浙江大學

《EDA技术应用》实验报告

项目名称：I2C 与 WM8731

姓　　名：

学　　号：

学　　院：生物医学工程与仪器科学学院

专　　业：生物医学工程

指导老师：黄添添、裘利坚

2026 年 5 月 12 日

浙江大学 实验报告

课程名称：EDA 技术应用　实验类型：EDA 实验　指导老师：黄添添、裘利坚

实验名称：I2C 与 WM8731

姓　　名：　　学　　号：　　专　　业：生物医学工程

实验地点：玉泉教六 204　日　　期：2026 年 5 月 12 日

目　录

一、实验目的

(1) 了解 Quartus II 开发环境，熟悉 Verilog 语言基本语法

(2) 了解 I2C 总线协议

(3) 通过代码控制音频编/解码硬件芯片 WM8731，通过按键控制音频输出的音量

二、实验内容

实验指导资料：DE2_115_i2scond 工程及 WM8731 等手册

通过按键控制音频输出音量（可结合 LCD 显示音量等信息）

三、主要仪器设备

Quartus ii 13.1, WM8731 芯片与音响设备，DE2-115 开发板。

四、操作方法和实验步骤

4.1 I2C 控制与 WM8731 寻址机制

根据通信规范，FPGA 作为主控设备，通过 `I2C_SCLK`（时钟线）和 `I2C_SDAT`（数据线）与 WM8731 芯片通信。在 DE2-115 开发板中，WM8731 的 `CSB` 引脚已被拉低接地，因此该芯片在 I2C 总线上的设备读地址固定为 `0x34`，写地址为 `0x35`。在每次完整的 I2C 通信中，FPGA 会连续发送 24 位（Bit）数据流：

- **高 8 位：** 从设备地址（此处配置为 `0x34`）。
- **中 7 位：** 目标寄存器地址（SUB_ADDR）。
- **低 9 位：** 写入该寄存器的配置数据。

WM8731 的寄存器地址为 7 位，数据为 9 位，二者在底层代码拼接为 16 位的 `LUT_DATA` 后发送。

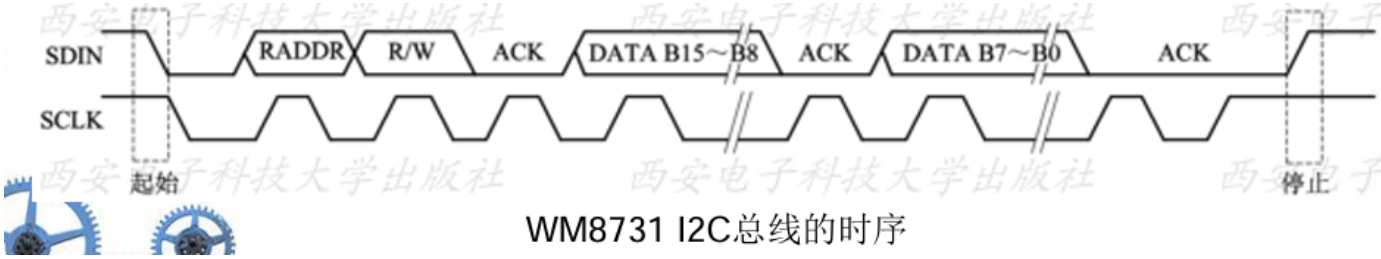

图 4.1 – I2C 总线时序

4.2 I2C 传输状态机设计

在完成 24 位音频配置数据（设备地址、寄存器地址、配置值）的组装后，需要通过状态机将这 11 个参数有序地传输给 WM8731 芯片。为确保 I2C 总线传输的稳定性和避免总线阻塞，本实验设计了具有 4 个核心状态的有限状态机。

在传输过程中，有两个关键的握手信号：

- `mI2C_END`：由底层 I2C 控制器生成，为高电平时标志 24 位数据传输完成。
- `mI2C_ACK`：应答校验信号。当从机设备成功接收时，该信号会被拉低（低电平有效）。

状态机的运转逻辑如下：

- **State 0**：配置并装载 24 位待传输参数，拉高 `mI2C_GO` 启动底层 I2C 传输。

- **State 1**：等待 mI2C_END 信号。传输结束后检测 mI2C_ACK，若为 0 (接收成功) 则进入 State 2；若为 1 (无响应) 则退回 State 0 重新发送。
- **State 2**：传输成功后判断 LUT_INDEX 是否达到上限。若未达到，则索引加一返回 State 0；若全部发送完毕，进入 State 3。
- **State 3**：休眠待机状态。当全部 11 个参数初始化结束后，系统停留在该状态释放总线。仅当检测到按键触发信号 VOL 为高时，将指针直接拉回音量寄存器并唤醒状态机。

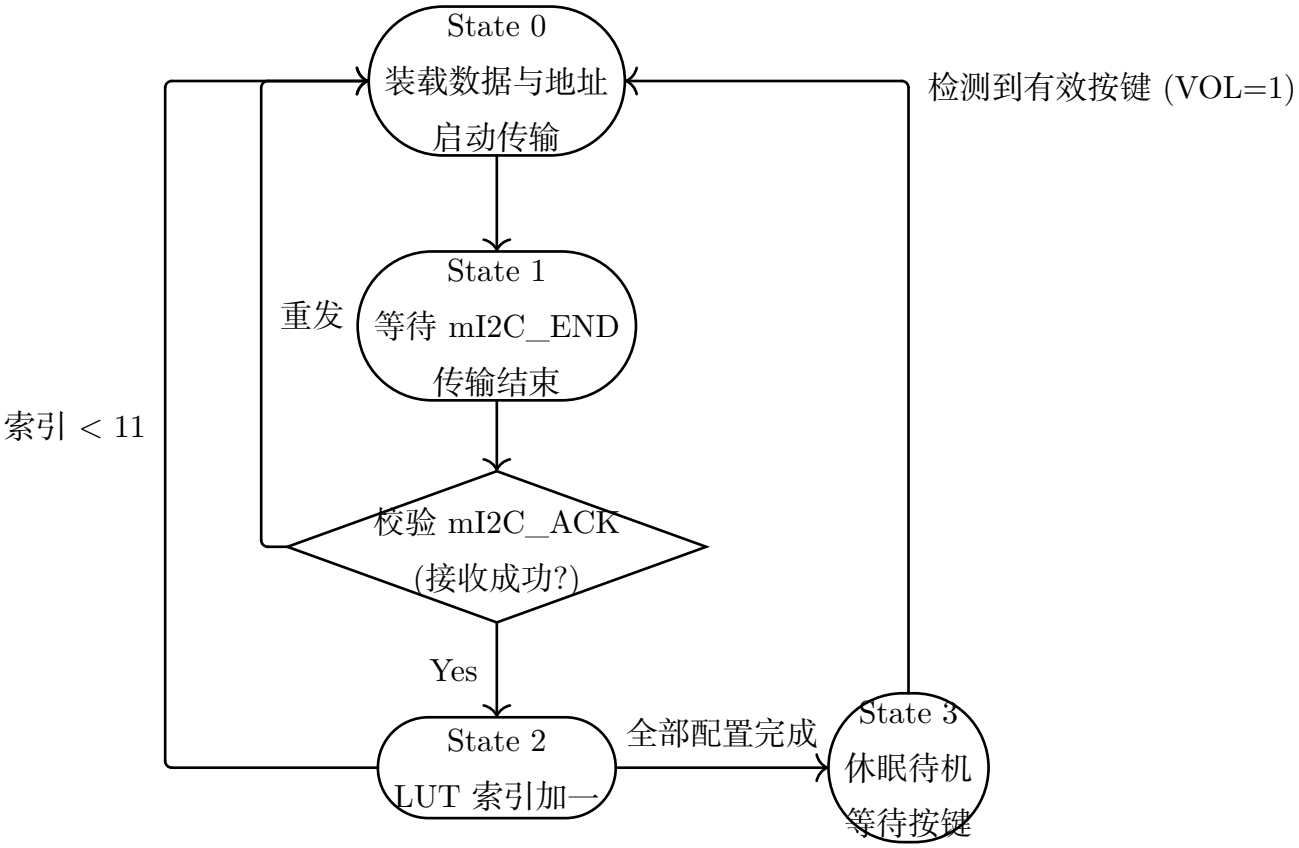

图 4.2 - I2C 参数传输与音量动态调节状态机流程图

状态机的核心实现代码如下：

```
////////////////////// I2C 传输状态机 ////////////////////////////
always@(posedge mI2C_CTRL_CLK or negedge iRST_N) begin
  if(!iRST_N) begin
    LUT_INDEX <=  0;
    mSetup_ST <=  0;
    mI2C_GO   <=  0;
  end else begin
    if(LUT_INDEX < LUT_SIZE) begin
      case(mSetup_ST)
      0: begin // 状态0: 准备数据并启动 I2C 传输
        mI2C_DATA <= {8'h34, LUT_DATA}; // 组合24位数据: 8'h34为设备写地址, LUT_DATA为16位寄存器配置数据
        mI2C_GO   <= 1;
        mSetup_ST <= 1;
      end
      1: begin // 状态1: 等待 I2C 传输结束并检查应答位 (ACK)
        if(mI2C_END) begin
          if(!mI2C_ACK) // mI2C_ACK 正常为低电平有效
              mSetup_ST <= 2; // 接收到ACK, 说明从机成功接收, 进入下一个状态
          else
```

```
                mSetup_ST <= 0; // 无ACK，说明传输错误，重试当前数据
              mI2C_GO <= 0;
            end
          end
          2: begin // 状态2: 传输成功，LUT 索引加 1，准备发送下一个配置参数
            LUT_INDEX <= LUT_INDEX + 1;
            mSetup_ST <= 0;
          end
          endcase
        end else begin
                // 全部 11 个参数配置完成后，如果按键被按下触发了音量改变，
                // 强制将索引指向最后一个寄存器 （音量控制），实现动态调音
          LUT_INDEX <= 6'd10;
        end
      end
    end
```

4.3 音量调节模块逻辑

本实验中的核心交互功能是通过按键动态控制音频输出音量。在代码中维护了一个 7 位的寄存器变量 `volum`，其初始值被设定为 `7'b0110000`（即 0x30，对应 −12dB）。当接收到消抖模块传入的有效按键单脉冲信号 `VOL` 时，若当前音量未达到上限 `7'b1111111`（+6dB），则进行累加操作；若已达最大值，则触发保护机制，将其回滚至初始默认音量，形成闭环循环。

```
////// 改变音量逻辑 (Change Voice) //////
always@(posedge mI2C_CTRL_CLK or negedge iRST_N) begin
    if(!iRST_N) begin
        volum <= 7'b0110000; // 复位时设置默认音量 (0x30 = -12dB)
    end else if(VOL) begin
        // 当接收到有效的按键脉冲时调节音量
        if(volum < 7'b1111111) begin
            volum <= volum + 1'b1; // 音量未满，加 1
        end else if(volum == 7'b1111111) begin
            volum <= 7'b0110000; // 达到最大值后，循环回到默认音量
        end
    end else begin
        volum <= volum; // 无操作时保持当前音量不变
    end
end
```

4.4 寄存器参数配置与查找表 (LUT)

WM8731 的初始化配置过程被封装于一个基于 `case` 语句构建的查找表（Index Table）中。表中记录了 11 个核心指令，涵盖了模拟线路激活、DAC 路径打通（`0x08F8`）、模块上电（`0x0C00`）以及 I2S 音频数据格式配置（`0x0E01`）等。为了实现音量动态刷新，查找表的最后一个索引（`LUT_INDEX == 10`）并未写死定值，而是采用了动态拼接：高 8 位固定为左耳机音量控制寄存器地址 `0x05` 及 1 位填充位 `0`，低 7 位直接挂载前文生成的 `volum` 变量。当音量发生改变时，状态机会重新发送该行参数实现调音。

```
//////////////////// 查找表 (Index Table) ////////////////////////
always @(*) begin
  case(LUT_INDEX)
        // 这些 16 位的十六进制数分别代表了 WM8731 的不同寄存器设置
        SET_LIN_L:    LUT_DATA <= 16'h001A; // 左通道线路输入静音
        SET_LIN_R:    LUT_DATA <= 16'h021A; // 右通道线路输入静音
        SET_HEAD_L:   LUT_DATA <= 16'h0530; // 左耳机输出音量设置
```

```
        SET_HEAD_R:   LUT_DATA <= 16'h0630; // 右耳机输出音量设置
        A_PATH_CTRL:  LUT_DATA <= 16'h08F8; // 模拟音频路径控制 (DAC 选中)
        D_PATH_CTRL:  LUT_DATA <= 16'h0A06; // 数字音频路径控制
        POWER_ON:     LUT_DATA <= 16'h0C00; // 电源管理 (全部模块开启)
        SET_FORMAT:   LUT_DATA <= 16'h0E01; // 音频数据格式 (16位 MSB对齐, 或者 I2S)
        SAMPLE_CTRL:  LUT_DATA <= 16'h1002; // 采样率控制
        SET_ACTIVE:   LUT_DATA <= 16'h1201; // 激活 WM8731 接口
        VOL_CONTRAL:  LUT_DATA <= {8'h05, 1'b0, volum}; // 动态音量调节配置映射
    endcase
end
```

4.5 按键消抖机制实现

机械开关在按下与松开的瞬间，其金属弹片会产生物理抖动，使得输出的电平信号在毫秒级别内反复横跳。若不作处理，FPGA 的高频时钟会将其误认为成百上千次的高频连击。为解决此问题，本模块在 `20kHz` 的控制时钟域内设计了两级同步加延时消抖逻辑：

(1) **两级同步：** 引入 `KEY_flop1` 和 `KEY_flop2` 两个触发器，将外部异步的按键信号同步到本地时钟域，消除亚稳态。

(2) **延时计数器：** 当检测到按键处于低电平时，启动一个 8 位计数器 `count` 开始计数。只有当低电平信号稳定保持 `200` 个时钟周期（约 $200 \times 50\mu s = 10\text{ms}$）时，才确认为有效的人工按压，并在下一时刻向后级输出一个单时钟周期的脉冲信号 `KEY_valid`。

```
// 第一级: 同步按键信号, 防止亚稳态
always @(posedge CLOCK or negedge RESET) begin
    if (!RESET) begin
        KEY_flop1 <= 1'b1;
        KEY_flop2 <= 1'b1;
    end else begin
        KEY_flop1 <= KEY[0];      // 采集按键0
        KEY_flop2 <= KEY_flop1;   // 打一拍
    end
end

// 第二级: 延时消抖与边沿检测脉冲输出
always @(posedge CLOCK or negedge RESET) begin
    if (!RESET) begin
        count <= 8'd0;
        KEY_valid <= 1'b0;
    end else begin
        KEY_valid <= 1'b0; // 默认输出低电平

        // 如果按键被按下 (低电平), 开始计数
        if (KEY_flop2 == 1'b0) begin
            if (count < 8'd200) begin
                count <= count + 1'b1; // 20kHz时钟下, 计数200次约等于10ms
            end else if (count == 8'd200) begin
                KEY_valid <= 1'b1;     // 延时满10ms, 确认按键稳定, 输出一个单脉冲
                count <= count + 1'b1; // 加1防止一直停留在200重复触发
            end
        end else begin
            // 按键松开, 计数器清零, 为下一次按压做准备
            count <= 8'd0;
        end
    end
end
```

五、实验结果与分析

将程序全编译并下载入 FPGA 后，解除了位于开发板侧边的 SW[17] 复位拨码开关。随后将音频数据线一端接入外部音源，另一端接入开发板的蓝色 Line-In 接口，耳机接入绿色 Line-Out 接口。

复位后，板载红色 LED 阵列（LEDR[6:0]）显示初始音量为 0x30（即中等音量）。按下 KEY[0] 后：

- 能够清晰听到耳机中播放的音乐音量阶梯式放大。
- 板载红灯以二进制加法的形式不断进位闪烁，绿色指示灯 LEDG[0] 也准确伴随物理按键的按下而点亮。
- 当音量累加达到上限 7'b1111111 时，再次按键，音量会自动回落至初始值，形成安全循环。这表明 FPGA 内部的 I2C 状态机及消抖模块均完美运作。

六、讨论与心得

在本次硬件实验中，一开始系统无法正常工作，排查后发现是没有为 WM8731 分配提供 AUD_XCK 主时钟，且将其配置为了 Slave（从机模式），导致芯片内部数字逻辑停摆。随后在顶层例化了 pll 锁相环模块后得以正常工作。

在数字代码确认无误后，仍然无法听到电脑发出的声音，但用手指触摸接口却能听到被芯片放大的 50Hz 静电底噪。经过深入探索，发现这是因为部分现代电脑的声卡具备高敏阻抗检测保护机制，检测到 FPGA 开发板极高的输入阻抗后，强行在底层截断了音频电流输出。

本次实验让我深刻认识到，硬件工程远不仅仅是编写对 Verilog 语法，更是对底层协议、时序逻辑以及跨域物理硬件匹配（阻抗、时钟同步）的综合考验。

七、附录

7.1 Audio_top.v

```
module Audio_top(
    input  RESET,           // 系统复位拨码开关
    input  CLOCK_50,        // 50MHz板载高频时钟

    // --- 音频硬件接口 ---
    input  AUD_ADCDAT,      // ADC 录音数字输入
    inout  AUD_ADCLRCK,     // 左右声道对齐信号
    inout  AUD_BCLK,        // 音频位时钟
    output AUD_DACDAT,      // DAC 播放数字输出
    inout  AUD_DACLRCK,
    output AUD_XCK,         // 提供给 WM8731 的主时钟

    // --- I2C 控制接口 ---
    output I2C_SCLK,        // I2C 时钟
    inout  I2C_SDAT,        // I2C 数据双向总线

    // --- 外设接口 ---
    output [8:0]  LEDG,
    output [17:0] LEDR,
    input  [3:0]  KEY
);

wire KEY_valid;             // 消抖后按键脉冲
wire [6:0] volum;           // 动态音量值

// 系统时钟分频器定义
parameter CLK_Freq = 50000000;
parameter I2C_Freq = 20000;
reg [15:0] mI2C_CLK_DIV;
reg mI2C_CTRL_CLK;          // 生成的 20kHz 慢速控制时钟

// --- 硬件纯物理直通与状态显示 ---
assign AUD_DACDAT = AUD_ADCDAT;  // 核心：输入音频数据直通输出
assign LEDG[0]  = ~KEY[0];       // 绿灯显示按键物理状态
assign LEDR[17] = ~RESET;        // 提示系统是否处于复位锁死状态
assign LEDR[6:0] = volum;        // 二进制红灯实时显示音量

// --- 50MHz 降频至 20kHz ---
always@(posedge CLOCK_50 or negedge RESET) begin
    if(!RESET) begin
        mI2C_CTRL_CLK <= 0;
        mI2C_CLK_DIV  <= 0;
    end else begin
        if(mI2C_CLK_DIV < (CLK_Freq/I2C_Freq))
            mI2C_CLK_DIV <= mI2C_CLK_DIV + 1'b1;
        else begin
            mI2C_CLK_DIV  <= 0;
            mI2C_CTRL_CLK <= ~mI2C_CTRL_CLK;
        end
    end
end

// --- 实例化各子模块 ---
I2C_Audio_Config I2C_Config(
    .iRST_N(RESET),
    .I2C_SCLK(I2C_SCLK),
    .I2C_SDAT(I2C_SDAT),
    .VOL(KEY_valid),
    .volum(volum),
    .mI2C_CTRL_CLK(mI2C_CTRL_CLK)
);

```

```
63  KEY_DEBOUNCE KEY0(
64      .CLOCK(mI2C_CTRL_CLK),
65      .RESET(RESET),
66      .KEY(KEY),
67      .KEY_valid(KEY_valid)
68  );
69
70  pll1 u_pll (
71      .inclk0(CLOCK_50),
72      .c0(AUD_XCK)            // 输出心脏时钟给编解码器
73  );
74  endmodule
```

7.2 按键延时消抖模块（KEY_DEBOUNCE.v）

```
1   module KEY_DEBOUNCE(
2       input CLOCK,            // 采样时钟 20kHz
3       input [3:0] KEY,
4       input RESET,
5       output reg KEY_valid    // 输出干净的单脉冲
6   );
7
8   reg KEY_flop1, KEY_flop2;
9   reg [7:0] count;            // 计数器用于 10ms 延时
10
11  // 第一级：同步打拍，防止亚稳态干扰
12  always @(posedge CLOCK or negedge RESET) begin
13      if (!RESET) begin
14          KEY_flop1 <= 1'b1;
15          KEY_flop2 <= 1'b1;
16      end else begin
17          KEY_flop1 <= KEY[0];
18          KEY_flop2 <= KEY_flop1;
19      end
20  end
21
22  // 第二级：延时消抖逻辑
23  always @(posedge CLOCK or negedge RESET) begin
24      if (!RESET) begin
25          count <= 8'd0;
26          KEY_valid <= 1'b0;
27      end else begin
28          KEY_valid <= 1'b0;
29
30          // 当按键被按下（低电平）时开始计时
31          if (KEY_flop2 == 1'b0) begin
32              if (count < 8'd200) begin
33                  count <= count + 1'b1; // 累加至200次即约 10ms
34              end else if (count == 8'd200) begin
35                  KEY_valid <= 1'b1;     // 延时满足，触发有效脉冲
36                  count <= count + 1'b1; // 加1防死锁重复触发
37              end
38          end else begin
39              count <= 8'd0; // 松开按键随时清零
40          end
41      end
42  end
43  endmodule
```

7.3 I2C 寄存器状态机配置模块（I2C_Audio_Config.v）

```
1   module I2C_Audio_Config (
```

```
    input  iRST_N,
    output I2C_SCLK,
    inout  I2C_SDAT,
    input  VOL,                     // 按键触发输入
    output reg [6:0] volum,         // 实时音量输出给 LED
    input  mI2C_CTRL_CLK            // 20kHz 控制时钟
);

reg [23:0] mI2C_DATA;
reg        mI2C_GO;
reg [15:0] LUT_DATA;
reg [5:0]  LUT_INDEX;
reg [3:0]  mSetup_ST;

wire mI2C_END;
wire mI2C_ACK;
parameter LUT_SIZE = 11;

// --- 音量安全累加控制逻辑 ---
always@(posedge mI2C_CTRL_CLK or negedge iRST_N) begin
    if(!iRST_N) begin
        volum <= 7'b0110000; // 默认音量 0x30
    end else if(VOL) begin
        if(volum < 7'b1111111) begin
            volum <= volum + 1'b1;
        end else if(volum == 7'b1111111) begin
            volum <= 7'b0110000; // 满后循环复位
        end
    end
end

// --- I2C 传输状态机核心 ---
always@(posedge mI2C_CTRL_CLK or negedge iRST_N) begin
    if(!iRST_N) begin
        LUT_INDEX <= 0;
        mSetup_ST <= 0;
        mI2C_GO   <= 0;
    end else begin
        case(mSetup_ST)
            0: begin
                // 装载设备写地址 0x34 与配置数据
                mI2C_DATA <= {8'h34, LUT_DATA};
                mI2C_GO   <= 1;
                mSetup_ST <= 1;
            end
            1: begin
                if(mI2C_END) begin
                    if(!mI2C_ACK) mSetup_ST <= 2; // ACK=0代表接收成功
                    else          mSetup_ST <= 0; // 否则重试
                    mI2C_GO <= 0;
                end
            end
            2: begin
                // 逐个发送 11 个参数，发完则进入休眠态
                if (LUT_INDEX < LUT_SIZE - 1) begin
                    LUT_INDEX <= LUT_INDEX + 1;
                    mSetup_ST <= 0;
                end else begin
                    mSetup_ST <= 3;
                end
            end
            3: begin
                // 休眠守候态：彻底解放总线，仅在按键按下时发单次音量数据
                if (VOL) begin
                    LUT_INDEX <= 6'd10;
```

```
                    mSetup_ST <= 0;
                end
            end
        endcase
    end
end

// --- 寄存器指令查找表 ---
always @(*) begin
    case(LUT_INDEX)
        0:  LUT_DATA <= 16'h001A; // 左通道线路输入静音
        1:  LUT_DATA <= 16'h021A; // 右通道线路输入静音
        2:  LUT_DATA <= 16'h0530; // 左耳机默认音量
        3:  LUT_DATA <= 16'h0630; // 右耳机默认音量
        // 关闭 Bypass 旁路，强制打通 ADC->FPGA->DAC 的纯数字通道
        4:  LUT_DATA <= 16'h0812;
        5:  LUT_DATA <= 16'h0A06; // 数字音频路径控制
        6:  LUT_DATA <= 16'h0C00; // 打开所有电源模块
        // 开启 I2S 格式，并强制 WM8731 为 Master 主模式自己产生位时钟
        7:  LUT_DATA <= 16'h0E42;
        8:  LUT_DATA <= 16'h1002; // 采样率控制
        9:  LUT_DATA <= 16'h1201; // 激活音频接口
        10: LUT_DATA <= {8'h05, 1'b0, volum}; // 将按键累加音量挂载至高位地址
    endcase
end

// 底层通信接口例化
i2c u0(
    .CLOCK(mI2C_CTRL_CLK),
    .I2C_SCLK(I2C_SCLK),
    .I2C_SDAT(I2C_SDAT),
    .I2C_DATA(mI2C_DATA),
    .GO(mI2C_GO),
    .END(mI2C_END),
    .ACK(mI2C_ACK),
    .RESET(iRST_N)
);
endmodule
```