![Texas Instruments logo]

# TLV320AIC33 WinCE 5.0 Driver for Samsung S3C244x Processors

*Data Acquisition Products*

**ABSTRACT**

The TLV320AIC33 audio driver was developed with an I$^2$C™ control interface and I$^2$S™ audio streaming. The code was tested on an SC32442A Samsung application processor, running on the Microsoft Windows® CE 5.0 operating system. This application report discusses the I$^2$C and I$^2$S drivers, including the hardware connection between the TLV320AIC33EVM and the SC32442A Samsung application processor platform, the Windows CE 5.0 driver code and structure, and the respective installations.

**Contents**

## 1    Introduction

Texas Instruments' TLV320AIC33 (AIC33) audio device is a low-power, high-performance stereo input and stereo output coder/decoder (codec). This device is ideal for portable audio and telephony applications, in which an embedded operating system (OS), such as Windows CE (WinCE), often resides and operates. This application report discusses the driver for the AIC33 codec that was developed to enable users to quickly set up, run, and use the codec device with the WinCE 5.0 OS.

The AIC33 driver was coded on the standard device driver platform-dependent device (PDD) layer. The PDD layer was further split to have an additional processor-dependent layer (PDL) to make the drivers easy to port into different host processors. See Application Report *TSC2301 WinCE Generic Drivers* (SLAA187, available for download at www.ti.com) for details on Windows CE PDD and TI PDL generic drivers.

The WinCE 5.0 driver described in this document was run and tested on an TLV320AIC33EVM board and a Samsung platform with the SC32442A application processor.

## 2    Connections

The AIC33 device must be wired and connected to a host processor, where the device driver code is ported and executed. The two buses (or ports) for AIC33 operation are the control bus and the audio data bus. The control bus on the AIC33 is an I$^2$C bus. The audio data streams through the I$^2$S bus on the AIC33.

In developing the AIC33 drivers for this application, the TI AIC33EVM board and the Samsung platform with the SC32442A application processor (see Ref 4 ) were used.

On the I²C-controlled AIC33, the seven digital signals that are essential for running the audio driver are:

- the I²C bus, two wires: SCL and SDA (at J16 or J17 of the AIC33EVM board);
- the main audio codec clock, MCLK (at J17 of AIC33EVM board); and
- the I²S bus, four wires: BCLK, WCLK, SDIN and SDOUT (at J17 of AIC33EVM board).

Figure 1 shows the wires and connections between the AIC33 and SMDK2442 processor for the I²C control interface.
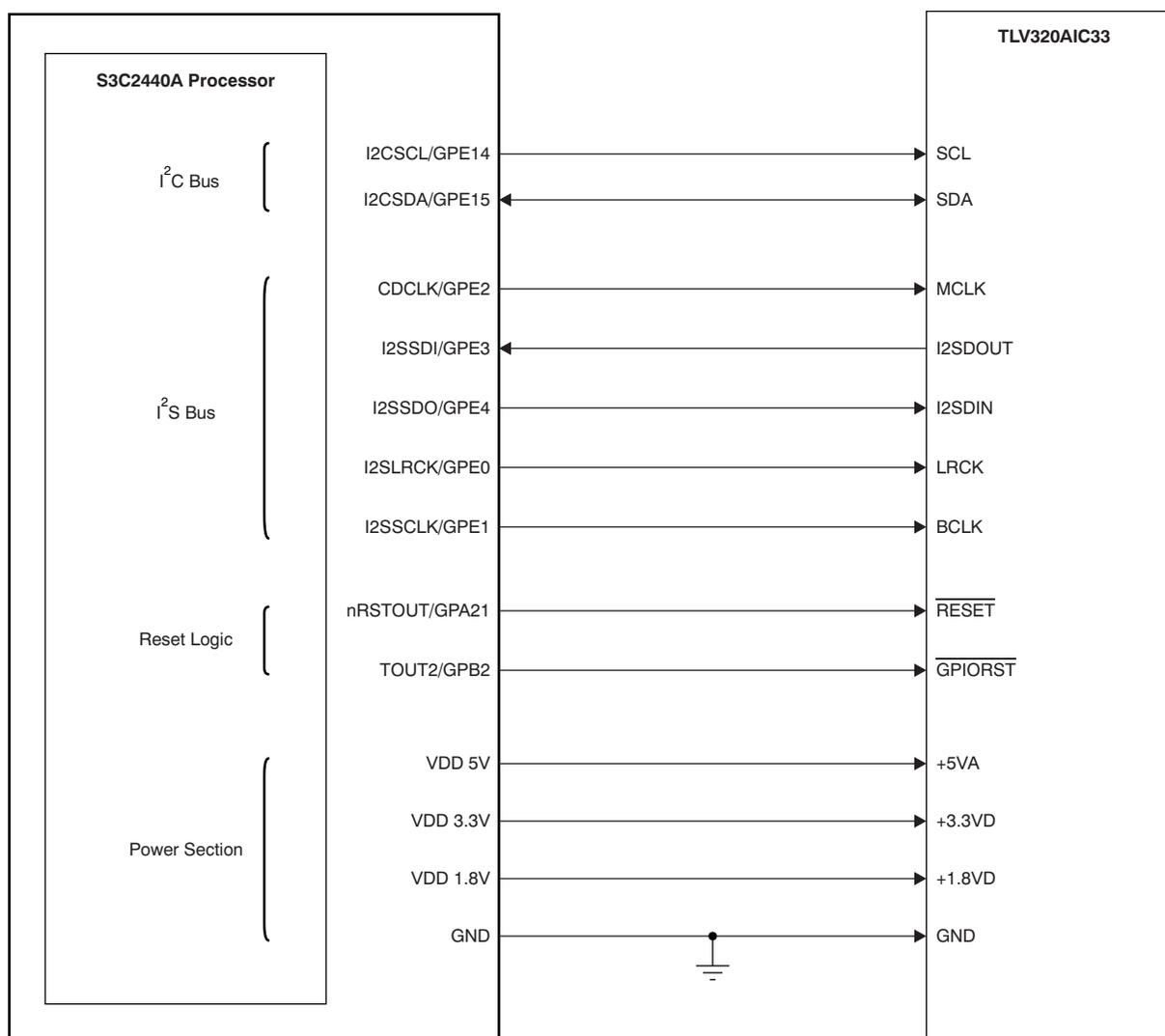


**Figure 1. TLV320AIC33 Connections to Samsung SC32440A Processor**

To implement the connection shown in Figure 1, ensure that these jumpers are correctly connected on the AIC33EVM board:

- Connect JMP10 between 2 and 3
- Connect JMP3 and JMP4 between 1 and 2
- Connect JMP9 between 1 and 2
- Connect JMP1 between 1 and 2
- Ensure that JMP11, JMP13, JMP14, and JMP15 are open
- Connect JMP12

This jumper configuration enables the internal MIC for recording and the HEADSET JACK for playing data from the codec.

The wiring diagram in Figure 2 describes the wiring details between the SMDK2442 interface and the AIC33.
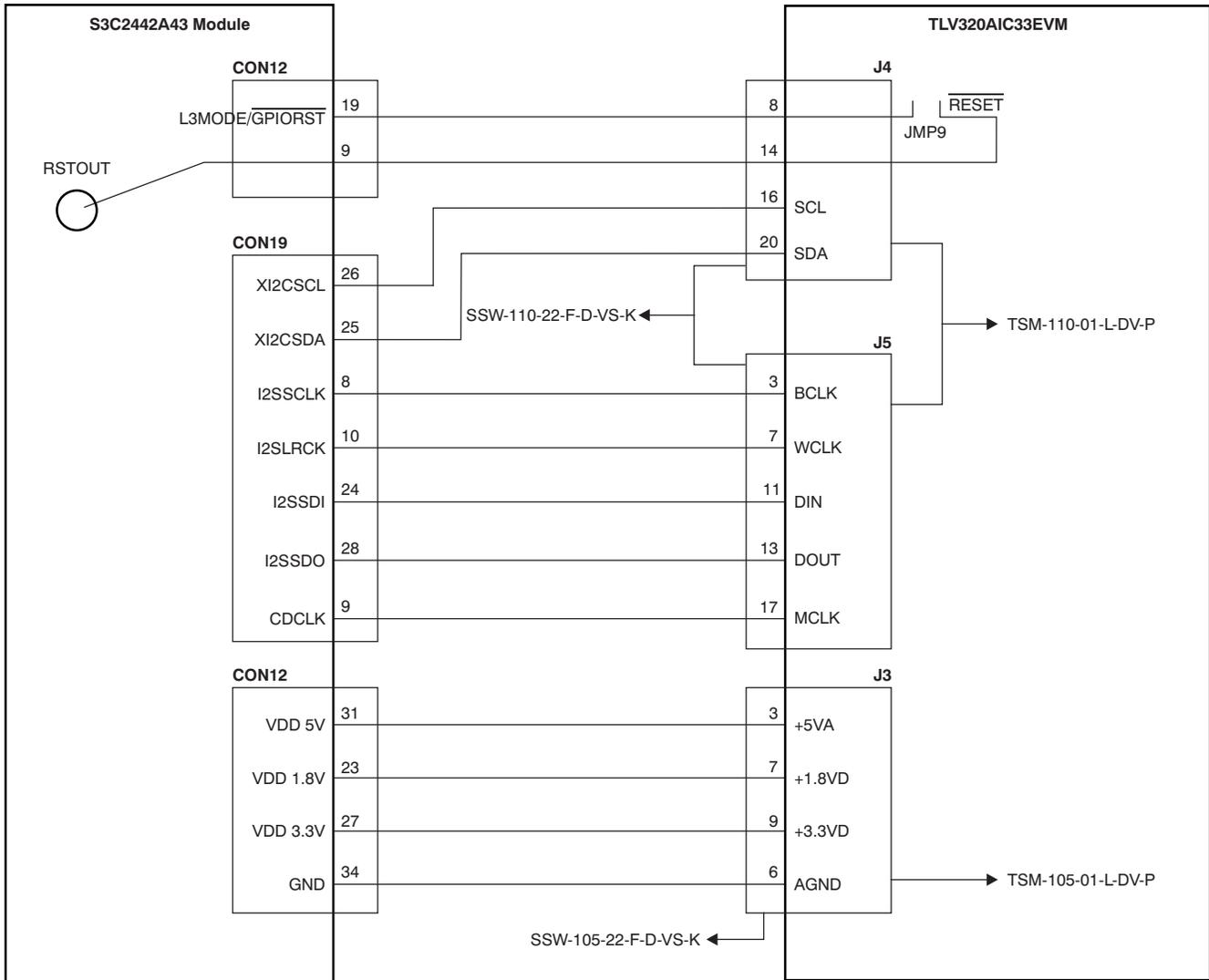


**Figure 2. TLV320AIC33EVM Connections to Samsung SMDK2440X Module**

See the TLV320AIC33EVM User's Guide (SBAU114, available for download at www.ti.com) for the schematic and other details of the EVM board. In Figure 2, the AIC33 is reset from two sources: RSTOUT and via a General Purpose Input/Output (GPIO) pin. Resetting from RSTOUT resets the AIC33 when the Samsung SMDK2442 processor powers on, putting the ACI33 into a known state. A reset via the GPIO Port B 2 pin is a response to a host processor instruction. Software can issue an active low pulse longer than 10ns in duration on this port pin to reset the AIC33. By setting JMP9 as directed, we are setting up the board to use the GPIO reset.

# 3    Device Driver

illustrates the locations of the AIC33 audio device driver files for both the SPI and I²C control interfaces. The files starting with **Host**… are the processor-dependent code or PDL, such as *HostAudio.C* or *HostI2CComm.H*.
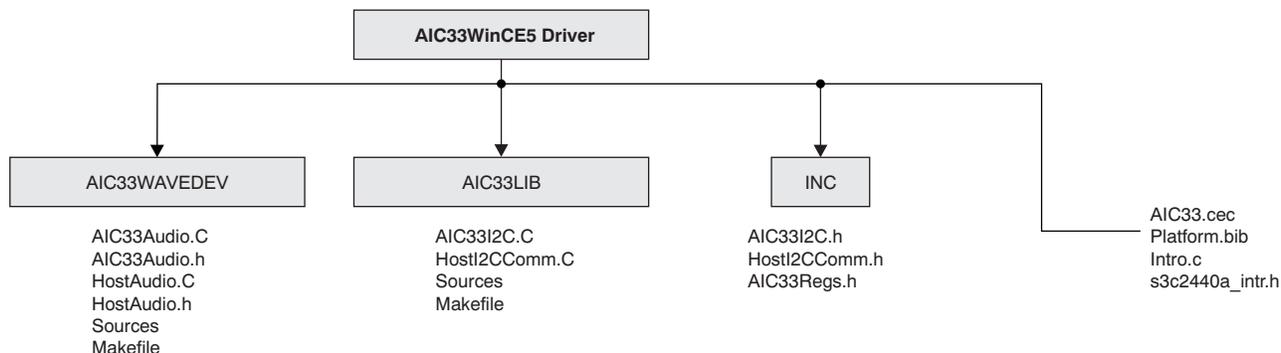


**Figure 3. AIC33 WinCE 5.0 Driver Files with I²C Control Interface**

## 3.1    I²C Interface

The two AIC33 I²C bus pins, SCL and SDA, are connected to the GPIO Port E 14 and GPIO Port E 15 of the SMDK2442 processor, respectively. On the host side, the SMDK2442 GPIO, I²C, and clock management control registers are used to set up the I²C interface to communicate with the AIC33 via the I²C interface. The *HWInitI2C ()* routine implements this set-up.

```
///////
// Function: void HWInitI2C(BOOL InPowerHandle)
// Purpose: This function must be called from the power handler
// of the respective drivers using this library. This
// function will configure the GPIO pins according to
// the functionality shown in the table below
// Signals Pin# Direction Alternate Function
// SCL GPE14 output 1
// SDA GPE15 output(at init) 1
///////
BOOL HWInitI2C(BOOL InPowerHandle)
{
UINT8 reg = 0x00;
RETAILMSG(1,(TEXT("Setup Host GPIO & I2C for an I2C Interface...\r\n")));
// init I2C control register (disabled I2C unit)
// enable I2C unit clock (the clock should be enabled first)
g_pClockRegs->CLKCON |= S3C_CLKEN_I2C;
// set up GPE
g_pGPIORegs->GPEDN |= GPE_DN; //0xc000, Pull-up
disable
g_pGPIORegs->GPECON |= (GPE14_IIC_SCL | GPE15_IIC_SDA); //Making GPE15=>IICSDA
, GPE14=>IICSCL
//Enable ACK, Prescaler IICCLK=PCLK/16, Enable interrupt, Transmit clock value Tx
clock=IICCLK/16
//e.g. If PCLK 50.7MHz, IICCLK = 3.17MHz, Tx Clock = 0.198MHz
reg = ICR_ACK | ICR_INTR;
reg &= ~(ICR_TXCLK);
reg |= ICR_TXCLKVAL;
g_pI2CRegs->IICCON = reg;
g_pI2CRegs->IICADD = 0x10; //2442 slave address
[7:1]
```

```
g_pI2CRegs->IICSTAT |= ISR_ENOP; //IIC bus data output
enable(Rx/Tx)
g_pI2CRegs->IICLC = ILCR_FEN | ILCR_SDADLY; // Filter
enable, 15 clocks SDA output delay
DumpRegsI2C();
return(TRUE);
}
```

Two other important I²C interface routines are the *HWI2CWriteRegs()* and *HWI2CReadRegs()*. These routines allow the SMDK2442 to write to or read from AIC33 control registers using the I²C bus. The I²C write and read protocols have been defined (see Figure 5 and Figure 6 of the TLV320AIC33 data sheet).

### HWI2CWriteRegs():

```
///////
// Function: HWI2CWriteRegs Routine
// Purpose: This routine allows the SMDK2442 to write to AIC33
// control register(s) using I2C bus.
// Note: The first byte in bytesBuf is the starting address
// for writing; and the 2nd and on are bytes/contents
// writing to AIC33
///////
BOOL HWI2CWriteRegs(UINT8 *bytesBuf, int bytesCount,
BOOL InPowerHandle)
{
if (!InPowerHandle)
{
UINT8 reg;
iicMod = WR_DATA;
iicPtr = 0;
iicDat[0] = *bytesBuf++; //Putting 1st byte i.e
register address
iicDat[1] = *bytesBuf; //Putting 2nd byte i.e.
actual data
iicDCount = bytesCount;
g_pI2CRegs->IICDS = I2C_WRITE; //Putting AIC33 slave
address (7bit address + 0 'write bit')
reg = g_pI2CRegs->IICSTAT;
reg = (ISR_MTX | ISR_START | ISR_ENOP); //Master transmit mode, START
signal genration, Enable output
g_pI2CRegs->IICSTAT = reg;
/*Clearing the pending bit isn't needed because the pending bit has been
cleared*/
while(iicDCount != -1)
Run_Iic_Poll();
iicMod = POLL_ACK;
while(1)
{
g_pI2CRegs->IICDS = I2C_WRITE;
iicStat = 0x100;
reg = g_pI2CRegs->IICSTAT;
reg = (ISR_MTX | ISR_START | ISR_ENOP);
//Master transmit mode, START signal genration, Enable output
g_pI2CRegs->IICSTAT = reg;
reg = g_pI2CRegs->IICCON;
reg = ICR_ACK | ICR_INTR | ICR_TXCLKVAL;reg &= ~(ICR_PENITR);
//Resumes IIC operation.
g_pI2CRegs->IICCON = reg;
while(iicStat==0x100)
Run_Iic_Poll();
if(!(iicStat & 0x1))
break;
//When ACK is received
}
```

```
g_pI2CRegs->IICSTAT = ~(ISR_STOP); //Stop
MasTx condition
reg = g_pI2CRegs->IICCON;
//Resumes IIC operation.
reg = ICR_ACK | ICR_INTR | ICR_TXCLKVAL;
reg &= ~(ICR_PENITR);
g_pI2CRegs->IICCON = reg;
Delay(3);
//Wait until stop condtion is in effect.
/*Write is completed.*/
return(TRUE);
}
else
{
RETAILMSG(1, (TEXT("HW Tx Error...\r\n")));
return(FALSE);
}
}
```

## HWI2CReadRegs():

```
///////
// Function: HWI2CReadRegs Routine
// Purpose: This routine allows the SMDK2442 to read from AIC33
// control register(s) using I2C bus.
// Note: The first byte in bytesBuf is the starting address for
// reading; and the 2nd and on are values reading from AIC33
///////
BOOL HWI2CReadRegs(UINT8 *bytesBuf, INT bytesCount,
BOOL InPowerHandle)
{
if (!InPowerHandle)
{
UINT8 reg;
iicMod = SETRD_ADDR;
iicPtr = 0;
iicDat[0] = *bytesBuf++; //Putting 1st
byte i.e. register address
iicDCount = 1;
g_pI2CRegs->IICDS = I2C_WRITE; //Putting
slave address of AIC33 for write mode [7:0]
Delay(1);
reg = g_pI2CRegs->IICSTAT;
reg = (ISR_MTX | ISR_START | ISR_ENOP); //Master transmit
mode, START signal genration, Enable output
g_pI2CRegs->IICSTAT = reg;
/*Clearing the pending bit isn't needed because the pending bit has been
cleared.*/
while(iicDCount!=-1)
Run_Iic_Poll();
iicMod = RD_DATA;
iicPtr = 0;
iicDCount = 1;
g_pI2CRegs->IICDS = I2C_READ; //Putting slave
address of AIC33 for read mode[7:1]
Delay(1);
reg = g_pI2CRegs->IICSTAT;
reg = (ISR_MRX | ISR_START | ISR_ENOP);
g_pI2CRegs->IICSTAT = reg; //Mater Rx, Start signal
reg = g_pI2CRegs->IICCON;
reg = ICR_ACK | ICR_INTR | ICR_TXCLKVAL ;
reg &= ~(ICR_PENITR);
g_pI2CRegs->IICCON = reg; //Resumes IIC operation.
while(iicDCount!=-1)
Run_Iic_Poll();
reg = g_pI2CRegs->IICCON;
reg = ICR_ACK | ICR_INTR | ICR_TXCLKVAL;
```

```
reg &= ~(ICR_PENITR);
g_pI2CRegs->IICCON = reg;
*bytesBuf++ = (UINT8) iicDat[1];
return(TRUE);
}
else
{
RETAILMSG(1, (TEXT("HW Rx Error...\r\n")));
return(FALSE);
}
}
```

## 3.2 Audio Driver

From a hardware standpoint, the AIC33 audio driver must have both I²C and I²S buses (for audio control and audio data streaming, respectively). The I²C bus controls the audio codec operation by writing to the AIC33 audio control registers; the I²S bus transfers audio data between the host and the AIC33. Additionally, the AIC33 MCLK pin should receive an external clock that provides the necessary timing for the AIC33 audio delta-sigma ($\Delta\Sigma$) ADC and DAC to operate. MCLK to the AIC33 should be generated from the same source as the I²S clocks; that is, MCLK should also run from the host processor, which is the I²S master as described in this application report. The AIC33 audio driver was built on the standard audio driver, WaveDev, and is located in the directory *AIC3xWaveDev*.

On the host side, the SMDK2442 GPIO GPE0 to GPE4 pins were used as the I²S source, and connected to the AIC33 WCLK, BCLK, MCLK, SDIN and SDOUT pins respectively (see Figure 1). The GPIO pin GPE2 is programmed as the I²S SYSCLK and is connected to MCLK, which is programmed to generate a 16.9344MHz clock. The I²S setup was implemented at the routine, *HWEnableI2S()*.

**HWEnableI2S():**

```
//
//-------------------------------------------------------------
// Processor Related Routines Used at AudioPowerOn() and
// AudioPowerOff(),
// which include: PDD_AudioInitialize(),
// PDD_AudioDeinitialize()
// and PDD_AudioPowerHandler().
//-------------------------------------------------------------
//
///////
// Function: HWEnableI2S()
///////
void HWEnableI2S(void)
{
RETAILMSG(1,(TEXT("+++HWEnableI2S\n")));
RETAILMSG(1,(TEXT("Setup Host GPIO & I2S Interface... \r\n")));
/* Basic Outline: */
/* Configue the GPIO registers and set to I2S mode
// Set up I2S control registers at default condition
/* Enable the CPU clock to the IIS controller */
v_pClockRegs->CLKCON |= IIS_INTERNAL_CLOCK_ENABLE;
/* Set up GPIO to route I2S signals */
//GPE4 - I2SSDO
//GPE3 - I2SSDI
//GPE2 - CDCLK
//GPE1 - I2SSCLK
//GPE0 - I2SLRCK
v_pGPIORegs->GPEDN |= 0x1f; //Disable
v_pGPIORegs->GPECON |= 0x2aa; // Select I2S
/* configure IIS registers */
//IISCON : Tx DMA REQ Enbl
// Rx DMA REQ Enbl
// Enable IIS Prescaler
// Disable IIS interface (stop)
v_pI2SRegs->IISCON = RECEIVE_DMA_REQUEST_ENABLE
TRANSMIT_DMA_REQUEST_ENABLE
```

```
IIS_PRESCALER_ENABLE;
//IISMOD : MPLLIN, IIS Master Mode, Tx and Rx Mode,Low for Left Ch, IIS Format,16
bit per channel,256fs, 32fs - IISCLK
// MASTER_CLOCL_PCLK |
//IIS_TRANSMIT_RECEIVE_MODE |
v_pI2SRegs->IISMOD = MASTER_CLOCL_MPLLIN |
IIS_MASTER_MODE |
ACTIVE_CHANNEL_LEFT |
SERIAL_INTERFACE_IIS_COMPAT |
DATA_16_BITS_PER_CHANNEL |
MASTER_CLOCK_FREQ_384fs |
SERIAL_BIT_CLOCK_FREQ_32fs;
//IISFCON: Tx FIFO:DMA, Rx FIFO: DMA, Enbl Tx FIFO, Enbl Rx FIFO
v_pI2SRegs->IISFCON = ( TRANSMIT_FIFO_ACCESS_DMA |
TRANSMIT_FIFO_ENABLE |
RECEIVE_FIFO_ACCESS_DMA |
RECEIVE_FIFO_ENABLE);
// Clock configuration; Set Prescaler register
//IISPSR
SetI2SClockRate((DWORD)IS2LRCLK_44100); // Set fs = =44.1kHz; Only freq
supported by Hardware
// Enable the I2S clock
v_pI2SRegs->IISCON |= IIS_INTERFACE_ENABLE;
DumpRegsGPIO();
DumpRegsClock();
DumpRegsI2S();
RETAILMSG(1,(TEXT("---HWEnableI2S\n")));
}
///////
// Function: HWDisableI2S()
///////
void HWDisableI2S(void)
{
RETAILMSG(1,(TEXT("+++HWDisableI2S\n")));
// disable I2S
v_pI2SRegs->IISCON &= ~(IIS_INTERFACE_ENABLE);
RETAILMSG(1,(TEXT("---HWDisableI2S\n")));
}
```

The codec can be used according to given application environments. As an example, for this application report, the AIC33 was initially configured in this manner:

- I$^2$S interface:
    1. The I$^2$S interface is at 16 bits, standard I$^2$S mode, with 44.1kHz ADC and DAC sample rates.
    2. The AIC33 is the slave because the host is the I$^2$S master (the AIC33 can be I$^2$S slave or master, but SMDK2442 can only be the master).
- Audio input circuitry:
    1. The left and right ADC input are from the stereo, single-ended LINE3 (MICIN3).
    2. ADC input gain is controlled by its PGA, with an initial gain setting of 0dB gain.
- Audio output circuitry:
    1. The left and right DAC outputs are routed to the stereo, single-ended headphone, HPL/R with HPLCOM and HPRCOM being shorted as the VCOM.
    2. Headphone output is in the CAPLESS mode.
    3. DAC gains and HPL/R output gains are all initialized to 0dB.
- Other functions:
    1. The input high-pass filter has not been enabled.
    2. The output digital boost, emphasis, and 3-D functions have not been enabled.
    3. PLL is disabled.
    4. The pop-reduction function is set to slowest rate and is enabled.

All AIC33 audio control registers (in Page0 of the AIC33 memory space) were set up or initialized, as previously stated, with the routine *InitAIC33Audio()* and called by the audio PDD routine, *PDD_AudioInitialize()*. The audio initialization routine is given below.

**Audio Initialization Routine:**

```
//
//----------------------------------------------------------------
// Audio Initialization
//----------------------------------------------------------------
//
// Initalize AIC33 Audio Register at Default
void InitAIC33Audio(BOOL bInPowerHandler)
{
/*The register which are not used in AIC33 are commentet out*/
RETAILMSG(1, (TEXT("InitAIC33Audio.\r\n")));
// init for digital functions
AIC33WriteReg(AIC33_RATE, RATE_INIT_VALUE, bInPowerHandler);
AIC33WriteReg(AIC33_PLLa, PLLa_INIT_VALUE, bInPowerHandler);
AIC33WriteReg(AIC33_PLLb, PLLb_INIT_VALUE, bInPowerHandler);
AIC33WriteReg(AIC33_PLLc, PLLc_INIT_VALUE, bInPowerHandler);
AIC33WriteReg(AIC33_PLLd, PLLd_INIT_VALUE, bInPowerHandler);
AIC33WriteReg(AIC33_DATAPATH, DATAPATH_INIT_VALUE, bInPowerHandler);
AIC33WriteReg(AIC33_INTERFa, INTERFa_INIT_VALUE, bInPowerHandler);
AIC33WriteReg(AIC33_INTERFb, INTERFb_INIT_VALUE, bInPowerHandler);
AIC33WriteReg(AIC33_INTERFc, INTERFc_INIT_VALUE, bInPowerHandler);
AIC33WriteReg(AIC33_DIGFILT, DIGFILT_INIT_VALUE, bInPowerHandler);
AIC33WriteReg(AIC33_HEDETb, HEDETb_INIT_VALUE, bInPowerHandler);
// init for analog input functions
AIC33WriteReg(AIC33_ADCPGAL, ADCPGAL_INIT_VALUE, bInPowerHandler);
AIC33WriteReg(AIC33_ADCPGAR, ADCPGAR_INIT_VALUE, bInPowerHandler);
AIC33WriteReg(AIC33_MIC3_ADCL, MIC3_ADCL_INIT_VALUE, bInPowerHandler);
AIC33WriteReg(AIC33_MIC3_ADCR, MIC3_ADCR_INIT_VALUE, bInPowerHandler);
AIC33WriteReg(AIC33_MICBIAS, MICBIAS_INIT_VALUE, bInPowerHandler);
// init for analog output functions
AIC33WriteReg(AIC33_OUTPWR, OUTPWR_INIT_VALUE, bInPowerHandler);
AIC33WriteReg(AIC33_OUTDRIVE, OUTDRIVE_INIT_VALUE, bInPowerHandler);
AIC33WriteReg(AIC33_OUTSTAGE, OUTSTAGE_INIT_VALUE, bInPowerHandler);
AIC33WriteReg(AIC33_OUTPOP, OUTPOP_INIT_VALUE, bInPowerHandler);
AIC33WriteReg(AIC33_DACLGAIN, DACLGAIN_INIT_VALUE, bInPowerHandler);
AIC33WriteReg(AIC33_DACRGAIN, DACRGAIN_INIT_VALUE, bInPowerHandler);
AIC33WriteReg(AIC33_DACL_HPL, DACL_HPL_INIT_VALUE, bInPowerHandler);
AIC33WriteReg(AIC33_HPLLEVEL, HPLLEVEL_INIT_VALUE, bInPowerHandler);
AIC33WriteReg(AIC33_DACR_HPR, DACR_HPR_INIT_VALUE, bInPowerHandler);
AIC33WriteReg(AIC33_HPRLEVEL, HPRLEVEL_INIT_VALUE, bInPowerHandler);
AIC33WriteReg(AIC33_PWRSTATUS, PWRSTATUS_HPRO_PWUP, bInPowerHandler);
AIC33WriteReg(AIC33_CLKGEN, CLKGEN_INIT_VALUE, bInPowerHandler);
RETAILMSG(1, (TEXT("Done InitAIC33Audio.\r\n")));
}
```

## 4  Installation

This section presents the installation steps for running the AIC33 WinCE 5.0 drivers on an SMDK2442 platform. The SC32442 Application Processor board support package (BSP) can be obtained from Samsung and installed on a PC. It is recommended to load the BSP after installing the Platform Builder 5.0 at (for example) **C:\WinCE500\PLATFORM\**.To install the AIC33 Windows CE 5.0 audio driver into one of the SMDK2442 workspaces, perform the following steps.

Step 1.  Copy:

    a.  Copy the file \AIC33WinCE5Driver\AIC33.cec to this location:
        C:\WINCE500\PUBLIC\COMMON\OAK\CATALOG\CEC\

    b.  Copy all files inside \AIC33WinCE5Drivers\INC\ into:
        C:\WINCE500\PLATFORM\SMDK2442\SRC\INC\

    c.  Copy the file \AIC33WinCE5Driver\intr.c into:
        C:\WINCE500\PLATFORM\SMDK2442\Src\Common\Intr

    d.  Copy the file \AIC33WinCE5Driver\s3c2440a_intr.h into:
        C:\WINCE500\PLATFORM\SMDK2442\Src\Inc

    e.  Copy the directories **AIC33LIB** and **AIC33WaveDev** into:
        C:\WINCE500\PLATFORM\SMDK2442\SRC\DRIVERS\

Step 2.  Set Up:

    a.  Run Platform Builder 5.0, and the Platform Builder IDE appears.

    b.  At the Platform Builder 5.0 IDE, open **Manage Catalog Items** from the menu *File\Manage CatalogItems …\*. When the Manage Catalog Items window appears, click the **Import** button on the right side of the window; navigate, find, and select **AIC33.cec** in the directory C:\WINCE500\PUBLIC\COMMON\OAK\CATALOG\CEC\. Then click on **Open** so that the item is ported in.

    c.  Click and drag to select all **\*.cec** files in the *Manage Catalog Items* window. Then click on the **Refresh** button to make sure the new item is loaded.

    d.  Close the *Manage Catalog Items* window by clicking **OK**.

This step sets up the catalog to include the AIC33 device drivers.

Step 3.  Open:

This step, in the Platform Builder 5.0 IDE, opens a new or existing SMDK2442 workspace according to the application instructions. This procedure is ignored here.

Step 4.  Add:

    a.  In the *Catalog* window of the Platform Builder 5.0 IDE, find the TI AIC33 Audio CODEC Driver, right-click on it, and select **Add to OS Design** to add the audio driver to the OS.

    b.  The audio device driver should appear under the Device Drivers section at the *OSDesignView* window of the WorkSpace.

This step ports the AIC33 device drivers from the Catalog into the existing OS design.

5. Modify:

    a. Open the **dirs** file in the directory:
       C:\WINCE500\PLATFORM\SMDK2442\SRC\DRIVERS\

    b. Add on the **AIC3xLIB** and **AIC3xWAVEDEV**.
       For example, the **dirs** file could be:

```
DIRS=\
ceddk\
keybd\
PowerButton\
pccard\
serial\
usb\
nleddrvr\
Battdrvr\
Backlight\
cs8900\
Display\
SDHC\
touch\
wavedev\
AIC33LIB\
AIC33WAVEDEV
```

This step modifies the building device drivers in order to include TI AIC33 drivers.

Step 5.  Update:

    a. Open the existing platform.reg file from **Hardware Specific** section of the *ParameterView* window of the workspace.

    b. Edit the platform.reg file; delete the old audio .dll and add in the AIC33 audio dll file:

```
IF BSP_NOAUDIO !
; @CESYSGEN IF CE_MODULES_WAVEAPI
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\Audio]
"Prefix"="WAV"
"Dll"="wavedev.dll"
"Index"=dword:1
"Order"=dword:0
"Priority256"=dword:d2
; @CESYSGEN ENDIF CE_MODULES_WAVEAPI
ENDIF BSP_NOAUDIO !
```

    c. Save and close the updated platform.reg file.

    d. Edit the platform.bib file in the same manner:

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

    e. Save and close the updated platform.bib file.

This step updates the Hardware Specific Files, so that the operating system will use AIC33 device drivers.

## 5    WinCE 5.0 Driver Code

To obtain the driver code discussed in this application report, contact the TI Applications Support Group at: support@ti.com.

## 6 References

The following documents are available for download through the Texas Instruments web site (www.ti.com), except where noted.

1. Chammings, Y. and Fang, W.X. (2003.). *TSC2301 WinCE Generic Drivers*. Application report SLAA187.
2. TLV320AIC33: Low Power Stereo Audio Codec for Portable Audio/Telephony. Product data sheet SLAS480.
3. *TLV320AIC33EVM User's Guide*. User guide SBAU113.
4. Samsung SC32442A Processor Developer's Kit. User guide.

# IMPORTANT NOTICE