

psoc_e84-edgi-talk SDK

开发文档

Edgi-Talk 开发板基于 *PSoC™ E84 MCU*，为工程师提供了一个灵活、全面的开发平台。板上集成了丰富的外设接口和示例模块，助力开发者快速完成多传感器、显示和通信应用的开发。

Version 1.0.0

中文

2025年10月27日

www.rt-thread.org

Copyright (c) 2006-2025, RT-Thread Development Team

目录

- 1. 基础篇
 - 1.1. Edgi-Talk_M33_Blink_LED 示例工程
 - 1.2. Edgi-Talk_M55_Blink_LED 示例工程
 - 1.3. Edgi-Talk_M33_S_Template 示例工程
 - 1.4. Edgi-Talk_M33_Template 示例工程
- 2. 驱动篇
 - 2.1. Edgi-Talk_ADC 示例工程
 - 2.2. Edgi-Talk_M33_AHT20 示例工程
 - 2.3. Edgi-Talk_Audio 示例工程
 - 2.4. Edgi-Talk_emUSB-device_CDC_Echo 示例工程
 - 2.5. Edgi-Talk_M33_HyperRam 示例工程
 - 2.6. Edgi-Talk_M33_S_HyperRam 示例工程
 - 2.7. Edgi-Talk_Key_Irq 示例工程
 - 2.8. Edgi-Talk_LSM6DS3 示例工程
 - 2.9. Edgi-Talk_M55_MIPI_LCD 示例工程
 - 2.10. Edgi-Talk_RTC 示例工程
 - 2.11. Edgi-Talk_SDCARD 示例工程
 - 2.12. Edgi-Talk_WIFI 示例工程
- 3. 示例
 - 3.1. Edgi-Talk_M55_CoreMark 示例工程
 - 3.2. Edgi-Talk_M55_LVGL 示例工程
 - 3.3. Edgi-Talk_WavPlayer 示例工程
 - 3.4. 小智示例工程

1. 基础篇

1.1. Edgi-Talk_M33_Blink_LED 示例工程

[中文](#) | [English](#)

简介

本示例工程基于 **Edgi-Talk 平台**，演示 蓝色 LED 灯闪烁 功能，运行在 **RT-Thread 实时操作系统** 上。

通过本工程，用户可以快速验证板级 GPIO 配置及 LED 控制逻辑，为后续硬件控制和应用开发提供基础参考。

GPIO 简介

GPIO (General Purpose Input/Output) 是 MCU 最常用的外设接口之一，能够在软件控制下配置为 **输入模式** 或 **输出模式**：

- **输入模式**：用于读取外部电平状态，例如按键输入。
- **输出模式**：用于控制外设电平，例如点亮 LED、驱动蜂鸣器。

RT-Thread 对 GPIO 的抽象

RT-Thread 提供了 **PIN 设备驱动框架**，通过统一的接口屏蔽底层硬件差异：

- `rt_pin_mode(pin, mode)`：设置引脚工作模式（输入/输出/上拉/下拉等）
- `rt_pin_write(pin, value)`：输出电平（高/低）
- `rt_pin_read(pin)`：读取输入电平

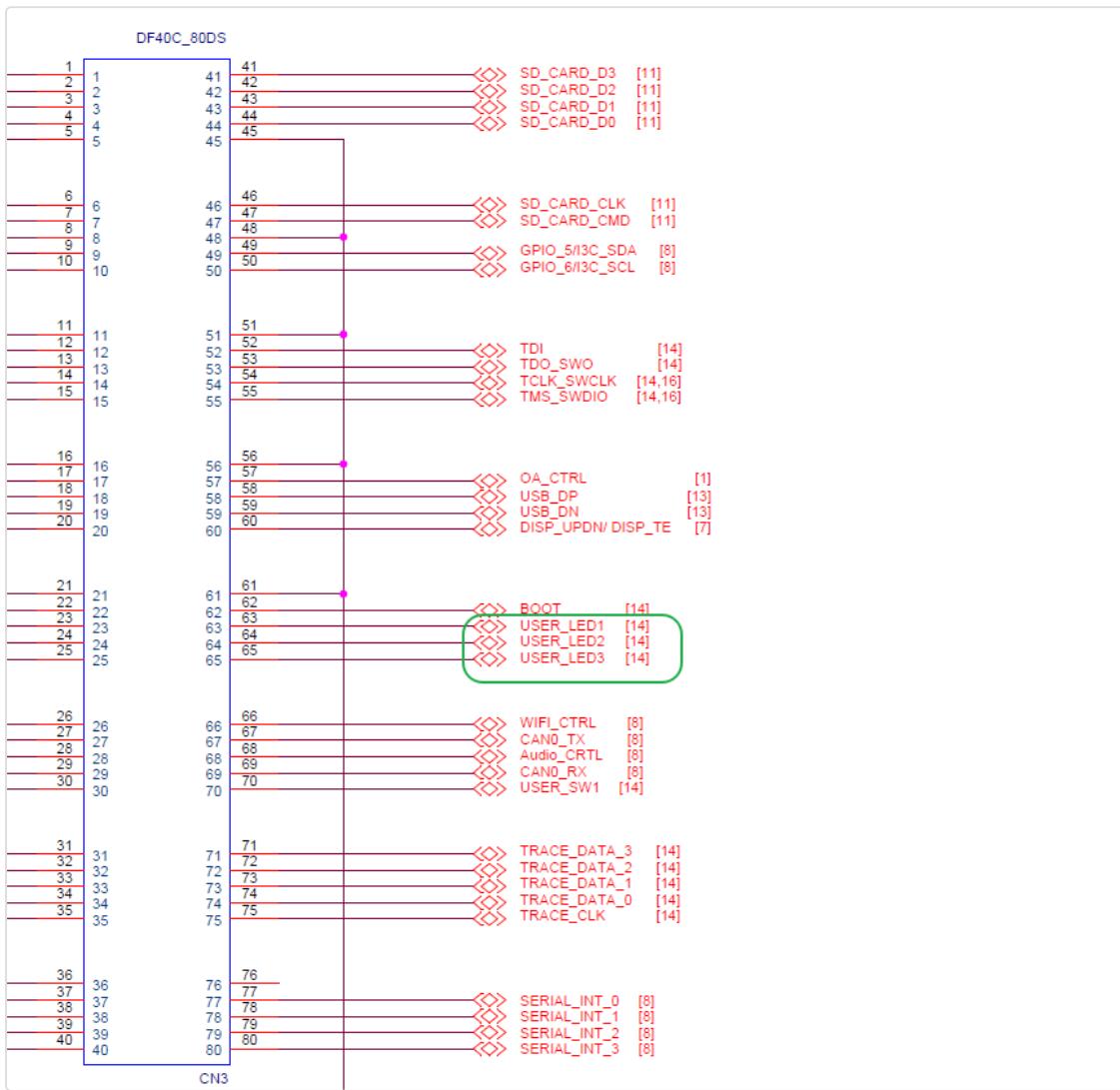
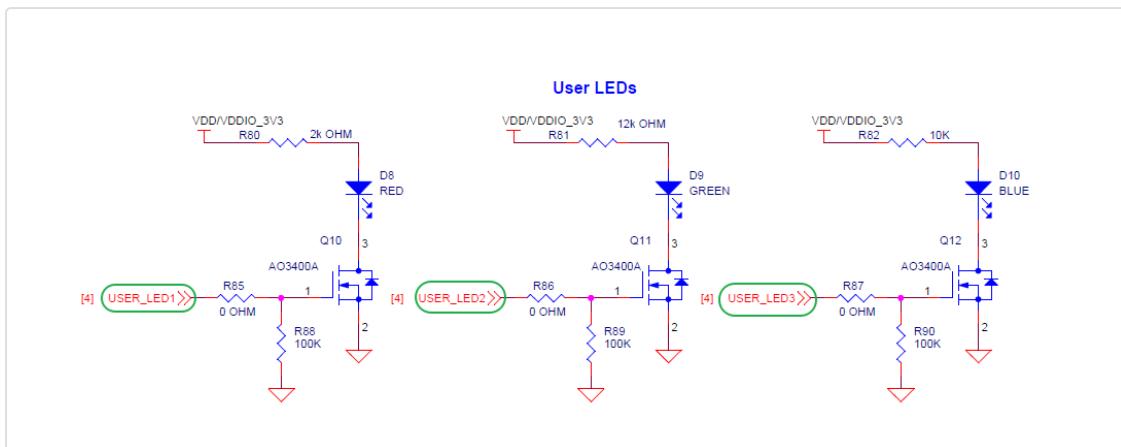
这样开发者不需要直接操作寄存器，而是通过 RT-Thread 的 API 即可完成 GPIO 控制。

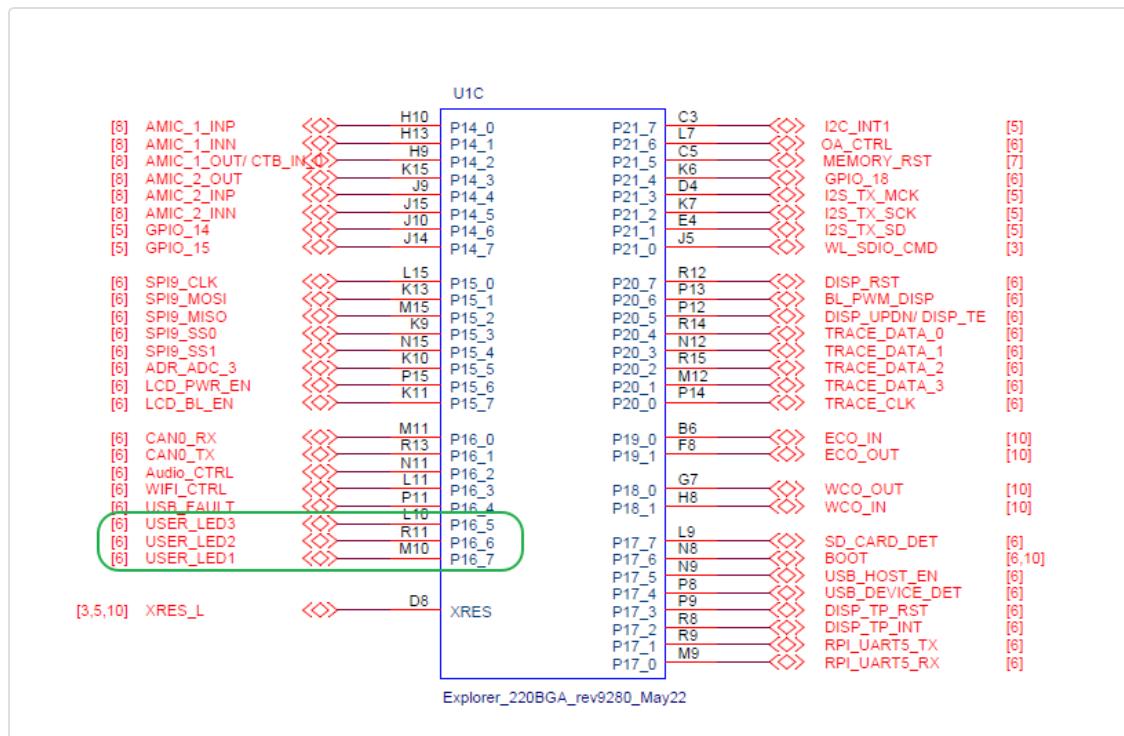
在本示例中，LED 引脚被配置为 **输出模式**，软件循环输出高低电平，从而实现 LED 闪烁。

软件说明

- 工程基于 **Edgi-Talk** 平台开发。
- 示例功能包括：
 - 蓝色 LED 灯周期性闪烁
 - GPIO 输出控制
- 工程结构简洁，便于理解 LED 控制逻辑及硬件驱动接口。

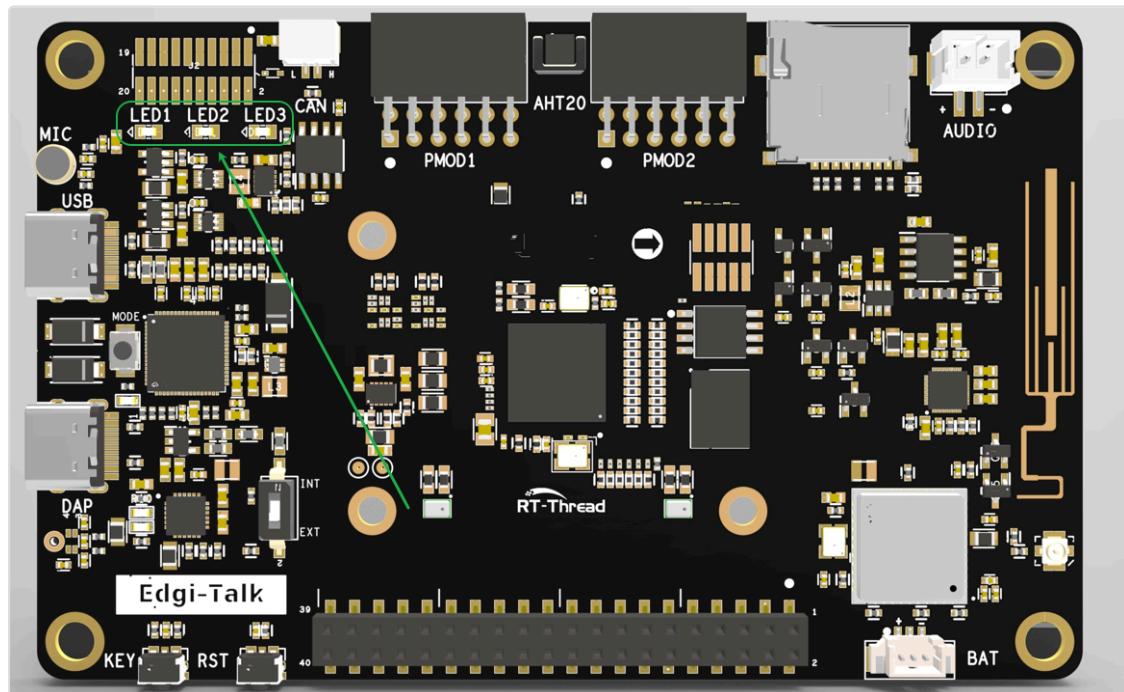
硬件说明





如上图所示，Edgi-Talk 提供三个用户 LED，分别为 USER_LED1 (RED)、USER_LED2 (GREEN)、USER_LED3 (BLUE)，其中 USER_LED2 对应引脚 P16_6。单片机引脚输出高电平即可点亮LED，输出低电平则会熄灭LED。

LED在开发板中的位置如下图所示：



使用方法

编译与下载

1. 打开工程并完成编译。
2. 使用 **板载下载器 (DAP)** 将开发板的 USB 接口连接至 PC。
3. 通过编程工具将生成的固件烧录至开发板。

运行效果

- 烧录完成后，开发板上电即可运行示例工程。
- 蓝色 LED 灯每 500ms 闪烁一次，表示系统 GPIO 控制和调度正常。
- 用户可根据需求修改闪烁周期或 LED 控制逻辑。

注意事项

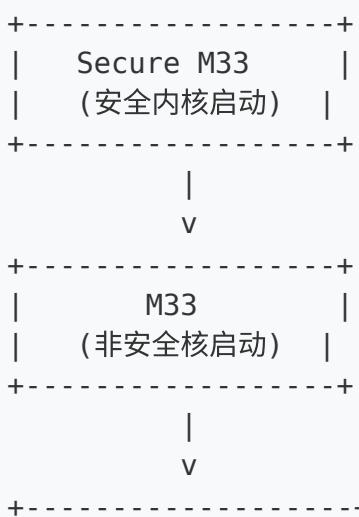
- 如需修改工程的 **图形化配置**，请使用以下工具打开配置文件：

```
tools/device-configurator/device-configurator.exe  
libs/TARGET_APP_KIT_PSE84_EVAL_EPC2/config/design.modus
```

- 修改完成后保存配置，并重新生成代码。

启动流程

系统启动顺序如下：



M55
(应用处理器启动)

⚠ 请严格按照以上顺序烧写固件，否则系统可能无法正常运行。

- 若示例工程无法正常运行，建议先编译并烧录 **Edgi-Talk_M33_S_Template** 工程，确保初始化与核心启动流程正常，再运行本示例。
- 若要开启 M55，需要在 **M33 工程** 中打开配置：

```
RT-Thread Settings --> 硬件 --> select SOC Multi Core Mode -->
```

1.2. Edgi-Talk_M55_Blink_LED 示例工程

[中文](#) | [English](#)

简介

本示例工程基于 **Edgi-Talk 平台**，演示 **绿色 LED 灯闪烁** 功能，运行在 **RT-Thread 实时操作系统** 上。

通过本工程，用户可以快速验证板级 GPIO 配置及 LED 控制逻辑，为后续硬件控制和应用开发提供基础参考。

GPIO 简介

GPIO (General Purpose Input/Output) 是 MCU 最常用的外设接口之一，能够在软件控制下配置为 **输入模式** 或 **输出模式**：

- **输入模式**：用于读取外部电平状态，例如按键输入。
- **输出模式**：用于控制外设电平，例如点亮 LED、驱动蜂鸣器。

RT-Thread 对 GPIO 的抽象

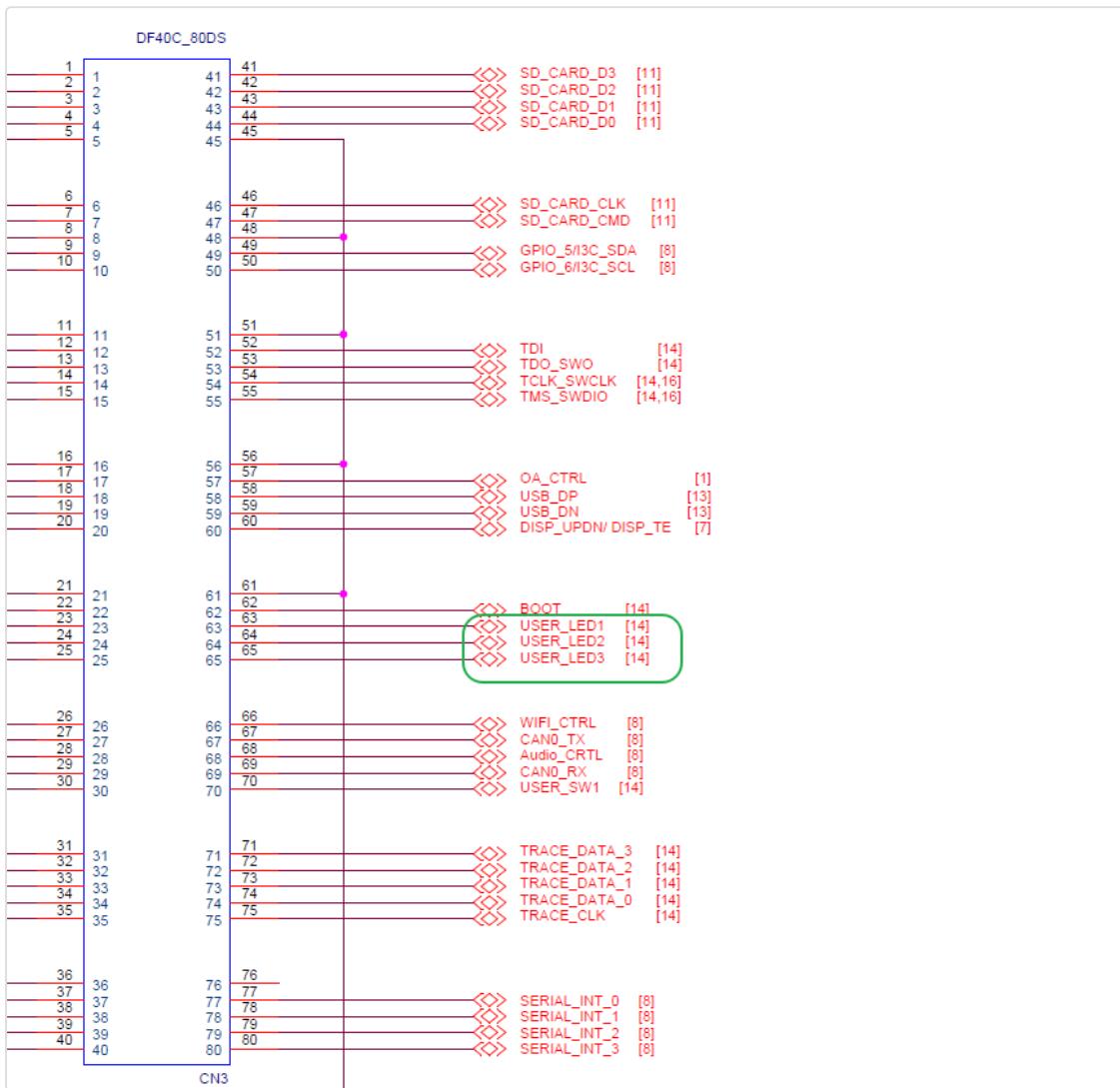
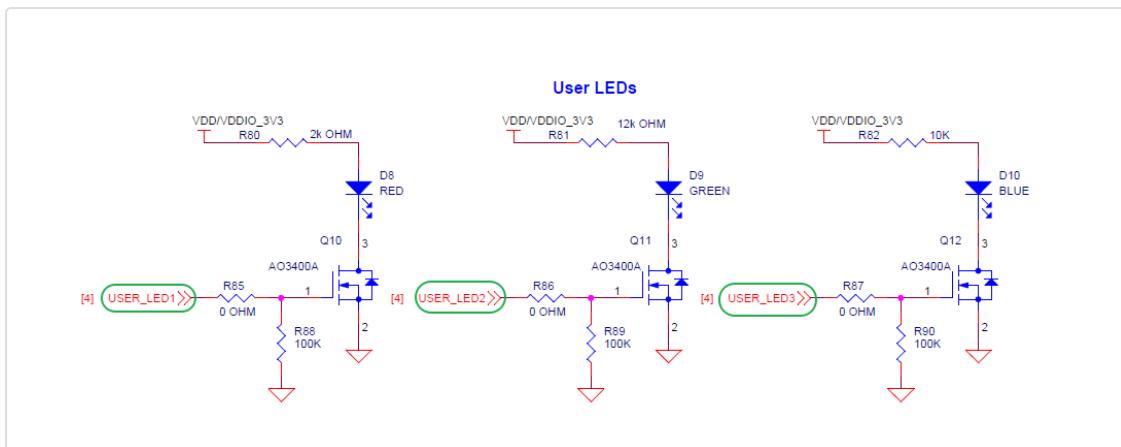
RT-Thread 提供了 **PIN 设备驱动框架**，通过统一的接口屏蔽底层硬件差异：

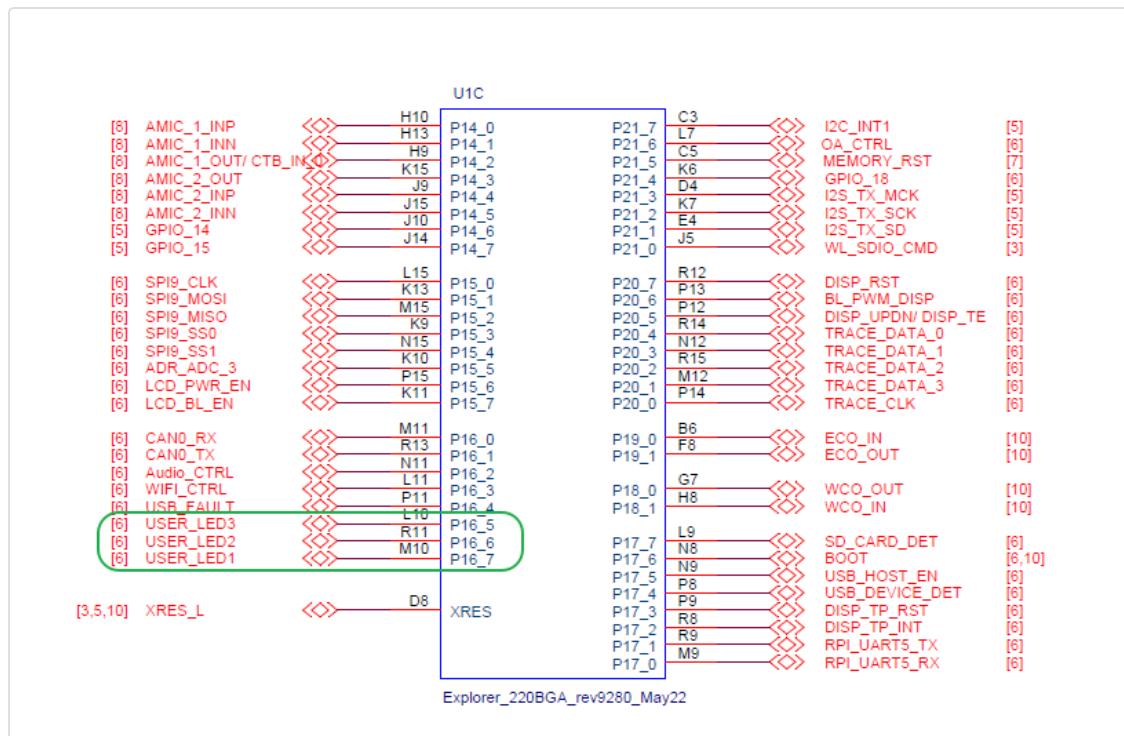
- `rt_pin_mode(pin, mode)`：设置引脚工作模式（输入/输出/上拉/下拉等）
- `rt_pin_write(pin, value)`：输出电平（高/低）
- `rt_pin_read(pin)`：读取输入电平

这样开发者不需要直接操作寄存器，而是通过 RT-Thread 的 API 即可完成 GPIO 控制。

在本示例中，LED 引脚被配置为 **输出模式**，软件循环输出高低电平，从而实现 LED 闪烁。

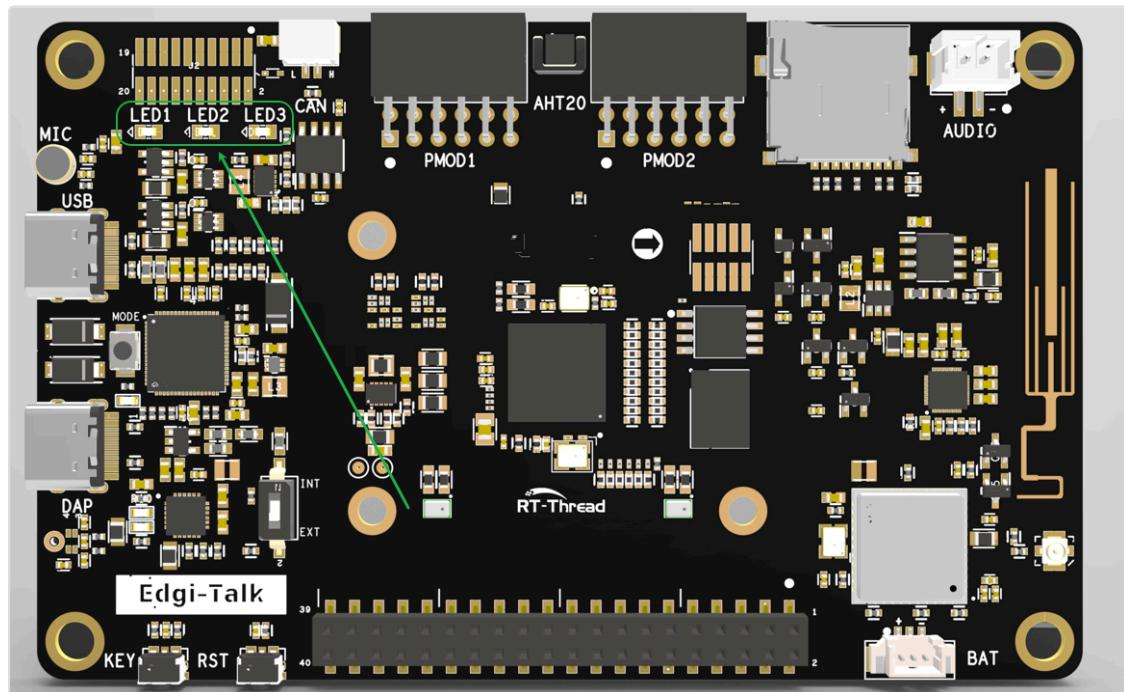
硬件说明





如上图所示，Edgi-Talk 提供三个用户 LED，分别为 USER_LED1 (RED)、USER_LED2 (GREEN)、USER_LED3 (BLUE)，其中 USER_LED2 对应引脚 P16_7。单片机引脚输出高电平即可点亮LED，输出低电平则会熄灭LED。

LED在开发板中的位置如下图所示：



软件说明

- 工程基于 **Edgi-Talk** 平台开发。
- 示例功能包括：
 - 绿色 LED 灯周期性闪烁
 - GPIO 输出控制
- 工程结构简洁，便于理解 LED 控制逻辑及硬件驱动接口。

使用方法

编译与下载

1. 打开工程并完成编译。
2. 使用 **板载下载器 (DAP)** 将开发板的 USB 接口连接至 PC。
3. 通过编程工具将生成的固件烧录至开发板。

运行效果

- 烧录完成后，开发板上电即可运行示例工程。
- **绿色 LED 灯每 500ms 闪烁一次**，表示系统 GPIO 控制和调度正常。
- 用户可根据需求修改闪烁周期或 LED 控制逻辑。

注意事项

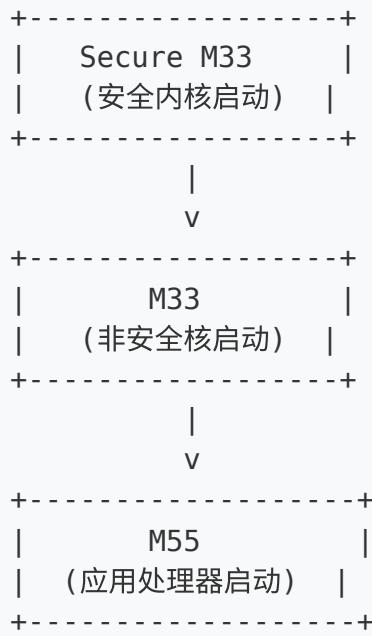
- 如需修改工程的 **图形化配置**，请使用以下工具打开配置文件：

```
tools/device-configurator/device-configurator.exe  
libs/TARGET_APP_KIT_PSE84_EVAL_EPC2/config/design.modus
```

- 修改完成后保存配置，并重新生成代码。

启动流程

系统启动顺序如下：



⚠ 请严格按照以上顺序烧写固件，否则系统可能无法正常运行。

- 若示例工程无法正常运行，建议先编译并烧录 **Edgi-Talk_M33_S_Template** 工程，确保初始化与核心启动流程正常，再运行本示例。
- 若要开启 M55，需要在 **M33 工程** 中打开配置：

```
RT-Thread Settings --> 硬件 --> select SOC Multi Core Mode -->
```

1.3. Edgi-Talk_M33_S_Template 示例工程

中文 | English

简介

本示例工程基于 **裸机 (Bare Metal)** 架构，主要用于演示和配置 **Secure M33 内核** 的相关功能。

同时，该工程也可作为二次开发或项目创作的基础模板，帮助用户快速上手并进行功能扩展。

软件说明

- 工程基于 **Edgi-Talk** 平台开发。
- 示例涵盖以下内容：
 - 安全区域配置**
 - 基本启动流程演示**
- 工程代码结构简洁清晰，便于理解和移植。

使用方法

编译与下载

- 打开工程并完成编译。
- 使用 **板载下载器 (DAP)** 将开发板的 USB 接口连接至 PC。
- 通过编程工具将编译生成的固件烧录至开发板。

运行效果

- 烧录完成后，开发板上电即可运行示例工程。
- 系统将正常启动，并顺利跳转至 **M33 内核**，表明安全配置已生效。

注意事项

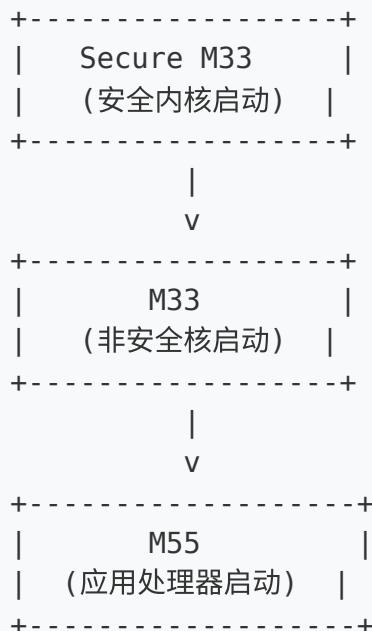
- 如需修改工程的 **图形化配置**，请使用以下工具打开配置文件：

```
tools/device-configurator/device-configurator.exe  
libs/TARGET_APP_KIT_PSE84_EVAL_EPC2/config/design.modus
```

- 修改完成后保存配置，并重新生成代码。

启动流程

系统启动顺序如下：



⚠ 请严格按照以上顺序烧写固件，否则系统可能无法正常运行。

1.4. Edgi-Talk_M33_Template 示例工程

中文 | English

简介

本示例工程基于 **RT-Thread** 实时操作系统，运行于 **M33 内核**。
通过本工程，用户可以快速体验 RT-Thread 在 M33 平台上的运行效果。
在固件成功烧录并启动后，开发板上的 **蓝色指示灯** 将周期性闪烁，表示系统已正常运行。
同时，该工程也可作为二次开发或项目创作的基础模板，帮助用户快速上手并进行功能扩展。

软件说明

- 工程基于 **Edgi-Talk** 平台开发。
- 使用 **RT-Thread** 作为操作系统内核。
- 示例功能：
- 系统初始化
- LED 指示灯任务（闪烁）
- 工程结构清晰，适合作为学习 RT-Thread 或开发应用的起点。

使用方法

编译与下载

1. 打开工程并完成编译。
 2. 使用 **板载下载器 (DAP)** 将开发板的 USB 接口连接至 PC。
 3. 通过编程工具将生成的固件烧录至开发板。
- 工程在烧录过程中会自动调用以下工具，将签名固件合并后完成烧录：

```
tools/edgeprotecttools/bin/edgeprotecttools.exe
```

- 默认会将目录下的 `proj_cm33_s_signed.hex` 一并合并烧录到目标设备。

运行效果

- 烧录完成后，开发板上电即运行示例工程。
- 蓝色指示灯** 会周期性闪烁，表明 RT-Thread 系统调度已成功启动。

注意事项

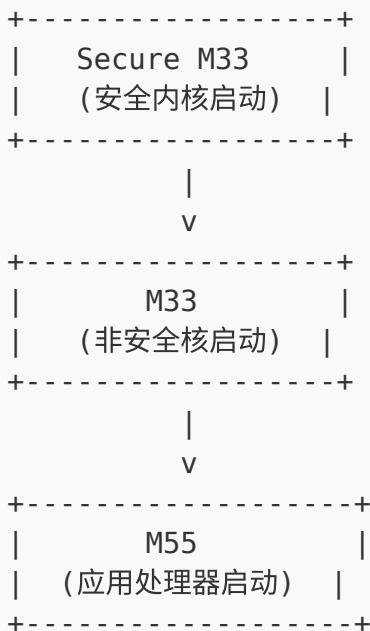
- 如需修改工程的 **图形化配置**，请使用以下工具打开配置文件：

```
tools/device-configurator/device-configurator.exe  
libs/TARGET_APP_KIT_PSE84_EVAL_EPC2/config/design.modus
```

- 修改完成后保存配置，并重新生成代码。

启动流程

系统启动顺序如下：



⚠ 请严格按照以上顺序烧写固件，否则系统可能无法正常运行。

- 若要开启M55需要在RT-Thread Settings --> 硬件 --> select SOC Multi Core Mode --> Enable CM55 Core 打开配置

2. 驱动篇

2.1. Edgi-Talk_ADC 示例工程

中文 | English

简介

本示例工程基于 **Edgi-Talk 平台**，运行于 **RT-Thread 实时操作系统**，用于演示 **ADC（模数转换器）** 的使用方法。

通过本工程，用户可以快速体验 ADC 数据采集与处理功能，为后续开发模拟信号采集应用提供参考。

运行时，蓝色指示灯周期性闪烁，表示系统已正常启动并运行。

1. ADC概述

ADC (Analog-to-Digital Converter) 是将连续的模拟信号转换为离散的数字信号的器件或模块，是现代数字控制系统、信号处理和测量系统中的核心部件。

- **功能：**把电压、电流等连续信号转换为数字值，以便微控制器（MCU）、DSP 或 FPGA 进行处理。
- **重要指标：**
- **分辨率 (Resolution)：**ADC输出的数字位数，表示可区分的电平数。Edgi 为 **12位**，即 $2^{12} = 4096$ 个不同电平。
- **采样率 (Sampling Rate)：**ADC每秒采样次数，影响可捕捉的信号频率范围。
- **输入范围 (Input Range)：**ADC能处理的模拟电压范围。
- **精度 (Accuracy)：**表示ADC输出与实际输入信号的接近程度，受噪声、非线性、偏移误差影响。

2. ADC工作原理

ADC通常分为几个阶段：

1. 采样与保持 (Sample & Hold, S/H)

- 将连续变化的模拟信号在采样瞬间固定，保证后续转换过程中信号不变。

2. 量化 (Quantization)

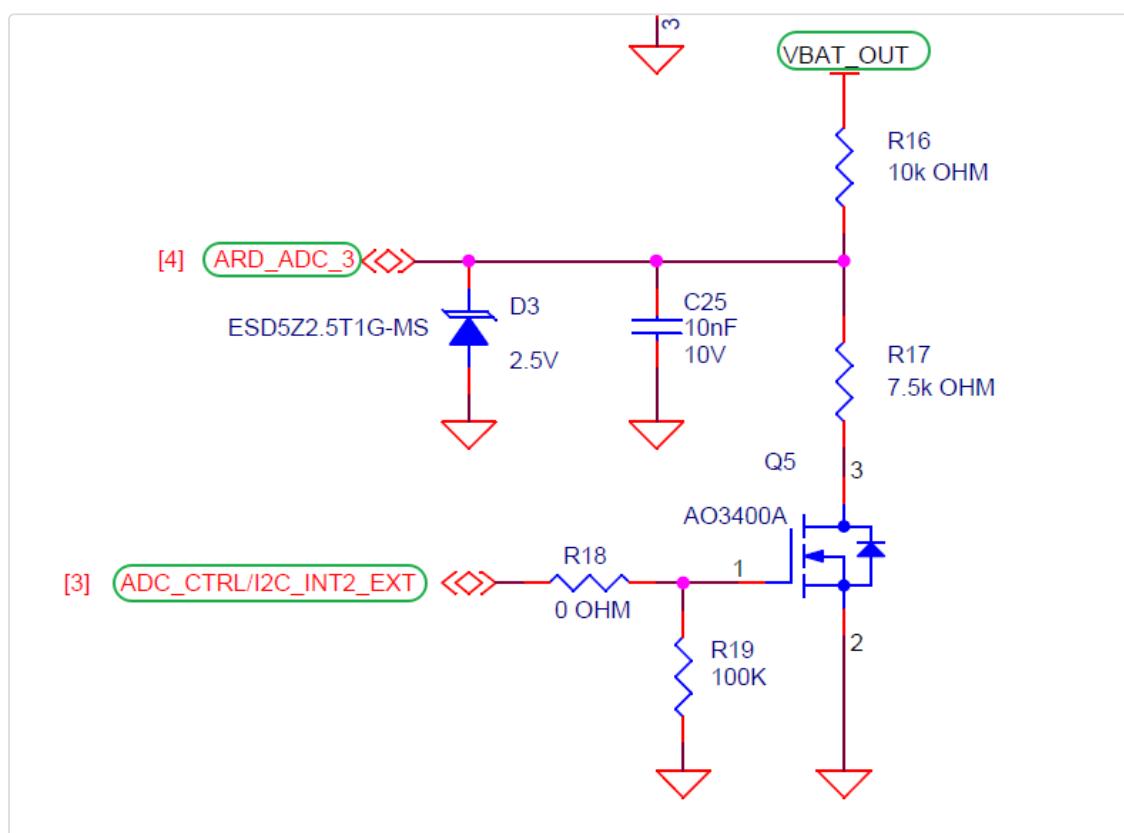
- 将模拟信号划分为离散电平，每个电平对应一个数字编码。
- 12位ADC将输入电压范围分为 4096 个电平，量化精度可以表示为： $\Delta V = V_{REF} / 4096$ 。

3. 编码 (Encoding)

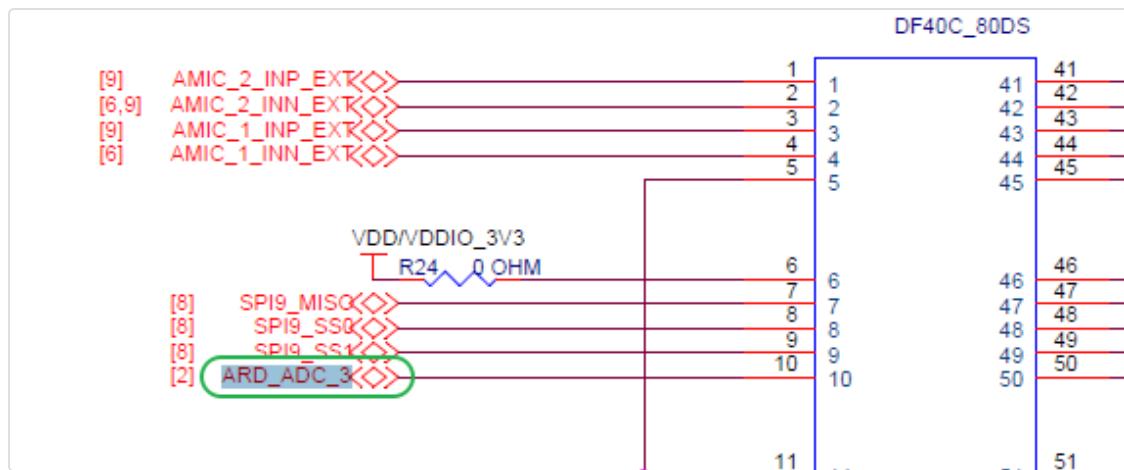
- 将量化后的电平转换成二进制代码输出。

硬件说明

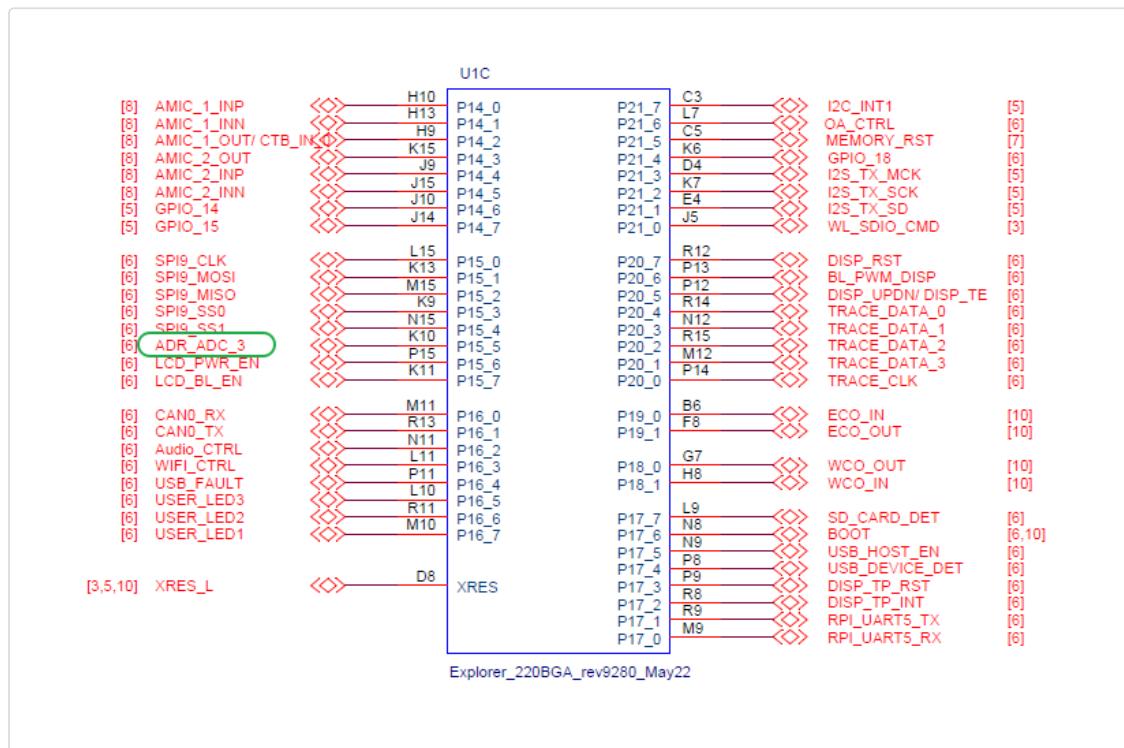
连接接口



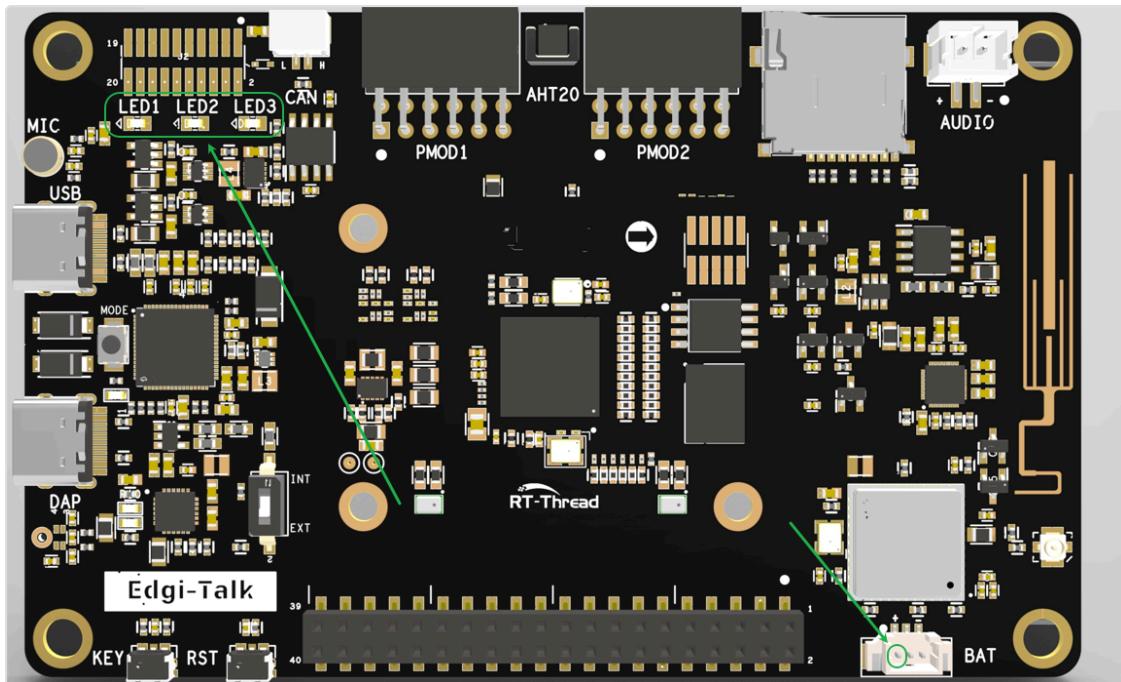
BTB座子



MCU引脚



实物图位置



软件说明

- 工程基于 **Edgi-Talk** 平台开发。
- 使用 **RT-Thread** 作为操作系统内核。
- 示例功能：
 - ADC 初始化与采样
 - LED 指示灯闪烁
 - ADC 采样结果通过串口打印
- 工程结构清晰，便于用户理解 ADC 驱动与 RT-Thread 线程机制。

使用方法

编译与下载

- 打开工程并完成编译。
- 使用 **板载下载器 (DAP)** 将开发板的 USB 接口连接至 PC。
- 通过编程工具将生成的固件烧录至开发板。

运行效果

- 烧录完成后，开发板上电即可运行示例工程。
- 蓝色指示灯 每 500ms 闪烁一次，表示系统调度正常。
- ADC 采样电池电压数据会通过串口打印，示例输出如下：

```
Value is: 3.123 V  
Value is: 3.125 V  
...
```

注意事项

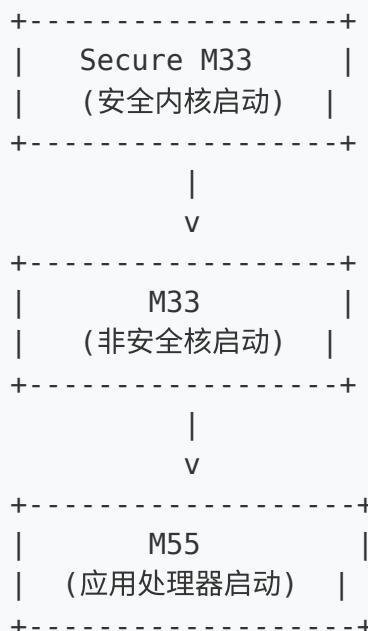
- 如需修改工程的 **图形化配置**，请使用以下工具打开配置文件：

```
tools/device-configurator/device-configurator.exe  
libs/TARGET_APP_KIT_PSE84_EVAL_EPC2/config/design.modus
```

- 修改完成后保存配置，并重新生成代码。

启动流程

系统启动顺序如下：



⚠ 请严格按照以上顺序烧写固件，否则系统可能无法正常运行。

- 若示例工程无法正常运行，建议先编译并烧录 **Edgi-Talk_M33_S_Template** 工程，确保初始化与核心启动流程正常，再运行本示例。
- 若要开启 M55，需要在 **M33 工程** 中打开配置：

```
RT-Thread Settings --> 硬件 --> select SOC Multi Core Mode -->
```



2.2. Edgi-Talk_M33_AHT20 示例工程

中文 | English

简介

本示例工程基于 **Edgi-Talk** 平台，演示 **AHT20** 温湿度传感器 的驱动和使用方法。

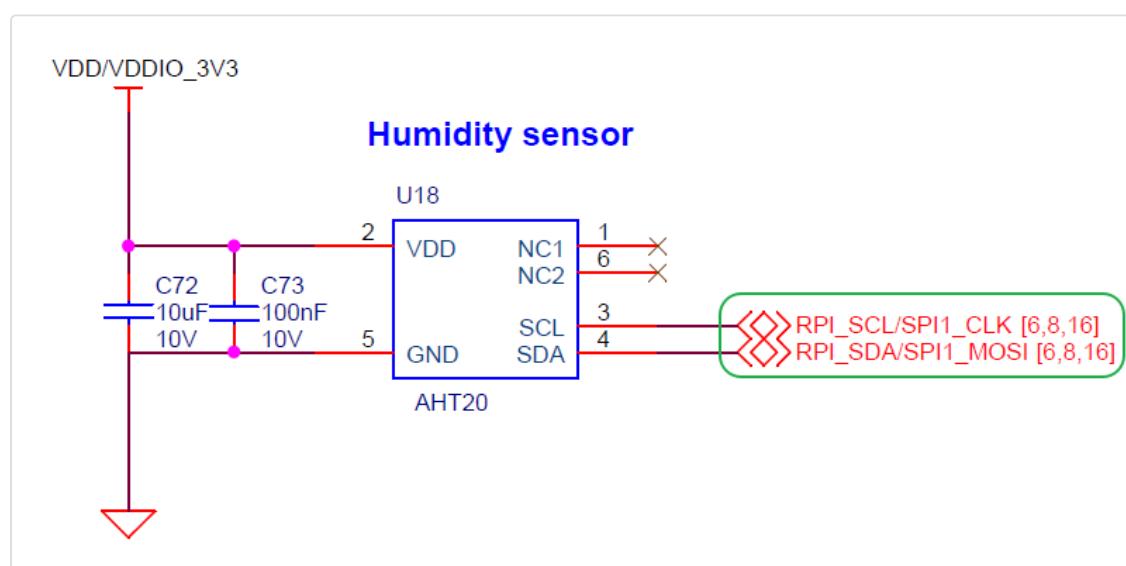
通过本工程，用户可以快速体验 AHT20 的数据采集与处理，并在开发板上通过串口查看采样结果。

AHT10 软件包简介

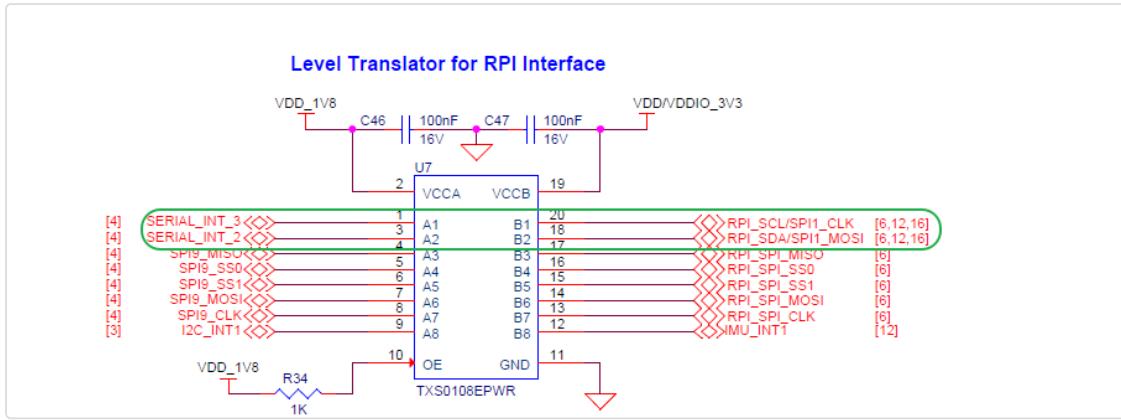
AHT10 软件包提供了使用温度与湿度传感器 aht10 基本功能，并且提供了软件平均数滤波器可选功能，如需详细了解该软件包，请参考 AHT10 软件包中的 README。

硬件说明

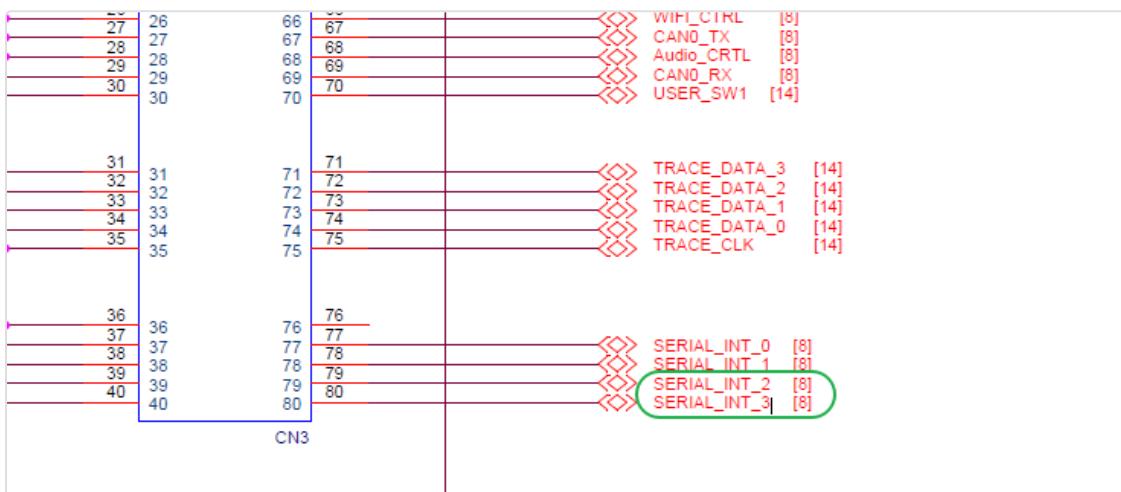
传感器连接接口



电平转换



BTB座子



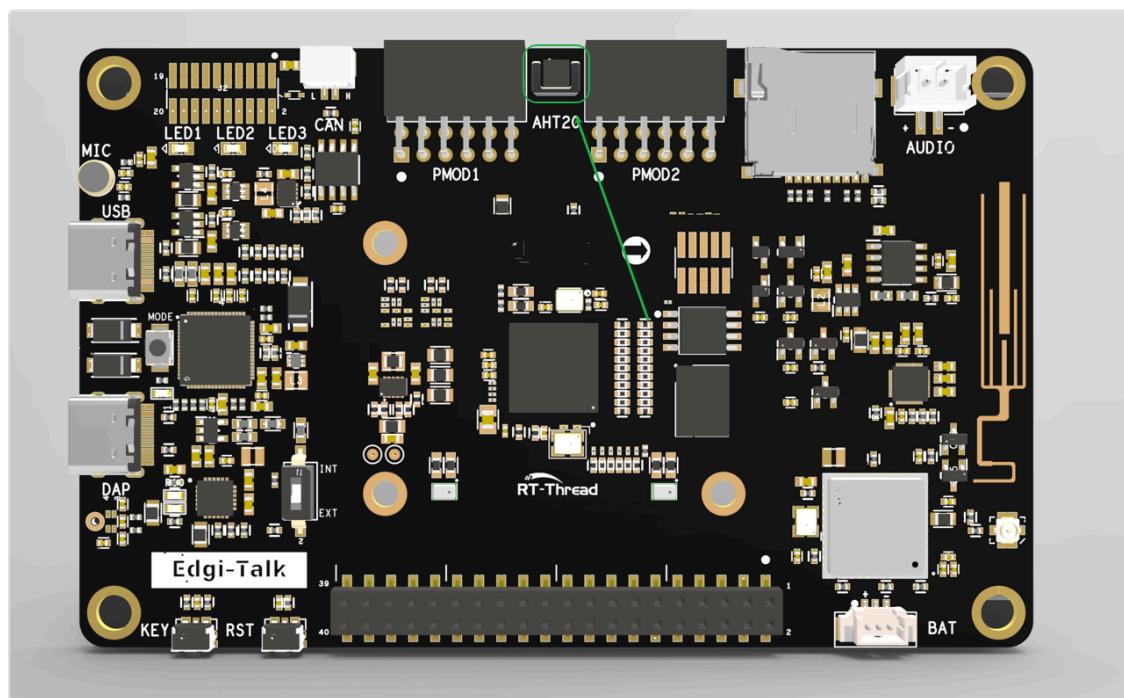
MCU引脚

M5	P7_3	P11_1	E11	GPIO_11	[5]
P4	P7_4	P11_0	E14	BT_REG_ON	[5]
L4	P7_5		C14	GPIO_12	[5]
N4	P7_6	P10_7	D15	BT_DEV_WAKE	[3,5]
L5	P7_7	P10_6	B14	GPIO_13	[5]
H7	P8_0	P10_5	D14	BT_HOST_WAKE	[3,5]
J8	P8_1	P10_4	A14	BT_UART_RTS	[5]
J7	P8_2	P10_3	C15	BT_UART_CTS	[5]
K8	P8_3	P10_2	A13	BT_UART_TXD	[5]
F6	P8_4	P10_1	B15	BT_UART_RXD	[5]
E7	P8_5	P10_0			[3,5]
G6	P8_6	P9_3	N13	SERIAL_INT_3	[6]
H6	P8_7	P9_2	M14	SERIAL_INT_2	[6]
		P9_1	M13	SERIAL_INT_1	[6]
		P9_0	L14	SERIAL_INT_0	[6]

Explorer_220BGA_rev9280_May22

DD/VDDIO_1V8

实物图位置



软件说明

- 工程基于 **Edgi-Talk** 平台开发。
- 示例功能包括：

- AHT20 初始化与通信 (I^2C)
- 温湿度数据读取与解析
- 串口打印采样数据
- 工程结构清晰，便于用户理解 I^2C 驱动与传感器接口。

使用方法

编译与下载

1. 打开工程并完成编译。
2. 使用 **板载下载器 (DAP)** 将开发板的 USB 接口连接至 PC。
3. 通过编程工具将生成的固件烧录至开发板。

运行效果

- 烧录完成后，开发板上电即可运行示例工程。
- 系统会初始化 AHT20 并开始采样温湿度数据。
- 采样结果通过串口打印，示例输出如下：

```
\ | /
- RT -      Thread Operating System
/ | \      5.0.2 build Sep  5 2025 14:13:02
2006 - 2022 Copyright by RT-Thread team
Hello RT-Thread
This core is cortex-m33
msh >[I/aht10] AHT10 has been initialized!
[D/aht10] Humidity    : 44.4 %
[D/aht10] Temperature: 29.7
[D/aht10] Humidity    : 44.4 %
[D/aht10] Temperature: 29.7
```

注意事项

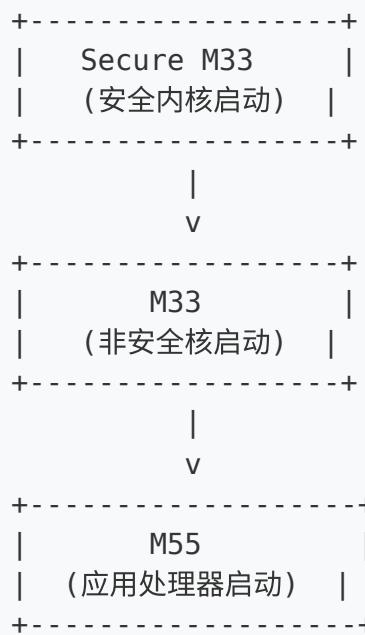
- 如需修改工程的 **图形化配置**，请使用以下工具打开配置文件：

```
tools/device-configurator/device-configurator.exe
libs/TARGET_APP_KIT_PSE84_EVAL_EPC2/config/design.modus
```

- 修改完成后保存配置，并重新生成代码。

启动流程

系统启动顺序如下：



⚠ 请严格按照以上顺序烧写固件，否则系统可能无法正常运行。

-
- 若示例工程无法正常运行，建议先编译并烧录 **Edgi-Talk_M33_S_Template** 工程，确保初始化与核心启动流程正常，再运行本示例。
 - 若要开启 M55，需要在 **M33 工程** 中打开配置：

```
RT-Thread Settings --> 硬件 --> select SOC Multi Core Mode -->
```

2.3. Edgi-Talk_Audio 示例工程

中文 | English

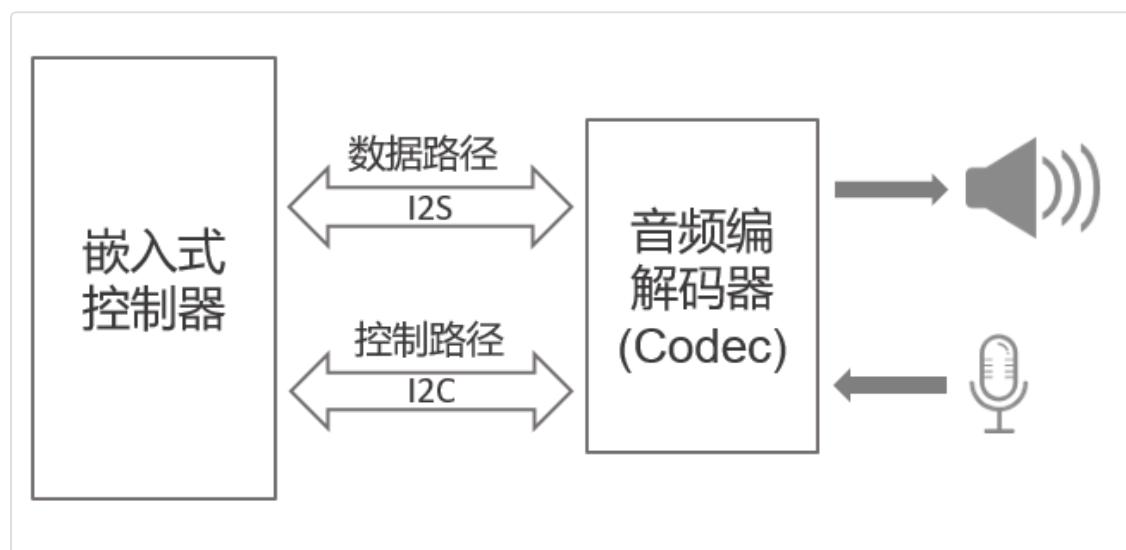
简介

本示例工程基于 **Edgi-Talk** 平台，演示 **音频录制与播放** 功能，运行在 **RT-Thread** 实时操作系统上。

通过本工程，用户可以体验麦克风采集音频数据并通过扬声器播放，同时通过按钮控制播放/停止状态，并通过 LED 指示当前播放状态。

Audio 简介

Audio（音频）设备是嵌入式系统中非常重要的一个组成部分，负责音频数据的采样和输出。Audio 设备通常由数据总线接口、控制总线接口、音频编解码器（Codec）、扬声器和麦克风等组成，如下图所示：



Audio 设备特性

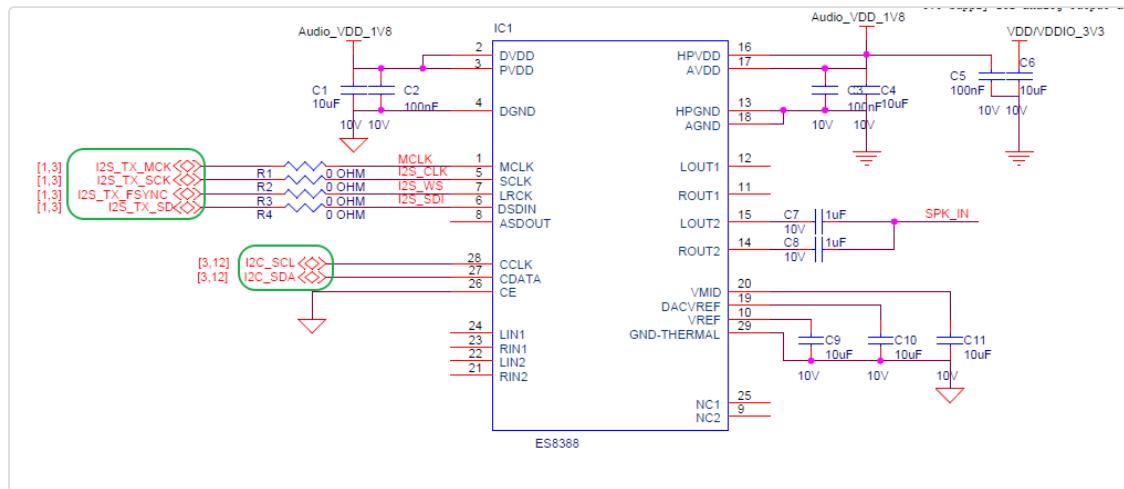
RT-Thread Audio 设备驱动框架是 Audio 框架的底层部分，主要负责原生音频数据的采集和输出、音频流的控制、音频设备的管理、音量调节以及不同硬件和 Codec 的抽象等。

- 接口：标准 device 接口(open/close/read/control)。
- 同步模式访问。
- 支持播放和录音。

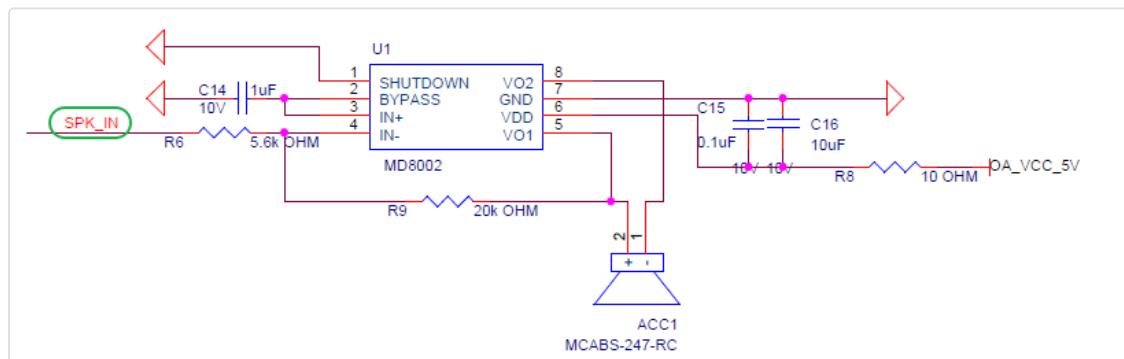
- 支持音频参数管理。
- 支持音量调节。

硬件说明

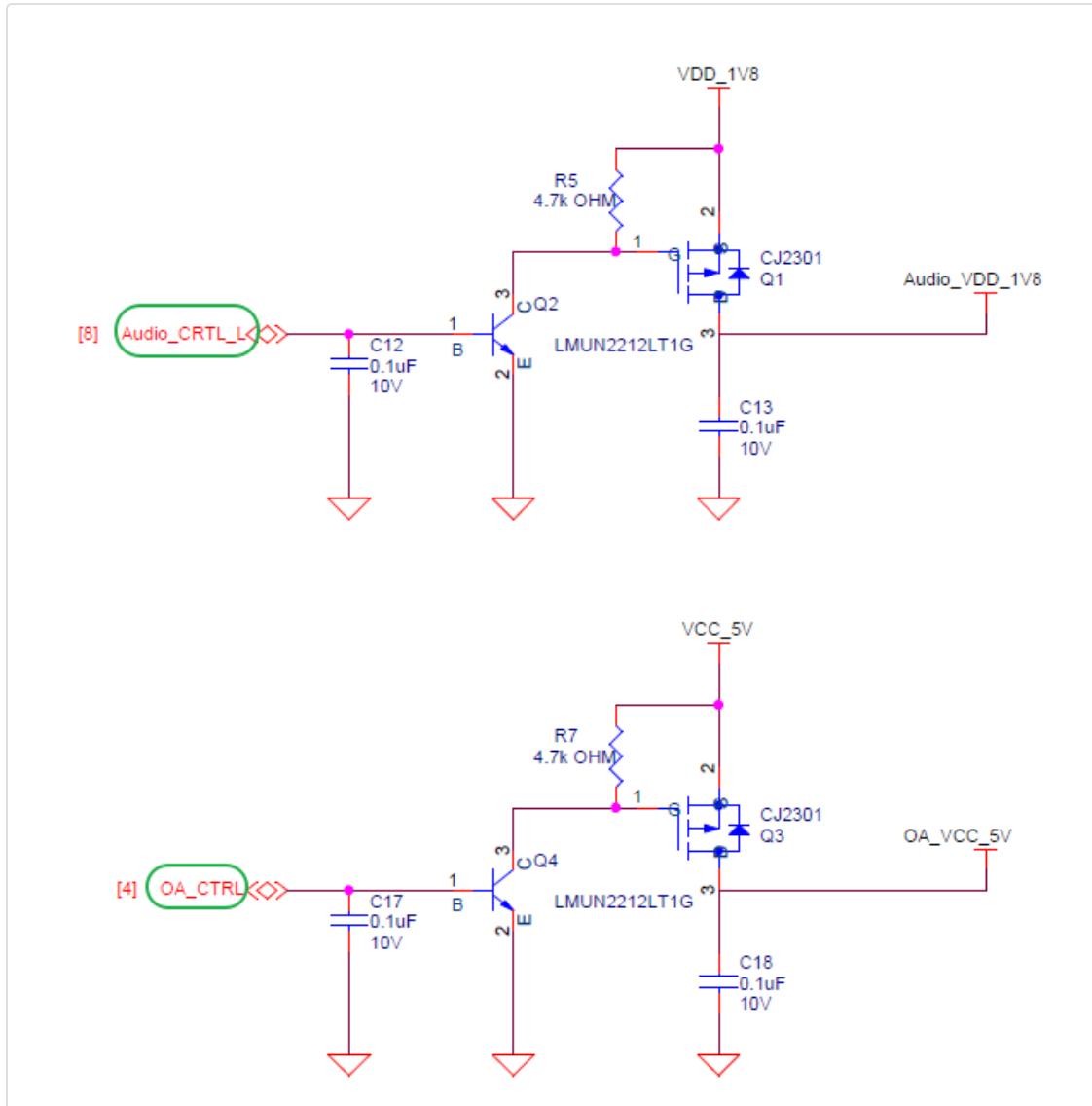
ES8388连接接口



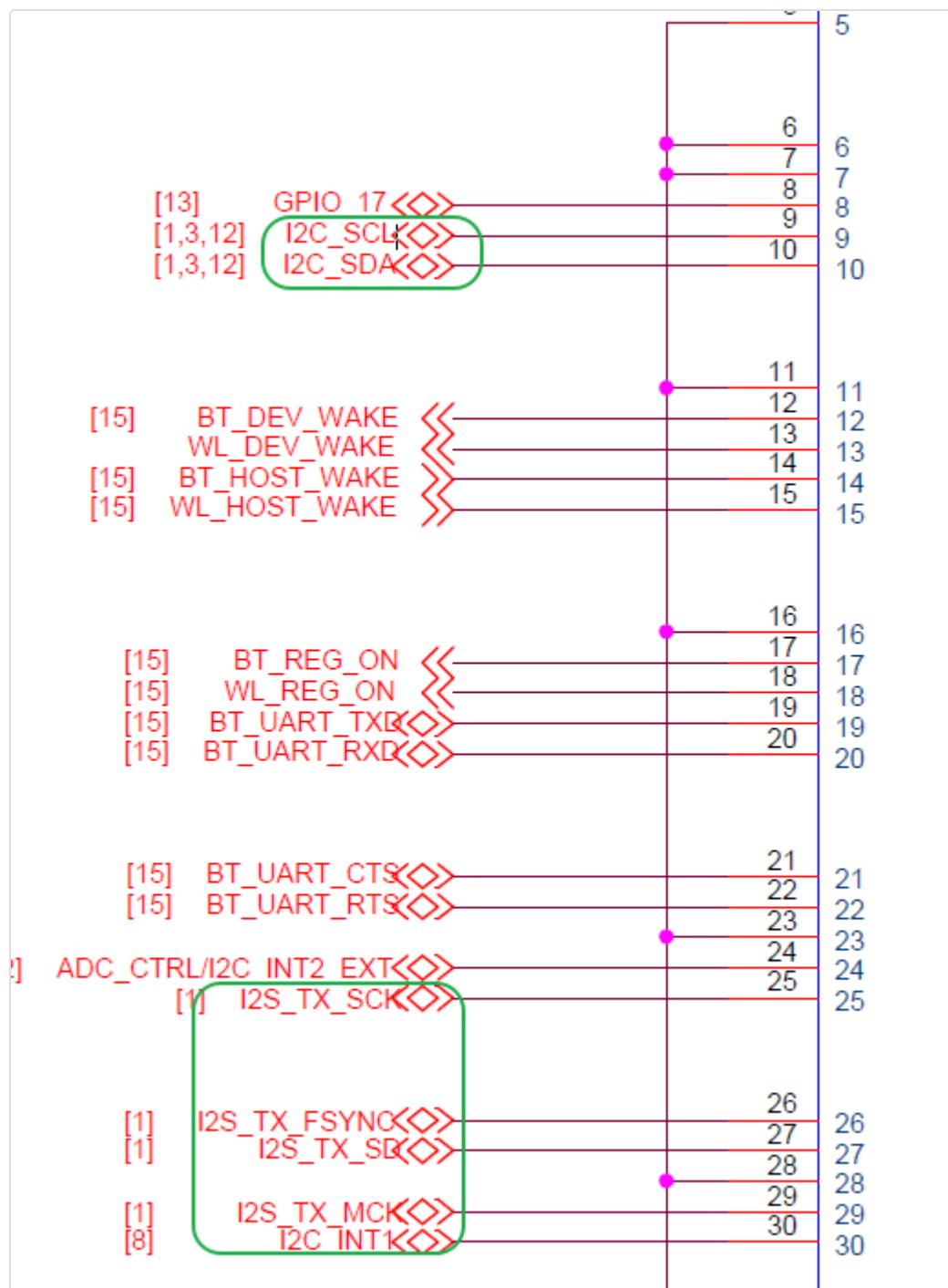
喇叭接口



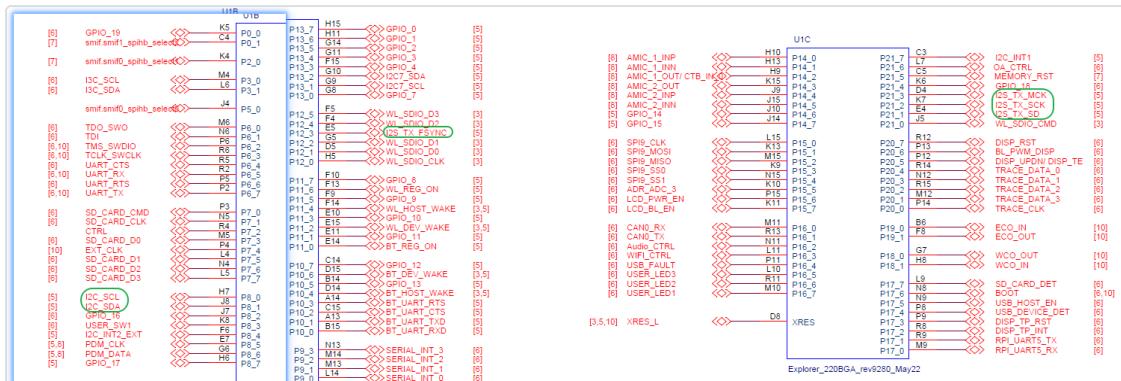
控制引脚



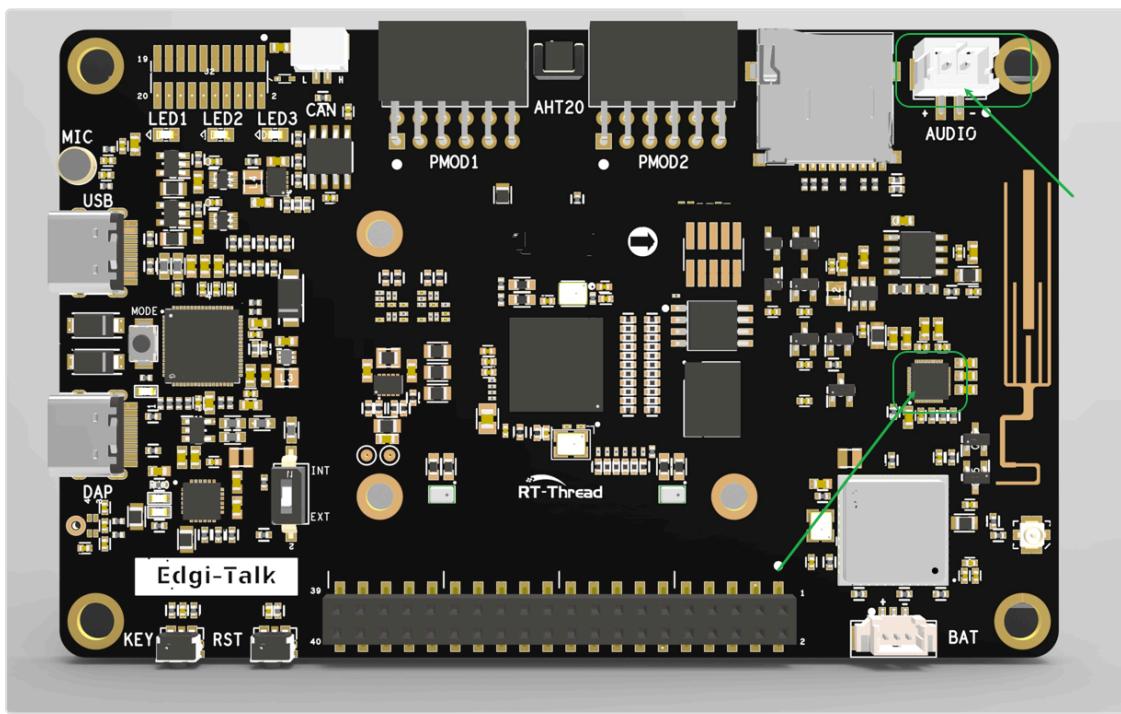
BTB座子



MCU接口



实物图位置



软件说明

- 工程基于 Edgi-Talk M33 平台开发。
- 示例功能包括：
 - 麦克风音频采集
 - 音频数据播放到扬声器
 - 按钮控制音频播放/停止

- LED 指示播放状态
- 工程结构清晰，便于理解 RT-Thread 音频设备驱动和事件处理机制。

使用方法

编译与下载

1. 打开工程并完成编译。
 2. 使用 **板载下载器 (DAP)** 将开发板的 USB 接口连接至 PC。
 3. 通过编程工具将生成的固件烧录至开发板。
- 工程可自动调用签名工具（如 `tools/edgeprotecttools/bin/edgeprotecttools.exe`）合并烧录固件（如 `proj_cm33_s_signed.hex`）。

运行效果

- 烧录完成后，开发板上电即可运行示例工程。
- LED 灯默认点亮表示音频播放已启用。
- 按下按钮可 **切换音频播放状态**：
 - **播放**：LED 点亮，麦克风采集的音频播放到扬声器
 - **停止**：LED 熄灭，音频播放暂停
- 系统可连续采集和播放音频数据，实现实时音频回放。

注意事项

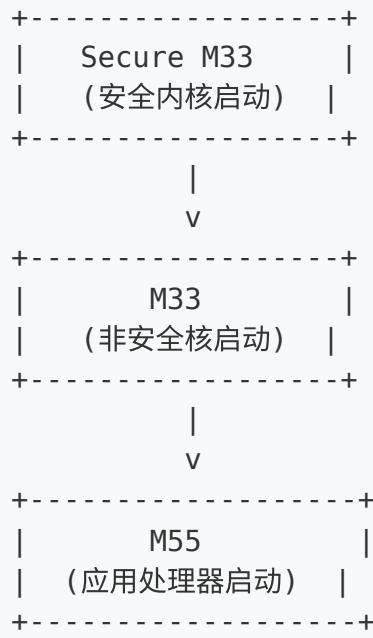
- 如需修改工程的 **图形化配置**，请使用以下工具打开配置文件：

```
tools/device-configurator/device-configurator.exe  
libs/TARGET_APP_KIT_PSE84_EVAL_EPC2/config/design.modus
```

- 修改完成后保存配置，并重新生成代码。

启动流程

系统启动顺序如下：



⚠ 请严格按照以上顺序烧写固件，否则系统可能无法正常运行。

- 若示例工程无法正常运行，建议先编译并烧录 **Edgi-Talk_M33_S_Template** 工程，确保初始化与核心启动流程正常，再运行本示例。
- 若要开启 M55，需要在 **M33 工程** 中打开配置：

```
RT-Thread Settings --> 硬件 --> select SOC Multi Core Mode -->
```

2.4. Edgi-Talk_emUSB-device_CDC_Echo 示例工程

中文 | English

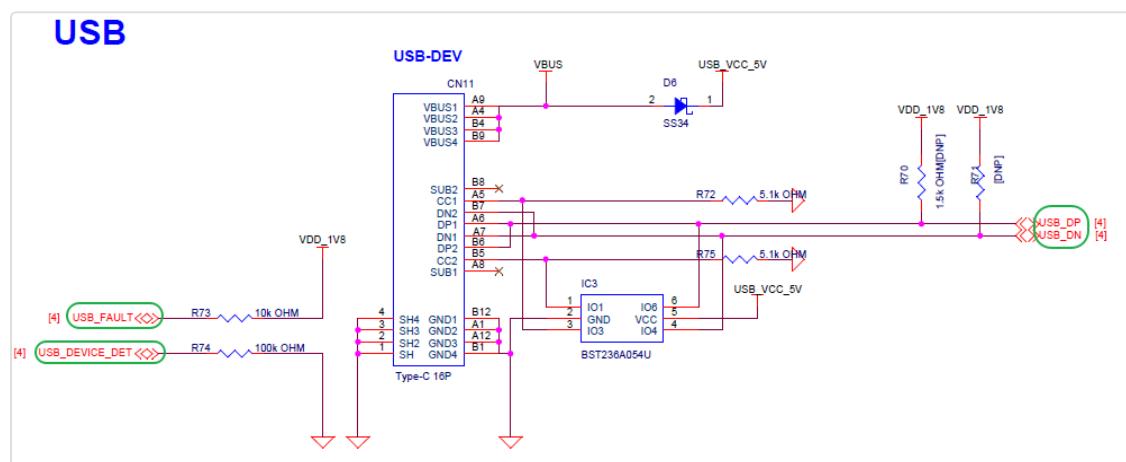
简介

本示例工程基于 Edgi-Talk 平台，演示 USB CDC（虚拟串口）回显功能，运行在 RT-Thread 实时操作系统（M33核）上。

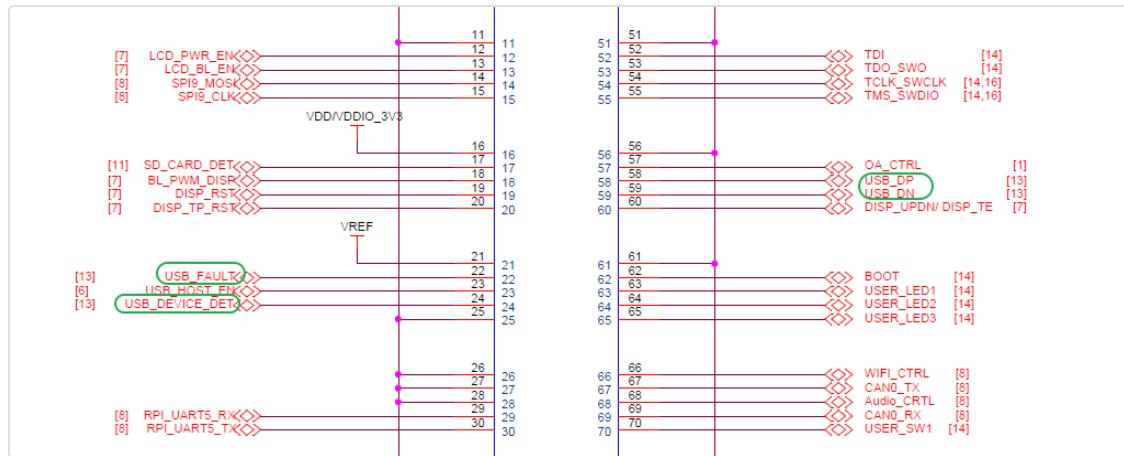
通过本工程，用户可以快速体验 USB CDC 设备通信机制，并验证数据回显功能，为后续 USB 通信和多核应用开发提供参考。

硬件说明

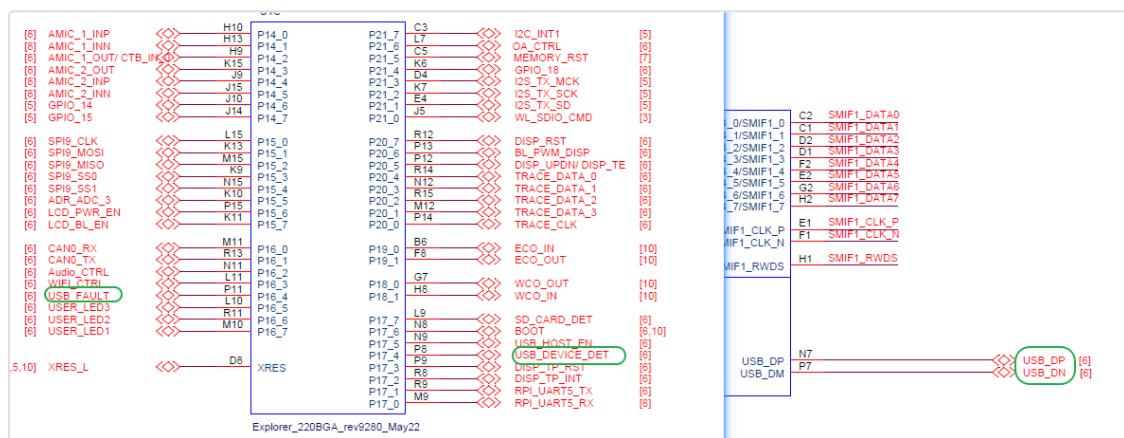
USB接口



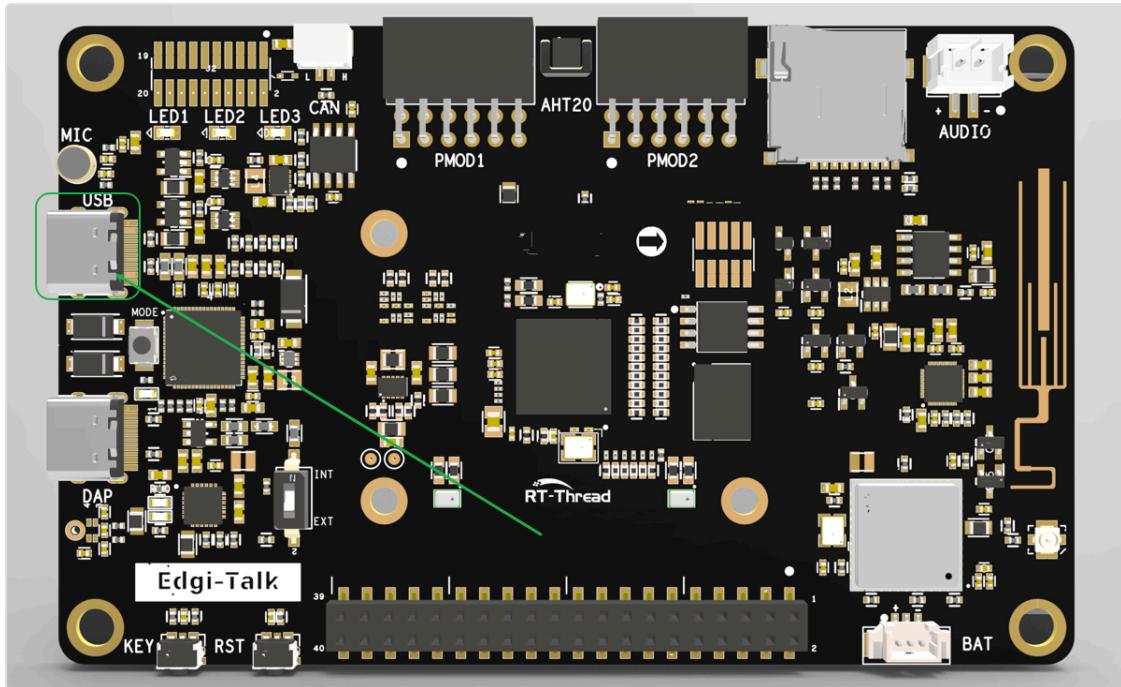
BTB座子



MCU接口



实物图位置



软件说明

- 工程基于 **Edgi-Talk** 平台开发。
- 示例功能包括：
- USB CDC 设备初始化
- 虚拟串口数据回显
- 工程结构清晰，便于理解 USB 设备驱动在 M33 核上的运行方式。

使用方法

编译与下载

- 打开工程并完成编译。
- 使用 **板载下载器 (DAP)** 将开发板的 USB 接口连接至 PC。
- 通过编程工具将生成的固件烧录至开发板。

运行效果

- 烧录完成后，开发板上电即可运行示例工程。
- 用户需在串口命令行手动输入：

```
cdc_sample
```

- 系统输出如下启动信息：

```
\ | /  
- RT -      Thread Operating System  
/ | \      5.0.2 build Sep  8 2025 09:57:30  
2006 - 2022 Copyright by RT-Thread team  
Hello RT-Thread  
This core is cortex-m33  
msh >cdc_sample  
***** PSOC Edge MCU: CDC echo using emUSB-device**
```

- 在 PC 上使用任意串口工具连接开发板的 USB 虚拟串口。
1. 打开 PC 串口调试工具，连接开发板的虚拟串口（波特率任意）。
 2. 在串口工具中输入一段字符串，并以换行符 `\n` 结尾发送。
 3. 开发板会将接收到的完整字符串回显至串口终端。

```
> hello  
hello  
> 12345  
12345
```

注意事项

- 回显触发条件：

只有当接收数据的最后一个字符为 换行符 `\n` 时，才会触发一次完整回显。

如果输入未包含 `\n`，数据会暂存于缓冲区，不会立即回显。

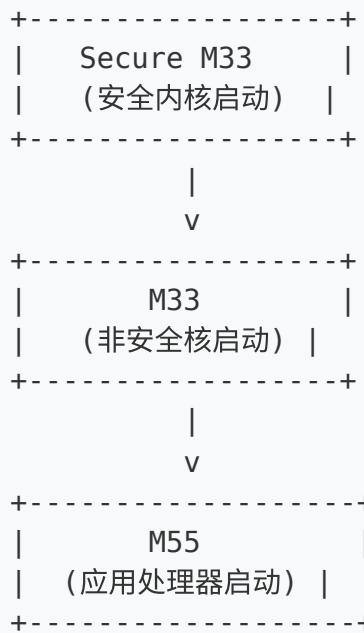
- 如需修改工程的 图形化配置，请使用以下工具打开配置文件：

```
tools/device-configurator/device-configurator.exe  
libs/TARGET_APP_KIT_PSE84_EVAL_EPC2/config/design.modus
```

- 修改完成后保存配置，并重新生成代码。

启动流程

系统启动顺序如下：



⚠ 请严格按照以上顺序烧写固件，否则系统可能无法正常运行。

-
- 若示例工程无法正常运行，建议先编译并烧录 **Edgi-Talk_M33_S_Template** 工程，确保初始化与核心启动流程正常，再运行本示例。
 - 若要开启 M55，需要在 **M33 工程** 中打开配置：

```
RT-Thread Settings --> 硬件 --> select SOC Multi Core Mode -->
```

2.5. Edgi-Talk_M33_HyperRam 示例工程

[中文](#) | [English](#)

简介

本示例工程基于 **Edgi-Talk** 平台，演示 **HyperRam** 功能，运行在 **RT-Thread** 实时操作系统 上。

通过本工程，用户可以快速验证HyperRam，为后续硬件控制和应用开发提供基础参考。

软件说明

- 工程基于 **Edgi-Talk** 平台开发。
- 示例功能包括：
- 工程结构简洁，便于理解 HyperRam 控制逻辑及硬件驱动接口。

```
#include "rtthread.h"

#define DRV_DEBUG
#define LOG_TAG          "drv_hyperam"
#include <drv_log.h>

#define PSRAM_ADDRESS           (0x64800000)

#ifdef BSP_USING_HYPERAM
#ifdef RT_USING_MEMHEAP_AS_HEAP
    struct rt_memheap system_heap;
#endif
#endif

static int hyperam_init(void)
{
    LOG_D("hyperam init success, mapped at 0x%X, size is %d byt
#ifdef RT_USING_MEMHEAP_AS_HEAP
    /* If RT_USING_MEMHEAP_AS_HEAP is enabled, HYPERAM is initi
        rt_memheap_init(&system_heap, "hyperam", (void *)PSRAM_ADDR
#endif
    return RT_EOK;
}
INIT_BOARD_EXPORT(hyperam_init);
#endif
```

使用方法

编译与下载

1. 打开工程并完成编译。
2. 使用 **板载下载器 (DAP)** 将开发板的 USB 接口连接至 PC。
3. 通过编程工具将生成的固件烧录至开发板。

运行效果

- 烧录完成后，开发板上电即可运行示例工程。

注意事项

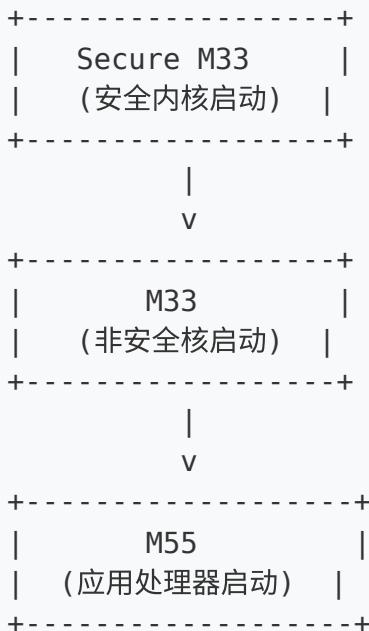
- 如需修改工程的 **图形化配置**，请使用以下工具打开配置文件：

```
tools/device-configurator/device-configurator.exe  
libs/TARGET_APP_KIT_PSE84_EVAL_EPC2/config/design.modus
```

- 修改完成后保存配置，并重新生成代码。

启动流程

系统启动顺序如下：



 请严格按照以上顺序烧写固件，否则系统可能无法正常运行。

- 若示例工程无法正常运行，建议先编译并烧录 **Edgi-Talk_M33_S_Template** 工程，确保初始化与核心启动流程正常，再运行本示例。
- 若要开启 M55，需要在 **M33 工程** 中打开配置：

```
RT-Thread Settings --> 硬件 --> select SOC Multi Core Mode -->
```

2.6. Edgi-Talk_M33_S_HyperRam 示例工程

中文 | English

简介

本示例工程基于 **裸机 (Bare Metal)** 架构，主要用于演示和配置 **Secure M33 HyperRam** 的相关功能。

同时，该工程也可作为二次开发或项目创作的基础模板，帮助用户快速上手并进行功能扩展。

HyperRAM 简介

1. 概述

HyperRAM 是一种由 Cypress (现 Infineon) 首次推出的 **高性能低引脚数 (Low Pin Count, LPC) DRAM**。

它基于 **HyperBus 接口**，主要面向 **嵌入式系统、显示控制、IoT 设备、汽车电子** 等需要 **高带宽、低功耗、简单接口** 的应用。

HyperRAM 属于 **pSRAM (Pseudo-SRAM, 伪静态RAM)**，它在外部表现得像 SRAM (简单读写，无需用户刷新)，但内部实际上是低功耗 DRAM (自刷新)。

2. 架构与接口

HyperRAM 使用 **HyperBus 接口**，其特点是：

- 引脚数少**：通常仅需 **13 个信号引脚** (8-bit 数据总线 + 控制/时钟)，相比传统 SDRAM (几十个引脚) 大幅减少 PCB 复杂度。
- 双数据速率 (DDR) 传输**：在时钟上升沿和下降沿传输数据，提高带宽。
- 串行控制协议**：通过命令-地址-数据序列访问内存，简化设计。

接口结构如下：

- 数据线 DQ[7:0]**：8 位双向数据
- RWDS (Read-Write Data Strobe)**：数据同步信号
- CLK**：时钟输入
- CS#**：片选信号

- **RESET#**: 复位
- **CKE**: 时钟使能

3. 工作原理

HyperRAM 通过 **命令+地址+数据** 的方式访问：

1. 命令阶段

- 主机发送读/写命令和目标地址。

2. 延迟阶段

- HyperRAM 准备内部存储阵列（延迟可配置）。

3. 数据传输阶段

- 以 DDR 方式在 **DQ[7:0]** 上传输数据，RWDS 提供数据同步。

内部采用 DRAM 技术，支持 **自刷新**，但对外表现为“像 SRAM 一样”——用户无需关心刷新操作。

4. 性能特性

- **数据总线宽度**: 8 位
- **工作电压**: 1.8 V 或 3.0 V 低功耗设计
- **数据速率**: 最高可达 **400 MB/s (200 MHz DDR × 8-bit)**
- **容量范围**: 32 Mb ~ 512 Mb (4 MB ~ 64 MB)
- **低功耗**: 支持深度睡眠模式，待机电流 < 10 μ A
- **简单接口**: 13 根引脚即可完成高速访问

5. HyperRAM 的优势

1. 低引脚数

- 与传统 SDRAM/PSRAM (30+ 引脚) 相比大幅减少引脚需求，节省 PCB 走线。

2. 高带宽

- DDR 接口，带宽可达 400 MB/s，足以支持 **图像缓存、显示刷新** 等应用。

3. 低功耗

- 适合电池供电设备，如 IoT、可穿戴设备。

4. 易用性

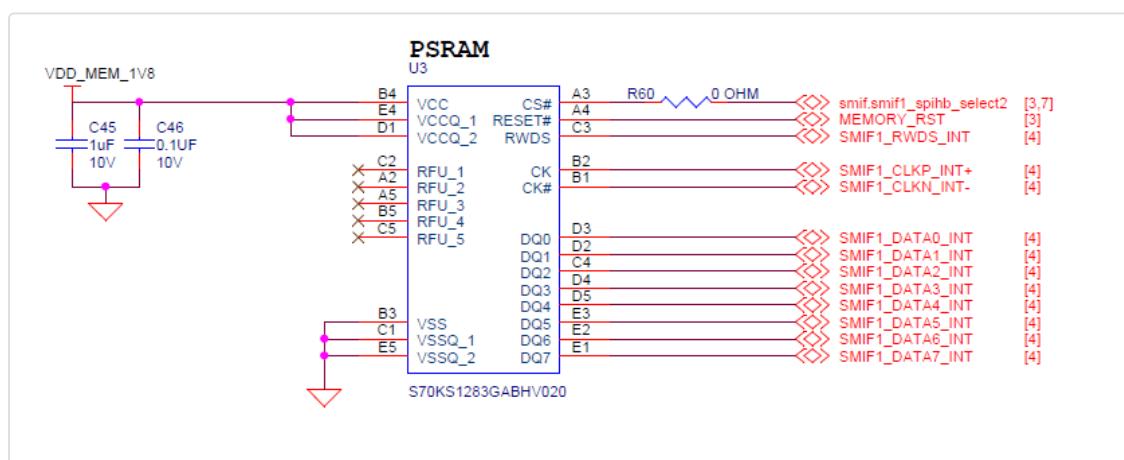
- 对外表现类似 SRAM，简单易用，无需用户刷新。

6. 与其他存储的对比

特性	HyperRAM	SDRAM / DDR	SRAM	NOR Flash
接口	HyperBus (13-pin)	并行 16~32 位	并行/串行	SPI/QSPI
容量范围	32Mb ~ 512Mb	64Mb ~ 1Gb+	小 (Kb~Mb)	4Mb ~ 2Gb
带宽	~400 MB/s	~800 MB/s+	~50 MB/s	~100 MB/s
功耗	低	较高	较低	较低
应用场景	缓存/帧缓冲	系统主存	高速小容量	程序存储

硬件说明

HyperRam接口



软件说明

- 工程基于 **Edgi-Talk** 平台开发。
- 示例涵盖以下内容：
 - 安全区域配置**
 - 基本启动流程演示**
- 工程代码结构简洁清晰，便于理解和移植。

使用方法

编译与下载

- 打开工程并完成编译。
- 使用 **板载下载器 (DAP)** 将开发板的 USB 接口连接至 PC。
- 通过编程工具将编译生成的固件烧录至开发板。

运行效果

- 烧录完成后，开发板上电即可运行示例工程。
- 系统将正常启动，并顺利跳转至 **M33 内核**，表明安全配置已生效。

注意事项

- 如需修改工程的 **图形化配置**，请使用以下工具打开配置文件：

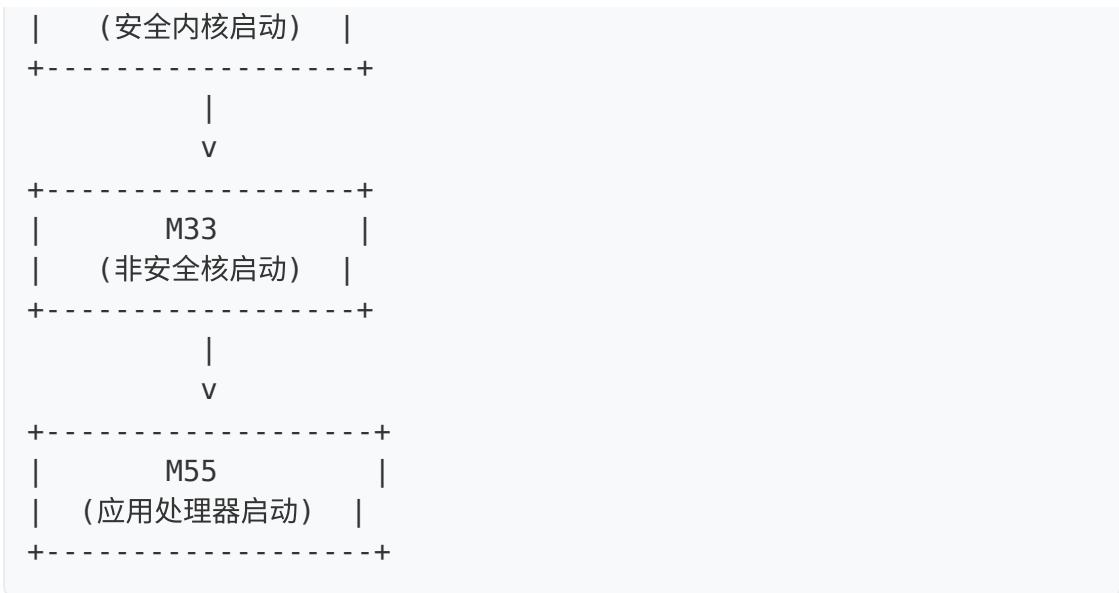
```
tools/device-configurator/device-configurator.exe  
libs/TARGET_APP_KIT_PSE84_EVAL_EPC2/config/design.modus
```

- 修改完成后保存配置，并重新生成代码。

启动流程

系统启动顺序如下：

```
+-----+  
| Secure M33 |
```



⚠️ 请严格按照以上顺序烧写固件，否则系统可能无法正常运行。

2.7. Edgi-Talk_Key_Irq 示例工程

[中文](#) | [English](#)

简介

本示例工程基于 **Edgi-Talk** 平台，运行于 **RT-Thread** 实时操作系统，演示 **按键中断 (Key IRQ)** 的使用方法。

通过本工程，用户可以学习如何在 RT-Thread 中配置 GPIO 中断，并实现按键触发事件的响应逻辑，为后续人机交互类应用提供参考。

MCU 中断体系概述

具有如下中断特性：

1. NVIC (嵌套向量中断控制器)

- 支持 **中断嵌套**：高优先级中断可打断低优先级中断
- 支持 **优先级分组** (Preemption Priority 与 Subpriority)
- IRQn 映射到中断向量表，由 NVIC 调用对应 ISR

2. GPIO 外部中断 (EXTI)

- 每个 GPIO 引脚可配置为外部中断输入
- 支持 **上升沿、下降沿或双沿触发**
- 中断通道由 **ICU (Interrupt Controller Unit)** 管理
- 可以通过 Renesas FSP 或底层寄存器配置
- 中断响应延迟低，适合按键、传感器等事件驱动

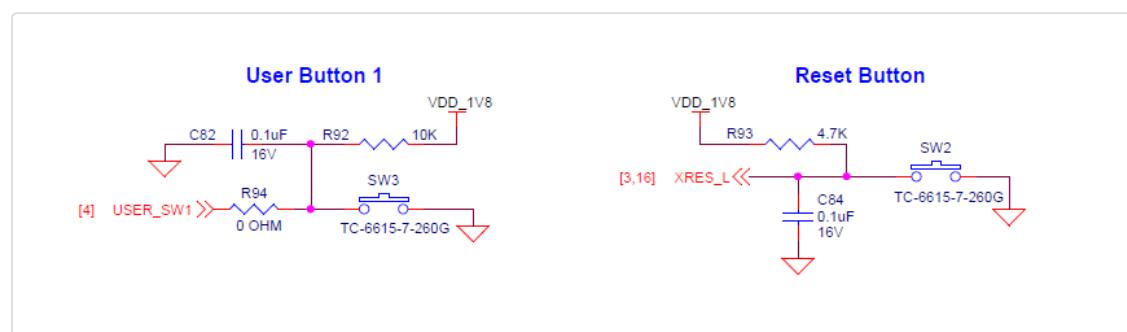
3. RT-Thread PIN 驱动模型

- GPIO 在 RT-Thread 中被封装为 **PIN 设备**
- 提供统一接口：
 - `rt_pin_mode(pin, mode)`：配置输入/输出模式
 - `rt_pin_read(pin)` / `rt_pin_write(pin, value)`：读写 GPIO
 - `rt_pin_attach_irq(pin, mode, callback, args)`：注册中断回调
 - `rt_pin_irq_enable(pin, enable)`：使能/禁用中断

使用 RT-Thread PIN 驱动，无需直接操作 NVIC 或 MCU 寄存器即可实现中断响应。

硬件说明

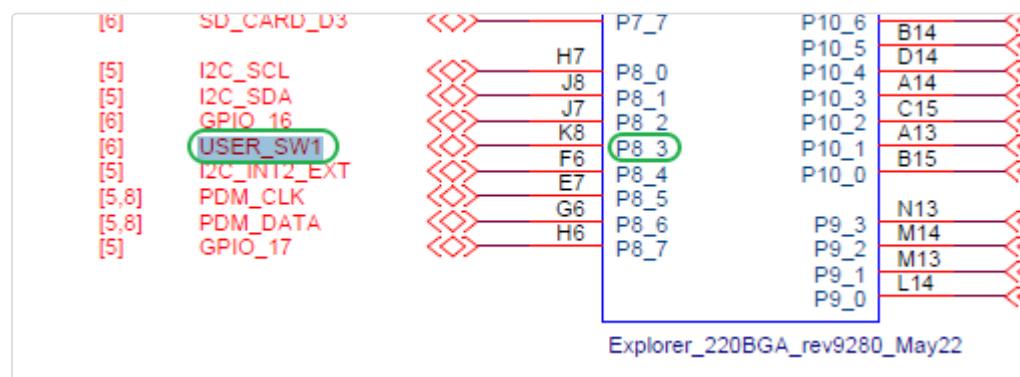
按钮接口



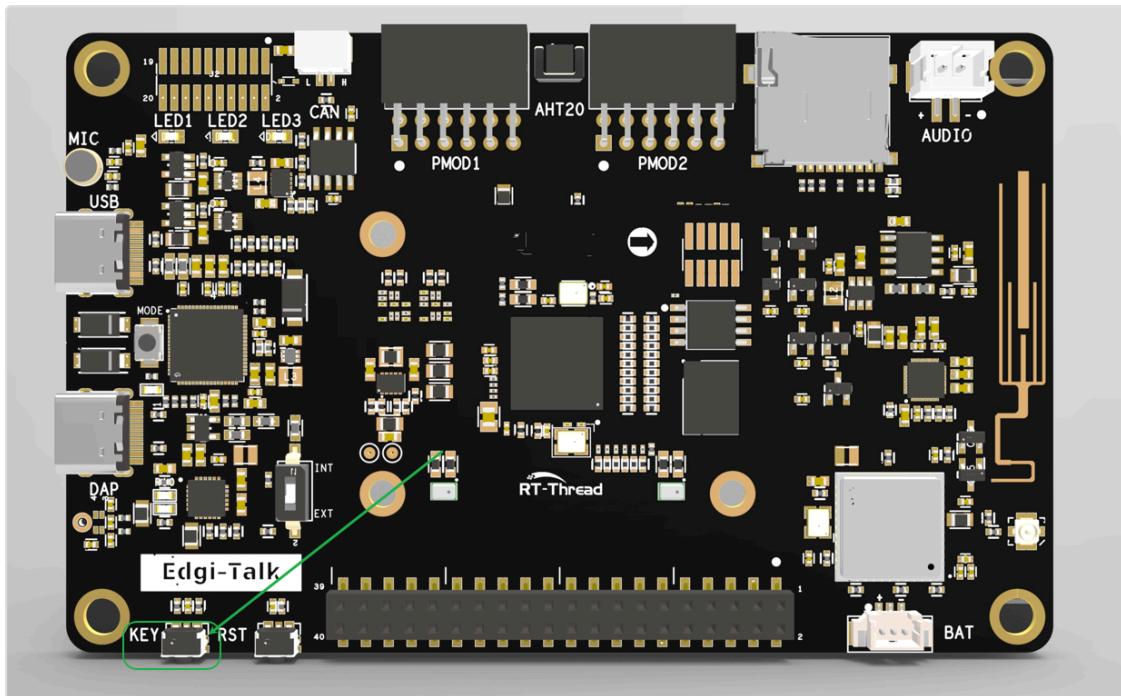
BTB座子



MCU接口



实物图位置



软件说明

- 工程基于 **Edgi-Talk** 平台开发。
- 使用 **RT-Thread** 作为操作系统内核。
- 示例功能包括：
 - 配置按键 GPIO 为中断输入模式
 - 按键按下时触发中断回调函数
 - 蓝色 LED 以 500ms 为周期闪烁，表示系统正常运行

使用方法

编译与下载

1. 打开工程并完成编译。
2. 使用 **板载下载器 (DAP)** 将开发板 USB 接口连接至 PC。
3. 通过编程工具将生成的固件烧录至开发板。

运行效果

- 烧录完成后，开发板上电即可运行示例工程。
- 蓝色 LED 将以 500ms 为周期闪烁，表示系统调度正常。
- 当用户按下按键时，会触发中断回调，并在串口终端输出：

```
The button is pressed
```

注意事项

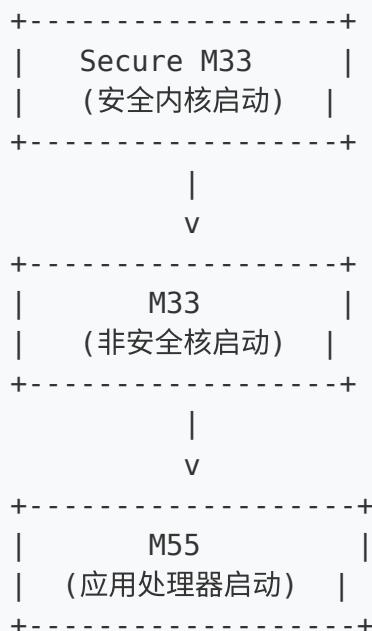
- 如需修改工程的 图形化配置，请使用以下工具打开配置文件：

```
tools/device-configurator/device-configurator.exe  
libs/TARGET_APP_KIT_PSE84_EVAL_EPC2/config/design.modus
```

- 修改完成后保存配置，并重新生成代码。

启动流程

系统启动顺序如下：



⚠ 请严格按照以上顺序烧写固件，否则系统可能无法正常运行。

- 若示例工程无法正常运行，建议先编译并烧录 **Edgi-Talk_M33_S_Template** 工程，确保初始化与核心启动流程正常，再运行本示例。
- 若要开启 M55，需要在 **M33 工程** 中打开配置：

```
RT-Thread Settings --> 硬件 --> select SOC Multi Core Mode -->
```

2.8. Edgi-Talk_LSM6DS3 示例工程

[中文](#) | [English](#)

简介

本示例基于 **Edgi-Talk** 平台，运行于 **RT-Thread** 实时操作系统，演示如何驱动 **LSM6DS3** 六轴传感器（加速度计 + 陀螺仪 + 温度）。

LSM6DS3TR 简介

LSM6DS3TR 是 STMicroelectronics（意法半导体）推出的一款 **低功耗六轴惯性测量单元（IMU）**，集成了三轴加速度计与三轴陀螺仪。

主要特性

- **三轴加速度计**: $\pm 2/\pm 4/\pm 8/\pm 16\text{ g}$
- **三轴陀螺仪**: $\pm 125/\pm 245/\pm 500/\pm 1000/\pm 2000\text{ dps}$
- **工作电压**: $1.71\text{ V} \sim 3.6\text{ V}$
- **功耗低**，支持多种省电模式
- **内置 FIFO 缓冲**（最长 8 KB）
- 支持 **I²C** 与 **SPI** 通信接口

应用场景

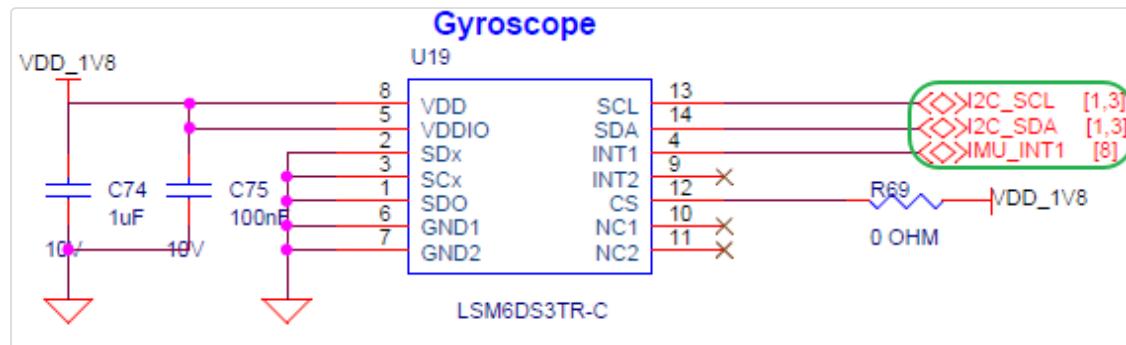
- 智能手机、可穿戴设备
- 运动检测与姿态识别
- 手势识别与步态分析
- 机器人与无人机姿态控制

通过该示例，用户可以学习：

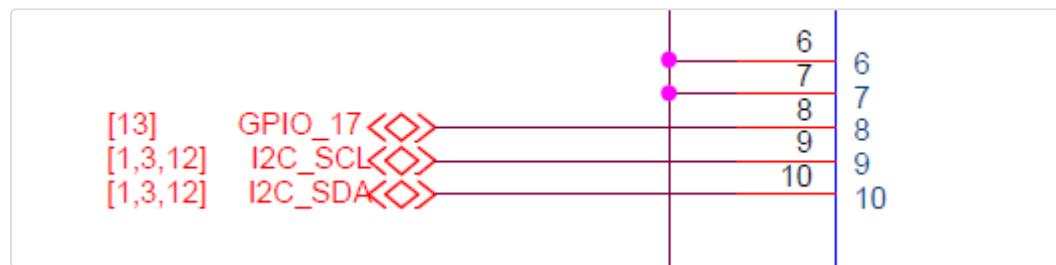
- 使用 RT-Thread 的 I²C 设备驱动框架
- 初始化并配置 LSM6DS3 寄存器
- 读取 **三轴加速度**、**三轴角速度** 和 **温度数据**
- 将传感器数据通过串口输出

硬件说明

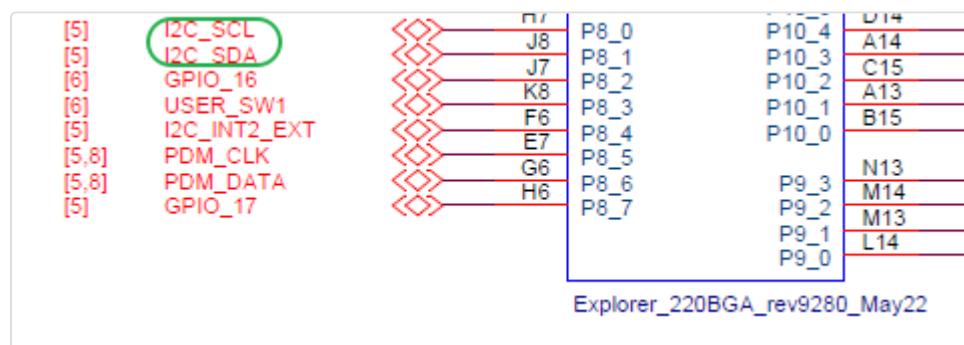
LSM6DS3TR接口



BTB座子



MCU接口



软件说明

- 工程基于 Edgi-Talk 平台开发。
- 使用 RT-Thread 作为操作系统内核。

- 示例功能包括：
- 检测并验证设备 ID
- 传感器复位并恢复默认配置
- 配置输出速率与量程
- 轮询方式读取加速度、角速度、温度数据
- 串口打印输出

使用方法

编译与下载

1. 打开工程并完成编译。
2. 使用 **板载下载器 (DAP)** 将开发板 USB 接口连接至 PC。
3. 通过编程工具将生成的固件烧录至开发板。

运行效果

- 烧录完成后，开发板上电即可运行示例工程。

```
Acceleration [mg]: 15.23 -3.12 1000.45  
Angular rate [mdps]: 2.50 -1.25 0.75  
Temperature [degC]: 26.54
```

- **蓝色 LED** 将以 500ms 为周期闪烁，表示系统调度正常。

注意事项

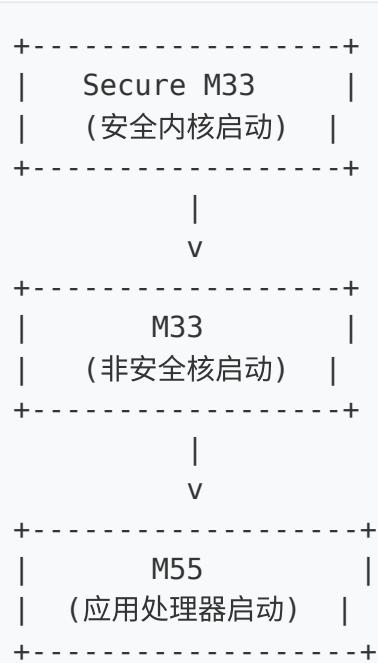
- 如需修改工程的 **图形化配置**，请使用以下工具打开配置文件：

```
tools/device-configurator/device-configurator.exe  
libs/TARGET_APP_KIT_PSE84_EVAL_EPC2/config/design.modus
```

- 修改完成后保存配置，并重新生成代码。

启动流程

系统启动顺序如下：



⚠ 请严格按照以上顺序烧写固件，否则系统可能无法正常运行。

- 若示例工程无法正常运行，建议先编译并烧录 **Edgi-Talk_M33_S_Template** 工程，确保初始化与核心启动流程正常，再运行本示例。
- 若要开启 M55，需要在 **M33 工程** 中打开配置：

```
RT-Thread Settings --> 硬件 --> select SOC Multi Core Mode -->
```

2.9. Edgi-Talk_M55_MIPI_LCD 示例工程

中文 | English

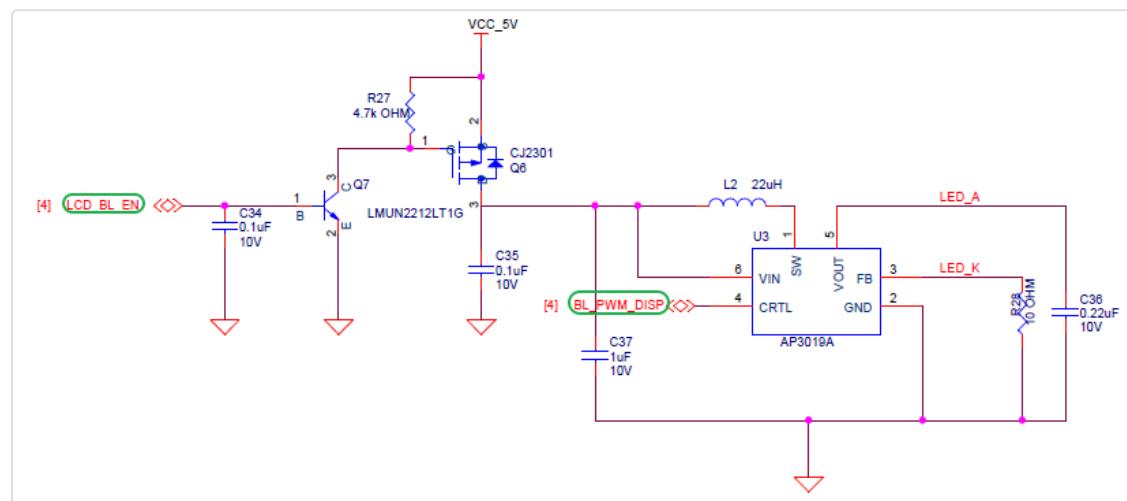
简介

本示例工程基于 Edgi-Talk 平台，演示 MIPI LCD 屏幕刷屏功能，运行在 RT-Thread 实时操作系统上。

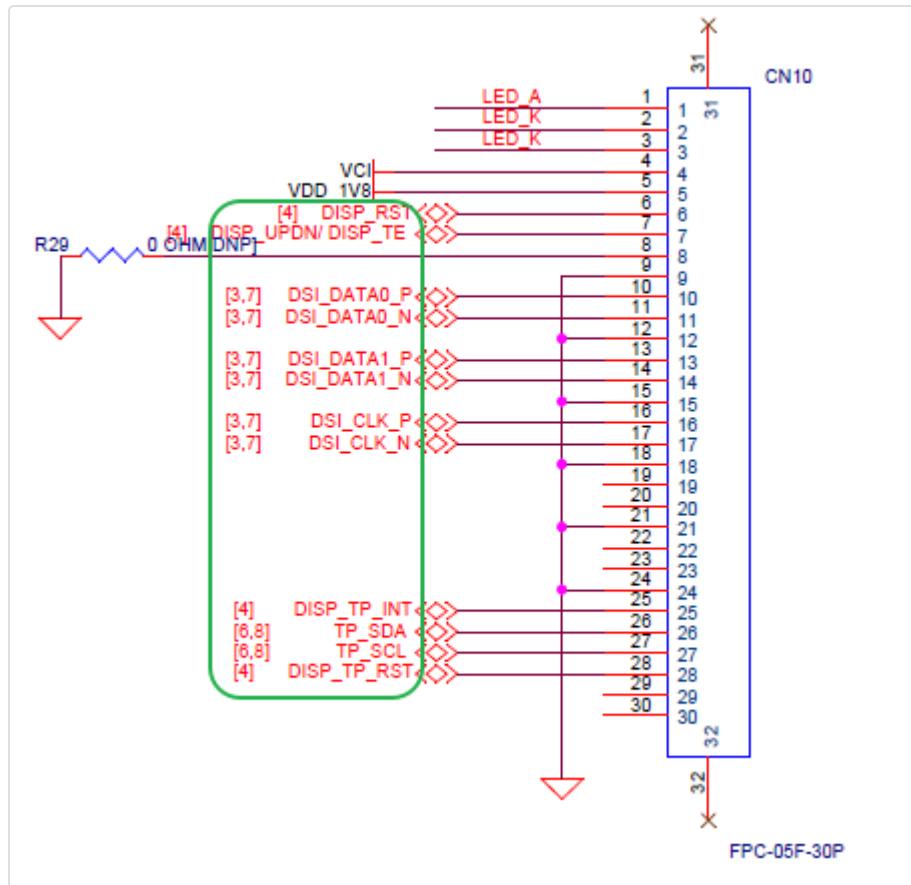
通过本工程，用户可以快速验证 MIPI DSI 接口驱动 和 LCD 显示初始化，为后续 GUI 应用或 LVGL 移植提供参考。

硬件说明

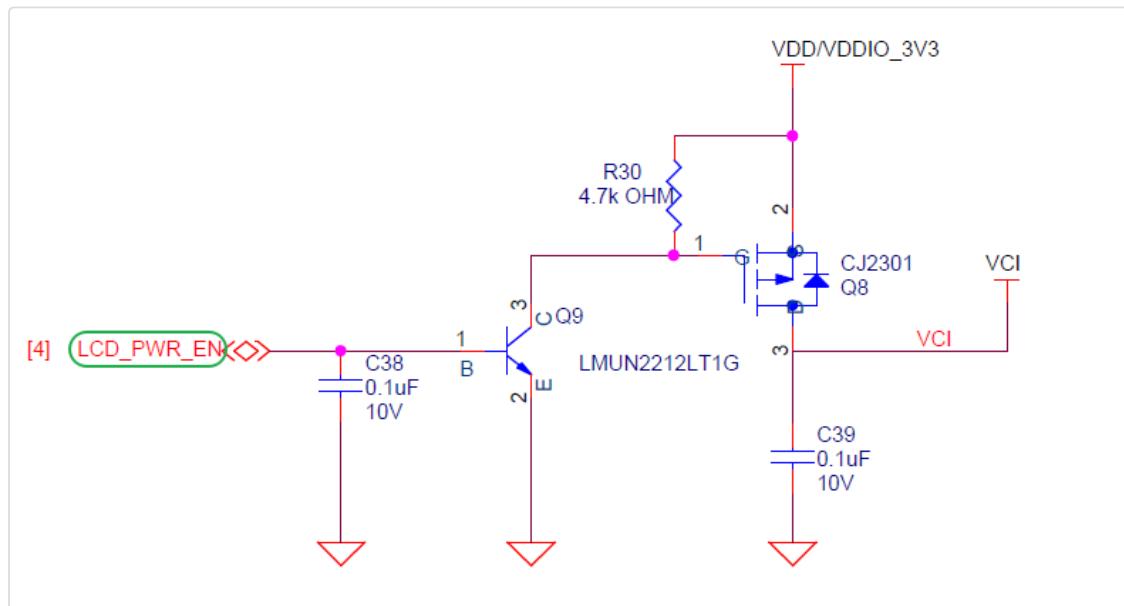
背光接口



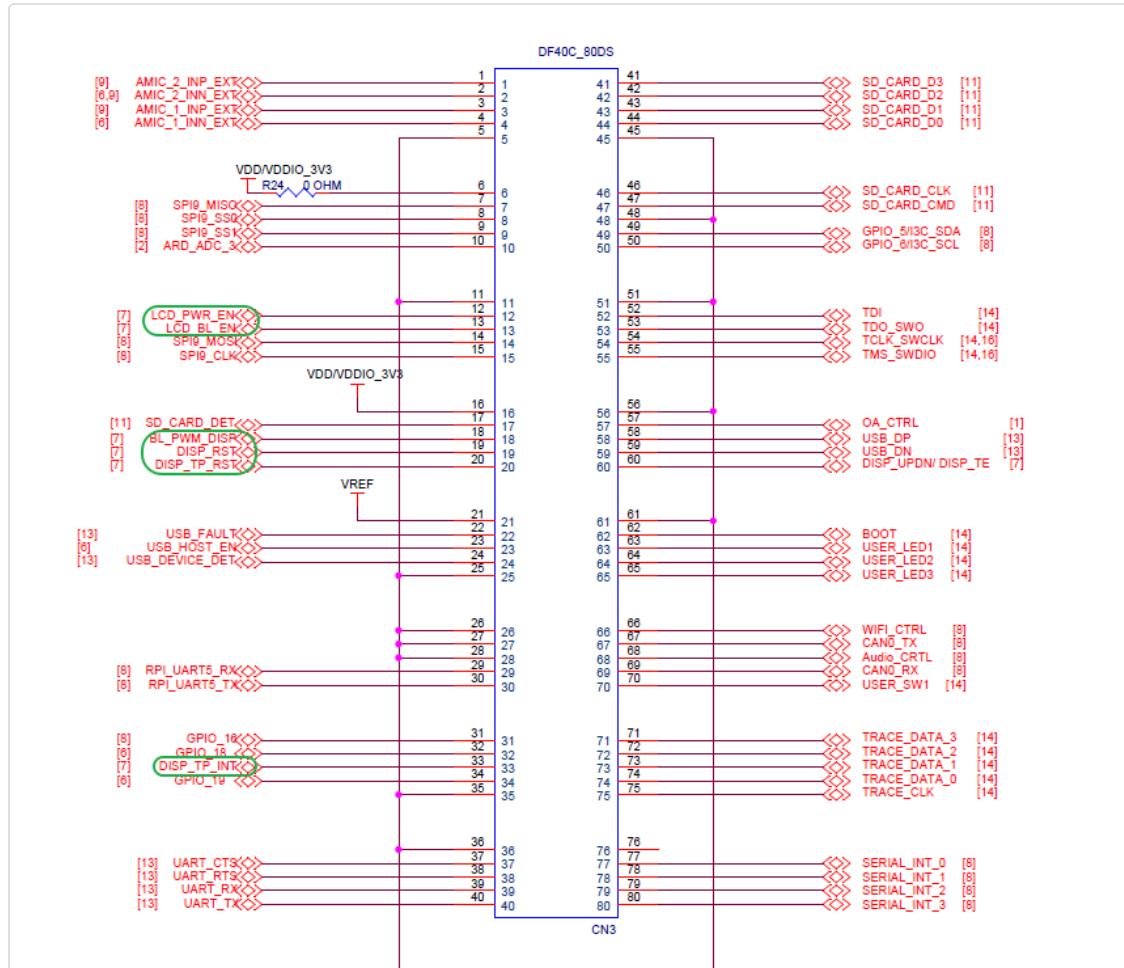
MIPI接口

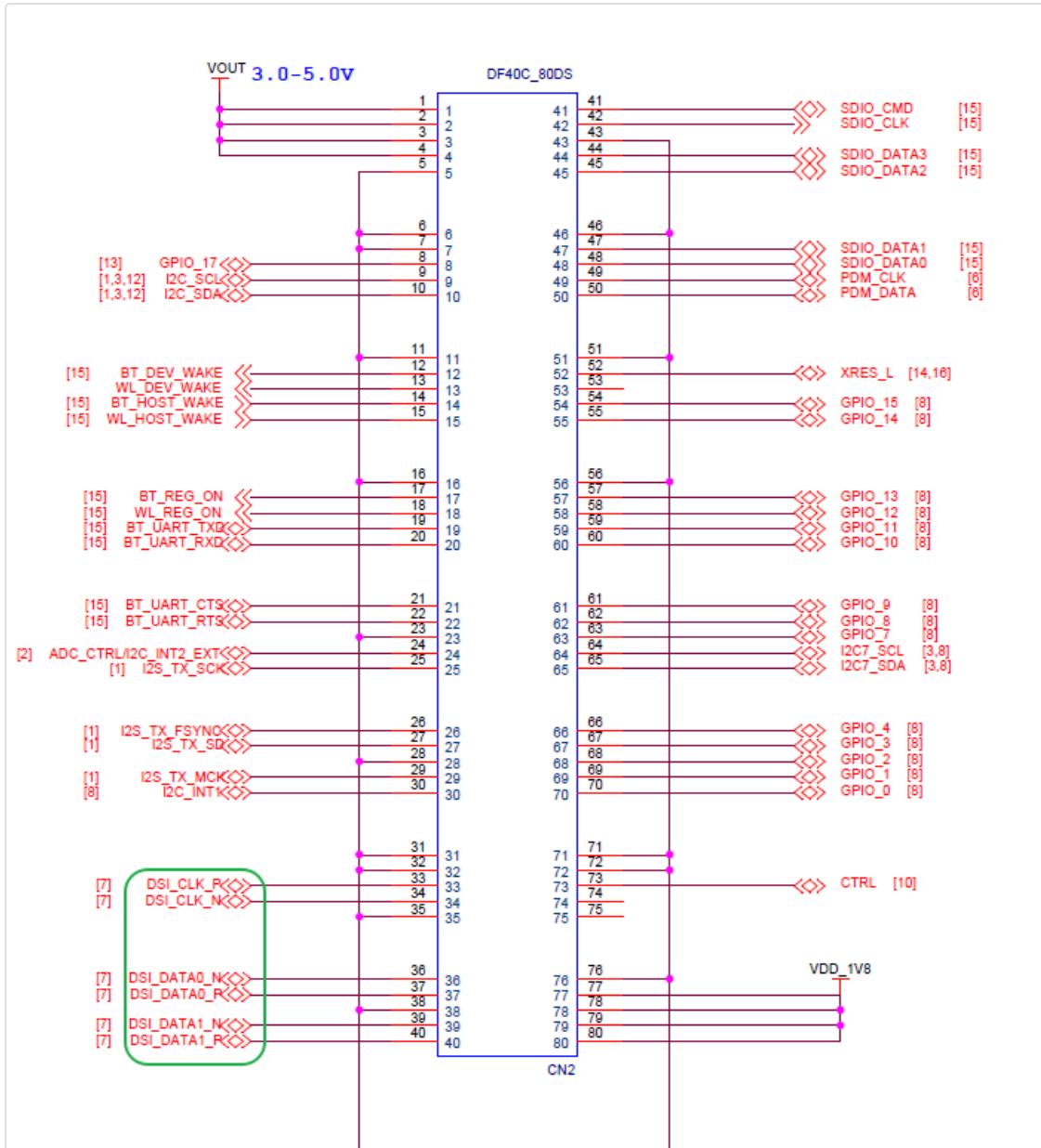


PWR接口

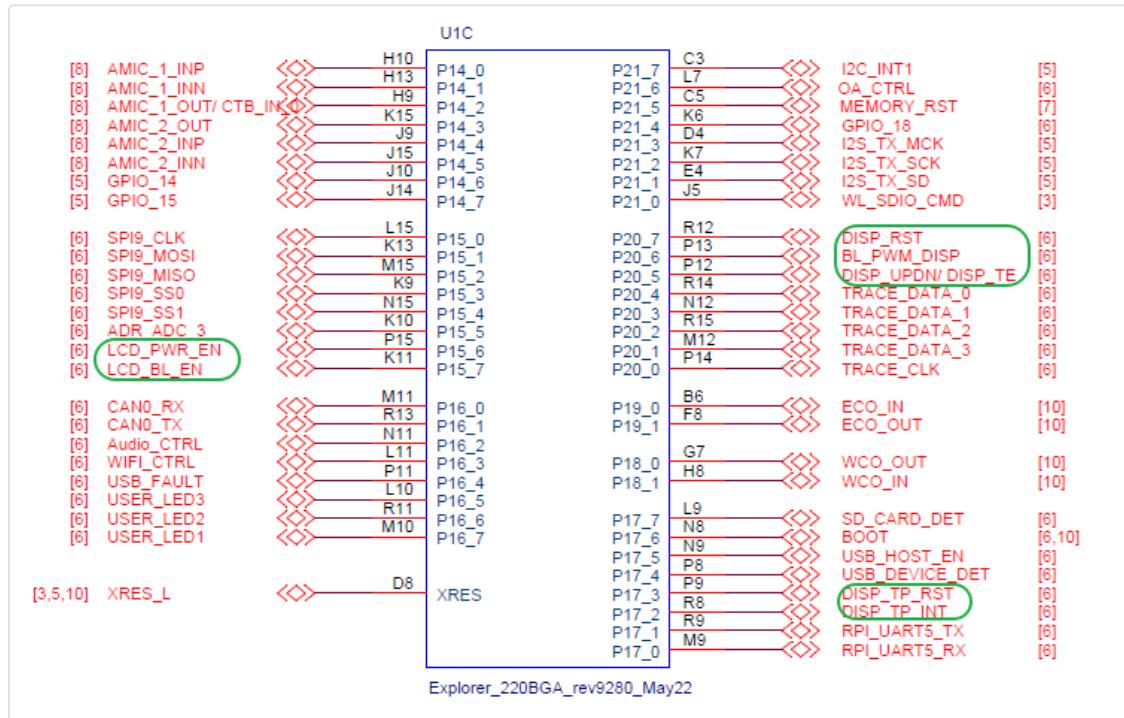
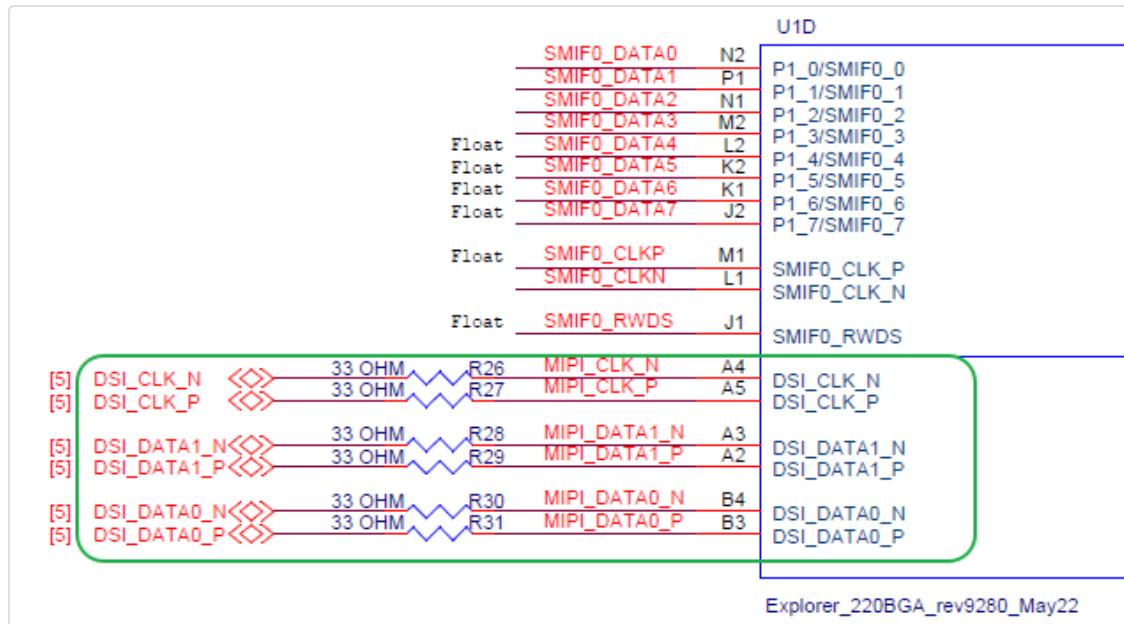


BTB座子





MCU接口



软件说明

- 工程基于 Edgi-Talk 平台开发，运行在 M55 应用核上。
- 示例功能包括：
- 初始化 MIPI DSI 接口与 LCD 面板

- 在 LCD 上进行单色填充
- 循环刷屏以验证 LCD 显示性能
- 工程结构简洁，便于理解 MIPI DSI 初始化流程 和 LCD 刷屏逻辑。

使用方法

编译与下载

1. 打开工程并完成编译。
2. 使用 **板载下载器 (DAP)** 将开发板的 USB 接口连接至 PC。
3. 通过编程工具将生成的固件烧录至开发板。

运行效果

- 烧录完成后，开发板上电即可运行示例工程。
- LCD 屏幕会依次 **刷单色**，每种颜色保持。
- 刷屏过程持续循环，用户可根据需要修改刷新周期或增加更多测试图案。

注意事项

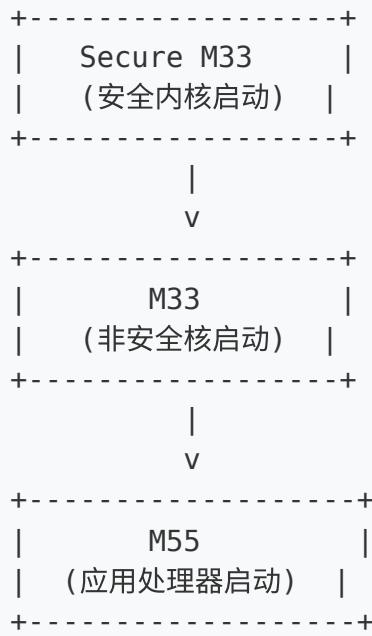
- 如需修改工程的 **图形化配置**，请使用以下工具打开配置文件：

```
tools/device-configurator/device-configurator.exe  
libs/TARGET_APP_KIT_PSE84_EVAL_EPC2/config/design.modus
```

- 修改完成后保存配置，并重新生成代码。
- 若屏幕无显示，请检查：
 - MIPI DSI 硬件连接是否正常
 - LCD 电源、背光供电是否开启

启动流程

系统启动顺序如下：



⚠ 请严格按照以上顺序烧写固件，否则系统可能无法正常运行。

- 若示例工程无法正常运行，建议先编译并烧录 **Edgi-Talk_M33_S_Template** 工程，确保初始化与核心启动流程正常，再运行本示例。
- 若要开启 M55，需要在 **M33 工程** 中打开配置：

```
RT-Thread Settings --> 硬件 --> select SOC Multi Core Mode --> E
```

2.10. Edgi-Talk_RTC 示例工程

中文 | English

简介

本示例工程基于 Edgi-Talk 平台，演示 RTC（实时时钟）功能，运行在 RT-Thread 实时操作系统（M33 核）上。

通过本工程，用户可以快速体验 RTC 时间设置、读取与延时打印功能，为后续时间管理与定时任务开发提供参考。

RTC（实时时钟）简介

1. 概述

RTC（Real-Time Clock）是一种用于 **计量和跟踪实际时间（年、月、日、时、分、秒）** 的电子模块或芯片。它通常内置在 **微控制器（MCU）** 或作为独立芯片存在，用于提供 **系统时间、定时唤醒、定时事件触发** 等功能。

RTC 的核心特点是 **低功耗、长时间稳定运行**，即使系统主电源断电，也可通过备用电源（如电池或超级电容）继续计时。

2. 工作原理

RTC 基本上是一个 **低功耗振荡器 + 计数器** 系统：

1. 时钟源

- RTC 通常使用 **32.768 kHz 石英晶振** ($1 \text{ Hz} = 2^{15}$ 周期)
- 提供稳定时基

2. 分频计数

- 将晶振信号分频得到 1 Hz 的秒脉冲
- 累计计数生成分钟、小时、天、月、年

3. 寄存器存储

- RTC 内部寄存器保存当前时间、日期、闹钟设置等

4. 电源冗余

- 内部电池或超级电容供电，使 RTC 在主电源断电时继续运行

3. RTC 类型

1. 内部 RTC

- 集成在 MCU 内部
- 优点：节省芯片数量，成本低
- 缺点：晶振精度受 PCB 布线、温度影响

2. 外部 RTC 芯片

- 独立芯片，如 DS3231、PCF8563
- 优点：高精度、可用 I²C/SPI 接口连接 MCU
- 缺点：增加 PCB 封装面积和成本

4. 关键参数

参数	描述
振荡器频率	通常 32.768 kHz，低功耗稳定
时间精度	ppm / 秒/天，决定时钟漂移
电源电压	1.8~5V，支持备用电源
功耗	1~5 μA (低功耗模式)
接口类型	I ² C、SPI 或 MCU 内部总线
功能扩展	闹钟输出、方波输出、温度补偿、定时唤醒

5. RTC 功能

1. 实时时钟

- 提供当前时间和日期

6. RTC 应用场景

• 嵌入式设备

- MCU 实时时间追踪，事件记录

- **低功耗物联网设备**
- RTC 唤醒 MCU 进行周期性数据采集
- **手表与智能穿戴设备**
- 提供精准计时功能
- **数据记录仪与工业控制**
- 时间戳、日志记录
- **汽车电子**
- 行车记录仪、车载娱乐系统计时

软件说明

- 工程基于 **Edgi-Talk** 平台开发。
- 示例功能包括：
- RTC 设备初始化
- 日期和时间设置
- 延时获取当前时间并打印
- 将 RTC 操作导出为 msh 命令 `rtc_sample`
- 工程结构清晰，便于理解 RTC 驱动在 **M33 核** 上的运行方式。

使用方法

编译与下载

1. 打开工程并完成编译。
2. 使用 **板载下载器 (DAP)** 将开发板的 USB 接口连接至 PC。
3. 通过编程工具将生成的固件烧录至开发板。

运行效果

- 烧录完成后，开发板上电即可运行示例工程。
- 用户需在串口命令行手动输入：

rtc_sample

- 系统会依次执行以下操作：

- 初始化 RTC 设备

- 设置日期为 **2025-07-01**

- 设置时间为 **23:59:50**

- 打印当前时间

- 延时 3 秒

- 再次打印当前时间

- 示例输出：

```
Tue Jul 1 23:59:50 2025
```

```
Tue Jul 1 23:59:53 2025
```

注意事项

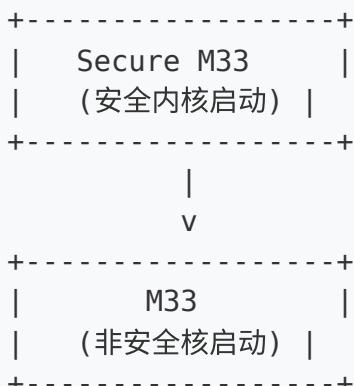
- 请确保 RTC 设备已正确连接并可被系统识别。
- 如需修改工程的 **图形化配置**，请使用以下工具打开配置文件：

```
tools/device-configurator/device-configurator.exe  
libs/TARGET_APP_KIT_PSE84_EVAL_EPC2/config/design.modus
```

- 修改完成后保存配置，并重新生成代码。

启动流程

系统启动顺序如下：





⚠ 请严格按照以上顺序烧写固件，否则系统可能无法正常运行。

- 若示例工程无法正常运行，建议先编译并烧录 **Edgi-Talk_M33_S_Template** 工程，确保初始化与核心启动流程正常，再运行本示例。
- 若要开启 M55，需要在 **M33 工程** 中打开配置：

```
RT-Thread Settings --> 硬件 --> select SOC Multi Core Mode --> E
```

2.11. Edgi-Talk_SDCARD 示例工程

中文 | English

简介

本示例工程基于 Edgi-Talk 平台，演示 SD 卡文件操作功能，运行在 RT-Thread 实时操作系统 (M33 核) 上。

通过本工程，用户可以快速体验 SD 卡的挂载、文件写入与读取功能，并通过 串口命令行 操作文件，为后续存储应用开发提供参考。

SD 卡简介

1. 概述

SD 卡 (Secure Digital Card) 是一种小型、便携的非易失性存储设备，广泛用于 嵌入式系统、相机、手机、数据记录仪 等场景。

SD 卡由 **控制器 + NAND Flash 存储芯片** 组成，外部通过标准接口与主机通信。

主要特点：

- 小巧轻便，体积通常为 **32 × 24 × 2.1 mm**（标准卡）
- 采用 **非易失性闪存 (NAND Flash)** 存储数据
- 支持热插拔和掉电保护

2. SD 卡类型

1. 按尺寸分类

- **标准 SD 卡 (Standard SD)**: 32 × 24 mm
- **Mini SD**: 21.5 × 20 mm
- **Micro SD (TF 卡)**: 15 × 11 mm，最常用

2. 按容量分类

类型	容量范围
SDSC	1 MB ~ 2 GB

类型	容量范围
SDHC	4 GB ~ 32 GB
SDXC	32 GB ~ 2 TB
SDUC	2 TB ~ 128 TB

3. 按速度等级

- **Class 2/4/6/10**: 最低写入速度分别为 2、4、6、10 MB/s
- **UHS (Ultra High Speed)**: UHS-I/UHS-II/UHS-III，速率可达 **312 MB/s**
- **Video Speed Class (V6/V10/V30/V60/V90)**: 适用于高清视频录制

3. SD 卡接口

1. SPI 模式

- 使用 SPI 总线 (MISO, MOSI, SCK, CS)
- 简单易用，适合 MCU
- 数据传输速率较低

2. SD 模式 (1-bit / 4-bit)

- 使用专用 SD 总线
- 支持 1-bit 或 4-bit 数据线
- 速率高于 SPI 模式

3. UHS 模式

- 支持高速数据传输，常用于摄像机和高性能嵌入式应用

4. 工作原理

1. 命令/数据传输

- 主机通过 SD 卡协议发送命令 (CMD)
- 卡片返回响应 (R1, R2 等)
- 读写数据块 (Block)，每块通常为 **512 Byte**

2. 控制器管理

- 内部控制器负责 坏块管理、错误纠正（ECC）、逻辑到物理地址映射
- 外部主机无需直接管理 NAND Flash 特性

3. 数据存储

- 数据存储在 NAND Flash 中
- 支持多次擦写和擦写寿命管理（典型寿命 10 万次擦写）

5. SD 卡性能指标

参数	描述
容量	1 GB ~ 128 TB
数据块大小	512 Byte (标准)
接口速率	SPI/SD 1-bit/4-bit/UHS
最大传输速度	25 MB/s (标准), 312 MB/s (UHS-III)
工作电压	3.3 V (部分 Micro SD 支持 1.8 V)
工作温度	-25 °C ~ 85 °C (工业级)
耐用性	擦写次数 10^4 ~ 10^5

6. SD 卡应用场景

1. 消费电子

- 手机、平板、数码相机、摄像机存储

2. 嵌入式系统

- MCU/FPGA 数据存储
- 日志记录、配置文件保存

3. 工业应用

- 工业控制器、数据采集系统
- 高温环境工业级 SD 卡可使用

4. 音视频应用

- 高速视频录制 (UHS/V Class)

5. 汽车电子

- 行车记录仪、导航系统数据存储

软件说明

- 工程基于 **Edgi-Talk** 平台开发。
- 示例功能包括：
- SD 卡挂载与初始化
- 使用 `echo` 命令写入文件
- 使用 `cat` 命令读取文件内容
- 串口打印操作结果
- 工程结构清晰，便于理解 RT-Thread 文件系统与 SD 卡接口。

使用方法

编译与下载

1. 打开工程并完成编译。
2. 使用 **板载下载器 (DAP)** 将开发板的 USB 接口连接至 PC。
3. 通过编程工具将生成的固件烧录至开发板。
4. 插入 SD 卡至开发板对应接口。

运行效果

- 烧录完成后，开发板上电即可运行示例工程。
- 系统会自动挂载 SD 卡，并初始化文件系统。
- 用户可在 **串口终端** 使用以下命令操作文件：

1. 写入文件 `test.txt`：

```
echo "Hello RT-Thread SDCARD!" ./test.txt
```

2. 读取文件内容：

```
cat ./test.txt
```

- 串口输出示例：

```
\ | /  
- RT -      Thread Operating System  
/ | \      5.0.2 build Sep  8 2025 11:02:30  
2006 - 2022 Copyright by RT-Thread team  
found part[0], begin: 1048576, size: 29.739GB  
Hello RT-Thread  
This core is cortex-m33  
Mount SD card success!
```

```
> echo "Hello RT-Thread SDCARD!" ./test.txt  
> cat ./test.txt  
Hello RT-Thread SDCARD!
```

- 使用 `echo` 命令可以写入任意字符串到 SD 卡文件中，`cat` 命令可以读取并显示文件内容。

注意事项

- 请确保 SD 卡已正确插入且格式化为 FAT 文件系统 (FAT16/FAT32)。
- 如需修改工程的 **图形化配置**，请使用以下工具打开配置文件：

```
tools/device-configurator/device-configurator.exe  
libs/TARGET_APP_KIT_PSE84_EVAL_EPC2/config/design.modus
```

- 修改完成后保存配置，并重新生成代码。
- 如果 SD 卡无法挂载，请检查硬件接口和电源配置。

启动流程

系统启动顺序如下：





⚠ 请严格按照以上顺序烧写固件，否则系统可能无法正常运行。

- 若示例工程无法正常运行，建议先编译并烧录 **Edgi-Talk_M33_S_Template** 工程，确保初始化与核心启动流程正常，再运行本示例。
- 若要开启 M55，需要在 **M33 工程** 中打开配置：

```
RT-Thread Settings --> 硬件 --> select SOC Multi Core Mode --> E
```

2.12. Edgi-Talk_WIFI 示例工程

中文 | English

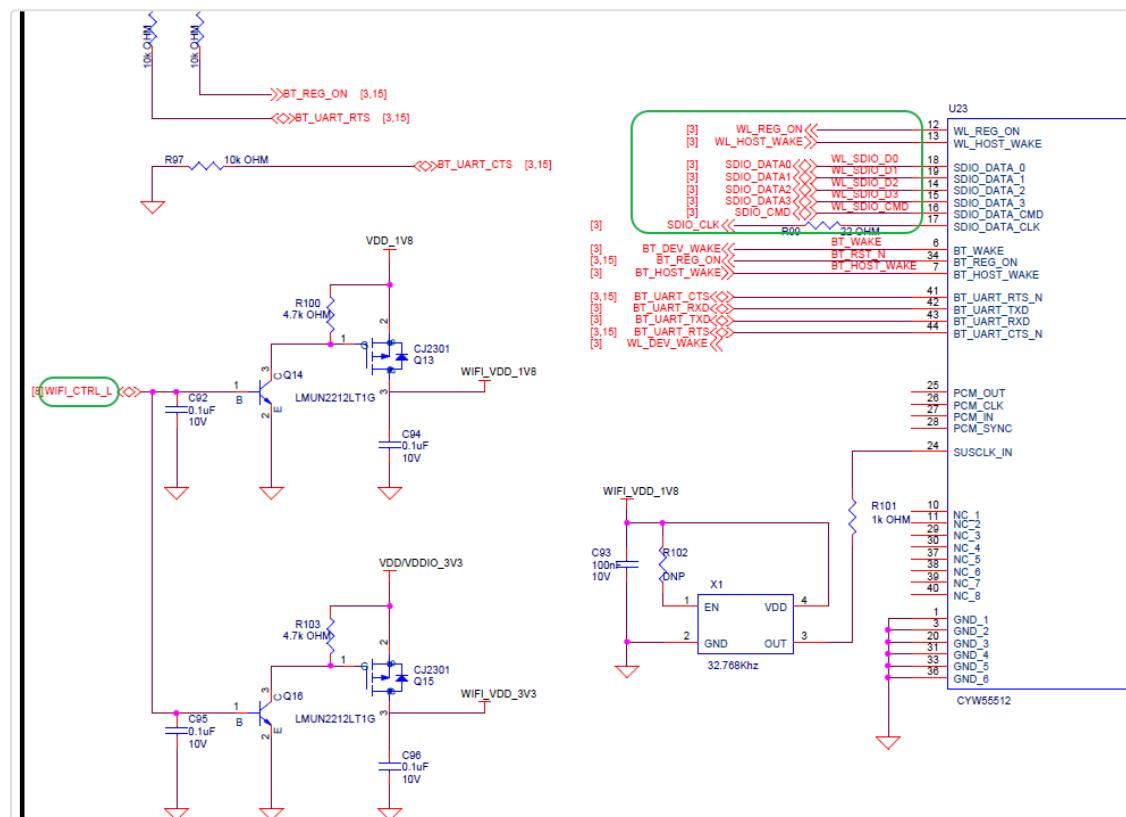
简介

本示例工程基于 Edgi-Talk 平台，演示 WIFI 功能，运行在 RT-Thread 实时操作系统 (M55 核) 上。

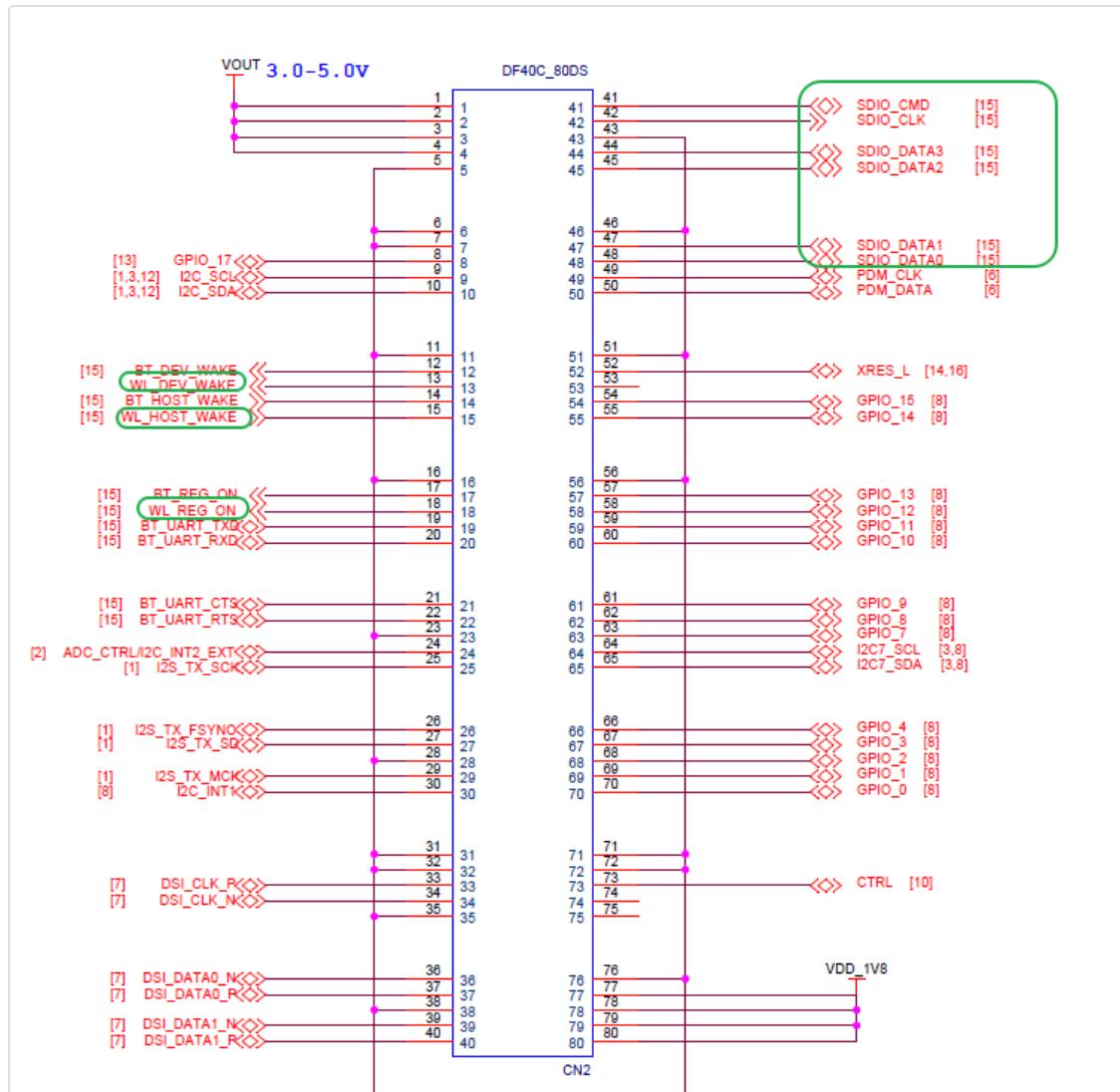
通过本工程，用户可以快速体验 WIFI 的联网功能，并验证 WIFI 模块的接口，为后续 WIFI 的开发提供参考。

硬件说明

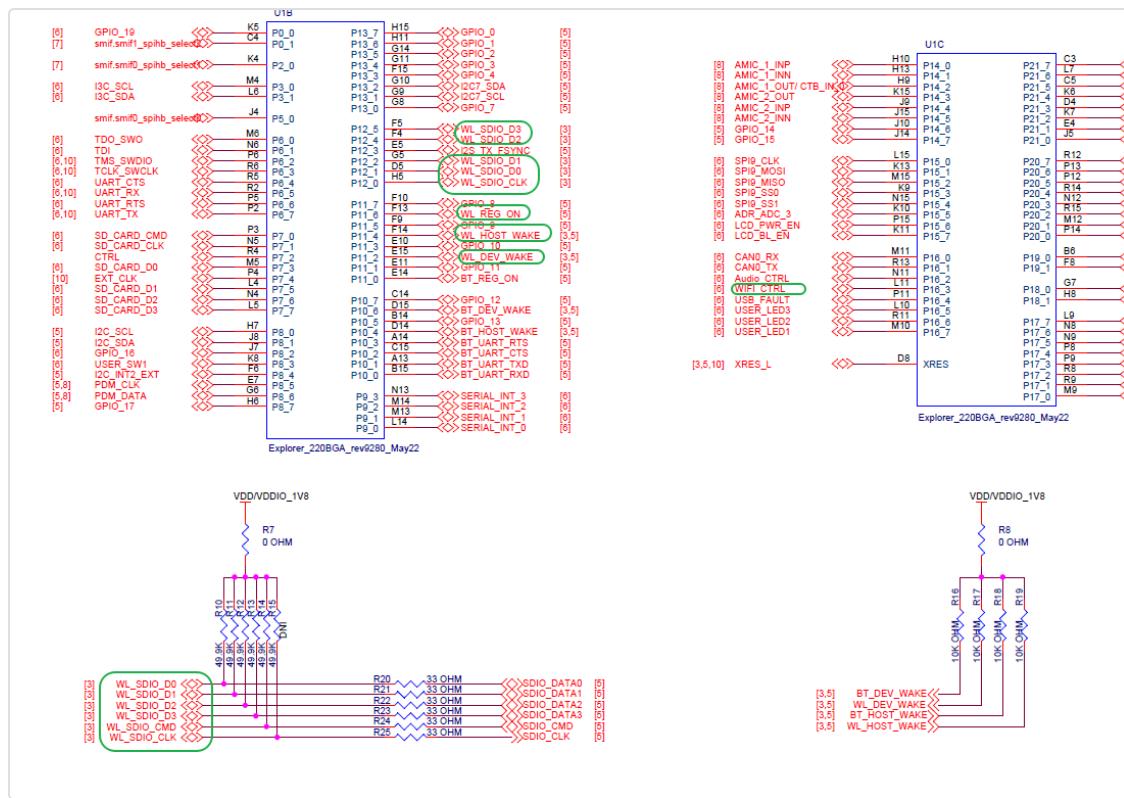
WIFI 接口



BTB座子



MCU接口



软件说明

- 工程基于 Edgi-Talk 平台开发。
- 示例功能包括：
- WIFI的扫描
- WIFI的连接
- Iperf测速
- 工程结构清晰，便于理解WIFI驱动和 RT-Thread 系统的配合使用。

使用方法

编译与下载

- 打开工程并完成编译。
- 使用 **板载下载器 (DAP)** 将开发板的 USB 接口连接至 PC。

3. 通过编程工具将生成的固件烧录至开发板。

运行效果

- 烧录完成后，开发板上电即可运行示例工程。
- 系统会自动初始化 WIFI 设备。
- 用户可在串口终端使用以下命令连接 WIFI：

```
wifi scan
```

```
[I/WLAN.dev] wlan init success
[I/WLAN.lwip] eth device init ok name:w0
[I/WLAN.dev] wlan init success
[I/WLAN.lwip] eth device init ok name:w1
wifi scan
          SSID           MAC      security   rssi  chn Mbps
-----+-----+-----+-----+-----+-----+-----+
realthread_5G        3c:6a:48:77:74:15  WPA2_AES_PSK -87   36  300
realthread_5G        3c:6a:48:c9:c2:79  WPA2_AES_PSK -81   36  300
realthread_VIP5G     3e:6a:48:97:74:14  WPA2_AES_PSK -88   36  300
realthread_VIP5G     3e:6a:48:99:c2:78  WPA2_AES_PSK -81   36  300
realthread_5G        3c:6a:48:77:74:15  WPA2_AES_PSK -88   36  300
realthread_VIP5G     3e:6a:48:97:74:14  WPA2_AES_PSK -88   36  300
TK-ML                9c:74:6f:b9:5d:b0  WPA_AES_PSK -79    1  144
TK-ML-GUEST         9c:74:6f:b9:5d:b1  WPA_AES_PSK -79    1  144
TK-ML                9c:74:6f:b9:5d:b0  WPA_AES_PSK -79    1  144
TK-ML-GUEST         9c:74:6f:b9:5d:b1  WPA_AES_PSK -79    1  144
TK-ML                9c:74:6f:b9:5d:b0  WPA_AES_PSK -79    1  144
TK-ML-GUEST         9c:74:6f:b9:5d:b1  WPA_AES_PSK -79    1  144
TEST                 82:5f:0e:3f:19:c8  WPA2_AES_PSK -39    6  300
TEST                 82:5f:0e:3f:19:c8  WPA2_AES_PSK -39    6  300
realthread            3c:6a:48:77:74:14  WPA2_AES_PSK -70    6  300
realthread_VIP        3e:6a:48:17:74:14  WPA2_AES_PSK -71    6  300
MLMEETING02          68:dd:b7:f9:d1:49  WPA2_AES_PSK -77   11  300
realthread_5G        3c:6a:48:77:74:15  WPA2_AES_PSK -87   36  300
realthread_VIP5G     3e:6a:48:97:74:14  WPA2_AES_PSK -88   36  300
realthread_5G        3c:6a:48:c9:c2:79  WPA2_AES_PSK -80   36  300
realthread_VIP5G     3e:6a:48:99:c2:78  WPA2_AES_PSK -80   36  300
TK-ML-GUEST         9c:74:6f:b9:5d:b1  WPA_AES_PSK -79    1  144
ChinaNet             00:23:89:04:be:e1  OPEN      -78    1  54
realthread_VIP        3e:6a:48:17:74:14  WPA2_AES_PSK -96    6  300
TEST                 82:5f:0e:3f:19:c8  WPA2_AES_PSK -39    6  300
realthread            3c:6a:48:c9:c2:78  WPA2_AES_PSK -71    6  300
realthread_VIP        3e:6a:48:19:c2:78  WPA2_AES_PSK -71    6  300
realthread            3c:6a:48:77:74:14  WPA2_AES_PSK -71    6  300
realthread_VIP        3e:6a:48:17:74:14  WPA2_AES_PSK -71    6  300
TEST                 82:5f:0e:3f:19:c8  WPA2_AES_PSK -39    6  300
SH-online            00:23:89:04:bf:30  OPEN      -72   11  54
ChinaNet             00:23:89:04:bf:31  OPEN      -72   11  54
```

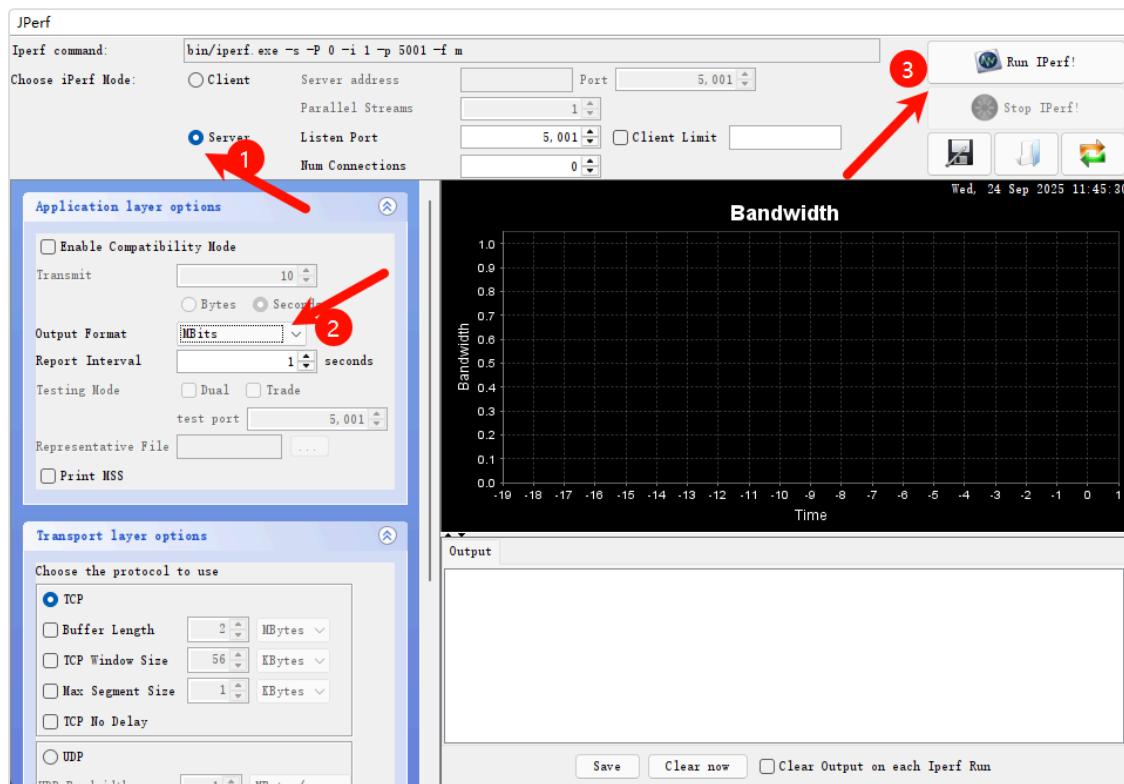
```
wifi join 名称 密码
```

```
msh />wifi join TEST 88888888
[I/WLAN.mgmt] wifi connect success ssid:TEST
msh />[I/WLAN.lwip] Got IP address : 10.19.22.51
```

```
ping www.rt-thread.org
```

```
msh />ping www.rt-thread.org
ping: not found specified netif, using default netdev w0.
60 bytes from 111.31.66.86 icmp_seq=0 ttl=52 time=61 ms
60 bytes from 111.31.66.86 icmp_seq=1 ttl=53 time=53 ms
60 bytes from 111.31.66.86 icmp_seq=2 ttl=53 time=37 ms
60 bytes from 111.31.66.86 icmp_seq=3 ttl=53 time=65 ms
```

- 网络连接完成后，可使用 iperf 进行性能测试。
- 在 packages\netutils-latest\tools 目录下提供了 jperf.rar 测速工具。
- 将其解压后，双击其中的 .bat 文件，即可启动工具，界面如下图所示：



- 在开发板终端输入以下命令（其中 电脑的 IP 请替换为实际地址），即可开始测速：

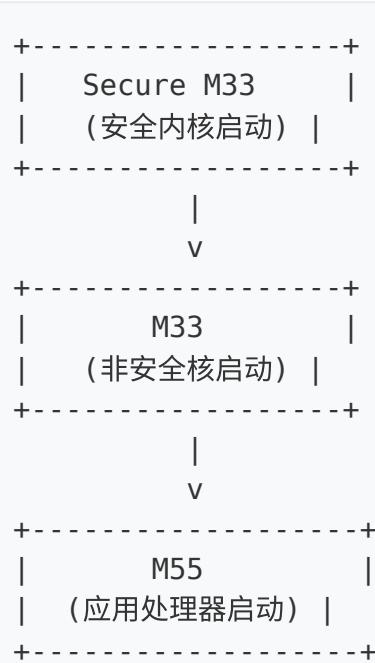
```
iperf -c <电脑的IP>
```

注意事项

- 可以使用电脑开热点进行测试，频段最好为2.4G。

启动流程

系统启动顺序如下：



⚠ 请严格按照以上顺序烧写固件，否则系统可能无法正常运行。

- 若示例工程无法正常运行，建议先编译并烧录 **Edgi-Talk_M33_S_Template** 工程和 **Edgi-Talk_M33_Template** 工程，确保初始化与核心启动流程正常，再运行本示例。
- 若要开启 M55，还需要在 **Edgi-Talk_M33_Template** 工程中打开配置：

```
RT-Thread Settings --> 硬件 --> select SOC Multi Core Mode --> E
```

3. 示例

3.1. Edgi-Talk_M55_CoreMark 示例工程

[中文](#) | [English](#)

简介

本示例工程基于 **Edgi-Talk** 平台，演示 M55 核心运行 **CoreMark** 基准测试的功能，运行在 **RT-Thread** 实时操作系统上。

通过本工程，用户可以快速验证 M55 核心性能，并了解多核协处理器在实时操作系统下的运行情况。

CoreMark 简介

CoreMark 是由 *EEMBC*（嵌入式微控制器基准评测联盟）开发的一个标准化嵌入式 CPU 性能测试基准。

它主要用于衡量微控制器或处理器的 **核心运算性能**，而不依赖特定的硬件外设。

测试内容

CoreMark 通过以下四类典型算法评估 CPU 性能：

- **列表处理 (List processing)**
- **矩阵运算 (Matrix operations)**
- **状态机 (State machine)**
- **CRC 校验 (Cyclic Redundancy Check)**

测试结果

输出结果以 **CoreMark/MHz** 或 **CoreMark** 表示，用于比较不同处理器或编译优化下的性能。

特点

- 开源、可移植、轻量级
- 结果可重复、易于验证
- 专注于 CPU 的整数计算能力

软件说明

- 工程基于 **Edgi-Talk** 平台开发。
- 示例功能包括：
 - M55 核心运行 CoreMark 基准测试
 - 串口打印测试结果
- 工程结构简洁，便于用户理解 M55 的启动流程及性能测试方法。

使用方法

编译与下载

1. 打开工程并完成编译。
2. 使用 **板载下载器 (DAP)** 将开发板的 USB 接口连接至 PC。
3. 通过编程工具将生成的固件烧录至开发板。

运行效果

- 烧录完成后，开发板上电即可启动 RT-Thread。
- 串口输出如下内容表示系统正常启动：

```
\ | /
- RT -      Thread Operating System
/ | \      5.0.2 build Sep  5 2025 15:19:27
2006 - 2022 Copyright by RT-Thread team
msh >Hello RT-Thread
It's cortex-m55
```

- 用户需在串口命令行手动输入：

core_mark

- 系统开始 CoreMark 测试，并打印测试结果，例如：

```
Benchmark started, please make sure it runs for at least 10s.
```

```
2K performance run parameters for coremark.  
CoreMark Size      : 666  
Total ticks        : 30218  
Total time (secs) : 30  
Iterations/Sec     : 1200  
Iterations         : 36000  
Compiler version   : GCC10.2.1 20201103 (release)  
Compiler flags     :  
Memory location    : STACK  
seedcrc            : 0xe9f5  
[0]crcclist        : 0xe714  
[0]crcmatrix       : 0x1fd7  
[0]crcstate        : 0x8e3a  
[0]crcfinal        : 0xcc42  
Correct operation validated. See README.md for run and reporting  
CoreMark 1.0 : 1200 / GCC10.2.1 20201103 (release) / STACK
```

注意事项

- 如需修改工程的 **图形化配置**，请使用以下工具打开配置文件：

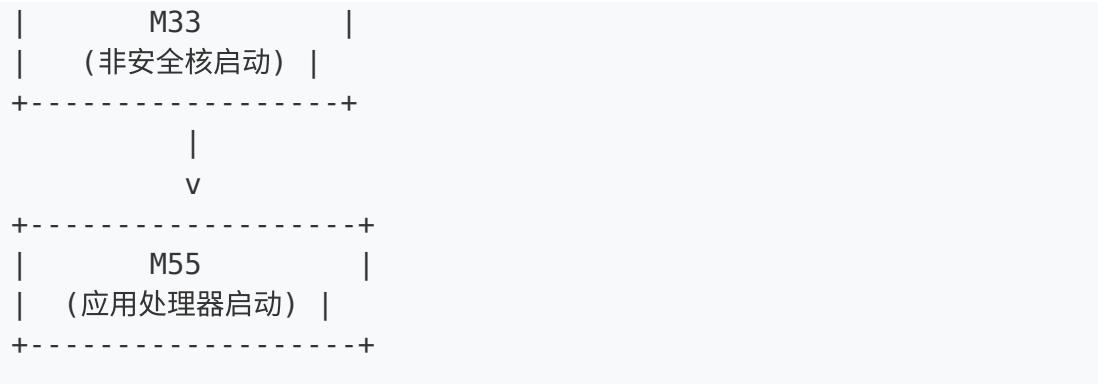
```
tools/device-configurator/device-configurator.exe  
libs/TARGET_APP_KIT_PSE84_EVAL_EPC2/config/design.modus
```

- 修改完成后保存配置，并重新生成代码。

启动流程

系统启动顺序如下：





⚠ 请严格按照以上顺序烧写固件，否则系统可能无法正常运行。

- 若示例工程无法正常运行，建议先编译并烧录 **Edgi-Talk_M33_S_Template** 工程，确保初始化与核心启动流程正常，再运行本示例。
- 若要开启 M55，需要在 **M33 工程** 中打开配置：

```
RT-Thread Settings --> 硬件 --> select SOC Multi Core Mode -->
```

3.2. Edgi-Talk_M55_LVGL 示例工程

中文 | English

简介

本示例工程基于 **Edgi-Talk 平台**，演示 **LVGL 图形库 Stress** 示例，运行在 **RT-Thread 实时操作系统** 上。

通过本工程，用户可以快速验证板级 **LCD 显示驱动** 和 **LVGL 图形框架** 的移植效果，为后续 GUI 应用开发提供基础参考。

LVGL 简介

LVGL (Light and Versatile Graphics Library，轻量且多功能的图形库) 是一个开源的嵌入式 GUI (图形用户界面) 开发框架。它的设计初衷是为资源受限的嵌入式设备提供流畅、现代化的图形界面，因此在运行效率、内存占用和可移植性方面都做了大量优化。无论是在简单的低端 MCU 上，还是在功能更强大的 MPU 平台上，LVGL 都能够高效运行，并提供丰富的图形控件和交互体验。

主要特点

1. 轻量级

LVGL 最大的优势在于轻量化，它对资源的需求非常低。在一个内存只有几十 KB 的微控制器上，LVGL 依然可以流畅运行。框架本身的内存开销较小，渲染算法经过精心优化，既可以保证较低的 CPU 使用率，又能在有限的硬件条件下呈现出较好的视觉效果。这使得 LVGL 特别适合那些对硬件资源敏感的应用场景，比如低功耗设备、可穿戴设备、家用电器控制面板等。

2. 跨平台

LVGL 的跨平台特性非常突出，它能够运行在多种操作系统之上，包括 FreeRTOS、RT-Thread、Zephyr、Linux 等，也可以直接运行在裸机环境中。开发者只需要为 LVGL 提供底层的显示驱动和输入驱动接口，便可以快速完成移植，从而在不同的硬件平台之间复用同一套 UI 代码。这种灵活性大大降低了开发成本，使得 LVGL 成为嵌入式 GUI 的通用解决方案。

3. 丰富的控件 (Widgets)

LVGL 内置了种类丰富的 GUI 控件，例如按钮、标签、滑条、进度条、复选框、列表、表格、图表等。这些控件几乎涵盖了常见人机交互界面的需

求，开发者无需从零开始设计和实现组件，大大缩短了开发周期。同时，LVGL 还允许用户基于现有控件扩展新的组件，从而构建符合特定需求的界面。无论是简单的数字显示，还是复杂的图形控制界面，LVGL 都能胜任。

4. 多样的渲染能力

在视觉呈现方面，LVGL 提供了丰富的渲染功能。它支持抗锯齿、透明度、渐变、阴影、边框、圆角等效果，可以让界面看起来更加美观和现代。除此之外，LVGL 还内置动画系统，支持多种缓动函数，能够实现平滑的控件移动、渐变过渡和动态效果。这些特性使得界面不仅功能完善，还能带来良好的用户体验。

5. 输入设备支持

LVGL 支持多种输入设备类型，包括触摸屏、电容屏、鼠标、键盘和编码器，甚至可以实现多点触控交互。它提供了统一的输入接口层，开发者只需实现底层的驱动适配，就能轻松把输入事件传递到 LVGL 的事件系统中。这样一来，开发者可以更加专注于上层的界面逻辑，而不必花费过多时间在输入事件处理上。

6. 国际化与多语言

LVGL 在国际化方面也有很好的支持。它使用 UTF-8 编码，可以处理几乎所有语言的字符集。同时，它支持双向文本渲染，可以正确显示如阿拉伯语、希伯来语等从右到左书写的语言。这使得 LVGL 能够应用在面向全球用户的产品中，不同国家和地区的用户都能通过本地化界面获得良好的体验。

7. 可扩展性

LVGL 提供了灵活的主题和样式系统，开发者可以方便地定制控件的外观样式，以实现不同风格的 UI。通过更换主题，可以快速切换界面的整体视觉风格。此外，LVGL 还可以与第三方图形库、文件系统和图像解码器结合使用，从而扩展其功能。例如，可以使用文件系统来加载外部字体和图片，或者集成 JPEG/PNG 解码库来显示复杂图像。

应用场景

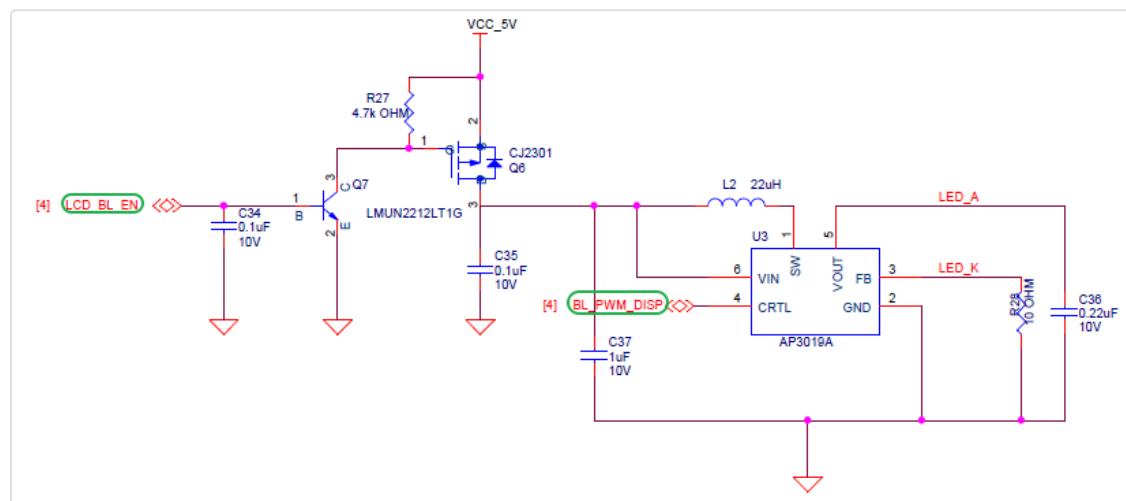
LVGL 在实际应用中有着非常广泛的覆盖范围。在消费电子领域，它常用于智能家居控制面板、家电显示屏、智能手表和健身设备等，这些产品往往需要在有限硬件资源下提供友好的用户交互界面。在工业控制领域，LVGL 被用于 HMI（人机交互界面）和各种仪器仪表，帮助用户直观地监控和操作设备。在汽车电子中，LVGL 可以驱动中控屏、副驾娱乐屏，甚至车载仪表盘。而在医疗设备方面，它则适合做小型显示界面，例如手持检测仪、便携式监护设备等。

生态与社区

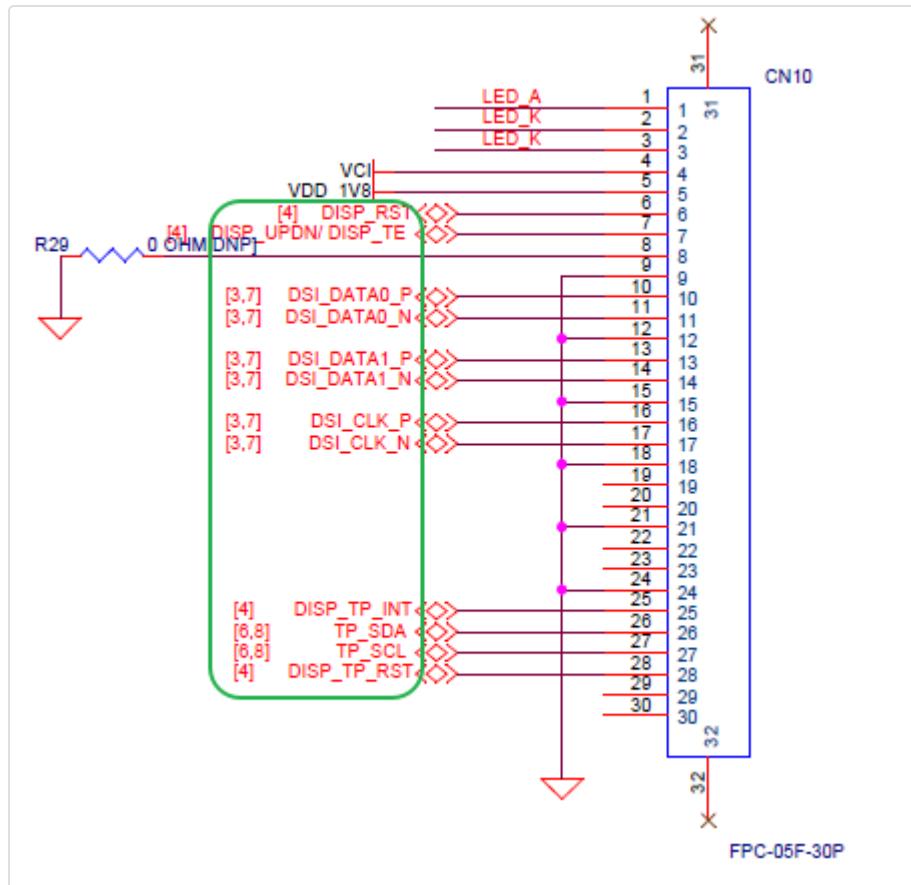
LVGL 作为一个开源项目，采用 **MIT License** 开源协议，既适合个人学习，也可以在商业产品中免费使用。围绕 LVGL，已经形成了一个活跃的生态系统。官方提供了 **SquareLine Studio** 这类可视化设计工具，支持拖拽式设计界面并导出 LVGL 代码，极大地提升了开发效率。此外，**LVGL Simulator** 可以让开发者直接在 PC 上进行界面调试，不需要每次都烧录到目标硬件上。全球开发者社区十分活跃，贡献了大量开源控件、主题和移植案例，为新手和企业开发者提供了丰富的资源和支持。

硬件说明

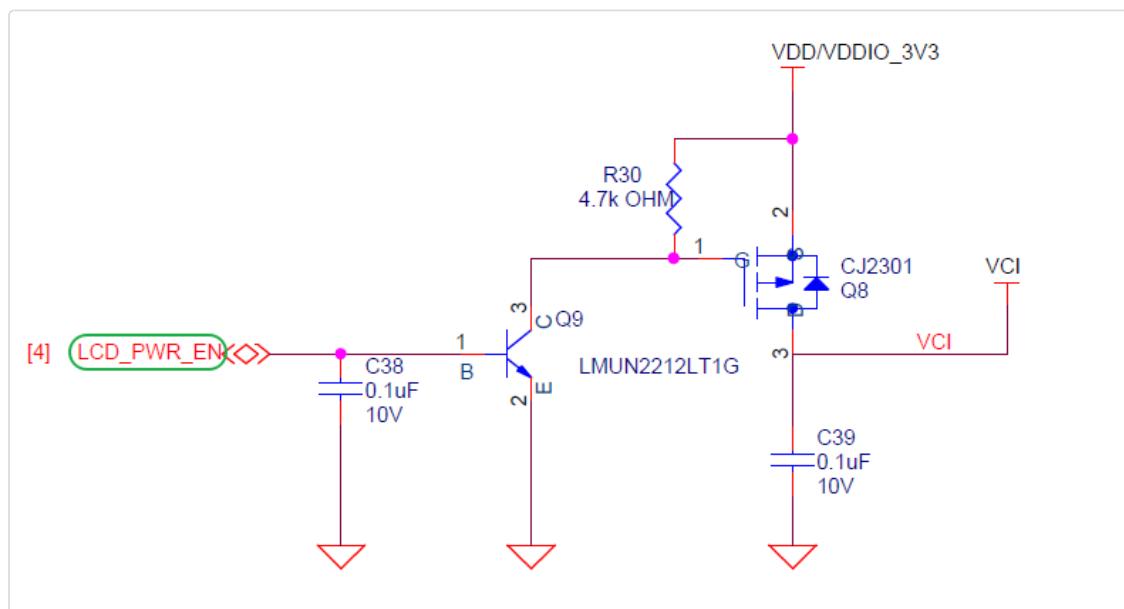
背光接口



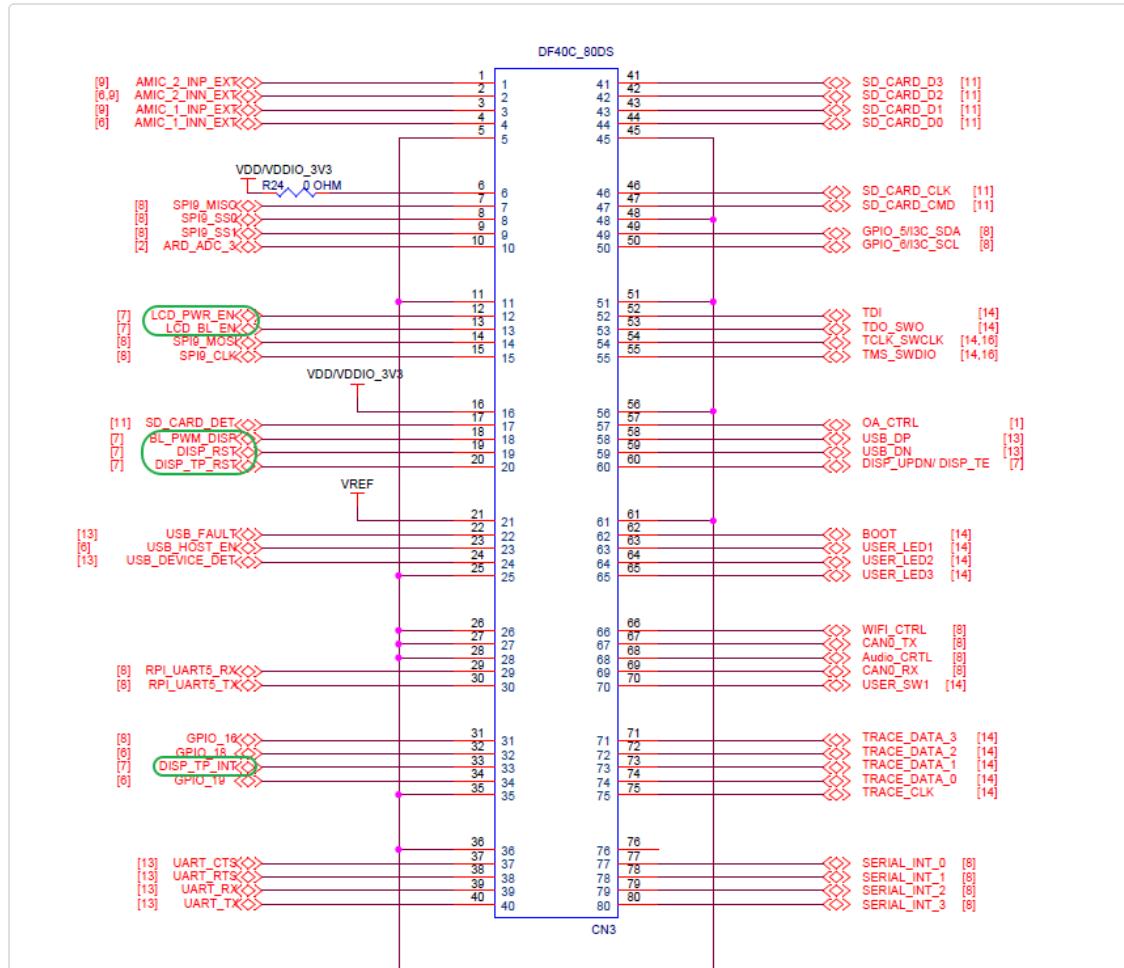
MIPI接口

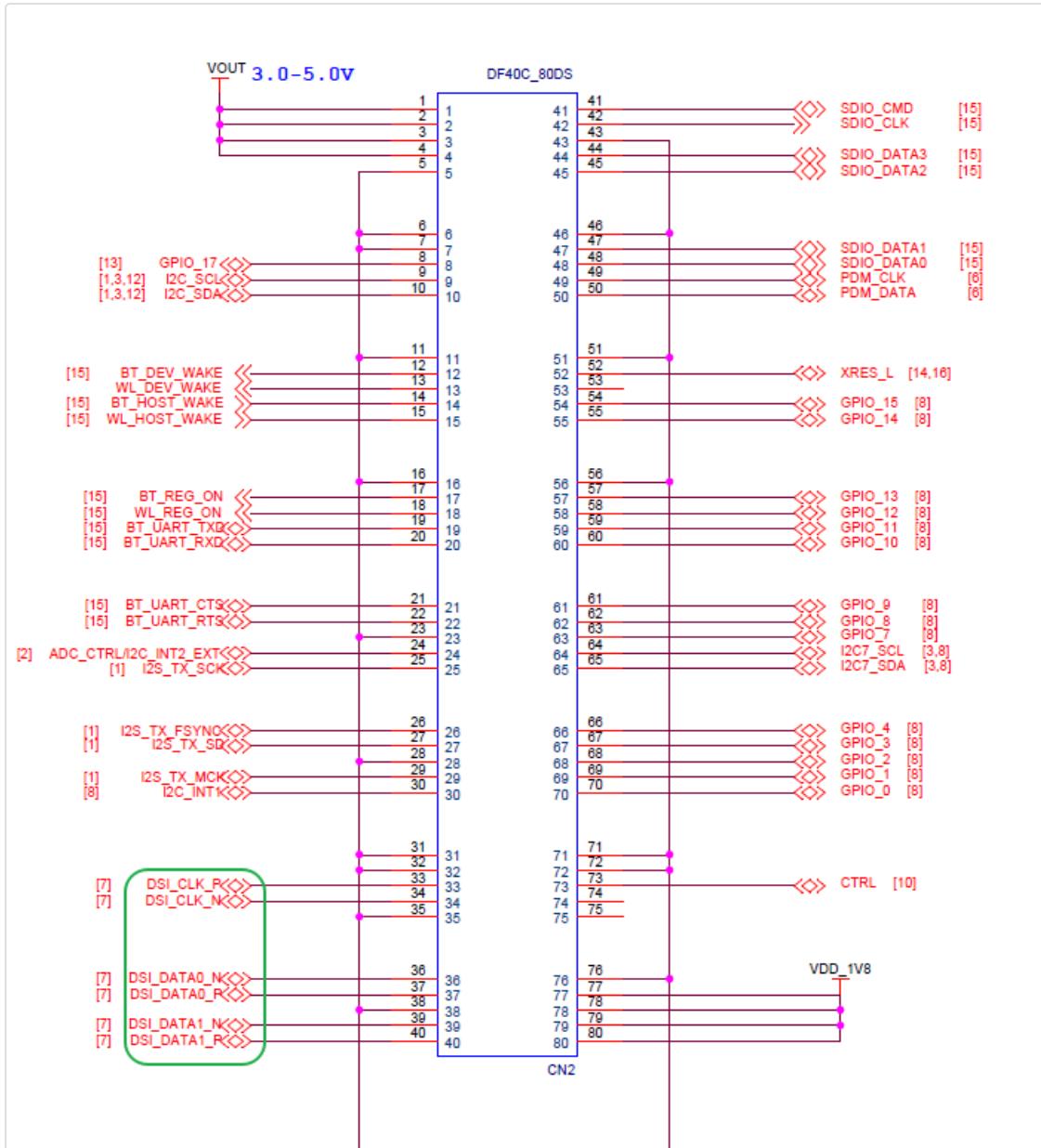


PWR接口

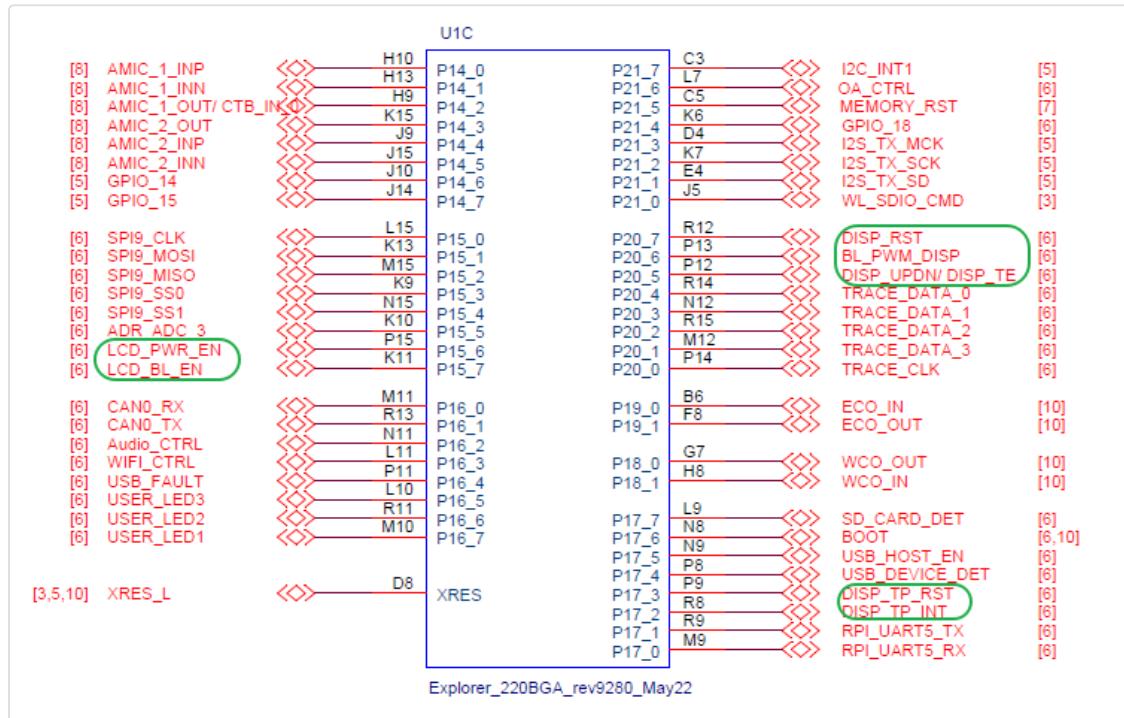
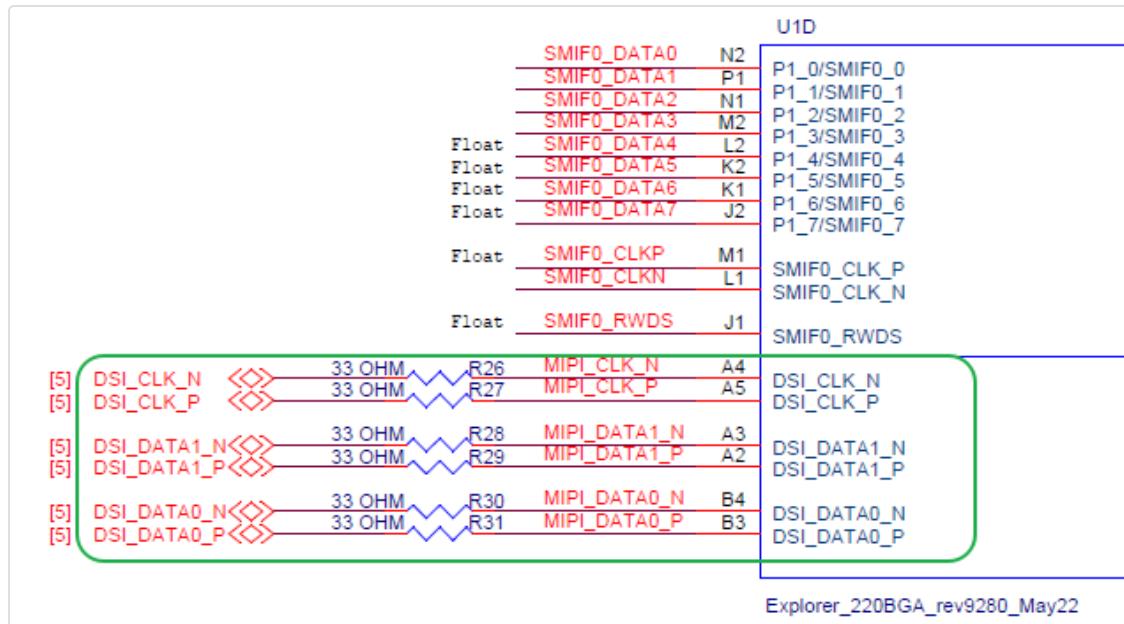


BTB座子





MCU接口



软件说明

- 工程基于 Edgi-Talk 平台开发，运行在 M55 应用核上。
- 示例功能包括：
- 初始化 LVGL 图形库

- 启动 `lv_demo_stress` 示例
- 在 LCD 上运行性能测试与渲染演示
- 工程结构简洁，便于理解 显示驱动接口 和 LVGL 移植流程。

使用方法

编译与下载

1. 打开工程并完成编译。
2. 使用 **板载下载器 (DAP)** 将开发板的 USB 接口连接至 PC。
3. 通过编程工具将生成的固件烧录至开发板。

运行效果

- 烧录完成后，开发板上电即可运行示例工程。
- 系统会自动启动 `lv_demo_stress`，在 LCD 上显示 LVGL 的性能测试画面。
- 用户可通过修改 `applications/main.c` 中的示例入口函数，切换至其它 LVGL demo（如 `lv_demo_widgets`、`lv_demo_music`）。

注意事项

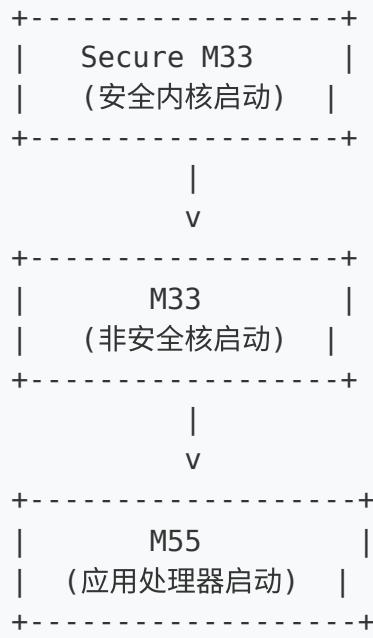
- 如需修改工程的 **图形化配置**，请使用以下工具打开配置文件：

```
tools/device-configurator/device-configurator.exe  
libs/TARGET_APP_KIT_PSE84_EVAL_EPC2/config/design.modus
```

- 修改完成后保存配置，并重新生成代码。
- 若显示屏幕无输出，请检查：
 - LCD 硬件连接与电源供给是否正常
 - `lv_port_disp.c` 和 `lv_port_indev.c` 的配置是否与实际硬件匹配

启动流程

系统启动顺序如下：



⚠ 请严格按照以上顺序烧写固件，否则系统可能无法正常运行。

- 若示例工程无法正常运行，建议先编译并烧录 **Edgi-Talk_M33_S_Template** 工程，确保初始化与核心启动流程正常，再运行本示例。
- 若要开启 M55，需要在 **M33 工程** 中打开配置：

```
RT-Thread Settings --> 硬件 --> select SOC Multi Core Mode --> E
```

3.3. Edgi-Talk_WavPlayer 示例工程

中文 | English

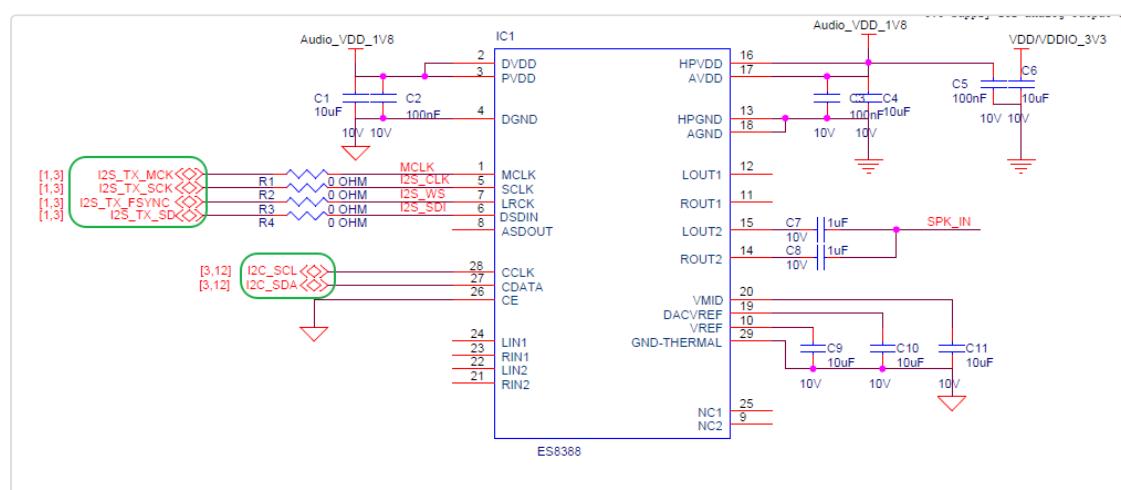
简介

本示例工程基于 Edgi-Talk 平台，演示 WAV 音频播放功能，运行在 RT-Thread 实时操作系统 (M33 核) 上。

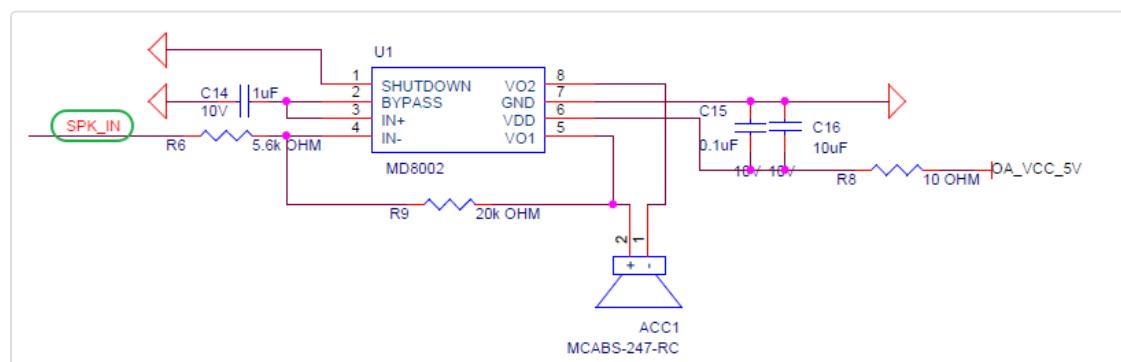
通过本工程，用户可以快速体验 WAV 音频文件的播放机制，并验证音频解码和驱动接口，为后续音频应用开发提供参考。

硬件说明

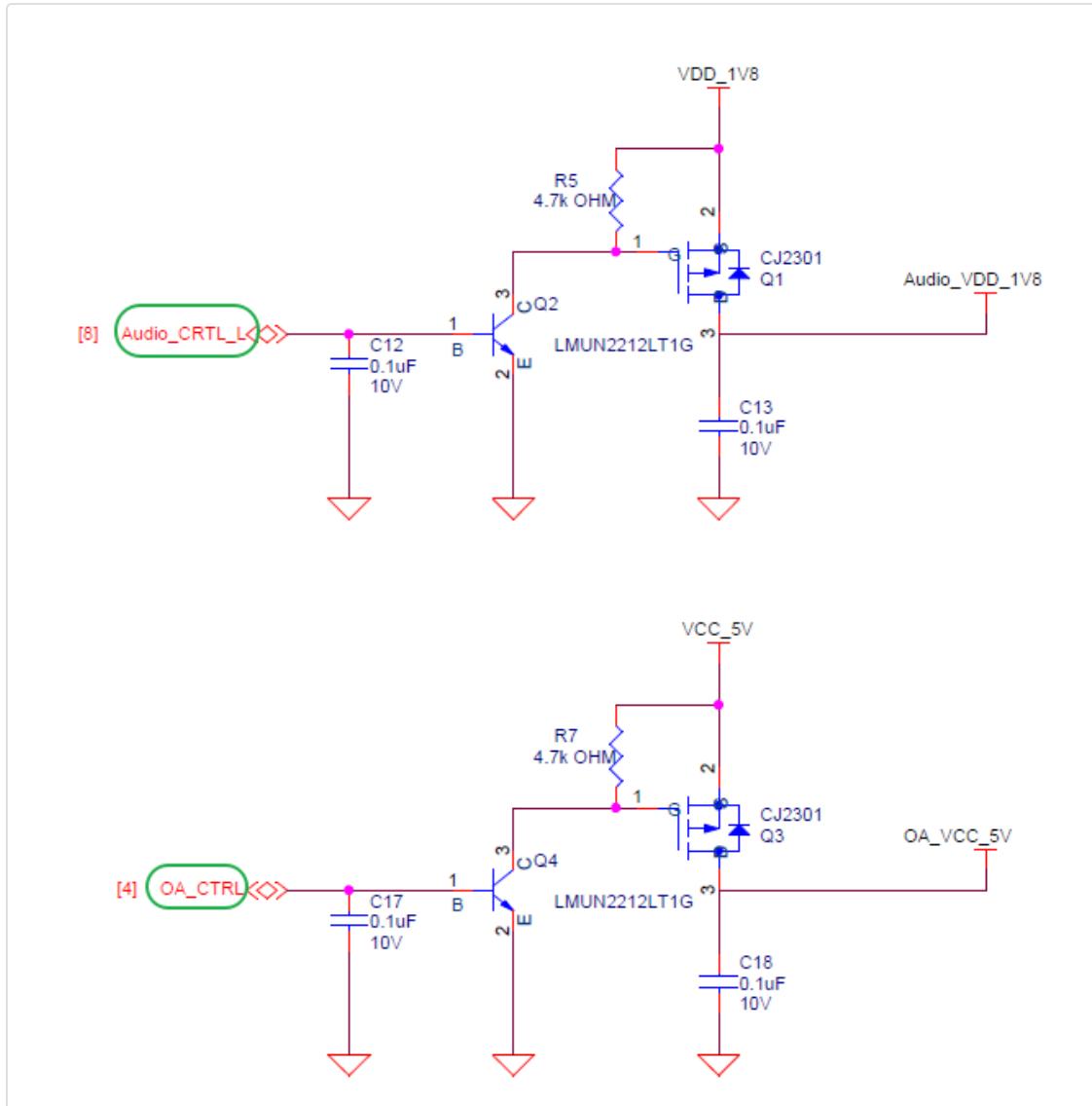
ES8388连接接口



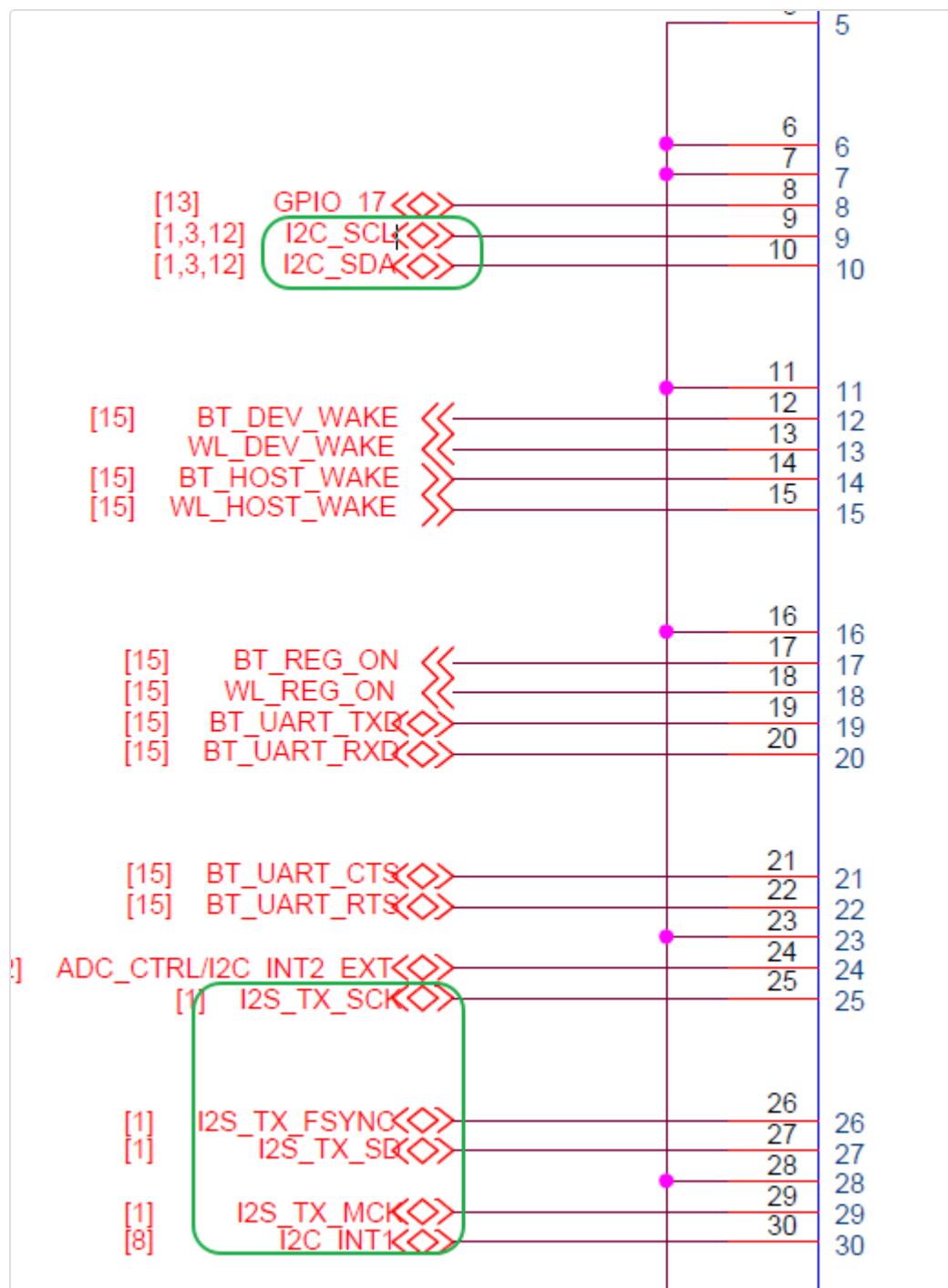
喇叭接口



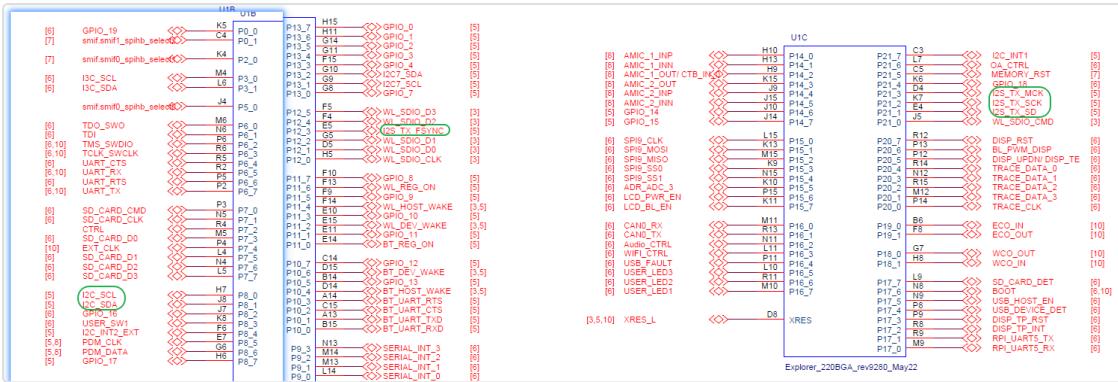
控制引脚



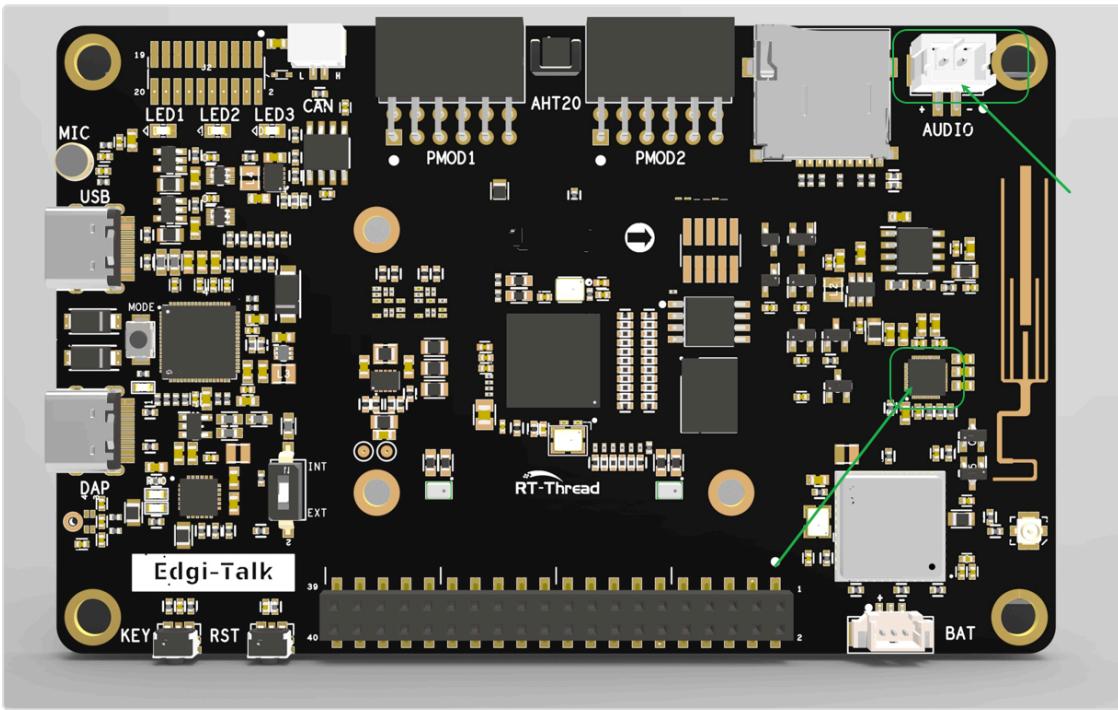
BTB座子



MCU接口



实物图位置



软件说明

- 工程基于 **Edgi-Talk** 平台开发。
- 示例功能包括：
- WAV 文件解析与播放
- 音频数据通过板载 DAC 或音频外设输出
- 支持 PCM16 格式 WAV 文件

- 支持 **16 kHz、24 kHz、48 kHz** 采样率
- 支持 **双声道** 输出
- 串口打印播放状态信息
- 工程结构清晰，便于理解音频播放驱动和 RT-Thread 文件系统的配合使用。

使用方法

编译与下载

1. 打开工程并完成编译。
2. 使用 **板载下载器 (DAP)** 将开发板的 USB 接口连接至 PC。
3. 通过编程工具将生成的固件烧录至开发板。
4. 将 WAV 音频文件拷贝至 SD 卡或外部存储设备的根目录，例如 **16000.wav**。

运行效果

- 烧录完成后，开发板上电即可运行示例工程。
- 系统会自动初始化 I2C、I2S 音频设备，并挂载存储设备。
- 用户可在 **串口终端** 使用以下命令启动播放：

```
wavplay -s 16000.wav
```

- 串口输出示例：

```
\ | /
- RT -      Thread Operating System
/ | \      5.0.2 build Sep  8 2025 11:21:16
2006 - 2022 Copyright by RT-Thread team
[I/I2C] I2C bus [i2c0] registered
[I/i2s] ES8388 init success.
[I/drv.mic] audio pdm registered.
[I/drv.mic] !!!Note: pdm depends on i2s0, they share clock.
found part[0], begin: 1048576, size: 29.739GB
Hello RT-Thread
This core is cortex-m33
msh />wavplay -s 16000.wav
[D/WAV_PLAYER] EVENT:PLAYSTOPPAUSERESUME, STATE:STOPPED -> PLAY
```

```
[D/WAV_PLAYER] open wavplayer, device sound0
[D/WAV_PLAYER] Information:
[D/WAV_PLAYER] samplerate 16000
[D/WAV_PLAYER] channels 2
[D/WAV_PLAYER] sample bits width 16
[I/WAV_PLAYER] play start, uri=16000.wav
[I/i2s] Ready for I2S output
msh />
```

- 播放期间，串口将显示 WAV 文件采样率、声道数、位宽及播放状态信息。

注意事项

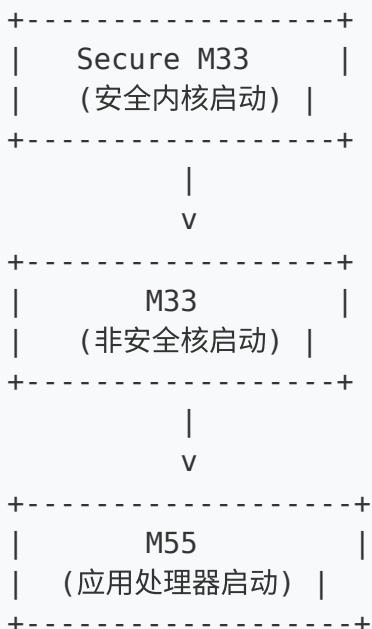
- WAV 文件需为 **PCM16 格式**，采样率可选择 **16 kHz、24 kHz 或 48 kHz**，输出为 **双声道**。
- 如需修改工程的 **图形化配置**，请使用以下工具打开配置文件：

```
tools/device-configurator/device-configurator.exe
libs/TARGET_APP_KIT_PSE84_EVAL_EPC2/config/design.modus
```

- 修改完成后保存配置，并重新生成代码。
- 请确保存储设备正确插入且挂载成功，否则无法播放音频文件。

启动流程

系统启动顺序如下：



 请严格按照以上顺序烧写固件，否则系统可能无法正常运行。

- 若示例工程无法正常运行，建议先编译并烧录 **Edgi-Talk_M33_S_Template** 工程，确保初始化与核心启动流程正常，再运行本示例。
- 若要开启 M55，需要在 **M33 工程** 中打开配置：

```
RT-Thread Settings --> 硬件 --> select SOC Multi Core Mode --> E
```

3.4. 小智示例工程

[中文](#) | [English](#)

简介

本示例工程基于 **Edgi-Talk** 平台，演示 小智语音交互设备的基本功能，运行在 **RT-Thread** 实时操作系统 上。

通过本工程，用户可以快速验证设备 WiFi 连接、按键唤醒 与 语音交互 功能，为后续应用开发提供基础参考。

软件说明

- 工程基于 **Edgi-Talk** 平台开发。
- 示例功能包括：
 - WiFi 连接与状态显示
 - 按键唤醒与语音交互
 - 设备状态管理（待机、监听、休眠等）

使用方法

WIFI修改

1. 在 `main.c` 36行中找到以下代码：

```
while (rt_wlan_connect("TEST", "88888888"));
```

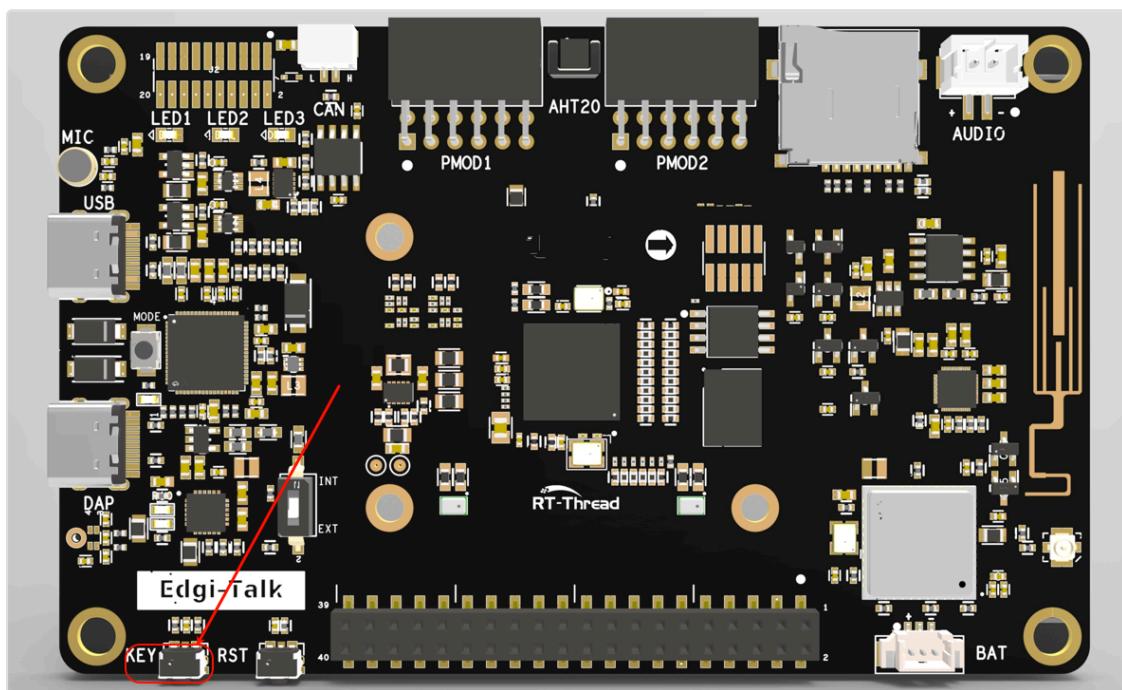
2. 将 "TEST" 改为你的 WiFi 名称，"88888888" 改为密码，重新编译并烧录。
3. WIFI 详细使用参考：[WIFI](#)

编译与下载

1. 打开工程并完成编译。
2. 使用 **板载下载器 (DAP)** 将开发板的 USB 接口连接至 PC。
3. 通过编程工具将生成的固件烧录至设备。

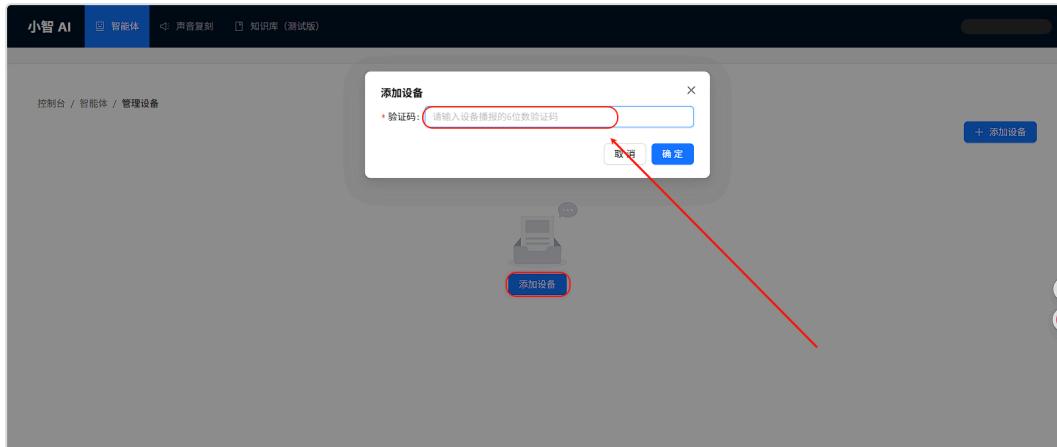
运行效果

- 烧录完成后，设备上电即可运行示例工程。
- 屏幕会显示当前状态，包括：
 - **Connecting**: 正在连接 WiFi
 - **On standby**: 待机中
 - **Listening**: 监听中，可与设备对话
 - **Sleeping**: 休眠状态
- 按下顶部按键，可进入 **Listening** 状态进行语音交互。



注意事项

- 第一次需要进入 [小智官网](#) 进行后台绑定



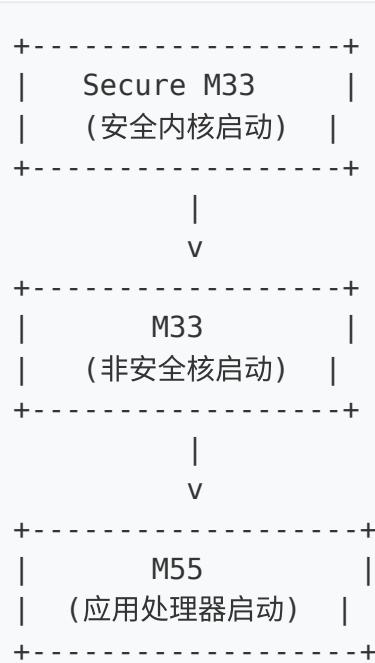
- 请确保 WiFi 名称与密码正确，并使用 **2.4GHz 频段**。
- 设备需在可访问互联网的环境下使用。
- 如需修改工程的 **图形化配置**，请使用以下工具：

```
tools/device-configurator/device-configurator.exe  
libs/TARGET_APP_KIT_PSE84_EVAL_EPC2/config/design.modus
```

- 修改完成后保存配置，并重新生成代码。

启动流程

系统启动顺序如下：



⚠ 请严格按照以上顺序烧写固件，否则设备可能无法正常运行。

- 若示例工程无法正常运行，建议先编译并烧录 **Edgi-Talk_M33_S_Template** 工程，确保初始化与核心启动流程正常，再运行本示例。
- 若要开启 M55，需要在 **M33 工程** 中打开配置：

```
RT-Thread Settings --> 硬件 --> select SOC Multi Core Mode --> E
```