Application Report
SLAA251 – May 2005

TSC2101 WinCE 5.0 DRIVERS

Wendy X. Fang — Data Acquisition Industrial Audio

ABSTRACT

TSC2101 Microsoft™Windows™ CE 5.0 touch and audio drivers have been developed, and the code has been tested on an Intel™ Mainstone II development platform. This application report discusses these TSC2101 drivers, including the hardware connection between the TSC2101 and the platform, the Windows CE 5.0 drivers' code and structure, and the installation process.

1 INTRODUCTION

The TSC2101 Windows™ CE (WinCE) 5.0 drivers were developed to help TSC2101 users quickly set up, run, and use the device in their Windows CE 5.0 systems. Like the corresponding Windows CE 4.x drivers (see SLAA200), these drivers were coded on the standard device driver's PDD (platform-dependent device) layer, and the PDD layer was further split to have an additional processor-dependent layer (PDL) so as to make the TSC2101 drivers easy to port into different host processors. See TI application reports SLAA200 and SLAA187 for details.

Although the TSC2101 WinCE 5.0 drivers presented in this application report can be integrated and adapted into the customer's software system under different host processors, they were run and tested only on the Intel™ Mainstone II platform with the PXA270 Step B0 processor.

2 CONNECTIONS

The TSC2101 device must be wired and connected to a host processor, where the device driver code is ported and executed. The following 10 digital signals are essential for running the TSC2101 touch and audio drivers.

- the SPI bus, 4 wires: SCLK, $\overline{\text{SS}}$, MOSI, and MISO (at J14 on TSC2101 EVM)
- the touch Pen-Down or data available interrupt, $\overline{\text{PINTDAV}}$ (at J14 on TSC2101 EVM)
- the main audio codec clock, MCLK (at J13 on TSC2101 EVM)
- the I2S bus, 4 wires: BCLK, WCLK, SDIN, and SDOUT (at J13 on TSC2101 EVM)

For developing the TSC2101 drivers, the TSC2101 evaluation module (see SLAU112) and the Intel Mainstone II platform with the PXA270 Step B0 processor (see Reference 4) were used. The wires and connection between the two systems are illustrated in Figure 1.

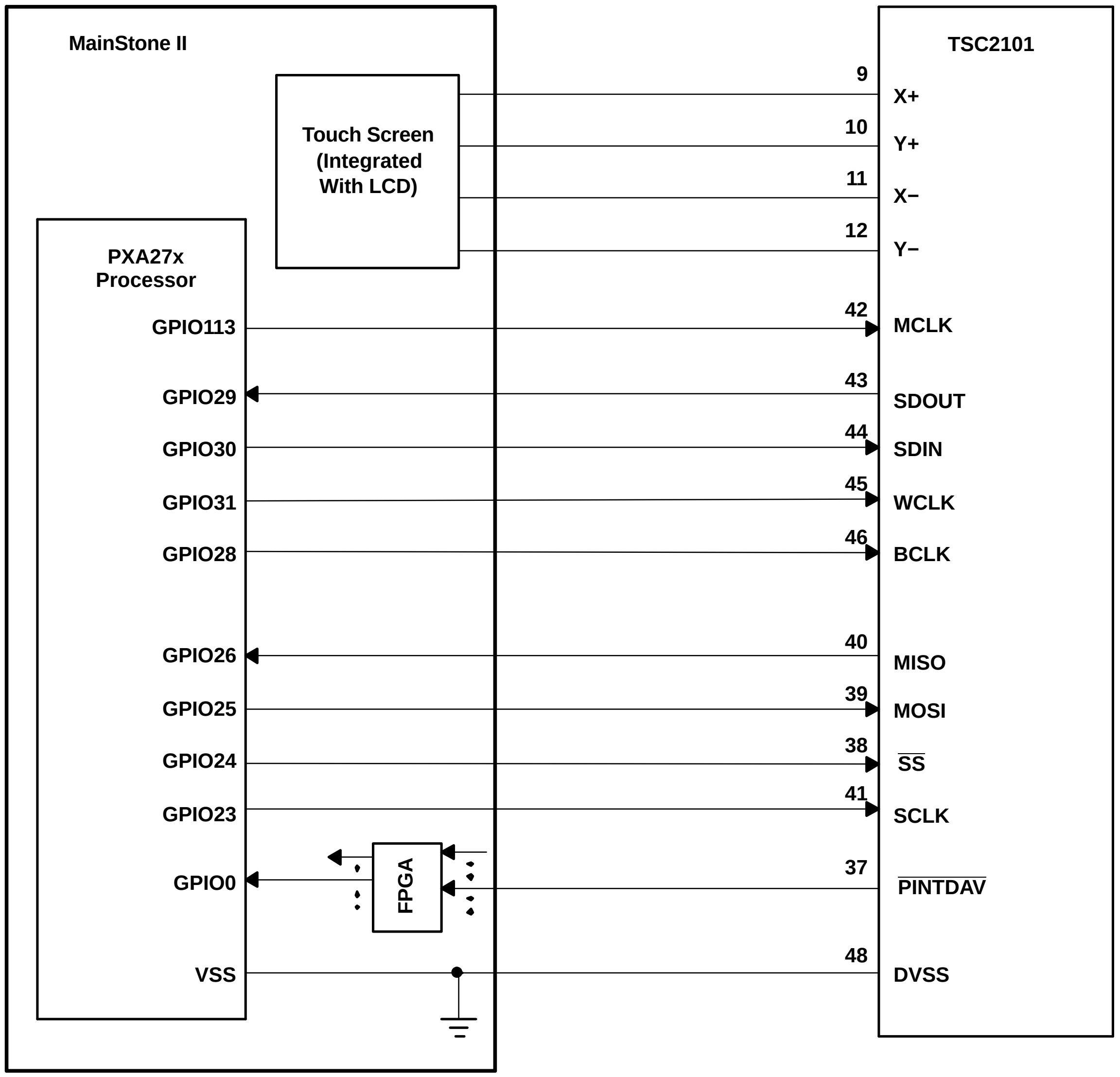

Figure 1. TSC2101 Connections to Mainstone II System

On the TSC2101 evaluation module (EVM) board, the USB-SPI, USB-I2S, and USB-MCLK of the SW2 were turned OFF so that the external connections from the host processor could be attached and interfaced to the TSC2101 device. See user's guide SLAU112 for the schematic and other details of the EVM system.

On the Mainstone II system, the original touch/audio module, connected on the Mainstone II main board, was removed, replaced with the connections as shown in Figure 1. See Reference 4 and other Intel documentation for additional information of the Mainstone II Platform.

In addition to the preceding 10 digital signal pins, the TSC2101 pin 2, $\overline{\text{PWR\_DN}}$, can be connected to one of the GPIO pins of the host, if desired. The pin 3, $\overline{\text{RESET}}$, can be routed to the system $\overline{\text{RESET}}$ or a GPIO of the host processor. For the testing reported in this application report, both pins were pulled high.

3 DEVICE DRIVERS

Figure 2 lists the TSC2101 touch and audio device drivers' code files; the files having filenames starting with “Host…” are the processor-dependent code or PDL, such as HostTouch.CPP or HostSPIComm.H.

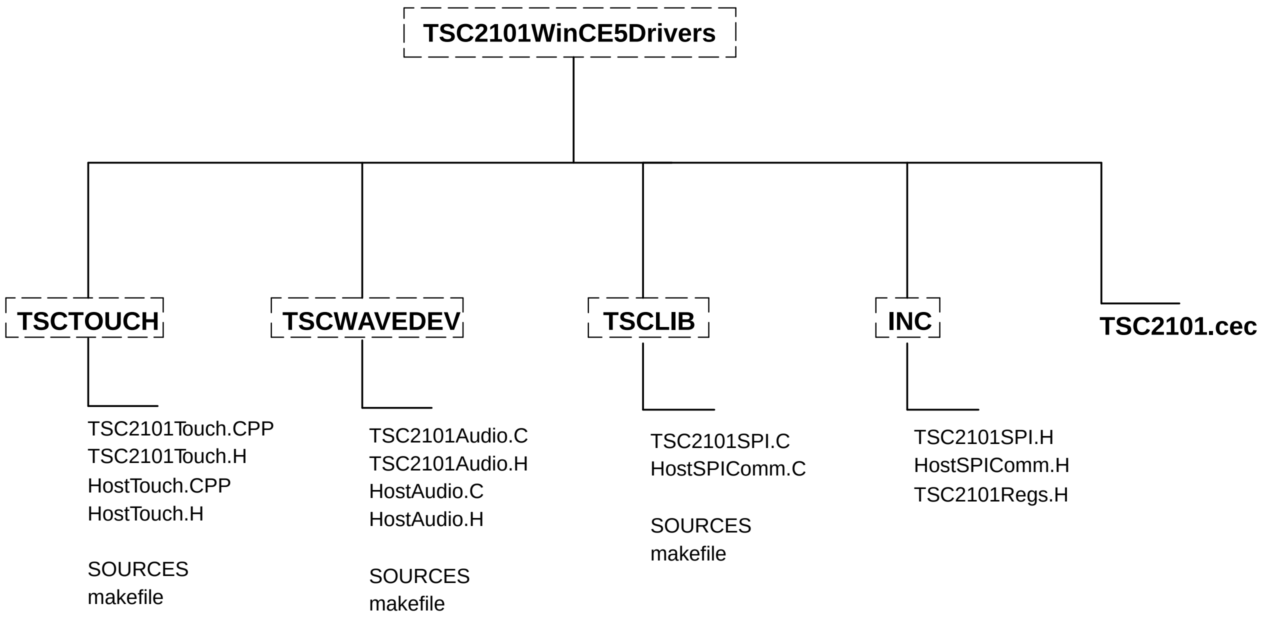

Figure 2. TSC2101 WinCE 5.0 Drivers Files

As the TSC2101 data sheet (SLAS392) shows, on a TSC2101 device, two digital serial buses, SPI and I2S, are the keys to the TSC2101 drivers.

The SPI bus is the control bus for the TSC2101 device, through which the host processor accesses the TSC2101 control registers and controls both touch and audio functions. Additionally, the host reads the touch-screen data via this SPI interface. Both the touch and audio drivers use the SPI communication code; therefore, this code was developed as a library and located in the directory **TSCLIB**.

On the other hand, the I2S bus is only the audio data bus, through which the host streams audio ADC and DAC data in and out the TSC2101. Thus, the I2S interface is used only in the audio device driver, and the code is located in the directory **TSCWaveDev**.

3.1 *SPI INTERFACE*

The four TSC2101 pins, SCLK, $\overline{\text{SS}}$, MOSI, and MISO are connected to the GPIO23, GPIO24, GPIO25, and GPIO26 of the PXA27x processor, respectively.

On the host side, the PXA27x GPIO, SSP, and Clock management control registers were used for setting up the SPI interface to communicate with the TSC2101’s SPI. The setup was implemented at the routine, HWSetupSPI():

```
void HWSetupSPI(BOOL InPowerHandle)
{
    RETAILMSG(1,(TEXT("Setup Host GPIO & SSP for an SPI Interface... \r\n")));
    // disable Unit clock
    g_pClockRegs->cken &= ~XLLP_CLKEN_SSP1;
    // disable SSP1
    g_pSSPRegs->sscr0 &= ~SSE_ENABLE;
    // Set up the GPIO24=SFRM = 1 (GPSR0)
    g_pGPIORegs->GPSR0 |= ( GPIO_24_SFRM );
    // Program direction of the GPIOs (GPDR0)
    // (GPIO23/24/25 as outputs and GPIO26 as input)
    g_pGPIORegs->GPDR0 |= GPIO_23_SCLK;
    g_pGPIORegs->GPDR0 |= GPIO_24_SFRM;
    g_pGPIORegs->GPDR0 |= GPIO_25_MOSI;
    g_pGPIORegs->GPDR0 &= ~GPIO_26_MISO;
    // Program GPIO alternate function register (GAFR0_U)
    g_pGPIORegs->GAFR0_U &= 0xFFC03FFF;
    g_pGPIORegs->GAFR0_U |= GPIO_23_AF2_SSPSCLK;
    // GPIO24 is used here as GPO
    g_pGPIORegs->GAFR0_U |= GPIO_25_AF2_SSPTXD;
    g_pGPIORegs->GAFR0_U |= GPIO_26_AF1_SSPRXD;
    // Set up SSP registers (when disabled SSP)
    // set up SSP control register 0 and 1
    g_pSSPRegs->sscr0 = (SCR_590_KHZ | SSE_DISABLE | ECS_INTERNAL |
                         FRF_MOTOROLA | DSS_16_BIT );
    g_pSSPRegs->sscr1 = (RFT_SEVEN | TFT_ZERO | MWDS_16_BIT | SPH_HALF_DELAY |
                         SPO_IDLE_LOW | LBM_DISABLE | TIE_DISABLE | RIE_DISABLE );
    // Enable SSP last
    g_pSSPRegs->sscr0 |= SSE_ENABLE;
    // enable SPI1 Unit clock
        g_pClockRegs->cken |= XLLP_CLKEN_SSP1;
}
```

Note:

(1) The host's GPIO24 was used as the SPI's $\overline{\text{SS}}$, but this pin in PXA27x processor should be programmed as a GPO and be set high and low by SW (not through the PXA27x SSP function); and (2) the SSP1 clock should be turned OFF before the setup and turned ON after the setup.

3.2 TOUCH-SCREEN DRIVER

Concerning hardware, the touch driver requires only five digital pins of the TSC2101: the 4-wire SPI bus, and the pen-down or data-available interrupt pin, $\overline{\text{PINTDAV}}$. See Figure 1.

The SPI interface provides communications for the controls and the touch data between the host and the TSC2101; the interrupt signal from the TSC2101 triggers the host to implement a touch data-reading or samples-getting procedure. In the Mainstone II system, the $\overline{\text{PENIRQ}}$ was connected to an FPGA, where the $\overline{\text{PENIRQ}}$ could be ORed with other secondary interrupts and fed to the PXA27x GPIO0 pin (see Reference 4). During the testing reported in this application report, the TSC2101's $\overline{\text{PINTDAV}}$ pin was wired to Mainstone II's $\overline{\text{PENIRQ}}$, which actually went to the PXA27x GPIO0 pin (see Figure 1).

The touch-screen device driver is in the directory **TSCTOUCH**, developed on the PDD layer of the standard touch device driver structure.

In this application report, the TSC2101 touch control registers (in Page1 of the TSC2101 memory space) were set up or initialized in the routine **InitTSC2101Touch()** and called by the touch PDD routine **DdsiTouchPanelEnable()**, which is:

```
// Initialize TSC2101 Touch Screen Registers for
// Normal X/Y TouchScreen Operation
void InitTSC2101Touch(BOOL bInPowerHandler)
{
    RETAILMSG(1, (TEXT("InitTSC2101Touch.\r\n")));
    TSC2101WriteReg(TSC2101_ADC, ADC_STOP_CONVERSIONS, bInPowerHandler);
    TSC2101WriteReg(TSC2101_STATUS, STATUS_INT_DATA, bInPowerHandler);
    TSC2101WriteReg(TSC2101_REF, REF_SETUP_VALUE, bInPowerHandler);
    TSC2101WriteReg(TSC2101_CFG, CFG_SETUP_VALUE, bInPowerHandler);
    TSC2101WriteReg(TSC2101_ADC, ADC_SETUP_VALUE, bInPowerHandler);
    RETAILMSG(1, (TEXT("Done InitTSC2101Touch.\r\n")));
}
```

See the TSC2101Regs.H file for the setup values in the foregoing routine.

Note:

(1) The touch data ADC was programmed into the TSC-controlled and XY-data mode; and (2) the $\overline{\text{PINTDAV}}$ pin is programmed as the $\overline{\text{DAV}}$.

Whenever the touch panel was touched, the TSC2101 started to sample the X and then Y data; when the ADC conversions for both X and Y were finished and the data was ready for reading, the $\overline{\text{DAV}}$ signal (from TSC2101 $\overline{\text{PINTDAV}}$ pin) would go low to trigger the PXA27x GPIO0 interrupt. As the result, the touch PDD routine, named **DdsiTouchPanelGetPoint ()**, was called to get both X and Y data, and also to send the data and status to the up-layer of the driver, i.e., MDD. For example:

```
VOID DdsiTouchPanelGetPoint(TOUCH_PANEL_SAMPLE_FLAGS *pTipStateFlags,INT *pUncalX,INT *pUncalY)
{
      static BOOL TouchIrq = TRUE;
      UINT32 InterruptType = SYSINTR_NOP;
       RETAILMSG(1,(TEXT("Calling DdsiTouchPanelGetPoint()\r\n")));
       *pTipStateFlags = TouchSampleIgnore;
      if (TouchIrq)
      {
           // The pen was previously up - it just transitioned to down state.
           TouchIrq = FALSE;
           InterruptType   = SYSINTR_TOUCH;
           // Read X and Y data through the SPI
          *pTipStateFlags = PDDSampleTouchScreen(pUncalX, pUncalY);
           // reset touch IRQ
           HWDisableTouchIRQ();
           HWResetTouchIRQ();
           // The next expected interrupt will come from sampling timer
           // (pen-up doesn't cause an interrupt).
           g_NextExpectedInterrupt = PEN_UP_OR_TIMER;
      }
      else
      {
           // The timer irq ...
           // The pen could now be either up or down at this point
           InterruptType   = SYSINTR_TOUCH_CHANGED;
           // Read X and Y data through the SPI
           *pTipStateFlags = PDDSampleTouchScreen(pUncalX, pUncalY);
           // The next expected interrupt ..
           g_NextExpectedInterrupt = PEN_UP_OR_TIMER;
       }
       if ((g_NextExpectedInterrupt == PEN_UP_OR_TIMER) && !PenIsDown())
       {
           TouchIrq = TRUE;
           // the pen isn't currently down, send the MDD a pen-up "event".
           *pTipStateFlags = TouchSampleValidFlag;
           // The next expected interrupt will come from pen-down event.
           g_NextExpectedInterrupt = PEN_DOWN;
       }

       // Make sure the next expected interrupt is configured and enabled.
       PrepareNextInterrupt(g_NextExpectedInterrupt);
       // Tell the OAL to clear and unmask interrupt just occurred.
       InterruptDone(InterruptType);
}
```

3.3 AUDIO DRIVER

Concerning hardware, the TSC2101 audio driver required both SPI (for audio control) and I2S (for audio data steaming) buses. The SPI bus controls the audio codec’s operation by writing to TSC2101 audio control registers, and the I2S bus transfers audio data between the host and the TSC2101.

The TSC2101 MCLK pin should receive an external clock that provides the necessary clock for TSC2101 audio sigma-delta ADC and DAC to operate or run. The MCLK to the TSC2101 should be generated from the same resource as the I2S clocks, i.e., the MCLK should also from the host processor, which is the I2S master in this application report.

The TSC2101 audio driver was built on the standard audio driver, WaveDev, and located in the directory **TSCWaveDev**.

On the host side, the PXA27x GPIO28, GPIO29, GPIO30, and GPIO31 pins were used for the I2S bus and connected to TSC2101's BCLK, SDOUT, SDIN, and WCLK, respectively (see Figure 1). The GPIO113 is programmed as the I2S SYSCLK and connected to the MCLK, which is programmed to generate a 11.346-MHz clock. The I2S setup was implemented at the routine, **HWEnableI2S()**, such as:

```
//
//-------------------------------------------------------------------
// Function: HWEnableI2S()
//-------------------------------------------------------------------
//
void HWEnableI2S(void)
{
    RETAILMSG(1,(TEXT("Setup Host GPIO & I2S Interface... \r\n")));
    //Basic Outline:
    // configure the GPIO registers and set to I2S mode
    // Set up I2S control registers at default condition
    // insert reset for I2S
    v_pI2SRegs->sacr0 |= 0x00000008;
    // un-insert the reset
    v_pI2SRegs->sacr0 = 0x00007700;
    // disable I2S unit clock
    v_pClockRegs->cken &= ~XLLP_CLKEN_I2S;
    // setup GPIO direction regs
    v_pGPIORegs->GPDR0 |= XLLP_GPIO_BIT_I2SBITCLK     |
                         XLLP_GPIO_BIT_I2S_SYNC      |
                    XLLP_GPIO_BIT_I2S_SDATA_OUT;
    v_pGPIORegs->GPDR0 &= ~XLLP_GPIO_BIT_I2S_SDATA_IN;
    v_pGPIORegs->GPDR3 |= XLLP_GPIO_BIT_I2S_SYSCLK;           // GPIO113: SYSCLK as output
    // configure GPIO alternate function registers
    v_pGPIORegs->GAFR0_U &= 0x00FFFFFF;
    v_pGPIORegs->GAFR0_U |= XLLP_GPIO_AF_BIT_I2SBITCLK_OUT |
              XLLP_GPIO_AF_BIT_I2S_SDATA_IN  |
              XLLP_GPIO_AF_BIT_I2S_SDATA_OUT |
              XLLP_GPIO_AF_BIT_I2S_SYNC;
    v_pGPIORegs->GAFR0_U &= ~XLLP_GPIO_AF_BIT_I2S_SYSCLK_MASK;
    v_pGPIORegs->GAFR3_U |= XLLP_GPIO_AF_BIT_I2S_SYSCLK;
    // configure I2S reg sacr0 but not enable I2S yet
    v_pI2SRegs->sacr0 = 0x00001104;
    // configure system for I2S mode
    v_pI2SRegs->sacr1 = 0x00000000;
    // configure clock divider
    v_pI2SRegs->sadiv = I2SRATE_44_1;  // divider for 44.1kHz audio
    // enable I2S
    v_pI2SRegs->sacr0 |= 0x00000001;
    // enable Unit clock
        v_pClockRegs->cken |= XLLP_CLKEN_I2S;
    //    DumpRegsI2S();
        return ;
}
```

On the TSC2101 side, the audio codec was initially set up or programmed in such a way that:

- About the I2S interface:
  1. The I2S interface is at 16 bits, standard I2S mode, with 44.1-kHz sample rate.
  2. The TSC2101 is the slave because the host is the I2S master.
- About the audio input circuitry:
  1. ADC is input from the handset microphone (MICIN_HND pin).
  2. ADC input gain is controlled by its PGA with initial gain 0 dB.
- About the audio output circuitry:
  1. The left and right DAC results are routed to the stereo headphone SPK1 and SPK2, respectively.
  2. The headphone output is in the CAPLESS mode.
  3. The output gains were initialized to 0 dB.
- About other features:

1. Side-tone is enabled at 0-dB gain and routed to Loud Speaker.
2. Key-click is enabled, at med-gain/1-kHz/32-bit length and routed to Loud Speaker.
3. CP_IN is directly routed to Loud Speaker.
4. All output short-circuit protections were enabled.
5. Power-up pop reduction was enabled.

All TSC2101 audio control registers (in Page2 of the TSC2101 memory space) were set up or initialized, as previously stated, in the routine **InitTSC2101Audio()** and called by the audio PDD routine **PDD_AudioInitialize()**. The audio initialization routine is:

```
// Initialize TSC2101 Audio Register at Default
void InitTSC2101Audio(BOOL bInPowerHandler)
{
    RETAILMSG(1, (TEXT("InitTSC2101Audio.\r\n")));
    TSC2101WriteReg(TSC2101_AUDCTL1, AUDCTL1_SETUP_VALUE, bInPowerHandler);
    TSC2101WriteReg(TSC2101_HEDVOL, HEDVOL_SETUP_VALUE, bInPowerHandler);
    TSC2101WriteReg(TSC2101_DACVOL, DACVOL_SETUP_VALUE, bInPowerHandler);
    TSC2101WriteReg(TSC2101_MIXER, MIXER_SETUP_VALUE, bInPowerHandler);
    TSC2101WriteReg(TSC2101_AUDCTL2, AUDCTL2_SETUP_VALUE, bInPowerHandler);
    TSC2101WriteReg(TSC2101_AUDCTL3, AUDCTL3_SETUP_VALUE, bInPowerHandler);
    TSC2101WriteReg(TSC2101_PLL1, PLL1_SETUP_VALUE, bInPowerHandler);
    TSC2101WriteReg(TSC2101_PLL2, PLL2_SETUP_VALUE, bInPowerHandler);
    TSC2101WriteReg(TSC2101_AUDCTL4, AUDCTL4_SETUP_VALUE, bInPowerHandler);
    TSC2101WriteReg(TSC2101_HNDVOL, HNDVOL_SETUP_VALUE, bInPowerHandler);
    TSC2101WriteReg(TSC2101_CELLVOL, CELLVOL_SETUP_VALUE, bInPowerHandler);
    TSC2101WriteReg(TSC2101_AUDCTL5, AUDCTL5_SETUP_VALUE, bInPowerHandler);
    TSC2101WriteReg(TSC2101_AUDCTL6, AUDCTL6_SETUP_VALUE, bInPowerHandler);
    TSC2101WriteReg(TSC2101_AUDCTL7, AUDCTL7_SETUP_VALUE, bInPowerHandler);
    TSC2101WriteReg(TSC2101_GPIO, GPIO_SETUP_VALUE, bInPowerHandler);
    TSC2101WriteReg(TSC2101_CELLAGC, CELLAGC_SETUP_VALUE, bInPowerHandler);
    TSC2101WriteReg(TSC2101_DRVPD, DRVPD_SETUP_VALUE, bInPowerHandler);
    TSC2101WriteReg(TSC2101_MICAGC, MICAGC_SETUP_VALUE, bInPowerHandler);
    TSC2101WriteReg(TSC2101_CELLAGC2, CELLAGC2_SETUP_VALUE, bInPowerHandler);
    TSC2101WriteReg(TSC2101_AUDPD, AUDPD_SETUP_VALUE, bInPowerHandler);
    RETAILMSG(1, (TEXT("Done InitTSC2101Audio.\r\n")));
```

See the file TSC2101Regs.H for the actual setup values in the preceding routine.

As application report SLAA230 points out, the power-up/down sequences of the TSC2101 are important for reducing power-up pop noise and for eliminating possible malfunction. See the routines **PowerDownADC()** and **PowerDown DAC()** for examples of how to power up and down the CODEC in proper sequences.

4 INSTALLATION

To run the TSC2101 WinCE 5.0 drivers on Intel Mainstone II Platform, the following installation steps are provided.

Included with the Microsoft™ Windows CE 5.0 platform builder CD ROM is the Board Support Package (BSP) of the Mainstone II, called \MAINSTONEII\, which can be located on your PC after installing the Platform Builder 5.0 at, for example:

C:\WinCE500\PLATFORM\.

To install the TSC2101 Windows CE 5.0 touch and audio drivers into one of Mainstone II WorkSpaces, execute the following steps.

Step I: Copy —

1. Copy \TSC2101WinCE5Drivers\TSC2101.cec file into:
   C:\WINCE500\PUBLIC\COMMON\OAK\CATALOG\CEC\
2. Copy all files inside \TSC2101WinCE5Drivers\INC\ into:
   C:\WINCE500\PLATFORM\MAINSTONEII\SRC|\INC\
3. Copy the directories **TSCLIB, TSCTouch**, and **TSCWaveDev** into:
   C:\WINCE500\PLATFORM\MAINSTONEII\SRC\DRIVERS\

Step II: Set Up —

This step sets up the catalog to include the TSC2101 device drivers.

1. Run "**Platform Builder 5.0**", and the Platform Builder IDE appears.
2. At the Platform Builder 5.0 IDE, open **Manage Catalog Items** from the menu **File\Manage Catalog Items**… \. When the Manage Catalog Items window appears, click on Import button on the right side of the window, navigate, find, and select TSC2101.cec in the directory:
   C:\WINCE500\PUBLIC\COMMON\OAK\CATALOG\CEC\,
   and then click on **Open** so that the item is ported in.
3. Click and drag to select all *.cec files in the **Manage Catalog Items** Window, and then click on the **Refresh** button to make sure the new item is loaded.
4. Close the **Manage Catalog Items** window by clicking on its **OK** button.

Step III: Open –

Per the application, this step opens a new or existing Mainstone II workspace in the Platform Builder 5.0 IDE . The procedure is ignored

Step IV: Add –

This step adds the TSC2101 device drivers from the **Catalog** into the existing **OS design**.

1. In the **Catalog** window of the **Platform Builder 5.0 IDE**, find **TI TSC2101 Touch Controller Driver**, right-click on it, and select **Add to OS Design** to add the touch controller driver to the OS.
2. Similarly, find **TI TSC2101 Audio CODEC Driver**, right-click on it, and select **Add to OS Design** to add the audio driver to the operating system (OS).
3. As a result, both device drivers should appear under the **Device Drivers section** at the **OSDesignView** window of the WorkSpace.

Step V: Modify –

This step modifies the building device drivers so as to include TI TSC2101 drivers.

1. Open the **dirs** file in the directory:
   C:\WINCE500\PLATFORM\MAINSTONEII\SRC\DRIVERS\
2. Eliminate the original **touch** from the list, and add on the **TSCLIB, TSCTOUCH** and **TSCWAVEDEV**. For example, the dirs file could be:

```
DIRS=\
    TSCLIB \
    TSCTOUCH \
    TSCWAVEDEV \
# @CESYSGEN IF CE_MODULES_POINTER
#   touch \
# @CESYSGEN ENDIF CE_MODULES_POINTER
# @CESYSGEN IF CE_MODULES_DEVICE
# @CESYSGEN IF CE_MODULES_USBD
    hcd   \
# @CESYSGEN ENDIF CE_MODULES_USBD
# @CESYSGEN IF CE_MODULES_SERIAL
    serial \
# @CESYSGEN ENDIF CE_MODULES_SERIAL
```

3. Save and close the modified **dirs** file.

Step VI: Update —

This step updates the Hardware Specific Files, so that the OS uses TSC2101 device drivers.

1. Open the existing **platform.reg** file from **Hardware Specific** section of the **ParameterView** window of the workspace.
2. Edit the **platform.reg** file so as to delete the old audio **dll** and to add in the TSC2101 audio:

```
; -----------------------------------------------------------------------------
; @CESYSGEN IF CE_MODULES_WAVEAPI
IF BSP_NOAUDIO !
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\WaveDev]
   "Prefix"="WAV"
```

```
;  "Dll"="pxa27x_wavedev.dll"
   "Dll"="wavedev.dll"
   "Index"=dword:1
   "Order"=dword:0
   "Priority256"=dword:95
   "Sysintr"=dword:19
```

3. Save and close the updated **platform.reg** file.
4. Similarly, edit the **platform.bib** file such as:

```
; -----------------------------------------------------------------------------
; @CESYSGEN IF CE_MODULES_WAVEAPI
IF BSP_NOAUDIO !
;    pxa27x_wavedev.dll     $(_FLATRELEASEDIR)\pxa27x_wavedev.dll    NK  SH
     wavedev.dll     $(_FLATRELEASEDIR)\wavedev.dll    NK  SH
ENDIF BSP_NOAUDIO !
; @CESYSGEN ENDIF CE_MODULES_WAVEAPI
; -----------------------------------------------------------------------------
```

5. Save and close the updated **platform.bib** file.

Step VII: Change –

This step changes one secondary interrupt of the GPIO0 from the AC97 link to $\overline{\text{PENIRQ}}$ (TSC2101 $\overline{\text{PINTDAV}}$).

1. At the menu **File\Open**…, navigate, find, and open the software code file intr.c inside the directory: C:\WINCE500\PLAFORM\MAINSTONEII\SRC\KERNEL\OAL\
2. Change the line in the **BSPIntrInit()** routine from:
   OALIntrStaticTranslate(SYSINTR_TOUCH, IRQ_GPIO0_UCB1400);
   Into:
   OALIntrStaticTranslate(SYSINTR_TOUCH, IRQ_GPIO0_PENIRQ);
3. Save and close the **intr.c** code file.

5 REFERENCES

1. *TSC2101 Touch Screen, Battery, and Audio WinCE Drivers* (SLAA200)
2. *TSC2301 WinCE Generic Drivers* (SLAA187)
3. *TSC2101EVM Touch Screen Controller Evaluation Module* (SLAU112)
4. Intel® PXA27x Processor Developer's Kit, Order#: 278827-005
5. *TSC2101, Audio CODEC with Integrated Headphone, Speaker Amplifier, and Touch Screen Controller* (SLAS392)
6. *Programming Audio Power Up/Down on TSC210x and TLV320AIC26/28* (SLAA230)

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