发布时间：2022/09/20

适用版本：wireless_mic_sdk-v1.4.1_xxx 及以上版本

1、功能说明

该文档主要针对方案要加入第三方算法处理应用情景，提供算法数据流处理 demo 例程。

2、配置说明

a) 数据流注册说明， 参考如下截图

```
172
173 static const struct adapter_stream_fmt slave_audio_stream_list[] = {
174
175 #if 0 <-------- 改成1， 开启算法数据流处理demo
176     {
177     ¦   .attr = ADAPTER_STREAM_ATTR_STREAM_DEMO,
178     ¦   .value = {
179     ¦   ¦   .demo = {
180     ¦   ¦   ¦   .data_handle = NULL,//如果要关注处理前的数据可以在这里注册回调
181     ¦   ¦   },
182     ¦   },
183     },
184 #endif
185
NORMAL / PASTE  +0 ~0 -0  release/adapter-v1.4.0  apps/adapter/app_main/wireless_mic_2t1/2t1_tx.c
```

b) Demo 数据流处理文件

```
adapter_stream_demo.c
adapter_stream_demo.h
```

c) SDK 提供了同步和异步两种模式的处理方式， 配置如下：

```
2 #include "adapter_stream_demo.h"
3 #include "stream_entry.h"
4
5
6 #define DEMO_TASK_NAME   "demo_task1"
7
8 #define DEMO_STREAM_ASYNC_EN        1//1:异步处理，会增加一个处理单元的延时， 0:同步处理， 不增加延时， 但是要求在一个cpu核要可以做完算法处理
9
10 static int adapter_stream_demo_data_handle(void *priv, struct audio_data_frame *in)
11 {
12     //putchar('$');
13     struct __stream_demo_hdl *hdl = (struct __stream_demo_hdl *)priv;
14
15     if (in->offset) {
16     ¦   return in->data_len;
17     }
18 #if (DEMO_STREAM_ASYNC_EN)
19     u8 ready = 1;
20     if (in->data_len && hdl->status) {
21     ¦   u32 r_len, w_len;
22     ¦   hdl->r_busy = 1;
23     ¦   //先写数据流过来的数据到in_cbuf做运算准备
24     ¦   w_len = cbuf_write(&hdl->in_cbuf, in->data, in->data_len);
25     ¦   if (w_len != in->data_len) {
26     ¦   ¦   putchar('b');
27     ¦   }
28     ¦   os_sem_set(&hdl->sem, 0);
29     ¦   os_sem_post(&hdl->sem);
30     ¦   //从out_cbuf读取已经算法处理完的数据
 PASTE  release/adapter-v1.4.0  apps/adapter/audio/adapter_stream_demo.c  c  utf-8
```

d) **异步处理，** **会增加一个处理单元的延时（SDK 默认 5ms），** 也就是说以 SDK 默认为 5ms， 48K 采样率为例， **一个处理单元的数据点数是 240， 要做好算法对齐，**

SDK 会将算法执行放到另外一个 cpu 核进行异步处理。

e) **同步处理**，不增加延时， 但是要求所有算法处理要在一个 cpu 核内完成（也就是双核只能用一个核）， 要综合 SDK 资源情况评估是否能跑下来。

f) 算法初始化、释放、执行， 如下截图描述：

初始化：

```
#define PROCESS_POINT    WIRELESS_MIC_ADC_POINT_UNIT//480        //一次算法处理的点数

struct __stream_demo_hdl *adapter_stream_demo_open(void)
{
    struct __stream_demo_hdl *hdl = zalloc(sizeof(struct __stream_demo_hdl));
    ASSERT(hdl);

    hdl->stream = stream_entry_open(hdl, adapter_stream_demo_data_handle, 0);
    ASSERT(hdl->stream);

#if (DEMO_STREAM_ASYNC_EN)
    hdl->status = 1;
    hdl->fill_flag = 1;
    hdl->skip_counter = 300;
    os_mutex_create(&hdl->mutex);
    os_sem_create(&hdl->sem, 0);
```

申请算法资源和初始化

```
AL / PASTE  +1 ~0 -0   release/adapter-v1.4.0   apps/adapter/audio/adapter_stream_demo.c   c
```

同步算法执行：

```
static int adapter_stream_demo_data_handle(void *priv, struct audio_data_frame *in)
{
    //putchar('$');
    struct __stream_demo_hdl *hdl = (struct __stream_demo_hdl *)priv;

    if (in->offset) {
        return in->data_len;
    }
#if (DEMO_STREAM_ASYNC_EN)
    u8 ready = 1;
    if (in->data_len && hdl->status) {
        u32 r_len, w_len;
        hdl->r_busy = 1;
        //先写数据流过来的数据到in_cbuf做运算准备
        w_len = cbuf_write(&hdl->in_cbuf, in->data, in->data_len);
        if (w_len != in->data_len) {
            putchar('b');
        } else {
            if (hdl->skip_counter) {
                hdl->skip_counter--;
                memset(in->data, 0, in->data_len);
            }
        }
        //通知算法处理线程， 有新的数据写入， 是否需要做运算
        hdl->r_busy = 0;
    }
#else
    //处理算法
    //拿data的数据来做算法运算
    //memset(in->data, 0, in->data_len);
#endif

    return in->data_len;
}
```

算法run执行， 注意算法一次处理点数（或者理解为一次处理的数据长度）

异步算法执行：

```
64
65 static void adapter_stream_demo_task(void *priv)
66 {
67     u32 r_len, w_len;
68     struct __stream_demo_hdl *hdl = (struct __stream_demo_hdl *)priv;
69     ASSERT(hdl);
70     while (1) {
71         os_sem_pend(&hdl->sem, 0);
72         if (hdl->status) {
73             hdl->w_busy = 1;
74             r_len = cbuf_read(&hdl->in_cbuf, hdl->frame_buf, hdl->frame_buf_size);
75             if (r_len == hdl->frame_buf_size) {
76                 if (hdl->status) {
77 #if 1
78                     os_mutex_pend(&hdl->mutex, 0);
79                     //处理算法
80                     //拿data的数据来做算法运算
81                     //memset(hdl->data, 0, hdl->data_len);
82                     os_mutex_post(&hdl->mutex);
83 #endif
84                 }
85 //                putchar('W');
86                 w_len = cbuf_write(&hdl->out_cbuf, hdl->frame_buf, hdl->frame_buf_size);
87                 if (w_len != hdl->frame_buf_size) {
88                     putchar('A');
89                 } else {
90                     //putchar('C');
91                 }
92             } else {
93                 //putchar('a');
94             }
95             hdl->w_busy = 0;
96         } else {
97             break;    算法run执行， 注意算法一次处理点数（或者理解为一次数据处理的长度）
98         }
99
100     }
101     printf("task wait kill!!\n");
102     while (1) {
NORMAL / PASTE  +1 ~0 -0  release/adapter-v1.4.0  apps/adapter/audio/adapter_stream_demo.c
```

算法释放：

```
161 void adapter_stream_demo_close(struct __stream_demo_hdl **p)
162 {
163     struct __stream_demo_hdl *hdl = *p;
164     if (p && hdl) {
165 #if (DEMO_STREAM_ASYNC_EN)
166         hdl->status = 0;
167         while (hdl->r_busy) {
168             os_time_dly(2);
169         }
170
171         while (hdl->w_busy) {
172             os_sem_set(&hdl->w_sem, 0);
173             os_sem_post(&hdl->w_sem);
174             os_time_dly(2);
175         }
176
177         printf("%s, wait busy ok\n", __FUNCTION__);
178
179         stream_entry_close(&hdl->stream);
180
181
182         task_kill(DEMO_TASK_NAME);
183
184         if (hdl->frame_buf) {
185             free(hdl->frame_buf);
186         }
187         if (hdl->in_buf) {
188             free(hdl->in_buf);
189         }
190         if (hdl->out_buf) {
191             free(hdl->out_buf);
192         }
193                                                    释放算法资源
194 #else
195         stream_entry_close(&hdl->stream);
196 #endif
197         free(hdl);
198         *p = NULL;
199         mem_stats();
NORMAL / PASTE  +1 ~0 -0  release/adapter-v1.4.0  apps/adapter/audio/adapter_stream_demo.c
```