```
#ifndef __AUDIO_RECORD_H__
#define __AUDIO_RECORD_H__

#include "application.h"
//=============================================================
// audio encode configure set
//=============================================================
#define ADC_LINE_IN_EN          1
#if ((defined MCU_VERSION) && (MCU_VERSION < GPL326XX))
#define BUILD_IN_MIC_EN         0
#else
#define BUILD_IN_MIC_EN         1
#endif
#define GPY0050_IN_EN       0

// input device
#define ADC_LINE_IN             (1 << 0)
#define BUILD_IN_MIC        (1 << 1)
#define GPY0050_IN              (1 << 2)
#define DOUBLE_LINEIN       (1 << 3)

// for adc and mic use
#define C_ADC_USE_TIMER                 ADC_AS_TIMER_C
    //adc:ADC_AS_TIMER_C ~ F, mic: ADC_AS_TIMER_C ~ F
#define C_ADC_USE_CHANNEL               ADC_LINE_1              //adc
channel: 0 ~ 3

// for GPY0050 use
#define C_AUDIO_RECORD_TIMER            TIMER_B
    //timer, A,B,C
#define C_GPY0050_SPI_CS_PIN        IO_F5                   //gpy0500
spi interface cs pin

#define C_AUD_PCM_BUFFER_NO             3                       //pcm
buffer number
#define C_A1800_RECORD_BUFFER_SIZE      A18_ENC_FRAMESIZE
    //PCM buffer size, fix 320
#define C_WAVE_RECORD_BUFFER_SIZE       1024*16
```

```
    //PCM buffer size, depend on SR=16KHz
#define C_BS_BUFFER_SIZE                1024*32                 //file
buffer size, fix 64Kbyte
#define C_VR_RECORD_BUFFER_SIZE         512*2



#define A1800_TIMES                     30
    //a1800 encode times, 320*30, <80*30, depend on SR=16KHz
#define ADPCM_TIMES                     16
    //adpcm encode times, 500*16, 256*16, depend on SR=16KHz
#define MP3_TIME                        15                      //MP3
encode times, 1152*15

// pcm energy detect threshold
#define PCM_GLOBAL_THR                  0x800
    //Frame Energy Threshold to judge as active
#define PCM_LOCAL_THR                   0x800
    //Local Energy Threshold of each sample

// adc record and dac play at same time
#define RECORD_WITH_DAC_OUT_EN          0                       //1:
enable, 0: disable

//============================================================
// audio encode status
//============================================================
#define C_GP_FS             0
#define C_USER_DEFINE       1

#define C_MONO_RECORD       1
#define C_STEREO_RECORD     2

#if DBG_MESSAGE == 1
    #define DEBUG_MSG(x) {x;}
#else
    #define DEBUG_MSG(x) {}
#endif
```

```
#define C_STOP_RECORD             0x00000000
#define C_STOP_RECORDING 0x00000001
#define C_START_RECORD            0x00000002
#define C_START_FAIL          0x80000001

#define AUD_RECORD_STATUS_OK                    0x00000000
#define AUD_RECORD_STATUS_ERR                   0x80000001
#define AUD_RECORD_INIT_ERR                     0x80000002
#define AUD_RECORD_DMA_ERR                      0x80000003
#define AUD_RECORD_RUN_ERR                      0x80000004
#define AUD_RECORD_FILE_WRITE_ERR               0x80000005
#define AUD_RECORD_MEMORY_ALLOC_ERR             0x80000006

//gpy0500 command
#define C_CMD_RESET_IN1                 0x83
#define C_CMD_RESET_IN4                 0x89
#define C_CMD_ENABLE_ADC                0x98
#define C_CMD_ENABLE_MIC_AGC_ADC    0x9B
#define C_CMD_ENABLE_MIC_ADC            0x9B
#define C_CMD_ENABLE_MIC_AGC            0x93
#define C_CMD_DUMMY_COM                     0xC0
#define C_CMD_ADC_IN1                   0x82
#define C_CMD_ADC_IN4                   0x88
#define C_CMD_ZERO_COM                  0x00
#define C_CMD_POWER_DOWN                0x90
#define C_CMD_TEST_MODE                 0xF0

typedef enum
{
    MSG_ADC_DMA_DONE = C_DMA_STATUS_DONE,
    MSG_AUDIO_ENCODE_START = 0x10000000,
    MSG_AUDIO_ENCODE_STOPING,
    MSG_AUDIO_ENCODE_STOP,
    MSG_AUDIO_ENCODE_ERR
} AUDIO_RECORD_ENUM;

typedef struct
{
```

```
    INT8U     RIFF_ID[4];     //= {'R','I','F','F'};
    INT32U    RIFF_len; //file size -8
    INT8U     type_ID[4];     // = {'W','A','V','E'};
    INT8U     fmt_ID[4];      // = {'f', 'm','t',' '};
    INT32U    fmt_len;  //16 + extern format byte
    INT16U    format;         // = 1;     //pcm
    INT16U    channel;  // = 1;     // mono
    INT32U    sample_rate;// = 8000;
    INT32U    avg_byte_per_sec;// = 8000*2;     //AvgBytesPerSec = SampleRate *
BlockAlign
    INT16U    Block_align;// = (16 / 8*1) ;                    //BlockAlign =
SignificantBitsPerSample / 8 * NumChannels
    INT16U    Sign_bit_per_sample;// = 16;       //8, 16, 24 or 32
    INT8U     data_ID[4];// = {'d','a','t','a'};
    INT32U    data_len; //extern format byte
} AUD_ENC_WAVE_HEADER;

typedef struct
{
    INT32U    Status;
    INT32U    SourceType;
    INT16S    FileHandle;
    INT8U     InputDevice;          //signal source
    INT8U     Channel;              //1,mono or 2,stereo
    INT32U    AudioFormat;
    INT32U    SampleRate;               //sample rate
    INT32U    BitRate;                  //bite rate
    INT32U    FileLenth;                //byte
    INT32U    NumSamples;               //sample

#if   APP_DOWN_SAMPLE_EN
    INT8U     bEnableDownSample;
    INT8U     *DownSampleWorkMem;
    INT8U     DownsampleFactor;
#endif

    INT8U     *EncodeWorkMem;               //wrok memory
    INT8U     *Bit_Stream_Buffer;      //encode bit stream buffer
```

```
    INT32U    read_index;              //bit stream buffer index
    INT32U    PackSize;            //for file write size, byte
    INT8U     *pack_buffer;        //a1800 and wav lib use
    INT32U    PCMInFrameSize;          //pcm input buffer, short
    INT32U    OnePCMFrameSize;         //short

    //allow record size
    INT64U    disk_free_size;

    INT32U    ring_buffer;
    INT32U    aud_pcm_buffer[C_AUD_PCM_BUFFER_NO];
    INT32U    mic_pcm_buffer[C_AUD_PCM_BUFFER_NO];
    DMA_STRUCT adc_dma_dbf;
    DMA_STRUCT mic_dma_dbf;

#if GPY0050_IN_EN == 1
    INT16U    buffer_index;
    INT16U    pre_value;
    INT32U    buffer_cnt;
    INT32U    ready_buffer;
#endif
} Audio_Encode_Para;

//task api
INT32S adc_record_task_create(INT8U priority);
INT32S adc_record_task_del(INT8U priority);
INT32S adc_record_task_start(void);
INT32S adc_record_task_stop(void);

//api
void audio_record_set_status(INT32U status);
INT32U audio_record_get_status(void);
void audio_record_set_source_type(INT32U type);
INT32U audio_record_get_source_type(void);
INT64U audio_record_set_file_handle_free_size(INT16S file_handle);
void audio_record_set_info(INT32U audio_format, INT32U sample_rate, INT32U
bit_rate);
void audio_record_set_channel(INT8U device, INT8U channel);
```

```
INT32S audio_record_set_down_sample(BOOLEAN b_down_sample, INT8U ratio);
INT32S pcm_energy_detect(INT16S* buffer_addr, INT32U pcm_size);

#endif
/*************************************************************************
*****************************
audio record to storage:
The encode format can be
1. A1800 (GP)
2. WAVE    (PCM)
3. WAVE    (microsoft ADPCM)
4. WAVE    (IMA ADPCM)
5. MP3      sample rate, 48k, 44.1K, 32k, 24K, 22.05k, 16K, 12K, 11.025K, 8k
            bit rate, 32,40,48,56,64,80,96,112,128,160,192,224,256,320 kbps
                        8,16,24,32,40,48,56, 64, 80, 96,112,128,144,160 kbps
*************************************************************************
*****************************/
#include "audio_record.h"
#include "vr_demo_global.h"
#include <stdlib.h>

extern char SoundAppeared;

/*    define */
#define ADC_RECORD_STACK_SIZE                    1024
#define AUD_ENC_QACCEPT_MAX                          5

/* os stack */
INT32U    AdcRecordStack[ADC_RECORD_STACK_SIZE];

/* os global varaible */
void        *aud_enc_reqQ_area[AUD_ENC_QACCEPT_MAX];
OS_EVENT      *aud_enc_reqQ;
void        *aud_enc_reqQ_area2[AUD_ENC_QACCEPT_MAX];
OS_EVENT      *aud_enc_reqQ2;
#if RECORD_WITH_DAC_OUT_EN == 1
void        *aud_enc_reqQ_area3[AUD_ENC_QACCEPT_MAX];
OS_EVENT      *aud_enc_reqQ3;
```

```
#endif

//#define USE_DISK        FS_SD
//#define USE_DISK        FS_NAND1
extern char M_USE_DISK;
extern INT16U NoiseCnt;
extern char AudState;
extern INT32U wifi_aud_file_size;


/* global varaible */
static Audio_Encode_Para *pAudio_Encode_Para;

/* function */
void adc_record_entry(void *param);

//dma
static INT32S aud_adc_double_buffer_put(INT16U *data,INT32U cwlen, INT8U
aud_in, OS_EVENT *os_q);
static INT32U aud_adc_double_buffer_set(INT16U *data, INT32U cwlen, INT8U
aud_in);
static INT32S aud_adc_dma_status_get(INT8U aud_in);
static void aud_adc_double_buffer_free(INT8U aud_in);
static INT32S adc_hardware_start(INT8U InputDevice, INT8U channel, INT32U
SampleRate);
static INT32S adc_hardware_stop(INT8U InputDevice, INT8U channel);

//other
static INT32S adc_memory_allocate(INT32U CbLen, INT8U adc_in);
static INT32S adc_memory_free(void);
static void adc_work_mem_free(void);
#if RECORD_WITH_DAC_OUT_EN == 1
static void dac_out_start(INT32U channel, INT32U samplerate);
static void dac_out_stop(INT32U channel);
#endif

//wave
#if 0
```

```
static AUD_ENC_WAVE_HEADER    WaveHeadPara;
static void aud_enc_RIFF_init(INT32U samplerate);
static INT32S wave_encode_start(void);
static INT32S wave_encode_stop(void);
static int wave_encode_once(void *workmem, const short* buffer_addr, int cwlen);
#endif

#if APP_A1800_ENCODE_EN
static INT32S a1800_encode_start(void);
static INT32S a1800_encode_stop(void);
static int a1800_encode_once(void *workmem, const short* buffer_addr, int cwlen);
#endif

#if APP_WAV_CODEC_EN
static INT32S wave_encode_lib_start(INT32U AudioFormat);
static INT32S wave_encode_lib_stop(void);
static int wave_encode_lib_once(void *workmem, const short* buffer_addr, int
cwlen);
#endif

#if APP_MP3_ENCODE_EN
static INT32S mp3_encode_start(void);
static INT32S mp3_encode_stop(void);
static int mp3_encode_once(void *workmem, const short* buffer_addr, int cwlen);
#endif

#if ((MCU_VERSION == GPL326XXB) || (MCU_VERSION == GPL32670B))
#if APP_VR_ENCODE_EN
// VR global info
//#include "vr_demo_global.h"

static INT32S vr_encode_start(void);
static INT32S vr_encode_stop(void);
static int vr_encode_once(void *workmem, const short* buffer_addr, int cwlen);

#endif
#endif
```

```
#if BUILD_IN_MIC_EN == 1
extern INT32S mic_timer_stop(INT8U timer_id);
extern void mic_fifo_clear(void);
extern void mic_fifo_level_set(INT8U level);
extern INT32S mic_auto_sample_start(void);
extern INT32S mic_sample_rate_set(INT8U timer_id, INT32U hz);
#endif

#if GPY0050_IN_EN == 1
static INT16U gpy0050_get_value(void);
static void gpy0050_start(void);
static void gpy0050_stop(void);
static void gpy0050_isr(void);
#endif

//=============================================================
#define wifi_audio_buffer_size      16000*2*10
//Ricky
extern char TFileStu;
extern INT8S* Wifi_Aud_Buff;
extern INT32S Aud_In_Idx;
extern INT32S Aud_Out_Idx;
//#define aud_buff_size (2048-6)
extern INT8U RecordEnd;
//=================================================================
============================
//    audio task create
//    parameter:     priority.
//    return:        none.
//=================================================================
============================
INT32S adc_record_task_create(INT8U priority)
{
      INT8U err;

      aud_enc_reqQ = OSQCreate(aud_enc_reqQ_area, AUD_ENC_QACCEPT_MAX);
      if (!aud_enc_reqQ) {
            return STATUS_FAIL;
```

```
    }

    aud_enc_reqQ2 = OSQCreate(aud_enc_reqQ_area2,
AUD_ENC_QACCEPT_MAX);
    if (!aud_enc_reqQ2) {
        return STATUS_FAIL;
    }

#if RECORD_WITH_DAC_OUT_EN == 1
    aud_enc_reqQ3 = OSQCreate(aud_enc_reqQ_area3,
AUD_ENC_QACCEPT_MAX);
    if (!aud_enc_reqQ3) {
        return STATUS_FAIL;
    }
#endif

    if(pAudio_Encode_Para == 0) {
        pAudio_Encode_Para = (Audio_Encode_Para
*)gp_malloc_align(sizeof(Audio_Encode_Para), 4);
        if(pAudio_Encode_Para == 0) {
            return STATUS_FAIL;
        }
    }

    gp_memset((INT8S *)pAudio_Encode_Para, 0, sizeof(Audio_Encode_Para));
#if ((defined MCU_VERSION) && (MCU_VERSION < GPL326XX))
    pAudio_Encode_Para->InputDevice = ADC_LINE_IN;
#else
    pAudio_Encode_Para->InputDevice = BUILD_IN_MIC;
#endif
    pAudio_Encode_Para->Channel = C_MONO_RECORD;
    err = OSTaskCreate(adc_record_entry, (void *)pAudio_Encode_Para, (void
*)&AdcRecordStack[ADC_RECORD_STACK_SIZE - 1], priority);
    if(err != OS_NO_ERR) {
        return STATUS_FAIL;
    }

    DEBUG_MSG(DBG_PRINT("AudioEncodeTaskCreate[%d]\r\n", priority));
```

```
	return STATUS_OK;
}

//=============================================================================
============================
//	audio task detete
//	parameter:	priority.
//	return:		none.
//=============================================================================
============================
INT32S adc_record_task_del(INT8U priority)
{
	INT8U err;

	if(aud_enc_reqQ) {
		OSQFlush(aud_enc_reqQ);
	}

	if(aud_enc_reqQ2) {
		OSQFlush(aud_enc_reqQ2);
	}

#if RECORD_WITH_DAC_OUT_EN == 1
	if(aud_enc_reqQ3) {
		OSQFlush(aud_enc_reqQ3);
	}
#endif

	err = OSTaskDel(priority);
	DEBUG_MSG(DBG_PRINT("AudioEncodeTaskDel[%d]\r\n", priority));

	if(aud_enc_reqQ) {
		OSQDel(aud_enc_reqQ, OS_DEL_ALWAYS, &err);
		aud_enc_reqQ = 0;
	}

	if(aud_enc_reqQ2) {
		OSQDel(aud_enc_reqQ2, OS_DEL_ALWAYS, &err);
```

```
        aud_enc_reqQ2 = 0;
    }

#if RECORD_WITH_DAC_OUT_EN == 1
    if(aud_enc_reqQ3) {
        OSQDel(aud_enc_reqQ3, OS_DEL_ALWAYS, &err);
        aud_enc_reqQ3 = 0;
    }
#endif

    if(pAudio_Encode_Para) {
        gp_free((void *)pAudio_Encode_Para);
        pAudio_Encode_Para = 0;
    }
    return err;
}

//================================================================
============================
//  audio record task start
//  parameter:    none.
//  return:       status.
//================================================================
============================
INT32S adc_record_task_start(void)
{
    INT8U err;
    INT32S i, status;

    // a1800 only support mono record
    if(pAudio_Encode_Para->AudioFormat == WAVE_FORMAT_A1800) {
        pAudio_Encode_Para->Channel = C_MONO_RECORD;
        if(pAudio_Encode_Para->InputDevice == DOUBLE_LINEIN) {
        #if ((defined MCU_VERSION) && (MCU_VERSION < GPL326XX))
            pAudio_Encode_Para->InputDevice = ADC_LINE_IN;
        #else
            pAudio_Encode_Para->InputDevice = BUILD_IN_MIC;
        #endif
```

```
            }
        }

        err = OSQPost(aud_enc_reqQ, (void *) MSG_AUDIO_ENCODE_START);
        if(err != OS_NO_ERR) {
            return STATUS_FAIL;
        }

        //wait 10 second
        for(i=0; i<1000; i++) {
            status = audio_record_get_status();
            if(status == C_START_RECORD) {
                return STATUS_OK;
            }

            if(status == C_START_FAIL) {
                return STATUS_FAIL;
            }
            OSTimeDly(1);
        }

        if(i == 1000) {
            return STATUS_FAIL;
        }

        return STATUS_OK;
}

//=============================================================================
============================
//  audio record task start
//  parameter:    none.
//  return:       status.
//=============================================================================
============================
INT32S adc_record_task_stop(void)
{
        INT8U err;
```

```
	INT32S i, status;

	err = OSQPost(aud_enc_reqQ, (void *)MSG_AUDIO_ENCODE_STOPING);
	if(err != OS_NO_ERR) {
		return STATUS_FAIL;
	}

	//wait 10 second
	for(i=0; i<1000; i++) {
		status = audio_record_get_status();
		if(status == C_STOP_RECORD) {
			return STATUS_OK;
		}

		OSTimeDly(1);
	}

	if(i == 1000) {
		return STATUS_FAIL;
	}

	return STATUS_OK;
}

//=============================================================================
============================
//	audio_record_set_status
//	parameter:	status
//	return:
//=============================================================================
============================
void audio_record_set_status(INT32U status)
{
	pAudio_Encode_Para->Status = status;
}

//=============================================================================
============================
```

```
//  audio_record_get_status
//  parameter:
//  return:         status.
//=============================================================================
============================
INT32U audio_record_get_status(void)
{
    return pAudio_Encode_Para->Status;
}

//=============================================================================
============================
//  audio_record_set_source_type
//  parameter:    type = GP_FS/user_define
//  return:
//=============================================================================
============================
void audio_record_set_source_type(INT32U type)
{
    pAudio_Encode_Para->SourceType = type;
}

//=============================================================================
============================
//  audio_record_get_source_type
//  parameter:
//  return:         type.
//=============================================================================
============================
INT32U audio_record_get_source_type(void)
{
    return pAudio_Encode_Para->SourceType;
}

//=============================================================================
============================
//  audio_record_set_file_handle_free_size
//  parameter:    file_handle = wave file handle
```

```
//    return:          status.
//==========================================================================
============================
INT64U audio_record_set_file_handle_free_size(INT16S file_handle)
{
      INT8U    fs_disk;
      INT64U disk_free = 0;

      if(file_handle >= 0) {
            fs_disk = GetDiskOfFile(file_handle);
            disk_free = vfsFreeSpace(fs_disk);
            pAudio_Encode_Para->FileHandle = file_handle;
            pAudio_Encode_Para->disk_free_size = disk_free - 10*1024; //reserved
10K
      } else {
            pAudio_Encode_Para->FileHandle = -1;
            pAudio_Encode_Para->disk_free_size = 0;
      }
      return disk_free;
}

//==========================================================================
============================
//    audio_record_set_info
//    parameter:     audio_format = format
//                   sample_rate =
//                   bit_rate =
//                   channel =
//    return:        status.
//==========================================================================
============================
void audio_record_set_info(INT32U audio_format, INT32U sample_rate, INT32U
bit_rate)
{
      pAudio_Encode_Para->AudioFormat = audio_format;
      pAudio_Encode_Para->SampleRate = sample_rate;
      pAudio_Encode_Para->BitRate = bit_rate;
}
```

```
void audio_record_set_channel(INT8U device, INT8U channel)
{
	pAudio_Encode_Para->InputDevice = device;
	pAudio_Encode_Para->Channel = channel; //1, mono, 2, stereo
}

//=================================================================
============================
//	audio_record_set_down_sample
//	parameter:	b_down_sample = enable/disable
//			ratio = 2~4
//	return:		status.
//=================================================================
============================
INT32S audio_record_set_down_sample(BOOLEAN b_down_sample, INT8U ratio)
{
#if	APP_DOWN_SAMPLE_EN
	pAudio_Encode_Para->bEnableDownSample = b_down_sample;
	pAudio_Encode_Para->DownsampleFactor = ratio;
	return AUD_RECORD_STATUS_OK;
#else
	return AUD_RECORD_STATUS_ERR;
#endif
}

//=================================================================
============================
//	pcm_energy_detect
//	parameter:	none.
//	return:		status.
//=================================================================
============================
INT32S pcm_energy_detect(INT16S* buffer_addr, INT32U pcm_size)
{
	INT16S temp;
	INT16U avg_value;
	INT64U temp_total;
```

```
    INT32S i, cnt, local_cnt;

    temp_total = 0;
    cnt = 0;
    for (i = 0; i < pcm_size; i++) {
        temp = abs(*(buffer_addr + i));
        temp_total += (INT64U)temp;
        if (temp > PCM_LOCAL_THR) {
            cnt += 1;
        }
    }

    // average
    avg_value = (INT64U)temp_total / pcm_size;
    DBG_PRINT("temp_total = 0x%x\r\n", temp_total);
    DBG_PRINT("avg_value = 0x%x, cnt = %d\r\n", avg_value, cnt);

    //samples above Local_Thr to judge as active
    local_cnt = pcm_size / 6;

    if((avg_value > PCM_GLOBAL_THR)||(cnt>local_cnt)) {
        return 0;  //active
    } else {
        return 1;       //not active
    }
}

//=============================================================================
============================
//    audio task main entry
//=============================================================================
============================
void adc_record_entry(void *param)
{
    INT8U    dac_flag;
    INT8U    adc_bufidx;
    INT8U    mic_bufidx;
    INT8U    buffer_index;
```

```
    INT8U      err, device, channel;
    INT16U     mask, t, t2;
    INT16U     SampleRate;
    INT32U     msg_id;
    INT32S     i, nRet;
    INT32U     ready_addr, free_addr, pcm_addr;
    INT16U     *ptr, *ptr1, *ptr2, *temp;
    INT32U     adc_ready_addr, adc_free_addr, adc_pcm_addr;
    INT32U     mic_ready_addr, mic_free_addr, mic_pcm_addr;
    INT32U     encode_addr, aud_dst_buffer;
    void       *hProc;
    INT32S     (*pfn_audio_encode_stop)(void);
   INT32S    (*pfn_audio_encode_once)(void *workmem, const short* buffer_addr,
int cwlen);
    Audio_Encode_Para *pAudRec = (Audio_Encode_Para *)param;
#if 0
    INT32U     L_pcm_ptr, R_pcm_ptr;
#endif

    while(1)
    {
        msg_id = (INT32U) OSQPend(aud_enc_reqQ, 0, &err);
        if(err != OS_NO_ERR) {
            continue;
        }

        switch(msg_id)
        {
        case MSG_ADC_DMA_DONE:
            switch(device)
            {
            case ADC_LINE_IN:
            case BUILD_IN_MIC:
                pcm_addr = ready_addr;
                ready_addr = free_addr;
                buffer_index++;
                if(buffer_index >= C_AUD_PCM_BUFFER_NO) {
                    buffer_index = 0;
```

```
                    }

                    if(device == ADC_LINE_IN) {
                        free_addr = pAudRec->aud_pcm_buffer[buffer_index];
                    } else {
                        free_addr =    pAudRec->mic_pcm_buffer[buffer_index];
                    }

                    if(pAudRec->Status == C_STOP_RECORDING) {
                        // check dma is done and stop
                        if(aud_adc_dma_status_get(device) == 0) {
                            dac_flag = 3;
                            OSQPost(aud_enc_reqQ, (void *)
MSG_AUDIO_ENCODE_STOP);
                        }
                    } else {
                        // set dma buffer
                        aud_adc_double_buffer_set((INT16U *)free_addr,
pAudRec->PCMInFrameSize, device);
                    }

                    // invalid cache
                    cache_invalid_range(pcm_addr, pAudRec->PCMInFrameSize <<
1);

                    // unsigned to signed
                    ptr = (INT16U*)pcm_addr;
                    for(i=0; i<pAudRec->PCMInFrameSize; i++)
                    {
                        t = *ptr;
                    #if ((defined MCU_VERSION) && (MCU_VERSION < GPL327XX))
                        t ^= mask;
                    #endif
                    #if APP_LPF_ENABLE == 1
                        t = (INT16U)LPF_process(t);
                    #endif
                        *ptr++ = t;
                    }
```

```
                encode_addr = pcm_addr;
                break;

            #if GPY0050_IN_EN == 1
            case GPY0050_IN:
                pcm_addr = pAudRec->ready_buffer;
                if(pAudRec->Status == C_STOP_RECORDING) {
                    dac_flag = 3;
                    OSQPost(aud_enc_reqQ, (void
*)MSG_AUDIO_ENCODE_STOP);
                }

                // invalid cache
                cache_invalid_range(pcm_addr, pAudRec->PCMInFrameSize << 1);


                // unsigned to signed
                ptr = (INT16U*)pcm_addr;
                for(i=0; i<pAudRec->PCMInFrameSize; i++)
                {
                    t = *ptr;
                    t ^= mask;
                #if APP_LPF_ENABLE == 1
                    t = (INT16U)LPF_process(t);
                #endif
                    *ptr++ = t;
                }

                encode_addr = pcm_addr;
                break;
            #endif

            case DOUBLE_LINEIN:
                msg_id = (INT32U) OSQPend(aud_enc_reqQ2, 0, &err);
                if (msg_id != MSG_ADC_DMA_DONE) {
                    DBG_PRINT("MIC DMA error !\r\n");
                    continue;
```

```
                    }

                    adc_pcm_addr = adc_ready_addr;
                    adc_ready_addr = adc_free_addr;
                    mic_pcm_addr = mic_ready_addr;
                    mic_ready_addr = mic_free_addr;
                    adc_bufidx++;
                    mic_bufidx++;
                    if(adc_bufidx >= C_AUD_PCM_BUFFER_NO) {
                        adc_bufidx = 0;
                    }

                    if(mic_bufidx >= C_AUD_PCM_BUFFER_NO) {
                        mic_bufidx = 0;
                    }

                    adc_free_addr = pAudRec->aud_pcm_buffer[adc_bufidx];
                    mic_free_addr = pAudRec->mic_pcm_buffer[mic_bufidx];
                    if(pAudRec->Status == C_STOP_RECORDING) {
                        // check dma is done and stop
                        if ((aud_adc_dma_status_get(ADC_LINE_IN) == 0) &&
(aud_adc_dma_status_get(BUILD_IN_MIC) == 0)) {
                            dac_flag = 3;
                            OSQPost(aud_enc_reqQ, (void *)
MSG_AUDIO_ENCODE_STOP);
                        }
                    } else {
                        // set dma buffer
                        aud_adc_double_buffer_set((INT16U *)adc_free_addr,
pAudRec->PCMInFrameSize>>1, ADC_LINE_IN);
                        aud_adc_double_buffer_set((INT16U *)mic_free_addr,
pAudRec->PCMInFrameSize>>1, BUILD_IN_MIC);
                    }

                    // invalid cache
                    cache_invalid_range(adc_pcm_addr,
pAudRec->PCMInFrameSize);
                    cache_invalid_range(mic_pcm_addr,
```

```
pAudRec->PCMInFrameSize);

                        // unsigned to signed
                        ptr1 = (INT16U*)adc_pcm_addr;
                        ptr2 = (INT16U*)mic_pcm_addr;
                        temp = (INT16U*)aud_dst_buffer;
                        for (i=0; i<(pAudRec->PCMInFrameSize>>1); i++)
                        {
                            t = *ptr1++;
                            t2 = *ptr2++;
                        #if ((defined MCU_VERSION) && (MCU_VERSION < GPL327XX))
                            t ^= mask;
                            t2 ^= mask;
                        #endif
                        #if APP_LPF_ENABLE == 1
                            t = (INT16U)LPF_process(t);
                            t2 = (INT16U)LPF_process(t2);
                        #endif
                            *temp++ = t;
                            *temp++ = t2;
                        }

                        encode_addr = aud_dst_buffer;
                        break;
                    }

                #if 0
                    write(L_pcm_ptr, (INT32U)adc_pcm_addr,
pAudRec->PCMInFrameSize);
                    write(R_pcm_ptr, (INT32U)mic_pcm_addr,
pAudRec->PCMInFrameSize);
                #endif
                #if RECORD_WITH_DAC_OUT_EN == 1
                    if(dac_flag == 0) {
                        dac_flag = 1;
                        dac_cha_dbf_put((INT16S *)encode_addr,
pAudRec->PCMInFrameSize, aud_enc_reqQ3);
                    } else if(dac_flag == 1) {
```

```
                    dac_flag = 2;
                    // mute
                    nRet = dac_pga_get();
                    dac_pga_set(0);
                    // dac start
                    dac_cha_dbf_set((INT16S *)encode_addr,
pAudRec->PCMInFrameSize);
                    dac_out_start(channel, SampleRate);
                    // volume up
                    for (i=0; i<=nRet; i++) {
                        dac_pga_set(i);
                    }
                } else if(dac_flag == 2){
                    msg_id = (INT32U) OSQPend(aud_enc_reqQ3, 0, &err);
                    if (msg_id == MSG_ADC_DMA_DONE) {
                        dac_cha_dbf_set((INT16S *)encode_addr,
pAudRec->PCMInFrameSize);
                    }
                } else {
                    dac_flag = 0xFF;
                }
            #endif

                // energy detect
                //pcm_energy_detect((INT16S *)encode_addr,
pAudio_Encode_Para->PCMInFrameSize);

                // encode pcm wave
//              DEBUG_MSG(DBG_PRINT("."));
                nRet = pfn_audio_encode_once(hProc, (const short*)encode_addr,
pAudRec->PCMInFrameSize);
                if(nRet < 0) {
                    OSQPost(aud_enc_reqQ, (void *) MSG_AUDIO_ENCODE_ERR);
                    continue;
                }

                // check storage is full
                if(pAudRec->SourceType == C_GP_FS) {
```

```
                    if(pAudRec->FileLenth >= pAudRec->disk_free_size) {
                        OSQPost(aud_enc_reqQ, (void *)
MSG_AUDIO_ENCODE_STOPING);
                    }
                }
                break;

            case MSG_AUDIO_ENCODE_START:
                DEBUG_MSG(DBG_PRINT("[MSG_AUDIO_ENCODE_START]\r\n"));
                dac_flag = 0;
                hProc = 0;
                pAudRec->ring_buffer = 0;
                mask = 0x8000;
            #if 0
                L_pcm_ptr = open("L_pcm.pcm", O_WRONLY|O_CREAT|O_TRUNC);
                R_pcm_ptr = open("R_pcm.pcm", O_WRONLY|O_CREAT|O_TRUNC);
            #endif
                switch(pAudRec->AudioFormat)
                {
                #if APP_A1800_ENCODE_EN
                case WAVE_FORMAT_A1800:
                    nRet = a1800_encode_start();
                    pfn_audio_encode_once = a1800_encode_once;
                    pfn_audio_encode_stop = a1800_encode_stop;
                    break;
                #endif

                #if 0
                case WAVE_FORMAT_PCM:
                    nRet = wave_encode_start();
                    pfn_audio_encode_once = wave_encode_once;
                    pfn_audio_encode_stop = wave_encode_stop;
                    break;
                #endif

                #if   APP_WAV_CODEC_EN
                case WAVE_FORMAT_PCM:
                case WAVE_FORMAT_IMA_ADPCM:
```

```
            case WAVE_FORMAT_ADPCM:
            case WAVE_FORMAT_ALAW:
            case WAVE_FORMAT_MULAW:
                nRet = wave_encode_lib_start(pAudRec->AudioFormat);
                pfn_audio_encode_once = wave_encode_lib_once;
                pfn_audio_encode_stop = wave_encode_lib_stop;
                break;
            #endif

            #if APP_MP3_ENCODE_EN
            case WAVE_FORMAT_MP3:
                nRet = mp3_encode_start();
                pfn_audio_encode_once = mp3_encode_once;
                pfn_audio_encode_stop = mp3_encode_stop;
                break;
            #endif
            #if ((MCU_VERSION == GPL326XXB) || (MCU_VERSION ==
GPL32670B))
            #if APP_VR_ENCODE_EN
            case PCM_FORMAT_VR:
                nRet = vr_encode_start();
                pfn_audio_encode_once = vr_encode_once;
                pfn_audio_encode_stop = vr_encode_stop;
                break;
            #endif
            #endif
            default:
                DEBUG_MSG(DBG_PRINT("encode fmt err.\r\n"));
                goto __START_FAIL;
            }

            if(nRet < 0) {
                DEBUG_MSG(DBG_PRINT("encode start fail.\r\n"));
                goto __START_FAIL;
            }

        #if  APP_DOWN_SAMPLE_EN
            if(pAudRec->bEnableDownSample) {
```

```
                    if(pAudRec->DownsampleFactor * pAudRec->SampleRate >
48000) {
                        pAudRec->DownsampleFactor = 48000 /
pAudRec->SampleRate;
                    }

                    if(pAudRec->DownsampleFactor >= 2) {
                        pAudRec->DownSampleWorkMem =
DownSample_Create(pAudRec->PCMInFrameSize, pAudRec->DownsampleFactor);

                        DownSample_Link(pAudRec->DownSampleWorkMem,
                                        NULL,
                                        pfn_audio_encode_once,
                                        pAudRec->SampleRate,
                                        pAudRec->Channel,
                                        pAudRec->DownsampleFactor);

                        hProc = pAudRec->DownSampleWorkMem;
                        pfn_audio_encode_once = DownSample_PutData;
                        pAudRec->SampleRate =
DownSample_GetSampleRate(hProc);
                        pAudRec->Channel = DownSample_GetChannel(hProc);
                        pAudRec->PCMInFrameSize *=
pAudRec->DownsampleFactor;
                    }
                }
            #endif

            #if    APP_LPF_ENABLE == 1
                LPF_init(pAudRec->SampleRate, 3);
            #endif

                device = pAudRec->InputDevice;
                channel = pAudRec->Channel;
                SampleRate = pAudRec->SampleRate;
                DEBUG_MSG(DBG_PRINT("Dev=0x%x, Ch=%d, SR=%d\r\n", device,
channel, SampleRate));
                DEBUG_MSG(DBG_PRINT("PCMInFrameSize=%d\r\n",
```

```
pAudRec->PCMInFrameSize));

                // start dma and adc
                switch(device)
                {
                case ADC_LINE_IN:
                    if(channel != 1) {
                        DEBUG_MSG(DBG_PRINT("ChErr.\r\n"));
                        goto __START_FAIL;
                    }

                    nRet = adc_memory_allocate(pAudRec->PCMInFrameSize<<1,
ADC_LINE_IN);
                    if(nRet < 0) {
                        DEBUG_MSG(DBG_PRINT("alocate memory fail.\r\n"));
                        goto __START_FAIL;
                    }

                    ready_addr = pAudRec->aud_pcm_buffer[0];
                    free_addr = pAudRec->aud_pcm_buffer[1];
                    buffer_index = 1;
                    aud_adc_double_buffer_put((INT16U*)ready_addr,
pAudRec->PCMInFrameSize, ADC_LINE_IN, aud_enc_reqQ);
                    aud_adc_double_buffer_set((INT16U*)free_addr,
pAudRec->PCMInFrameSize, ADC_LINE_IN);
                    adc_hardware_start(ADC_LINE_IN, 1, SampleRate);
                    break;

                case BUILD_IN_MIC:
                    if(channel != 1) {
                        DEBUG_MSG(DBG_PRINT("ChErr.\r\n"));
                        goto __START_FAIL;
                    }

                    nRet = adc_memory_allocate(pAudRec->PCMInFrameSize<<1,
BUILD_IN_MIC);
                    if(nRet < 0) {
                        DEBUG_MSG(DBG_PRINT("alocate memory fail.\r\n"));
```

```
                goto __START_FAIL;
            }

            ready_addr = pAudRec->mic_pcm_buffer[0];
            free_addr = pAudRec->mic_pcm_buffer[1];
            buffer_index = 1;
            aud_adc_double_buffer_put((INT16U*)ready_addr,
pAudRec->PCMInFrameSize, BUILD_IN_MIC, aud_enc_reqQ);
            aud_adc_double_buffer_set((INT16U*)free_addr,
pAudRec->PCMInFrameSize, BUILD_IN_MIC);
            adc_hardware_start(BUILD_IN_MIC, 1, SampleRate);
            break;

        #if GPY0050_IN_EN == 1
        case GPY0050_IN:
            if(channel != 1) {
                DEBUG_MSG(DBG_PRINT("ChErr.\r\n"));
                goto __START_FAIL;
            }

            nRet = adc_memory_allocate(pAudRec->PCMInFrameSize<<1,
ADC_LINE_IN);
            if(nRet < 0) {
                DEBUG_MSG(DBG_PRINT("alocate memory fail.\r\n"));
                goto __START_FAIL;
            }

            pAudRec->buffer_index = 0;
            adc_hardware_start(BUILD_IN_MIC, 1, SampleRate);
            break;
        #endif

        case DOUBLE_LINEIN:
            if(channel != 2) {
                DEBUG_MSG(DBG_PRINT("ChErr.\r\n"));
                goto __START_FAIL;
            }
```

```
                    nRet = adc_memory_allocate(pAudRec->PCMInFrameSize,
ADC_LINE_IN|BUILD_IN_MIC);
                    if(nRet < 0) {
                        DEBUG_MSG(DBG_PRINT("alocate memory fail.\r\n"));
                        goto __START_FAIL;
                    }

                    aud_dst_buffer =
(INT32U)gp_malloc_align(pAudRec->PCMInFrameSize<<1, 32);
                    if(aud_dst_buffer == 0) {
                        DEBUG_MSG(DBG_PRINT("aud_dst_buffer fail.\r\n"));
                        goto __START_FAIL;
                    }

                    adc_ready_addr = pAudRec->aud_pcm_buffer[0];
                    adc_free_addr = pAudRec->aud_pcm_buffer[1];
                    mic_ready_addr = pAudRec->mic_pcm_buffer[0];
                    mic_free_addr = pAudRec->mic_pcm_buffer[1];
                    adc_bufidx = 1;
                    mic_bufidx = 1;
                    aud_adc_double_buffer_put((INT16U*)adc_ready_addr,
pAudRec->PCMInFrameSize>>1, ADC_LINE_IN, aud_enc_reqQ);
                    aud_adc_double_buffer_set((INT16U*)adc_free_addr,
pAudRec->PCMInFrameSize>>1, ADC_LINE_IN);
                    aud_adc_double_buffer_put((INT16U*)mic_ready_addr,
pAudRec->PCMInFrameSize>>1, BUILD_IN_MIC, aud_enc_reqQ2);
                    aud_adc_double_buffer_set((INT16U*)mic_free_addr,
pAudRec->PCMInFrameSize>>1, BUILD_IN_MIC);
                    adc_hardware_start(DOUBLE_LINEIN, 2, SampleRate);
                    break;

                default:
                    DEBUG_MSG(DBG_PRINT("InputDeviceErr.\r\n"));
                    goto __START_FAIL;
                }
                pAudRec->Status = C_START_RECORD;
                break;
```

```
__START_FAIL:
            DEBUG_MSG(DBG_PRINT("AudioEncodeStartFail!!!\r\n"));
            adc_work_mem_free();
            adc_memory_free();
            pAudRec->Status = C_START_FAIL;
            break;

        case MSG_AUDIO_ENCODE_STOPING:
            DEBUG_MSG(DBG_PRINT("[MSG_AUDIO_ENCODE_STOPING]\r\n"));
            pAudRec->Status = C_STOP_RECORDING;
            break;

        case MSG_AUDIO_ENCODE_STOP:
            DEBUG_MSG(DBG_PRINT("[MSG_AUDIO_ENCODE_STOP]\r\n"));
            adc_hardware_stop(device, channel);
            pfn_audio_encode_stop();
            adc_memory_free();
            switch(device)
            {
            case ADC_LINE_IN:
                aud_adc_double_buffer_free(ADC_LINE_IN);
                break;

            case BUILD_IN_MIC:
                aud_adc_double_buffer_free(BUILD_IN_MIC);
                break;

            case DOUBLE_LINEIN:
                aud_adc_double_buffer_free(ADC_LINE_IN);
                aud_adc_double_buffer_free(BUILD_IN_MIC);
                if (aud_dst_buffer) {
                    gp_free((void *)aud_dst_buffer);
                    aud_dst_buffer = 0;
                }
                break;
            }

        #if APP_DOWN_SAMPLE_EN
```

```
                if(pAudRec->bEnableDownSample) {
                    DownSample_Del(pAudRec->DownSampleWorkMem);
                    pAudRec->bEnableDownSample = FALSE;
                    pAudRec->DownsampleFactor = 0;
                }
            #endif
            #if 0
                close(L_pcm_ptr);
                close(R_pcm_ptr);
            #endif
            #if RECORD_WITH_DAC_OUT_EN == 1
                // wait dac dma finish
                while(dac_dbf_status_get() || dac_dma_status_get()) {
                    OSTimeDly(1);
                }

                dac_dbf_free();
                dac_out_stop(channel);
                OSQFlush(aud_enc_reqQ3);
            #endif
                OSQFlush(aud_enc_reqQ);
                OSQFlush(aud_enc_reqQ2);
                pAudRec->Status = C_STOP_RECORD;
                break;

            case MSG_AUDIO_ENCODE_ERR:
                DEBUG_MSG(DBG_PRINT("[MSG_AUDIO_ENCODE_ERR]\r\n"));
                OSQPost(aud_enc_reqQ, (void *) MSG_AUDIO_ENCODE_STOPING);
                break;
            }
        }
}

//================================================================
============================
//    aud_adc_double_buffer_put
//    parameter:    data = buffer_addr.
//                  len    = size in word
```

```
//                      os_q = OS _EVENT
//    return:           status.
//=============================================================================
============================
static INT32S aud_adc_double_buffer_put(INT16U *data,INT32U cwlen, INT8U
aud_in, OS_EVENT *os_q)
{
      INT32S status;
      DMA_STRUCT *pDma;

      if (aud_in == ADC_LINE_IN) {
            pDma = &pAudio_Encode_Para->adc_dma_dbf;
            pDma->s_addr = (INT32U) P_ADC_ASADC_DATA;
            pDma->t_addr = (INT32U) data;
            pDma->width = DMA_DATA_WIDTH_2BYTE;
            pDma->count = (INT32U) cwlen;
            pDma->notify = NULL;
            pDma->timeout = 0;
      } else if (aud_in == BUILD_IN_MIC) {
            pDma = &pAudio_Encode_Para->mic_dma_dbf;
            pDma->s_addr = (INT32U) P_MIC_ASADC_DATA;
            pDma->t_addr = (INT32U) data;
            pDma->width = DMA_DATA_WIDTH_2BYTE;
            pDma->count = (INT32U) cwlen;
            pDma->notify = NULL;
            pDma->timeout = 0;
      } else {
            return STATUS_FAIL;
      }

      status = dma_transfer_with_double_buf(pDma, os_q);
      if (status != 0) {
            return status;
      }

      return STATUS_OK;
}
```

```
//=====================================================================
============================
//    aud_adc_double_buffer_set
//    parameter:      data = buffer_addr.
//                    len    = size in word
//    return:         status.
//=====================================================================
============================
static INT32U aud_adc_double_buffer_set(INT16U *data, INT32U cwlen, INT8U
aud_in)
{
    INT32S status;
    DMA_STRUCT *pDma;

    if (aud_in == ADC_LINE_IN) {
        pDma = &pAudio_Encode_Para->adc_dma_dbf;
        pDma->s_addr = (INT32U) P_ADC_ASADC_DATA;
        pDma->t_addr = (INT32U) data;
        pDma->width = DMA_DATA_WIDTH_2BYTE;
        pDma->count = (INT32U) cwlen;
        pDma->notify = NULL;
        pDma->timeout = 0;

    } else if (aud_in == BUILD_IN_MIC) {
        pDma = &pAudio_Encode_Para->mic_dma_dbf;
        pDma->s_addr = (INT32U) P_MIC_ASADC_DATA;
        pDma->t_addr = (INT32U) data;
        pDma->width = DMA_DATA_WIDTH_2BYTE;
        pDma->count = (INT32U) cwlen;
        pDma->notify = NULL;
        pDma->timeout = 0;
    } else {
        return STATUS_FAIL;
    }

    status = dma_transfer_double_buf_set(pDma);
    if(status != 0) {
        return status;
```

```
	}

	return STATUS_OK;
}

//=============================================================================
============================
//	aud_adc_dma_status_get
//	parameter:	none
//	return:		status.
//=============================================================================
============================
static INT32S aud_adc_dma_status_get(INT8U aud_in)
{
	INT32S dbf = 0;
	INT32S status = 0;
	DMA_STRUCT *pDma;

	if (aud_in == ADC_LINE_IN) {
		pDma = &pAudio_Encode_Para->adc_dma_dbf;
	} else if (aud_in == BUILD_IN_MIC) {
		pDma = &pAudio_Encode_Para->mic_dma_dbf;
	} else {
		return STATUS_FAIL;
	}

	if (pDma->channel == 0xff) {
		return -1;
	}

	dbf = dma_dbf_status_get(pDma->channel);
	status = dma_status_get(pDma->channel);
	if(dbf || status) {
		return 1;
	}

	return 0;
}
```

```
//=============================================================================
============================
//    aud_adc_double_buffer_free
//    parameter:      none
//    return:         free double buffer dma channel.
//=============================================================================
============================
static void aud_adc_double_buffer_free(INT8U aud_in)
{
    DMA_STRUCT *pDma;

    if (aud_in == ADC_LINE_IN) {
        pDma = &pAudio_Encode_Para->adc_dma_dbf;
    } else if (aud_in == BUILD_IN_MIC) {
        pDma = &pAudio_Encode_Para->mic_dma_dbf;
    } else {
        return;
    }

    dma_transfer_double_buf_free(pDma);
    pDma->channel = 0xff;
}

//=============================================================================
============================
//    adc_hardware_start
//    parameter:      SampleRate = adc sample rate set.
//    return:         none.
//=============================================================================
============================
static INT32S adc_hardware_start(INT8U InputDevice, INT8U channel, INT32U
SampleRate)
{
    INT32U adc_pin;
    OS_CPU_SR cpu_sr;

    //set adc channel is input float
```

```
#if (defined MCU_VERSION) && (MCU_VERSION < GPL327XX)
	#if C_ADC_USE_CHANNEL == ADC_LINE_0
	adc_pin = IO_F6;
	#elif C_ADC_USE_CHANNEL == ADC_LINE_1
	adc_pin = IO_F7;
	#elif C_ADC_USE_CHANNEL == ADC_LINE_2
	adc_pin = IO_F8;
	#elif C_ADC_USE_CHANNEL == ADC_LINE_3
	adc_pin = IO_F9;
	#endif
#else
	#if C_ADC_USE_CHANNEL == ADC_LINE_0
	adc_pin = IO_F7;
	#elif C_ADC_USE_CHANNEL == ADC_LINE_1
	adc_pin = IO_F8;
	#endif
#endif

	switch(InputDevice)
	{
	#if ADC_LINE_IN_EN == 1
	case ADC_LINE_IN:
		if(channel != 1) {
			DEBUG_MSG(DBG_PRINT("ChERR.\r\n"));
			return STATUS_FAIL;
		}

		gpio_init_io(adc_pin, GPIO_INPUT);
		gpio_set_port_attribute(adc_pin, ATTRIBUTE_HIGH);

		adc_fifo_clear();
		adc_auto_ch_set(C_ADC_USE_CHANNEL);
		adc_fifo_level_set(4);
		adc_auto_sample_start();
		//OSTimeDly(50); //wait bais stable
		adc_sample_rate_set(C_ADC_USE_TIMER, SampleRate);
		break;
	#endif
```

```
#if BUILD_IN_MIC_EN == 1
case BUILD_IN_MIC:
	if(channel != 1) {
		DEBUG_MSG(DBG_PRINT("ChERR.\r\n"));
		return STATUS_FAIL;
	}

	mic_fifo_clear();
	mic_fifo_level_set(4);
	mic_agc_enable_set(1);
	mic_auto_sample_start();

#if (MCU_VERSION == GPL326XXB) || (MCU_VERSION == GP326XXXA)
	mic_dagc_setup(1,0x20,0,0,0);
	mic_dagc_set_att_rel_time(1,0xf8);
	mic_set_pga_gain(1,0);
	mic_dagc_enable(1);
#endif

	OSTimeDly(50);	//wait bais stable
	mic_sample_rate_set(C_ADC_USE_TIMER, SampleRate);
	break;
#endif

#if GPY0050_IN_EN == 1
case GPY0050_IN:
	if(channel != 1) {
		DEBUG_MSG(DBG_PRINT("ChERR.\r\n"));
		return STATUS_FAIL;
	}

	gpy0050_start();
	timer_freq_setup(C_AUDIO_RECORD_TIMER, SampleRate, 0, gpy0050_isr);
	break;
#endif

case DOUBLE_LINEIN:
```

```
        if(channel != 2) {
            DEBUG_MSG(DBG_PRINT("ChERR.\r\n"));
            return STATUS_FAIL;
        }

#if ADC_LINE_IN_EN == 1
        gpio_init_io(adc_pin, GPIO_INPUT);
        gpio_set_port_attribute(adc_pin, ATTRIBUTE_HIGH);

        adc_fifo_clear();
        adc_fifo_level_set(4);
        adc_auto_ch_set(C_ADC_USE_CHANNEL);
        adc_auto_sample_start();
#endif

#if BUILD_IN_MIC_EN == 1
        mic_fifo_clear();
        mic_fifo_level_set(4);
        mic_agc_enable_set(0);
#if (MCU_VERSION == GPL326XXB) || (MCU_VERSION == GP326XXXA)
        //mic_dagc_enable(1);
        mic_dagc_enable(0);
        mic_set_pga_gain(0, 0x19);
#endif
        mic_auto_sample_start();
#endif

        // wait bais stable
        OSTimeDly(70);

        OS_ENTER_CRITICAL();
#if ADC_LINE_IN_EN == 1 && BUILD_IN_MIC_EN == 1
        adc_sample_rate_set(C_ADC_USE_TIMER, SampleRate);
        mic_sample_rate_set(C_ADC_USE_TIMER, SampleRate);
#endif
        OS_EXIT_CRITICAL();
        break;
```

```
	default:
		DEBUG_MSG(DBG_PRINT("InputDeviceERR.\r\n"));
		return STATUS_FAIL;
	}

	return STATUS_OK;
}

//=====================================================================
============================
//	adc_hardware_start
//	parameter:	SampleRate = adc sample rate set.
//	return:		none.
//=====================================================================
============================
static INT32S adc_hardware_stop(INT8U InputDevice, INT8U channel)
{
	switch(InputDevice)
	{
	#if ADC_LINE_IN_EN == 1
	case ADC_LINE_IN:
		if(channel != 1) {
			DEBUG_MSG(DBG_PRINT("ChERR.\r\n"));
			return STATUS_FAIL;
		}
		adc_timer_stop(C_ADC_USE_TIMER);
		break;
	#endif

	#if BUILD_IN_MIC_EN == 1
	case BUILD_IN_MIC:
		if(channel != 1) {
			return STATUS_FAIL;
		}
		mic_timer_stop(C_ADC_USE_TIMER);
		break;
	#endif
```

```
        #if GPY0050_IN_EN == 1
        case GPY0050_IN:
            if(channel != 1) {
                DEBUG_MSG(DBG_PRINT("ChERR.\r\n"));
                return STATUS_FAIL;
            }
            timer_stop(C_AUDIO_RECORD_TIMER);
            gpy0050_stop();
            break;
        #endif

        case DOUBLE_LINEIN:
            if(channel != 2) {
                DEBUG_MSG(DBG_PRINT("ChERR.\r\n"));
                return STATUS_FAIL;
            }
        #if ADC_LINE_IN_EN == 1
            adc_timer_stop(C_ADC_USE_TIMER);
        #endif
        #if BUILD_IN_MIC_EN == 1
            mic_timer_stop(C_ADC_USE_TIMER);
        #endif
            break;

        default:
            return STATUS_FAIL;
        }

        return STATUS_OK;
}

#if GPY0050_IN_EN == 1
//=================================================================
============================
//    gpy0050 record
//    parameter:    none
//    return:       free double buffer dma channel.
//=================================================================
```

```
============================
#define SPI0_TXRX_BY_CPU(Txdata, RxData)\
{\
	R_SPI0_TX_DATA = Txdata; \
	while((R_SPI0_RX_STATUS & 0x07) == 0);\
	RxData = R_SPI0_RX_DATA;\
}\

static INT16U gpy0050_get_value(void)
{
	INT8U dummy;
	INT16U temp;

	gpio_write_io(C_GPY0050_SPI_CS_PIN, 0);   //pull cs low
	SPI0_TXRX_BY_CPU(C_CMD_DUMMY_COM, dummy);
	SPI0_TXRX_BY_CPU(C_CMD_ADC_IN4, dummy);
	temp = dummy;
	SPI0_TXRX_BY_CPU(C_CMD_ZERO_COM, dummy);
	temp <<= 8;
	temp |= dummy;
	gpio_write_io(C_GPY0050_SPI_CS_PIN, 1);   //pull cs high

	if(temp <= 0x00FF)
		temp = pAudio_Encode_Para->pre_value;
	else if(temp >= 0xFFC0)
		temp = pAudio_Encode_Para->pre_value;
	else
		pAudio_Encode_Para->pre_value = temp;

	return temp;
}

static void gpy0050_start(void)
{
	INT8U dummy;

	gpio_init_io(C_GPY0050_SPI_CS_PIN, GPIO_OUTPUT);
	gpio_set_port_attribute(C_GPY0050_SPI_CS_PIN, ATTRIBUTE_HIGH);
```

```
    gpio_write_io(C_GPY0050_SPI_CS_PIN, DATA_HIGH);  //pull cs high

    //GPL32 SPI IF 1 initialization
    R_SPI0_TX_STATUS = 0x8020;
    R_SPI0_RX_STATUS = 0x8020;
    R_SPI0_CTRL = 0x0800;  //reset SPI0
    R_SPI0_MISC = 0x0100;    //enable smart mode
    R_SPI0_CTRL = 0x8035;  //Master mode 3, SPI_CLK = SYS_CLK/64

    //sw reset gpy0050
    gpio_write_io(C_GPY0050_SPI_CS_PIN, 0);   //pull cs low
    SPI0_TXRX_BY_CPU(C_CMD_RESET_IN4, dummy);
    gpio_write_io(C_GPY0050_SPI_CS_PIN, 1);   //pull cs high

    //enable MIC AGC and ADC
    gpio_write_io(C_GPY0050_SPI_CS_PIN, 0);          //pull cs low
    SPI0_TXRX_BY_CPU(C_CMD_ENABLE_MIC_AGC_ADC, dummy);
    SPI0_TXRX_BY_CPU(C_CMD_ADC_IN4, dummy);
    gpio_write_io(C_GPY0050_SPI_CS_PIN, 1);   //pull cs high

    OSTimeDly(30);      //wait 300ms after AGC & MIC enable

    //dummy data
    gpy0050_get_value();
    gpy0050_get_value();
}

static void gpy0050_stop(void)
{
    INT8U dummy;

    //power down
    gpio_write_io(C_GPY0050_SPI_CS_PIN, 0);   //pull cs low
    SPI0_TXRX_BY_CPU(C_CMD_POWER_DOWN, dummy);
    gpio_write_io(C_GPY0050_SPI_CS_PIN, 1);   //pull cs high

    //GPL32 SPI disable
    R_SPI0_CTRL = 0;
```

```
	R_SPI0_MISC = 0;
}

static void gpy0050_isr(void)
{
	INT16U *ptr;

	ptr =
(INT16U*)pAudio_Encode_Para->aud_pcm_buffer[pAudio_Encode_Para->buffer_ind
ex];
	*(ptr + pAudio_Encode_Para->buffer_cnt++) = gpy0050_get_value();
	if(pAudio_Encode_Para->buffer_cnt >= pAudio_Encode_Para->PCMInFrameSize)
	{
		pAudio_Encode_Para->buffer_cnt = 0;
		pAudio_Encode_Para->ready_buffer =
pAudio_Encode_Para->aud_pcm_buffer[pAudio_Encode_Para->buffer_index];
		pAudio_Encode_Para->buffer_index++;
		if(pAudio_Encode_Para->buffer_index >= C_AUD_PCM_BUFFER_NO) {
			pAudio_Encode_Para->buffer_index = 0;
		}
		OSQPost(aud_enc_reqQ, (void*)MSG_ADC_DMA_DONE);
	}
}
#endif

////////////////////////////////////////////////////////////////////////////////
///////
////////////////////////////////////////////////////////////////////////////////
///////
////////////////////////////////////////////////////////////////////////////////
///////
////////////////////////////////////////////////////////////////////////////////
///////
////////////////////////////////////////////////////////////////////////////////
///////
////////////////////////////////////////////////////////////////////////////////
///////
void InitWifiRecordRam()
```

```
{
	Aud_In_Idx = 0;
	Aud_Out_Idx = 0;
	TFileStu = 0;
	RecordEnd = 0;
}

void Init_Wifi_AudioBuff()
{
	if (Wifi_Aud_Buff==NULL)
		Wifi_Aud_Buff = gp_malloc(wifi_audio_buffer_size);
	InitWifiRecordRam();
}


//=================================================================
============================
//	save_buffer_to_storage
//	parameter:
//	return:		status
//=================================================================
============================
void Save_Aud_Backup(INT8U	*addr, INT32S size)
{
	INT16U	i;
	DBG_PRINT("Audio in size=%d, total size=%d\r\n",size,Aud_In_Idx);
	for (i=0;i<size;i++)
	{
		if (Aud_In_Idx<wifi_audio_buffer_size)
		{
			*(Wifi_Aud_Buff + Aud_In_Idx) = *(addr + i);
			Aud_In_Idx++;
		}
		else
		{
			DBG_PRINT("audio memory full\r\n");
			return;
		}
```

```
	}
}
char WAV_WriteState = 0;
void Save_Aud_Backup_PCM()
{
	INT32U	i;
	if (WAV_WriteState==0)
	{
		for (i=0;i<wifi_audio_buffer_size;i=i+2)
		{
//			if (*(addr + i)
		}
	}
}
static INT32S save_buffer_to_storage(void)
{
	INT8U	*addr;
	INT32S size, write_cblen, file_size;
	char i = 0;

	if(pAudio_Encode_Para->ring_buffer == 0) {
		if(pAudio_Encode_Para->read_index > C_BS_BUFFER_SIZE/2) {
			pAudio_Encode_Para->ring_buffer = 1;
			addr = pAudio_Encode_Para->Bit_Stream_Buffer;
			size = C_BS_BUFFER_SIZE/2;
			if(pAudio_Encode_Para->SourceType == C_GP_FS) {
				write_cblen = write(pAudio_Encode_Para->FileHandle,
(INT32U)addr, size);
			} else {
				write_cblen = audio_encode_data_write(0, (INT32U)addr, size);
			}
			Save_Aud_Backup(addr, size);
			if(write_cblen != size) {
				return	AUD_RECORD_FILE_WRITE_ERR;
			}
		}
	} else {
		if(pAudio_Encode_Para->read_index < C_BS_BUFFER_SIZE/2) {
```

```
                pAudio_Encode_Para->ring_buffer = 0;
                addr = pAudio_Encode_Para->Bit_Stream_Buffer +
C_BS_BUFFER_SIZE/2;
                size = C_BS_BUFFER_SIZE/2;
                if(pAudio_Encode_Para->SourceType == C_GP_FS) {
                    write_cblen = write(pAudio_Encode_Para->FileHandle,
(INT32U)addr, size);
                } else {
                    write_cblen = audio_encode_data_write(0, (INT32U)addr, size);
                }
                Save_Aud_Backup(addr, size);
                if(write_cblen != size) {
                    return    AUD_RECORD_FILE_WRITE_ERR;
                }
            }
        }
        return AUD_RECORD_STATUS_OK;
}

//=================================================================
============================
//    save_final_data_to_storage
//    parameter:
//    return:        status
//=================================================================
============================
static INT32S save_final_data_to_storage(void)
{
        INT8U *addr;
        INT32S size, write_cblen;
        RecordEnd = 1;
        if(pAudio_Encode_Para->read_index > C_BS_BUFFER_SIZE/2) {
            addr = pAudio_Encode_Para->Bit_Stream_Buffer + C_BS_BUFFER_SIZE/2;
            size = pAudio_Encode_Para->read_index - C_BS_BUFFER_SIZE/2;
            if(pAudio_Encode_Para->SourceType == C_GP_FS) {
                write_cblen = write(pAudio_Encode_Para->FileHandle, (INT32U)addr,
size);
            } else {
```

```
                write_cblen = audio_encode_data_write(0, (INT32U)addr, size);
            }
//          Save_Aud_Backup(addr, C_BS_BUFFER_SIZE/2);
            Save_Aud_Backup(addr, size);
            DBG_PRINT("final %d\r\n",size);
//          CloseWifiAudFile();
            if(write_cblen != size) {
                return      AUD_RECORD_FILE_WRITE_ERR;
            }
        } else {
            addr = pAudio_Encode_Para->Bit_Stream_Buffer;
            size = pAudio_Encode_Para->read_index;
            if(pAudio_Encode_Para->SourceType == C_GP_FS) {
                write_cblen = write(pAudio_Encode_Para->FileHandle, (INT32U)addr,
size);
            } else {
                write_cblen = audio_encode_data_write(0, (INT32U)addr, size);
            }
//          Save_Aud_Backup(addr, C_BS_BUFFER_SIZE/2);
            Save_Aud_Backup(addr, size);
            DBG_PRINT("final %d\r\n",size);
//          CloseWifiAudFile();
            if(write_cblen != size) {
                return      AUD_RECORD_FILE_WRITE_ERR;
            }
        }
        return AUD_RECORD_STATUS_OK;
}

//=================================================================
============================
//  adc_memory_allocate
//  parameter:      cbLen = adc buffer size in byte.
//  return:         status.
//=================================================================
============================
static INT32S adc_memory_allocate(INT32U CbLen, INT8U adc_in)
{
```

```
    INT16U *ptr;
    INT32S i, j;

    if(adc_in & ADC_LINE_IN) {
        pAudio_Encode_Para->aud_pcm_buffer[0] =(INT32U)
gp_malloc_align(CbLen*C_AUD_PCM_BUFFER_NO, 4);
        if(pAudio_Encode_Para->aud_pcm_buffer[0] == 0) {
            return AUD_RECORD_MEMORY_ALLOC_ERR;
        }

        for(i=0; i<C_AUD_PCM_BUFFER_NO; i++) {
            pAudio_Encode_Para->aud_pcm_buffer[i] =
pAudio_Encode_Para->aud_pcm_buffer[0] + (CbLen * i);
            DEBUG_MSG(DBG_PRINT("aud_pcm_buffer[%d] = 0x%x\r\n", i,
pAudio_Encode_Para->aud_pcm_buffer[i]));
            ptr = (INT16U*)pAudio_Encode_Para->aud_pcm_buffer[i];
            for(j=0; j<CbLen/2; j++) {
                *ptr++ = 0x8000;
            }
        }
    }

    if(adc_in & BUILD_IN_MIC) {
        pAudio_Encode_Para->mic_pcm_buffer[0] =(INT32U)
gp_malloc_align(CbLen*C_AUD_PCM_BUFFER_NO, 4);
        if(pAudio_Encode_Para->mic_pcm_buffer[0] == 0) {
            return AUD_RECORD_MEMORY_ALLOC_ERR;
        }

        for(i=0; i<C_AUD_PCM_BUFFER_NO; i++) {
            pAudio_Encode_Para->mic_pcm_buffer[i] =
pAudio_Encode_Para->mic_pcm_buffer[0] + (CbLen * i);
            DEBUG_MSG(DBG_PRINT("mic_pcm_buffer[%d] = 0x%x\r\n", i,
pAudio_Encode_Para->mic_pcm_buffer[i]));
            ptr = (INT16U*)pAudio_Encode_Para->mic_pcm_buffer[i];
            for(j=0; j<CbLen/2; j++) {
                *ptr++ = 0x8000;
            }
```

```
            }
        }

        return AUD_RECORD_STATUS_OK;
}

//=================================================================
============================
//    adc_memory_free
//    parameter:      none.
//    return:         status.
//=================================================================
============================
static INT32S adc_memory_free(void)
{
        INT32S i;

        if(pAudio_Encode_Para->aud_pcm_buffer[0]) {
            gp_free((void *)pAudio_Encode_Para->aud_pcm_buffer[0]);
        }

        if(pAudio_Encode_Para->mic_pcm_buffer[0]) {
            gp_free((void *)pAudio_Encode_Para->mic_pcm_buffer[0]);
        }

        for(i=0; i<C_AUD_PCM_BUFFER_NO; i++) {
            pAudio_Encode_Para->aud_pcm_buffer[i] = 0;
            pAudio_Encode_Para->mic_pcm_buffer[i] = 0;
        }

        return AUD_RECORD_STATUS_OK;
}

//=================================================================
============================
//    adc_work_mem_free
//    parameter:
//    return:
```