

# **Guideline for Integrated SmartAMP Baremetal Driver & Factory Calibration Tool**

# Revision history

Ver	Date	Author	Description
1.0	2020/12/18		Initial
1.1	2021/5/4		Involve the factory calibration tool
1.2	2025/6/5		Add exTas256x_regdump, exTas256x_setvol, exTas256x_getvol illustration
1.3	2025/7/22		Add how to set factory temperature into the calibration tool.

# PreWork

- Before porting, kindly check the size of code section and data section for tas2563.
- Resource needed for tas2563 MCU projects(mono):
  - the minimum size of prm optimized is 9184 Bytes
  - each profile algo param is 1600 bytes
  - MCU reference code including two-profile algo param (Music and Voice) will be about 10K bytes(K=1024).

# Introduction I

- Reference Code used for MCU platform
  - Driver code
  - FCT code (kindly consult FAE for the code)
  - Support multiple PAs application, especially the burn-in test for many speakers in speaker vendor
- C files introduction:

File name	Remark
OSL_wrapper.c	Platform-dependency file, should be implemented by customers according to defined interfaces
tasdevice.c	Register R/W for all audio chips
tas2563_ftc.c	Factory test for calibration, f0, etc, <b>kindly consult FAE for the code</b>
tas2563_interface.c	External interfaces for MCU

# Introduction II | makefile

File name	Remark
makefile.i2c_mono	Mono/i2c/32-bit
makefile.i2c_mono_RX16	Mono/i2c/16-bit
makefile.i2c_mono_RX16_prm_in_nv	Mono/i2c/16-bit/PDM/prm_in_nv
makefile.i2c_pdm	Mono/i2c/32-bit/pdm
makefile.i2c_stereo	Stereo/i2c/32-bit
makefile.i2c_stereo_RX16	Stereo/i2c/16-bit
makefile.spi_pdm	Mono/spi/32-bit/PDM
makefile.spi_stereo	Stereo/spi/32-bit
makefile.spi_woofer_RX16	2.1channel/spi/16-bit

# Introduction III | Header files in inc/coef

- All these files are the default ones, can be substituted with new one released by tuning engineer.

File name	Remark
cfg0_music_COEFF_prim.h	Music acoustic params for channel 1
cfg0_pdm_music_COEFF.h	Music acoustic params for channel 1 on PDM project
cfg1_calibration_COEFF.h	Calibration params for all channels
cfg1_pdm_calibration_COEFF.h	Calibration params for PDM project
cfg2_Voice_COEFF_prim.h	Voice acoustic params for channel 1

# Introduction IV

- Header files in inc/prm\_reg

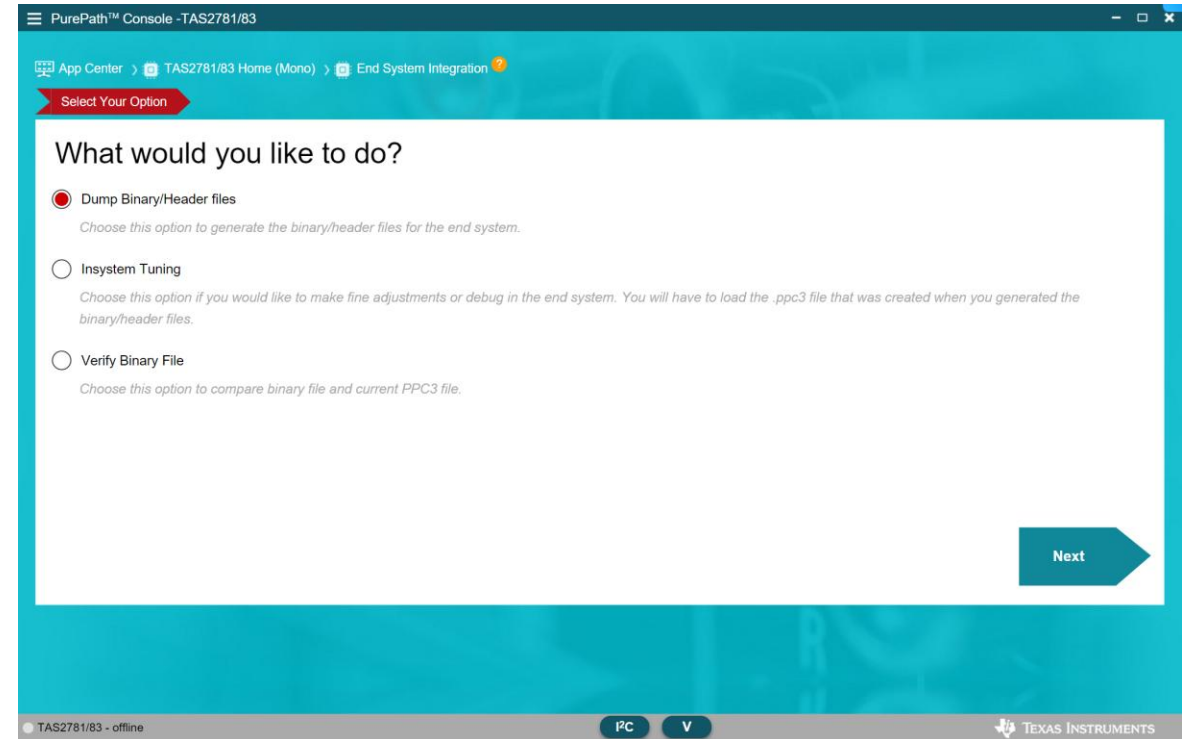
PS: all these files are the default ones, can be substituted

File name	Remark
prm_pdm_tuning_mode.h	DSP firmware for PDM project
prm_tuning_mode.h	DSP firmware
register_setting.h	Register setting

- TAS2563\_ftcfg.h ---- store the speaker characterization, used for calibration, file in different project should be different. **Usually, tuning engineer or FAE will release this file for the specific project.** File in Release is the default values.
- Prm.bin ---- the binary file for prm\_tuning\_mode.h, normally save it into flash for some limited storage projects. If this has been enabled, prm\_tuning\_mode.h will be deactivated automatically

# Introduction V ---- how to generate the header files

- `prm_tunning_mode_registers[]` in `prm_xxx.h`, `cfgxxxx[]` in `cfgxxx.h` and `TAS2563_ftcfg[]` in `TAS2563_ftcfg.h` in this patch are all default settings. Kindly replace them with the proper settings for your project.
- Apply for the PPC3 tool([PUREPATHCONSOLE Application software & framework | TI.com](https://www.ti.com/tool/purepathconsole)), all these settings are generated via PPC3 tools.





# Introduction VI ---- header files I

PurePath™ Console - TAS2781/83

App Center > TAS2781/83 Home (Mono) > End System Integration ?

Select Your OptionDump Binary File

Summary

The binary files are generated with the following configuration

I2C slave address : 0x70Name of DDC : sample

Application	Configuration Name	Sampling Frequencies	Clock Configuration	Snapshots
Tuning Mode	music	48 KHz	Auto	2 Snapshot 2 (Base)

Click here to open the folder location of generated binary/header files

Finish










TAS2781/83 - offline

I2CV

TEXAS INSTRUMENTS

# Introduction VII ---- header files II

- In the Header folder:
  1. TAS2563\_ftcfg[] is stored in .ftcfg file, customer will input them into TAS2563\_ftcfg.h manually.
  2. cfgxxxx[] is stored in Configuration\_xxxxx\_COEFF.h
  3. prm\_tunning\_mode\_registers[] is stored in program\_0\_Tuning Mode.h

Name	
	configuration_0_music_TuningMode_DEV_A_COEFF.h
	configuration_0_music_TuningMode_DEV_A_POST_SOFTWARE_RESET.h
	configuration_0_music_TuningMode_DEV_A_PRE.h
	configuration_0_music_TuningMode_DEV_A_PRE_SOFTWARE_RESET.h
	configuration_1_calibration_TuningMode_48KHz_DEV_A_COEFF.h
	configuration_1_calibration_TuningMode_48KHz_DEV_A_POST_SOFTWARE_RESET.h
	configuration_1_calibration_TuningMode_48KHz_DEV_A_PRE.h
	configuration_1_calibration_TuningMode_48KHz_DEV_A_PRE_SOFTWARE_RESET.h
	program_0_Tuning Mode.h

# Wrapper Layer I | Data Structure

- These interfaces defined in the Wrapper data structure **would be implemented according to different platforms by developer.**

```
struct os_interface {
    int (*dev_read)(void* dev_handle, int reg);
    int (*dev_write)(void* dev_handle, int reg, unsigned int Value);
    int (*dev_bulk_write)(void* dev_handle, int reg, int len, unsigned char *pData);
    int (*dev_bulk_read)(void* dev_handle, int reg, int len, unsigned char* pData);
    void (*GPIO_config)();
    void (*msleep)(unsigned int msecs);
    size_t (*nv_write)(const void *ptr, size_t size, size_t nmemb, void *nv_handle);
    size_t (*nv_read)(void *ptr, size_t size, size_t nmemb, void *nv_handle);
#ifdef PRM_IN_NV
    // in order to save the code section, store the prm into flash
    unsigned char* (*prm_download)(void *handle, int chn, unsigned int *len);
    void (*prm_remove)(unsigned char *pData);
#endif
};
```

# Wrapper Layer II | Register operation Interface Introduction I

- `int (* dev_read) (void *dev_handle, int reg);`
  - Illustration: Single byteread
  - Input arguments
    - ✓ void \*dev\_handle: device context
    - ✓ int reg: register number
  - Return Value
    - ✓ Failure: < 0
    - ✓ Success: >= 0
- `int (* dev_write) (void *dev_handle, int reg, unsigned int Value);`
  - Illustration: Single byte write
  - Input arguments
    - ✓ void \*dev\_handle: device context
    - ✓ int reg: register number
    - ✓ unsigned int Value: the value to be written
  - Return Value
    - ✓ Failure: != 0
    - ✓ Success: == 0
- `int (* dev_bulk_write) (void *dev_handle, int reg, int len, unsigned char *pData);`
  - Illustration: Multiple bytes write
  - Input arguments
    - ✓ void \*dev\_handle: device context
    - ✓ int reg: register number
    - ✓ int len: the size of the pData
    - ✓ unsigned char \*pData: Pointer to data to be written
  - Return Value
    - ✓ Failure: != 0
    - ✓ Success: == 0

# Wrapper Layer II | Register operation Interface Introduction II

- `int (*dev_bulk_read)(void* dev_handle, int reg, int len, unsigned char* pData)`
  - Illustration: Multiple bytes write
  - Input arguments
    - ✓ `void* dev_handle`: device context
    - ✓ `int reg`: register number
    - ✓ `int len`: the size of the `pData`
    - ✓ `unsigned char *pData`: Storage location for data
  - Return Value
    - ✓ Failure: `!= 0`
    - ✓ Success: `== 0`

# Wrapper Layer III | Platform Interface Introduction I

- `void (*GPIO_config)()`
  - Illustration: GPIO setting, including RESET-pin and I2C\_SPI-pin, called during initialization
  - Return Value: None
- `void (*msleep)(unsigned int msecs)`
  - Illustration: sleep safely
  - Input arguments: unsigned int msecs: Time in milliseconds to sleep for
  - Return Value: None
- `size_t (*nv_read)(void *ptr, size_t size, size_t nmemb, void *nv_handle);`
  - Illustration: Reads data from a storage device.
  - Input arguments
    - ✓ void\* nv\_handle: Pointer to storage structure
    - ✓ const void \*ptr: Storage location for data
    - ✓ size\_t size: Item size in bytes
    - ✓ size\_t nmemb: Maximum number of items to be read
  - Return Value: returns the number of full items actually read, which may be less than nmemb if an error occurs or if the end of the file is encountered before reaching nmemb
- `size_t (*nv_write)(const void *ptr, size_t size, size_t nmemb, void *nv_handle);`
  - Illustration: Writes data to a storage device.
  - Input arguments
    - ✓ void\* nv\_handle: Pointer to storage structure
    - ✓ const void \*ptr: Point to data to be written
    - ✓ size\_t size: Item size, in bytes
    - ✓ size\_t nmemb: Maximum number of items to be written
  - Return Value: returns the number of full items actually read, which may be less than nmemb if an error occurs or if the end of the file is encountered before reaching nmemb

# Wrapper Layer III | Platform Interface Introduction II

- `void (*GPIO_config)()`
  - Illustration: GPIO setting, including RESET-pin and I2C\_SPI-pin, called during initialization
  - Return Value: None
- `void (*msleep)(unsigned int msecs)`
  - Illustration: sleep safely
  - Input arguments: unsigned int msecs: Time in milliseconds to sleep for
  - Return Value: None
- `size_t (*nv_read)(void *ptr, size_t size, size_t nmemb, void *nv_handle);`
  - Illustration: Reads data from a storage device.
  - Input arguments
    - ✓ void\* nv\_handle: Pointer to storage structure
    - ✓ const void \*ptr: Storage location for data
    - ✓ size\_t size: Item size in bytes
    - ✓ size\_t nmemb: Maximum number of items to be read
  - Return Value: returns the number of full items actually read, which may be less than nmemb if an error occurs or if the end of the file is encountered before reaching nmemb
- `size_t (*nv_write)(const void *ptr, size_t size, size_t nmemb, void *nv_handle);`
  - Illustration: Writes data to a storage device.
  - Input arguments
    - ✓ void\* nv\_handle: Pointer to storage structure
    - ✓ const void \*ptr: Point to data to be written
    - ✓ size\_t size: Item size, in bytes
    - ✓ size\_t nmemb: Maximum number of items to be written
  - Return Value: returns the number of full items actually read, which may be less than nmemb if an error occurs or if the end of the file is encountered before reaching nmemb

# Wrapper Layer III| Platform Interface Introduction III

- This slide mainly introduce the interfaces specially for dsp program stored in flash instead of array in the header file in order to save code section
  - `unsigned char* (*prm_download)(void *handle, int chn, unsigned int *len);`
    - **Illustration:** Reads dsp program from a storage device, such as flash or SD card.
    - **Input arguments**
      - ✓ `void* handle`: a handle can store the smartamp info or anything else, it is defined by customers
      - ✓ `int chn`: it is still defined by customers
    - **Output arguments:** `unsigned int *len`: the len of dsp program, it is the output param.
    - **Return Value:** returns the point of memory stored the dsp program. This interface support different can download different dsp programs
  - `void (*prm_remove)(unsigned char *pData);`
    - **Illustration:** release the memory allocated by `prm_download`.
    - **Input arguments:** `unsigned char *pData`: Pointer to the memory allocated by `prm_download`
    - **Return Value:** None



# Interfaces for external use I | SmartAMP basic operation I

- **int exTas256x\_init(unsigned char \*prmData, int len)**
  - **Illustration:** Initialization, no requirement for pcm clk.
  - **Input arguments**
    - ✓ unsigned char \*prmData: Algorithm Program Data
    - ✓ int len: the size of prmData  
PS: both of the two arguments are used for the solution that algorithm program are stored into flash instead of array.
  - **Return Value**
    - ✓ Failure: != 0
    - ✓ Success: == 0
- **void exTas256x\_deinit()**
  - **Illustration:** Destroy the context of tas256x
  - **Input arguments:** None
  - **Return Value:** None
- **void exTas256x\_speakeroff()**
  - **Illustration:** power off tas256x, **after returning, pcm clk(both fysnc clk and bck) can be off.**
  - **Input arguments:** None
  - **Return Value:** None

# Interfaces for external use II | SmartAMP basic operation II

- **int exTas256x\_speakeron(unsigned int profile)**
  - **Illustration:** Switch the profile, and power on tas256x, **before calling, pcm clk(both fysnc clk and bck) must be on.**
  - **Input arguments**
    - ✓ unsigned int profile: supported profile id, customers can add their own profiles according to the requirements
      - MUSIC
      - VOICE
      - CALIBRATION
      - BYPASS
      - INDEPENDANT\_CAPTURE, program binary must support PDM
      - MIXTURE\_CAPTURE, program binary must support PDM
  - **Return Value**
    - ✓ Failure: != 0
    - ✓ Success: == 0

# Interfaces for external use III | SmartAMP basic operation III

- **int exTas256x\_setvol(unsigned char level);**
  - **Illustration:** set tas2563 digital gain.
  - **Input arguments**
    - ✓ unsigned char level: the value is from 0 ~ 255, its corresponding DB is -121.5 db ~ 6db.
  - **Return Value**
    - ✓ Failure: != 0
    - ✓ Success: == 0
- **int exTas256x\_getvol(unsigned char \*level);**
  - **Illustration:** get tas2563 digital gain.
  - **Input/ Output arguments**
    - ✓ unsigned char \*level:
      1. the argument must be the valid address for unsinged char
      2. the value is from 0 ~ 255, its corresponding DB is -121.5 db ~ 6db.
  - **Return Value**
    - ✓ Failure: != 0
    - ✓ Success: == 0

# Interfaces for external use III | SmartAMP basic operation IV

- **int exTas256x\_regdump(unsigned char book, unsigned char page);**
  - **Illustration:** interrupt handling routine
  - **Input arguments:**
    - ✓ unsigned char book: the book in which the register bank is
    - ✓ unsigned char page: the page at which the register bank is
  - **Return Value**
    - ✓ Failure: != 0
    - ✓ Success: == 0
- **void exTas256x\_irq(void);**
  - **Illustration:** interrupt handling routine
  - **Input arguments:** None
  - **Return Value:** None

PS: This interface should be implemented by customers.

# Interfaces for external use IV | SmartAMP Calibration I

- **void exTas256x\_calib\_start(char \*spk\_vendor);**
  - **Illustration:** Start calibration during playing silence
  - **Input arguments:** char \*spk\_vendor: the name of speaker vendor; inputting NULL or mismatched speaker name will load default speaker setting
  - **Return Value:** None
- **void exTas256x\_calib\_stop(void);**
  - **Illustration:** Stop calibration started by exTas256x\_calib\_start. Usually, after calling exTas256x\_calib\_start after **2~3 seconds**, call this interface to stop. That is exTas256x\_calib\_start and exTas256x\_calib\_stop should be called in pair.
    - This interface contain saving calibrated data into NV part, which would be implemented by the customer. See nv\_write introduction in [Wrapper Layer III | Platform Interface Introduction II](#)
  - **Input arguments:** None
  - **Return Value:** None

# Interfaces for external use V | SmartAMP Calibration II

- **Calibration Calling procedure**

1. exTas256x\_init
2. exTas256x\_speakeron(CALIBRATION)
3. Playing silence
4. exTas256x\_calib\_start
5. sleep 2~3 seconds (keep silence being played until **step 7**)
6. exTas256x\_calib\_stop
7. Stop playing
8. exTas256x\_speakeroff
9. exTas256x\_deinit

# Interfaces for external use VI | SmartAMP f0 read

- **int exTas256x\_get\_f0(unsigned int \*f0\_array);**
  - **Illustration:** Get the f0 to check whether the speaker is leakage or blocked
  - **Input arguments:** pointer the array to save the f0 value
  - **Return Value:** None
  - **Calling procedure**
    - exTas256x\_init
    - exTas256x\_speakeron(CALIBRATION)
    - Playing -15db pink noise
    - exTas256x\_get\_f0
    - sleep 5 seconds
    - Stop playing
    - exTas256x\_speakeroff
    - exTas256x\_deinit

# Macro definition I

- I2C\_ENABLE ---- If defined, use I2C interface; If not, use SPI interface.
- PDM\_ENABLE ---- If defined, PDM recording enable; If not, PDM recording disable.
- RX\_16BIT ---- If defined, rx channel is 16bit, else is 32bit
- PRM\_IN\_NV ---- If defined, algorithm program is stored into NV part, it won't take up any part of code section, widely used in small-code-section project; If not, algorithm program is stored in an array, it will take up code section.



# Macro definition II | frequently used macro settings

	I2C_ENABLE	PDM_ENABLE	PRM_IN_NV	RX_16BIT
Mono/i2c/32-bit/prm_in_nv	✓	x	✓	x
Mono/spi/16-bit/PDM	x	✓	x	✓
Mono/i2c/16-bit/PDM	✓	✓	x	✓
Mono/i2c/16-bit	✓	x	x	✓

# Macro definition III | factory temperature

- Change the value of **DEFAULT\_ENV\_TEMPERATURE** to the real factory temperature.

- e.g 22.0, 25.0, etc

The screenshot shows an IDE with several open files. The active file is `tas2563_interface.h`, which contains the following code:

```
58: * OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
59: * BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
60: * LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
61: * NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
62: * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
63: */
64:
65: #ifndef _TAS2563_INTERFACE_H_
66: #define _TAS2563_INTERFACE_H_
67:
68: #define DEFAULT_ENV_TEMPERATURE» (20.0)
69: #define CALIBRATION_COUNT» (1)
70:
71: enum spk_vendor_list {
72: »    GOERTEK,
73: »    SW
74: };
75:
76: typedef struct {
77: »    int mnPGID;
78: »    int mProfileId;
79: »    bool bPrmOptimized;
80: } dspfw_runtime_info;
81:
82: int exTas256x_init(void);
83: void exTas256x_deinit();
84: int exTas256x_speakeron(unsigned int profile);
85: void exTas256x_speakeroff();
```

The IDE interface includes a toolbar at the top, a project file list on the right, and a status bar at the bottom. The status bar indicates the current position is Line 68, Column 1, and the file encoding is UTF-8.

# Size of Code section and Data section (Verified on STM32-F429)

	Conf Sum for application	Dsp firmware stored into flash or not	Calibration profile sum	Code section (byte)	Data section (byte)
Mono/i2c/32-bit/prm_in_nv	2	Y	1	19142	5236
Mono/spi/16-bit/PDM	3	N	1	20140	14056
Mono/i2c/16-bit/PDM	3	N	1	19022	14052
Mono/i2c/16-bit	2	N	1	18878	14404

# Appendix I | suggestions on exTas256x\_irq I

- Due to different requirements on irq handling with different mcu platform. Here provide some basic procedures on irq handling:
  1. First define which irq the project needed, suggest
    - B0P0R0x1Abit2: TDM clock error mask bit, set 0 as unmask/enable; 1 as mask/disable
    - B0P0R0x1Abit1: Over current error mask bit, set 0 as unmask/enable; 1 as mask/disable
  2. For multiple-smartamp projects, confirm the connection of reset-pin on samrtamps. If all the reset-pins share the same GPIO, hardware reset will cause all the amps reset; If not, hardware reset can control the needed amp.
  3. IRQ handling
    - I. Mask/Disable irq
    - II. Set following variable to initial value instead of exTas256x\_deinit
    - III. `pTAS256X->tasdevice[chn].mPrm = -1;`
    - IV. `pTAS256X->tasdevice[chn].mProfileId = -1;`

# Appendix I | suggestions on exTas256x\_irq II

- V. pTAS256X->tasdevice[chn].mnCurrentBook = -1;
- VI. pTAS256X->tasdevice[chn].mnCurrentPage = -1;
- VII. Hardware reset the chip if GPIO to reset-pin is available; Skip this step, if not.
- VIII. Downloading the prm.
- IX. Call exTas256x\_speakeron

PS: If same interrupt triggers constantly. Constant resets are not a wise way, and will cause not only a bad user experience, but also make the efficiency of the whole system too poor. We suggest to set the max time of reset, and look into the root cause of the interrupt issue.

# Appendix II | How to add a new profile into the code

- Pls reference the MUSIC profile related code as example
  - Define a new profile id into profileId\_t
  - Add the cfg header file into the MCU code studio
  - Add code branch into tas256x\_dspon/ tas256x\_powerup

***Thanks!***