相关实现

apps\earphone\earphone.c

```
//============================================================================
=//
// 注意：这些是弱符号定义，用于解决编译问题，实际实现在其他文件中
// 这些函数提供了低延迟模式的默认实现，允许在链接时被其他同名函数覆盖

/**
 * @brief 获取A2DP编解码器的低延迟模式状态（弱符号）
 * @return int 默认返回0，表示低延迟模式关闭
 * @note 这是一个弱符号函数，实际实现在其他文件中
 *       如果其他文件没有实现此函数，则使用此默认实现
 */
__attribute__((weak))
int earphone_a2dp_codec_get_low_latency_mode()
{
    return 0;  // 默认返回0，表示低延迟模式关闭
}

/**
 * @brief 设置A2DP编解码器的低延迟模式（弱符号）
 * @param enable 是否启用低延迟模式
 *               - 1: 启用
 *               - 0: 禁用
 * @param msec 延迟时间（毫秒）
 * @return int 默认返回0，表示操作成功
 * @note 这是一个弱符号函数，实际实现在其他文件中
 *       如果其他文件没有实现此函数，则使用此默认实现
 */
__attribute__((weak))
int earphone_a2dp_codec_set_low_latency_mode(int enable, int msec)
{
    return 0;  // 默认返回0，表示操作成功
}
//============================================================================
=//
```

```c
/**
 * @brief 获取当前低延迟模式状态
 * @return int 返回当前低延迟模式状态
 *             - 1: 低延迟模式已开启
 *             - 0: 低延迟模式已关闭
 */
int bt_get_low_latency_mode()
{
    return earphone_a2dp_codec_get_low_latency_mode();
}

/**
 * @brief 设置低延迟模式
 * @param enable 控制开关
 *               - 1: 开启低延迟模式
 *               - 0: 关闭低延迟模式
 * @note 该函数用于控制蓝牙耳机的低延迟模式（游戏模式）开关
 *       在TWS模式下，只有主设备可以控制低延迟模式
 *       开启/关闭低延迟模式时会播放提示音
 */
void bt_set_low_latency_mode(int enable)
{
#if TCFG_USER_TWS_ENABLE  // 如果启用了TWS功能
    // 检查当前设备是否是TWS主设备
    if (tws_api_get_role() == TWS_ROLE_MASTER) {
        // 检查从设备是否已连接
        if (tws_api_get_tws_state() & TWS_STA_SIBLING_CONNECTED) {
            printf("[%s,%d]----samson---enable = %d-
-/n",__func__,__LINE__,enable);
            // 从设备已连接，通过TWS协议同步低延迟模式状态
            if (enable) {
                // 同步开启低延迟模式，300ms超时
                tws_api_sync_call_by_uuid('T', SYNC_CMD_LOW_LATENCY_ENABLE,
300);
            } else {
                // 同步关闭低延迟模式，300ms超时
                tws_api_sync_call_by_uuid('T', SYNC_CMD_LOW_LATENCY_DISABLE,
300);
            }
        } else {
            // 从设备未连接，直接设置本机低延迟模式
```

```
            printf("[%s,%d]----samson---enable = %d-
-/n",__func__,__LINE__,enable);
            // 设置低延迟模式，开启时延时800ms，关闭时延时600ms
            if (earphone_a2dp_codec_set_low_latency_mode(enable, enable ? 800
: 600) == 0) {
                printf("[%s,%d]----samson---enable = %d-
-/n",__func__,__LINE__,enable);
                // 根据模式播放对应的提示音
                if (enable) {
                    // 播放进入低延迟模式提示音
                    tone_play(TONE_LOW_LATENCY_IN, 1);
                } else {
                    // 播放退出低延迟模式提示音
                    tone_play(TONE_LOW_LATENCY_OUT, 1);
                }
            }
        }
    }
#else  // 未启用TWS功能
    // 直接设置低延迟模式
    earphone_a2dp_codec_set_low_latency_mode(enable, enable ? 800 : 600);
#endif
}
```

这段代码实现了蓝牙耳机的低延迟模式（游戏模式）控制功能，主要特点：

1. **TWS模式支持**：
    - 在TWS模式下，只有主设备可以控制低延迟模式
    - 从设备连接时，会通过TWS协议同步状态
    - 从设备未连接时，只设置本机状态
2. **提示音反馈**：
    - 开启低延迟模式时播放`TONE_LOW_LATENCY_IN`提示音
    - 关闭低延迟模式时播放`TONE_LOW_LATENCY_OUT`提示音
3. **延迟参数**：
    - 开启低延迟模式时设置800ms的延迟
    - 关闭低延迟模式时设置600ms的延迟
4. **调试信息**：
    - 使用`printf`输出调试信息，方便问题排查
5. **返回值处理**：

- `earphone_a2dp_codec_set_low_latency_mode`返回0表示设置成功

这个实现确保了在TWS模式下主从设备的低延迟状态保持同步，并提供了清晰的用户反馈。

弱符号定义

这段代码的作用：

1. **弱符号定义**：
    - 使用`__attribute__((weak))`声明这些函数为弱符号
    - 允许在其他文件中定义同名函数来覆盖这些默认实现
    - 如果没有其他实现，则使用这里的默认实现
2. **函数功能**：
    - `earphone_a2dp_codec_get_low_latency_mode()`：获取低延迟模式状态
        - 默认返回0，表示低延迟模式关闭
    - `earphone_a2dp_codec_set_low_latency_mode()`：设置低延迟模式
        - 参数`enable`控制开关
        - 参数`msec`指定延迟时间（毫秒）
        - 默认实现不做任何操作，直接返回成功(0)
3. **设计目的**：
    - 提供默认实现，避免链接错误
    - 允许在特定平台或配置下提供优化实现
    - 保持代码的可扩展性和灵活性
4. **实际使用**：
    - 这些函数的具体实现在`audio_dec.c`中
    - 在TWS模式下，主设备会调用这些函数来设置/获取低延迟状态
    - 实际实现会处理A2DP编解码器的低延迟参数调整

这种设计使得低延迟功能可以灵活地根据不同硬件平台或配置进行定制，同时保持接口的一致性。

调用时机

TWS（真无线）功能相关：

- 在 `bt_tws.c` 的 `bt_tws_poweron` 函数中，系统启动时会默认关闭低延迟模式

```
bt_set_low_latency_mode(0); // 开机默认关闭低延迟模式
```

按键事件处理：

- 在 `key_event_deal.c` 的 `app_earphone_key_event_handler` 函数中，通过按键切换低延迟模式的开关状态

```
bt_set_low_latency_mode(!bt_get_low_latency_mode());
```

远程配置（RCSP）相关：

- 在 `rcsp_adv_user_update.c` 中，当处理远程配置消息时，会关闭低延迟模式

```
bt_set_low_latency_mode(0);
```

- 在 `adv_work_setting.c` 中，根据工作设置开启或关闭低延迟模式

```
bt_set_low_latency_mode(0); // 关闭
bt_set_low_latency_mode(1); // 开启
```

通用接口：

- 在 `dhfcommon_interface.c` 的 `DhfCommon_iSetGameMode` 函数中，通过通用接口设置游戏模式

```
bt_set_low_latency_mode(*pucGameMode);
```

状态机：

- 在 `earphone.c` 的 `state_machine` 函数中，有一行被注释掉的代码

```
// bt_set_low_latency_mode(0); // 被注释掉的代码，用于在特定状态下关闭低延迟模式
```

总结：`bt_set_low_latency_mode` 函数主要在以下情况下被调用：

- 系统启动时初始化
- 用户通过物理按键切换模式时

- 通过手机APP远程控制时
- 通过通用接口设置游戏模式时

这些调用点共同构成了低延迟模式的控制逻辑，允许用户通过多种方式切换游戏/音乐模式。

关于低延迟模式持久性

在 `audio_dec.c` 中，低延迟模式的状态保存在静态变量 `a2dp_low_latency` 中，这个变量在设备重启后会被重置。

结论：

1. 低延迟模式的状态是**非持久性**的，因为：
    - 状态保存在 RAM 中的静态变量 `a2dp_low_latency` 中
    - 设备重启后，这个变量会被重新初始化为默认值
    - 没有发现将低延迟模式状态保存到非易失性存储（如 Flash）的代码
2. 低延迟模式的可用性：
    - 如果 `CONFIG_256K_FLASH` 被定义，则低延迟模式被禁用
    - 由于 `CONFIG_FLASH_SIZE` 被设置为 `FLASH_SIZE_1M`，`CONFIG_256K_FLASH` 很可能没有被定义，因此低延迟模式默认是启用的
3. 设备重启后，低延迟模式会恢复到默认状态（关闭状态），因为 `bt_tws_poweron` 函数中调用了 `bt_set_low_latency_mode(0)` 来禁用低延迟模式。
4. 如果需要实现低延迟模式的持久化，需要：
    - 在设置低延迟模式时，将状态保存到非易失性存储（如 Flash 或 EEPROM）
    - 在设备启动时，从非易失性存储中读取状态并恢复

这个结论与之前的分析一致，确认低延迟模式的状态不会在设备重启后保持。

流程图说明

```
graph TD
```

```
%% 样式定义
classDef startEnd fill:#e1f5fe,stroke:#01579b,stroke-width:3px,color:#000
classDef process fill:#f3e5f5,stroke:#4a148c,stroke-width:2px,color:#000
classDef decision fill:#fff3e0,stroke:#e65100,stroke-width:2px,color:#000
classDef event fill:#e8f5e8,stroke:#2e7d32,stroke-width:2px,color:#000
classDef system fill:#ffebee,stroke:#c62828,stroke-width:2px,color:#000
classDef audio fill:#f1f8e9,stroke:#558b2f,stroke-width:2px,color:#000

%% 流程图开始
START([开始]):::startEnd

%% 系统启动流程
START --> A[系统启动]:::system
A --> B["bt_tws_poweron()"]:::process
B --> C["bt_set_low_latency_mode(0)"]:::process
C --> D["earphone_a2dp_codec_set_low_latency_mode(0, 600)"]:::process
D --> E[设置a2dp_low_latency=0]:::process
D --> F[设置a2dp_low_latency_seqn=600]:::process
E --> READY[系统就绪]:::system
F --> READY

%% 按键事件处理
G[按键事件触发]:::event --> H["app_earphone_key_event_handler()"]:::process
H --> I["bt_set_low_latency_mode(!bt_get_low_latency_mode)"]:::process
I --> MODE_CHECK{检查当前模式}:::decision

%% 切换到低延迟模式
MODE_CHECK -->|当前为音乐模式| J["bt_set_low_latency_mode(1)"]:::process
J --> K["earphone_a2dp_codec_set_low_latency_mode(1, 800)"]:::process
K --> L[设置a2dp_low_latency=1]:::process
K --> M[设置a2dp_low_latency_seqn=800]:::process
L --> N["调用tws_api_low_latency_enable(1)"]:::process
M --> N
N --> O[播放TONE_LOW_LATENCY_IN提示音]:::audio
O --> LOW_LATENCY_MODE[低延迟模式激活]:::system

%% 切换到音乐模式
MODE_CHECK -->|当前为低延迟模式| P["bt_set_low_latency_mode(0)"]:::process
P --> Q["earphone_a2dp_codec_set_low_latency_mode(0, 600)"]:::process
Q --> R[设置a2dp_low_latency=0]:::process
Q --> S[设置a2dp_low_latency_seqn=600]:::process
R --> T["调用tws_api_low_latency_enable(0)"]:::process
S --> T
```

```
    T --> U[播放TONE_LOW_LATENCY_OUT提示音]:::audio
    U --> MUSIC_MODE[音乐模式激活]:::system

    %% TWS连接事件
    V[TWS连接事件]:::event --> W["bt_set_low_latency_mode()"]:::process
    W --> X[同步主从机低延迟状态]:::process
    X --> SYNC_COMPLETE[同步完成]:::system

    %% 远程控制
    Y[远程控制/APP]:::event --> Z["bt_set_low_latency_mode()"]:::process
    Z --> AA[设置低延迟/音乐模式]:::process
    AA --> REMOTE_SET[远程设置完成]:::system

    %% 音频数据处理
    AB[音频数据流输入]:::audio --> AC["low_latency_drop_a2dp_frame()"]:::process
    AC --> AUDIO_CHECK{检查延迟模式}:::decision
    AUDIO_CHECK -->|低延迟模式| AD[根据a2dp_low_latency_seqn丢弃部分音频
帧]:::audio
    AUDIO_CHECK -->|音乐模式| AE[正常处理所有音频帧]:::audio
    AD --> AUDIO_OUTPUT[音频输出]:::audio
    AE --> AUDIO_OUTPUT

    %% 系统关机
    AF[系统关机信号]:::event --> AG[低延迟状态丢失]:::system
    AG --> END([结束]):::startEnd

    %% 循环连接
    READY --> G
    LOW_LATENCY_MODE --> G
    MUSIC_MODE --> G
    SYNC_COMPLETE --> G
    REMOTE_SET --> G
    AUDIO_OUTPUT --> AB
```

流程图说明：

1. **系统启动**：
    - 调用`bt_tws_poweron()`初始化蓝牙
    - 默认设置为音乐模式（`bt_set_low_latency_mode(0)`）
    - 初始化相关变量（`a2dp_low_latency=0`,`a2dp_low_latency_seqn=600`）
2. **模式切换触发点**：

- 物理按键事件
- TWS 连接状态变化
- 远程控制/APP 指令

3. **低延迟模式**:
    - 设置`a2dp_low_latency=1`
    - 设置`a2dp_low_latency_seqn=800`
    - 调用 TWS 同步接口
    - 播放进入低延迟模式提示音
    - 音频处理层根据`a2dp_low_latency_seqn`丢弃部分音频帧
4. **音乐模式**:
    - 设置`a2dp_low_latency=0`
    - 设置`a2dp_low_latency_seqn=600`
    - 调用 TWS 同步接口
    - 播放退出低延迟模式提示音
    - 音频处理层正常处理所有音频帧
5. **TWS 同步**:
    - 主设备切换模式时，会通过 TWS 协议同步到从设备
    - 确保主从设备保持相同的低延迟状态
6. **音频数据处理**:
    - 在低延迟模式下，`low_latency_drop_a2dp_frame`函数会根据`a2dp_low_latency_seqn`丢弃部分音频帧
    - 在音乐模式下，所有音频帧都会被正常处理
7. **系统关机**:
    - 低延迟状态不会保存
    - 下次开机时恢复为默认的音乐模式

这个流程图展示了低延迟模式与音乐模式在整个系统生命周期中的切换过程，包括各种触发条件和状态变化。