psoc_e84-edgi-talk SDK

Documentation

The Edgi-Talk board is based on the PSoC™ E84 MCU and provides a flexible and comprehensive development platform.It integrates a variety of peripheral interfaces and sample modules, helping developers quickly implement applications involving multi-sensor integration, display, and communication.

Version 1.0.0 | English | October 27, 2025

www.rt-thread.org

Contents

1. Basics

1.1. Edgi-Talk_M33_Blink_LED Example Project

中文 | **English**

Introduction

This example project is based on the **Edgi-Talk platform** and demonstrates the **blue LED blinking** function running on the **RT-Thread real-time operating system**.
Through this project, users can quickly verify the board-level GPIO configuration and LED control logic, providing a fundamental reference for future hardware control and application development.

GPIO Overview

GPIO (General Purpose Input/Output) is one of the most commonly used peripheral interfaces in MCUs.
It can be configured in software as either **input mode** or **output mode**:

- **Input mode**: Used to read external voltage levels, such as button input.
- **Output mode**: Used to control peripheral signals, such as lighting an LED or driving a buzzer.

RT-Thread GPIO Abstraction

RT-Thread provides a **PIN device driver framework**, which abstracts hardware differences through a unified API interface:

- `rt_pin_mode(pin, mode)`: Set the pin mode (input/output/pull-up/pull-down, etc.)

- `rt_pin_write(pin, value)`: Output a voltage level (high/low)
- `rt_pin_read(pin)`: Read the input voltage level

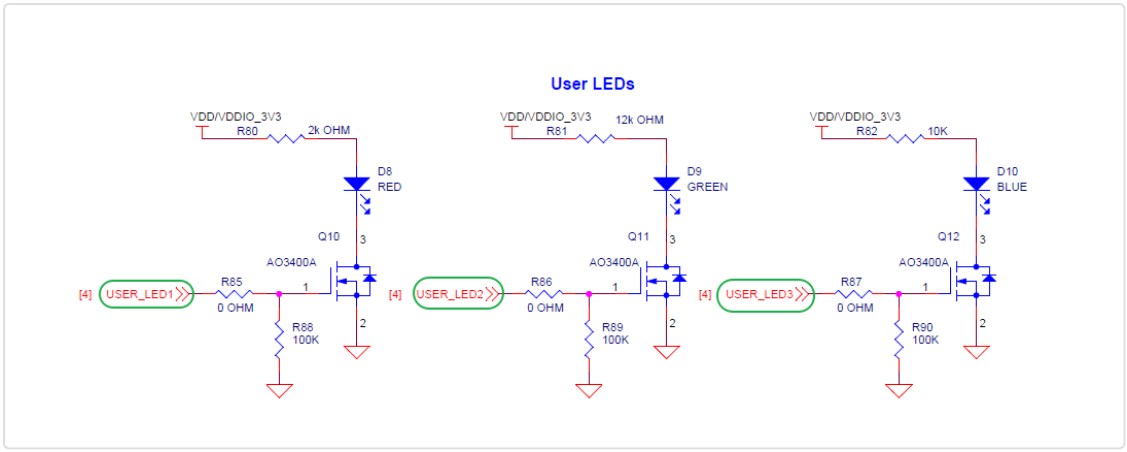
This allows developers to perform GPIO control without directly manipulating registers, using RT-Thread's API instead.

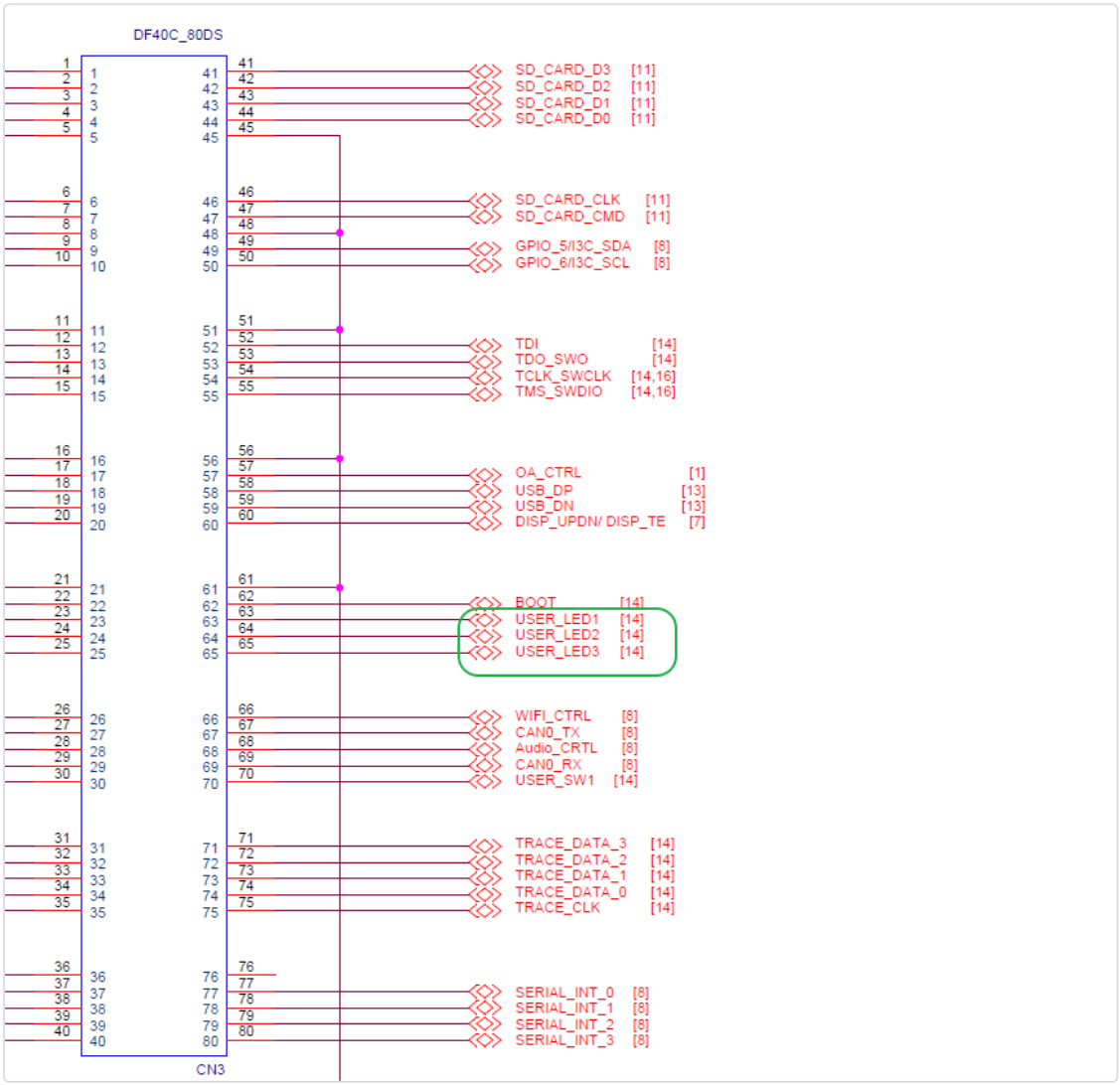
In this example, the LED pin is configured as **output mode**, and software toggles the output level in a loop to make the LED blink.

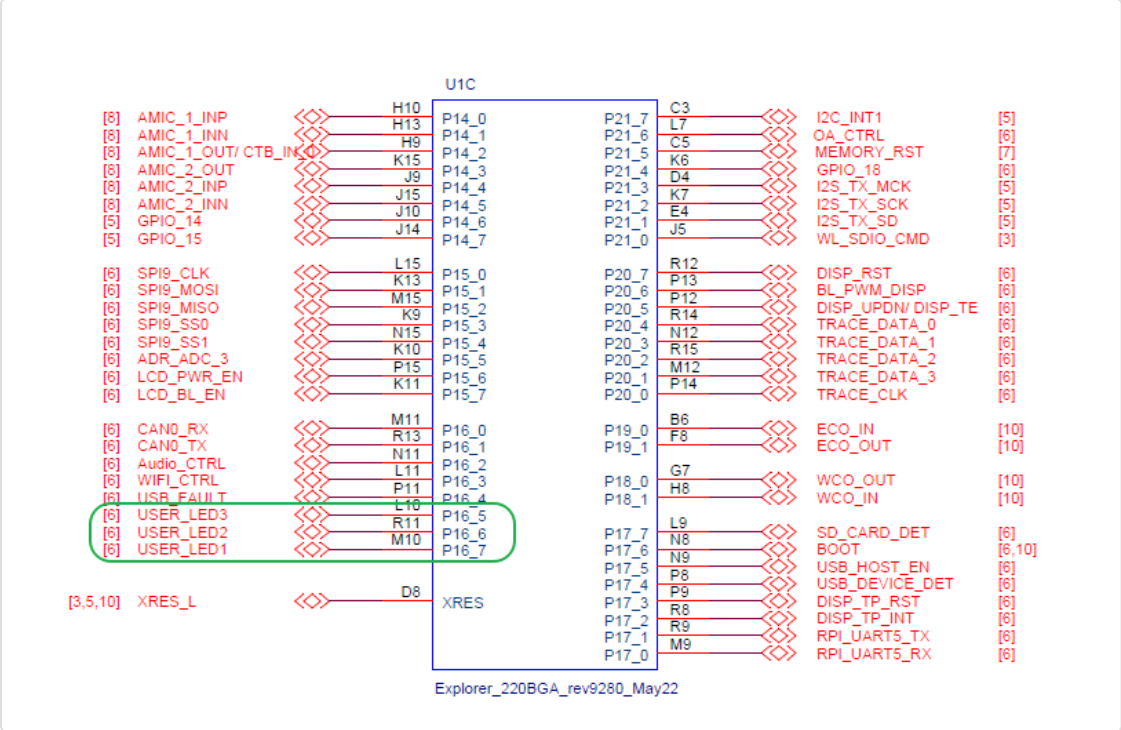
Software Description

- The project is developed based on the **Edgi-Talk** platform.
- Example functionalities include:
- Blue LED blinking periodically
- GPIO output control
- The project structure is simple and easy to understand, helping users grasp LED control logic and hardware driver interfaces.

Hardware Description

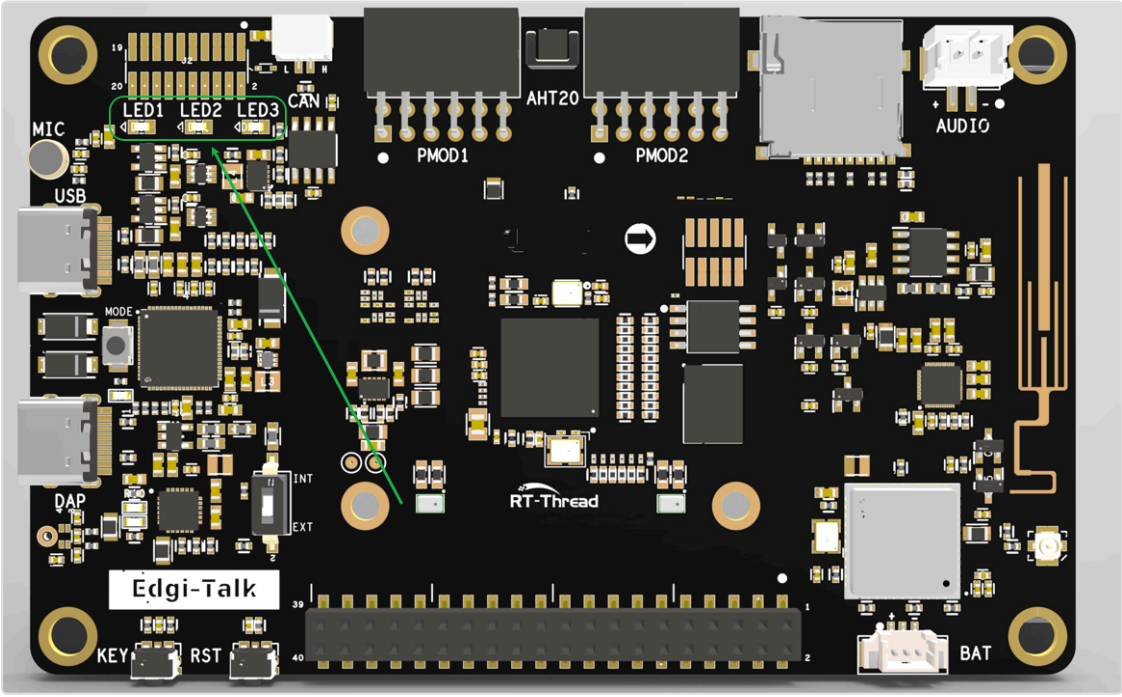






As shown above, the Edgi-Talk board provides three user LEDs: USER_LED1 (RED), USER_LED2 (GREEN), and USER_LED3 (BLUE).
USER_LED2 corresponds to pin **P16_6**.
When the MCU outputs a **high level**, the LED turns **on**; when the MCU outputs a **low level**, the LED turns **off**.

The LED location on the board is shown below:



Usage Instructions

Compilation and Download

1. Open the project and complete the compilation.
2. Connect the board's USB port to the PC using the **onboard debugger (DAP)**.
3. Use the programming tool to flash the generated firmware to the development board.

Runtime Behavior

- After flashing, power on the board to run the example project.
- The **blue LED blinks every 500 ms**, indicating normal GPIO operation and system scheduling.
- Users can modify the blinking interval or LED control logic as needed.

Notes

- To modify the **graphical configuration** of the project, open the configuration file using the following tool:

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

- After modification, save the configuration and regenerate the code.

Boot Sequence

The system boot sequence is as follows:

```
+------------------+
|   Secure M33     |
|   (Secure Core)  |
+------------------+
          |
          v
+------------------+
|       M33        |
```

```
| (Non-Secure Core)|
+------------------+
          |
          v
+-------------------+
|       M55         |
| (Application Core)|
+-------------------+
```

⚠️ Please strictly follow the boot sequence above when flashing firmware; otherwise, the system may not run properly.

- If the example project does not run correctly, compile and flash the **Edgi-Talk_M33_S_Template** project first to ensure proper initialization and core startup before running this example.
- To enable the M55 core, configure the **M33 project** as follows:

```
  RT-Thread Settings --> Hardware --> select SOC Multi Core Mod
```

1.2. Edgi-Talk_M55_Blink_LED Example Project

中文 | **English**

Introduction

This example project is based on the **Edgi-Talk platform** and demonstrates the **green LED blinking** function running on the **RT-Thread real-time operating system**.
Through this project, users can quickly verify the GPIO configuration and LED control logic of the board, providing a basic reference for subsequent hardware control and application development.

GPIO Overview

GPIO (General Purpose Input/Output) is one of the most commonly used peripheral interfaces of MCUs. It can be configured via software as **input mode** or **output mode**:

- **Input Mode**: Used to read external voltage levels, such as button inputs.
- **Output Mode**: Used to control peripheral voltage levels, such as lighting an LED or driving a buzzer.

GPIO Abstraction in RT-Thread

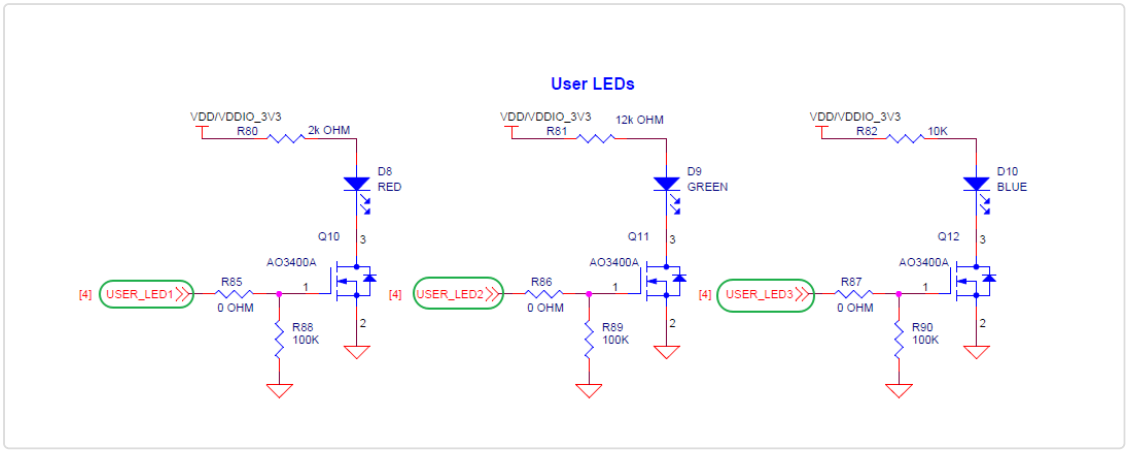
RT-Thread provides a **PIN device driver framework**, which abstracts hardware differences through a unified interface:

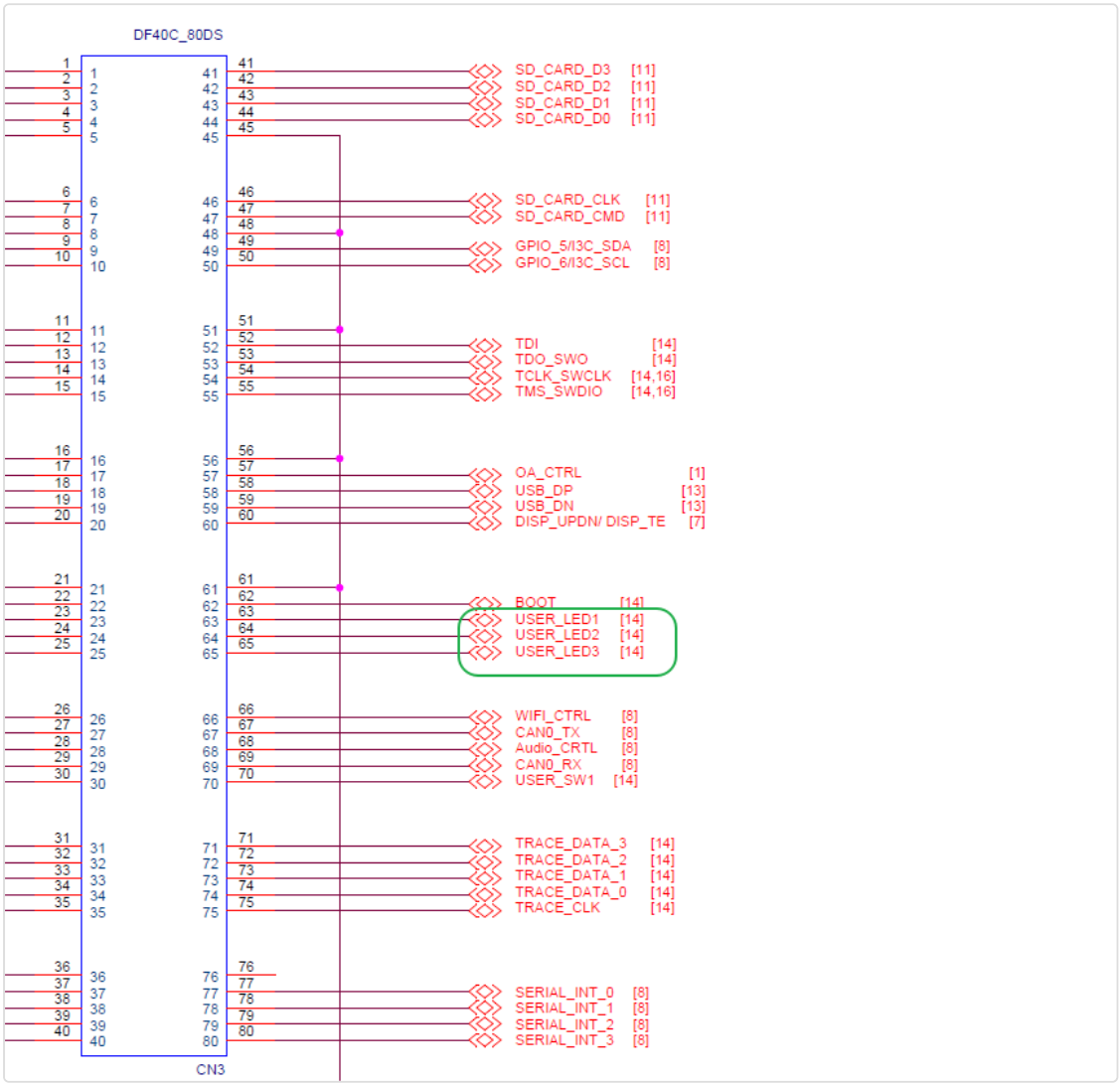
- `rt_pin_mode(pin, mode)`: Set the pin mode (input/output/pull-up/pull-down, etc.)
- `rt_pin_write(pin, value)`: Output a voltage level (high/low)
- `rt_pin_read(pin)`: Read input level

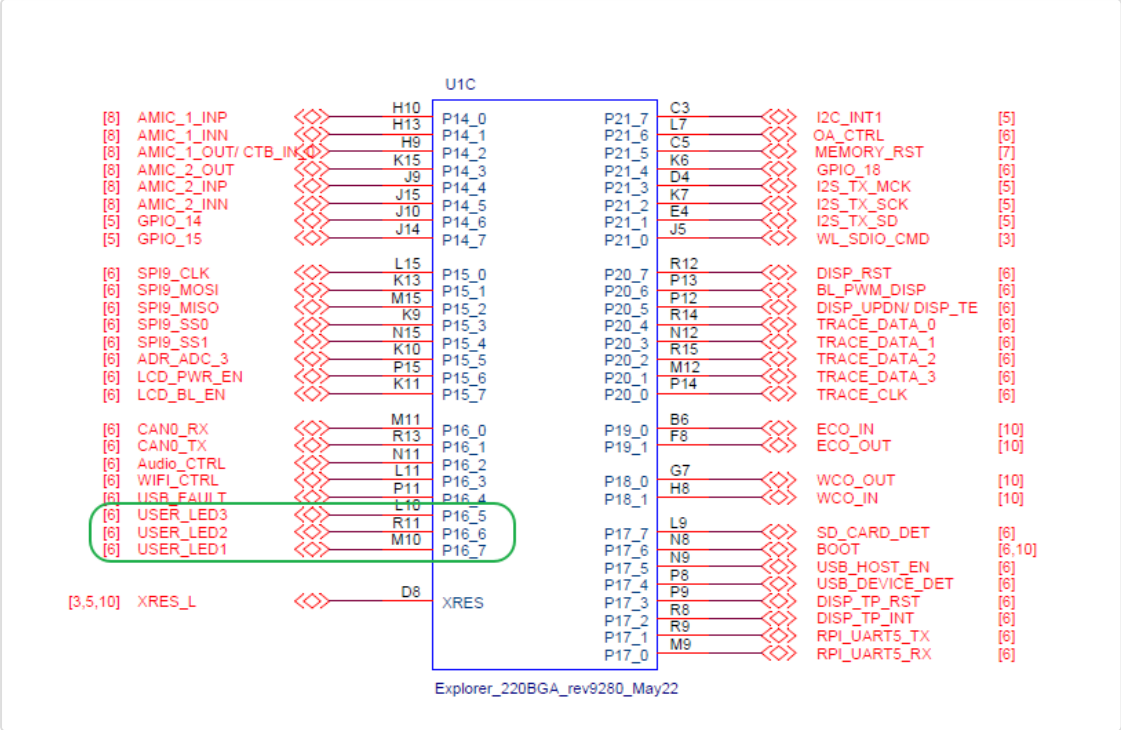
This allows developers to perform GPIO control using RT-Thread APIs without directly manipulating hardware registers.

In this example, the LED pin is configured as **output mode**. The software toggles the output between high and low levels in a loop to achieve LED blinking.

Hardware Description

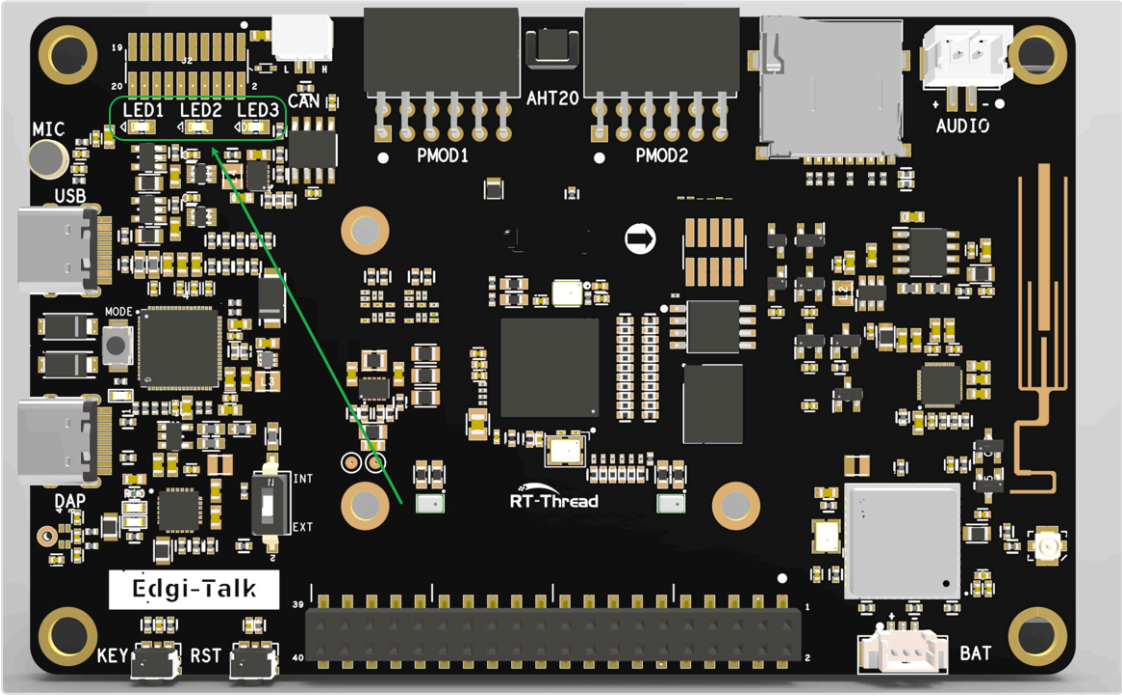






As shown above, the Edgi-Talk board provides three user LEDs: USER_LED1 (RED), USER_LED2 (GREEN), and USER_LED3 (BLUE).
Among them, USER_LED2 corresponds to pin P16_7.
When the MCU outputs a **high level**, the LED lights up; when it outputs a **low level**, the LED turns off.

The position of the LEDs on the development board is shown below:



Software Description

- The project is developed based on the **Edgi-Talk** platform.
- Example features include:
- Periodic blinking of the green LED
- GPIO output control
- The project structure is simple and helps users understand LED control logic and hardware driver interfaces.

Usage

Build and Download

1. Open the project and compile it.
2. Connect the board’s USB interface to the PC using the **onboard debugger (DAP)**.
3. Use the programming tool to flash the generated firmware to the development board.

Running Result

- After flashing, power on the board to run the example project.
- The **green LED blinks every 500 ms**, indicating that GPIO control and system scheduling are functioning correctly.
- Users can modify the blinking period or LED control logic as needed.

Notes

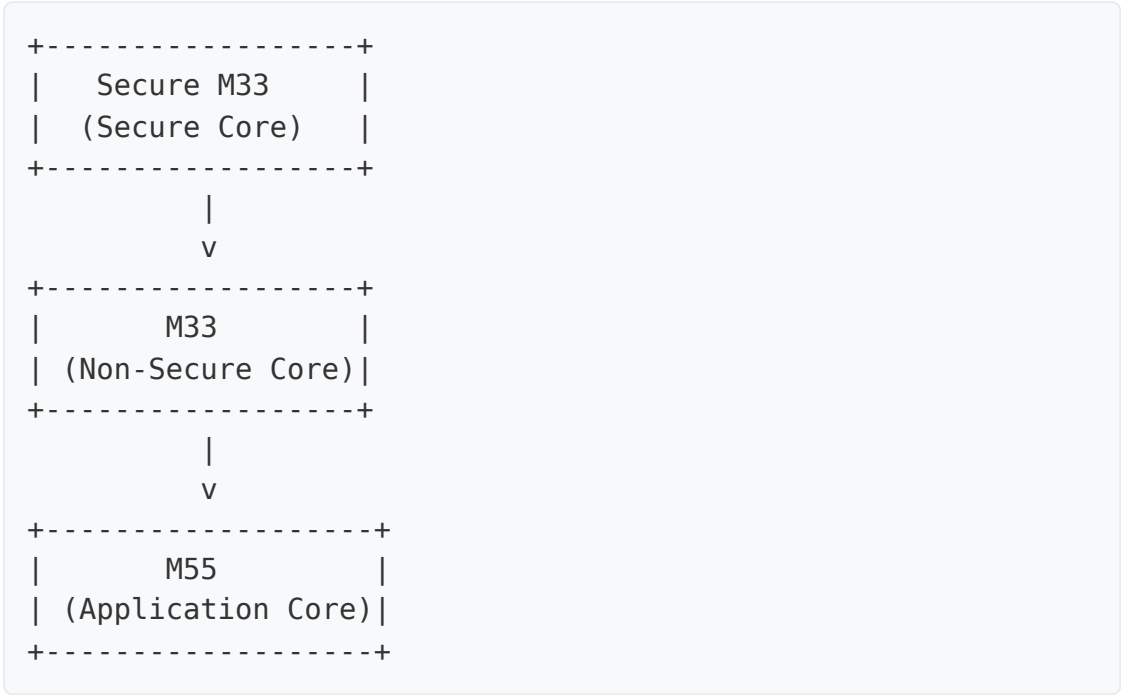
- To modify the **graphical configuration** of the project, open the configuration file using the following tool:

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

- After modification, save the configuration and regenerate the code.

Startup Sequence

The system starts in the following order:



⚠️ Please follow the above flashing order strictly; otherwise, the system may fail to run properly.

- If the example does not run correctly, first compile and flash the **Edgi-Talk_M33_S_Template** project to ensure that initialization and core startup are working properly before running this example.
- To enable the M55 core, enable the following configuration in the **M33 project**:

```
RT-Thread Settings --> Hardware --> select SOC Multi Core Mod
```

1.3. Edgi-Talk_M33_S_Template Example Project

中文 | **English**

Introduction

This example project is based on a **bare-metal architecture** and demonstrates the configuration and usage of the **Secure M33 core**.
It can also serve as a **template** for further development or project creation, helping users quickly get started and extend functionalities.

Software Description

- Developed on the **Edgi-Talk platform**.
- Example features include:
- **Secure region configuration**
- **Basic startup flow demonstration**
- The project code is structured clearly, making it easy to understand and port.

Usage

Build and Download

1. Open and compile the project.
2. Connect the board's USB interface to your PC using the **onboard debugger (DAP)**.
3. Flash the compiled firmware to the board using your programming tool.

Running Result

- After flashing and powering on, the board will start the system normally.
- It will successfully boot into the **M33 core**, indicating that the secure configuration is effective.

Notes

- To modify the **graphical configuration**, use the following tools:

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

- Save changes and regenerate code after editing.

Startup Sequence

```
+------------------+
|   Secure M33     |
|  (Secure Core)   |
+------------------+
          |
          v
+------------------+
|       M33        |
| (Non-Secure Core)|
+------------------+
          |
          v
+--------------------+
|       M55          |
| (Application Core)|
+--------------------+
```

⚠️ Follow this flashing order strictly; otherwise, the system may not operate correctly.

1.4. Edgi-Talk_M33_Template Example Project

中文 | **English**

Introduction

This example project runs on the **M33 core** with **RT-Thread Real-Time Operating System**.
It allows users to quickly experience RT-Thread running on the M33 platform.

After flashing and powering on the board, the **blue LED** will blink periodically, indicating that the system is running normally.
This project can also serve as a **template** for further development or project creation, helping users quickly get started and extend functionalities.

Software Description

- Developed on the **Edgi-Talk platform**.
- Uses **RT-Thread** as the OS kernel.
- Example features:
- System initialization
- LED task (blinking)
- The project structure is clear, making it a good starting point for learning RT-Thread or developing applications.

Usage

Build and Download

1. Open and compile the project.
2. Connect the board's USB interface to your PC using the **onboard debugger (DAP)**.
3. Flash the compiled firmware to the board.

- During flashing, the following tool will be automatically invoked to merge the signed firmware:

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

- By default, `proj_cm33_s_signed.hex` in the directory will be merged and flashed to the target device.

Running Result

- After flashing and powering on, the board will run the example project.
- The **blue LED** will blink periodically, indicating that the RT-Thread scheduler has started successfully.

Notes

- To modify the **graphical configuration**, use the following tools:

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

- Save changes and regenerate code after editing.

Startup Sequence

```
+------------------+
|   Secure M33     |
|  (Secure Core)   |
+------------------+
          |
          v
+------------------+
|       M33        |
| (Non-Secure Core)|
+------------------+
          |
          v
+------------------+
|       M55        |
| (Application Core)|
+------------------+
```

⚠️ Follow this flashing order strictly; otherwise, the system may not operate correctly.

- To enable M55, open the configuration in RT-Thread Settings:

```
Hardware --> select SOC Multi Core Mode --> Enable CM55 Core
```

2. Drivers

2.1. Edgi-Talk_ADC Example Project

中文 | **English**

Introduction

This example project is based on the **Edgi-Talk platform**, running on the **RT-Thread real-time operating system**, and demonstrates how to use the **ADC (Analog-to-Digital Converter)**.
Through this project, users can quickly experience ADC data acquisition and processing, providing a reference for developing analog signal acquisition applications.
During runtime, the blue indicator LED blinks periodically, indicating that the system has started and is running properly.

1. Overview of ADC

ADC (Analog-to-Digital Converter) is a device or module that converts continuous analog signals into discrete digital signals, serving as a core component in modern digital control, signal processing, and measurement systems.

- **Function**: Converts continuous signals such as voltage or current into digital values for processing by a microcontroller (MCU), DSP, or FPGA.
- **Key Specifications**:
- **Resolution**: The number of bits in the ADC output, representing the number of distinguishable levels. Edgi uses **12-bit** resolution, i.e., 2^12 = 4096 levels.
- **Sampling Rate**: The number of samples the ADC takes per second, determining the range of detectable signal frequencies.
- **Input Range**: The range of analog voltages the ADC can handle.

- **Accuracy**: Indicates how closely the ADC output matches the actual input signal, affected by noise, nonlinearity, and offset errors.

2. ADC Working Principle

ADC operation typically involves the following stages:

1. **Sampling and Holding (Sample & Hold, S/H)**

- Captures the analog signal at a specific moment and holds it steady to ensure stability during conversion.

2. **Quantization**

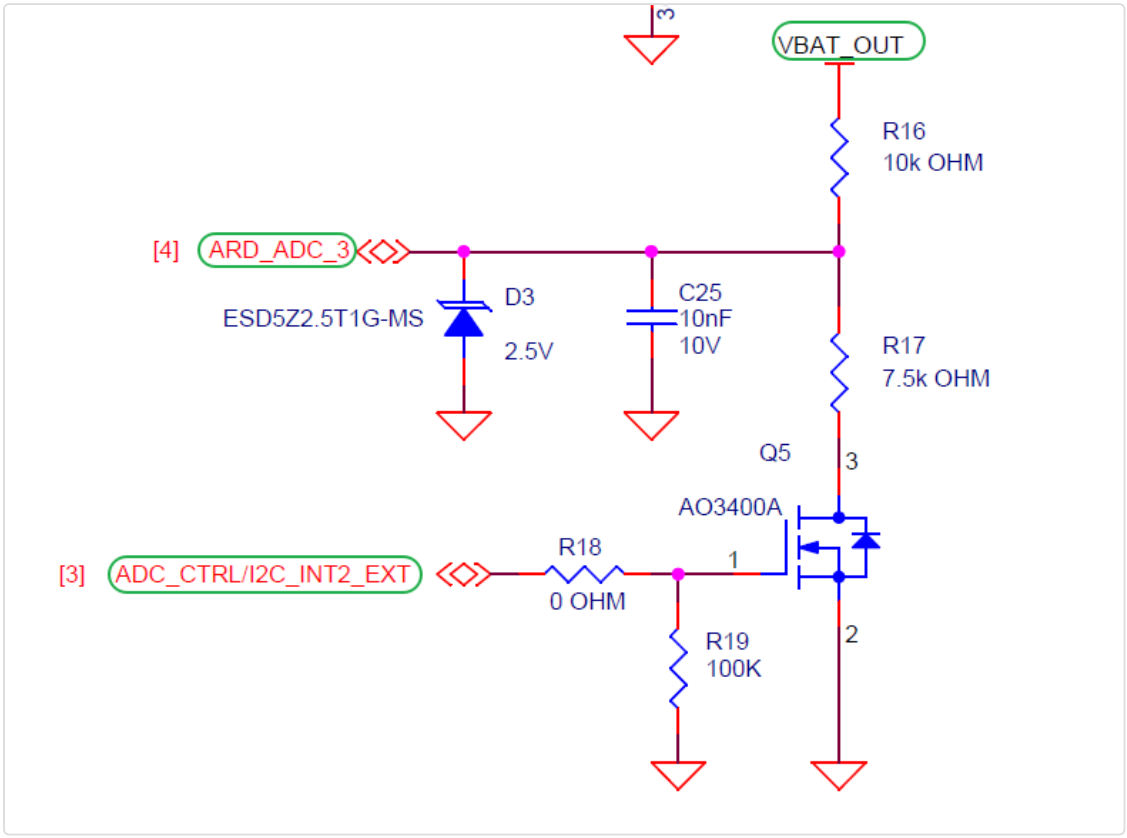
- Divides the analog signal into discrete levels, each corresponding to a digital code.
- A 12-bit ADC divides the input voltage range into 4096 levels, with quantization precision expressed as ΔV = VREF / 4096.

3. **Encoding**

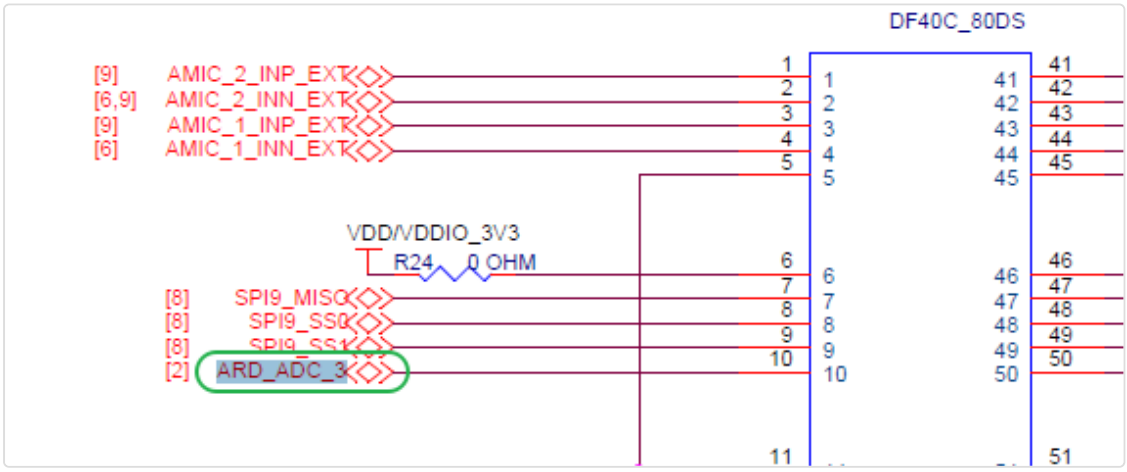
- Converts the quantized level into binary code for output.

Hardware Description

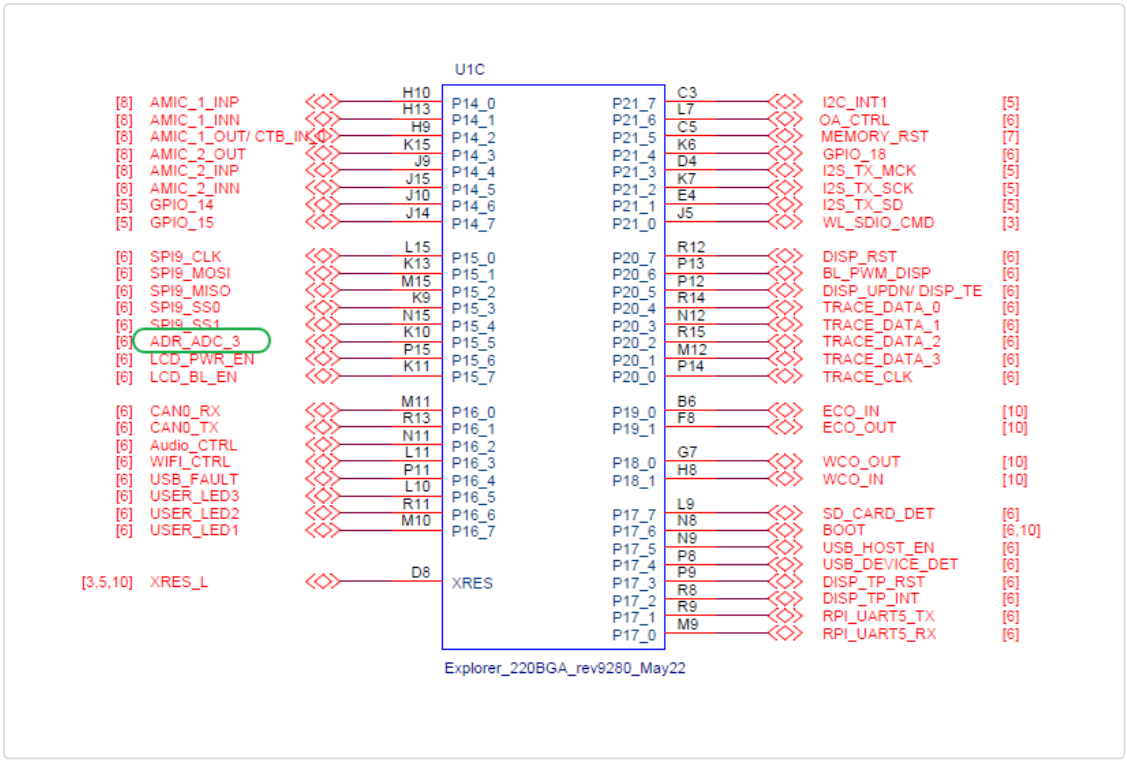
Connection Interface



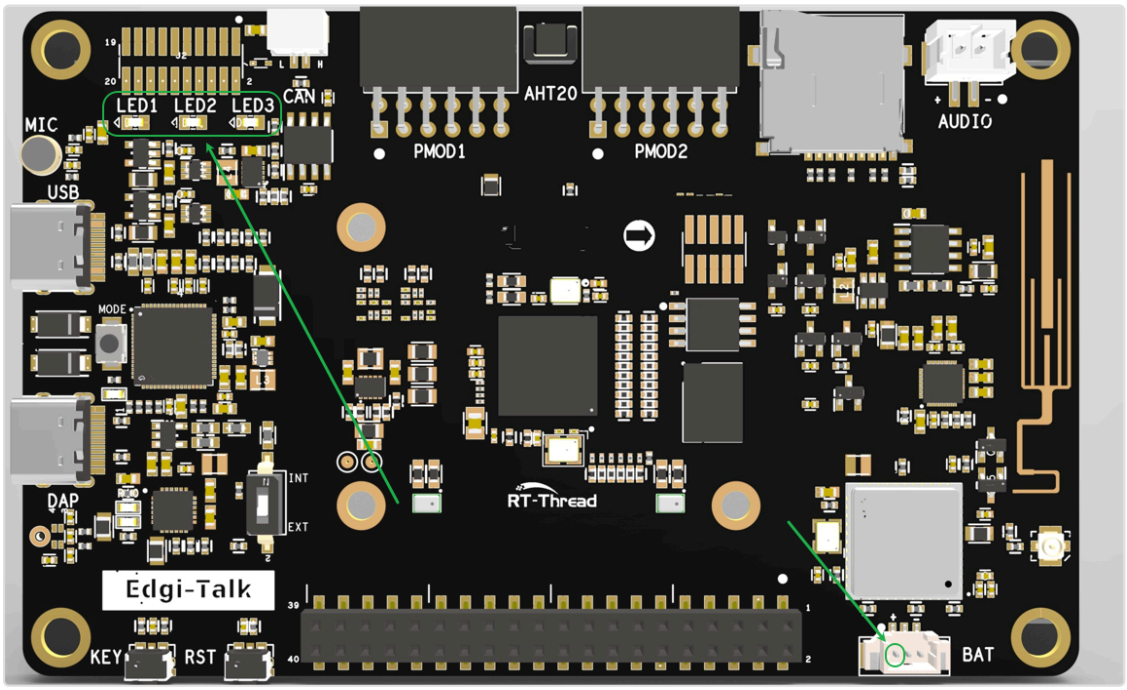
BTB Connector



MCU Pins



Physical Board Location



Software Description

- The project is developed based on the **Edgi-Talk** platform.
- Uses **RT-Thread** as the operating system kernel.
- Example features:
- ADC initialization and sampling
- LED indicator blinking
- ADC sampling results printed via serial port
- The project has a clear structure, making it easy for users to understand the ADC driver and RT-Thread threading mechanism.

Usage Instructions

Compilation and Download

1. Open the project and complete the compilation.
2. Connect the board’s USB port to the PC using the **onboard debugger (DAP)**.
3. Use the programming tool to flash the generated firmware onto the board.

Runtime Behavior

- After flashing, power on the board to run the example.
- The **blue indicator LED** blinks every 500 ms, indicating normal system operation.
- ADC samples battery voltage and prints results to the serial port as shown below:

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

Notes

- To modify the **graphical configuration** of the project, open the configuration file using the following tool:

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

- After making changes, save the configuration and regenerate the code.

Boot Sequence

The system boot sequence is as follows:

```
+------------------+
|   Secure M33     |
|   (Secure Core)  |
+------------------+
          |
          v
+------------------+
|       M33        |
| (Non-Secure Core)|
+------------------+
          |
          v
+-------------------+
|       M55         |
| (Application Core)|
+-------------------+
```

⚠️ Please strictly follow the boot sequence above when flashing firmware, or the system may fail to start properly.

- If the example project does not run correctly, compile and flash the **Edgi-Talk_M33_S_Template** project first to ensure proper initialization and core startup sequence before running this example.
- To enable the M55 core, configure the **M33 project** as follows:

```
RT-Thread Settings --> Hardware --> select SOC Multi Core Mod
```

2.2. Edgi-Talk_M33_AHT20 Example Project

中文 | English

Introduction

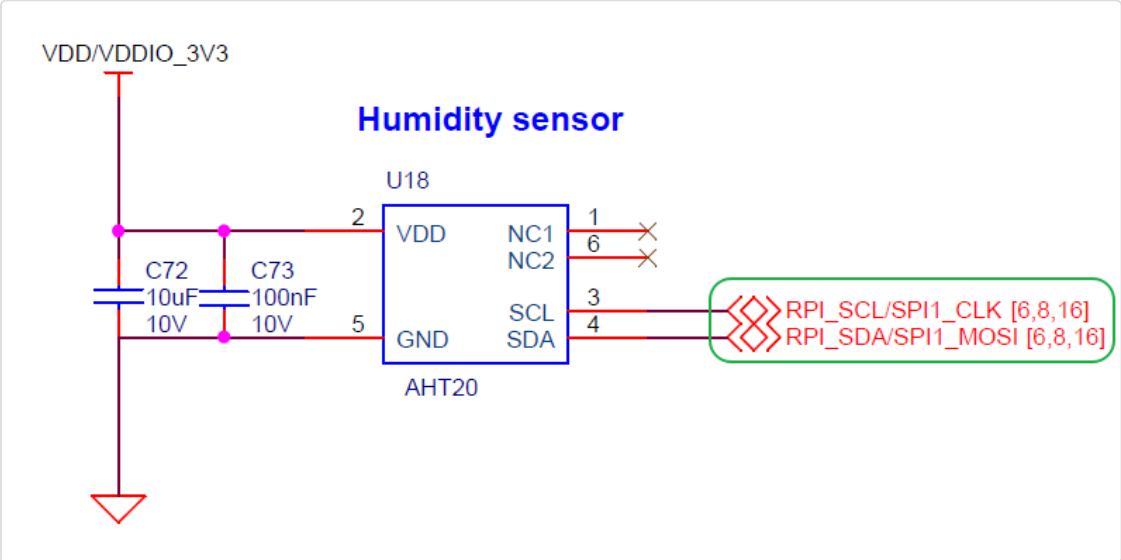
This example project is based on the **Edgi-Talk platform** and demonstrates how to drive and use the **AHT20 temperature and humidity sensor**.
Through this project, users can quickly experience AHT20 data acquisition and processing, and view the sampled results via the serial port on the development board.

AHT10 Software Package Overview

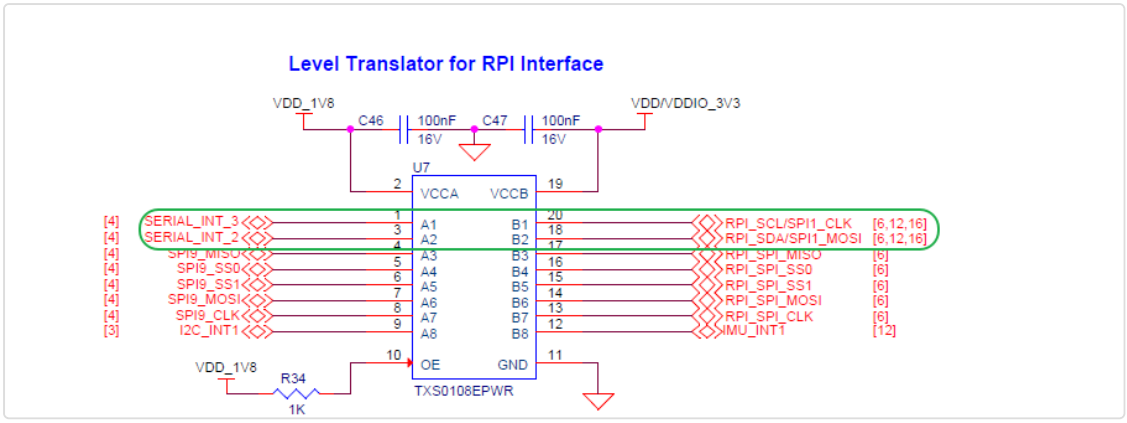
The AHT10 software package provides basic functions for using the AHT10 temperature and humidity sensor, and also includes an optional software-based moving average filter.
For more details, please refer to the README file in the AHT10 software package.

Hardware Description

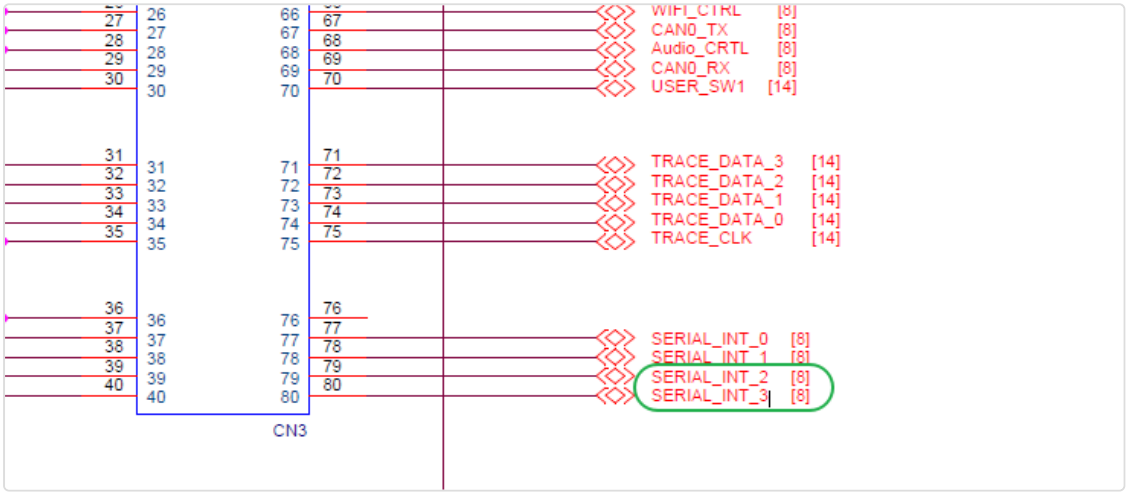
Sensor Connection Interface



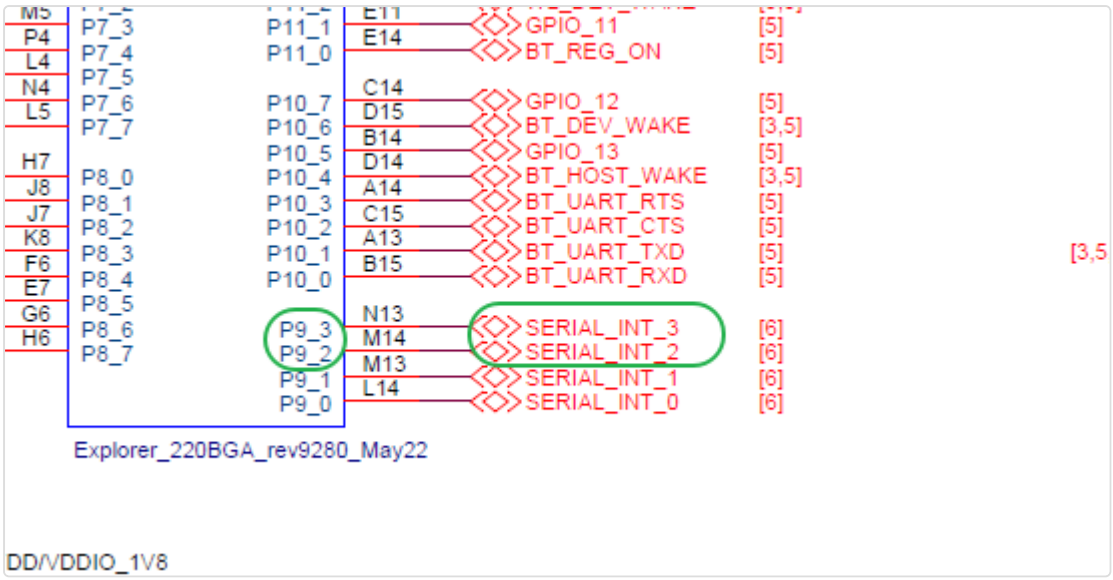
Level Shifting



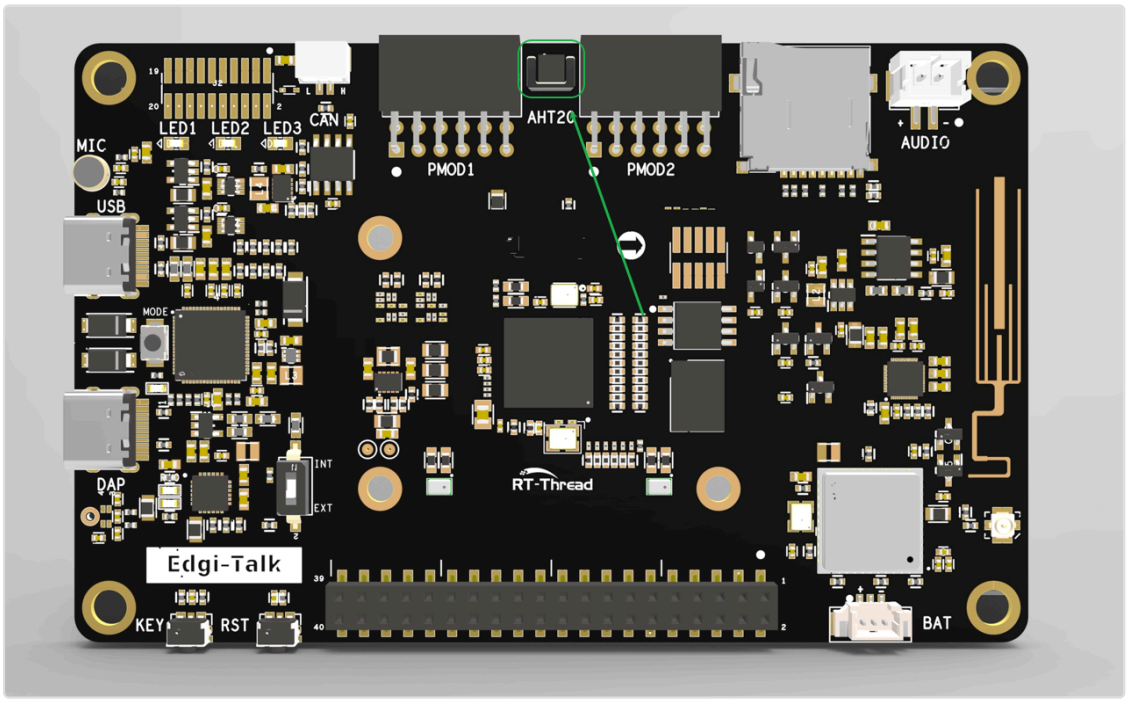
BTB Connector



MCU Pins



Physical Board Location



Software Description

- The project is developed based on the **Edgi-Talk** platform.
- Example functionalities include:

- AHT20 initialization and communication via I²C
- Temperature and humidity data acquisition and parsing
- Displaying sampled data via serial output
- The project has a clear structure, helping users understand I²C driver usage and sensor interfacing.

Usage Instructions

Compilation and Download

1. Open the project and complete the compilation.
2. Connect the board's USB port to the PC using the **onboard debugger (DAP)**.
3. Use the programming tool to flash the generated firmware onto the development board.

Runtime Behavior

- After flashing, power on the board to run the example project.
- The system will initialize the AHT20 sensor and begin sampling temperature and humidity data.
- The sampled data will be printed via the serial terminal, as shown below:

```
 \ | /
- 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
```

Notes

- To modify the **graphical configuration** of the project, open the configuration file using the following tool:

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

- After editing, save the configuration and regenerate the code.

Boot Sequence

The system boot sequence is as follows:

```
+------------------+
|   Secure M33     |
|   (Secure Core)  |
+------------------+
          |
          v
+------------------+
|       M33        |
| (Non-Secure Core)|
+------------------+
          |
          v
+-------------------+
|       M55         |
| (Application Core)|
+-------------------+
```

⚠️ Please strictly follow the boot sequence above when flashing firmware; otherwise, the system may fail to start properly.

- If the example project does not run correctly, compile and flash the **Edgi-Talk_M33_S_Template** project first to ensure proper initialization and core startup sequence before running this example.
- To enable the M55 core, configure the **M33 project** as follows:

```
  RT-Thread Settings --> Hardware --> select SOC Multi Core Mod
```

2.3. Edgi-Talk_Audio Example Project

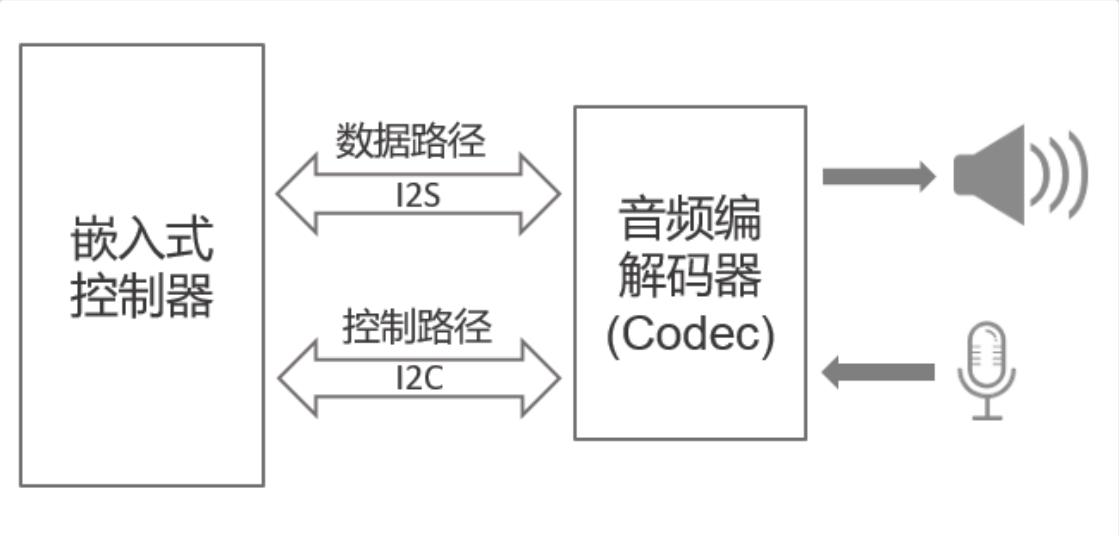
中文 | **English**

Introduction

This example project is based on the **Edgi-Talk platform** and demonstrates **audio recording and playback** features running on the **RT-Thread real-time operating system**.
Through this project, users can experience microphone audio data acquisition and speaker playback. The playback can be controlled using a button, while the LED indicator reflects the current playback state.

Audio Overview

Audio devices are an essential component of embedded systems, responsible for audio data sampling and output.
An audio system typically consists of a data bus interface, control bus interface, audio codec (Codec), speaker, and microphone, as shown below:



Audio Device Features

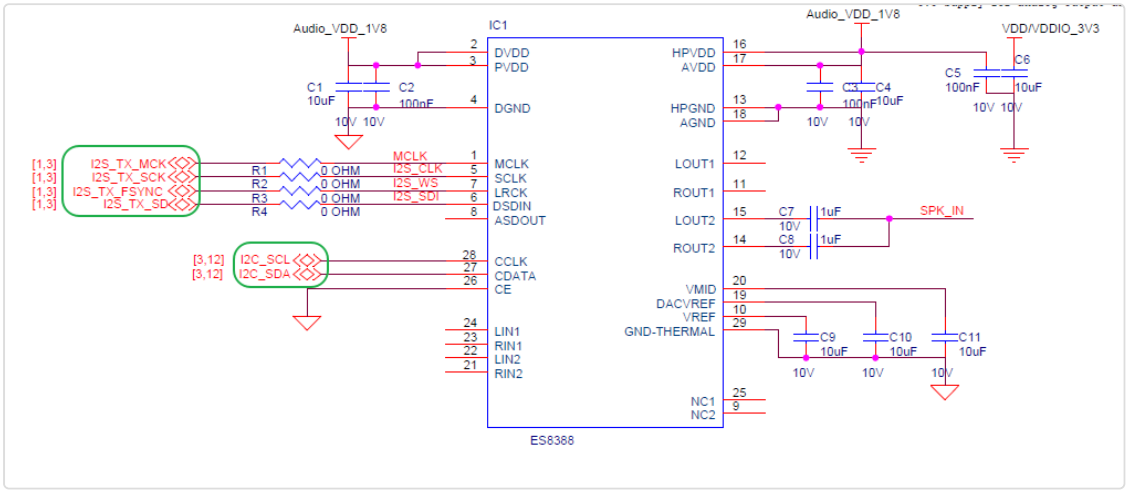
The RT-Thread Audio device driver framework forms the low-level foundation of the Audio subsystem. It manages raw audio data input/output, stream control, device management, volume control, and hardware/codec abstraction.

- Interface: Standard device interface (open/close/read/control)

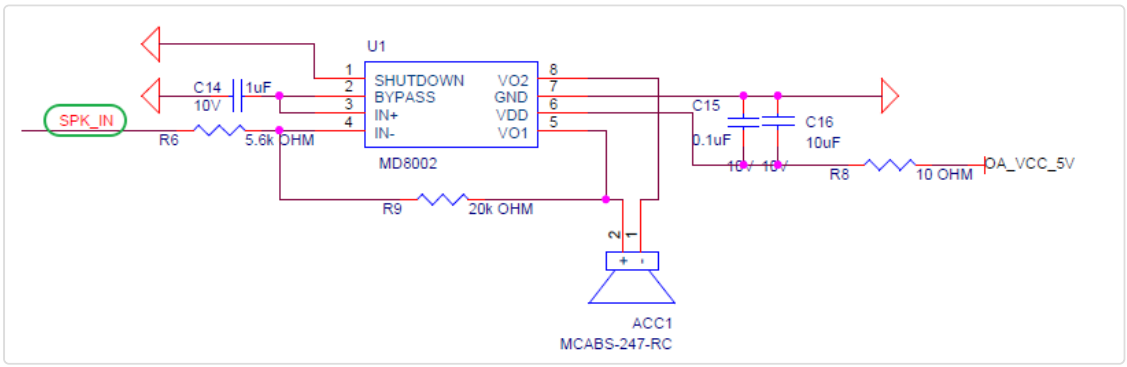
- Supports synchronous access
- Supports both playback and recording
- Supports audio parameter management
- Supports volume control

Hardware Description

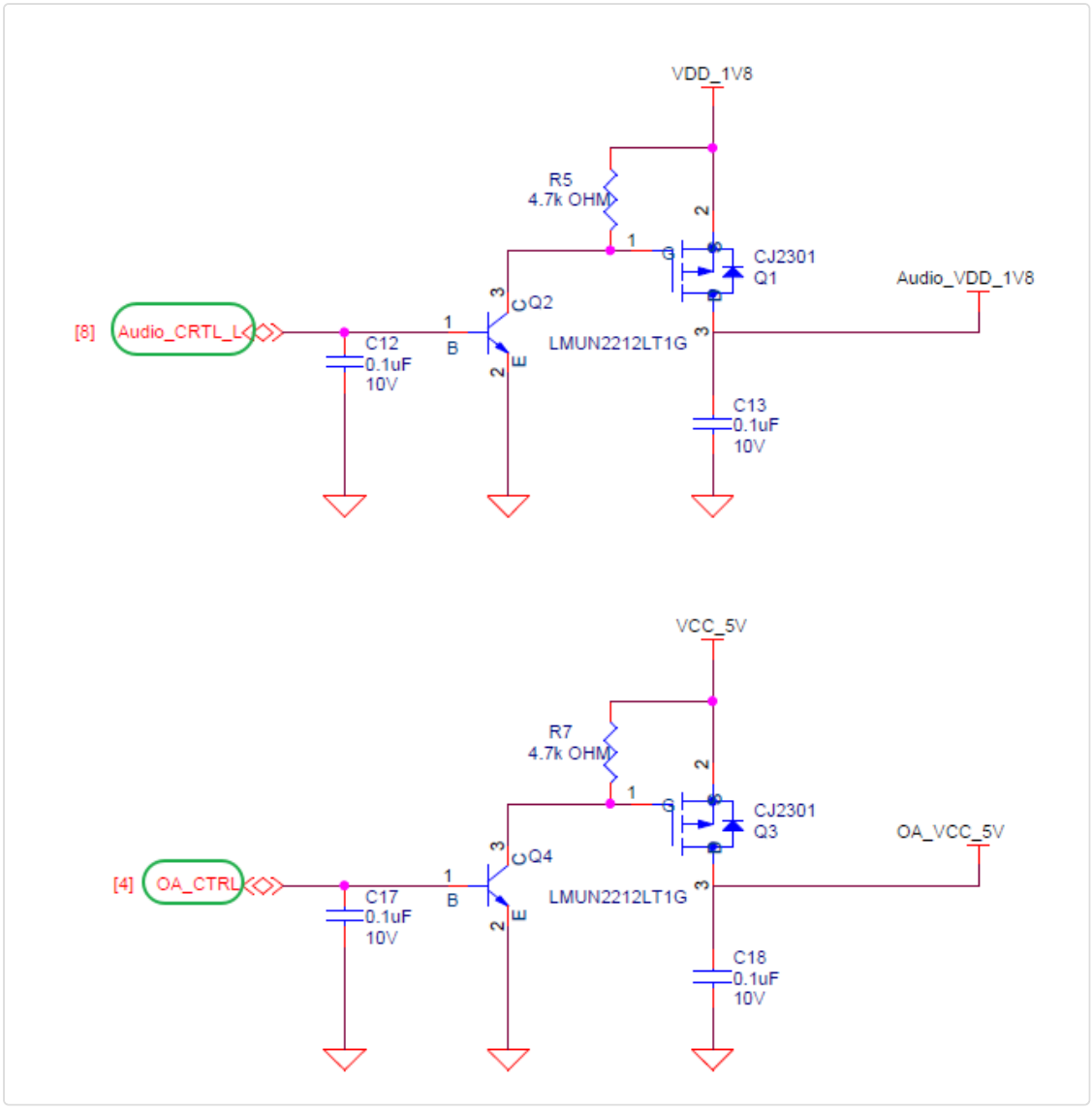
ES8388 Connection Interface



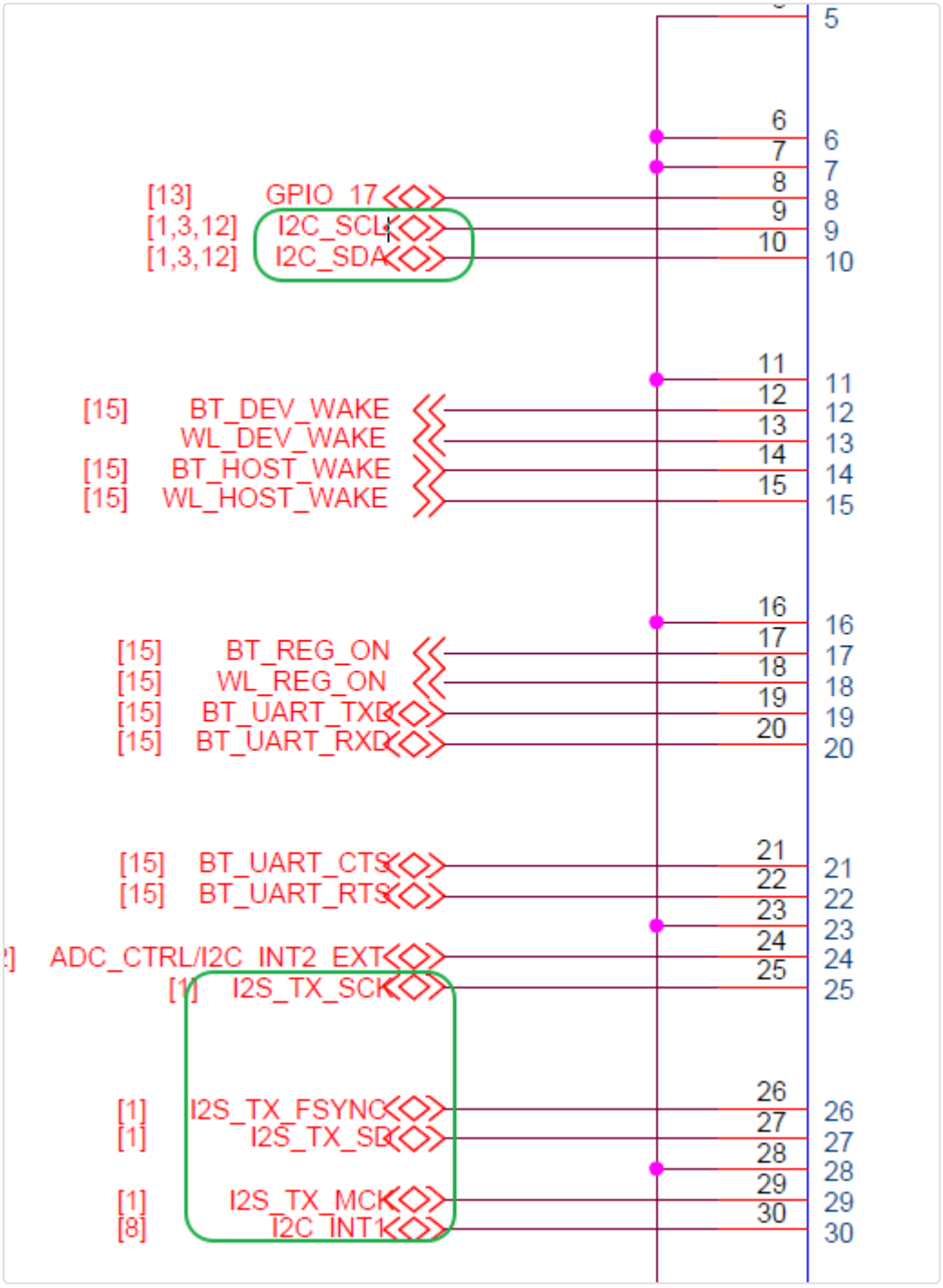
Speaker Interface



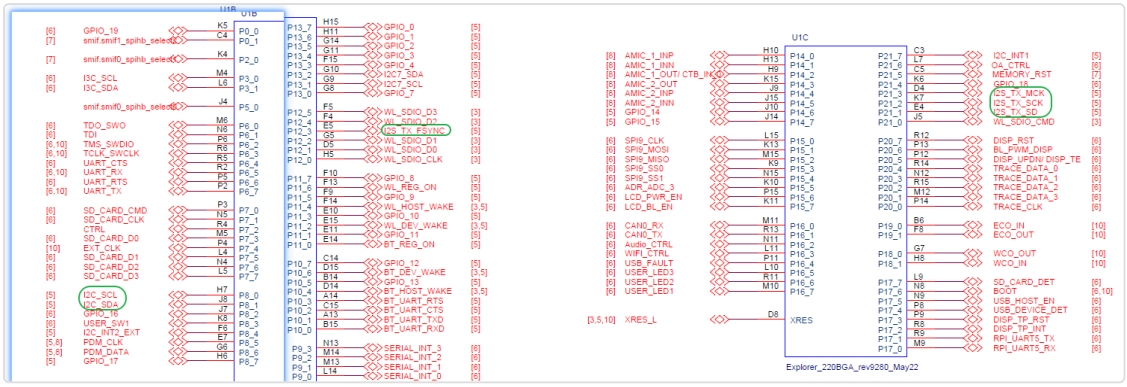
Control Pins



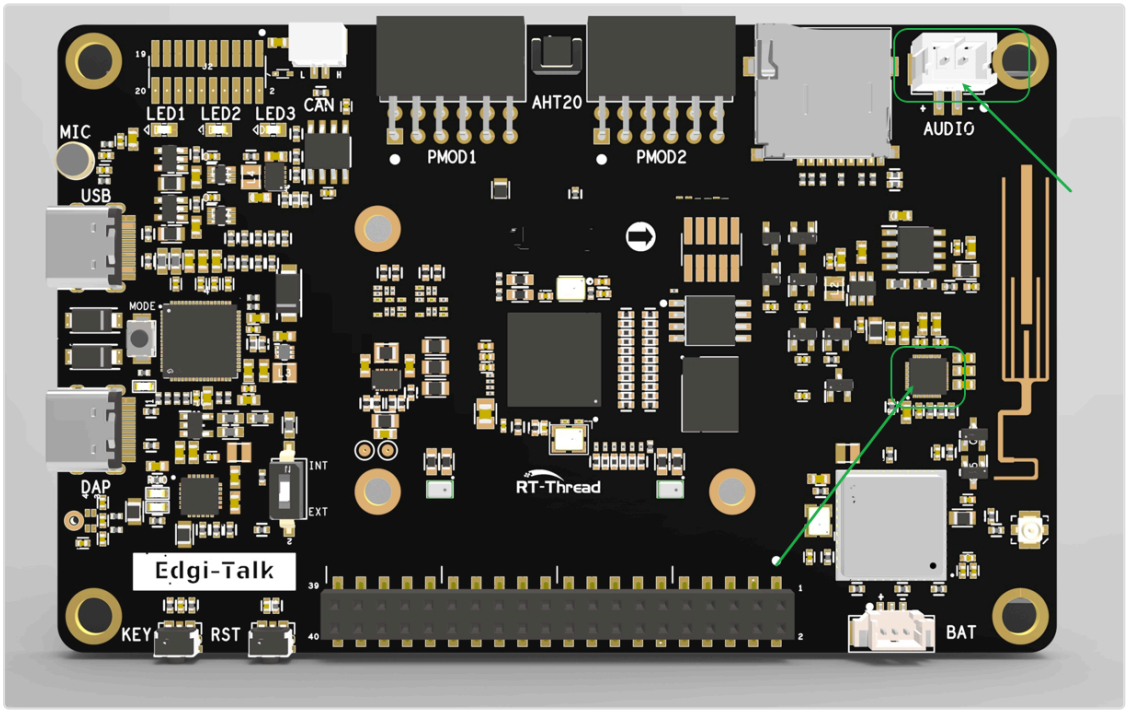
BTB Connector



MCU Interface



Physical Board Location



Software Description

- The project is developed based on the **Edgi-Talk M33** platform.
- Example features include:
- Microphone audio data acquisition
- Audio playback through the speaker
- Button control for play/stop

- LED indication for playback status
- The project structure is clear and helps users understand RT-Thread's audio device driver and event handling mechanisms.

Usage Instructions

Compilation and Download

1. Open the project and complete the compilation.
2. Connect the board's USB port to the PC using the **onboard debugger (DAP)**.
3. Use the programming tool to flash the generated firmware to the development board.
- The project can automatically call the signing tool (e.g., `tools/edgeprotecttools/bin/edgeprotecttools.exe`) to merge and flash the signed firmware (e.g., `proj_cm33_s_signed.hex`).

Runtime Behavior

- After flashing, power on the board to run the example.
- The LED turns **on by default**, indicating that audio playback is enabled.
- Press the button to **toggle the playback state**:
- **Play**: LED on — microphone-captured audio is played through the speaker
- **Stop**: LED off — audio playback is paused
- The system continuously captures and plays audio data, achieving real-time audio loopback.

Notes

- To modify the **graphical configuration** of the project, open the configuration file using the following tool:

```
tools/device-configurator/device-configurator.exe
```

```
libs/TARGET_APP_KIT_PSE84_EVAL_EPC2/config/design.modus
```

- After editing, save the configuration and regenerate the code.

Boot Sequence

The system boot sequence is as follows:

```
+------------------+
|   Secure M33     |
|   (Secure Core)  |
+------------------+
          |
          v
+------------------+
|       M33        |
| (Non-Secure Core)|
+------------------+
          |
          v
+--------------------+
|       M55          |
| (Application Core)|
+--------------------+
```

⚠️ Please strictly follow the boot sequence above when flashing the firmware, otherwise the system may fail to run properly.

- If the example project does not run correctly, compile and flash the **Edgi-Talk_M33_S_Template** project first to ensure proper initialization and core startup sequence before running this example.
- To enable the M55 core, configure the **M33 project** as follows:

```
RT-Thread Settings --> Hardware --> select SOC Multi Core Mod
```

2.4. Edgi-Talk_emUSB-device_CDC_Echo Example Project

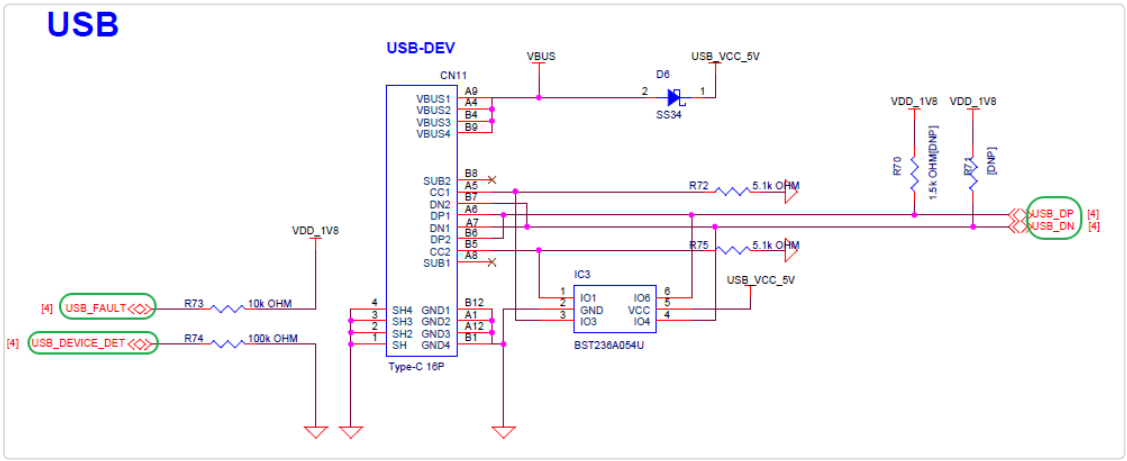
中文 | **English**

Introduction

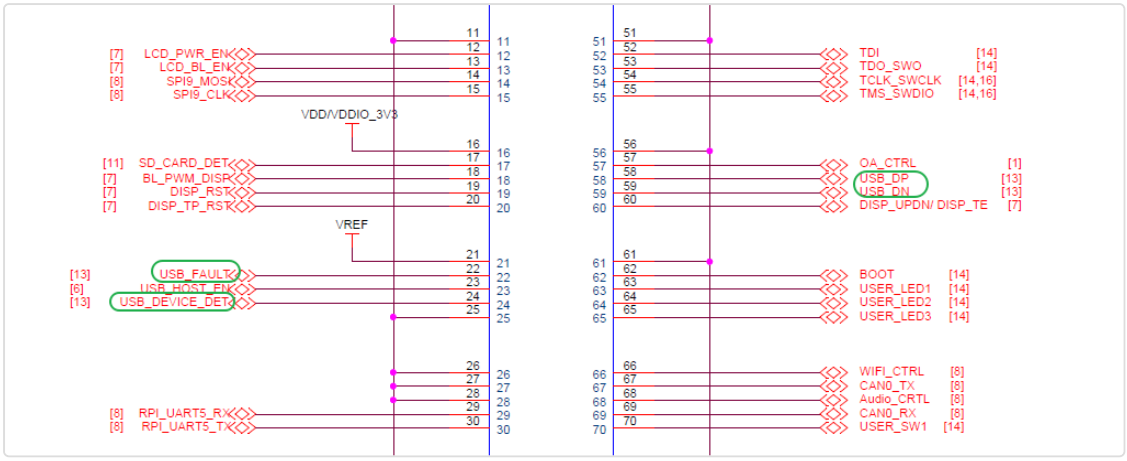
This example project is based on the **Edgi-Talk platform**, demonstrating the **USB CDC (Virtual COM Port) echo functionality** running on the **RT-Thread real-time operating system (M33 core)**.
Through this project, users can quickly experience USB CDC device communication and verify data echo functionality, providing a reference for future USB communication and multi-core application development.

Hardware Description

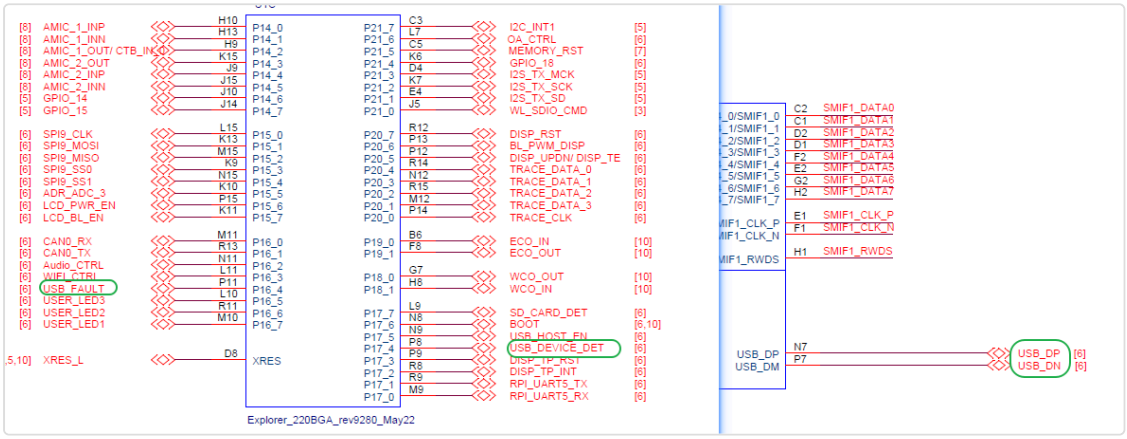
USB Interface



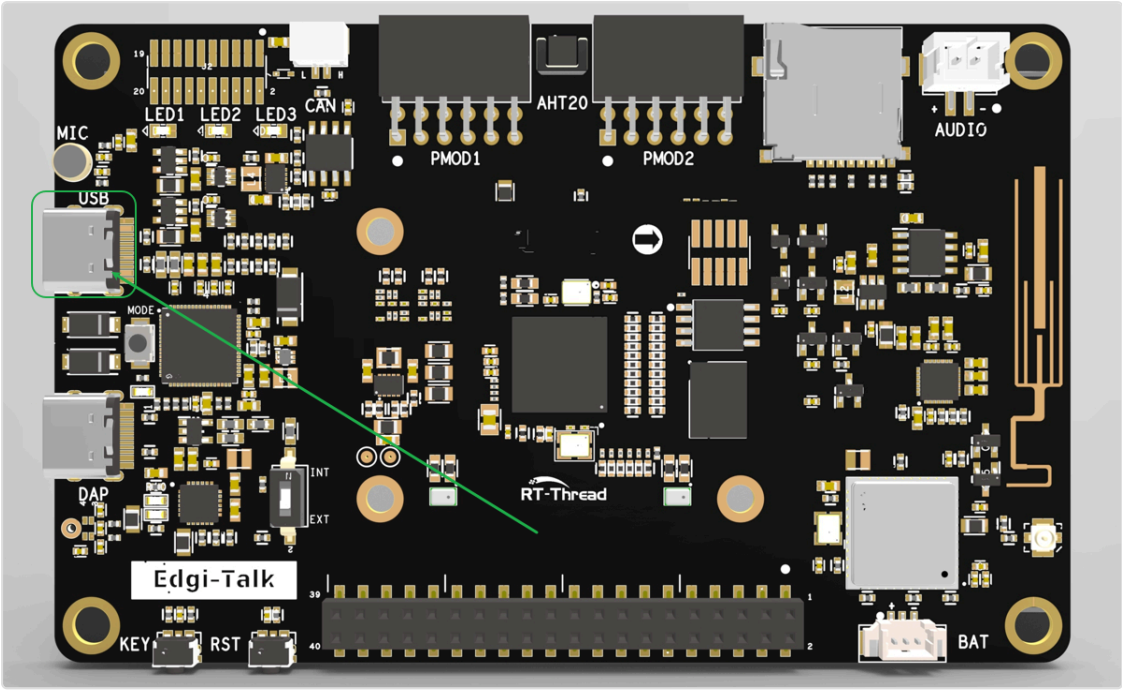
BTB Socket



MCU Interface



Physical Board LED/Port Location



Software Description

- The project is developed based on the **Edgi-Talk** platform.
- Example features include:
- USB CDC device initialization
- Virtual COM port data echo
- The project structure is clear, making it easy to understand how the USB device driver runs on the **M33 core**.

Usage

Build and Download

1. Open the project and compile it.
2. Connect the board’s USB interface to the PC using the **onboard debugger (DAP)**.
3. Use the programming tool to flash the generated firmware to the development board.

Running Result

- After flashing, power on the board to run the example.
- Manually enter the following command in the serial terminal:

```
cdc_sample
```

- The system outputs the following startup information:

```
 \ | /
- 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**
```

- On the PC, use any serial terminal tool to connect to the board's USB virtual COM port:

1. Open the serial terminal tool on your PC and connect to the board's virtual COM port (baud rate can be any value).
2. Enter a string and **end it with a newline character** `\n`.
3. The board will echo the complete string back to the terminal:

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

Notes

- **Echo trigger condition**:
  Echo occurs only when the last character of the received data is a **newline** `\n`.
  If the input does not include `\n`, the data is stored in the buffer and will not be echoed immediately.
- To modify the **graphical configuration** of the project, open the configuration file using the following tool:

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

- After modification, save the configuration and regenerate the code.

Startup Sequence

The system starts in the following order:

```
+------------------+
|   Secure M33     |
|  (Secure Core)   |
+------------------+
          |
          v
+------------------+
|       M33        |
| (Non-Secure Core)|
+------------------+
          |
          v
+--------------------+
|       M55          |
| (Application Core)|
+--------------------+
```

⚠️ Please strictly follow the above flashing sequence; otherwise, the system may fail to run properly.

- If the example does not run correctly, first compile and flash the **Edgi-Talk_M33_S_Template** project to ensure proper initialization and core startup, then run this example.
- To enable the M55 core, enable the following configuration in the **M33 project**:

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

2.5. Edgi-Talk_M33_HyperRam Example Project

中文 | **English**

Introduction

This example project is based on the **Edgi-Talk platform**, demonstrating the **HyperRam functionality** running on the **RT-Thread real-time operating system**.
Through this project, users can quickly verify HyperRam operation, providing a basic reference for subsequent hardware control and application development.

Software Description

- The project is developed based on the **Edgi-Talk** platform.
- Example features include:
- The project structure is simple, making it easy to understand HyperRam control logic and hardware driver interfaces.

```
#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

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
```

Usage

Build and Download

1. Open the project and compile it.
2. Connect the board's USB interface to the PC using the **onboard debugger (DAP)**.
3. Use the programming tool to flash the generated firmware to the development board.

Running Result

- After flashing, power on the board to run the example project.

Notes

- To modify the **graphical configuration** of the project, open the configuration file using the following tool:

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

- After modification, save the configuration and regenerate the code.

Startup Sequence

The system starts in the following order:

```
+------------------+
|   Secure M33     |
|  (Secure Core)   |
+------------------+
          |
          v
+------------------+
```

```
|       M33        |
| (Non-Secure Core)|
+------------------+
          |
          v
+------------------+
|       M55        |
| (Application Core)|
+------------------+
```

⚠️ Please strictly follow the above flashing sequence; otherwise, the system may fail to run properly.

- If the example does not run correctly, first compile and flash the **Edgi-Talk_M33_S_Template** project to ensure proper initialization and core startup, then run this example.
- To enable the M55 core, enable the following configuration in the **M33 project**:

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

2.6. Edgi-Talk_M33_S_HyperRam Example Project

中文 | **English**

Introduction

This example project is based on a **Bare Metal** architecture, mainly demonstrating and configuring **Secure M33 HyperRam** features.
Additionally, this project can serve as a foundational template for secondary development or project creation, helping users quickly get started and extend functionality.

HyperRAM Overview

1. General Description

HyperRAM is a **high-performance, low-pin-count (LPC) DRAM** initially introduced by **Cypress (now Infineon)**.

It uses the **HyperBus interface**, primarily targeting applications requiring **high bandwidth, low power, and simple interfaces**, such as **embedded systems, display control, IoT devices, and automotive electronics**.

HyperRAM is classified as **pSRAM (Pseudo-SRAM)**: externally it behaves like SRAM (simple read/write without user refresh), but internally it is low-power DRAM (self-refresh).

2. Architecture and Interface

HyperRAM uses the **HyperBus interface**, characterized by:

- **Few pins**: typically only **13 signal pins** (8-bit data bus + control/clock), greatly reducing PCB complexity compared to traditional SDRAM (dozens of pins).
- **Double Data Rate (DDR) transfer**: data is transmitted on both rising and falling clock edges to increase bandwidth.
- **Serial control protocol**: accesses memory using command-address-data sequences, simplifying design.

Interface structure:

- **Data lines DQ[7:0]**: 8-bit bidirectional data
- **RWDS (Read-Write Data Strobe)**: data synchronization signal
- **CLK**: clock input
- **CS#**: chip select
- **RESET#**: reset
- **CKE**: clock enable

3. Working Principle

HyperRAM is accessed via **command + address + data**:

1. **Command Phase**

- The host sends a read/write command and target address.

2. **Latency Phase**

- HyperRAM prepares its internal memory array (configurable latency).

3. **Data Transfer Phase**

- Data is transferred on **DQ[7:0]** in DDR mode, with RWDS providing data synchronization.

Internally it uses DRAM technology with **self-refresh**, but externally it behaves like SRAM — users do not need to manage refresh operations.

4. Performance Features

- **Data bus width**: 8-bit
- **Operating voltage**: 1.8 V or 3.0 V (low-power design)
- **Data rate**: up to **400 MB/s (200 MHz DDR × 8-bit)**
- **Capacity range**: 32 Mb ~ 512 Mb (4 MB ~ 64 MB)
- **Low power**: supports deep sleep mode, standby current < 10 µA
- **Simple interface**: high-speed access using only 13 pins

5. Advantages of HyperRAM

1. **Low pin count**

- Significantly reduces pin requirements compared to traditional SDRAM/PSRAM (30+ pins), saving PCB routing.

2. **High bandwidth**

- DDR interface, bandwidth up to 400 MB/s, suitable for **image buffering and display refresh**.

3. **Low power consumption**

- Ideal for battery-powered devices such as IoT and wearable devices.

4. **Ease of use**

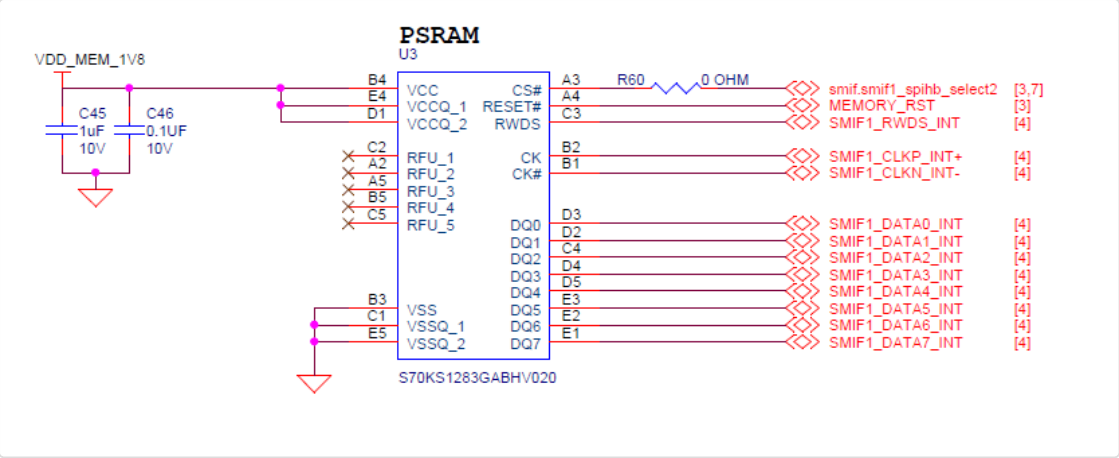
- Behaves like SRAM externally, simple and user-friendly, no manual refresh needed.

6. Comparison with Other Memories

Feature	HyperRAM	SDRAM / DDR	SRAM	NOR Flash
Interface	HyperBus (13-pin)	Parallel 16~32-bit	Parallel/Serial	SPI/QSPI
Capacity	32Mb ~ 512Mb	64Mb ~ 1Gb+	Small (Kb~Mb)	4Mb ~ 2Gb
Bandwidth	~400 MB/s	~800 MB/s+	~50 MB/s	~100 MB/s
Power	Low	High	Low	Low
Applications	Cache/Frame Buffer	System Memory	High-speed small storage	Program Storage

Hardware Description

HyperRam Interface



Software Description

- The project is developed based on the **Edgi-Talk** platform.
- The example covers:
- **Secure region configuration**
- **Basic startup process demonstration**
- The project code structure is simple and clear, making it easy to understand and port.

Usage

Build and Download

1. Open the project and compile it.
2. Connect the board’s USB interface to the PC using the **onboard debugger (DAP)**.
3. Use the programming tool to flash the compiled firmware to the board.

Running Result

- After flashing, power on the board to run the example.
- The system will start successfully and jump to the **M33 core**, indicating that the secure configuration is effective.

Notes

- To modify the **graphical configuration**, open the configuration file using the following tool:

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

- After modification, save the configuration and regenerate the code.

Startup Sequence

The system starts in the following order:

```
+------------------+
|   Secure M33     |
|  (Secure Core)   |
+------------------+
          |
          v
+------------------+
|       M33        |
| (Non-Secure Core)|
+------------------+
          |
          v
+-------------------+
|       M55         |
| (Application Core)|
+-------------------+
```

⚠️ Please strictly follow the above flashing sequence; otherwise, the system may fail to run properly.

2.7. Edgi-Talk_Key_Irq Example Project

中文 | **English**

Introduction

This example project is based on the **Edgi-Talk platform**, running on the **RT-Thread real-time operating system**, demonstrating the usage of **Key IRQ (Button Interrupts)**.
Through this project, users can learn how to configure GPIO interrupts in RT-Thread and implement button-triggered event responses, providing a reference for future human-machine interaction applications.

MCU Interrupt System Overview

Key interrupt features:

1. **NVIC (Nested Vector Interrupt Controller)**

- Supports **interrupt nesting**: high-priority interrupts can preempt lower-priority ones
- Supports **priority grouping** (Preemption Priority and Subpriority)
- IRQn maps to the interrupt vector table, and NVIC calls the corresponding ISR

2. **GPIO External Interrupt (EXTI)**

- Each GPIO pin can be configured as an external interrupt input
- Supports **rising edge, falling edge, or both edges trigger**
- Interrupt channels are managed by the **ICU (Interrupt Controller Unit)**
- Configurable via Renesas FSP or low-level registers
- Low response latency, suitable for buttons, sensors, and other event-driven inputs

3. **RT-Thread PIN Driver Model**

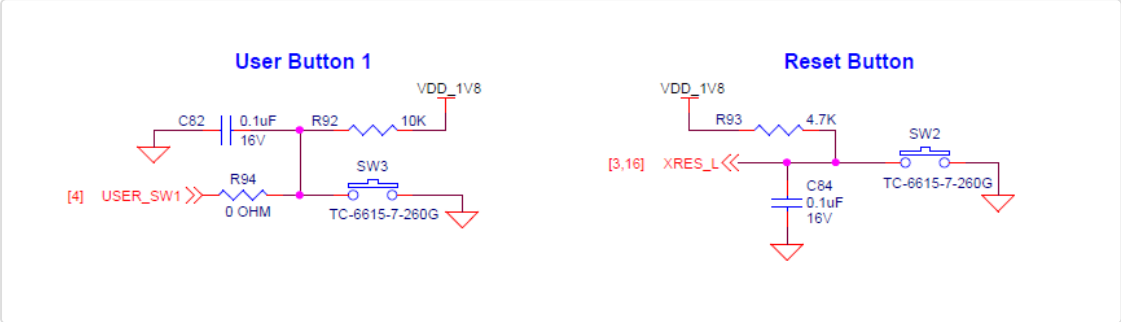
- GPIOs are encapsulated as **PIN devices** in RT-Thread
- Provides a unified interface:
    - `rt_pin_mode(pin, mode)`: configure input/output mode

- `rt_pin_read(pin)` / `rt_pin_write(pin, value)`: read/write GPIO
- `rt_pin_attach_irq(pin, mode, callback, args)`: register interrupt callback
- `rt_pin_irq_enable(pin, enable)`: enable/disable interrupt

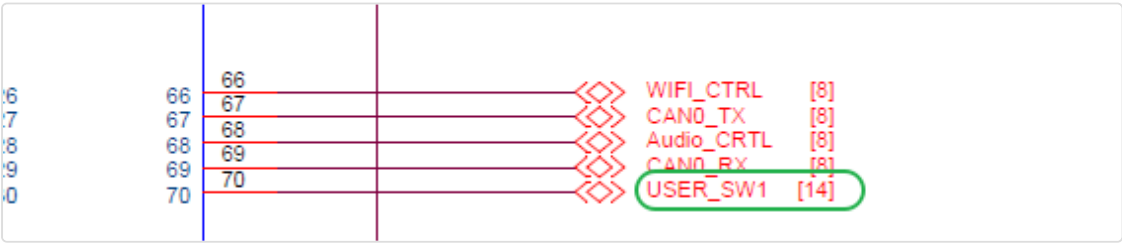
> *Using the RT-Thread PIN driver, you can handle interrupts without directly manipulating NVIC or MCU registers.*

Hardware Description

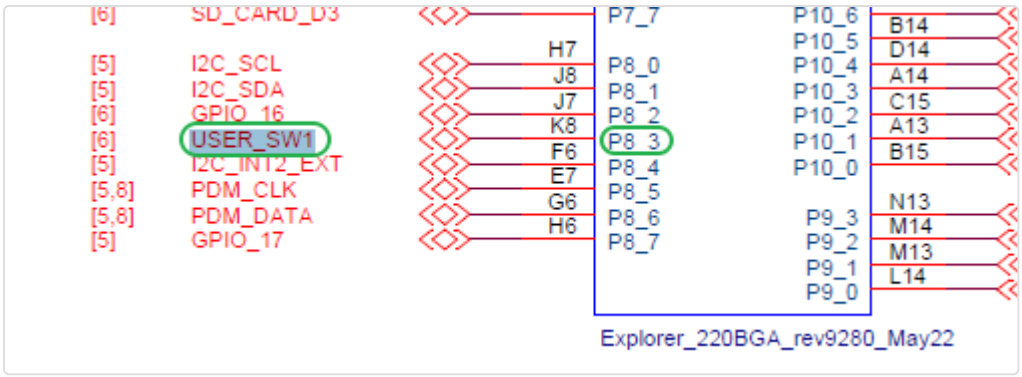
Button Interface



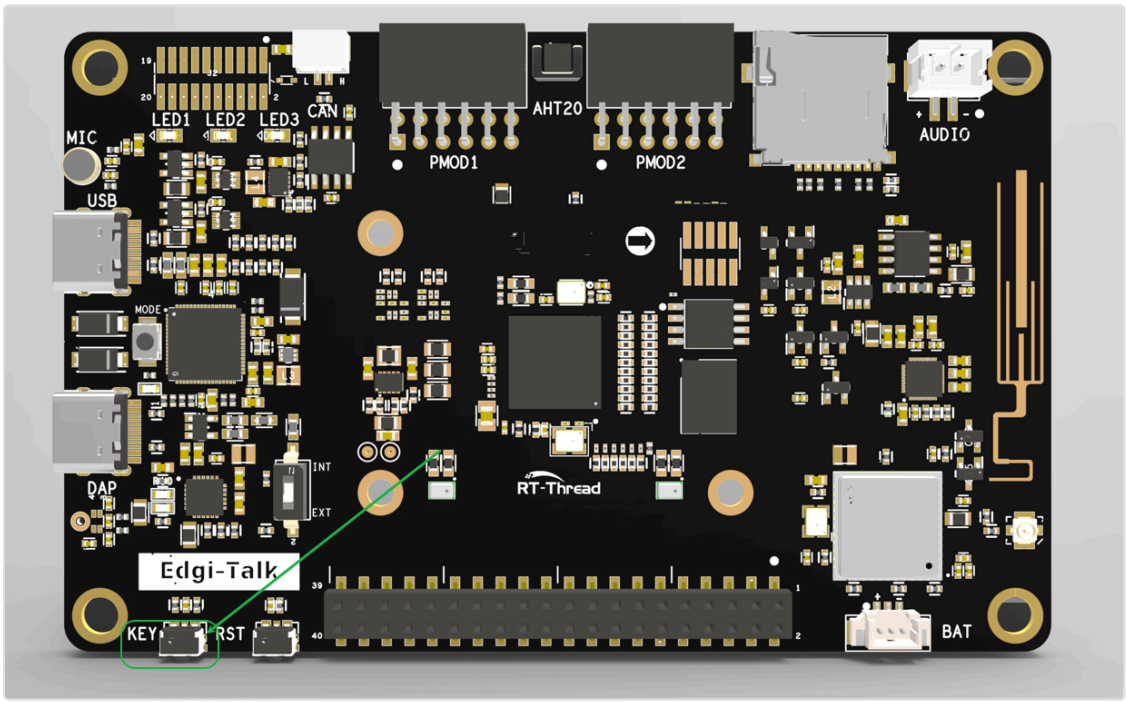
BTB Socket



MCU Interface



Physical Board Location



Software Description

- Developed on the **Edgi-Talk** platform.
- Uses **RT-Thread** as the OS kernel.
- Example features include:
- Configure button GPIO as interrupt input
- Trigger interrupt callback when the button is pressed

- Blue LED blinks at a 500ms period, indicating normal system operation

Usage

Build and Download

1. Open the project and compile it.
2. Connect the board's USB interface to the PC using the **onboard debugger (DAP)**.
3. Flash the generated firmware to the board using a programming tool.

Running Result

- After flashing, power on the board to run the example.
- The **blue LED** will blink at a 500ms period, indicating the system scheduler is running.
- When a user presses the button, the interrupt callback is triggered, and the following message is printed to the serial terminal:

```
The button is pressed
```

Notes

- To modify the **graphical configuration**, open the configuration file using the following tool:

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

- After modification, save the configuration and regenerate the code.

Startup Sequence

The system starts in the following order:

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

```
|  (Secure Core)   |
+------------------+
          |
          v
+------------------+
|       M33        |
| (Non-Secure Core)|
+------------------+
          |
          v
+--------------------+
|       M55          |
| (Application Core)|
+--------------------+
```

⚠️ Please strictly follow the above flashing sequence; otherwise, the system may fail to run properly.

- If the example does not run correctly, first compile and flash the **Edgi-Talk_M33_S_Template** project to ensure proper initialization and core startup, then run this example.
- To enable the M55 core, enable the following configuration in the **M33 project**:

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

2.8. Edgi-Talk_LSM6DS3 Example Project

中文 | **English**

Introduction

This example is based on the **Edgi-Talk platform**, running on the **RT-Thread real-time operating system**, demonstrating how to drive the **LSM6DS3 six-axis sensor (accelerometer + gyroscope + temperature)**.

LSM6DS3TR Overview

LSM6DS3TR is a **low-power six-axis Inertial Measurement Unit (IMU)** from STMicroelectronics, integrating a three-axis accelerometer and a three-axis gyroscope.

Key Features

- **Three-axis accelerometer**: ±2/±4/±8/±16 g
- **Three-axis gyroscope**: ±125/±245/±500/±1000/±2000 dps
- **Operating voltage**: 1.71 V – 3.6 V
- Low power consumption, supporting multiple power-saving modes
- Built-in FIFO buffer (up to 8 KB)
- Supports **I²C** and **SPI** communication interfaces

Applications

- Smartphones and wearable devices
- Motion detection and posture recognition
- Gesture recognition and gait analysis
- Robotics and drone attitude control

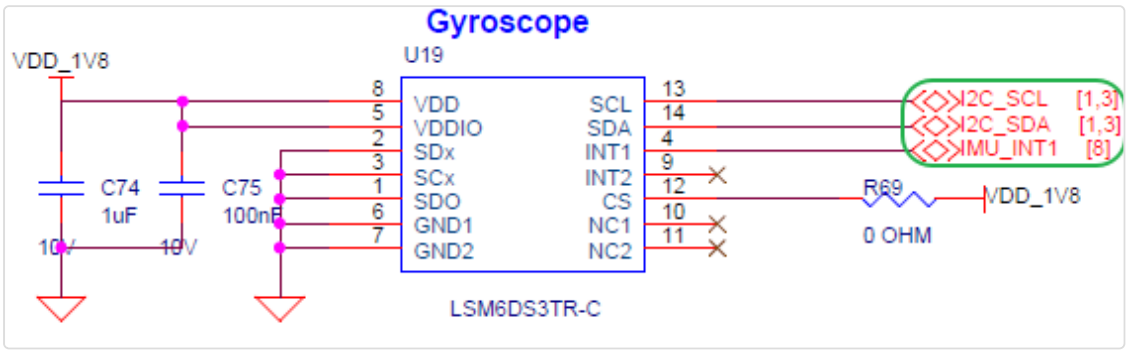
Through this example, users can learn to:

- Use RT-Thread's **I²C device driver framework**
- Initialize and configure LSM6DS3 registers

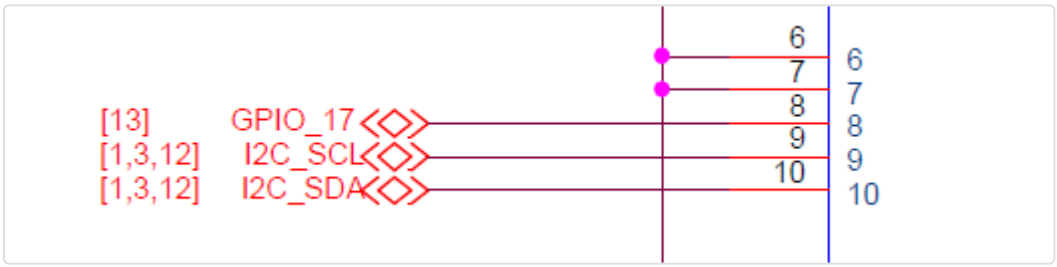
- Read **three-axis acceleration**, **three-axis angular velocity**, and **temperature data**
- Output sensor data via the serial port

Hardware Description

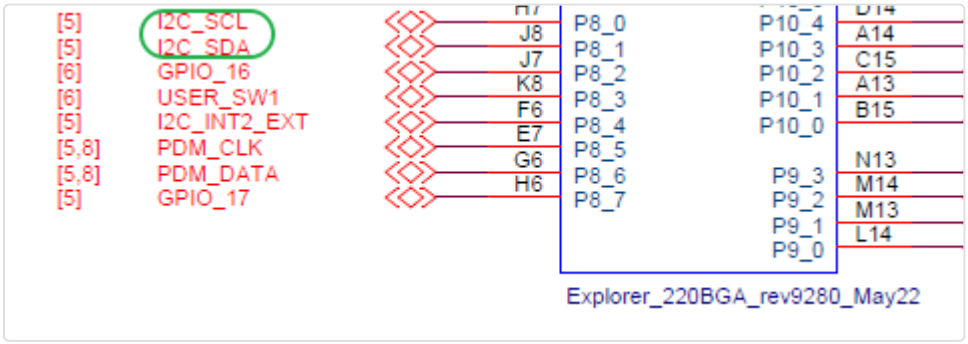
LSM6DS3TR Interface



BTB Socket



MCU Interface



Software Description

- Developed on the **Edgi-Talk** platform
- Uses **RT-Thread** as the OS kernel
- Example features include:
- Detect and verify device ID
- Reset sensor and restore default configuration
- Configure output data rate and full-scale range
- Polling mode to read acceleration, angular rate, and temperature
- Print sensor data to the serial terminal

Usage

Build and Download

1. Open the project and compile it.
2. Connect the board's USB interface to the PC using the **onboard debugger (DAP)**.
3. Flash the generated firmware to the board using a programming tool.

Running Result

- After flashing, power on the board to run the example:

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

- The **blue LED** will blink at a 500ms period, indicating that the system scheduler is running normally.

Notes

- To modify the **graphical configuration**, open the configuration file using the following tool:

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

- After modification, save the configuration and regenerate the code.

Startup Sequence

The system starts in the following order:

```
+------------------+
|   Secure M33     |
|  (Secure Core)   |
+------------------+
          |
          v
+------------------+
|       M33        |
| (Non-Secure Core)|
+------------------+
          |
          v
+------------------+
|       M55        |
| (Application Core)|
+------------------+
```

⚠️ Please strictly follow the above flashing sequence; otherwise, the system may fail to run properly.

- If the example does not run correctly, first compile and flash the **Edgi-Talk_M33_S_Template** project to ensure proper initialization and core startup, then run this example.
- To enable the M55 core, enable the following configuration in the **M33 project**:

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

2.9. Edgi-Talk_M55_MIPI_LCD Example Project

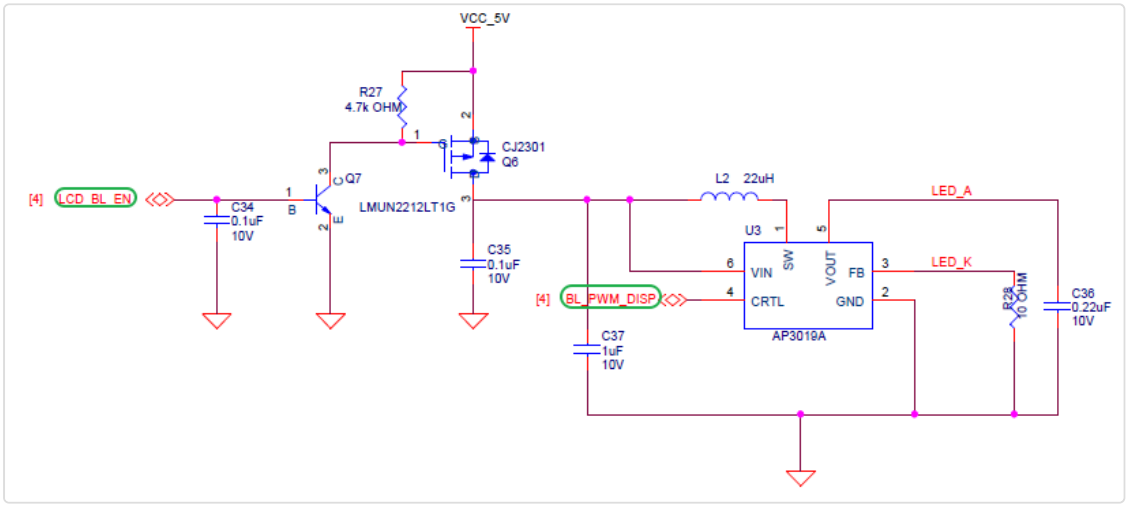
中文 | **English**

Introduction

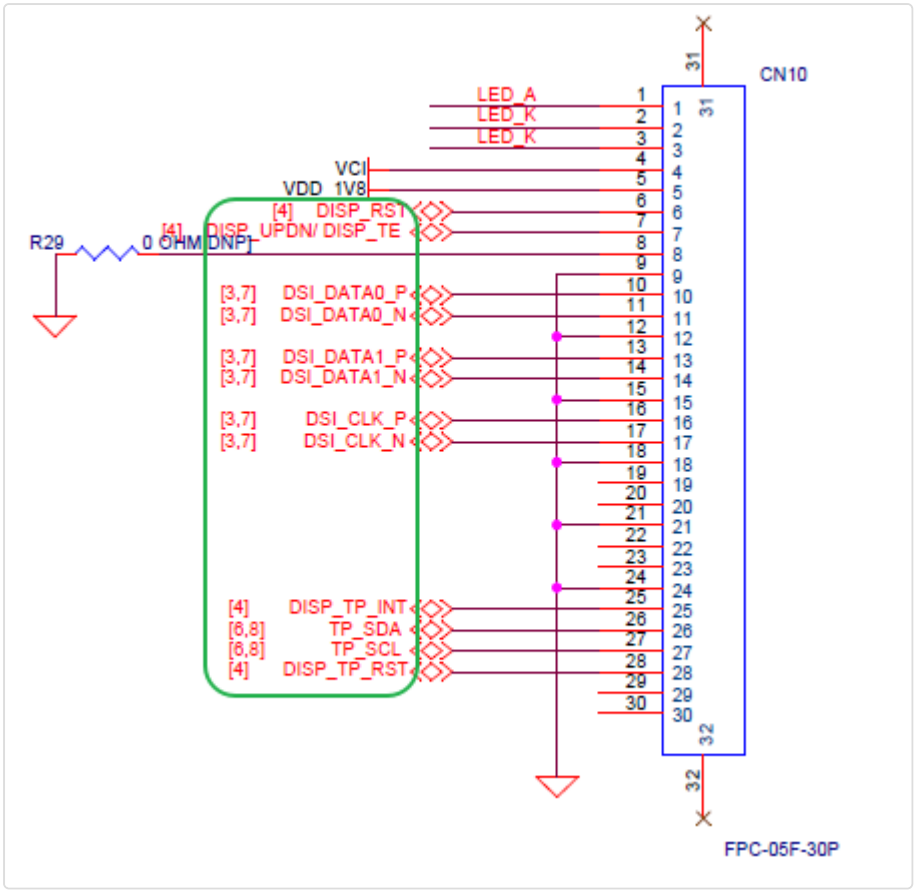
This example project runs on the **M55 application core** with **RT-Thread RTOS**, demonstrating **MIPI LCD screen refresh functionality**.
It allows users to quickly validate the **MIPI DSI interface driver** and **LCD initialization**, providing a reference for future GUI applications or LVGL porting.

Hardware Description

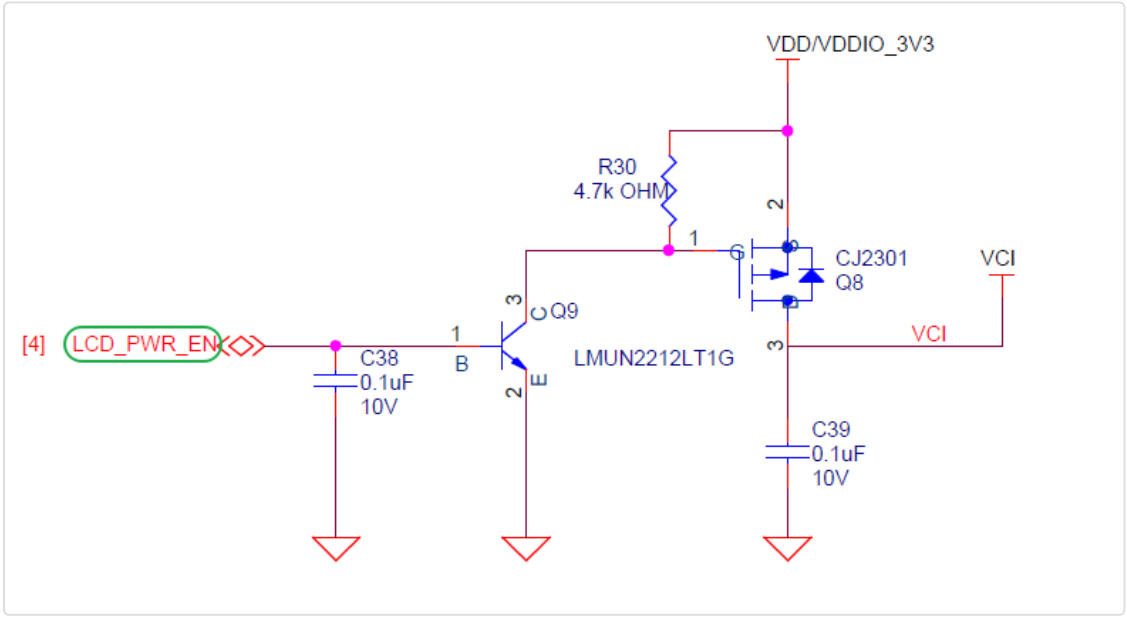
Backlight Interface



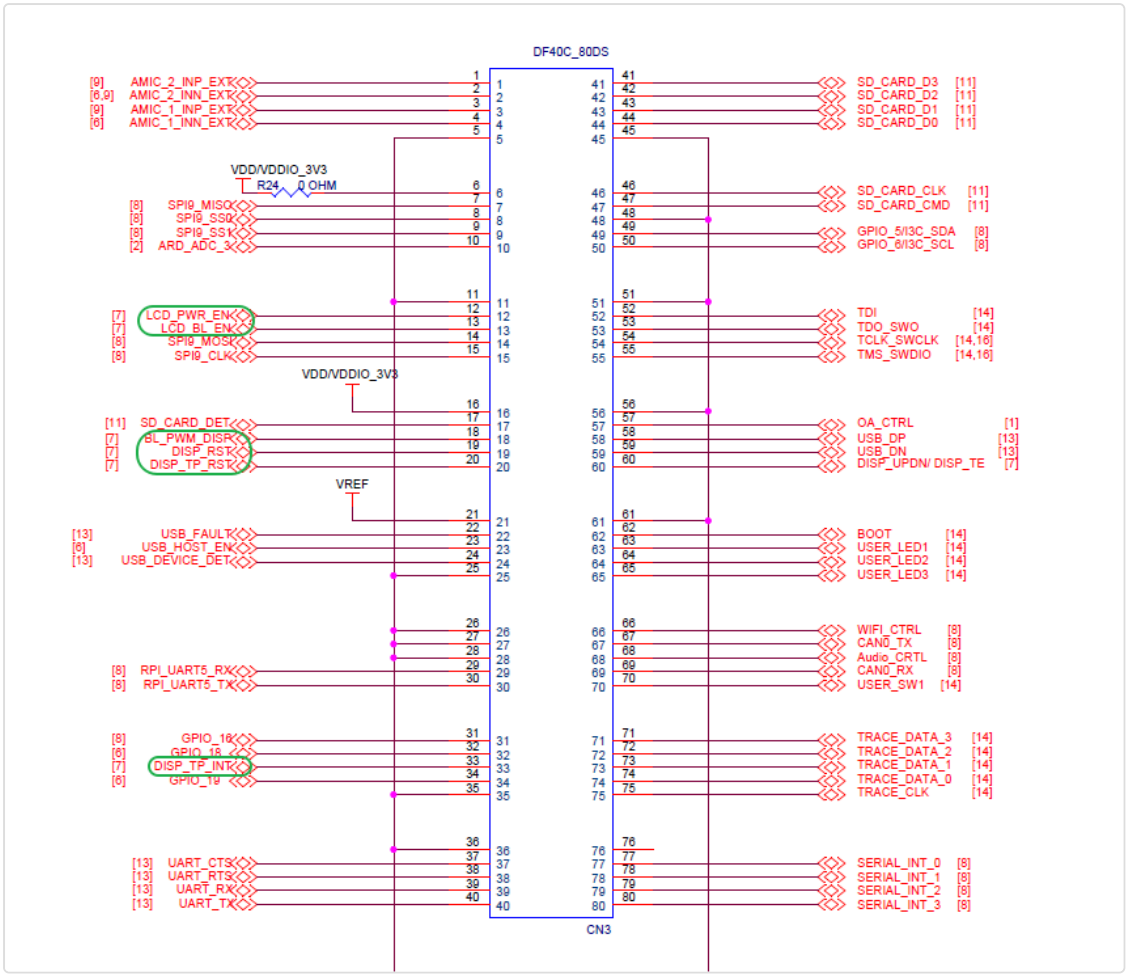
MIPI Interface

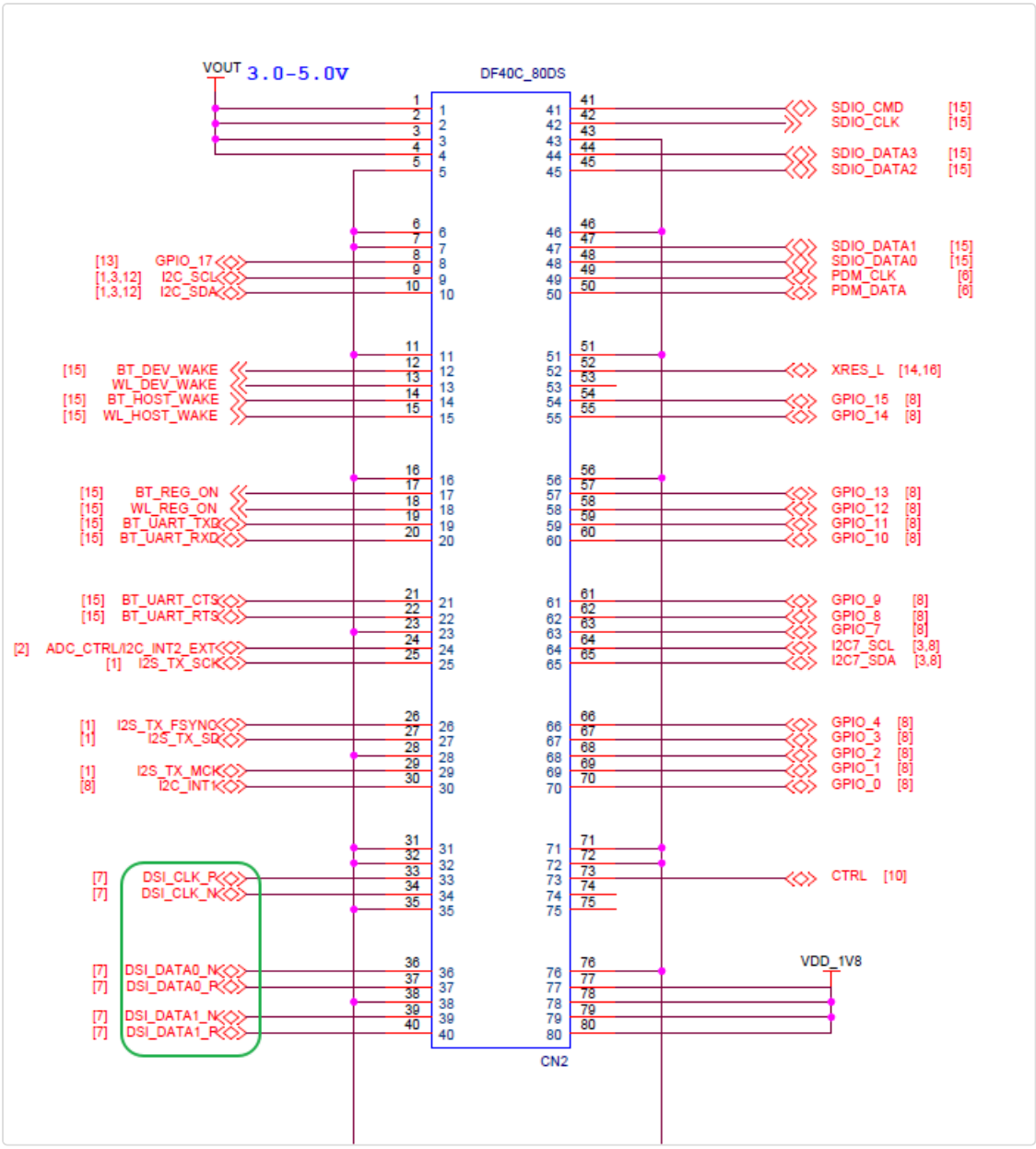


Power Interface

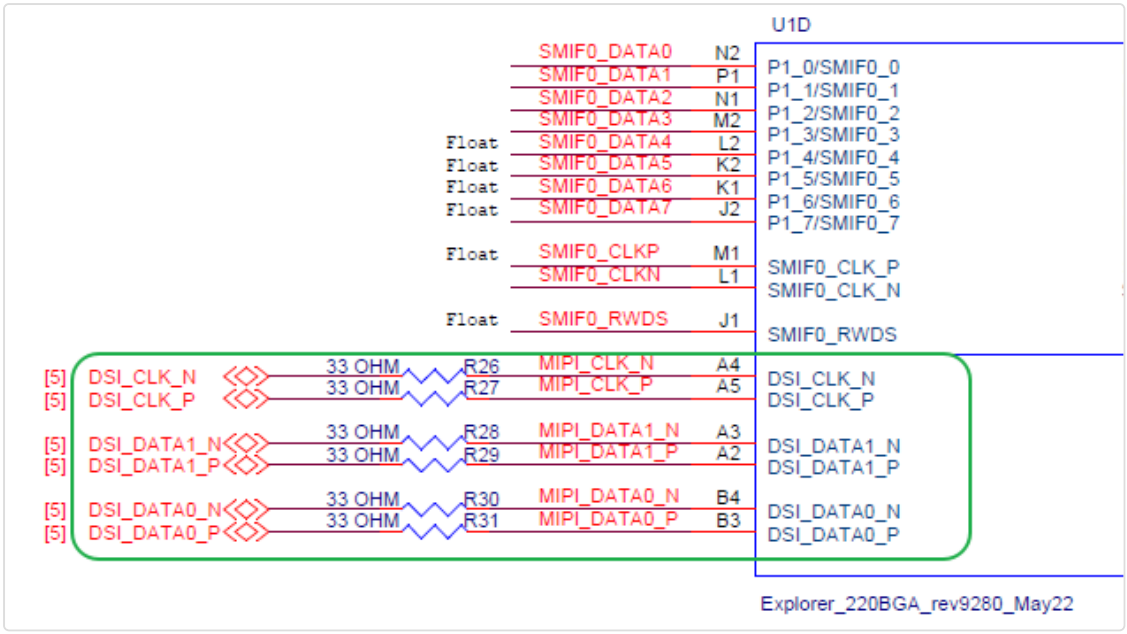


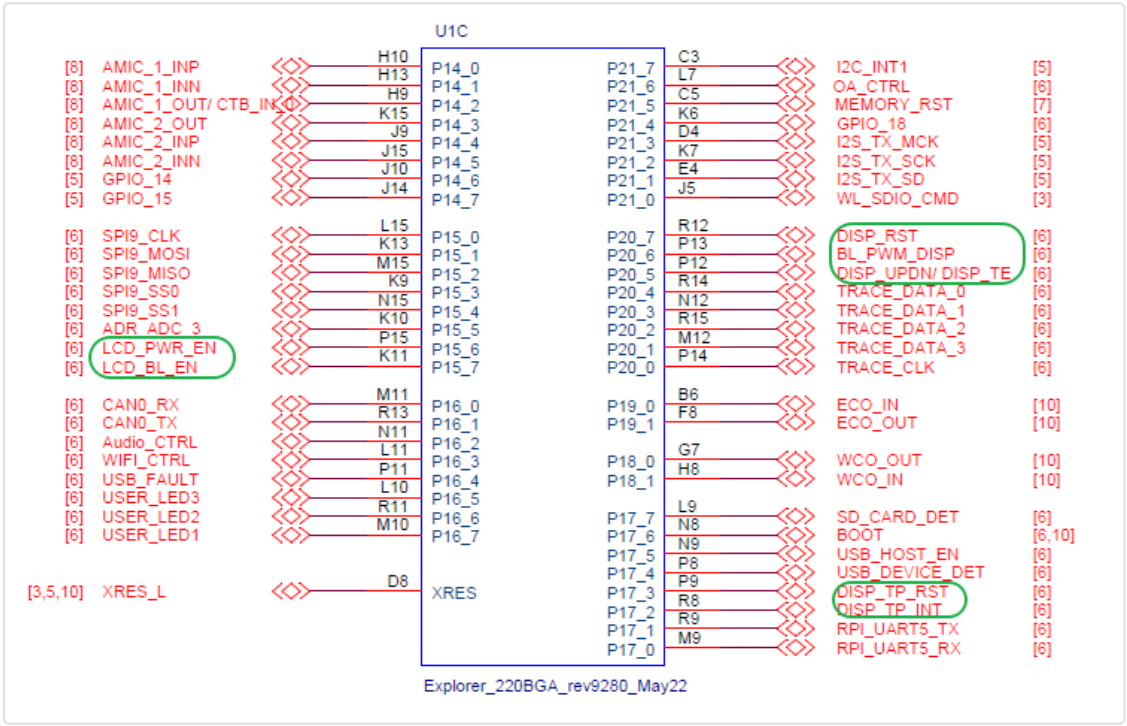
BTB Socket


DF40C_80DS
CN3
VDD/VDDIO_3V3
R24 0 OHM
VDD/VDDIO_3V3
VREF
[9] AMIC_2_INP_EXT
[6,9] AMIC_2_INN_EXT
[9] AMIC_1_INP_EXT
[6] AMIC_1_INN_EXT
[8] SPI9_MISO
[6] SPI9_SS0
[6] SPI9_SS1
[2] ARD_ADC_3
[7] LCD_PWR_EN
[7] LCD_BL_EN
[8] SPI9_MOSI
[8] SPI9_CLK
[11] SD_CARD_DET
[7] BL_PWM_DISP
[7] DISP_RST
[7] DISP_TP_RST
[13] USB_FAULT
[6] USB_HOST_EN
[13] USB_DEVICE_DET
[8] RPI_UART5_RX
[8] RPI_UART5_TX
[8] GPIO_16
[6] GPIO_18
[7] DISP_TP_INT
[6] GPIO_19
[13] UART_CTS
[13] UART_RTS
[13] UART_RX
[13] UART_TX
SD_CARD_D3 [11]
SD_CARD_D2 [11]
SD_CARD_D1 [11]
SD_CARD_D0 [11]
SD_CARD_CLK [11]
SD_CARD_CMD [11]
GPIO_5/I3C_SDA [8]
GPIO_6/I3C_SCL [8]
TDI [14]
TDO_SWO [14]
TCLK_SWCLK [14,16]
TMS_SWDIO [14,16]
OA_CTRL [1]
USB_DP [13]
USB_DN [13]
DISP_UPDN/ DISP_TE [7]
BOOT [14]
USER_LED1 [14]
USER_LED2 [14]
USER_LED3 [14]
WIFI_CTRL [8]
CAN0_TX [8]
Audio_CRTL [8]
CAN0_RX [8]
USER_SW1 [14]
TRACE_DATA_3 [14]
TRACE_DATA_2 [14]
TRACE_DATA_1 [14]
TRACE_DATA_0 [14]
TRACE_CLK [14]
SERIAL_INT_0 [8]
SERIAL_INT_1 [8]
SERIAL_INT_2 [8]
SERIAL_INT_3 [8]


VOUT 3.0-5.0V
DF40C_80DS
SDIO_CMD [15]
SDIO_CLK [15]
SDIO_DATA3 [15]
SDIO_DATA2 [15]
[13] GPIO_17
[1,3,12] I2C_SCL
[1,3,12] I2C_SDA
SDIO_DATA1 [15]
SDIO_DATA0 [15]
PDM_CLK [6]
PDM_DATA [6]
[15] BT_DEV_WAKE
WL_DEV_WAKE
[15] BT_HOST_WAKE
[15] WL_HOST_WAKE
XRES_L [14,16]
GPIO_15 [8]
GPIO_14 [8]
[15] BT_REG_ON
[15] WL_REG_ON
[15] BT_UART_TXD
[15] BT_UART_RXD
GPIO_13 [8]
GPIO_12 [8]
GPIO_11 [8]
GPIO_10 [8]
[15] BT_UART_CTS
[15] BT_UART_RTS
[2] ADC_CTRL/I2C_INT2_EXT
[1] I2S_TX_SCK
GPIO_9 [8]
GPIO_8 [8]
GPIO_7 [8]
I2C7_SCL [3,8]
I2C7_SDA [3,8]
[1] I2S_TX_FSYNC
[1] I2S_TX_SD
[1] I2S_TX_MCK
[8] I2C_INT1
GPIO_4 [8]
GPIO_3 [8]
GPIO_2 [8]
GPIO_1 [8]
GPIO_0 [8]
[7] DSI_CLK_P
[7] DSI_CLK_N
CTRL [10]
[7] DSI_DATA0_N
[7] DSI_DATA0_P
[7] DSI_DATA1_N
[7] DSI_DATA1_P
VDD_1V8
CN2

MCU Interface





Software Description

- Developed on the **Edgi-Talk** platform, running on **M55 core**.
- Example features:
- Initialize MIPI DSI interface and LCD panel

- Perform single-color fills on the LCD
- Loop screen refresh to verify LCD performance
- The project structure is simple, making it easy to understand the **MIPI DSI initialization** and **LCD refresh logic**.

Usage

Build and Download

1. Open the project and complete compilation.
2. Connect the board's USB port to your PC via the **onboard debugger (DAP)**.
3. Flash the compiled firmware to the board.

Running Result

- After flashing and powering on, the board will run the example project.
- The LCD will **fill with single colors sequentially**, each color held briefly.
- The refresh process loops continuously. Users can adjust the refresh period or add more test patterns as needed.

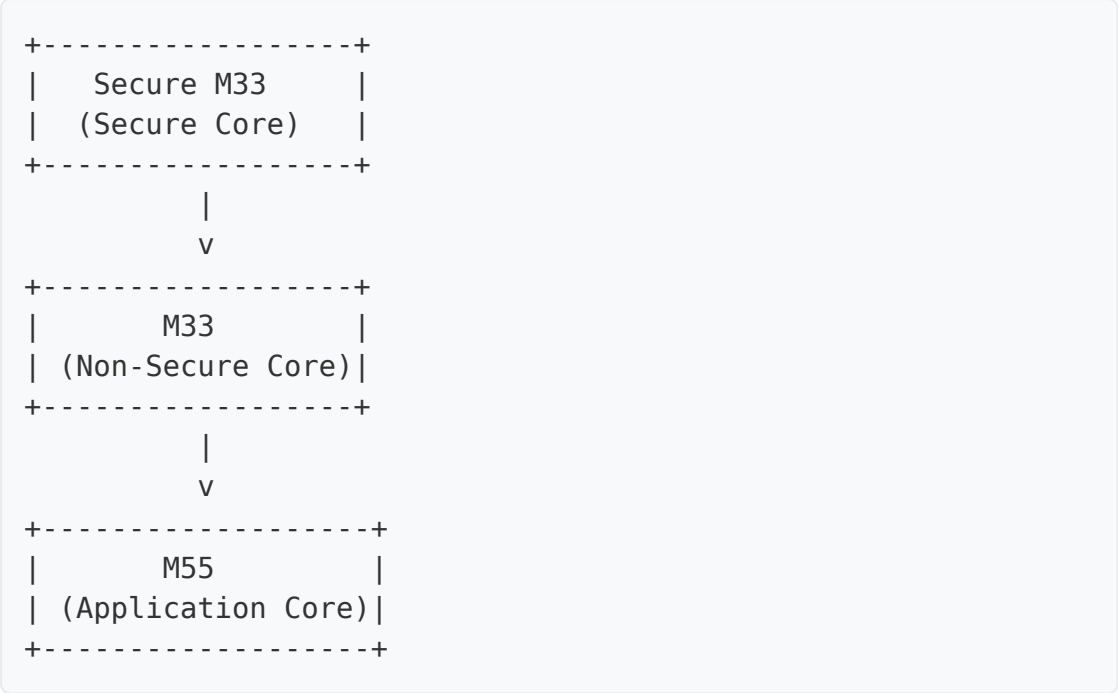
Notes

- To modify the **graphical configuration**, use the following tools:

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

- Save changes and regenerate code after editing.
- If the screen shows no output, check:
- MIPI DSI hardware connections
- LCD power and backlight supply

Startup Sequence



⚠️ Follow this flashing order strictly; otherwise, the system may not operate correctly.

- To enable M55, open the configuration in RT-Thread Settings:

```
Hardware --> select SOC Multi Core Mode --> Enable CM55 Core
```

2.10. Edgi-Talk_RTC Example Project

中文 | **English**

Introduction

This example project runs on the **M33 core** with **RT-Thread RTOS**, demonstrating the **RTC (Real-Time Clock)** functionality.
It allows users to quickly learn how to set, read, and print RTC time, providing a reference for time management and periodic tasks in embedded systems.

RTC Overview

1. Overview

RTC (Real-Time Clock) is an electronic module or chip that **keeps track of actual time** (year, month, day, hour, minute, second).
It can be integrated in an **MCU** or exist as an external chip, providing system time, alarm triggers, and timed event functions.

RTC is **low-power and stable**, continuing to operate even when the main power is off, using backup power such as a battery or supercapacitor.

2. Working Principle

RTC consists of a **low-power oscillator + counter**:

1. **Clock source**

- Typically uses a **32.768 kHz crystal**, providing a stable time base.

2. **Frequency division**

- Divides the crystal signal to produce a 1 Hz second pulse.
- Counts seconds to generate minutes, hours, days, months, and years.

3. **Registers**

- Internal registers store current time, date, alarms, etc.

4. **Power backup**

- Battery or supercapacitor maintains RTC operation during main power loss.

3. RTC Types

- **Internal RTC**
- Integrated inside MCU.
- Pros: lower cost, fewer components.
- Cons: crystal accuracy affected by PCB and temperature.
- **External RTC**
- Standalone chips, e.g., DS3231, PCF8563.
- Pros: high accuracy, I²C/SPI interface.
- Cons: increases PCB space and cost.

4. Key Parameters

Parameter	Description
Oscillator	Typically 32.768 kHz, low-power and stable
Accuracy	ppm / seconds per day, affects drift
Supply voltage	1.8~5V, supports backup power
Power consumption	1~5 µA (low-power mode)
Interface type	I²C, SPI, or internal MCU bus
Extended functions	Alarm, square wave, temperature compensation, timed wakeup

5. RTC Functions

- **Real-time clock**: provides current time and date.

6. Applications

- **Embedded devices**: time tracking and event logging.
- **Low-power IoT**: periodic wake-up for data collection.
- **Wearables**: precise timing for watches or fitness devices.
- **Data loggers & industrial control**: timestamps and logs.
- **Automotive electronics**: dashboard timing, recorders, infotainment.

Software Description

- Developed on the **Edgi-Talk** platform.
- Example features:
- Initialize RTC device
- Set date and time
- Delay and read current time
- Export RTC functions as `rtc_sample` msh command
- The structure clearly demonstrates RTC driver usage on **M33 core**.

Usage

Build and Download

1. Open the project and compile it.
2. Connect the board USB to your PC via **DAP**.
3. Flash the compiled firmware.

Running Result

- After powering on, open the serial terminal and run:

```
rtc_sample
```

- The system performs:

1. Initialize RTC device

2. Set date to `2025-07-01`
3. Set time to `23:59:50`
4. Print current time
5. Delay 3 seconds
6. Print time again

- Example output:

```
Tue Jul  1 23:59:50 2025
Tue Jul  1 23:59:53 2025
```

Notes

- Ensure RTC is properly connected and recognized by the system.
- To modify the graphical configuration:

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

- Save changes and regenerate code after editing.

Startup Sequence

```
+------------------+
|   Secure M33     |
|  (Secure Core)   |
+------------------+
          |
          v
+------------------+
|       M33        |
| (Non-Secure Core)|
+------------------+
          |
          v
+------------------+
|       M55        |
| (Application Core)|
+------------------+
```

⚠️ Follow this flashing order strictly; otherwise, the system may not operate correctly.

- If the example does not run, first compile and flash **Edgi-Talk_M33_S_Template** to ensure core initialization.
- To enable M55, open:

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

2.11. Edgi-Talk_SDCARD Example Project

中文 | **English**

Introduction

This example project runs on the **M33 core** with **RT-Thread RTOS**, demonstrating **SD card file operations**.
It allows users to quickly learn how to mount an SD card, write and read files, and operate files through the **serial command line**.

SD Card Overview

1. Overview

SD Card (Secure Digital Card) is a compact, portable non-volatile storage device widely used in **embedded systems, cameras, smartphones, and data loggers**.
It consists of a **controller + NAND Flash memory** and communicates with the host through a standard interface.

Key features:

- Small size: typically **32 × 24 × 2.1 mm** (standard)
- Non-volatile NAND Flash storage
- Supports hot-plug and power-loss protection

2. SD Card Types

By size:

Type	Dimensions
Standard SD	32 × 24 mm
Mini SD	21.5 × 20 mm
Micro SD (TF)	15 × 11 mm, most common

By capacity:

Type	Capacity
SDSC	1 MB ~ 2 GB
SDHC	4 GB ~ 32 GB
SDXC	32 GB ~ 2 TB
SDUC	2 TB ~ 128 TB

By speed class:

- **Class 2/4/6/10**: minimum write speed 2/4/6/10 MB/s
- **UHS-I/II/III**: up to 312 MB/s
- **Video Speed Class V6/V10/V30/V60/V90**: suitable for video recording

3. SD Card Interface

1. **SPI mode**: uses SPI bus (MISO, MOSI, SCK, CS), simple but slower.
2. **SD mode (1-bit/4-bit)**: dedicated SD bus, faster data transfer.
3. **UHS mode**: high-speed interface, used in cameras and high-performance embedded systems.

4. Working Principle

1. **Command/Data transfer**: host sends commands (CMD), card responds (R1, R2, etc.), read/write blocks (512 bytes each).
2. **Controller management**: handles bad blocks, ECC, logical-to-physical mapping.
3. **Data storage**: stored in NAND Flash, supports multiple erasures (typical 100,000 cycles).

5. Performance

Parameter	Description
Capacity	1 GB ~ 128 TB
Block size	512 Byte
Interface speed	SPI/SD 1-bit/4-bit/UHS
Max transfer	25 MB/s (standard), 312 MB/s (UHS-III)
Voltage	3.3 V (some Micro SD support 1.8 V)
Temperature	-25 ℃ ~ 85 ℃
Endurance	10^4 ~ 10^5 write cycles

6. Applications

- **Consumer electronics**: phones, tablets, cameras
- **Embedded systems**: MCU/FPGA storage, logging, configuration
- **Industrial**: data acquisition, controllers
- **Audio/Video**: high-speed video recording
- **Automotive**: dashcams, navigation storage

Software Description

- Developed on **Edgi-Talk** platform.
- Example features:
- Mount and initialize SD card
- Write file using `echo` command
- Read file using `cat` command
- Print results on serial terminal

- Provides a clear example of **RT-Thread filesystem and SD card interface usage**.

Usage

Build and Download

1. Open and compile the project.
2. Connect board USB to PC via **DAP**.
3. Flash the compiled firmware.
4. Insert SD card into the board.

Running Result

- After powering on, the system mounts the SD card and initializes the filesystem.
- Use the **serial terminal** to perform file operations:

1. Write `test.txt`:

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

2. Read file content:

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

- Sample serial output:

```
 \ | /
- 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!
```

- `echo` writes strings to files, `cat` reads and displays file content.

Notes

- Ensure the SD card is inserted and formatted with FAT filesystem (FAT16/FAT32).
- To modify graphical configuration:

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

- Save changes and regenerate code.
- If SD card fails to mount, check hardware connections and power supply.

Startup Sequence

```
+------------------+
|   Secure M33     |
|  (Secure Core)   |
+------------------+
          |
          v
+------------------+
|       M33        |
| (Non-Secure Core)|
+------------------+
          |
          v
+--------------------+
|       M55          |
| (Application Core)|
+--------------------+
```

⚠️ Follow this flashing order strictly.

- If the example does not run, first compile and flash **Edgi-Talk_M33_S_Template** to ensure core initialization.
- To enable M55, open:

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

2.12. Edgi-Talk_WIFI Example Project

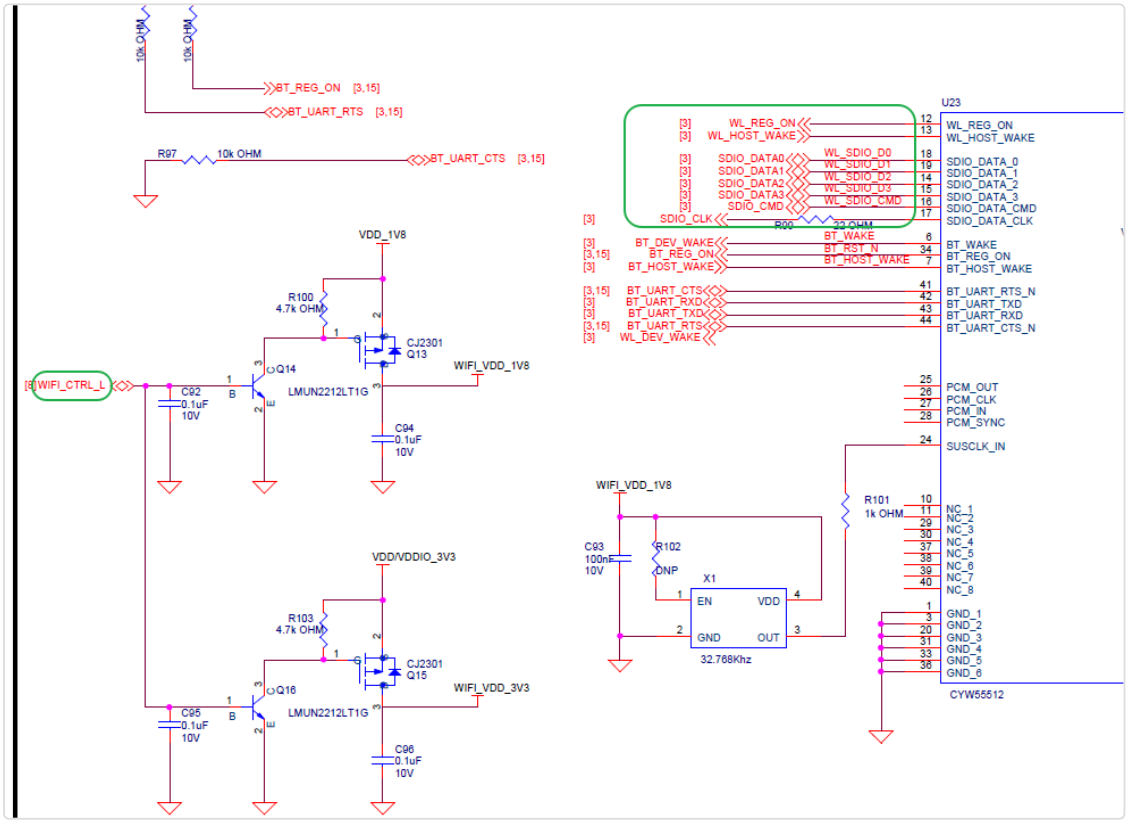
中文 | English

Introduction

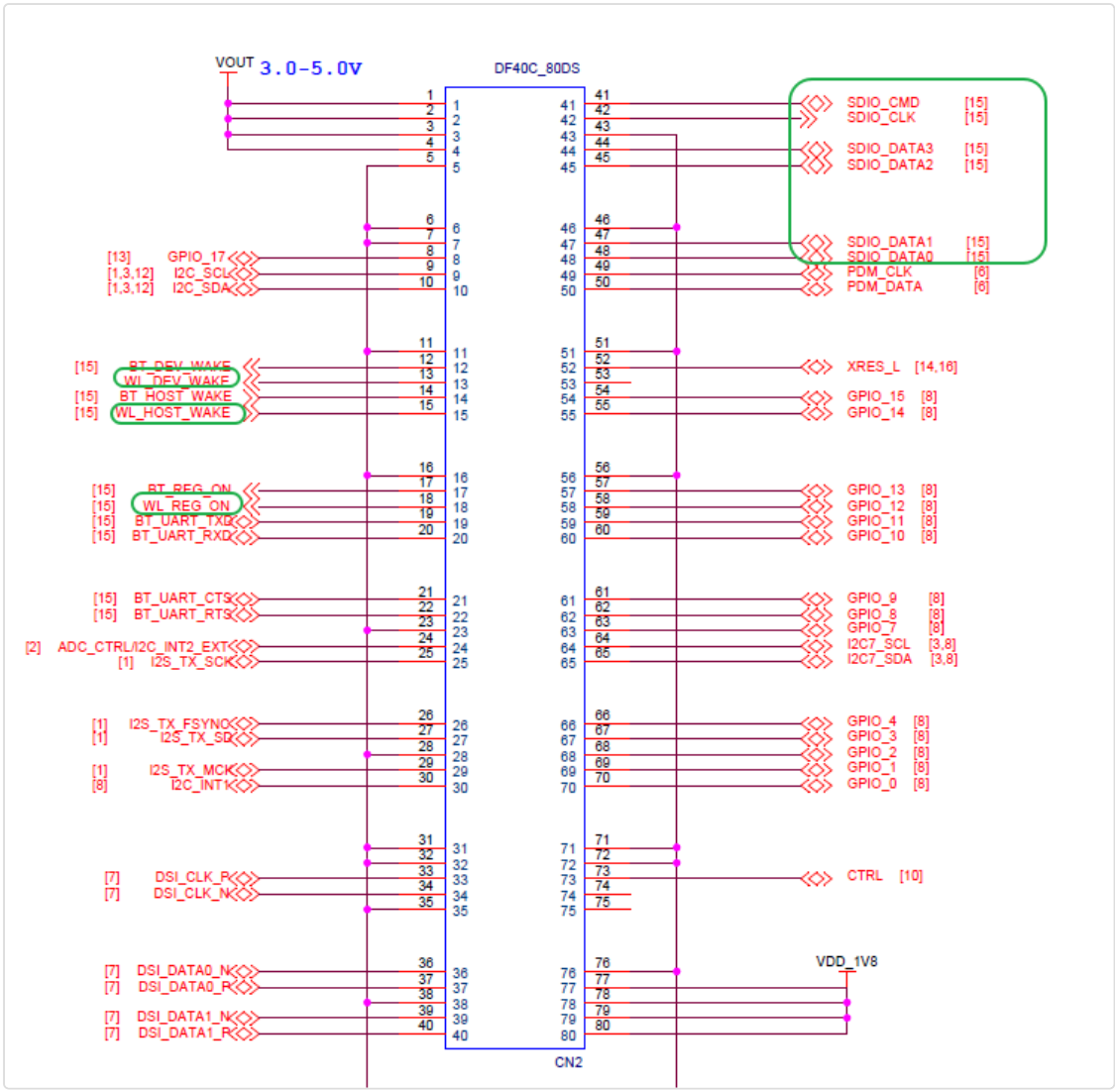
This example demonstrates **Wi-Fi functionality** on the **M55 core** using **RT-Thread RTOS**.
It allows users to quickly test Wi-Fi scanning, connection, and performance, verifying the Wi-Fi module interface.

Hardware Overview

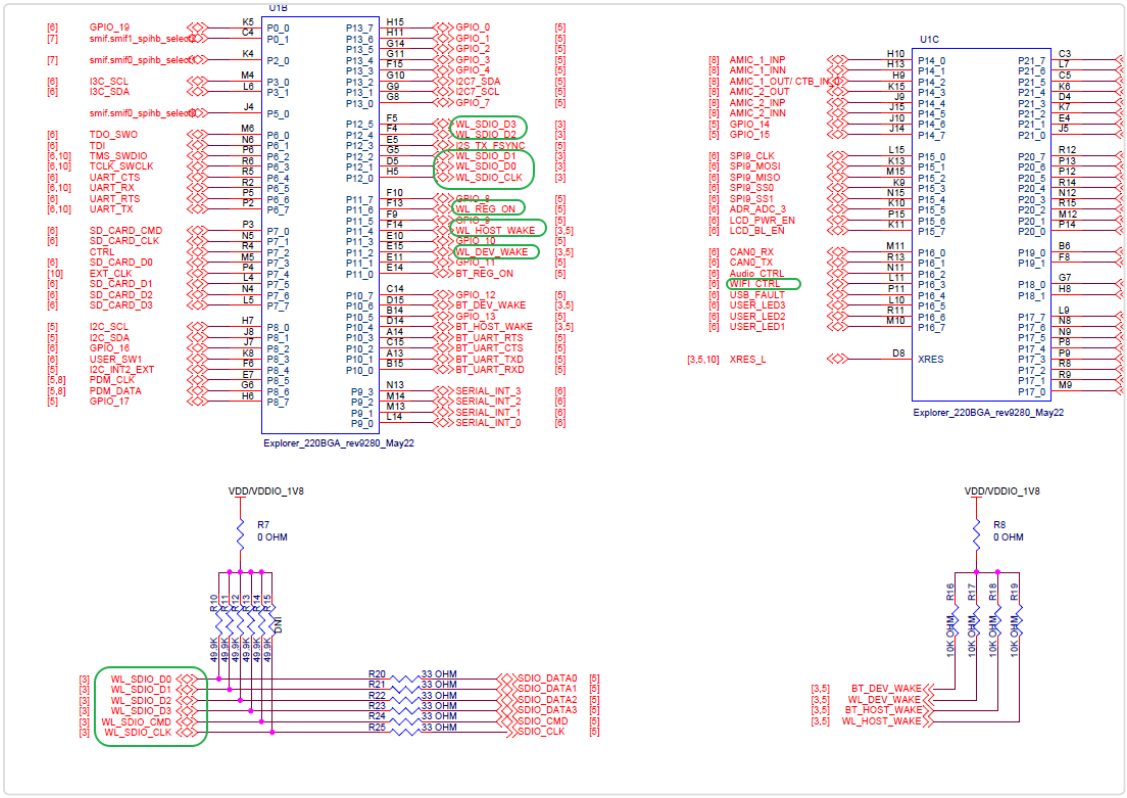
Wi-Fi Interface



BTB Socket


VOUT 3.0-5.0V
DF40C_80DS
CN2
SDIO_CMD [15]
SDIO_CLK [15]
SDIO_DATA3 [15]
SDIO_DATA2 [15]
SDIO_DATA1 [15]
SDIO_DATA0 [15]
PDM_CLK [6]
PDM_DATA [6]
[13] GPIO_17
[1,3,12] I2C_SCL
[1,3,12] I2C_SDA
[15] BT_DEV_WAKE
WL_DEV_WAKE
[15] BT_HOST_WAKE
[15] WL_HOST_WAKE
XRES_L [14,16]
GPIO_15 [8]
GPIO_14 [8]
[15] BT_REG_ON
WL_REG_ON
[15] BT_UART_TXD
[15] BT_UART_RXD
GPIO_13 [8]
GPIO_12 [8]
GPIO_11 [8]
GPIO_10 [8]
[15] BT_UART_CTS
[15] BT_UART_RTS
[2] ADC_CTRL/I2C_INT2_EXT
[1] I2S_TX_SCK
GPIO_9 [8]
GPIO_8 [8]
GPIO_7 [8]
I2C7_SCL [3,8]
I2C7_SDA [3,8]
[1] I2S_TX_FSYNC
[1] I2S_TX_SD
[1] I2S_TX_MCK
[8] I2C_INT
GPIO_4 [8]
GPIO_3 [8]
GPIO_2 [8]
GPIO_1 [8]
GPIO_0 [8]
[7] DSI_CLK_P
[7] DSI_CLK_N
CTRL [10]
[7] DSI_DATA0_N
[7] DSI_DATA0_P
[7] DSI_DATA1_N
[7] DSI_DATA1_P
VDD_1V8

MCU Interface



Software Description

- Developed on **Edgi-Talk** platform.
- Example features:
- Wi-Fi scanning
- Wi-Fi connection
- Iperf performance test
- Provides a clear example of **Wi-Fi driver integration with RT-Thread**.

Usage

Build and Download

1. Open and compile the project.
2. Connect the board USB to PC via **DAP**.

3. Flash the compiled firmware.

Running Result

- After power-on, the system initializes the Wi-Fi device.
- Connect to a Wi-Fi network via serial terminal:

```
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 <SSID> <PASSWORD>
```

```
msh />wifi join TEST 88888888
[I/WLAN.mgnt] 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
```

- After connection, perform throughput test with iperf.
- A GUI tool (`jperf`) is provided under `packages/netutils-latest/tools`.
- Extract `jperf.rar` and run the `.bat` file to launch the tool.
- Start iperf test from the board (replace `<PC_IP>` with actual PC IP):

```
iperf -c <PC_IP>
```

- Prefer 2.4 GHz network for testing (can use PC hotspot).

Notes

- Ensure the Wi-Fi module is correctly connected.
- Serial terminal commands allow scanning, joining, and testing Wi-Fi.

Startup Sequence

```
+------------------+
|   Secure M33     |
|  (Secure Core)   |
+------------------+
          |
          v
+------------------+
|       M33        |
| (Non-Secure Core)|
+------------------+
          |
          v
+------------------+
|       M55        |
| (Application Core)|
+------------------+
```

⚠️ Flash in this order strictly.

- If the example does not run, first compile and flash **Edgi-Talk_M33_S_Template** and **Edgi-Talk_M33_Template**.
- To enable M55:

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

3. Example

3.1. Edgi-Talk_M55_CoreMark Example Project

明白了，以下是更新为你指定格式的英文版本👇

中文 | **English**

Introduction

This example project is based on the **Edgi-Talk platform**, demonstrating the **CoreMark benchmark running on the M55 core** under the **RT-Thread real-time operating system**.
Through this project, users can quickly verify the performance of the M55 core and understand how the multi-core coprocessor operates under a real-time OS environment.

About CoreMark

CoreMark is a standardized embedded CPU benchmark developed by *EEMBC (Embedded Microprocessor Benchmark Consortium)*.
It is primarily used to measure the **core computational performance** of a microcontroller or processor, independent of specific hardware peripherals.

Test Contents

CoreMark evaluates CPU performance through four representative algorithm categories:

- **List processing**
- **Matrix operations**
- **State machine**
- **CRC (Cyclic Redundancy Check)**

Test Results

The output is represented as **CoreMark/MHz** or **CoreMark**, which allows performance comparison between different processors or compiler optimization levels.

Features

- Open source, portable, and lightweight
- Repeatable and verifiable results
- Focused on CPU integer computation capability

Software Description

- The project is developed based on the **Edgi-Talk** platform.
- The example includes:
- Running CoreMark benchmark on the M55 core
- Printing benchmark results through UART
- The project has a clean structure, making it easy to understand the M55 startup process and performance testing method.

Usage

Build and Download

1. Open the project and compile it.
2. Connect the board’s USB interface to your PC using the **onboard debugger (DAP)**.
3. Use the programming tool to flash the generated firmware to the development board.

Running Result

- After flashing, power on the board to start RT-Thread.
- The following serial output indicates the system started successfully:

```
 \ | /
- 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
```

- Then, manually enter the following command in the serial terminal:

```
core_mark
```

- The system will start the CoreMark test and print the benchmark results, for example:

```
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]crclist       : 0xe714
[0]crcmatrix     : 0x1fd7
[0]crcstate      : 0x8e3a
[0]crcfinal      : 0xcc42
Correct operation validated. See README.md for run and reportin
CoreMark 1.0 : 1200 / GCC10.2.1 20201103 (release)  / STACK
```

Notes

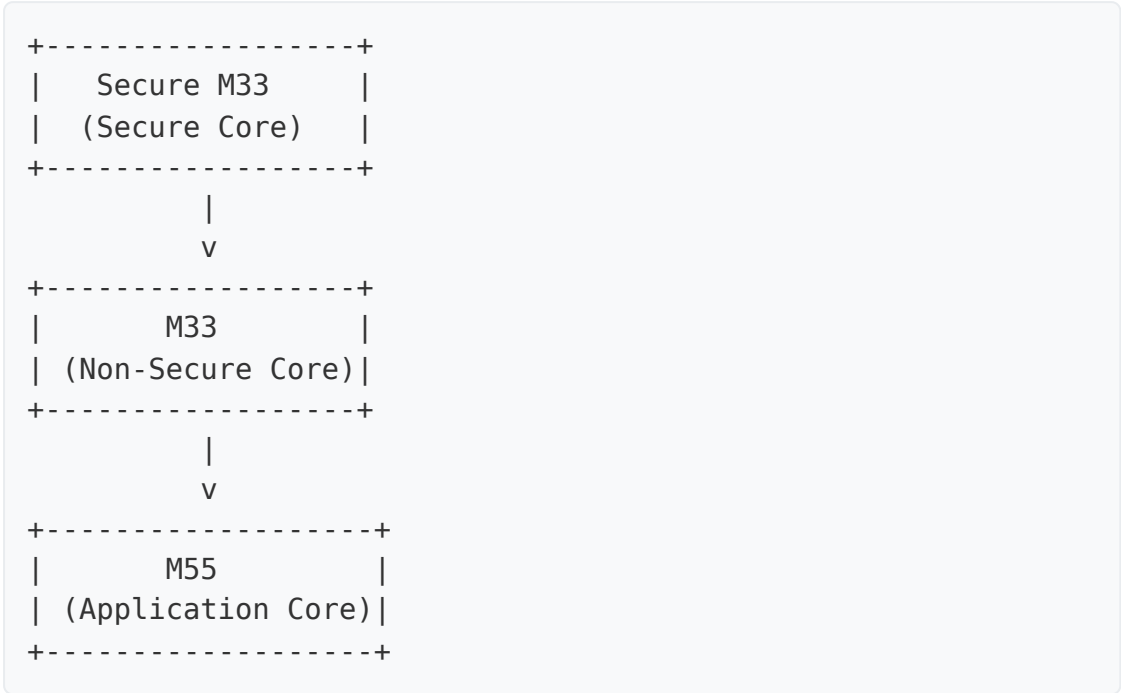
- To modify the **graphical configuration** of the project, open the configuration file using the following tool:

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

- After modification, save the configuration and regenerate the code.

Startup Sequence

The system starts in the following order:



⚠️ Please strictly follow the above flashing sequence; otherwise, the system may fail to run properly.

- If the example does not run correctly, first compile and flash the **Edgi-Talk_M33_S_Template** project to ensure proper initialization and core startup, then run this example.
- To enable the M55 core, enable the following configuration in the **M33 project**:

```
RT-Thread Settings --> Hardware --> select SOC Multi Core Mod
```

3.2. Edgi-Talk_M55_LVGL Example Project

中文 | **English**

Introduction

This example is based on the **Edgi-Talk platform**, demonstrating the **LVGL stress demo** running on **RT-Thread real-time operating system**.
It allows users to quickly verify the board-level **LCD display driver** and the **LVGL graphics framework** porting, providing a reference for future GUI application development.

LVGL Overview

LVGL (Light and Versatile Graphics Library) is an open-source embedded GUI development framework designed for resource-constrained devices. It provides modern graphical interfaces with optimized CPU and memory usage, running efficiently on both low-end MCUs and more powerful MPU platforms.

Key Features

1. **Lightweight**
   Optimized for minimal memory and CPU usage, ideal for low-power devices and resource-constrained environments.
2. **Cross-platform**
   Runs on multiple operating systems (FreeRTOS, RT-Thread, Zephyr, Linux) or bare-metal platforms. Only requires display and input drivers to be ported.
3. **Rich Widgets**
   Includes buttons, labels, sliders, charts, tables, lists, etc., and allows custom widget extensions.
4. **Advanced Rendering**
   Supports anti-aliasing, transparency, gradients, shadows, rounded corners, and animations for modern UIs.
5. **Input Device Support**
   Supports touchscreens, capacitive touch, mouse, keyboard, encoder, and multi-touch. Events are unified via LVGL’s event system.

6. **Internationalization**
   UTF-8 encoding with support for bidirectional text (e.g., Arabic, Hebrew).
7. **Extensibility**
   Flexible themes, styles, and integration with file systems and image decoders.

Applications

LVGL is widely used in:

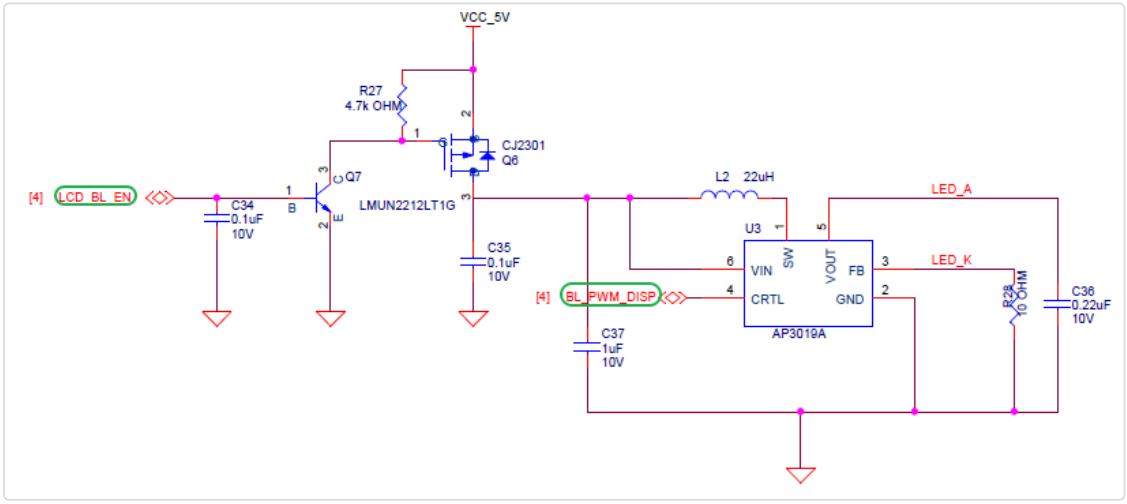
- Consumer electronics (smart home panels, smartwatches, appliances)
- Industrial HMI and instrumentation
- Automotive displays (central console, passenger screen, instrument cluster)
- Medical devices (portable monitors, handheld instruments)

Ecosystem & Community

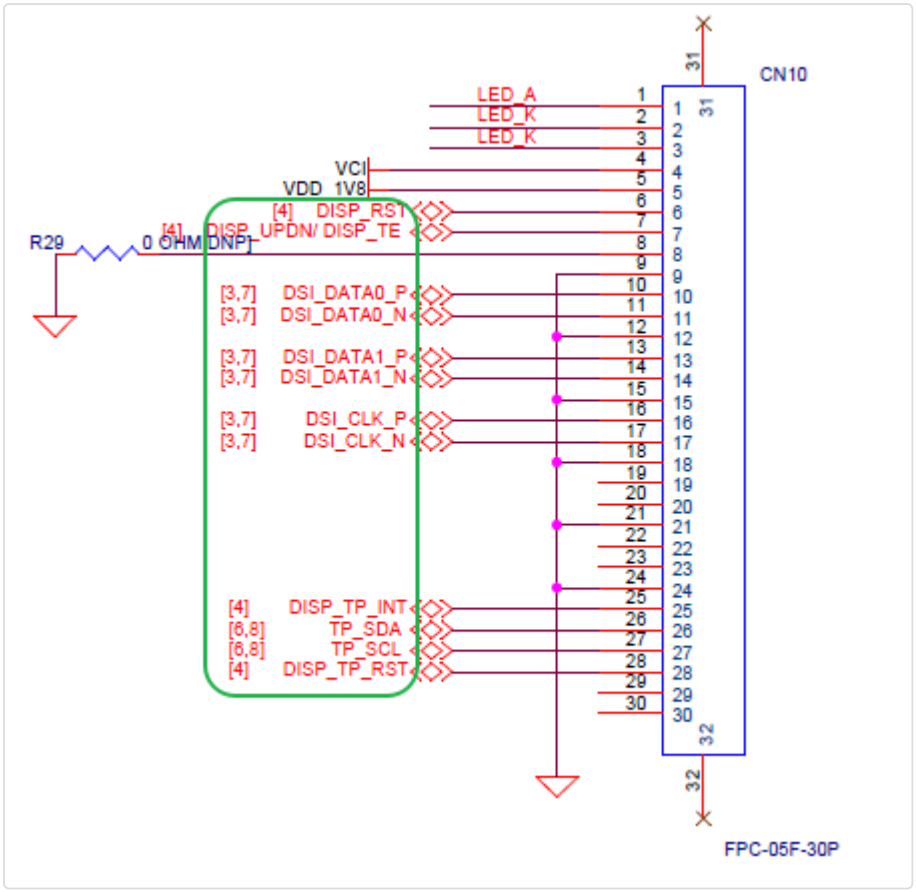
LVGL is **MIT licensed** and supported by **SquareLine Studio** for GUI design and **LVGL Simulator** for PC-based development. A large community provides open-source widgets, themes, and porting examples.

Hardware Description

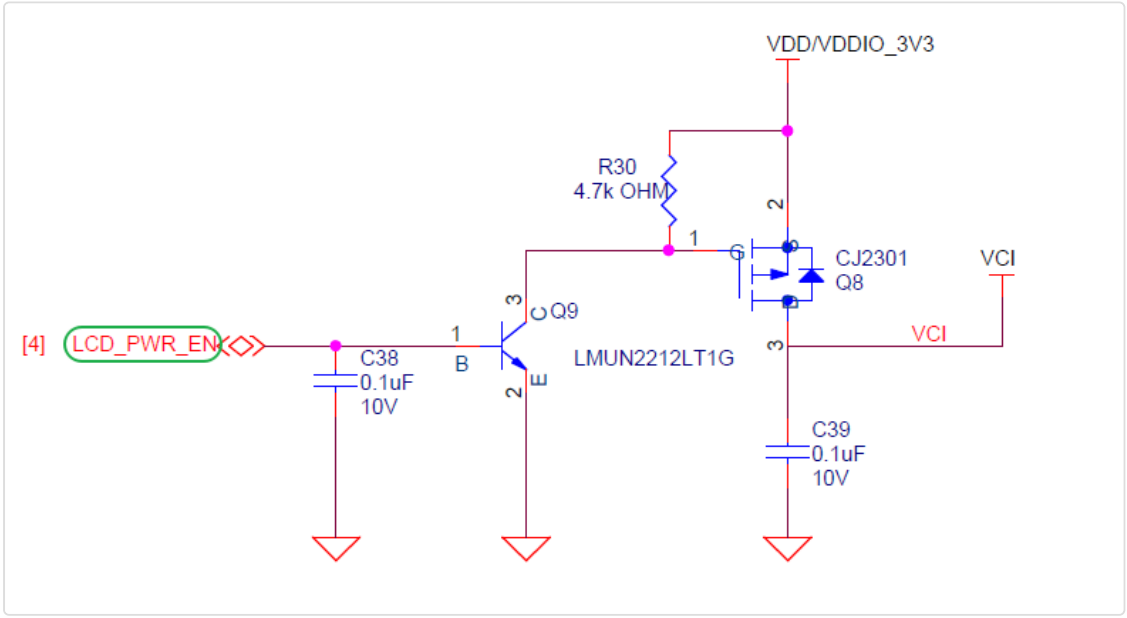
Backlight Interface



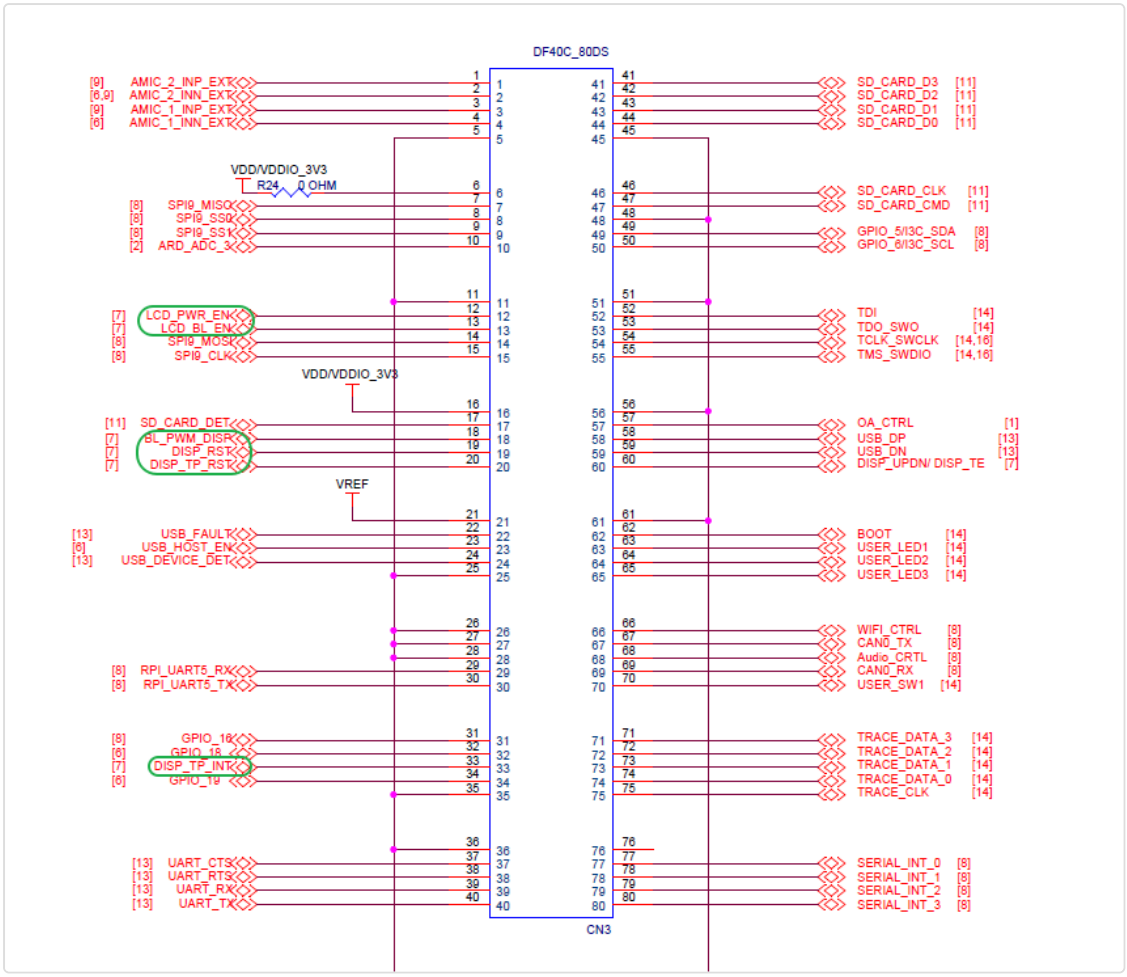
MIPI Interface

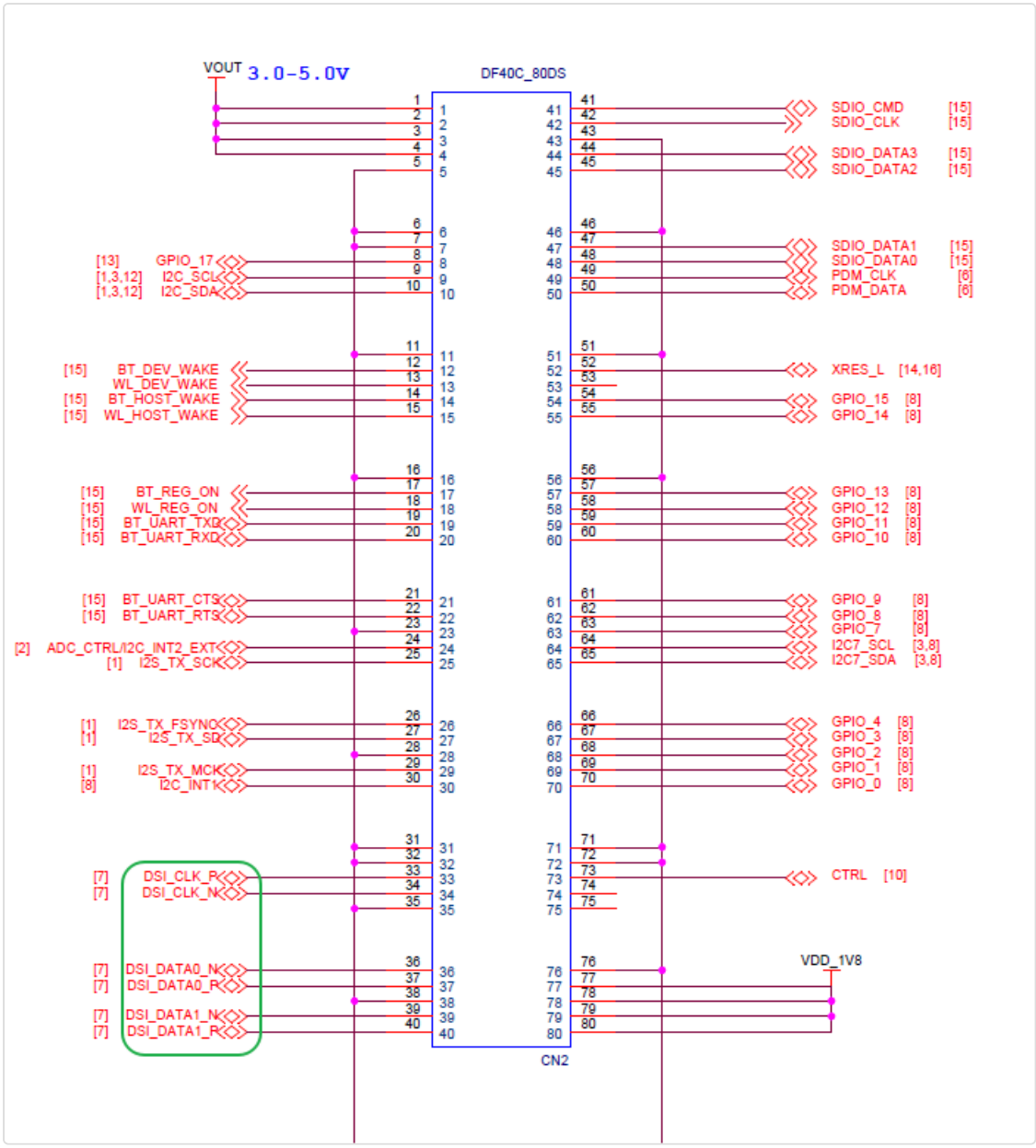


PWR Interface

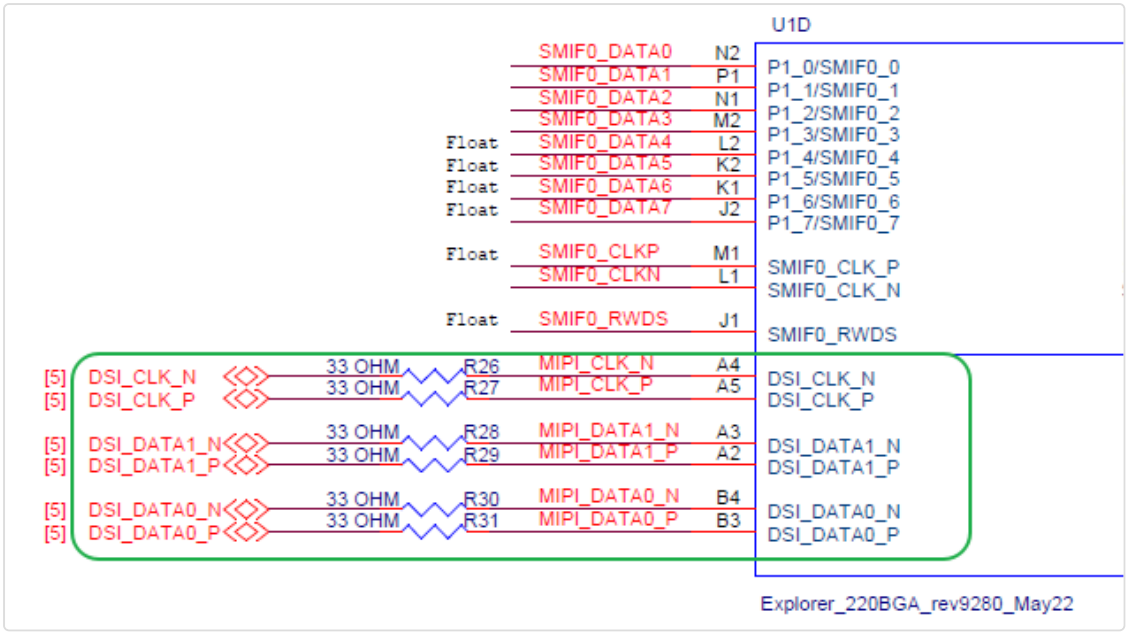


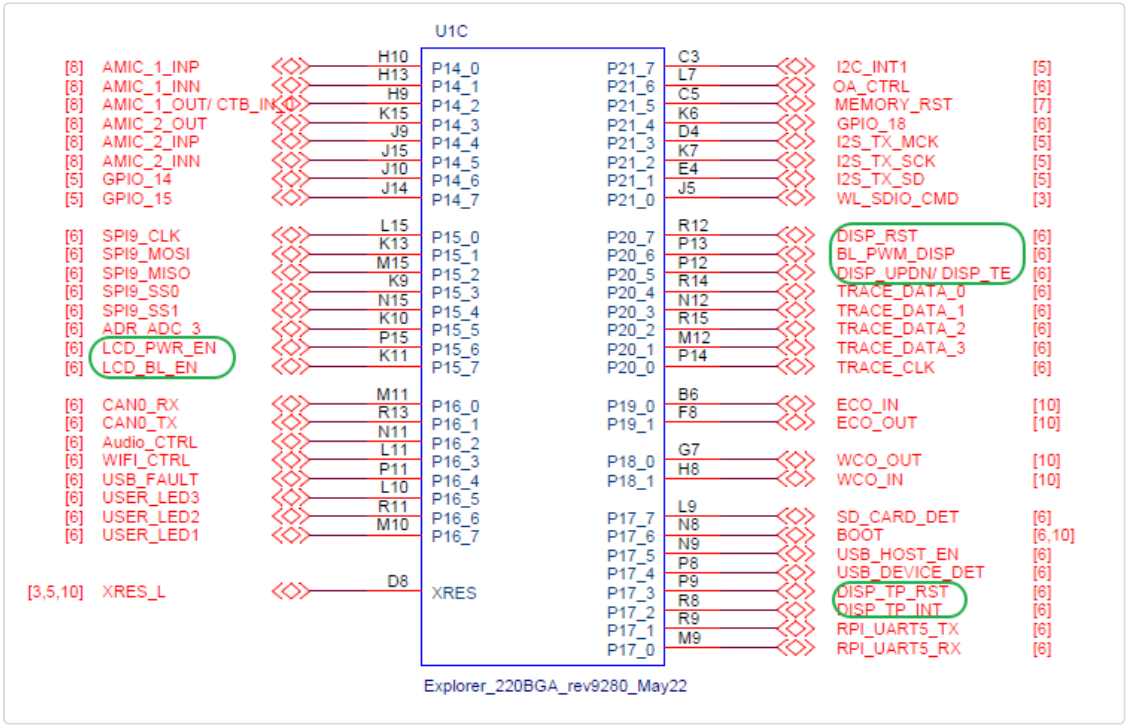
BTB Socket


DF40C_80DS
CN3
VDD/VDDIO_3V3
R24 0 OHM
VDD/VDDIO_3V3
VREF
[9] AMIC_2_INP_EXT
[6,9] AMIC_2_INN_EXT
[9] AMIC_1_INP_EXT
[6] AMIC_1_INN_EXT
[8] SPI9_MISO
[8] SPI9_SS0
[8] SPI9_SS1
[2] ARD_ADC_3
[7] LCD_PWR_EN
[7] LCD_BL_EN
[8] SPI9_MOSI
[8] SPI9_CLK
[11] SD_CARD_DET
[7] BL_PWM_DISP
[7] DISP_RST
[7] DISP_TP_RST
[13] USB_FAULT
[6] USB_HOST_EN
[13] USB_DEVICE_DET
[8] RPI_UART5_RX
[8] RPI_UART5_TX
[8] GPIO_16
[6] GPIO_18
[7] DISP_TP_INT
[6] GPIO_19
[13] UART_CTS
[13] UART_RTS
[13] UART_RX
[13] UART_TX
SD_CARD_D3 [11]
SD_CARD_D2 [11]
SD_CARD_D1 [11]
SD_CARD_D0 [11]
SD_CARD_CLK [11]
SD_CARD_CMD [11]
GPIO_5/I3C_SDA [8]
GPIO_6/I3C_SCL [8]
TDI [14]
TDO_SWO [14]
TCLK_SWCLK [14,16]
TMS_SWDIO [14,16]
OA_CTRL [1]
USB_DP [13]
USB_DN [13]
DISP_UPDN/ DISP_TE [7]
BOOT [14]
USER_LED1 [14]
USER_LED2 [14]
USER_LED3 [14]
WIFI_CTRL [8]
CAN0_TX [8]
Audio_CRTL [8]
CAN0_RX [8]
USER_SW1 [14]
TRACE_DATA_3 [14]
TRACE_DATA_2 [14]
TRACE_DATA_1 [14]
TRACE_DATA_0 [14]
TRACE_CLK [14]
SERIAL_INT_0 [8]
SERIAL_INT_1 [8]
SERIAL_INT_2 [8]
SERIAL_INT_3 [8]


VOUT 3.0-5.0V
DF40C_80DS
CN2
SDIO_CMD [15]
SDIO_CLK [15]
SDIO_DATA3 [15]
SDIO_DATA2 [15]
SDIO_DATA1 [15]
SDIO_DATA0 [15]
PDM_CLK [6]
PDM_DATA [6]
XRES_L [14,16]
GPIO_15 [8]
GPIO_14 [8]
GPIO_13 [8]
GPIO_12 [8]
GPIO_11 [8]
GPIO_10 [8]
GPIO_9 [8]
GPIO_8 [8]
GPIO_7 [8]
I2C7_SCL [3,8]
I2C7_SDA [3,8]
GPIO_4 [8]
GPIO_3 [8]
GPIO_2 [8]
GPIO_1 [8]
GPIO_0 [8]
CTRL [10]
VDD_1V8
[13] GPIO_17
[1,3,12] I2C_SCL
[1,3,12] I2C_SDA
[15] BT_DEV_WAKE
WL_DEV_WAKE
[15] BT_HOST_WAKE
[15] WL_HOST_WAKE
[15] BT_REG_ON
[15] WL_REG_ON
[15] BT_UART_TXD
[15] BT_UART_RXD
[15] BT_UART_CTS
[15] BT_UART_RTS
[2] ADC_CTRL/I2C_INT2_EXT
[1] I2S_TX_SCK
[1] I2S_TX_FSYNC
[1] I2S_TX_SD
[1] I2S_TX_MCK
[8] I2C_INT1
[7] DSI_CLK_P
[7] DSI_CLK_N
[7] DSI_DATA0_N
[7] DSI_DATA0_P
[7] DSI_DATA1_N
[7] DSI_DATA1_P

MCU Interface





Software Description

- Developed on the **Edgi-Talk platform**, running on the **M55 application core**.
- Example features:

- Initialize LVGL graphics library
- Run **lv_demo_stress** on the LCD
- Demonstrate rendering and performance testing
- Code structure is clear for understanding display driver integration and LVGL porting.

Usage

Build and Download

1. Open and compile the project.
2. Connect the board USB to the PC using the **onboard debugger (DAP)**.
3. Flash the generated firmware to the board.

Running Result

- After flashing and powering on, the system automatically runs **lv_demo_stress** on the LCD.
- Users can modify `applications/main.c` to switch to other LVGL demos (e.g., `lv_demo_widgets`, `lv_demo_music`).

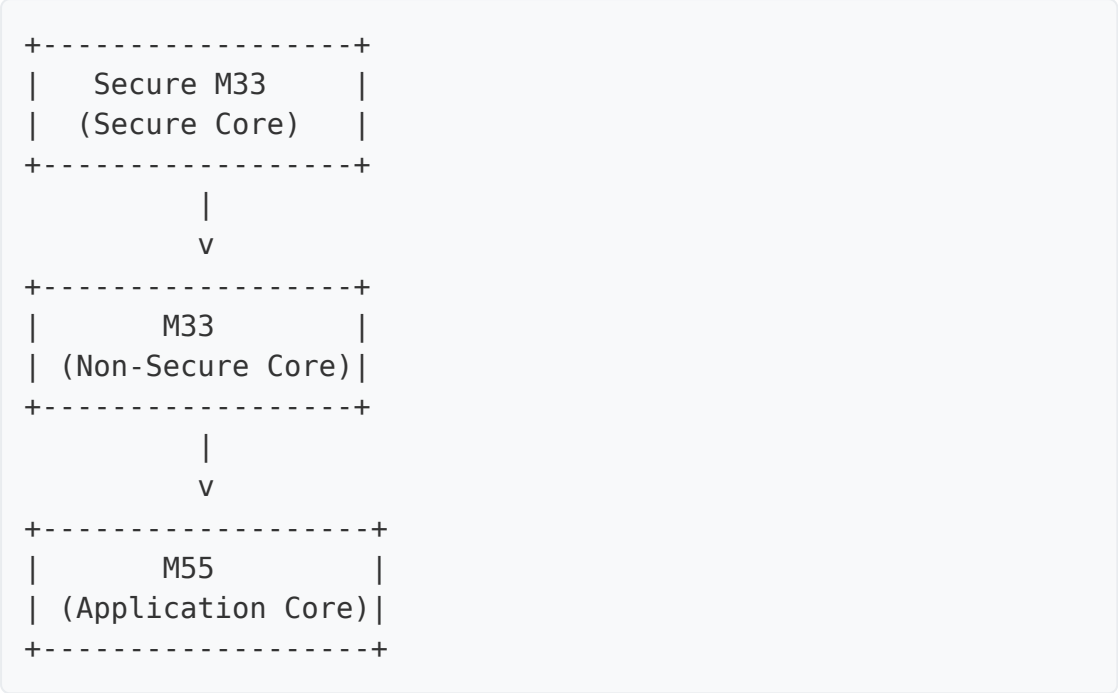
Notes

- To modify the **graphical configuration**, use:

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

- Save and regenerate code after modifications.
- If the screen shows no output, check:
- LCD connections and power supply
- `lv_port_disp.c` and `lv_port_indev.c` match the actual hardware

Startup Sequence



⚠️ Strictly follow the flashing order to ensure proper system operation.

- If the example fails, first flash **Edgi-Talk_M33_S_Template** to ensure proper initialization.
- To enable M55, configure in **M33 project**:

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

3.3. Edgi-Talk_WavPlayer Example Project

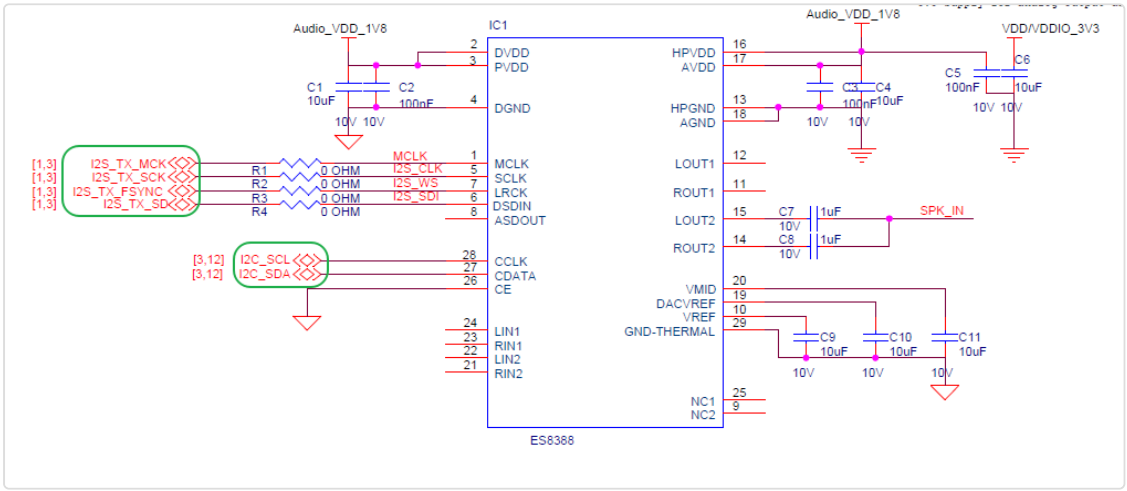
中文 | English

Introduction

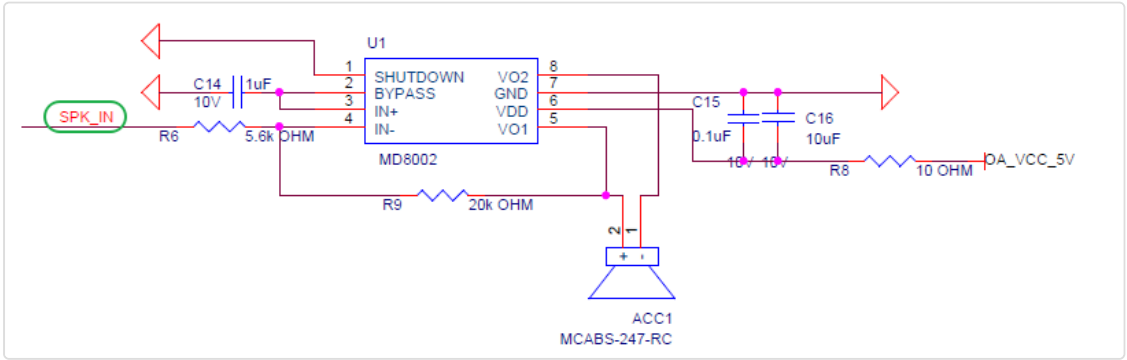
This example project demonstrates **WAV audio playback** on the **M33 core** with **RT-Thread RTOS**.
It allows users to experience WAV file parsing, playback mechanism, and verify audio decoding and driver interfaces.

Hardware Overview

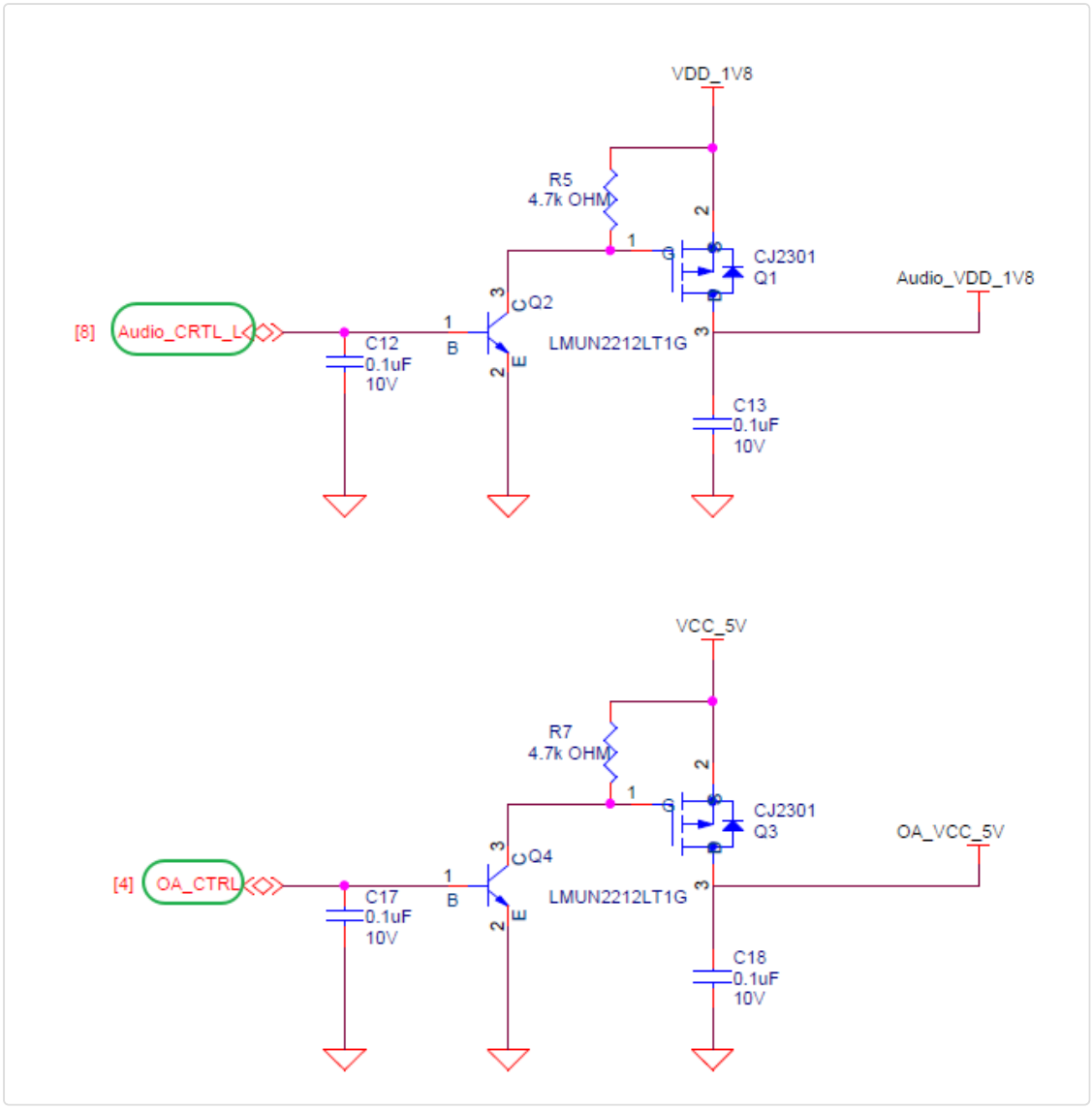
ES8388 Connection



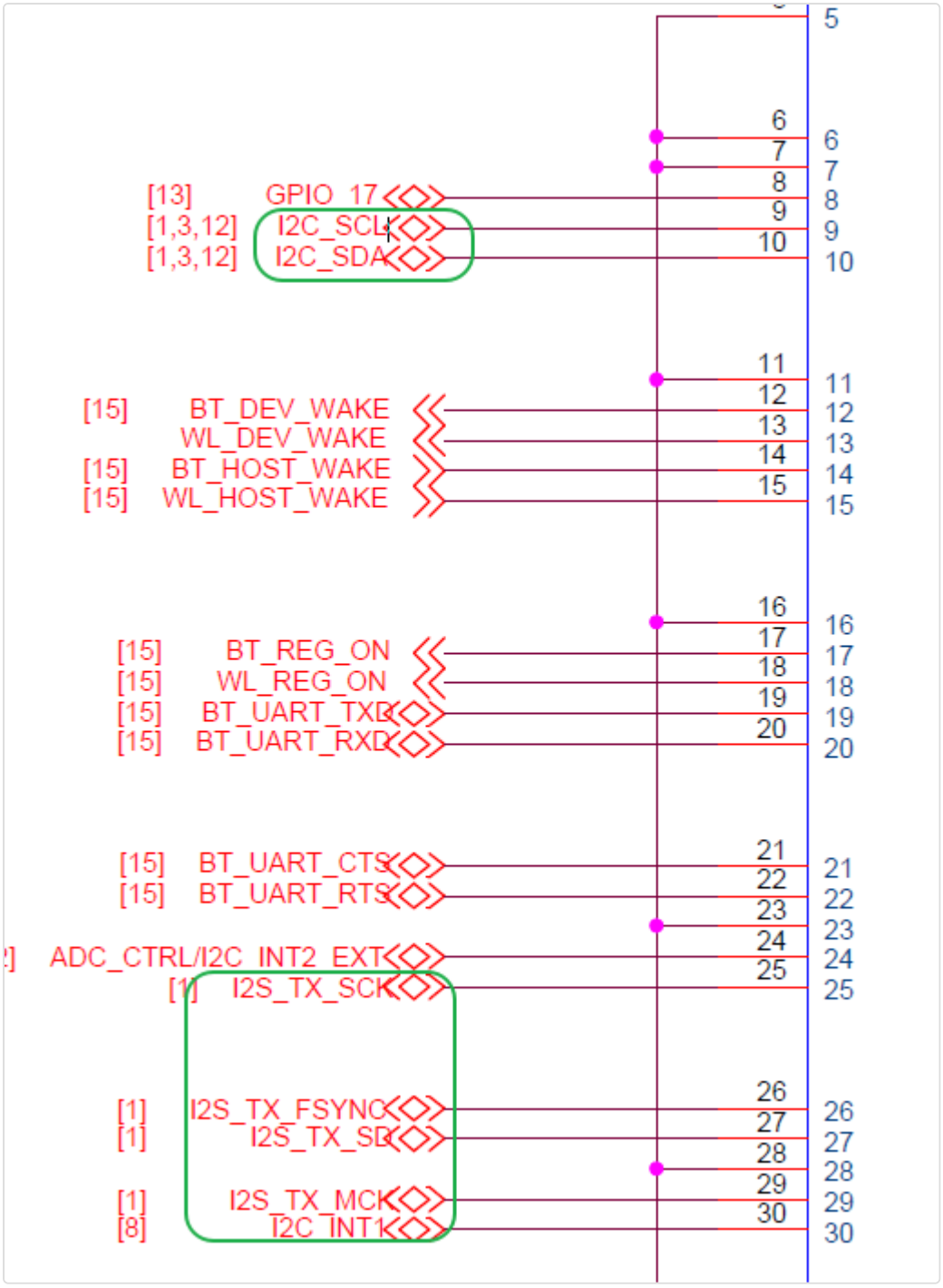
Speaker Interface



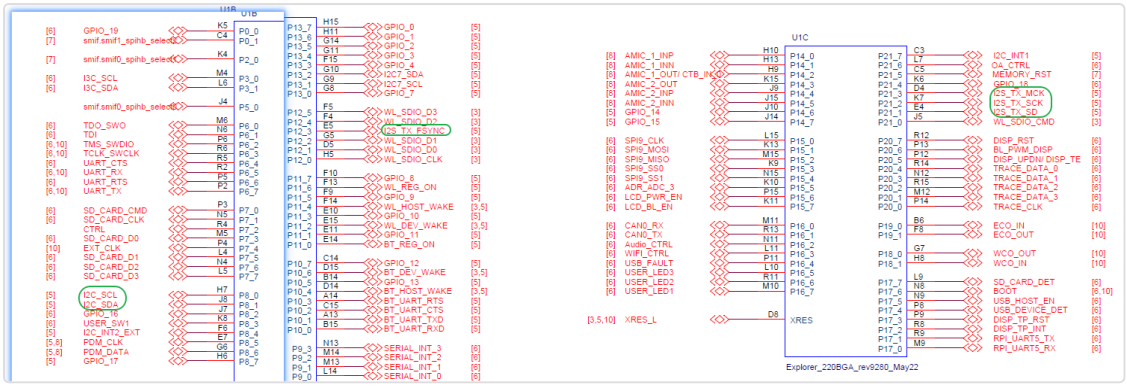
Control Pins



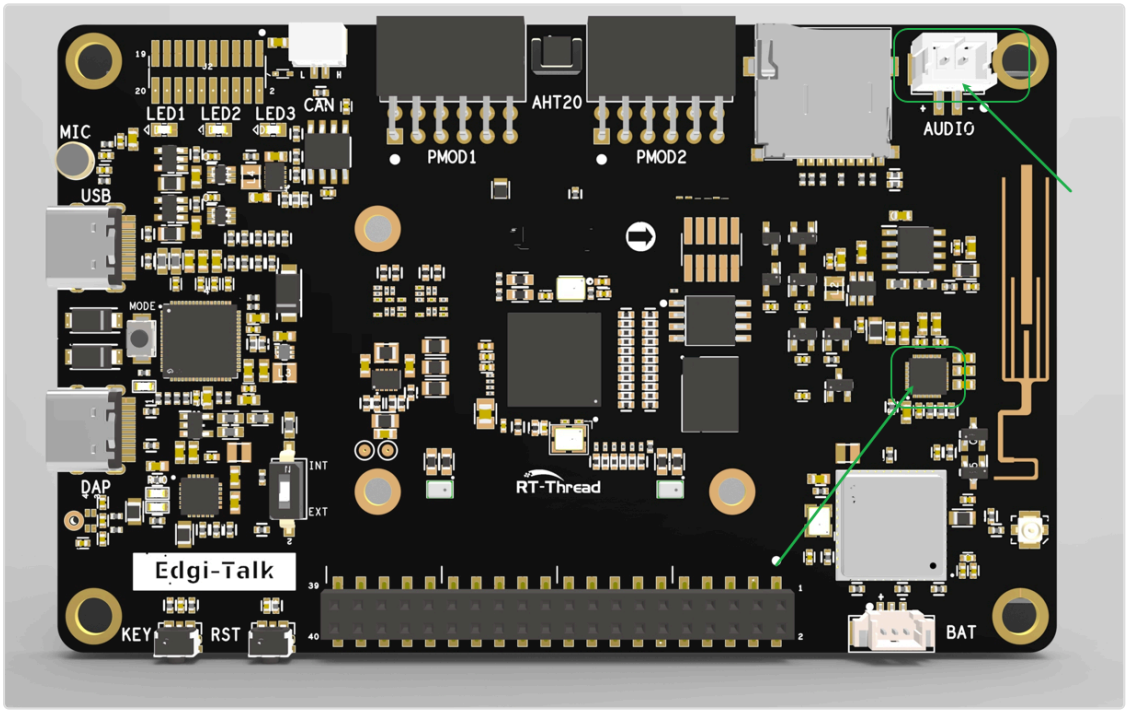
BTB Socket


[13] GPIO_17
[1,3,12] I2C_SCL
[1,3,12] I2C_SDA
[15] BT_DEV_WAKE
WL_DEV_WAKE
[15] BT_HOST_WAKE
[15] WL_HOST_WAKE
[15] BT_REG_ON
[15] WL_REG_ON
[15] BT_UART_TXD
[15] BT_UART_RXD
[15] BT_UART_CTS
[15] BT_UART_RTS
ADC_CTRL/I2C_INT2_EXT
[1] I2S_TX_SCK
[1] I2S_TX_FSYNC
[1] I2S_TX_SD
[1] I2S_TX_MCK
[8] I2C_INT1
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30

MCU Interface



Physical Board Layout



Software Description

- Developed on **Edgi-Talk** platform.
- Example features:
- WAV file parsing and playback
- Audio output via onboard DAC or audio peripherals
- Supports **PCM16** WAV files

- Sample rates: **16 kHz, 24 kHz, 48 kHz**
- Stereo output
- Playback status printed to serial console
- Provides a clear example of **audio playback driver integration with RT-Thread filesystem**.

Usage

Build and Download

1. Open and compile the project.
2. Connect the board USB to PC via **DAP**.
3. Flash the compiled firmware.
4. Copy WAV files to SD card or external storage root directory, e.g., `16000.wav`.

Running Result

- After power-on, the system initializes I2C, I2S audio devices, and mounts storage.
- Start playback via serial terminal:

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

- Sample serial output:

```
 \ | /
- 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 />
```

- Playback status, sample rate, channels, and bit width are displayed on the serial console.

Notes

- WAV files must be **PCM16** format, **16 kHz, 24 kHz, or 48 kHz**, stereo output.
- To modify graphical configuration:

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

- Save changes and regenerate code.
- Ensure storage is properly inserted and mounted, otherwise playback will fail.

Startup Sequence

```
+------------------+
|   Secure M33     |
|  (Secure Core)   |
+------------------+
          |
          v
+------------------+
|       M33        |
| (Non-Secure Core)|
+------------------+
          |
          v
+------------------+
|       M55         |
```

```
| (Application Core)|
+-------------------+
```

⚠️ Flash in this order strictly.

- If the example does not run, first compile and flash **Edgi-Talk_M33_S_Template**.
- To enable M55:

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

3.4. XiaoZhi Example Project

中文 | **English**

Introduction

This example demonstrates the **basic functionality of XiaoZhi voice interaction device** on the **Edgi-Talk platform**, running **RT-Thread RTOS**.
It allows users to quickly test Wi-Fi connection, key wake-up, and voice interaction, serving as a foundation for further application development.

Software Description

- Developed on **Edgi-Talk** platform.
- Example features:
- Wi-Fi connection and status display
- Key wake-up and voice interaction
- Device state management (standby, listening, sleep)

Usage

WIFI Modification

1. In `main.c` at line 36, locate the following code:

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

2. Replace "TEST" with your WiFi name and "88888888" with the password, then recompile and flash.
3. For detailed WiFi usage, refer to: **WIFI**

Build and Download

1. Open and compile the project.
2. Connect the board USB to PC via **DAP**.

3. Flash the compiled firmware to the device.

Running Result

- After power-on, the device runs the example automatically.
- The screen displays the current status:
- **Connecting** – connecting to Wi-Fi
- **On standby** – idle mode
- **Listening** – ready for voice interaction
- **Sleeping** – low-power sleep mode
- Press the top button to enter **Listening** mode for voice interaction.

Notes

- First-time setup requires backend binding via XiaoZhi Official Website.

- Ensure Wi-Fi SSID and password are correct and use **2.4 GHz** frequency.
- Device must have internet access.
- To modify graphical configurations, use:

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

- Save changes and regenerate code after modification.

Startup Sequence

```
+------------------+
|   Secure M33     |
|  (Secure Core)   |
+------------------+
          |
          v
+------------------+
|       M33        |
| (Non-Secure Core)|
+------------------+
          |
          v
+------------------+
|       M55        |
```

```
| (Application Core)|
+-------------------+
```

⚠️ Flash in this order strictly to ensure proper operation.

- If the example does not run, first compile and flash **Edgi-Talk_M33_S_Template**.
- To enable M55:

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