KERNEL\** directory needs to be updated.
- To set up the DMAC for the audio driver, the header files, ***dmac.h*** and ***dmacbits.h*** also need updating.

References

1. *Programmable Touch Screen Controller with Integrated Stereo Audio CODEC and Headphone/Speaker Amplifier* (SLAS378)
2. *TSC2301 WinCE Generic Drivers* (SLAA187)
3. *Windows CE .Net Touch Screen, Keypad, and Audio Device Driver for the TSC2301* (SLAA169)