Freescale Semiconductor, Inc.
Application Note

Document Number: AN3663
Rev. 3.0, 2/2014

SGTL5000 Initialization and Programming

1 Description

The objective of this document is to familiarize the reader with the code and methodology required to initialize and program the SGTL5000 using C code. The first section describes the programming prototypes needed to access the SGTL5000's registers. The next section is a guide to initializing the SGTL5000 out of reset, and configuring various features. Programming examples are provided, intended for the end-user.

Contents

2 Programming Examples

This section provides programming examples that show how to configure SGTL5000. The registers can be written/read by using I^{2}C communication protocol. SGTL5000 also supports SPI communication protocol but only register write operation is supported.

2.1 Prototype for Reading and Writing a Register

The generic register read write prototype will be used throughout this section as shown below. The I^{2}C or SPI implementation will be specific to the I2C/SPI hardware used in the system.\

```
// This prototype writes a value to the entire register. All
// bit-fields of the register will be written. Write REGISTER REGISTERVALUE

// This prototype writes a value only to the bit-field specified.
// In the actual implementation, the other bit-fields should be
// masked to prevent them from being written. Also, the
// actual implementation should left-shift the BITFIELDVALUE
// by appropriate number to match the starting bit location of
// the BITFIELD.
Modify REGISTER -> BITFIELD, BITFIELDVALUE //Bitfield Location

// Example implementation
// Modify DAP_EN (bit 0) bit to value 1 to enable DAP block
Modify( DAP_CONTROL_REG, 0xFFFE, 1 << DAP_EN_STARTBIT );

// Example Implementation of Modify
void Modify( unsigned short usRegister,
     unsigned short usClearMask,
     unsigned short usSetValue )
{
unsigned short usData;

// 1) Read current value
ReadRegister( usRegister, &usData );

// 2) Clear out old bits
usData = usData & usClearMask;

// 3) set new bit values usData = usData | usSetValue;

// 4) Write out new value created
WriteRegister( usRegister, usData );
}
```

2.2 Chip Configuration

All outputs (LINEOUT, HP_OUT, I2S_OUT) are muted by default on powerup. To avoid any pops/clicks, the outputs should remain muted during these chip configuration steps. Refer to **Section 2.2.6, Volume Control**, page 8 for volume and mute control.

2.2.1 Initialization

2.2.1.1 Chip Powerup and Supply Configurations

After the power supplies for chip is turned on, following initialization sequence should be followed. Please note that certain steps may be optional or different values may need to be written based on the power supply voltage used and desired configuration. The initialization sequence below assumes VDDIO = 3.3 V and VDDA = 1.8 V.

```
//--------------- Power Supply Configuration----------------
// NOTE: This next 2 Write calls is needed ONLY if VDDD is
// internally driven by the chip
// Configure VDDD level to 1.2V (bits 3:0) Write CHIP_LINREG_CTRL 0x0008
// Power up internal linear regulator (Set bit 9) Write CHIP_ANA_POWER 0x7260

// NOTE: This next Write call is needed ONLY if VDDD is
// externally driven
// Turn off startup power supplies to save power (Clear bit 12 and 13) Write CHIP_ANA_POWER
0x4260

// NOTE: The next 2 Write calls is needed only if both VDDA and
// VDDIO power supplies are less than 3.1V.
// Enable the internal oscillator for the charge pump (Set bit 11) Write CHIP_CLK_TOP_CTRL
0x0800
// Enable charge pump (Set bit 11) Write CHIP_ANA_POWER 0x4A60

// NOTE: The next modify call is only needed if both VDDA and
// VDDIO are greater than 3.1V
// Configure the charge pump to use the VDDIO rail (set bit 5 and bit 6) Write CHIP_LINREG_CTRL
0x006C

//------ Reference Voltage and Bias Current Configuration----------
// NOTE: The value written in the next 2 Write calls is dependent
// on the VDDA voltage value.
// Set ground, ADC, DAC reference voltage (bits 8:4). The value should
// be set to VDDA/2. This example assumes VDDA = 1.8V. VDDA/2 = 0.9V.
// The bias current should be set to 50% of the nominal value (bits 3:1) Write CHIP_REF_CTRL
0x004E
// Set LINEOUT reference voltage to VDDIO/2 (1.65V) (bits 5:0) and bias current (bits 11:8) to
the recommended value of 0.36mA for 10kOhm load with 1nF capacitance
Write CHIP_LINE_OUT_CTRL 0x0322

//----------------Other Analog Block Configurations------------------
// Configure slow ramp up rate to minimize pop (bit 0) Write CHIP_REF_CTRL 0x004F

// Enable short detect mode for headphone left/right
// and center channel and set short detect current trip level
// to 75mA
Write CHIP_SHORT_CTRL 0x1106

// Enable Zero-cross detect if needed for HP_OUT (bit 5) and ADC (bit 1) Write CHIP_ANA_CTRL
0x0133

//----------------Power up Inputs/Outputs/Digital Blocks-------------
// Power up LINEOUT, HP, ADC, DAC Write CHIP_ANA_POWER 0x6AFF

// Power up desired digital blocks
// I2S_IN (bit 0), I2S_OUT (bit 1), DAP (bit 4), DAC (bit 5),
// ADC (bit 6) are powered on
Write CHIP_DIG_POWER 0x0073

//--------------------Set  LINEOUT Volume Level-----------------------
// Set the LINEOUT volume level based on voltage reference (VAG)
// values using this formula
// Value = (int)(40*log(VAG_VAL/LO_VAGCNTRL) + 15)
// Assuming VAG_VAL and LO_VAGCNTRL is set to 0.9V and 1.65V respectively, the
// left LO volume (bits 12:8) and right LO volume (bits 4:0) value should be set
// to 5
Write CHIP_LINE_OUT_VOL 0x0505
```

2.2.1.2 System MCLK and Sample Clock

```
// Configure SYS_FS clock to 48kHz
// Configure MCLK_FREQ to 256*Fs
Modify CHIP_CLK_CTRL->SYS_FS 0x0002 // bits 3:2
Modify CHIP_CLK_CTRL->MCLK_FREQ  0x0000 // bits 1:0

// Configure the I2S clocks in master mode
// NOTE: I2S LRCLK is same as the system sample clock
Modify CHIP_I2S_CTRL->MS 0x0001 // bit 7
```

2.2.2 PLL Configuration

These programming steps are needed only when the PLL is used. Please refer to the SGTL5000 datasheet for details on when to use the PLL.

To avoid any pops/clicks, the outputs should be muted during these chip configuration steps. Refer to **Section 2.2.6, Volume Control**, page 8 for volume and mute control.

```
// Power up the PLL
Modify CHIP_ANA_POWER->PLL_POWERUP  0x0001 // bit 10
Modify CHIP_ANA_POWER->VCOAMP_POWERUP  0x0001 // bit 8

// NOTE: This step is required only when the external SYS_MCLK
// is above 17MHz. In this case the external SYS_MCLK clock
// must be divided by 2
Modify CHIP_CLK_TOP_CTRL->INPUT_FREQ_DIV2 0x0001 // bit 3
```

```
Sys_MCLK_Input_Freq = Sys_MCLK_Input_Freq/2;

// PLL output frequency is different based on the sample clock
// rate used.
    if (Sys_Fs_Rate == 44.1kHz) PLL_Output_Freq = 180.6336MHz
else
    PLL_Output_Freq = 196.608MHz

// Set the PLL dividers
Int_Divisor = floor(PLL_Output_Freq/Sys_MCLK_Input_Freq)
Frac_Divisor = ((PLL_Output_Freq/Sys_MCLK_Input_Freq) - Int_Divisor)*2048
Modify CHIP_PLL_CTRL->INT_DIVISOR  Int_Divisor // bits 15:11
Modify CHIP_PLL_CTRL->FRAC_DIVISOR  Frac_Divisor // bits 10:0
```

2.2.3 Input/Output Routing

To avoid any pops/clicks, the outputs should be muted during these chip configuration steps. Refer to **Section 2.2.6, Volume Control**, page 8 for volume and mute control.

A few example routes are shown below:

```
// Example 1: I2S_IN -> DAP -> DAC -> LINEOUT, HP_OUT

// Route I2S_IN to DAP
Modify CHIP_SSS_CTRL->DAP_SELECT  0x0001 // bits 7:6
// Route DAP to DAC
Modify CHIP_SSS_CTRL->DAC_SELECT  0x0003 // bits 5:4
// Select DAC as the input to HP_OUT
Modify CHIP_ANA_CTRL->SELECT_HP  0x0000 // bit 6

// Example 2: MIC_IN -> ADC -> I2S_OUT

// Set ADC input to MIC_IN
Modify CHIP_ANA_CTRL->SELECT_ADC  0x0000 // bit 2
// Route ADC to I2S_OUT

Modify CHIP_SSS_CTRL->I2S_SELECT  0x0000 // bits 1:0

// Example 3: LINEIN -> HP_OUT

// Select LINEIN as the input to HP_OUT
Modify CHIP_ANA_CTRL->SELECT_HP  0x0001 // bit 6
```

2.2.4 Digital Audio Processor Configuration

To avoid any pops/clicks, the outputs should be muted during these chip configuration steps. Refer to **Section 2.2.6, Volume Control**, page 8 for volume and mute control.

```
// Enable DAP block
// NOTE: DAP will be in a pass-through mode if none of DAP
// sub-blocks are enabled.
Modify DAP_CONTROL->DAP_EN 0x0001 // bit 0
```

2.2.4.1 Dual Input Mixer

These programming steps are needed only if dual input mixer feature is used.

```
// Enable Dual Input Mixer
Modify DAP_CONTROL->MIX_EN 0x0001 // bit 4

// NOTE: This example assumes mix level of main and mix
// channels as 100% and 50% respectively

// Configure main channel volume to 100% (No change from input
// level)
Write DAP_MAIN_CHAN 0x4000

// Configure mix channel volume to 50% (attenuate the mix
// input level by half) Write DAP_MIX_CHAN 0x4000
```

2.2.4.2 SGTL Surround

The SGTL Surround on/off function will be typically controlled by the end-user. End-user driven programming steps are shown in **Section 2.3, End-user Driven Chip Configuration**, page 8.

The default WIDTH_CONTROL of 4 should be appropriate for most applications. This optional programming step shows how to configure a different width value.

```
// Configure the surround width
// (0x0 = Least width, 0x7 = Most width). This example shows
// a width setting of 5
Modify DAP_SGTL_SURROUND->WIDTH_CONTROL 0x0005 // bits 6:4
```

2.2.4.3 SGTL Bass Enhance

The SGTL Bass Enhance on/off function will be typically controlled by the end-user. End-user driven programming steps are shown in **Section 2.3, End-user Driven Chip Configuration**, page 8.

The default LR_LEVEL value of 0x0005 results in no change in the input signal level and BASS_LEVEL value of 0x001F adds some harmonic boost to the main signal. The default settings should work for most applications. This optional programming step shows how to configure a different value.

```
// Gain up the input signal level
Modify DAP_BASS_ENHANCE_CTRL->LR_LEVEL 0x0002 // bits 7:4

// Add harmonic boost
Modify DAP_BASS_ENHANCE_CTRL->BASS_LEVEL 0x003F); // bits 6:0
```

2.2.4.4 7-Band Parametric EQ / 5-Band Graphic EQ / Tone Control

Only one audio EQ block can be used at a given time. The psuedocode in this section shows how to select each block.

Some parameters of the audio EQ will typically be controlled by end-user. End-user driven programming steps are shown in **Section 2.3, End-user Driven Chip Configuration**, page 8.

```
// 7-Band PEQ Mode
// Select 7-Band PEQ mode and enable 7 PEQ filters
Write DAP_AUDIO_EQ 0x0001
Write DAP_PEQ 0x0007

// Tone Control mode
Write DAP_AUDIO_EQ 0x0002

// 5-Band GEQ Mode
Write DAP_AUDIO_EQ 0x0003
```

2.2.4.5 Automatic Volume Control (AVC)

The AVC on/off function will be typically controlled by the end-user. End-user driven programming steps are shown in **Section 2.3, End-user Driven Chip Configuration**, page 8.

The default configuration of the AVC should work for most applications. However, the following example shows how to change the configuration if needed.

```
// Configure threshold to -18dB Write DAP_AVC_THRESHOLD 0x0A40

// Configure attack rate to 16dB/s
Write DAP_AVC_ATTACK 0x0014

// Configure decay rate to 2dB/s

Write DAP_AVC_DECAY 0x0028
```

2.2.5 I^2S Configuration

By default the I^2S port on the chip is configured for 24-bits of data in I^2S format with SCLK set for 64*Fs. This can be modified by setting various bit-fields in CHIP_I2S_CTRL register.

2.2.6 Volume Control

The outputs should be unmuted after all the configuration is complete.

```
//---------------- Input Volume Control---------------------
// Configure ADC left and right analog volume to desired default.
// Example shows volume of 0dB Write CHIP_ANA_ADC_CTRL 0x0000

// Configure MIC gain if needed. Example shows gain of 20dB Modify CHIP_MIC_CTRL->GAIN 0x0001
// bits 1:0

//---------------- Volume and Mute Control---------------------
// Configure HP_OUT left and right volume to minimum, unmute
// HP_OUT and ramp the volume up to desired volume. Write CHIP_ANA_HP_CTRL 0x7F7F
Modify CHIP_ANA_CTRL->MUTE_HP 0x0000 // bit 4

// Code assumes that left and right volumes are set to same value
// So it only uses the left volume for the calculations usCurrentVolLeft = 0x7F;
usNewVolLeft = usNewVol & 0xFF;
usNumSteps = usNewVolLeft - usCurrentVolLeft;
if (usNumSteps == 0) return;

// Ramp up
for (int i = 0; i < usNumSteps; i++ )
{
++usCurrentVolLeft;
usCurrentVol = (usCurrentVolLeft << 8) | (usCurrentVolLeft); Write CHIP_ANA_HP_CTRL
usCurrentVol;
}

// LINEOUT and DAC volume control
Modify CHIP_ANA_CTRL->MUTE_LO 0x0000 // bit 8
// Configure DAC left and right digital volume. Example shows
// volume of 0dB
Write CHIP_DAC_VOL 0x3C3C
Modify CHIP_ADCDAC_CTRL->DAC_MUTE_LEFT 0x0000 // bit 2

Modify CHIP_ADCDAC_CTRL->DAC_MUTE_RIGHT 0x0000 // bit 3

// Unmute ADC
Modify CHIP_ANA_CTRL->MUTE_ADC 0x0000 // bit 0
```

2.3 End-user Driven Chip Configuration

End-users will control features like volume up/down, audio EQ parameters such as Bass and Treble. This will require programming the chip without introducing any pops/clicks or any disturbance to the output. This section shows examples on how to program these features.

2.3.1 Volume and Mute Control

Refer to **Section 2.2.6, Volume Control**, page 8 for examples on how to program volume when end-user changes the volume or mutes/unmutes output. Note that the DAC volume ramp is automatically handled by the chip.

2.3.2 7-Band PEQ Preset Selection

This programming example shows how to load the filter coefficients when the end-user changes PEQ presets such as Rock, Speech, Classical etc.

```
// Load the 5 coefficients for each band and write them to
// appropriate filter address. Repeat this for all enabled
// filters (this example shows 7 filters)
for (i = 0; i < 7; i++)
{
// Note that each 20-bit coefficient is broken into 16-bit MSB
//   (unsigned short usXXMSB) and 4-bit LSB (unsigned short
//    usXXLSB)
Write DAP_COEF_WR_B0_LSB usB0MSB[i] Write DAP_COEF_WR_B0_MSB usB0LSB[i] Write
DAP_COEF_WR_B1_LSB usB1MSB[i] Write DAP_COEF_WR_B1_MSB usB1LSB[i] Write DAP_COEF_WR_B2_LSB
usB2MSB[i] Write DAP_COEF_WR_B2_MSB usB2LSB[i] Write DAP_COEF_WR_A1_LSB usA1MSB[i] Write
DAP_COEF_WR_A1_MSB usA1LSB[i] Write DAP_COEF_WR_A2_LSB usA2MSB[i] Write DAP_COEF_WR_A2_MSB
usA2LSB[i]
// Set the index of the filter (bits 7:0) and load the
// coefficients
Modify DAP_FILTER_COEF_ACCESS->INDEX  (0x0101 + i) // bit 8
}
```

2.3.3 5-Band GEQ Volume Change

This programming example shows how to program the GEQ volume when end-user changes the volume on any of the 5 bands.

GEQ volume should be ramped in 0.5 dB steps in order to avoid any pops. The example assumes that volume is ramped on Band 0. Other bands can be programmed similarly.

```
// Read current volume set on Band 0 usCurrentVol = Read DAP_AUDIO_EQ_BASS_BAND0

// Convert the new volume to hex value usNewVol = 4*dNewVolDb + 47;

// Calculate the number of steps
usNumSteps = abs(usNewVol - usCurrentVol);
if (usNumSteps == 0) return;
for (int i = 0; i++; usNumSteps )
{
if (usNewVol > usCurrentVol)
     ++usCurrentVol;
else
     --usCurrentVol;

Write DAP_AUDIO_EQ_BASS_BAND0 usCurrentVol;
}
```

2.3.4 Tone Control - Bass and Treble Change

This programming example shows how to program the Tone Control Bass and Treble when end-user changes it on the fly.

Tone Control Bass and Treble volume should be ramped in 0.5 dB steps in order to avoid any pops. The example assumes that Treble is changed to a new value. Bass can be programmed similarly.

```
// Read current Treble value
usCurrentVal = Read DAP_AUDIO_EQ_TREBLE_BAND4

// Convert the new Treble value to hex value usNewVal = 4*dNewValDb + 47;

// Calculate the number of steps
usNumSteps = abs(usNewVal - usCurrentVal);
if (usNumSteps == 0) return;
for (int i = 0; i++; usNumSteps )
{
if (usNewVal > usCurrentVal)
    ++usCurrentVal;
else
--usCurrentVal;
Write DAP_AUDIO_EQ_TREBLE_BAND4  usCurrentVal;
}
```

2.3.5 SGTL Surround On/Off

This programming example shows how to program the Surround when end-user turns it on/off on his device.

The Surround width should be ramped up to highest value before enabling/disabling the Surround to avoid any pops.

```
// Read current Surround width value
// WIDTH_CONTROL bits 6:4
usOriginalVal = (Read DAP_SGTL_SURROUND >> 4) && 0x0003;
usNextVal = usOriginalVal;

// Ramp up the width to maximum value of 7 for (int i = 0; i++; (7 - usOriginalVal)
{
++usNextVal;
Modify DAP_SGTL_SURROUND->WIDTH_CONTROL usNextVal;
}

// Enable (To disable, write 0x0000) Surround
// SELECT bits 1:0
Modify DAP_SGTL_SURROUND->SELECT  0x0003;

// Ramp down the width to original value for (int i = 0; i++; (7 - usOriginalVal)
{
--usNextVal;
Modify DAP_SGTL_SURROUND->WIDTH_CONTROL usNextVal;
}
```

2.3.6 Bass Enhance On/Off

This programming example shows how to program the Bass Enhance on/off when end-user turns it on/off on his device. The Bass level should be ramped down to the lowest Bass before Bass Enhance feature is turned on/off.

```
// Read current Bass level value
// BASS_LEVEL bits 6:0
usOriginalVal = Read DAP_BASS_ENHANCE_CTRL && 0x007F;
usNextVal = usOriginalVal;

// Ramp Bass level to lowest bass (lowest bass = 0x007F)

Conclusion

usNumSteps = abs(0x007F - usOriginalVal);
for (int i = 0; i++; usNumSteps )
{
++usNextVal;
Modify DAP_BASS_ENHANCE_CTRL->BASS_LEVEL usNextVal;
}

// Enable (To disable, write 0x0000) Bass Enhance
// EN bit 0
Modify DAP_BASS_ENHANCE->EN 0x0001;

// Ramp Bass level back to original value for (int i = 0; i++; usNumSteps )
{
--usNextVal;
Modify DAP_BASS_ENHANCE_CTRL->BASS_LEVEL usNextVal;
}
```

2.3.7 Automatic Volume Control (AVC) On/Off

This programming example shows how to program the AVC on/off when end-user turns it on/off on his device.

```
// Enable AVC (To disable, write 0x0000)
Modify DAP_AVC_CTRL->EN 0x0001 // bit 0
```

3 Conclusion

Using this document as a guide, the reader will have the basis for accessing registers, and initializing and configuring SGTL5000 out of reset.

4 Revision History

Revision	Date	Description
2.0	11/2008	• Initial release
3.0	2/2014	• Change section 2.2.1.1 NOTE for CHIP_LINREG_CTRL from 2 modify calls to one call • Change section 2.2.6 MUTE_HP bit number from 5 to 4 • Corrected a note in Chip Powerup and Supply Configurations

Document Number: AN3663
Rev. 3.0
2/2014

freescale™