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

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, **kindly consult FAE for the code**
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_4pas	4chan/spi/32-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_music_COEFF_quat.h	Music acoustic params for channel 4
cfg0_music_COEFF_sec.h	Music acoustic params for channel 2
cfg0_music_COEFF_tert.h	Music acoustic params for channel 3
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
cfg2_Voice_COEFF_quat.h	Voice acoustic params for channel 4
cfg2_Voice_COEFF_sec.h	Voice acoustic params for channel 2
cfg2_Voice_COEFF_tert.h	Voice acoustic params for channel 3

Introduction IV

- Header files in inc/prm_reg

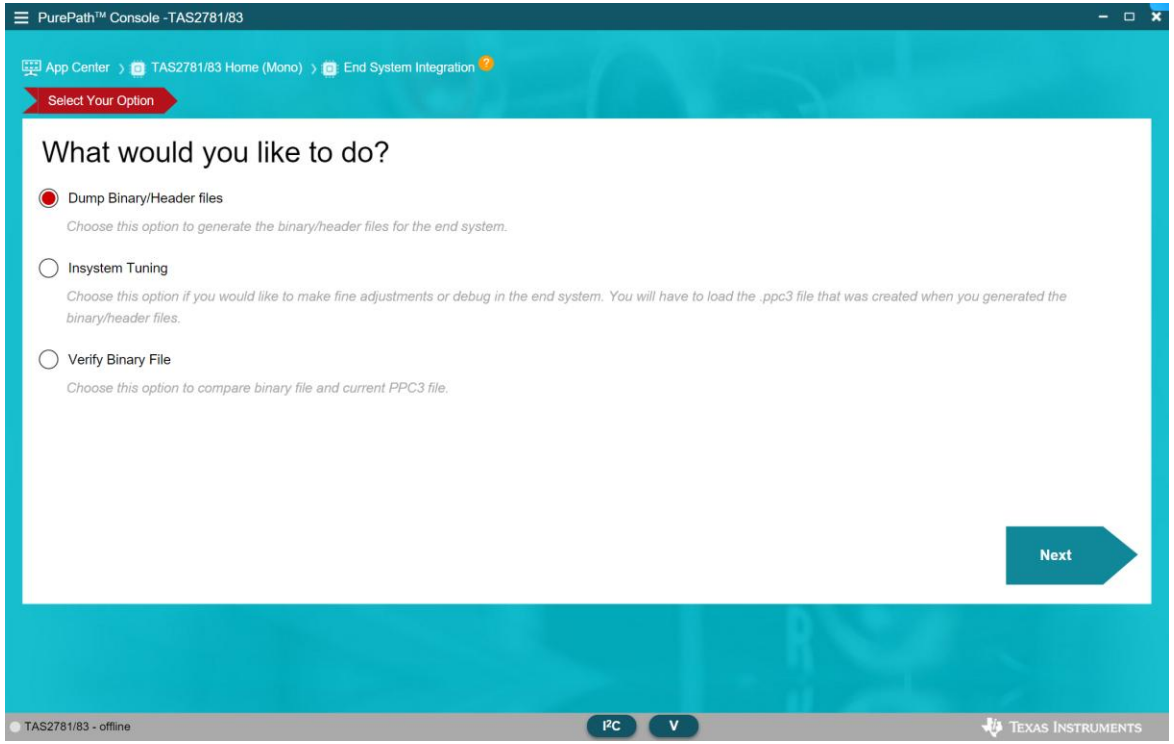
  PS: all these files are the default ones, can be substituted

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

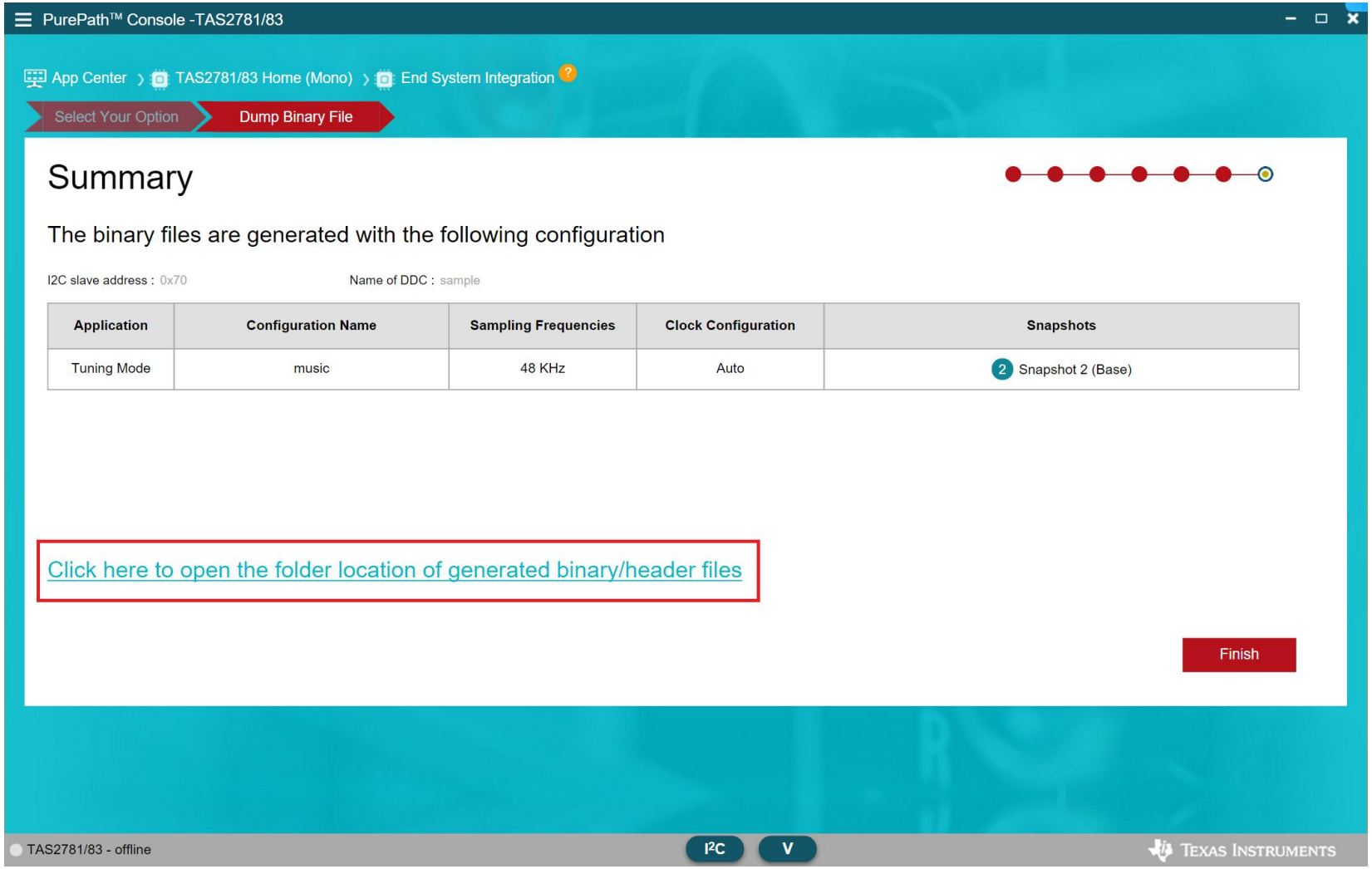
- TAS2563_ftcfg.h ---- store the speaker characterization, used for calibration, file in different project should be different. **Usually, tuning engineer or FAE will release this file for the specific project.** File in Release is the default values.
- Prm.bin ---- the binary file for prm_tuning_mode.h, 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), all these settings are generated via PPC3 tools.

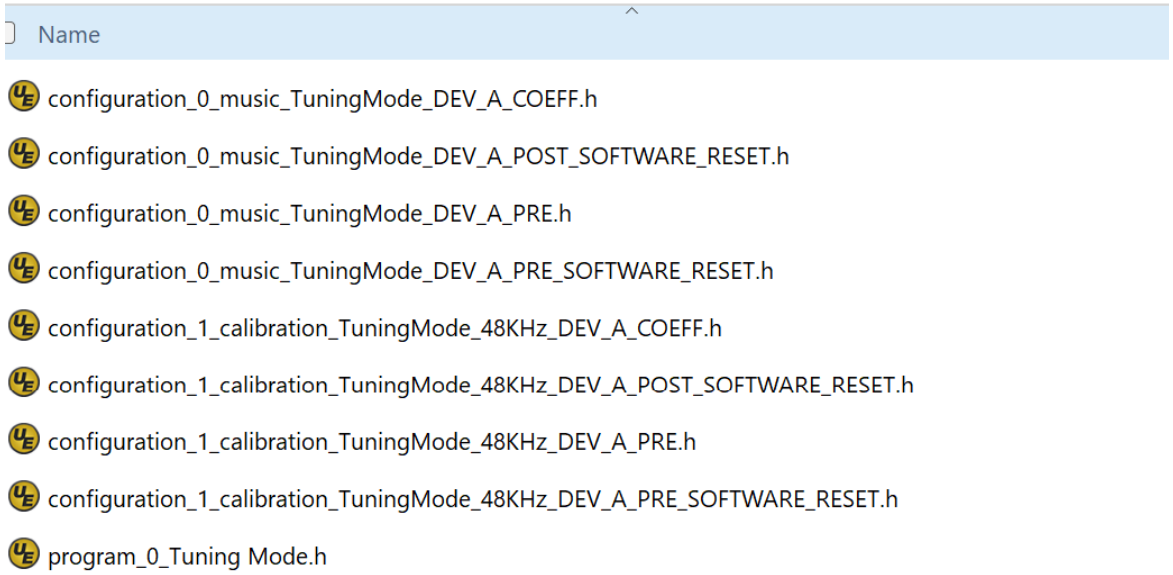


Introduction VI ---- header files I



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



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_handl e, 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
  - **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
  - **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
  - **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

- **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; inputing 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
  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
- FOUR_PAS ---- four Smartamps
- WOOFER_TWEETERS ---- 2.1 channel Smartamps
- Stereo ---- duel channel Smartamps
- 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	STEREO	WOOFER_TWEETRS	FOUR_PAS
Mono/i2c/32-bit/prm_in_nv	✓	×	✓	×	×	×	×
Mono/spi/16-bit/PDM	×	✓	×	✓	×	×	×
Mono/i2c/16-bit/PDM	✓	✓	×	✓	×	×	×
Mono/i2c/16-bit	✓	×	×	✓	×	×	×
Stereo/i2c/32-bit	✓	×	×	×	✓	×	×
2.1channel/spi/16-bit/prm_in_nv	×	×	✓	✓	×	✓	×
4channel/spi/16-bit/prm_in_nv	✓	×	×	×	×	×	✓

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
Stereo/i2c/32-bit	2*2slots=4	N	1	19029	17928
2.1channel/spi/16-bit/prm_in_nv	2*3slots=6	Y	1	20479	11896
4channel/spi/16-bit/prm_in_nv	2*4slots=8	N	1	19236	24376

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!