ISSN: 3065-1220 Volume 2, Number 2, April 2025

Hans汉斯

嵌入式技术与智能系统

Embedded Technology and Intelligent Systems

Qian Ru Shi Ji Shu Yu Zhi Neng Xi Tong 2025年4月2卷2期

https://www.hanspub.org/journal/etis

Editorial Board

编委名单

ISSN: 3065-1220

https://www.hanspub.org/journal/etis

Embedded Technology and Intelligent Systems, Vol.2, No.2, 59-132
嵌入式技术与智能系统, 2 卷 2 期
Published Online April 2025 in Hans (https://www.hanspub.org/journal/etis)

TABLE OF CONTENTS
目　　录

期刊信息
期刊中文名称：《嵌入式技术与智能系统》
期刊英文名称：**Embedded Technology and Intelligent Systems**
期刊缩写：**ETIS**
出刊周期：双月刊
语　　种：中文

出版机构：汉斯出版社(Hans Publishers, https://www.hanspub.org/)
编辑单位：《嵌入式技术与智能系统》编辑部
主　　编：何立民，北京航空航天大学教授
网　　址：https://www.hanspub.org/journal/etis

订阅信息
订阅邮箱：sub@hanspub.org
订阅价格：180 美元每年

广告服务
联系邮箱：adv@hanspub.org

Embedded Technology and Intelligent Systems 嵌入式技术与智能系统, 2025, 2(2), 59-77
Published Online April 2025 in Hans. https://www.hanspub.org/journal/etis
https://doi.org/10.12677/etis.2025.22006

Hans汉斯

基于Zephyr RTOS的嵌入式软件开发实践

何灵渊[1]，何小庆[2]

[1]博通公司，美国 森尼韦尔
[2]嵌入式系统联谊会，北京

收稿日期：2025年4月1日；录用日期：2025年4月23日；发布日期：2025年4月30日

摘　要

Zephyr开源项目由Linux基金会维护，是一个针对资源受限的嵌入式设备优化的小型、可缩放、多体系结构实时操作系统(RTOS)。近年来，Zephyr RTOS在嵌入式开发中的采用度逐步增加，支持的开发板和传感器不断增加，其广泛的设备支持和高度的可扩展性吸引了开发者的关注。相比FreeRTOS等小型RTOS而言，教育生态不够成熟的Zephyr系统规模更大，结构更复杂，这提高了开发者入门和精通的门槛。文章对Zephyr硬件抽象层和设备驱动的架构与实现进行系统性分析，重点阐述了设备驱动模型和设备树的作用。为了展示基于Zephyr的嵌入式软件开发，文章在BBC micro:bit V2开源硬件上构建样例Zephyr设备驱动和应用程序，并做解释和验证。

关键词

嵌入式系统，软件开发，实时操作系统，Zephyr项目

Embedded Software Development Practice with Zephyr RTOS

Lingyuan He[1], Xiaoqing He[2]

[1]Broadcom Inc., Sunnyvale, USA
[2]Embedded System Association, Beijing

Received: Apr. 1st, 2025; accepted: Apr. 23rd, 2025; published: Apr. 30th, 2025

Abstract

Zephyr, an open-source initiative managed by the Linux Foundation, is a small, scalable, multi architecture Real-time Operating System (RTOS) optimized for resource-constrained embedded systems. In recent years, the adoption rate of Zephyr RTOS has increased significantly. The range of supported

文章引用: 何灵渊, 何小庆. 基于 Zephyr RTOS 的嵌入式软件开发实践[J]. 嵌入式技术与智能系统, 2025, 2(2): 59-77.
DOI: 10.12677/etis.2025.22006

boards and sensors continues to rise. Developers are increasingly interested in Zephyr because of its board device support and scalability. Compared to a minimum RTOS like FreeRTOS, Zephyr, whose education infrastructure has not yet matured, has a larger scale and a more complex architecture. This means Zephyr's learning curve for a developer is steep. This paper systematically describes the architecture and implementation of Zephyr's hardware abstraction and device driver, especially its device driver model and its usage of the device tree. It also demonstrates embedded software development with Zephyr by building, analyzing, and verifying a custom driver and an example application based on the BBC micro:bit V2 open-source hardware.

Keywords

Embedded System, Software Development, Real-Time Operating System, Zephyr Project

1. 引言

Zephyr 是由 Linux 基金会管理的开源实时操作系统(RTOS) [1]，其前身为用于数字信号处理的 Virtuoso 操作系统，后被风河(Wind River)收购，更名为 Rocket RTOS。2016 年它成为了 Linux 基金会的项目，更名为 Zephyr。

Zephyr 得到了多家半导体企业的支持，包括恩智浦、意法半导体、瑞萨、北欧半导体(Nordic)、英特尔和德州仪器等，并已经被应用到了众多设备中，覆盖了消费电子、能源、医疗、工业、农业等领域[2]。Zephyr 的 Apache 2.0 开源协议授权让它在非商用和商用解决方案中都可免费使用[3]。

近年来 Zephyr 的热度逐渐上升，在嵌入式开发中的采用度逐步增加。Eclipse 基金会的《2024 年物联网和嵌入式开发者调查报告》表明，在资源受限设备上使用 Zephyr 的开发者从 2022 年的 8%增长到了 2024 年的 21%，这已经和裸机直接编程的比例相当，也非常接近第二位的 FreeRTOS (29%) [4]。

相比 FreeRTOS 等小型 RTOS 而言，教育生态不够成熟的 Zephyr 系统规模更大，结构更复杂，这提高了开发者入门和精通的门槛。本文对 Zephyr 硬件抽象层和设备驱动的架构与实现进行系统性分析，重点阐述了设备驱动模型和设备树的作用。为了展示基于 Zephyr 的嵌入式软件开发，本文在 BBC micro:bit V2 开源硬件上构建样例 Zephyr 设备驱动和应用程序，并做解释和验证。

2. Zephyr 的硬件抽象层和配置概述

Zephyr 有着完善的设备驱动支持，而且高度可配置。作为 Linux 基金会的项目，它用到了和 Linux 内核类似的工具，特别是设备树(Device Tree)和 Kconfig 配置语言。本章将对与开发息息相关的硬件抽象化和配置进行概述。

2.1. 设备驱动模型

Zephyr 的设备驱动模型负责初始化系统中所有的驱动程序，为系统中的所有设备驱动提供了统一的配置方法[5]。如图 1 所示的是设备驱动模型的概览。

Zephyr 中每一种子系统驱动(UART、I2C 等)都有着泛用类型(Generic Type，非设备特定)的接口，具体的驱动实现会提供实现这些驱动接口函数的指针。在图 1 中可以看到，在子系统 2 中有两种设备驱动

的实例，但是两种驱动都会提供泛用 API 1 到 3 的实现。应用程序代码可以在兼容的设备上直接使用泛用 API，具体驱动的实现代码会被调用。如子系统 1 中所示，同一种驱动可以在系统中多次实例化，比如多个 UART 接口。

设备驱动代码在初始化时也会为每个设备提供驱动特定的配置，即图 1 中的 struct config。在实际代码中这可能是通过 Kconfig 配置的参数，比如显示器的刷新频率。驱动代码还可以为每个驱动指定一个结构用于存储相关的数据。

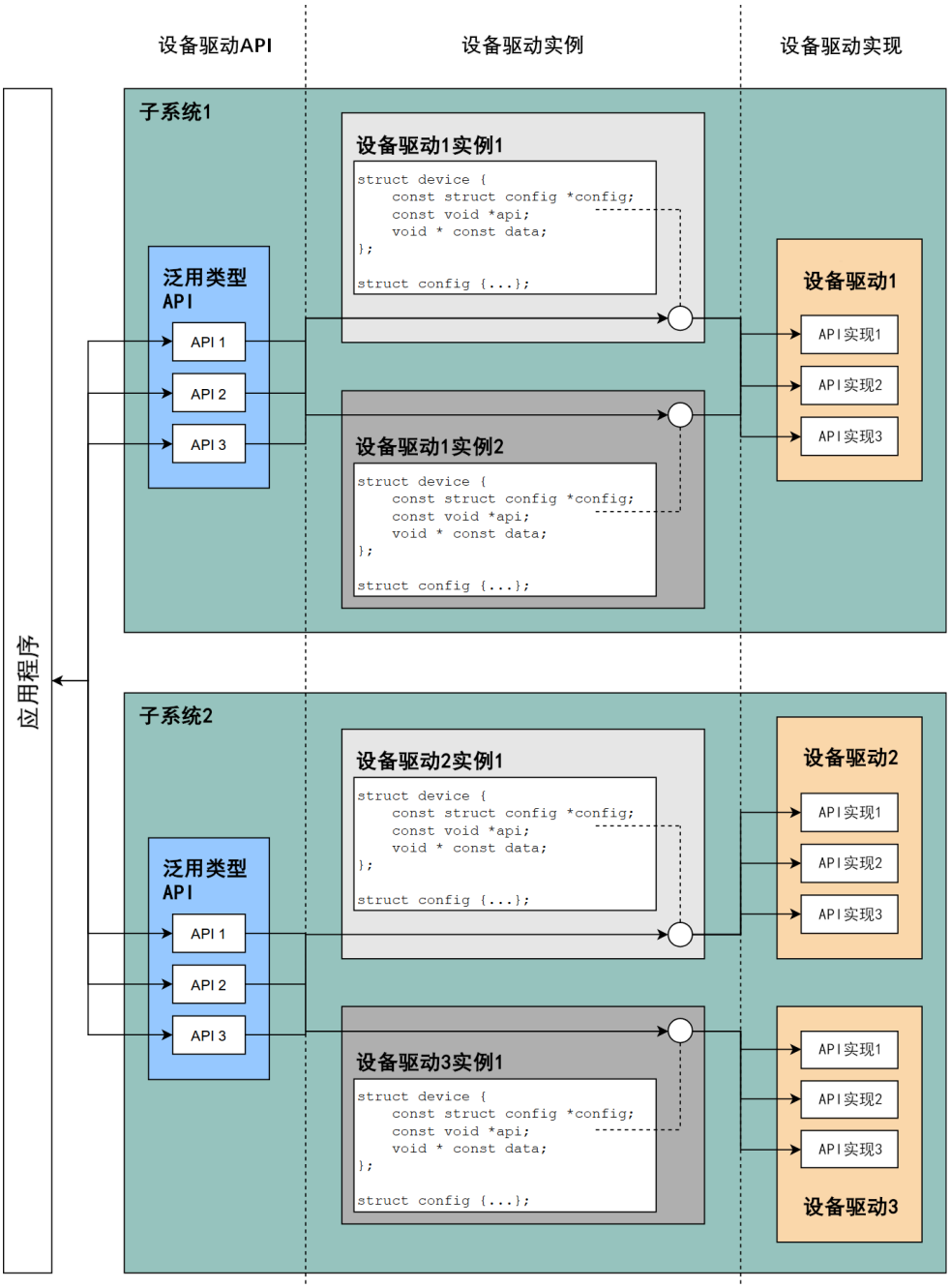

Figure 1. An overview of the device driver model (source: zephyrproject.org)
图 1. 设备驱动模型概览(来源：zephyrproject.org)

一个驱动的泛用接口定义会出现在驱动的头文件中，图 2 中定义了 subsystem 子系统的泛用接口 subsystem_do_this 和 subsystem_do_that 函数。图 3 中的 my_driver 驱动实现了自己的 do_this 和 do_that 函数，并将它们的指针填入了驱动 API 结构(do_this 和 do_that 成员)。注意应用程序代码应该直接使用 subsystem_do_this/that 函数，这两个函数会通过 DEVICE_API_GET 宏进入正确的驱动接口实现，即 my_driver_do_this/that 函数。在实际的驱动中，subsystem 会被替代为能够代表设备的名称，例如在通用的显示驱动接口(include/zephyr/drivers/display.h)中，subsystem 被替代为了 display。

```
typedef int (*subsystem_do_this_t)(const struct device *dev, int foo, int bar);
typedef void (*subsystem_do_that_t)(const struct device *dev, void *baz);

__subsystem struct subsystem_driver_api {
      subsystem_do_this_t do_this;
      subsystem_do_that_t do_that;
};

static inline int subsystem_do_this(const struct device *dev, int foo, int bar)
{
      return DEVICE_API_GET(subsystem, dev)->do_this(dev, foo, bar);
}

static inline void subsystem_do_that(const struct device *dev, void *baz)
{
      DEVICE_API_GET(subsystem, dev)->do_that(dev, baz);
}
```

Figure 2. A sample driver interface definition (source: zephyrproject.org)
图 2. 样例驱动接口定义(来源：zephyrproject.org)

```
static int my_driver_do_this(const struct device *dev, int foo, int bar)
{
      ...
}

static void my_driver_do_that(const struct device *dev, void *baz)
{
      ...
}

static DEVICE_API(subsystem, my_driver_api_funcs) = {
      .do_this = my_driver_do_this,
      .do_that = my_driver_do_that,
};
```

Figure 3. A sample driver implementation (source: zephyrproject.org)
图 3. 样例驱动实现(来源：zephyrproject.org)

在进行具体子系统驱动的实例化时，驱动代码还会提供初始化代码和初始化的优先级。

2.2. 设备树

设备树(Device Tree)是用于描述硬件的层级化数据结构。设备树规范[6]描述了设备树的概念、用途、结构、设备树绑定(binding)和设备树语言。

2.2.1. 设备树的作用

Zephyr 和 Linux 同样使用设备树，Zephyr 为了减少运行时的数据和代码，会使用设备树的数据产生 C 语言头文件[7]。Zephyr 中定义了一整套宏，用于访问设备树节点和取得设备树节点的属性。

Zephyr 中设备树有两项主要作用：

- 在设备驱动模型中描述硬件。
- 提供硬件的初始配置。

设备树和 Kconfig 在 Zephyr 中都起到了配置语言的作用，设备树用于描述硬件和启动时的配置，Kconfig 则主要用于配置软件。

设备树有两种输入文件：设备树源文件和设备树绑定[8]。源文件描述了设备树本身，绑定则用于描述设备树的内容，特别是数据类型和结构。Zephyr 在构建时使用这两种文件生成 C 头文件，devicetree.h 头文件提供通用的宏访问设备树(以“DT_”打头)。

2.2.2. 设备树的语法

图 4 所示的是一个最小的样例设备树源文件[9]：

```
/dts-v1/;

/ {
        a-node {
                subnode_nodelabel: a-sub-node {
                        foo = <3>;
                };
        };
};
```

Figure 4. A minimum device tree file (source: zephyrproject.org)
图 4. 设备树最小样例(来源：zephyrproject.org)

图中“/”代表根节点，a-node 是根节点的子节点，a-sub-node 是 a-node 的子节点，a-sub-node 还有一个 label (标签) subnode_nodelabel。标签是可选的，在设备树中每个标签只能出现一次，代码可以通过标签直接访问节点。每个节点都有自己的路径，和 Linux 文件路径相似，例如 a-sub-node 的全路径为：/a-node/a-sub-node。

图 5 所示的是一个较为贴近实际硬件的设备树样例：

```
/dts-v1/;

/ {
        soc {
                i2c@40003000 {
                        compatible = "nordic,nrf-twim";
                        reg = <0x40003000 0x1000>;

                        apds9960@39 {
                                compatible = "avago,apds9960";
                                reg = <0x39>;
                        };
                        ti_hdc@43 {
                                compatible = "ti,hdc", "ti,hdc1010";
                                reg = <0x43>;
                        };
                        mma8652fc@1d {
                                compatible = "nxp,fxos8700", "nxp,mma8652fc";
                                reg = <0x1d>;
                        };
                };
        };
};
```

Figure 5. A complete device tree example (source: zephyrproject.org)
图 5. 一个完整的设备树样例(来源：zephyrproject.org)

在图 5 中可以看到节点名的命名方法为“总线类型或设备名@地址”，这样的惯例不仅有助于区分类似的节点，还能够帮助快速确定节点指向的设备和总线类型。地址的惯例根据设备类型有所不同：

- 在内存中映射的外设：使用寄存器映射的基地址，例如 i2c@40003000 表示 I2C 映射的寄存器基地址为 0x40003000。
- I2C 外设：使用外设在 I2C 总线上的地址，例如 apds9960 的 I2C 地址为 0x39。
- SPI 外设：使用外设的片选线序号，如果没有则使用 0。
- 内存：使用物理内存的起始地址，例如 memory@2000000 表示从 0x2000000 物理地址开始的 RAM。
- 在内存中映射的闪存：和 RAM 类似使用物理起始地址，例如 flash@8000000。
- 固定的闪存分区：使用分区的偏移量，例如在 flash@8000000 设备中可以有一个 partitions 节点代表分区表，其中有 partition@0 和 partition@20000 两个节点，分别意味着起始地址 0x8000000 和 0x8020000 的两个分区。

设备树节点中每个属性有一个名称和一个值，属性的值可以是字符串、整型数、布尔值、8 位整型数组、字符串数组、混合类型数组、指向节点的 phandle (类似 C 语言中的指针)、复数的 phandle 或是 phandle 数组。

设备树节点中几个重要的属性如下：

- compatible：表示节点所代表的硬件设备，本文翻译为兼容名。兼容名属性在构建过程中十分重要，驱动程序通过兼容名的值查找可以适配的硬件。兼容名的值可以是字符串数组，将数个驱动程序从最特定到最泛用进行排列，首个匹配的驱动程序会被加载。
- reg：用于设备寻址，其格式为 16 进制的<地址，长度>。
- status：用于表示节点是否启用。Zephyr 支持“okay”和“disabled”，分别表示启用和禁用。节点必须启用，Zephyr 的驱动模型才会应用到节点上。

除了标签，设备树源文件中还可以定义 chosen (选择)和 alias (别名)来帮助应用代码或驱动寻找特定的节点，如图 6 所示。

```
/dts-v1/;

/ {
    chosen {
            zephyr,console = &uart0;
    };

    aliases {
            my-uart = &uart0;
    };

    soc {
            uart0: serial@12340000 {
                    ...
            };
    };
};
```

Figure 6. Use chosen and aliases nodes in a device tree file (source: zephyrproject.org)
图 6. 在设备树中使用 chosen 和 aliases 节点(来源：zephyrproject.org)

图中/alias 和/chosen 节点都不指向实际的硬件设备，它们被用来指定设备树中的其他节点：my-uart是/soc/serial@12340000 路径的别名(uart0 标签名)，uart0 标签还被选为“zephyr, console”。选择和别名可以帮助抽象化不同的开发板，例如闪灯样例(samples/basic/blinky/src/main.c)中使用 led0 别称节点达到支持多种开发板的目的，只要开发板的设备树文件中有别称为 led0 的节点，样例即可运行。

Zephyr 中每个支持的开发板都有自己的主设备树文件，micro:bit V2 的文件位于路径 boards/bbc/microbit_v2/bbc_microbit_v2.dts，其中可以看到 GPIO 按钮、LED 显示矩阵、I2C 总线和 I2C 总线上的传感器等硬件。应用也可以提供专门针对开发板的设备树覆盖文件，路径为“<应用或模块路径>/boards/<开发板名>.overlay”。覆盖文件中可以增加新的选择/别名节点，也可以配合新的设备树绑定文件(见下节)增加节点。

2.2.3. 设备树绑定

设备树自身的结构相对自由，需要有设备树绑定才能够正确、完整地描述硬件[10]。设备树绑定中包含对设备树节点格式和内容的要求。Zephyr 使用 YAML 文件存储设备树绑定。

图 7 所示的是一个样例绑定文件[11]：

```
description: Custom properties

# The following compatible key does not use the value "custom,props-basics"
# since the vendor "custom" doesn't exist in the vendor-prefixes.txt.
# The generation doesn't fail but it leads to the following message:
# compatible 'custom-props-basics' has unknown vendor prefix 'dummy'
compatible: "custom-props-basics"

properties:
  existent-boolean:
    type: boolean
  int:
    type: int
    required: true
  array:
    type: array
  uint8-array:
    type: uint8-array
  string:
    type: string
  string-array:
    type: string-array
  enum-int:
    type: int
    enum:
      - 100
      - 200
      - 300
  enum-string:
    type: string
    enum:
      - "whatever"
      - "works"
```

Figure 7. A sample device tree binding file (source: Martin Lampacher’s code on GitHub)
图 7. 一个样例设备树绑定文件(来源：Martin Lampacher 在 GitHub 上的代码)

从图 7 中可以看到 3 个重要的键值[12]：

- description (描述)：描述绑定文件适配的硬件的字符串。
- compatible (兼容名)：和设备树中的兼容名对应，一个绑定文件的兼容名如果和一个设备树节点一致，则该设备树节点的格式应当符合绑定文件的内容。
- properties (属性)：描述了符合绑定的节点中的属性与格式。

图 8 所示的是设备树节点符合图 7 中的定义：

```
label_with_props: node_with_props {
  compatible = "custom-props-basics";
  existent-boolean;
  int = <1>;
  array = <0xA second_value: 0xB 0xC>;
  uint8-array = [ 12 34  ];
  string = string_value: "foo bar baz";
  string-array = "foo", "bar", "baz";
  enum-int = <200>;
  enum-string = "whatever";
 };
};
```

Figure 8. A device tree node that is compatible with the binding (source: Martin Lampacher's code on GitHub)
图 8. 符合绑定文件的设备树节点(来源：Martin Lampacher 在 GitHub 上的代码)

从图 8 中可以看到：

- 节点的兼容名和绑定的一致。
- 每一个属性都有按照绑定中 type 的类型赋值。

Zephyr 中默认包括的绑定文件位于 dts/bindings 子目录下，按照类型进行分类，以兼容名的名称进行命名。

除非向 Zephyr 中添加新的硬件支持，一般开发中不添加新的绑定文件。需要时应用可以增加新的绑定文件(<应用或模块路径>/dts/bindings/<兼容名>.yaml)，并在设备树覆盖文件中添加符合绑定定义的节点。

2.2.4. 在程序中访问设备树节点和属性

从 C/C++应用代码中可以用多种方式访问设备树节点。

```
/dts-v1/;

/ {

        aliases {
                sensor-controller = &i2c1;
        };

        soc {
                i2c1: i2c@40002000 {
                        compatible = "vnd,soc-i2c";
                        label = "I2C_1";
                        reg = <0x40002000 0x1000>;
                        status = "okay";
                        clock-frequency = < 100000 >;
                };
        };
};
```

Figure 9. Methods to access a device tree node (source: zephyrproject.org)
图 9. 访问设备树节点的方法(来源：zephyrproject.org)

以图 9 为例，多种宏都可以得到 i2c@40002000 节点(注意：将所有不是字母数字的字符替换为下划线)：

- DT_PATH(soc, i2c_40002000)：将全路径以逗号隔开，省略所有“/”。
- DT_NODELABEL(i2c1)：使用标签名。
- DT_ALIAS(sensor_controller)：使用别名。
- DT_INST(x, vnd_soc_i2c)：寻找第 x 个兼容名为“vnd,soc-i2c”的节点。在本例中因为只有一个节点，x 应为 0。在多“vnd, soc-i2c”节点的情况下，x 和设备树中节点的对应关系不能保证。

对于 chosen 节点(图 9 中不包括)，使用 DT_CHOSEN 指定节点，例如针对图 6 中的设备树可以使用“DT_CHOSEN(zephyr_console)”。

注意：上述宏不能用于变量，只能用于宏定义。

DT_NODE_HAS_PROP 宏可以用于检测节点是否有特定属性，例如“DT_NODE_HAS_PROP(DT_NODELABEL(i2c1), clock_frequency)”的值为 1。访问节点的属性时使用 DT_PROP 宏，例如“DT_PROP(DT_PATH(soc, i2c_40002000), clock_frequency)”的值为 100000。DT_PROP 的值可以用于变量初始化或是静态定义。

Zephyr 定义了众多与设备树相关的宏，在官方文档中有分类总结。在开发中请根据需要查阅文档，并参考 Zephyr 丰富的开发板/传感器样例库。

2.3. Kconfig 配置工具

Kconfig 是在构建时配置 Zephyr 内核和子系统的主要方式，Kconfig 也是 Linux 内核的配置系统。Zephyr 中的 Kconfig 配置选项按照文件夹的层级结构分布，从 Zephyr 代码库根目录的 Kconfig.zephyr 文件开始。根 Kconfig 文件用包含(include)语句包括了子系统(例如内核、驱动和代码库)的 Kconfig 文件，子系统还可以进一步深入定义更深层的 Kconfig 结构和选项。

开发板和应用可以指定需要启用的配置。BBC micro:bit V2 板的默认选项位于文件 boards/bbc/micro-bit_v2/bbc_microbit_v2_defconfig 中，包括系统时钟、串口和控制台等选项。每个应用中的 prj.conf 则包含了应用所需的选项。

与 Linux 类似，在 Zephyr 中可以通过命令行界面进行 Kconfig 选项配置[13]。针对应用构建后产生的 build 文件夹运行命令“west build --build-dir ./build -t menuconfig”即可进入命令行界面(见图 10)。

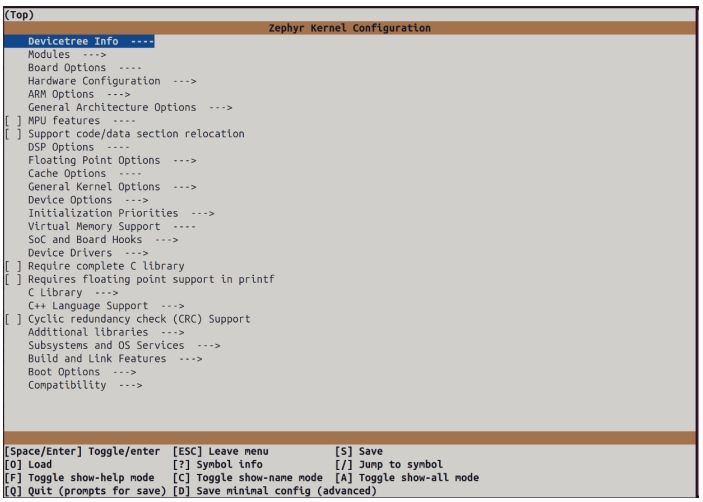

Figure 10. Kconfig menuconfig interface
图 10. Kconfig 配置命令行界面

在界面中可以通过方向键和 ESC/空格键进行导航，在选项上通过空格键进行选择。修改选项后 D 键保存最小配置到文件，也就是当前界面中定义的 Kconfig 选项和 Zephyr 定义的开发板默认选项的区别。图 11 所示的是 micro:bit V2 LED 矩阵显示样例的输出结果(第 3.1 节会使用这一样例)：

```
CONFIG_SERIAL=y
CONFIG_GPIO=y
CONFIG_CONSOLE=y
CONFIG_NRFX_GPIOTE_NUM_OF_EVT_HANDLERS=1
CONFIG_CLOCK_CONTROL_NRF_K32SRC_RC=y
CONFIG_CLOCK_CONTROL_NRF_K32SRC_250PPM=y
CONFIG_UART_CONSOLE=y
CONFIG_DISPLAY=y
CONFIG_MICROBIT_DISPLAY=y
```

Figure 11. Kconfig minimum config output
图 11. Kconfig 最小配置输出

对比前面提到的开发板默认 Kconfig 选项和应用添加的选项(samples/boards/bbc/microbit/display/prj.conf)，可以看到只有“CONFIG_NRFX_GPIOTE_NUM_OF_EVT_HANDLERS”选项是上述两个文件中没有包括的，这是因为北欧半导体的 HAL 层自动定义了这一选项(modules/hal_nordic/nrfx/nrfx_kconfig.h)。除了这样的例外情况，一般在命令行界面中选中了新的选项，用最小选项输出就可以帮助确定新的选项名，之后就可以将其加入到 prj.conf 文件中，从而在编译过程中包括这一选项。

Kconfig 选项除了用于开启子系统功能之外，也用于配置驱动、应用代码，以及下一章将要讲解的日志系统。在代码中可以用“CONFIG_<Kconfig 配置名>”宏取得配置值，构建用的 CMakeLists.txt 文件也可以用“CONFIG_<Kconfig 配置名>”读取配置值。

3. 基于 Zephyr 的嵌入式应用开发

本章中，我们将结合样例在 Zephyr 上实践嵌入式应用开发，帮助理解上一章中的理论。

3.1. 环境配置和运行第一个程序

首先，跟随 Zephyr 项目入门指南(https://docs.zephyrproject.org/latest/develop/getting_started/index.html)完成环境配置、Zephyr 和 Zephyr SDK 的安装。总体来说，Zephyr 在 Linux 中的安装和配置步骤最为简洁，推荐在 Ubuntu Linux 上进行 Zephyr 的实验和开发。本章提及的命令和环境细节均以在 Ubuntu 24.04 版本上使用 Zephyr 4.0.99 开发版本为准，运行时使用 BBC micro:bit V2 开发板(见图 12)。

Figure 12. micro:bit V2 board (source: microbit.org [14])
图 12. micro:bit V2 板(来源：microbit.org [14])

Zephyr 的样例库中包括众多开发板和传感器的样例，不过指南中提到的闪灯样例(Blinky，路径 samples/basic/blinky)并不能直接套用在 micro:bit V2 上。此处我们采用 micro:bit V2 的 LED 矩阵显示样例(路径 samples/boards/bbc/microbit/display)。连接开发板到 Ubuntu 系统上，运行图 13 中的命令进行编译和烧录。命令中的“-p”选项意味着进行全新编译，当对工程进行重复编译时使用“-p auto”选项允许 west 工具只对更改的部分进行重新编译，这适合在开发迭代时节约时间。

```
west build -b bbc_microbit_v2 samples/boards/bbc/microbit/display -p
west flash
```

Figure 13. Commands to compile and flash the sample onto the micro:bit V2 board
图 13. 针对 micro:bit V2 板编译和烧录的命令

成功后开发板会自动启动 Zephyr，开发板背后(见图 12，有 BBC micro:bit v2 字样的面为正面) 5 乘 5 的 LED 矩阵会显示数字倒计时 9 到 0，然后是 LED 的逐行逐列“行军”，最后开始持续滚动显示“Hello Zephyr！”的字样。

该实例展示了较为复杂的单组件运作，从主函数(samples/boards/bbc/microbit/display/src/main.c)可以看到样例通过一个针对 micro:bit 板专用的中间层(drivers/display/mb_display.c)对泛用的显示驱动(头文件 zephyr/drivers/display.h)进行扩充，实现了大多数的功能，例如初始化、打印数字或字母，以及按照 0/1 矩阵点亮 LED 等。

3.2. 闪灯样例和设备树问题

上一节提到，Zephyr 的闪灯样例在 micro:bit V2 上不能运行，本节让我们了解其背后的理由和如何修复与设备树相关的问题。

运行图 14 所示的命令尝试编译闪灯样例：

```
cd ~/zephyrproject/zephyr
west build -b bbc_microbit_v2 samples/basic/blinky -p
```

Figure 14. Commands to compile the blinky sample
图 14. 编译闪灯样例的命令

运行的结果是图 15 所示的编译错误：

Figure 15. Compile error of the blinky sample
图 15. 闪灯样例的编译错误

第 2.2.4 节中提到，Zephyr 提供一整套设备树宏，本例中 GPIO 代码使用的 DT_ALIAS 宏不能完全展开。Zephyr 中设备树宏错误的原因一般都与编译错误中提到的头文件无关，而是设备树有格式/内容的错误，或者访问设备树的方式有误。几种常见的错误如下：

- 混淆了选择、别名、标签名和节点名，或者输入了错误的字符串(例如没有将非字母数字的字符转换为下划线)。
- 在硬件特定的宏中(例如图 15 的 GPIO_DT_SPEC_GET 需要指向一个 GPIO phandle 节点)使用了不同硬件的节点。
- 设备树的节点和绑定的格式要求不一致，导致节点未能生成正确的头文件，因此应用或者驱动中的宏无法展开。注意：这和简单的设备树语法错误不同，语法问题在编译设备树时就会导致编译失败，内容的问题则可能导致在应用代码中无法使用特定属性或宏。
- 使用了错误的宏组合或者宏的参数错误，特别是 For-Each 循环宏和硬件特定的宏。

打开 micro:bit V2 的设备树文件(boards/bbc/microbit_v2/bbc_microbit_v2.dts)，可以看到 aliases 节点下没有 led0，缺少 led0 别名导致了编译的失败[15]。

micro:bit V2 的 LED 矩阵由十个 GPIO 输出控制，个别改变一个控制引脚(pin)并不能点亮 LED。红色的电源指示灯和黄色的 USB 指示灯也并没有连接到 GPIO 上，因此只是依靠开发板本身，我们并不能通过扩展设备树简单地修改好闪灯样例。不过，micro:bit V2 可以外接 LED，将外接 LED 的 GPIO 添加到设备树中就可以修复闪灯样例。

添加设备树覆盖文件 samples/basic/blinky/boards/bbc_microbit_v2.overlay 修复编译错误[16]，见图 16：

```
/ {
    leds {
        compatible = "gpio-leds";
        led0_label: led_0 {
            gpios = <&gpio0 4 GPIO_ACTIVE_HIGH>;
        };
    };

    aliases {
        led0 = &led0_label;
    };
};
```

Figure 16. The device tree overlay file to fix the compilation error
图 16. 修复编译错误的设备树覆盖文件

可以看到文件增加了一个兼容名为 gpio-leds 的节点 leds，然后为含有 GPIO 信息的 led_0 子节点增加别名 led0。gpio-leds 的驱动(drivers/led/led_gpio.c)提供了开关和设定亮度的接口，不过在闪灯样例中，代码(samples/basic/blinky/src/main.c)只是通过 GPIO_DT_SPEC_GET 宏从设备树取得了 GPIO 引脚的信息，然后直接使用 gpio_pin_toggle_dt 切换 GPIO 输出状态。

对比主设备树文件的 edge_connector (边缘连接器)节点和开发板的引脚图[17]可以看到，图 16 中 gpio0 接入点引脚 4 对应 P2 引脚(开发板下侧标记 2 的金手指)，运行时如果有连接外接 LED，闪灯样例就能够运行。

类似的设备树覆盖文件方法，只要正确地修改 GPIO 接入点和引脚号，也可以让没有 led0 别名的开发板支持闪灯样例。

3.3. 样例应用和详解

本节将使用基于官方样例[18]改编的样例应用(https://github.com/lingyuan-he/zephyr-example)。除了主

程序代码还包括：

- 一个简单的自定义代码库(accel)：从 3-轴加速度传感器取得加速度数值，该库可以通过 Kconfig 启用或禁用。
- 一个简单的自定义 LED 矩阵驱动层(ledmatrix)：不使用 Zephyr 的显示驱动，手动通过 GPIO 点亮单个行或列的 LED，该驱动层可以通过 Kconfig 启用/禁用和配置。
- 设备树覆盖文件：用于辅助自定义代码库和 LED 矩阵驱动层，并展示简单的设备树功能。

驱动、代码库和主函数各自配置了日志模块，可以通过 Kconfig 配置日志级别。

3.3.1. 3-轴加速度传感器的代码调用

从主设备树文件上可以看到，micro:bit V2 上内建了 ST 的 lsm303agr 3-轴加速度传感器(见图 17)。在样例应用中，custom-module/lib/accel/accel.c 源代码和 custom-module/include/app/lib/accel.h 头文件将寻找传感器设备和从传感器设备取得 3-轴加速度值的功能包装到了一个简单的自定义库 accel 中。

```
lsm303agr_accel: lsm303agr-accel@19 {
        compatible = "st,lis2dh", "st,lsm303agr-accel";
        status = "okay";
        reg = <0x19>;
        irq-gpios = <&gpio0 25 GPIO_ACTIVE_HIGH>;
};
```

Figure 17. micro:bit V2 device tree file snippet (source: Zephyr on GitHub)
图 17. micro:bit V2 设备树文件片段(来源：Zephyr GitHub 代码库)

accel 库代码中，寻找传感器设备的 get_accel_device 函数通过别名 accel 寻找设备树中的加速度传感器设备，这一别名在 micro:bit V2 的主设备树文件中并不存在(其中只有 accel0)，而是由样例应用设备树覆盖文件(app/boards/bbc_microbit_v2.overlay)提供的。其中增加了 accel 别名，指向标签为 lsm303agr_accel 的节点。

设备树覆盖文件能在开发板的主设备树文件上进行增添和修改，它的几项用途如下[19]：

- 增加别名(本例的 accel)或者选择。
- 覆写已有节点的属性值，例如更改串口的数据速率。
- 删除节点的一个属性。
- 增加子节点，例如总线上新的子设备。

回到 accel.c 代码中，get_accel_values 函数用于获取 3-轴加速度值，其中 sensor_sample_fetch 和 sensor_channel_get 函数调用完成了样本刷新和取样本值的功能。了解它们是如何针对特定的传感器完成代码调用的，能够帮助我们更加深入地理解 Zephyr 的设备驱动模型(第 2.1 节)。

sensor_sample_fetch 和 sensor_channel_get 函数均为泛用传感器驱动 API，从 Zephyr 代码库头文件 include/zephyr/drivers/sensor.h 可以看到两个函数会分别调用设备驱动 API sample_fetch 和 channel_get 函数。设备树中设备的兼容名决定了适配的驱动程序。在设备树文件中，传感器的兼容名有两个：“st,lis2dh”和“st,lsm303agr-accel”。驱动的适配顺序是先查找第一个兼容名，在 Zephyr 代码中搜索 st_lis2dh (非字母数字的字符替代为下划线)，可以找到 drivers/sensor/st/lis2dh/lis2dh.c 文件包含定义驱动的语句“#define DT_DRV_COMPAT st_lis2dh”。图 18 所示的是该驱动的驱动 API 结构定义：

```
static DEVICE_API(sensor, lis2dh_driver_api) = {
        .attr_set = lis2dh_attr_set,
#if CONFIG_LIS2DH_TRIGGER
        .trigger_set = lis2dh_trigger_set,
#endif
        .sample_fetch = lis2dh_sample_fetch,
        .channel_get = lis2dh_channel_get,
};
```

Figure 18. lis2dh device driver API definition (source: Zephyr on GitHub)
图 18. lis2dh 设备驱动的 API 定义(来源：Zephyr GitHub 代码库)

可以看到该驱动将 lis2dh_sample_fetch 和 list2dh_channel_get 函数的指针指定为设备 sample_fetch 和 channel_get API 的实现。lis2dh 驱动支持 I2C 和 SPI 总线，在主设备树文件中可以看到，micro:bit V2 中的传感器是在 i2c 总线上的。图 19 所示的是 lis2dh 驱动的部分初始化代码：

```
#define LIS2DH_CONFIG_I2C(inst)                                         \
        {                                                               \
                .bus_init = lis2dh_i2c_init,                            \
                .bus_cfg = { .i2c = I2C_DT_SPEC_INST_GET(inst), },      \
                .hw = { .is_lsm303agr_dev = IS_LSM303AGR_DEV(inst),     \
                        .disc_pull_up = DISC_PULL_UP(inst),             \
                        .anym_on_int1 = ANYM_ON_INT1(inst),             \
                        .anym_latch = ANYM_LATCH(inst),                 \
                        .anym_mode = ANYM_MODE(inst), },                \
                LIS2DH_CFG_TEMPERATURE(inst)                            \
                LIS2DH_CFG_INT(inst)                                    \
        }

#define LIS2DH_DEFINE_I2C(inst)                                         \
        static struct lis2dh_data lis2dh_data_##inst;                   \
        static const struct lis2dh_config lis2dh_config_##inst =        \
                LIS2DH_CONFIG_I2C(inst);                                \
        LIS2DH_DEVICE_INIT(inst)
/*
 * Main instantiation macro. Use of COND_CODE_1() selects the right
 * bus-specific macro at preprocessor time.
 */

#define LIS2DH_DEFINE(inst)                                             \
        COND_CODE_1(DT_INST_ON_BUS(inst, spi),                          \
                    (LIS2DH_DEFINE_SPI(inst)),                          \
                    (LIS2DH_DEFINE_I2C(inst)))

DT_INST_FOREACH_STATUS_OKAY(LIS2DH_DEFINE)
```

Figure 19. lis2dh device driver initialization code (source: Zephyr on GitHub)
图 19. lis2dh 设备驱动初始化代码(来源：Zephyr GitHub 代码库)

代码通过 DT_INST_FOREACH_STATUS_OKAY 宏，对每一个状态为 okay 的兼容设备扩展

LIS2DH_DEFINE 宏，后者会通过 DT_INST_ON_BUS 判断设备是否在 spi 总线上，如果是，就进一步扩展 LIS2DH_DEFINE_SPI 初始化驱动，否则会扩展 LIS2DH_DEFINE_I2C 宏(micro:bit V2 的情况)。那么，设备树是如何让 DT_INST_ON_BUS 能够进行判定的呢？

micro:bit V2 设备树中传感器所在的 i2c 节点兼容名为“nordic,nrf-twim”，从其绑定文件 dts/bindings/i2c/nordic,nrf-twim.yaml 中可以看到，文件包含(include)了 nordic,nrf-twi-common.yaml (同文件夹下)，然后该文件又进一步包含了 i2c-controller.yaml，在这一文件中终于看到了“bus: i2c”的信息。也就是说，从设备树绑定可以得知传感器从属于使用 i2c 总线的控制器。

由于 lis2dh 驱动能够被正确地配置，系统不会查找兼容“st,lsm303agr-accel”的驱动。在运行时，accel 代码库中的 sensor_sample_fetch 和 sensor_channel_get 函数会调用 st_lis2dh 驱动的函数。

在 Zephyr 的在线文档中，通过兼容名可以找到设备树绑定的参考页面，例如本例中的驱动文档标题为“st,lis2dh (on i2c bus)”。

3.3.2. 设备树绑定和自定义驱动

在样例应用中，自定义的 ledmatrix 驱动(custom-module/drivers/ledmatrix/ledmatrix.c)使用 GPIO 在 LED 矩阵上实现了简单点亮矩阵边缘一排或一行 5 枚 LED 的功能。在前一节中提到，驱动需要匹配到设备树的设备节点上。本例中我们创建了自定义的“custom-ledmatrix”兼容名和其绑定，以及 ledmatrix 驱动实现。

图 20 和图 21 所示的分别是 custom-ledmatrix 设备树绑定文件(custom-module/dts/bindings/custom-ledmatrix.yaml)和 micro:bit V2 设备树覆盖文件中的对应节点：

```
description: |
  An LED matrix that consists of row and column GPIO pins.

# Device tree node will have the same "compatible" attribute to be picked up
# by a driver that intends to operate on the device.
compatible: "custom-ledmatrix"

include: base.yaml

properties:
  led-row-gpios:
    type: phandle-array
    required: true
    description: |
      Array of row GPIOs pins that are part of the matrix.

  led-col-gpios:
    type: phandle-array
    required: true
    description: |
      Array of column GPIOs pins that are part of the matrix.
```

Figure 20. The device tree binding file for custom-ledmatrix
图 20. custom-ledmatrix 设备树绑定文件

```
/*
 * Define a node of the custom "custom-ledmatrix" binding that contains GPIO
 * information from the 5x5 LED matrix. Provide the same GPIO pins as the
 * "led_matrix" node in the bbc_microbit_v2 board devicetree source file.
 */
ledmatrix_label: custom_ledmatrix {
        compatible = "custom-ledmatrix";
        status = "okay";
        led-row-gpios = <&gpio0 21 GPIO_ACTIVE_HIGH>,
                        <&gpio0 22 GPIO_ACTIVE_HIGH>,
                        <&gpio0 15 GPIO_ACTIVE_HIGH>,
                        <&gpio0 24 GPIO_ACTIVE_HIGH>,
                        <&gpio0 19 GPIO_ACTIVE_HIGH>;
        led-col-gpios = <&gpio0 28 GPIO_ACTIVE_LOW>,
                        <&gpio0 11 GPIO_ACTIVE_LOW>,
                        <&gpio0 31 GPIO_ACTIVE_LOW>,
                        <&gpio1 5 GPIO_ACTIVE_LOW>,
                        <&gpio0 30 GPIO_ACTIVE_LOW>;
};
```

Figure 21. The custom-ledmatrix device tree node
图 21. cutstom-ledmatrix 设备树节点

从图 20 中可以看到，custom-ledmatrix 绑定中有两个 GPIO 引脚 phandle 数组，分别代表 LED 矩阵的行 GPIO 引脚(推挽)和列 GPIO 引脚(开漏) [20]。在图 21 中，注意到 GPIO 接入点、引脚号和逻辑电平模式与开发板主设备树文件中“led_matrix”节点(兼容名“nordic,nrf-led-matrix”)是一致的[21]。样例中我们使用 GPIO 在不使用动态刷新的情况下进行亮、灭灯，所以不需要其他的属性。

需要特别注意的是，为了表示 phandle 每个说明符(specifier)成员的长度(例如 GPIO 除了接入点之外需要提供两个数据成员)，在绑定中一般应提供名称为“#*-cells”的属性。不过由于 GPIO 类 phandle 十分常见，只要属性的命名以“-gpios”结尾，如本例中的 led-row-gpios 和 led-col-gpios，就不需要提供这一属性。关于“#*-cells”属性的细节详见官方文档。

从图 21 中还可以看到设定设备状态就绪的语句(status 为“okay”)，节点能够使用该属性是因为绑定文件包含了 base.yaml。上一节中提到，标记设备就绪对于驱动的初始化是必须的，例如图 19 中用到的 DT_INST_FOREACH_STATUS_OKAY 宏。

自定义驱动的头文件定义见 custom-module/include/app/drivers/ledmatrix.h，可以看到驱动 API 由 5 个函数组成(见 ledmatrix_driver_api 结构定义)，分别负责点亮 LED 矩阵最边缘的行或是列(共 4 个 API)和关闭 LED 显示(第 5 个 API)。在驱动的实现(custom-module/drivers/ledmatrix/ledmatrix.c)中，这 5 个函数会被实现(见 driver_api 结构) [22]。现在读者应该能够理解 ledmatrix 驱动的基本结构。最后，图 22 所示的是驱动的初始化宏：

```
#define LEDMATRIX_DEFINE(inst)                                          \
                                                                        \
    static const struct ledmatrix_config config##inst = {              \
        .rows = { DT_INST_FOREACH_PROP_ELEM_SEP(inst, led_row_gpios,    \
                                        GPIO_DT_SPEC_GET_BY_IDX, (, )) },   \
        .cols = { DT_INST_FOREACH_PROP_ELEM_SEP(inst, led_col_gpios,    \
                                        GPIO_DT_SPEC_GET_BY_IDX, (, )) },   \
    };                                                                  \
                                                                        \
    DEVICE_DT_INST_DEFINE(inst, instance_init, NULL, NULL, &config##inst,   \
                          POST_KERNEL, CONFIG_LEDMATRIX_INIT_PRIORITY,  \
                          &driver_api);

DT_INST_FOREACH_STATUS_OKAY(LEDMATRIX_DEFINE)
```

Figure 22. The initialization of the ledmatrix driver
图 22. ledmatrix 驱动的初始化

LEDMATRIX_DEFINE 中使用 GPIO_DT_SPEC_GET_BY_IDX 配合 DT_INST_FOREACH_PROP_ELEM_SEP，从设备树循环提取 GPIO 引脚 phandle 数组中的成员，从而静态组成 gpio_dt_spec 数组[23]，用于在设备驱动配置结构(见图 23)中存储行和列 GPIO 引脚属性[24]。

```
struct ledmatrix_config {
    struct gpio_dt_spec rows[NUM_ROW];
    struct gpio_dt_spec cols[NUM_COL];
};
```

Figure 23. The configuration structure of the ledmatrix driver
图 23. ledmatrix 驱动的配置结构

和上一节提到的传感器驱动类似，DT_INST_FOREACH_STATUS_OKAY 针对每个状态为就绪的、兼容名为“custom-ledmatrix”的设备进行驱动初始化。

通过 ledmatrix 驱动层，样例应用的主函数就可以很容易地直接进行 LED 行或是列的点亮操作。配合加速度传感器的数据，样例应用实现了根据重力方向点亮 LED 矩阵对应边缘行或列的效果。

3.3.3. 日志系统

Zephyr 提供了日志系统的支持，应用代码、驱动、代码库可以注册各自的日志模块，并通过 Kconfig 配置模块的日志级别。日志的可能级别从低到高分别为：DBG (调试)、INF (信息)、WRN (警告)和 ERR (错误)。代码中通过调用 LOG_X (X 为级别)宏就可以使用与 printk 类似的语法写日志。以样例应用中的 accel 代码库为例，custom-module/lib/accel/accel.c 中包含了 zephyr/logging/log.h 头文件，然后使用 LOG_MODULE_REGISTER 宏定义了日志模块 accel，其日志级别为 CONFIG_ACCELLIB_LOG_LEVEL。

在 Kconfig 中，CONFIG_LOG 配置用于在全局启用日志，然后通过添加 CONFIG_<模块>_LOG_LEVEL_X (X 为级别)配置设定个别模块的级别。本例中应用的配置文件 app/prj.conf 通过 CONFIG_LOG=y 在全局开启了日志功能，然后通过 CONFIG_ACCELLIB_LOG_LEVEL_INF=y 选项将 accel 模块的日志级别定义为 INF(信息)级别。ledmatrix 驱动和应用主代码各自也有日志模块的配置。

使用日志系统相比使用 printk 更加可配置，例如只有调试时才需要的日志可以通过默认日志级别进行过滤，发布应用时也可以很容易地禁用日志输出。

4. 运行和调试 Zephyr 应用

4.1. 运行样例应用

编译和部署样例应用的命令如图 24 所示：

```
cd ~/zephyrproject
mkdir applications && cd applications
git clone https://github.com/lingyuan-he/zephyr-example.git
cd zephyr-example
west build -b bbc_microbit_v2 app -p
west flash
```

Figure 24. Commands to download and deploy the example application
图 24. 下载和部署样例应用的命令

样例应用开始运行时，将开发板平放于台面上，此时 LED 矩阵不会点亮，如果将开发板拿起，一侧垂直朝向地面时，检测到重力一侧的一排或一列 5 个 LED 会点亮。例如，当开发板垂直于台面正面并面

向读者时，LED 矩阵最下一行会点亮(见图 25)。

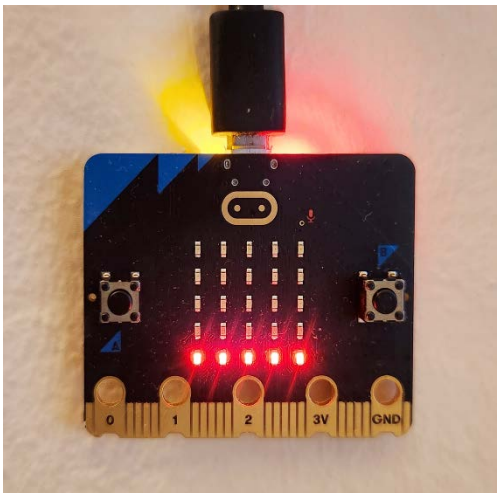

Figure 25. Illumination of the bottom row LEDs when the board is upright
图 25. 开发板垂直摆放时，最下一排的 LED 点亮

4.2. Zephyr 应用的调试

在 Zephyr 应用开发中，最简单的调试方法就是输出日志。micro:bit V2 运行样例应用时会将日志输出到串口，可以通过任何串口工具连接串口，例如使用 minicom 的命令：“minicom -D /dev/ttyACM0 -b 115200”。

样例应用的默认日志级别为 INF，编译时可以通过包括 debug.conf 的选项将日志级别降低为 DBG，程序就会输出传感器数据和 GPIO 操作细节，命令为：“west build -b bbc_microbit_v2 app -p --extra-conf debug.conf”。

Zephyr 支持在 micro:bit V2 上使用 GDB 进行远程调试，应用编译和烧录(“west build”和“west flash”)后，运行“west debug”就会启动 GDB。GDB 简单的用法例如：设置断点(“b main.c:<行数>”或“b <函数名>”)、逐行执行(n)、继续执行(c)和打印变量(“p <变量名>”)。“west debug”命令还可以指定 GDB 以外的调试接口[25]，例如 jlink 和 openocd。

5. 结语

Zephyr 在系统设计上借鉴了 Linux 等大型开源软件的设计理念，引入了 Linux 和桌面系统开发者熟悉的概念和开发过程，但相对常见的 RTOS，复杂度增加了数个级别。通过将硬件进行抽象化，以及提供帮助简化开发过程的工具和框架(例如 west 工具和 twister 测试框架)，Zephyr 希望能吸引不同领域的开发者和企业用户。但是，开发和调试难度的上升也让不少开发者望而却步，特别是熟悉面向硬件直接编程或是使用小型 RTOS 的嵌入式开发者。

希望在阅读本文后，读者对在 Zephyr 上进行嵌入式软件开发有了初步的了解。本文中的实例并不涉及过于具体的硬件细节或是复杂的应用需求，Zephyr 的官方文档、实例，以及北欧半导体等硬件厂商的样例项目都十分有参考价值。虽然官方文档的中文化有所欠缺，但国内开发者在各类平台上发布的学习笔记一直在增加，线上讨论也十分热烈。

Zephyr 近年来劲头强势，硬件厂商、开发者和开源社区的热情正盛，项目的开发活跃程度远超其他 RTOS。期待 Zephyr 项目在未来能够简化复杂的系统架构，改善学习难度高和代码调试困难等问题，并

覆盖更多的硬件和应用，成为一个全方位的主流物联网操作系统。

参考文献

[1] Zephyr Project (2025) About the Zephyr Project. https://zephyrproject.org/learn-about/

[2] Eclipse Foundation (2024) 2024 IoT & Embedded Developer Survey Report. https://outreach.eclipse.foundation/iot-embedded-developer-survey-2024

[3] 王洪波. 嵌入式虚拟化技术与应用: ACRN 开源项目实践[M]. 北京: 机械工业出版社, 2023.

[4] Zephyr Project (2024) Zephyr Project Overview. https://www.zephyrproject.org/wp-content/uploads/2024/10/Zephyr-Overview-20241017.pdf

[5] Zephyr Project (2025) Device Driver Model. https://docs.zephyrproject.org/latest/kernel/drivers/index.html

[6] The Devicetree Organization (2023) Devicetree Specification, Release v0.4. https://github.com/devicetree-org/devicetree-specification/releases/download/v0.4/devicetree-specification-v0.4.pdf

[7] Zephyr Project (2025) Devicetree. https://docs.zephyrproject.org/latest/build/dts/index.html

[8] Zephyr Project (2025) Scope and Purpose. https://docs.zephyrproject.org/latest/build/dts/intro-scope-purpose.html

[9] Zephyr Project (2025) Syntax and Structure. https://docs.zephyrproject.org/latest/build/dts/intro-syntax-structure.html

[10] Zephyr Project (2025) Devicetree Bindings. https://docs.zephyrproject.org/latest/build/dts/bindings.html

[11] Lampacher, M. (2025) Practical Zephyr Git Repository. https://github.com/lmapii/practical-zephyr/tree/main/03_devicetree_semantics

[12] Eliasz, A. (2024) Zephyr RTOS Embedded C Programming. Apress Berkeley.

[13] Lampacher, M. (2024) Practical Zephyr-Kconfig (Part 2). https://interrupt.memfault.com/blog/practical_zephyr_kconfig

[14] The Micro:Bit Organization (2025) Meet the New BBC Micro:Bit. https://microbit.org/new-microbit/

[15] Gammell, C. (2024) Zephyr for Hardware Engineers: GPIO. https://blog.golioth.io/zephyr-for-hardware-engineers-gpio/

[16] Valens, C. (2024) Getting Started with the Zephyr RTOS. *Elektor*, **2**, 98-105.

[17] The Micro:Bit Organization (2025) Micro:Bit Pins. https://makecode.microbit.org/device/pins

[18] Zephyr Project (2025) Example Application Git Repository. https://github.com/zephyrproject-rtos/example-application/tree/main

[19] Zephyr Project (2025) Devicetree HOWTOs. https://docs.zephyrproject.org/latest/build/dts/howtos.html

[20] Zephyr Project (2021) Zephyr and the BBC Microbit V2 Tutorial Part 1: GPIO. https://www.zephyrproject.org/zephyr-and-the-bbc-microbit-v2-tutorial-part-1-gpio/

[21] Lampacher, M. (2024) Practical Zephyr-Devicetree Semantics (Part 4). https://interrupt.memfault.com/blog/practical_zephyr_dt_semantics

[22] Szczys, M. (2024) How to Write a Zephyr Device Driver with a Custom API. https://blog.golioth.io/how-to-write-a-zephyr-device-driver-with-a-custom-api/

[23] Nordic Semiconductor (2025) GPIO Generic API. https://academy.nordicsemi.com/courses/nrf-connect-sdk-fundamentals/lessons/lesson-2-reading-buttons-and-controlling-leds/topic/gpio-generic-api/

[24] Lampacher, M. (2024) Practical Zephyr-Devicetree Practice (Part 5). https://interrupt.memfault.com/blog/practical_zephyr_05_dt_practice

[25] Zephyr Project (2025) Building, Flashing and Debugging. https://docs.zephyrproject.org/latest/develop/west/build-flash-debug.html

Embedded Technology and Intelligent Systems 嵌入式技术与智能系统, 2025, 2(2), 78-95
Published Online April 2025 in Hans. https://www.hanspub.org/journal/etis
https://doi.org/10.12677/etis.2025.22007

Hans汉斯

基于多模态情感交互的学生心理健康支持系统

史静怡[1]，杨鹏飞[1*]，黄嘉阳[1]，贾　瑞[1]，姚炫竹[1]，许　喆[1]，常志奇[1]，戴逸飞[1]，魏　萍[2]

[1]西安电子科技大学计算机科学与技术学院，陕西　西安
[2]西安电子科技大学心理健康教育中心，陕西　西安

收稿日期：2025年4月1日；录用日期：2025年4月23日；发布日期：2025年4月30日

摘　要

心理健康是社会普遍关注的问题，我国抑郁症患者群体人数持续扩大，发病群体呈现年轻化趋势，且高校学生群体占比持续升高，利用人工智能技术赋能高校心理健康工作刻不容缓。本文针对现有高校心理健康工作存在集中评测精度不足、隐患排查时效性差、传统面谈覆盖面窄等问题，提出了基于多模态情感交互的学生心理健康支持系统。该系统依托校园行为大数据，构建学生异常情绪及行为监测与预警机制；通过自主研发的大语言模型，实现学生情绪的动态识别与智能评估，并结合心理学理论，动态适配个性化疏导策略，支持多角色的情感陪伴与心理支持。系统在实际应用中表现出良好的效果，心理状态评测准确率超过85%，显著提升了心理服务的精准性与响应效率，为校园心理健康教育体系的智能化与科学化建设提供了有力的技术支撑。

关键词

心理健康评估，心理状态监测，多模态情感交互，个性化疏导，大语言模型应用

Student Mental Health Support System Based on Multimodal Emotional Interaction

Jingyi Shi[1], Pengfei Yang[1*], Jiayang Huang[1], Rui Jia[1], Xuanzhu Yao[1], Zhe Xu[1], Zhiqi Chang[1], Yifei Dai[1], Ping Wei[2]

[1]School of Computer Science and Technology, Xidian University, Xi'an Shaanxi
[2]Mental Health Education Center, Xidian University, Xi'an Shaanxi

Received: Apr. 1st, 2025; accepted: Apr. 23rd, 2025; published: Apr. 30th, 2025

Abstract

Mental health is a common concern in society. The number of depression patients in my country

*通讯作者。

文章引用: 史静怡, 杨鹏飞, 黄嘉阳, 贾瑞, 姚炫竹, 许喆, 常志奇, 戴逸飞, 魏萍. 基于多模态情感交互的学生心理健康支持系统[J]. 嵌入式技术与智能系统, 2025, 2(2): 78-95. DOI: 10.12677/etis.2025.22007

continues to expand, the incidence group shows a trend of younger age, and the proportion of college students continues to increase. It is urgent to use artificial intelligence technology to empower college mental health work. Aiming at the problems of insufficient centralized evaluation accuracy, poor timeliness of hidden danger investigation, and narrow coverage of traditional interviews in existing college mental health work, this paper proposes a student mental health support system based on multimodal emotional interaction. Relying on campus behavior big data, the system builds a monitoring and early warning mechanism for students' abnormal emotions and behaviors; through the independently developed large language model, it realizes the dynamic recognition and intelligent evaluation of students' emotions, and combines psychological theory to dynamically adapt personalized counseling strategies to support emotional companionship and psychological support for multiple roles. The system has shown good results in practical applications, with an accuracy rate of more than 85% in psychological state evaluation, which significantly improves the accuracy and response efficiency of psychological services, and provides strong technical support for the intelligent and scientific construction of the campus mental health education system.

Keywords

Mental Health Assessment, Mental State Monitoring, Multimodal Emotional Interaction, Personalized Counseling, Large Language Model Application

1. 引言

随着社会的快速发展，学生群体正在面临着激烈学业竞争、巨大就业压力、复杂人际关系等多重心理挑战，其心理健康问题愈发凸显。最新研究数据显示，我国大学生抑郁症状检出率高达 24.71% [1]；《中国国民心理健康发展报告(2021~2022)》进一步揭示，大学生群体中抑郁和焦虑的风险检出率分别为21.48%和 45.28% [2]。心理危机不仅影响大学生个体的成长与发展，也给家庭和学校带来诸多挑战，更有可能会导致极端行为的发生，进而对国家的人才培养质量和社会稳定性造成不可忽视的负面影响。

校园心理健康教育具有“育心、启智、养德、导行”的重要思政引领作用，是应对学生心理问题、建设教育强国的重要保障。当前，心理健康教育工作仍面临精准性不足、响应滞后、个性化支持匮乏等挑战，亟需从心理状态监测、心理状态评估与疏导这两个关键环节进行优化提升。在心理状态监测方面，现有方法大多停留在静态数据分析，缺乏持续性、系统性监测，难以揭示学生深层次的心理状态与问题根源，且评估结果单一，难以有效支撑个性化疏导。在心理状态评估与疏导方面，现有评估手段多依赖单一问卷或测试，缺乏多维度、多模态的数据支撑，评估结果难以全面刻画学生的个体心理特征。心理干预多依赖统一课程和普适性讲解，忽视个体差异与多样化需求，且心理咨询资源有限，服务隐蔽性不足，学生因羞耻心或担忧评价往往不愿敞开心扉，影响干预效果。

近年来，情感计算技术凭借在情绪识别与个性化支持方面的优势，广泛应用于医疗、教育等领域。针对上述问题，构建以情感交互技术为支撑的系统化心理健康支持体系成为关键突破口。情感数字人作为情感交互的应用载体，能够感知学生情绪变化，融合多模态数据，精准解析心理状态，并以无评判、私密的方式提供个性化心理支持，消除传统心理咨询中隐蔽性不足的问题，提升疏导效果。

基于此，本文以情感数字人为核心，探索其在校园心理健康教育中的应用路径与实践方法。在监测环节，系统融合多源异构数据，实时感知学生行为模式与心理状态，动态识别情绪波动与潜在风险，构

建全面、细粒度的心理变化监测机制。在评估与疏导环节，情感数字人基于学生个体差异，深入解析情绪演化过程与潜在心理需求，生成精准评估结果，并通过拟人化情感交互方式建立信任关系，提供具备情感共鸣与心理支持价值的个性化疏导服务。通过构建智能化心理监测系统与评估－疏导一体化服务机制，可以有效提升心理问题筛查的覆盖率、评估的精准性与干预的适配性，推动高校心理健康教育体系智能化、科学化转型，助力学生心理健康保障与社会可持续发展。本文主要创新点如下：

第一，提出了基于校园大数据的实时心理状态监测方法。该方法能够在不干扰学生正常学习与生活的前提下，持续采集其生活行为数据，实时监测心理状态变化并进行风险预警。避免了传统调查方式中由于有意隐瞒、环境干扰等导致的信息偏差问题，提升了心理健康风险识别的及时性与隐蔽性检测能力。

第二，提出了基于大语言模型的心理状态评估和情感支持方法。即以情感数字人为核心，基于自主研发的大模型，通过模拟多角色、多场景的情感交互，实现了学生心理状态的智能评估，结合历史数据提供个性化、动态适配的心理疏导服务，并生成个性化的智能报告。突破了传统依赖单一数据维度或经验判断的局限，打破了时空资源限制，增强了心理支持的情感共鸣与持续陪伴效果。

2. 方案设计

本文围绕学生心理健康问题的动态监测与精准干预，实现了基于多模态情感交互的学生心理健康支持系统，具体包含了两个功能模块：基于校园大数据的实时心理状态监测模块、基于大语言模型的心理状态评估和情感支持模块。系统总体架构如图 1 所示。

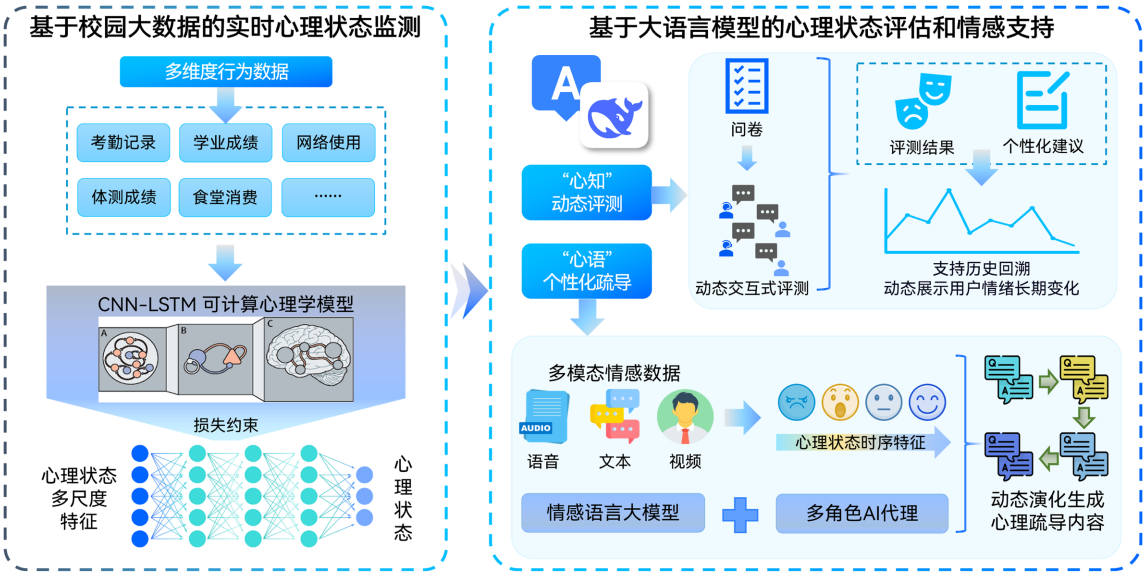

Figure 1. System overall architecture diagram
图 1. 系统总体架构图

基于校园大数据的实时心理状态监测模块旨在实现学生心理状态的连续感知与动态风险识别。本文首先对采集到的校园行为数据(如消费记录、考勤签到、课程成绩等)进行预处理与特征构建；随后引入位置编码与 Transformer Encoder 对行为序列进行建模，提取心理相关的时序特征；采用自监督学习策略，在时序数据中注入随机掩码噪声，增强模型对特征结构与时间依赖的学习能力。最终输出个体在不同时段的情绪状态标签，用于生成动态心理健康画像并驱动下游疏导模块的响应机制。

基于大语言模型的心理状态评估和情感支持模块旨在实现心理评估与个性化疏导的深度融合。本文

基于 DeepSeek：R1-14B 模型进行多轮微调，数据涵盖心理测评文本、情绪问答对话、心理咨询记录等；引入多模态融合机制，支持文本、语音、视频(表情与姿态)输入，避免用户的刻意隐瞒现象；构建多角色 AI 代理，分别训练老师、家人、朋友三种角色，并引入情感共鸣调控机制提升语言共情力。

最终，通过上述两个模块的协同运行，本文构建了集智能化心理监测、评估疏导于一体的服务机制。基于情感数字人技术，推动心理健康服务智能化升级，显著提升了校园心理健康教育的覆盖率、响应速度与服务质量。

3. 基于校园大数据的实时心理状态监测

基于校园大数据的实时心理状态监测模块旨在构建一套覆盖高校学生数据采集、特征提取、心理健康状态监测及危机分级预警的全流程智能化监测体系，其总体结构如图 2 所示。该模块依托信息化平台与大数据分析技术，从多源异构的学生数据中提取有效信息，进行心理健康状态的评估与预警，为早期干预提供支持。

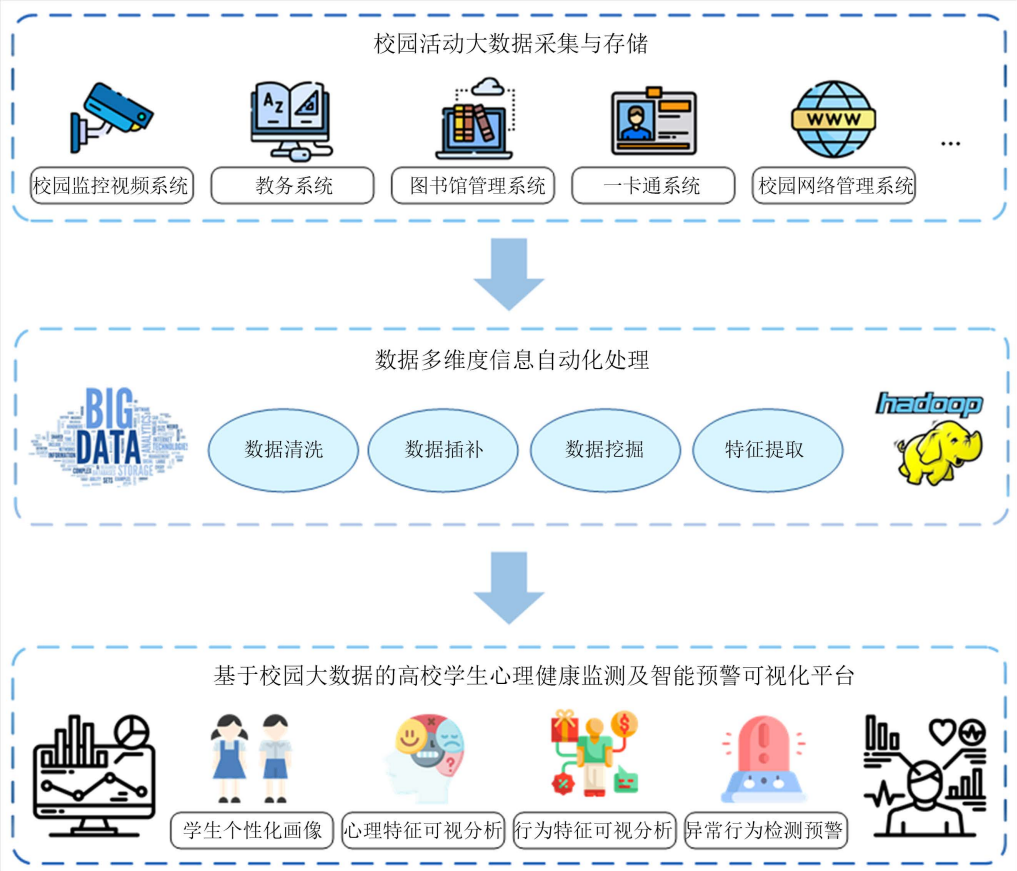

Figure 2. Real-time psychological state monitoring process based on campus big data
图 2. 基于校园大数据的实时心理状态监测流程

3.1. 数据采集

该模块通过整合学生在校内的多源异构数据，包括食堂消费记录、图书馆打卡情况、学业成绩波动、体测结果变化以及网络使用习惯等，经过数据清洗和预处理，构建了一个包含静态特征和动态特征的综合数据集，形成动态更新的个人行为画像，最终得到学生在校的相关数据如表 1 所示。

Table 1. Student data sheet
表 1. 学生在校相关数据表

数据	数据条数
本科生基本信息表	34,273
食堂消费记录表	3,141,047
图书馆打卡记录表	366,855
考试成绩表	2,727,288
学生体测成绩表	37,100
校园网使用记录表	6,123,024
学生心理健康信息表	996
贫困生申请记录表	11,543
奖学金申请记录表	27,621

3.2. 特征提取

学生特征可大致分为静态特征和动态特征。静态特征主要为不会随时间发生变化的特征，如学生性别、民族和专业等；同时，由于学生家庭经济状况、成绩和身体素质在短时间内基本不会发生巨大变化，本文暂将反映学生成绩和身体素质的特征也视为静态特征。动态特征则主要为学生消费、图书馆使用、网络使用情况等相关特征。最终构建的完整特征体系如表 2 所示。

Table 2. Table of static and dynamic characteristics of students
表 2. 学生静态特征与动态特征表

学生静态特征			学生动态特征		
特征来源	特征名	特征类别	特征来源	特征名	特征类别
基本信息	性别 是否少数民族 专业 年级 是否贫困生	分类特征	食堂消费记录	用餐总次数 用餐总金额 早餐用餐次数/金额 午餐用餐次数/金额 晚餐用餐次数/金额	数值特征
体测成绩	体测总分 BMI 肺活量 50 米成绩	数值特征	图书馆打卡记录	总打卡次数 上午打卡次数 下午打卡次数 晚上打卡次数	数值特征
学业成绩	初修成绩均分 初修成绩标准差 初修次数 重修次数 重考次数 获得奖学金次数 获得奖学金总额 平均获得奖学金金额	数值特征	校园网使用情况	校园网使用次数 校园网使用上行/下行流量数 上午校园网使用次数 上午校园网使用上行/下行流量数 下午校园网使用次数 下午校园网使用上行/下行流量数 晚上校园网使用次数 晚上校园网使用上行/下行流量数 深夜校园网使用次数 深夜校园网使用上行/下行流量数	数值特征

3.3. 心理健康状态监测

本文提出了一种融合静态和时序特征的自监督学习方法。在数据处理方面，将采样粒度细化到每日，构建更精细的时间序列数据；在模型构建上，设计了基于门控机制的特征融合模块，将学生的静态特征(如年级、性别、专业等)与动态行为序列进行自适应融合；同时，借助自监督学习策略，通过在时序数据中注入随机噪声并重建原始序列和静态特征的方式，充分利用大量未标注数据来提升模型的特征表示能力。

使用有标签的训练集 $D_{\text{train}}=\left\{\left(X^{(i)},s^{(i)},y^{(i)}\right)|1\le i\le N_{\text{train}}\right\}$、测试集 $D_{\text{test}}=\left\{\left(X^{(i)},s^{(i)},y^{(i)}\right)|1\le i\le N_{\text{test}}\right\}$，其中，$X^{(i)}\in\mathbb{R}^{d_t\times T}$ 为按天数采样的时间序列，T 为当月天数，d_t 为每个时间点的特征数量。$X^{(i)}=\left[\mathbf{x}_1^{(i)},\cdots,\mathbf{x}_t^{(i)},\cdots,\mathbf{x}_T^{(i)}\right]$，其中 $\mathbf{x}_t^{(i)}\in\mathbb{R}^{d_t}$ 为第 t 天的观测数据。$s^{(i)}\in\mathbb{R}^{d_s}$ 为静态特征，d_s 是静态特征维度。$y^{(i)}$ 表示学生的心理状态($y^{(i)}=1$ 表示存在心理危机风险，$y^{(i)}=0$ 表示心理状态正常)。无标签数据集定义为 $D_{\text{unlabel}}=\left\{\left(X^{(i)},s^{(i)}\right)|1\le i\le N_{\text{unlabel}}\right\}$。本节研究的问题可以形式化为：设计一个判别函数 $G(\cdot)$，任意给定一个学生一个月内的动态时序特征 $X^{(i)}$ 和静态特征 $s^{(i)}$，判断其心理状态 $\hat{y}^{(i)}=G\left(X^{(i)},s^{(i)}\right)$。该判别函数需要同时考虑时序特征中的动态变化模式和静态特征中的背景信息，从而实现对学生心理状态的准确预测。

本节的关键挑战在于如何有效建模时序数据中的动态变化模式，以及如何将静态特征与时序特征进行融合。如图 3 所示，本文提出的模型主要包含以下关键步骤。首先，通过门控机制将每个时间点的动态特征与静态特征进行融合，得到融合特征表示 $\overline{\mathbf{x}}_t^{(i)}$。其次，将静态特征通过线性变换编码为 $\overline{\mathbf{x}}_0^{(i)}$，并将其作为时间序列的第一个位置，这种设计类似于自然语言处理中[CLS]标记的作用。接着，对每一天的特征表示进行位置编码，并输入到由自注意力机制和前馈神经网络组成的编码器中，得到各个时间点的特征表示 $\mathbf{z}_t^{(i)}$，$0\le t\le T$。其中，$\mathbf{z}_0^{(i)}$ 作为整体序列的表示，将被输入到分类器中得到最终的分类结果。最后，所有时间点的特征表示 $\mathbf{z}_t^{(i)}$，$0\le t\le T$ 也将用于自监督学习任务，以提升模型的特征提取能力。

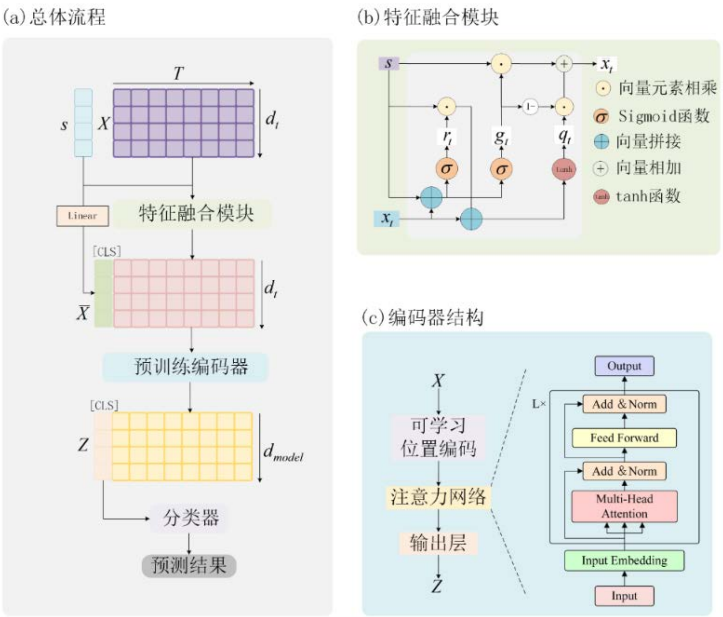

Figure 3. Schematic diagram of the self-supervised learning model architecture that integrates static and temporal features
图 3. 融合静态和时序特征的自监督学习模型架构示意图

3.3.1. 特征融合

在学生心理状态预测任务中，动态特征和静态特征各自承载着不同维度的信息价值。动态特征捕捉了学生行为模式随时间的变化，这些变化往往能反映学生心理状态的波动。而静态特征如性别、年级、学院等则构成了学生的基本画像，为行为解读提供了背景框架。两类异质特征的有效融合是模型性能的关键。若简单地将这两类特征直接拼接，则难以充分挖掘它们之间的复杂关系。在实际场景中，两类特征的信息密度和重要性在不同时间点存在显著差异，且它们之间存在复杂的交互关系。

针对这些考虑，本文使用了一种基于门控机制的特征融合方法，该方法能够自适应地调整两类特征的贡献比例。具体地，在 t 时刻：

$$g_t = \text{sigmoid}\left(W_g\left[\mathbf{x}_t^{(i)}, \mathbf{s}^{(i)}\right] + b_g\right) \tag{1}$$

$$r_t = \text{sigmoid}\left(W_r\left[\mathbf{x}_t^{(i)}, \mathbf{s}^{(i)}\right] + b_r\right) \tag{2}$$

$$q_t = \tanh\left(W_q\mathbf{x}_t^{(i)} + b_q + r_t \odot U_q\mathbf{s}^{(i)}\right) \tag{3}$$

$$\overline{\mathbf{x}}_t^{(i)} = g_t \odot \mathbf{s}_t^{(i)} + (1 - g_t) \odot q_t \tag{4}$$

在这个机制中，g_t 作为控制原始动态特征的保留程度，r_t 调节静态特征对候选状态 q_t 的影响强度。这种设计使模型能够根据不同时间点的特征表现，动态调整融合策略。当某个时间点出现异常行为模式时，模型可以通过门控机制赋予该动态特征更高的权重，而当动态特征不足以提供有效信息时，模型则可以更多地依赖静态特征进行判断。

此外，受到自然语言处理领域的启发，本文将每个学生的静态特征 $\mathbf{s}$ 通过线性变换编码为特殊的[CLS]标记，并将其作为时间序列的起始位置，起到类似[CLS]标记的作用：

$$\overline{\mathbf{x}}_0^{(i)} = W_{\text{cls}}\mathbf{s} + b_{\text{cls}} \tag{5}$$

这样，编码器的完整输入为：

$$\overline{\mathbf{X}}^{(i)} = \left[\overline{\mathbf{x}}_0^{(i)}, \overline{\mathbf{x}}_1^{(i)}, \cdots, \overline{\mathbf{x}}_T^{(i)}\right] \in \mathbb{R}^{d_t \times (T+1)} \tag{6}$$

这种设计使静态特征能够在自注意力机制中与所有时间点的动态特征进行全面交互，从而构建更为丰富的特征表示。通过这种双重融合机制，模型既能捕获局部时间点的特征交互，又能建立全局层面的特征关联，从而更全面地理解学生行为模式与心理状态之间的复杂关系。

3.3.2. 编码器

输入 $\overline{\mathbf{X}}^{(i)}$ 首先经过一个线性变换层，将每个时间点的特征映射到 d_{model} 维度空间，以满足后续注意力机制的计算需求：

$$\hat{\mathbf{X}}^{(i)} = W_{\text{proj}}\overline{\mathbf{X}}^{(i)} + b_{\text{proj}} \tag{7}$$

其中，$W_{\text{proj}} \in \mathbb{R}^{d_{\text{model}} \times d_t}$，$b_{\text{proj}} \in \mathbb{R}^{d_{\text{model}}}$。

本文采用可学习的位置编码矩阵 $P \in \mathbb{R}^{(T+1) \times d_{\text{model}}}$，其中每个元素都是可训练的参数。这种设计使得模型可以更灵活地学习序列中的位置信息。将位置编码与变换后的序列相加，得到带有位置信息的输入矩阵：

$$E_0^{(i)} = \hat{\mathbf{X}}^{(i)} + P \tag{8}$$

设编码器的层数为 L，对于第 l 层($l = 1, 2, \cdots, L$)，其输入为 $E_{l-1}^{(i)}$。每一层首先通过三个不同的线性变

换得到查询矩阵 Q_l、键矩阵 K_l 和值矩阵 V_l：

$$Q_l = W_{Q,l} E_{l-1}^{(i)} \tag{9}$$

$$K_l = W_{K,l} E_{l-1}^{(i)} \tag{10}$$

$$V_l = W_{V,l} E_{l-1}^{(i)} \tag{11}$$

其中，$W_{Q,l}, W_{K,l} \in \mathbb{R}^{d_k \times d_{\text{model}}}$，$W_{V,l} \in \mathbb{R}^{d_v \times d_{\text{model}}}$。设置 $d_k = d_v = d_{\text{model}}/H$，$H$ 为注意力头数。

对于第 h 个注意力头，其计算过程为：

$$\text{head}_h = \text{Attention}\left(Q_l, K_l, V_l\right) = V_l \text{softmax}\left(\frac{Q_l^{\mathrm{T}} K_l}{\sqrt{d_k}}\right) \tag{12}$$

将 H 个头的结果拼接并经过线性变换：

$$\hat{E}_l^{(i)} = W_l^O \left[\text{head}_1; \cdots; \text{head}_H\right] \tag{13}$$

其中，$W_l^O \in \mathbb{R}^{d_{\text{model}} \times Hd_v}$ 是一个线性变换权重矩阵，用于将拼接后的结果转换回与输入维度 d_{model} 相同的空间。

接着进行残差连接和层归一化：

$$\tilde{E}_l^{(i)} = \text{LayerNorm}\left(\hat{E}_l^{(i)} + E_{l-1}^{(i)}\right) \tag{14}$$

最后，利用前馈神经网络的激活函数为模型引入非线性变换的能力，进一步挖掘更复杂的更抽象的特征。前馈神经网络由两个全连接层组成，中间有一个激活函数。将计算结果与输入进行残差链接并进行层归一化得到一层多头注意力输出：

$$E_l^{(i)} = \text{LayerNorm}\left(W_{\text{fc},2} \text{ReLU}\left(W_{\text{fc},1} \tilde{E}_l^{(i)} + b_{\text{fc},1}\right) + b_{\text{fc},2} + \tilde{E}_l^{(i)}\right) \tag{15}$$

经过多个这样的多头注意力机制层和前馈神经网络层的处理后获得最终输出 $E_L^{(i)}$，在本节中记为 $\mathbf{Z}^{(i)} = \left[\mathbf{z}_0^{(i)}, \mathbf{z}_1^{(i)}, \cdots, \mathbf{z}_T^{(i)}\right]$。

3.3.3. 分类器

对于有标签的学生，其对应样本为 $\left(X^{(i)}, s^{(i)}, y^{(i)}\right)$，将 $X^{(i)}$ 和 $s^{(i)}$ 输入到骨干网络中，得到 $Z^{(i)} = \left[z_0^{(i)}, z_1^{(i)}, \cdots, z_T^{(i)}\right]$，从中取出 $z_0^{(i)}$ 作为总体特征表示，将其输入到由隐含层全连接神经网络构成的分类器中，得到 $\hat{y}^{(i)}$ 作为分类结果。

$$\hat{y}^{(i)} = p\left(z_0^{(i)}\right) = \text{sigmoid}\left(W_{\text{cls},2}\left(\text{ReLU}\left(W_{\text{cls},1} z_0^{(i)} + b_{\text{cls},1}\right)\right) + b_{\text{cls},2}\right) \tag{16}$$

3.3.4. 模型训练过程

在模型训练过程中，首先开展自监督学习阶段，旨在通过重建任务增强模型对数据内部结构和特征分布的理解能力。在该阶段，原始数据首先被注入随机掩码噪声，以构造具有挑战性的训练样本，从而模拟实际应用中可能出现的缺失或干扰情况。带噪后的数据被输入至特征融合模块与预训练编码器中，提取包含时间序列模式的高维特征表示。随后，模型利用多层感知机对动态特征进行重建，以恢复被扰动的数据，并从特征表示中还原静态特征，从而确保编码器提取的表示不仅包含时间演化信息，也保留了个体的背景属性。

为此，训练过程设计了包含动态特征重建误差与静态特征重建误差的联合损失函数，通过对二者的加权求和，引导模型学习数据的时序依赖与结构特征。该阶段训练结果为后续有监督学习奠定了良好的特征表达基础，有助于提升整体模型的鲁棒性与泛化能力。具体流程见图 4。

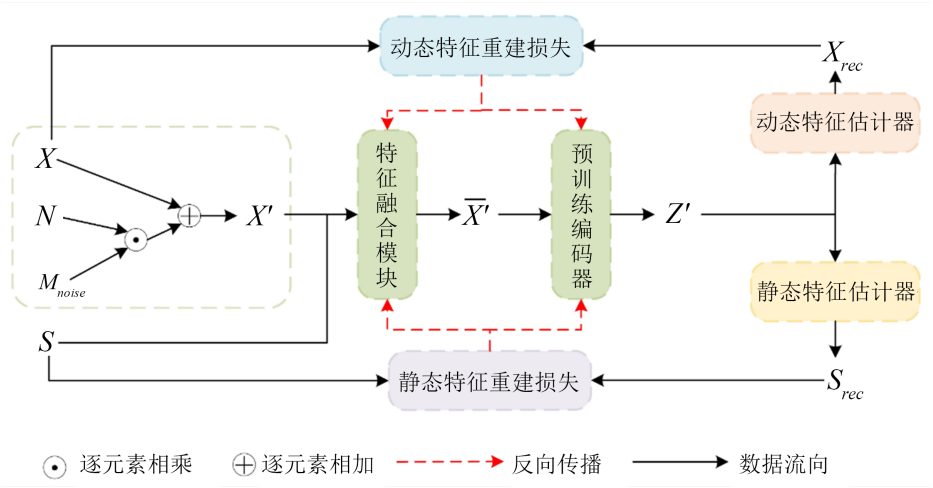

Figure 4. Schematic diagram of the self-supervised training process
图 4. 自监督训练过程示意图

3.4. 危机预警

在对学生心理状态进行初步判别后，系统基于二分类模型识别出存在心理健康风险的学生。一旦检测到潜在问题，系统将按照预设流程，生成预警信息并推送至辅导员及相关管理人员，以便及时介入并提供精准支持。系统根据预设的心理危机分级标准，生成不同级别的预警信息，确保多方协同介入，实现心理问题的提前预测与干预。

4. 基于大语言模型的心理状态评估和情感支持

在心理健康支持体系中，开展科学、动态、个性化的心理状态评估与情感支持，是实现早期识别、精准干预的关键路径。为突破传统测评方式在数据维度、交互模式与反馈响应上的局限，本文基于 DeepSeek 大语言模型架构，融合心理学知识体系与自然交互需求，构建了一个面向校园场景的智能化心理状态感知与疏导模型，提出心理状态评估与情感支持一体化的新范式。

通过融合学生历史对话内容与实时采集的多模态数据特征(包括文本、语音、视频)，模型能够动态识别个体情绪变化与深层心理需求；同时引入多角色 AI 代理机制，自动匹配情绪状态与心理疏导方案，生成贴合学生沟通偏好的个性化干预内容，显著提升心理支持的及时性、共情性与实效性。

4.1. 数据集构建

为支撑心理状态评估与情绪支持的多层次功能实现，本文围绕任务需求设计并构建了两类核心数据集：一类为面向大语言模型微调的心理对话数据集，用于增强模型在心理测评与情感疏导中的语言理解与生成能力；另一类为多模态辅助识别数据集，用作 CNN-LSTM 情绪识别模型的输入，以实现基于语音、视频等非语言信号的心理状态感知，构建多源信息融合的心理健康评估机制。

4.1.1. 大语言模型微调数据集

本数据集主要用于提升语言模型在校园心理健康场景下的交互自然性与专业性，数据构建涵盖两类

子任务：心理测评问答、多角色情感支持对话。

1. 心理测评数据集构建

本文对国际公认的心理量表，如 PHQ-9、DASS-21、SCL-90 进行了结构性转换。借助提示词工程，将量表条目改写为符合人机自然交互风格的提问表达。在完整保留评估维度与专业内涵的同时，显著提升了语言的亲和力与使用体验，克服了传统测评在交互形式上的限制。

2. 多角色情感支持数据集构建

本部分数据集基于开源语料如 PsyQA、CPsyCoun，结合本文设计的“角色档案”与“场景档案”，构建多轮、多角色的情绪疏导对话。在生成过程中，引入思维链提示策略，并严格参考认知行为疗法(Cognitive Behavioral Therapy, CBT)等心理干预原则，以保证对话的逻辑性、专业性与情绪连贯性。具体流程见图 5。

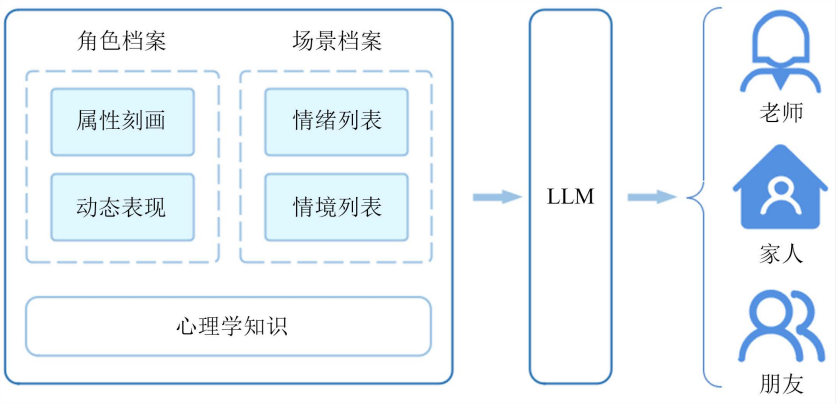

Figure 5. Multi-role emotional support dataset construction process
图 5. 多角色情感支持数据集构建流程

(1) 角色档案

对话式 AI 角色的关键在于深度地理解和模仿人类的交流，致力于创造逼真、可信且引人入胜的虚拟对话伙伴。所以在角色建模的过程中，本文将角色构成分为“属性刻画”与“动态表现”两大类，以构建稳定的且具有鲜明人格的对话代理。属性刻画包括身份、三观、经历；动态表现则体现在互动中的语言特征与性格。具体内容见表 3。

Table 3. Character files
表 3. 角色档案

角色定义	类别	具体内容
属性刻画	身份	姓名、性别、年龄、职业等
	三观	世界观、人生观、价值观
	经历	有意义的经历
动态表现	语言特征	口头禅、文风特点、常用词句等
	性格	温柔、冷漠等

(2) 场景档案

在多角色个性化疏导的设计中，仅依赖角色建模仍难以全面覆盖用户在实际交互中的多样化心理需

求。所以本文引入了场景档案的构建，旨在通过情绪与情境两个维度，系统地定义学生可能面临的心理状态与对话背景，为后续对话内容的生成与适配提供丰富且细致的上下文支持。具体内容见表 4。

Table 4. Scene files
表 4. 场景档案

场景定义	具体内容
情绪列表	快乐、悲伤、焦虑、愤怒、厌倦、困惑 同情、平静、欣赏、满足、尴尬、渴望
情境列表	学习、工作、家人、朋友、社交、恋爱 身心健康、兴趣爱好、日常生活、个人安全、未来

本文构建的多角色引入了“老师”、“家人”、“朋友”三种典型的角色，作为虚拟代理进行对话式的情感支持。通过角色属性和语言风格的设定，以及情绪、情境类别的组合，系统化生成多轮的对话样本，从而确保数据集在内容、风格与心理支持策略上的多样性与专业性。

4.1.2. 多模态辅助识别数据集

为支持基于非语言信息的心理状态识别，本文设计采集了文本、语音与视频三类模态的数据，主要作为 CNN-LSTM 识别模型的输入，以实现对学生情绪状态的综合感知。

数据采集基于统一交互平台完成，确保多模态数据在时间轴上的精确同步。文本模态通过记录用户输入的对话内容，完成分词、清洗与语义编码处理；语音模态由麦克风采集，提取梅尔频率倒谱系数(MFCC)、基频、能量等声学特征；视频模态通过摄像头捕捉面部表情与身体姿态，提取关键点位置信息与表情参数。

为确保模态之间信息的准确融合，系统对采集的多模态数据执行统一时间戳标记，进行精确的时间对齐处理。所有数据经标准化预处理流程，包括去噪、异常值剔除与归一化等操作，进一步提升特征提取的稳定性与输入的一致性。实验数据均来源于标准化采集流程，场景设计中充分考虑多样性与标签有效性，以支持后续模型训练的泛化能力。

4.2. 模型微调

为了进一步提高模型的专业适配性，本文基于上述构建的数据集，采用 LoRA (Low-Rank Adaptation) 技术对基础大语言模型 DeepSeek：R1-14B 进行高效微调。LoRA 通过在保持原有模型参数不变的前提下，仅对少量新增的低秩矩阵进行训练，从而显著降低了微调过程中的计算与存储开销，适合在资源受限的环境中部署和优化。同时集成了高效训练框架 Unsloth，以进一步加速训练流程。

通过将 LoRA 与 Unsloth 结合使用，本文在保证模型性能的基础上，实现了心理任务领域下的轻量化、高效率模型适配，为后续部署于校园的心理服务系统提供了坚实的模型基础。

4.3. 动态交互式评测

在完成对标准化心理量表的对话式转化后，本文进一步实现了面向真实应用场景的动态交互式评测。与传统的纸质问卷、人工统计评分的流程不同，本文借助了深度微调的大语言模型，通过自然语言的引导性提问与实时反馈机制，构建了智能化、自适应的交互式评测路径。

在评测过程中，模型以提示词为引导，以问卷条目为核心，结合情感识别，将每一条量表问题转化为自然的对话单元，并在用户作答后进行内容解析与情感识别。根据用户回答中出现的频率词汇和情绪表达，模型能够在对应症状维度进行即时得分。例如，当用户在睡眠质量相关的问题中使用了“经常惊

醒”、“多梦”等描述，模型就会在相应的指标上进行加分。各评测项目归属于不同的心理症状类别，如焦虑、抑郁、压力等，模型在对话的过程中逐步累积各维度得分，最终依据设定的分数阈值判断风险等级(例如，焦虑量表得分超过 10 分即判定为轻度焦虑)。

为了进一步提升评测的敏感性与精准性，本文设计了基于初筛结果的追问机制。当用户在某一指标上表现出了较高的风险倾向时，模型能够切换到针对性更强的细化问卷，展开更深层次的评测。具体示例见图 6。这种即时调整的策略有效地缩短了发现心理问题的时间差，提升了整体评测的效率与深度。

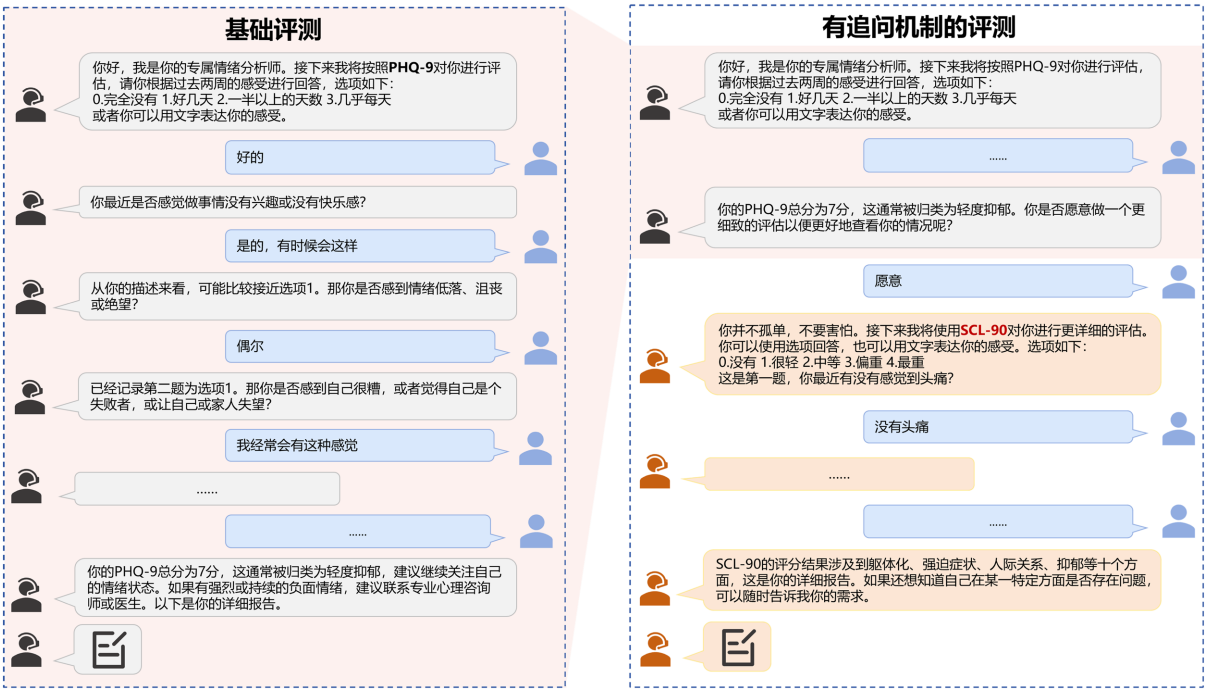

Figure 6. Comparison between basic evaluation and evaluation with follow-up mechanism
图 6. 基础评测与有追问机制评测的对比

此外，系统支持大规模在线评测与实时数据统计。可在短时间内完成对新生群体的心理普查，快速筛查出潜在的高风险个体，并对其心理状态变化进行持续跟踪，为后续心理干预提供数据支撑。

4.4. 智能化报告生成

在完成心理状态评测后，本文进一步设计了结构化、个性化的反馈机制。旨在为用户提供积极的心理暗示，提升他们的体验感，减少因为评测结果而引发的负面情绪。

系统支持 24 小时不间断评测服务，会主动关注深夜进行测试的用户。在特殊时段，模型在保持专业评测的同时，会根据用户的回答情绪适度提供共情反馈。在用户完成全部问题后，模型会自动生成鼓励性的话语以结束对话，如“恭喜你顺利完成了这次评测，下面一起来看看你的专属心理体检报告吧！”若检测到用户情绪低落，模型则会调整语气与表达方式，避免使用过于欢快的表述，以防用户产生反感或情绪抵触。

在正式反馈阶段，本文将传统的评测分数转化为具象化、游戏化的个性化心理体检报告。为了打破冰冷数据带来的疏离感，系统会根据用户的综合情绪状态自动生成相应风格的卡通角色形象。如压力值高时角色形象呈现出凌乱疲惫的状态，抑郁倾向明显时则呈现低落内敛，心理状态良好时则展现活泼轻松。这种具象化表达方式可以拉近系统与用户之间的距离，给予用户柔性化、正向的心理暗示，缓解因

心理评测带来的潜在不安情绪。

在报告生成后，系统还会进一步基于个体的评测结果提供针对性、个性化的建议。例如，针对“难以开始工作”的用户推荐他试试番茄工作法；对于有“社交焦虑”表现的用户则推送给他破冰对话技巧清单；对存在“入睡困难”的用户可以推荐白噪音助眠资源等，具体示例见图 7。对于心理风险等级较高的用户，报告中会标注紧急心理求助的热线信息，确保用户在需要时能够及时地获得专业的帮助。这为构建友好、可持续的心理健康支持体系提供了有力支撑。

Figure 7. Intelligent reporting
图 7. 智能化报告

4.5. 多角色个性疏导

为了实现对学生的个性化疏导，本文设计并实现了系统化、分阶段的智能情感支持体系。本模块整体架构由三个核心阶段构成：情绪识别、策略推理与回复生成。模型在充分理解用户心理状态的基础上，采用相应的对话策略生成支持性回复，在多轮对话的过程中帮助用户降低情绪困扰。各阶段相互协作、逐步推进，形成智能化心理疏导流程。完整的模块架构及处理流程如图 8 所示。

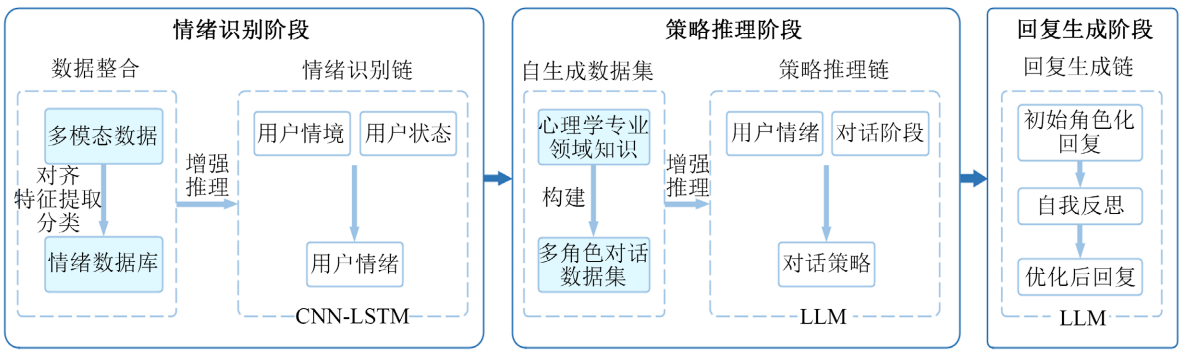

Figure 8. The overall framework of multi-role personality guidance
图 8. 多角色个性疏导的整体架构

4.5.1. 情绪识别阶段

在前述多模态数据采集与特征提取的基础上，本文进一步构建了系统化的学生情绪数据库，并提出了一种基于卷积神经网络与长短时记忆网络(CNN-LSTM)的多模态情绪识别方法，用于实现学生心理状态的实时、精准识别。

该方法融合了 CNN 在局部特征提取方面的优势与 LSTM 在建模时间动态特征方面的能力，通过特征级融合与跨模态注意力机制对多源特征进行权重动态调整，突出与心理状态高度相关的关键因素。最终通过全连接分类器输出用户的情绪状态，实现即时且高精度的情绪识别。具体流程见图 9。

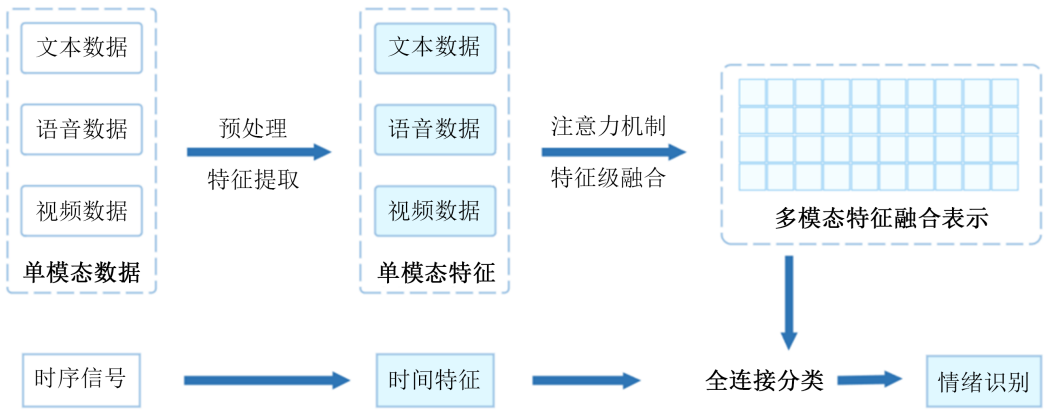

Figure 9. Multimodal emotion recognition process
图 9. 多模态情绪识别流程

整体模型结构包括局部特征提取、特征融合、时序建模与注意力加权四个阶段，具体描述如下：

首先，对于每种模态提取得到的特征表示 $\mathbf{x}_t^{(m)}$ (其中 m 表示模态类型，t 表示时间步)，通过一维卷积核 $\mathbf{W}_c$ 进行局部特征提取：

$$\mathbf{f}_t^{(m)} = \mathrm{ReLU}\left(\mathbf{W}_c * \mathbf{x}_t^{(m)} + \mathbf{b}_c\right) \tag{17}$$

其中，$\mathbf{f}_t^{(m)}$ 为卷积后的模态局部特征，$*$ 表示卷积操作，$\mathbf{b}_c$ 为偏置项。

随后，将所有模态在相同时间步上的特征拼接形成融合特征输入：

$$\mathbf{F}_t = \left[\mathbf{f}_t^{(1)}, \mathbf{f}_t^{(2)}, \cdots, \mathbf{f}_t^{(M)}\right] \tag{18}$$

该融合特征序列被输入至双向长短时记忆网络(Bi-LSTM)，以捕捉跨时间的心理状态动态演化特征：

$$\mathbf{h}_t = \mathrm{BiLSTM}\left(\mathbf{F}_t\right) \tag{19}$$

为增强模型对关键时刻与关键模态的关注能力，本文在时序建模基础上引入注意力机制，计算注意力权重 α_t 并将加权聚合隐藏状态表示为：

$$\alpha_t = \frac{\exp\left(\mathbf{v}^\top \tanh\left(\mathbf{W}_a \mathbf{h}_t + \mathbf{b}_a\right)\right)}{\sum i \exp\left(\mathbf{v}^\top \tanh\left(\mathbf{W}_a \mathbf{h}_i + \mathbf{b}_a\right)\right)} \tag{20}$$

$$\mathbf{H}_{att} = \sum_t \alpha_t \mathbf{h}_t \tag{21}$$

最终，将注意力加权后的融合特征表示 $\mathbf{H}_{att}$ 输入至全连接层，通过 Softmax 函数输出情绪分类结果：

$$\hat{y} = \mathrm{Softmax}\left(\mathbf{W}_o \mathbf{H}_{att} + \mathbf{b}_o\right) \tag{22}$$

通过上述结构，模型实现了从局部模态特征提取到时序建模再到情绪状态分类的完整识别链，兼具细节表达与动态理解能力。随后将识别结果作为输入反馈给大语言模型，为后续内容生成阶段提供情绪感知支持。

4.5.2. 策略推理阶段

在学生心理疏导对话的过程中，策略选择的合理性直接影响交互的有效性与支持的精准性。本文在前述情绪识别阶段给出用户情绪的基础上，结合当前与用户的对话阶段(如初始接触、情绪表达、疏导建议等)，共同构成策略推理的语境依据。

例如，在对话初期阶段，若用户情绪状态低落，系统将优先选用以“建立信任、传递支持感”为核心的引导型策略，生成体现倾听、认同与鼓励的响应内容，以降低用户的心理防御，促进情感表达。而在对话中后期，若检测到情绪逐渐趋于稳定，模型则可适时引入认知重构、问题解决等具有引导性的策略，以协助用户理性分析问题与形成积极的应对机制。

在准确捕捉学生情绪的基础上，经过学习由心理学专业领域知识指导生成的多角色对话数据集，并掌握其中基于认知行为疗法(CBT)推理范式的训练后，情感支持大语言模型具备匹配当前用户情绪特征的对话策略样本及相关知识。通过构建由对话阶段到对话策略的推理链，模型能够根据实时对话阶段与用户情绪状态，自适应生成最合适的对话策略。例如，在对话初期，模型更倾向于通过建立信任感主动引导交流。策略推理阶段对筛选和编码后的对话策略样本进行规范化处理，确保其准确反映策略要点及与学生情绪的契合度，并为后续语言生成环节提供高质量输入。

4.5.3. 回复生成阶段

在完成用户情绪识别与对话策略推理后，进入回复生成阶段。此阶段以大语言模型为核心，在多因素驱动下生成具备情绪感知与策略引导能力的响应内容。具体生成过程充分融合三类信息：一是识别出用户当前的情绪状态；二是当前所处的对话阶段及匹配策略；三是用户预设的情感支持角色(老师、朋友、家人)。

模型生成初步的角色化回复，回复生成过程中引入语言模型的自我反思机制，结合情感共鸣调控，动态检查生成内容是否真正贴合学生的情绪状态，语言表达是否符合其沟通偏好等要求。根据反思结果，进一步优化初步回复，例如将表达生硬的语句调整为更加亲和、安抚性的表述。最终输出的优化回复能够灵活地适应学生的个性化心理需求，有效提升情感疏导的针对性与干预成效。

5. 系统应用与测试

5.1. 学生异常情绪及行为监测与预警系统

本文基于校园大数据的实时心理状态监测设计并实现了学生异常情绪及行为监测与预警系统，如图10 所示。该系统通过动态化捕捉学生的多维度行为数据，融合时间序列建模与异常检测技术，实现对学生潜在异常情绪与行为的精准识别与实时预警，旨在为校园心理健康管理提供智能化、系统化支持。

目前，该系统已与智慧校园平台实现集成，覆盖超过两万名学生样本，累计采集特征数据规模过亿。与常规心理筛查结果进行对比分析显示，本系统的心理异常识别准确率达到 85%以上，验证了其实用性与有效性。

为了验证本系统中采用的时序模型在心理危机检测中的有效性，本文选择了以下几种针对时序数据的基线方法进行对比。

DTW-1NN 是基于动态时间规整(DTW)距离的最近邻算法，是处理时间序列分类的经典方法。

LSTM 是长短期记忆网络，专门设计用于捕捉序列数据中的长期依赖关系。

TSIN [3]该方法提出了一种基于双流 Informer 网络的学生心理健康预测模型，通过时间编码器捕捉时间序列的周期性趋势，行为编码器分析行为特征间的依赖关系，并利用中间融合模块和门控机制合并两个编码器输出。

TS2VEC [4]是一种基于对比学习的时间序列表示学习方法，通过最大化相似样本间的相似度来学习时间序列的表示。

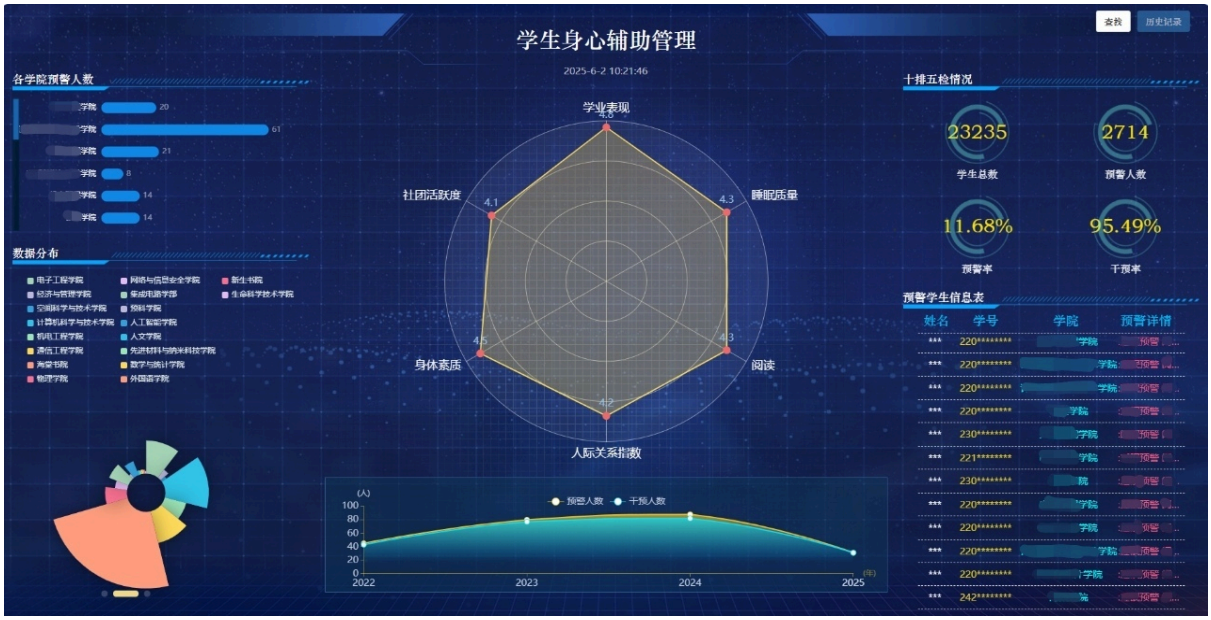

Figure 10. Students' abnormal emotions and behaviors monitoring and early warning system
图 10. 学生异常情绪及行为监测与预警系统

Table 5. Performance comparison of different methods
表 5. 不同方法的性能比较

方法	Accuracy	Precision	Recall	F1 Score
DTW-1NN	0.7828	0.7792	0.5933	0.6617
LSTM	0.8138	0.7857	0.6600	0.7174
TSIN	0.8533	0.7793	0.7533	0.7661
TS2VEC	0.8425	0.8088	0.7333	0.7692
Ours	0.8783	0.8510	0.8000	0.8247

表 5 展示了不同方法在测试集上的性能表现，本方法在所有评价指标上均优于其他方法。相比于传统的 DTW-1NN 方法，本方法的准确率提升了 9.55 个百分点，F1 分数提升了 16.3 个百分点；相比于深度学习方法 LSTM，准确率提升了 6.45 个百分点，F1 分数提升了 10.73 个百分点；相比于 TSIN 方法，准确率提升了 2.5 个百分点，F1 分数提升了 5.86 个百分点；相比于自监督时序学习方法 TS2VEC，准确率提升了 3.58 个百分点，F1 分数仍有 5.55 个百分点的提升。这些结果表明，本方法在捕捉学生行为的时序模式方面和无标签数据有效利用方面具有显著优势。

5.2. 心知心语平台

本系统旨在构建多终端、全场景的沉浸式心理疏导环境，为用户提供灵活多样的对话体验选择。系

统支持 Web 端、移动端和 MR 端独立运行，并实现数据互通，构建立体化的心理健康服务网络，为学生提供长期、动态的心理状态追踪服务，并建立个性化的心理健康档案。

5.2.1. Web 端与移动端

Web 平台和移动端均提供轻量化、便捷化的访问方式，采用极简交互设计，突出对话内容本身，确保交互过程的流畅性和专注度。为学生提供 7 × 24 小时不间断的心理健康支持，显著提升服务的即时性和可及性。如图 11 所示，分别为系统的登录界面、模块选择界面、聊天界面以及报告界面。

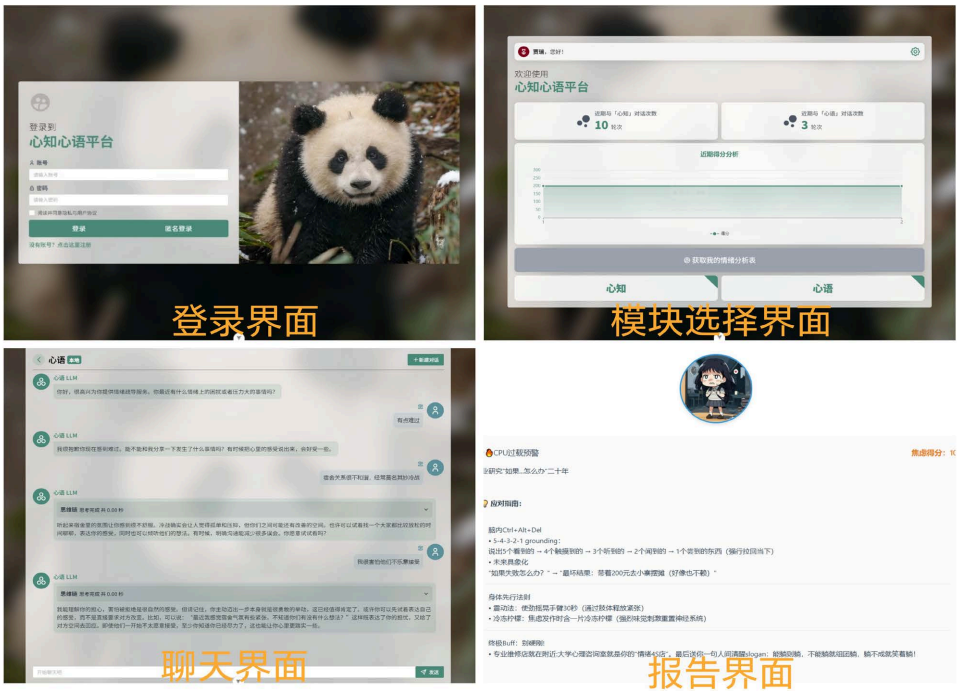

Figure 11. Interface display
图 11. 界面展示

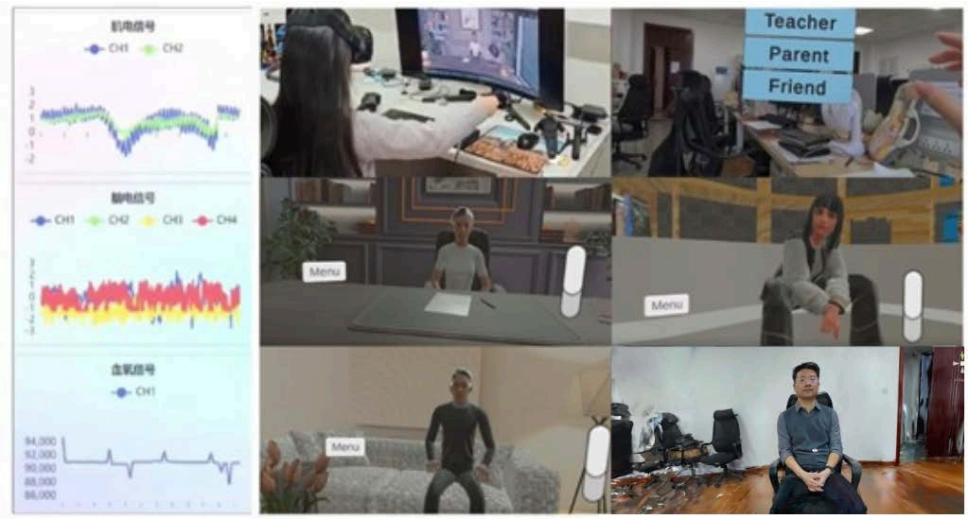

Figure 12. MR interactive display
图 12. MR 交互展示

5.2.2. MR 端

系统针对 VR 头显和 VisionPro 设备进行了深度适配，依托混合现实(MR)技术构建沉浸式心理疏导环境。通过高精度 3D 建模和实时渲染技术，创造逼真的虚拟对话场景，增强心理干预的深度与效果。系统支持自然语言交互、手势识别等多种交互方式，提供与虚拟角色面对面的深度交流体验，如图 12 所示。未来计划进一步拓展虚拟场景的选择范围，通过精心设计的视觉元素、环境音效和交互细节，创造更具疗愈效果的对话空间。

6. 结束语

在人工智能时代背景下，将 AI 技术引入心理健康服务体系，为心理问题的早期识别、精准评估与个性化干预提供了全新思路。本文围绕智能化心理监测、评估疏导全过程，构建了融合校园大数据与情感数字人的智能心理支持系统，显著提升了学生心理状态识别的实时性与准确性，切实减轻了高校学生管理工作的压力。为高校心理健康教育体系的智能化、科学化转型提供了可行路径与技术支撑，具有良好的推广前景与现实意义。

基金项目

本研究得到了陕西省重点研发重点产业创新链(群)项目(2024GX-ZDCYL-02-15)、陕西省杰出青年科学基金(2025JC-JCQN-079)的支持。

参考文献

[1] 王蜜源, 韩芳芳, 刘佳, 等. 大学生抑郁症状检出率及相关因素的 meta 分析[J]. 中国心理卫生杂志, 2020, 34(12): 1041-1047.

[2] 傅小兰, 张侃, 陈雪峰, 等. 中国国民心理健康发展报告(2021~2022) [M]. 北京: 社会科学文献出版社, 2023.

[3] Xu, J.M., Ding X.F., Ke, H.Y., *et al*. (2023) Student Behavior Prediction of Mental Health Based on Two-Stream Informer Network. *Applied Sciences*, **13**, Article 2371. https://doi.org/10.3390/app13042371

[4] Yue, Z., Wang, Y., Duan, J., Yang, T., Huang, C., Tong, Y. and Xu, B. (2022) TS2Vec: Towards Universal Representation of Time Series. *Proceedings of the* 36*th AAAI Conference on Artificial Intelligence*, 22 February-1 March 2022, 8980-8987. https://doi.org/10.1609/aaai.v36i8.20881

Embedded Technology and Intelligent Systems 嵌入式技术与智能系统, 2025, 2(2), 96-114
Published Online April 2025 in Hans. https://www.hanspub.org/journal/etis
https://doi.org/10.12677/etis.2025.22008

Hans汉斯

构建面向AGI时代的开源IoT AI智能体架构与实践

吴　薇[1,2]，曹宇伟[1]，区卉贤[2]，王坚豪[2]，徐朦饶[2]，胡雯轩[2]，陈睿轩[2]，邢成龙[2]，杨灵益[2]

[1]江苏矽望电子科技有限公司，江苏　南京
[2]杭州电子科技大学电子信息学院，浙江　杭州

收稿日期：2025年4月1日；录用日期：2025年4月23日；发布日期：2025年4月30日

摘　要

随着大语言模型(LLM)及边缘计算技术的发展，AI智能体(AI Agent)正逐步成为物联网(IoT)系统中的核心调度与控制单元。文章设计并实现了一套以AI智能体为核心的人工智能物联网(AIoT)系统架构，融合传感器/执行模块、边缘终端、本地/云端LLM推理引擎、云计算中心与n8n自动化平台。系统采用模块化设计，支持MCP调度架构与OPC UA、MQTT等协议通信，具备低延迟、高可扩展性和良好的工程可移植性。并重点介绍了系统构成、核心模块设计，以智能家具为典型应用实例及部署实验，展示其在各种AIoT行业应用场景下的实用性和开放性。

关键词

AI智能体，MCP，边缘计算，IoT模块，轻量级LLM，开源架构

Design and Implementation of an Open-Source IoT AI Agent Architecture for the AGI Era

Wei Wu[1,2], Yuwei Cao[1], Huixian Ou[2], Jianhao Wang[2], Mengrao Xu[2], Wenxuan Hu[2], Ruixuan Chen[2], Chenglong Xing[2], Lingyi Yang[2]

[1]Cynoware Electronics, Inc., Nanjing Jiangsu
[2]School of Electronic Engineering, Hangzhou Dianzi University, Hangzhou Zhejiang

Received: Apr. 1st, 2025; accepted: Apr. 23rd, 2025; published: Apr. 30th, 2025

Abstract

With the advancement of Large Language Models (LLMs) and edge computing technologies, Artificial

文章引用: 吴薇, 曹宇伟, 区卉贤, 王坚豪, 徐朦饶, 胡雯轩, 陈睿轩, 邢成龙, 杨灵益. 构建面向 AGI 时代的开源 IoT AI 智能体架构与实践[J]. 嵌入式技术与智能系统, 2025, 2(2): 96-114. DOI: 10.12677/etis.2025.22008

Intelligence Agents (AI Agents) are gradually becoming the core scheduling and control units in Internet of Things (IoT) systems. This paper designs and implements an AIoT system architecture centered around AI agents, integrating sensor/actuator modules, edge terminals, local/cloud-based LLM inference engines, cloud computing centers, and the n8n automation platform. The system adopts a modular design, supports the MCP scheduling framework, and communicates via protocols such as OPC UA and MQTT, offering low latency, high scalability, and strong engineering portability. This paper focuses on the system structure and core module design, and demonstrates its practicality and openness through a typical use case in smart home applications and deployment experiments, showcasing its potential across various AIoT industry scenarios.

Keywords

AI Agent, MCP, Edge Computing, IoT Module, Lightweight LLM, Open-Source Architecture

1. 引言

物联网(Internet of Things，文中简称 IoT)和人工智能(Artificial Intelligence，文中简称 AI)结合的人工智能物联网(文中简称 AIoT)作为当下智能社会的技术支柱，正与智慧城市与产业自动化等领域深度融合，广泛应用于工业制造、能源管理等对实时性与稳定性要求极高的行业，同时不断延伸至智能家居、适老健康监护等民生场景。然而，在生成式大语言模型 (Large Language Model，文中简称 LLM)取得突破之前，AIoT 系统受限于边缘设备的算力瓶颈与语义理解能力不足，主要依赖如 AlexNet、CNN 或客制化轻量化 AI 模型等进行感知与推理。这种技术路径导致 AIoT 系统普遍面临以下三大核心问题：一是任务固化(系统只能适应单一场景任务)；二是语义割裂(环境感知与决策过程相互分离，缺乏统一理解)；三是响应滞后(对用户动态需求和环境变化的适应能力有限)。

随着通用人工智能 AGI (Artificial General Intelligence，文中简称 AGI)技术的演进，AIoT 系统正迈向 AI 智能体(AI Agent，文中简称 AI 智能体或 AI Agent)发展，使得构建具备语义理解、工具组合调用与自我学习适应能力的“嵌入式智能体系统”成为可能。本文提出一种面向 AGI 时代的开源 AIoT 系统架构，集成 AI 智能体、MCP (Multi-Component Processing，多组件任务规划器，文中简称 MCP)中间件框架、IOT 边缘感知处理能力、轻量语言模型以及云平台可视化等功能，形成端－边－云协同的智能体系。本文以智能家居场景为例，对该系统进行了整个项目模块的集成实现与测试，验证了其在实际应用中的可行性与扩展潜力。项目软件系统将在 GITHUB 上公开，旨在与同行一起将此架构改进、完善并进行推广。作者单位之一的江苏矽望电子科技有限公司(文中简称矽望科技)将可为同行提供优惠的配套调试硬件。

2. IoT AI 智能体

IoT AI 智能体是一类部署于边缘或终端设备中的智能系统，具备环境感知、智能推理与行动执行三大核心能力。它融合了多模态感知技术与人工智能推理模型，能够对所处环境进行实时分析，并据此做出自主决策与响应操作，形成感知－决策－行动的闭环控制体系。

在功能上，IoT AI 智能体主要承担以下任务：

- 环境感知(Perception)：利用温度、湿度、图像、声音、光照等传感器采集环境信息，为后续推理提供

数据支持；

- 智能推理与决策(Reasoning and Decision-making)：通过部署轻量化 AI 模型(如卷积神经网络、小型语言模型或规则引擎)或通过本地 LLM 或云端 LLM 对感知数据进行分析，理解用户意图或判断环境状态，做出最优控制决策；
- 行动执行(Action Execution)：根据推理结果，驱动电机、继电器、显示器或控制信号，实现对外部世界的智能干预；
- 循环机制(Sense-Think-Act Loop)：上述功能以闭环形式持续运行，实现系统对动态环境的实时适应与响应。

该智能体系统的整体运行机制可概括为：“感知(Sense)→推理与决策(Think)→行动(Act)”，构成一个具有自适应能力的智能行为循环。其基本功能框图如图 1 所示，涵盖了从数据输入到控制输出的完整处理流程。

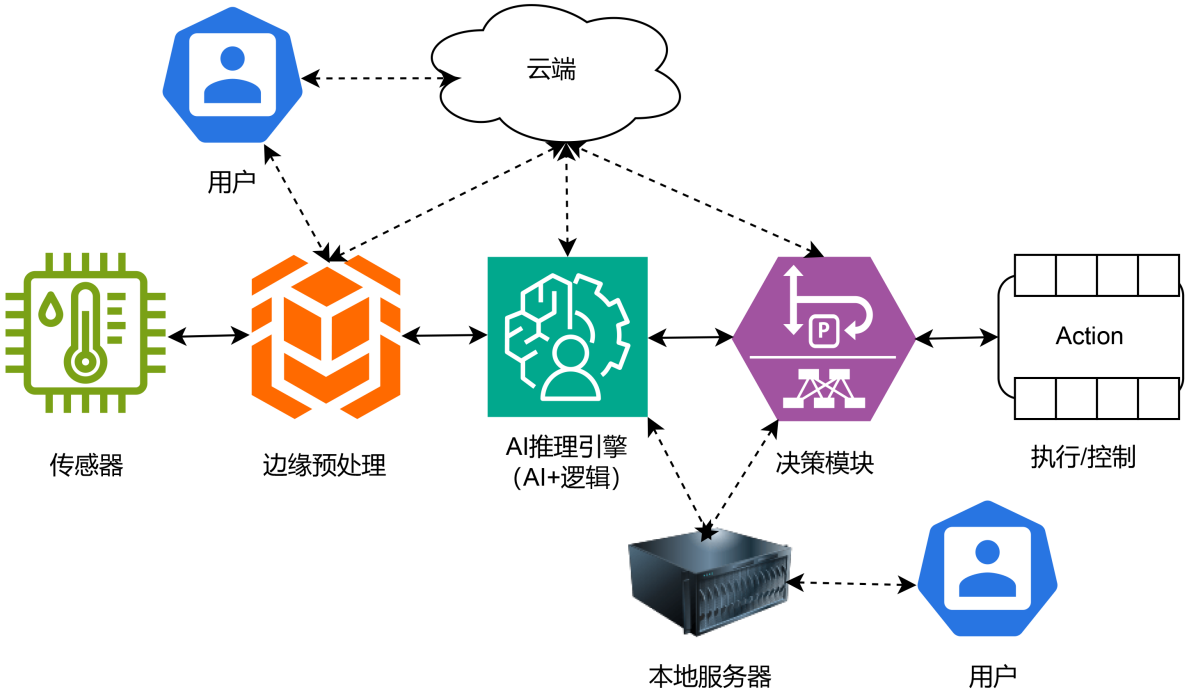

Figure 1. Functional block diagram of the IoT AI agent system
图 1. IoT AI 智能体系统的功能框图

通过引入 AI 智能体机制，IoT 系统的自主性与智能化水平得以显著提升，为智能家居、工业自动化、智慧医疗等领域带来了更具扩展性与适应性的解决方案。

3. 系统总体架构

典型的物联网架构有三、四、五、六、七层，其中图 2(a)的七层模型定义在论文[1]中有描述。值得指出的是，没有单一的、公认的物联网架构和层定义。根据特定的应用场景和业务任务的复杂性，物联网架构层数、层定义会有所不同[2]。

通过对图 2(a)进行简化，得到本文的 IoT AI 智能体系统架构，如图 2(b)所示，它包括传感器和执行模块、边缘终端、云平台、本地轻量级大语言模型 LLM、云端 LLM 和 AIoT 行业应用客户。

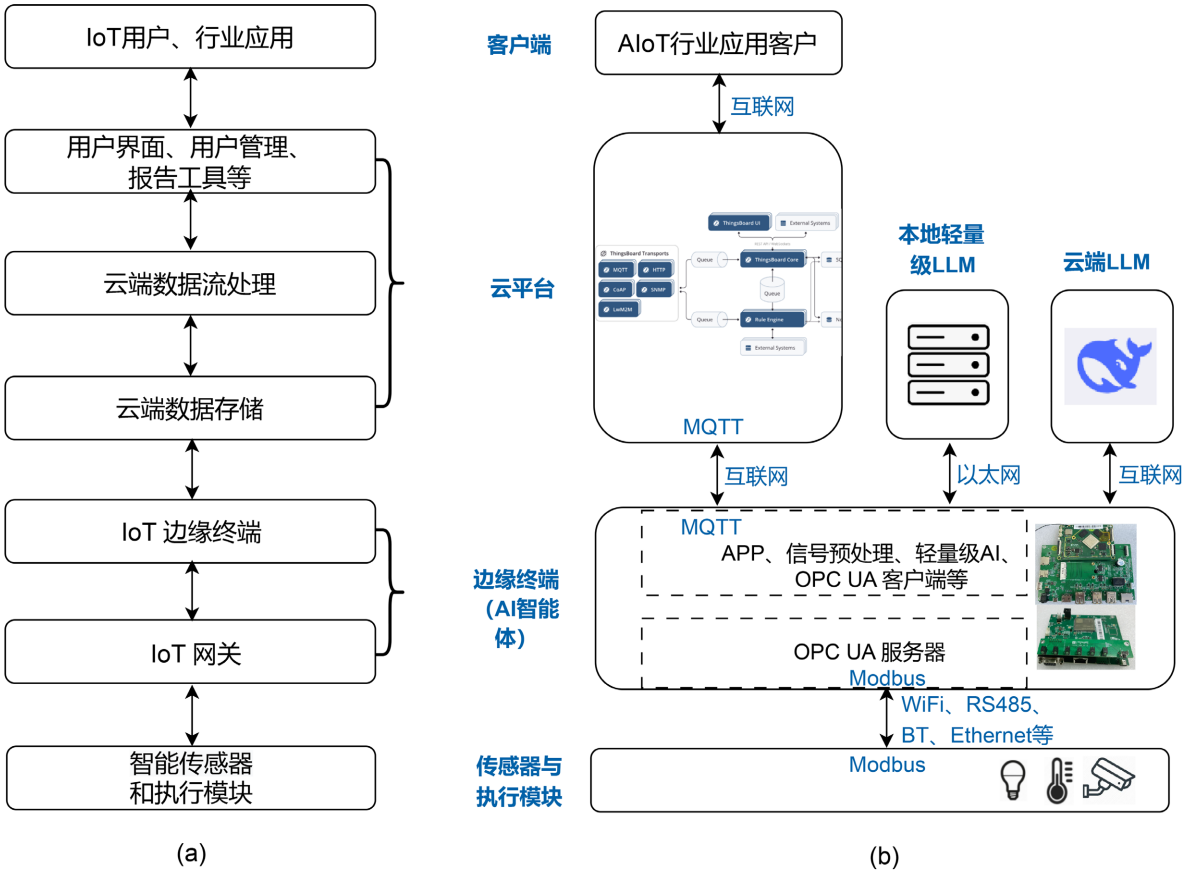

Figure 2. IoT AI agent system architecture
图 2. IoT AI 智能体系统架构

第一层传感器与执行模块和第二层具有物联网网关功能的边缘终端之间支持 Wi-Fi、蓝牙(BT)、RS485、以太网等多种无线和有线通信，建议主要使用 Modbus 数据通信协议。第二层和第三层云平台之间通信使用互联网的 MQTT 协议。第三层和第四层 AIoT 用户和行业应用客户端之间通信则可通过 WEB 客户端或 REST API 来实现。本地轻量级 LLM 可以使用 DeepSeek 小模型或 QwQ 等安装在本地服务器，以提高系统反应速度。表 1 描述了系统中各个组件的作用和示例。

Table 1. Components and functions of the IoT AI agent system
表 1. IoT AI 智能体系统的组件和作用

组件	作用	示例
传感器(输入)	从环境中收集数据	温度传感器、湿度传感器、运动检测器、摄像头等
AI 模型/算法(大脑)	分析数据并做出决策	机器学习模型、决策树、深度学习、大模型
本地处理(边缘计算)	在设备附近快速处理数据	单片机、Linux 终端、安卓终端等
执行器(输出)	根据决策执行动作	打开/关闭灯光、调节温控、发送通知等
通信模块	与其他设备或云端进行信息传输	Wi-Fi、蓝牙、Zigbee、MQTT、OPC UA、HTTP API、MCP 等
知识库(可选)	存储历史经验或预定义规则	本地数据库、云存储
用户界面(可选)	让用户可以与 AI Agent 交互	触摸屏、手机 App、语音指令等

4. 关键模块设计

4.1. 传感器和执行模块

第一层传感器和执行模块(文中简称 IoT 模块)主要完成对环境数据采集、信号预处理、执行器驱动及边缘终端通信[3]。如图 3 所示。该模块通过混合通信架构(无线/有线)与边缘计算终端进行数据交互，采用 Modbus 协议实现标准化数据传输，确保系统的兼容性与可扩展性。

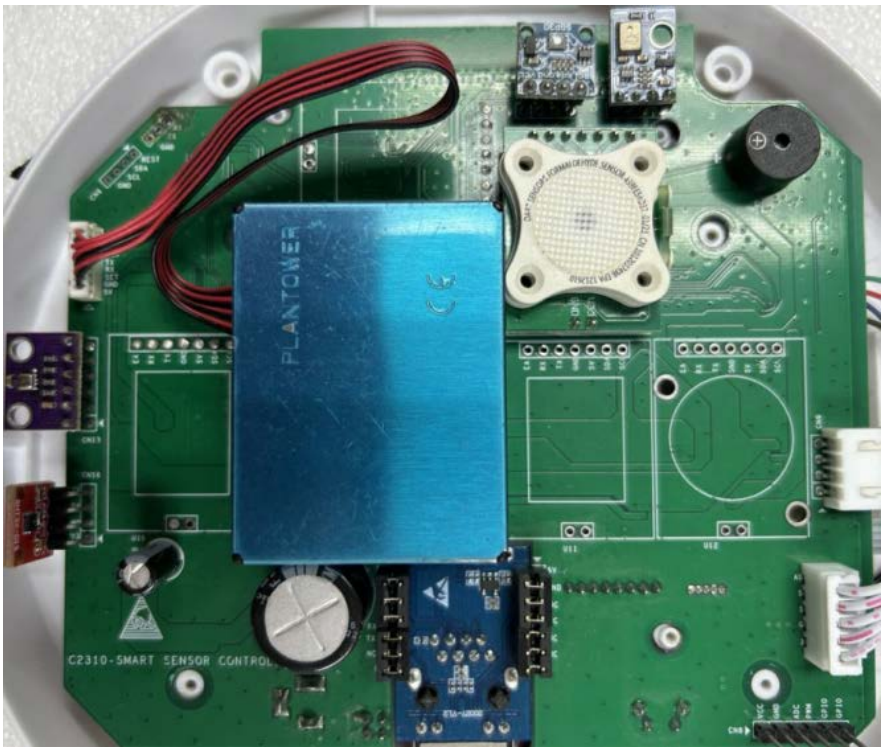

Figure 3. Sensors and actuator modules of the IoT AI agent system
图 3. IoT AI 智能体系统的传感器和执行模块

4.1.1. 硬件核心平台

IoT 模块采用国产 RISC-V 架构 W801 芯片作为主控单元，其技术特性包括：

- 处理性能：32 位 CPU 内核搭配 20KB ROM + 288KB SRAM + 2MB Flash 的三级存储架构，支持实时数据采集(采样率 ≥ 1 kHz)及简单算法如 PID 等控制算法的本地化执行。
- 接口扩展：集成 UART × 3、SPI × 2、I2C × 2、ADC × 5 等多类型数字接口，最大支持 10 个传感器/执行器的并行接入。
- 通信能力：双模无线支持(Wi-Fi 802.11b/g/n + BT/BLE 4.2)，有线接口包含 10/100M 以太网 PHY 和 RS485 收发器。
- 能效管理：动态电压调节技术实现待机功耗 < 10 μA，工作模式功耗 ≤ 120 mW@48 MHz。

4.1.2. 传感器与执行机构接口

模块通过标准化接口协议实现传感器网络的灵活配置：

- 模拟信号采集：16 位高精度 ADC 通道(采样率 1 MSPS)支持温度、湿度、光照等模拟传感器。
- 数字接口扩展：I2C 总线挂载大气压、CO_2浓度等数字传感器。

- SPI 接口连接工业级 IMU (惯性测量单元)。
- GPIO 驱动继电器、电磁阀等执行机构。
- 特殊信号处理：内置可编程增益放大器(PGA)支持 mV 级微弱信号检测。

4.1.3. 通信机制

模块采用分层通信架构：

- 物理层：无线传输选用 Wi-Fi (最大速率 72 Mbps)实现高带宽数据传输，BLE 4.2 用于低功耗设备组网，有线通信采用 RS485 (最大节点数 256 个，传输距离 1.2 km)。
- 协议层：基于 Modbus-TCP/RTU 协议实现数据帧标准化封装，定义设备状态字(16 bit)、传感器数据区(32 bit 浮点数组)、控制指令区(8 bit 命令码)三类寄存器。
- 安全机制：AES-128 硬件加密引擎保障数据传输安全，支持双向身份认证。

4.1.4. IoT 模块软件

W801 的 SDK 中内置有 FreeRTOS 实时操作系统。FreeRTOS 是一个开源嵌入式实时操作系统，其具有小巧简单、可移植性强的特点，使得它成为目前市场占有率最高的 RTOS (Real Time Operating System) [3]。FreeRTOS 支持多任务并发运行，它通过优先级管理、时间片轮转调度等机制管理多任务并确保 CPU 的高效利用。FreeRTOS 还提供了一系列的功能和服务，例如任务管理、时间管理、消息队列、信号量、互斥锁等，基于这些功能和服务开发人员能够更方便地进行嵌入式应用开发。

本文设计的 IoT 模块软件的任务流程如图 4 所示。软件启动后，系统创建了 N 个主要任务：任务 1 负责连接并监测网络，任务 2 负责监测 TCP 连接状态，任务 3 为 TCP 数据接收和处理，以及任务 N 等。

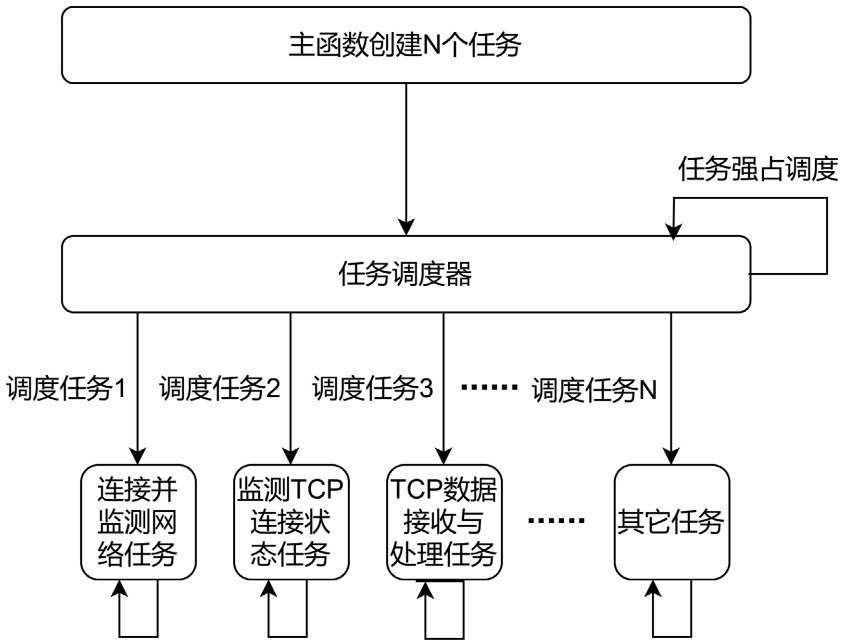

Figure 4. Task flow of the IoT module software
图 4. IoT 模块软件的任务流程

4.2. 具有物联网网关功能的 AI 边缘终端

边缘终端作为 AIoT 系统的核心边缘节点，不仅承担着多种通信协议(如 BT、Wi-Fi、RS485、Ethernet

等)下的设备接入与协同任务，还负责对环境数据预处理与协议统一。在 AI 能力层面，边缘终端集成本地轻量化 AI 模型(如 AlexNet、CNN 等)，用于进行初步的语义理解与智能推理，提升系统在断网或低延迟场景下的自适应能力。此外，本论文边缘终端内部通常还部署 MCP 与 AI 智能体执行框架，支持感知－推理－执行的闭环控制。其还内置 OPC UA 客户端/服务器、MQTT 客户端等标准工业协议模块，用于与 IoT 设备和云端平台(ThingsBoard)之间的数据交互与同步。

在面向 AGI 时代的系统演进中，边缘终端正从“数据中继器”转型为“本地智能体宿主”，具备更强的上下文理解、多模态感知与自主决策能力，成为真正意义上的“嵌入式认知节点”。

矽望科技近些年开发了采用十多款不同 CPU 的 Android 和 Linux 边缘终端。本文中系统由高性价比的国内主流终端芯片 RK3399 和 EC200A 的 Android 和 Linux 构成。

4.2.1. 边缘终端硬件

图 5 展示了矽望科技为支撑本文开源 IoT AI 智能体架构的边缘终端硬件系统框图，其核心以高性能嵌入式处理器(如 RK3399、EC200A 或其他 SoC)为中心，围绕处理器构建了五大模块：存储单元、网络与通信、用户接口、传感器扩展、电源管理。以下是各部分功能描述：

- 核心处理器(CPU)
  - 采用 Rockchip RK3399 或其他多核处理器，集成 GPU/NPU，支持本地 AI 推理。
  - 承担系统运行、数据处理、协议解析、AI 模型推理与边缘控制等核心任务。
- 存储模块(Storage)
  - DDR：用于系统运行所需的临时内存。
  - iNand：嵌入式非易失性存储，存放操作系统、模型与应用程序。
  - TF Card：支持外置存储卡扩展，便于数据日志存储与软件更新。
- IoT 网关与互联网通信(IoT Gateway & Internet)
  - RS485/RS232：适配传统工业串口设备，支持 Modbus 等协议。
  - Wi-Fi & BT：实现无线连接与蓝牙设备通信。
  - Ethernet：用于有线高速网络连接。
  - USB：支持外设接入(如摄像头、传感器、调制解调器等)。
  - 3G/4G/NB-IoT：提供广域网远程通信能力。
  - RFID/NFC：支持本地身份识别、门禁或物品追踪等应用。
- 用户交互接口(User Interfaces)
  - LCD + 触摸屏：实现可视化图形界面与人机交互。
  - Audio + Audio Amplifier + Microphone + Earphone：构成音频输入输出系统，支持语音识别与播报。
  - Camera：用于视觉识别、人脸检测等。
  - HDMI：可外接大屏幕显示。
  - Keys：支持按键操作输入。
- 电源管理与扩展接口(PMIC, Power, Extension)
  - PMIC(电源管理 IC)：用于稳定供电、充电管理与电池保护。
  - Power & Battery：支持外部直流供电与电池供电两种方式，满足移动或断电场景需求。
  - Extension：保留扩展接口，用于功能增强或定制化开发。

该边缘终端具备多协议兼容、多模态感知、本地 AI 推理与人机交互能力，可广泛应用于智能家居、工业自动化、智慧医疗等 AIoT 场景。它是实现“感知－推理－执行”智能体闭环的关键硬件基础。

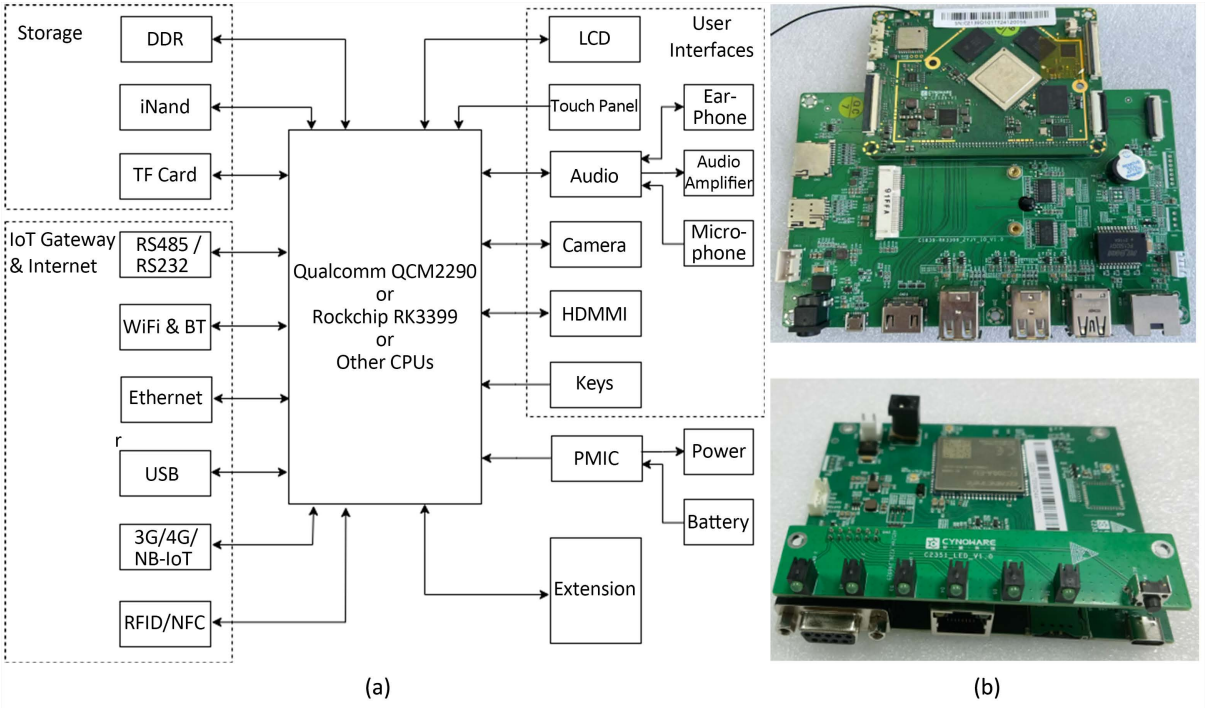

Figure 5. AIoT edge device hardware system
图 5. AIoT 边缘终端硬件系统

表 2 是基于 RK3399CPU 的边缘终端硬件规格书。

Table 2. Hardware specifications of the edge device based on RK3399 CPU
表 2. 基于 RK3399CPU 的边缘终端硬件规格书

Items	Product	Cynoware C2139 Edge Device
Main	CPU	RK3399 Dual-core Cortex-A72 + Quad-core Cortex-A53
	RAM	4 GB RAM
	Storage	16 GB
	LCD	10.1" (1920 × 1200) option
	Touch	10.1" multi-touch TP option
	Camera	1080p option
Audio	Speaker	2 W × 2 option
	Mic	×1 option
Connectivity	WIFI	IEEE802.11 b/g/n (2.4G) and BT 4.2
	Ethernet	1 × RJ45 with 10/100/1000 M
IO	USB	Type A USB 3.0 × 1, USB 2.0 × 1, Micro USB 2.0 × 1
	DC In	Input: AC 100~240 V; Output: DC 24 V/3.75 A
OS	OS	Android 10

4.2.2. 边缘终端软件系统架构

边缘终端不仅作为物理交互节点和 IoT 数据汇聚平台，其软件系统也承担着 AI 推理、协议桥接、任务调度与用户交互等关键职责。整体软件架构由操作系统、通信中间件、智能体执行模块、用户接口与

自动化工作流组件构成，主要包括以下部分：

- 操作系统层

边缘终端运行 Android 或嵌入式 Linux 操作系统，提供图形界面支持、硬件抽象层与网络协议栈，支撑设备驱动与 AI 模型的运行环境，图 6 所示的是 AIoT Android 系统。

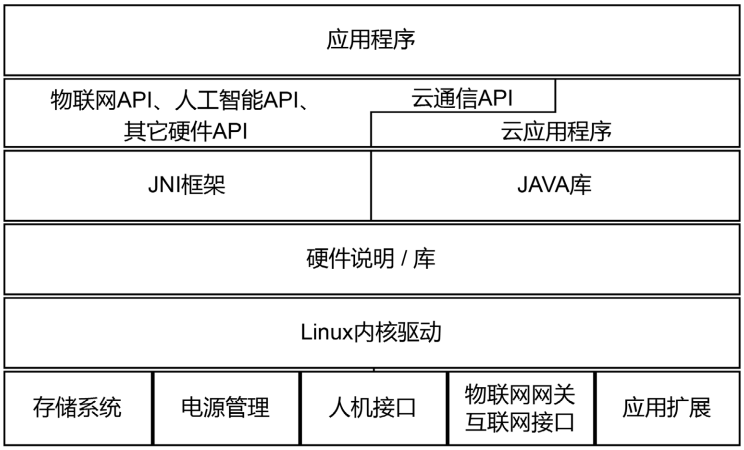

Figure 6. Android operating system of the AIoT edge device
图 6. AIoT 边缘终端的 Android 操作系统

- 通信中间件
  - MQTT 客户端：用于边缘终端与本地服务器(如 ThingsBoard)及云端平台的数据发布与订阅，实现轻量级的物联网通信。
  - OPC UA 服务器&客户端：支持工业标准信息建模与设备抽象，提供传感器/执行器的统一接口管理与上下行通信能力，可适配 Zigbee、RS485、Modbus 等底层协议。
- AI 智能体模块
  - 集成本地 AI 模型(如 AlexNet、MobileNet 或自定义轻量化卷积神经网络)，用于场景感知、行为识别、环境预测控制等任务。
  - 具备初级类人对话能力与自主任务规划能力，支持实时数据分析与事件驱动行为决策。
  - 当本地 AI 算力无法满足需求时，可调用远程 LLM (如 DeepSeek 等)完成复杂推理或任务分解，体现出智能体的任务迁移与资源感知能力。
- 用户交互界面
  - 提供基于语音识别与语音合成的自然语言交互入口，用户可通过口头指令控制设备或查询状态。
  - 集成触摸屏 GUI 与可视化仪表盘，展示环境数据(如温湿度、光照、空调状态等)，并提供手动控制选项。
- 自动化与工作流引擎(n8n)
  - 内嵌或集成开源自动化平台 n8n，用于构建事件驱动型业务流程，实现任务的条件判断、工具链调用与动作序列自动执行。
  - 支持自定义插件与 Webhook 接口，可与 AI 智能体模块协同触发复杂自动化响应。
- 任务调度与边缘－本地服务器协同机制
  - 边缘终端内置任务调度器，根据任务类型与资源占用情况，判断是否本地执行或委托本地服务器/云端推理处理。

- 实现“感知–判断–执行”闭环智能体循环的同时，具备跨层协同能力，适应多任务并发与动态场景变化。

4.3. 云平台模块：ThingsBoard [4]云端服务器

在整个系统中，云平台承担着数据汇聚、远程控制、可视化展示与智能分析等关键职能。本系统基于开源物联网平台 ThingsBoard 搭建云端服务器，结合 Dash 平台实现高级数据分析和多维度监控。

- 通信机制
  - 通过 MQTT 协议与边缘终端实现双向通信，支持设备数据上报与命令下发，确保系统低延迟、轻量化、稳定的网络交互。
- 核心功能(基于 ThingsBoard)
  - 设备管理与配置：支持批量设备注册、分组、远程参数配置与生命周期管理。
  - 实时数据监控：采集温度、湿度、光照、人体感应等 IoT 数据，支持图表化、仪表盘展示。
  - 数据存储与历史记录：可自定义数据保留周期与存储格式，支持 SQL/NoSQL 数据库集成。
  - 规则链引擎与自动化控制：内置流式处理机制，支持条件判断、联动控制、告警触发等自动化逻辑。
  - 远程控制与命令执行：用户可通过 Web 端或移动端下发开关、调节等控制指令至边缘设备。
  - 权限与安全管理：内置角色分级、Token 认证、访问日志记录，确保数据隐私与设备安全。
  - 性能分析与可扩展性：支持多租户部署，便于在工业、智能家居、智慧城市等场景下横向拓展。
- Dash 高级可视化平台
  - 通过集成 Dash (Plotly)或 ThingsBoard 内置高级可视化插件，展示 AI 推理结果、设备状态、历史趋势与预测数据，支持用户自定义仪表盘与动态图表组件。
- 远程访问与用户交互
  - 用户可通过 ThingsBoard Web 仪表盘或移动端应用，进行远程设备管理、参数查询、告警响应与命令交互，提升系统整体的可操作性与可维护性。

4.4. 本地与云端 LLM 协同推理模块

在面向 AGI 的 AIoT 系统中，大语言模型(LLM)作为核心的认知与推理引擎，为 AI 智能体提供语义理解、任务规划、自然交互等高阶能力。考虑到不同应用场景对延迟、算力和网络的要求差异，本系统设计支持本地 LLM 与云端 LLM 的协同部署机制，由 MCP 动态调度调用。

- 本地 LLM (轻量化 LLM 如 QwQ)
  - 部署位置：位于边缘设备或局域网内服务器。
  - 模型类型：通常为小参数量模型(如 TinyGPT、MiniLM、DistilBERT 等)，支持基础问答、指令解析与简易对话。
  - 优势：
    - ✓ 响应速度快，推理延迟低(毫秒级)；
    - ✓ 数据本地处理，保障隐私；
    - ✓ 可在网络不可用或不稳定时持续运行；
    - ✓ 适用场景：设备控制、单轮指令理解、环境反馈解析等对实时性要求高的任务。
- 云端 LLM (如 DeepSeek 等)
  - 部署方式：通过 API 接口远程访问云计算平台上的大型预训练语言模型。
  - 优势：
    - ✓ 模型规模大，具备更强的语言理解、推理和多轮对话能力；

- ✓ 支持知识检索、复杂逻辑推理、多任务规划等高级功能；
- ➢ 劣势：
  - ◆ 网络依赖性高，存在一定时延；
  - ◆ 数据需传输至云端，可能存在隐私风险；
  - ◆ 适用场景：复杂场景感知、跨模态分析、长期任务管理、人机自然语言交互等。

- MCP 调度策略
  - ➢ AI 智能体内嵌 MCP (多组件规划器)，可根据任务特征动态选择推理引擎，策略包括：
    - ■ 任务复杂度：简单任务由本地 LLM 处理，复杂推理任务委托云端 LLM；
    - ■ 实时性要求：对延迟敏感的任务优先采用本地推理；
    - ■ 网络状态感知：在弱网或离线情况下自动回落至本地模型；
    - ■ 模型可用性：基于设备资源(内存、CPU 占用等)判断本地模型是否可运行。

本文充分结合边缘计算与云计算优势，实现高性能－低延迟－高智能的 AIoT 认知系统，为 AI Agent 在 AGI 时代的泛化部署奠定基础。

5. MCP、AI 智能体和 n8n

在面向 AGI (通用人工智能)时代的 AIoT 系统中，实现系统级智能不仅依赖于单点设备的局部推理能力，更需要一种能够协调多源感知、模型资源、执行工具的智能体协作机制。为此，AI 智能体和 MCP (Multi-Component Planner)作为 AIoT 智能体架构中的两个核心角色，分别承担“智能终端交互”与“系统任务规划与调度”的职责。AI Agent 聚焦于环境感知与用户交互，具备初步的认知能力与本地决策功能；而 MCP 则作为智能中枢，负责对高层语义任务进行结构化分解、模型选择与跨模块执行调度。

5.1. 定义

1) AI Agent [5] (人工智能智能体)

AIoT 系统中的 AI Agent 是部署在边缘设备或云端的智能软件体，具备环境感知、语义理解、自主决策与动作执行能力，是 IoT 系统中“智能行为的实施者”。它通常集成轻量级模型或本地推理引擎，负责接收来自用户或环境的输入信息，并根据内置策略或外部规划结果完成特定任务。AI Agent 表现出一定程度的类人智能，能够持续与用户交互、自我适应，并保持上下文理解。

2) MCP [6] [7] (Multi-Component Planner，多组件任务规划器)

MCP 是部署在边缘终端或局域网服务器上的任务调度与智能规划核心模块，主要负责将 AI Agent 所提出的高层语义任务转化为可执行的多步骤任务流程，并动态调用工具、模型或服务来完成具体动作。MCP 实现了“感知－规划－推理－执行”智能体循环机制中的“推理－规划”中心，是整个系统的智能控制中枢。

3) n8n [8]

n8n (Node-Node)是一个开源、低代码的工作流自动化平台，允许用户通过图形化界面将不同的应用服务、API 接口与逻辑操作串联起来，构建自动化任务流程。n8n 支持超过 300 种第三方集成，具备高度可扩展性与自托管能力，广泛应用于数据同步、自动控制、智能通知、系统协调等场景。

AIoT 中的 MCP 体系通过将任务逻辑抽象成可规划的组件，解耦了智能体的感知与执行能力，以及任务推理和资源协调能力，实现了“轻智能终端 + 重调度中心”的系统模式。MCP 服务器与客户端之间的高效协同，是构建支持 AIoT 智能体系统的基础。而在 AIoT 系统中，n8n 被用作任务执行与工具编排层，配合 AI Agent 与 MCP 结构，完成传感器数据处理、控制指令下发、模型调用与结果反馈等功能，成为智能体系统中“执行”的核心组件之一。

5.2. AIoT MCP

之所以称为 AIoT MCP 是因为本文 MCP 用于 AIoT 系统中。本文 AIoT MCP 的架构设计中采用“服务器–客户端(Server-Client)”结构，主要是为了实现分布式协同、模块解耦、可扩展性强以及异构设备之间的统一通信与管理。

MCP 系统采用集中式调度 + 分布式协作的体系结构，由部署在边缘终端或局域网本地服务器的 MCP 服务器负责全局任务协调与资源分配，部署在各边缘终端的 MCP 客户端实现感知上报、任务请求与结果接收，本文 AIoT MCP 的架构如图 7 所示。

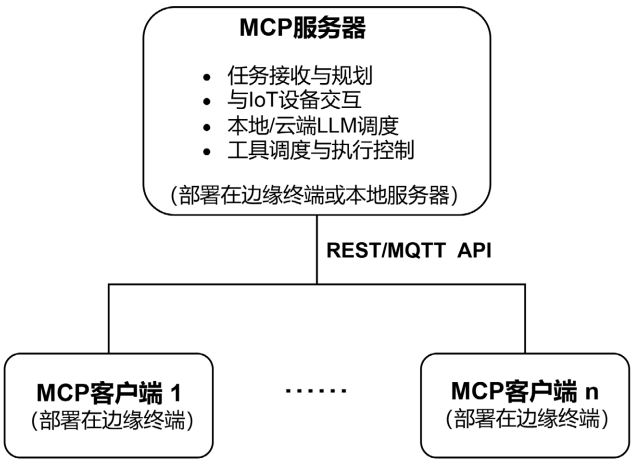

Figure 7. AIoT MCP architecture
图 7. AIoT MCP 架构

1) AIoT MCP 服务器(MCP Server)

- 核心功能
  - 任务接收与解析：接受 AI Agent 或应用层提交的自然语言或结构化任务请求。
  - 意图识别与任务规划：通过内置解析器与规划器(支持 HTN/PDDL/LLM 生成计划)将任务拆解为步骤序列。
  - 模型选择与调用：根据任务复杂度、网络状态及资源占用动态选择本地 LLM 或云端 LLM 进行推理。
  - 工具调度与控制：通过注册表统一管理所有工具模块，并通过 OPC UA、MQTT、REST 等协议执行任务。
  - 日志与状态反馈：记录任务执行过程与结果，支持返回至客户端或上传至云端平台(如 ThingsBoard)。
- 技术架构
  - 构建语言：FastAPI/Flask
  - 调度策略：基于规则 + LLM 增强计划(如 Plan-GPT)
  - 模型对接：本地 Llama.cpp/云端 DeepSeek/OpenAI API
  - 工具接口：JSON-RPC/OPC UA SDK/MQTT broker
  - 数据缓存：Redis + SQLite

2) AIoT MCP 客户端(MCP Client)

- 部署位置
  - 部署在**每个边缘终端设备(如智能屏、嵌入式网关)**上，与 AI Agent 运行在同一节点。
- 核心功能
  - 感知信息上传：从传感器采集数据(如温湿度、占用状态)并打包上传至 MCP 服务器；
  - 任务请求发起：当 AI Agent 识别到用户意图或环境事件后，构造任务请求并发送给 MCP 服务器；
  - 响应接收与执行：接收 MCP 返回的执行计划或最终动作命令，并调用本地工具或控制 API 执行；
  - 执行反馈回传：将执行状态或结果回传给 MCP 服务器，或通过 MQTT 同步至云平台。
- 示例通信流程

3) 协同机制与部署策略

MCP 服务器和客户端的协调机制和部署策略如表 3 所示。

Table 3. Coordination mechanism and deployment strategy of the MCP server and client
表 3. MCP 服务器和客户端的协调机制和部署策略

项目	MCP 服务器	MCP 客户端
推荐部署位置	边缘服务器或本地主控节点	每个 AI Agent 边缘终端
可服务对象	多个 MCP 客户端/Agent	对接单一 MCP 服务器
通信协议	REST API, MQTT, WebSocket	REST API/MQTT
安全机制	Token 认证 + IP 白名单	内置 Agent 身份标识
容器化建议	支持 Docker Compose 部署	作为边缘服务模块启动

5.3. AIoT AI 智能体

1) AIoT AI 智能体定义

之所以称为 AIoT AI 智能体是因为本文 AI 智能体是部署于 AIoT 系统边缘终端中的自治智能体模块，具备以下核心能力：

- 环境感知：从传感器收集温湿度、光照、人体存在等多模态数据；
- 语义理解：识别用户语音或文本输入的意图；
- 本地推理：基于轻量模型(如 TinyML、MiniLM、DistilBERT)进行本地判断；
- 用户交互：通过语音、图形界面或触摸交互与用户对话；
- 行动触发：根据理解结果或来自 MCP 的任务响应，执行动作控制或信息反馈。

AI Agent 是系统中任务的感知入口与行为执行者，它既能独立执行简单任务，也能将复杂任务委托给 MCP 系统。

2) AI 智能体功能结构

AI 智能体(AI Agent)功能结构如表 4 所示。

Table 4. Functional structure of the AI agent
表 4. AI 智能体功能结构

AI 智能体(边缘端)
• 感知模块(传感器数据采集)
• 语义识别模块
• 本地推理模块(轻量模型)
• UI 交互模块(触摸/语音界面)
• MCP 客户端接口(请求和回传)

3) AI Agent 与 MCP 的协同关系

AI Agent 并不直接承担复杂的任务规划与多模块调度，而是与 MCP 系统协同工作，形成以下表 5 所示的分工。

Table 5. The collaborative relationship between AI Agent and MCP
表 5. AI Agent 与 MCP 的协同关系

功能阶段	AI 智能体	MCP 客户端	MCP 服务器
感知与输入	采集环境数据、接收用户请求	封装任务请求并上传	接收请求、解析任务、生成计划
意图分析	初步识别指令或语义内容		高阶意图识别/任务分解
任务请求	调用本地 MCP 客户端	向 MCP 服务器发起任务请求	根据上下文选择工具/模型资源
推理/规划	若为简单任务则本地完成	负责传输任务结果或指令	复杂推理由本地/云端 LLM 完成
执行与反馈	控制设备或提示用户	上报执行状态	日志记录与反馈回传至智能体

AI Agent 是边缘侧的智能“执行终端”，MCP 是系统的智能“调度中枢”。通过将任务规划与推理能力上移至 MCP，Agent 设计得以更轻量化、响应更实时；通过将感知与执行下沉至 Agent，系统整体智能具备更强的实时性、鲁棒性与可扩展性。两者的分工与协作，共同构建了适应 AGI 时代的分布式 AIoT 智能体系统。图 8 所示的是 AI 智能体与 MCP 的协同机制。

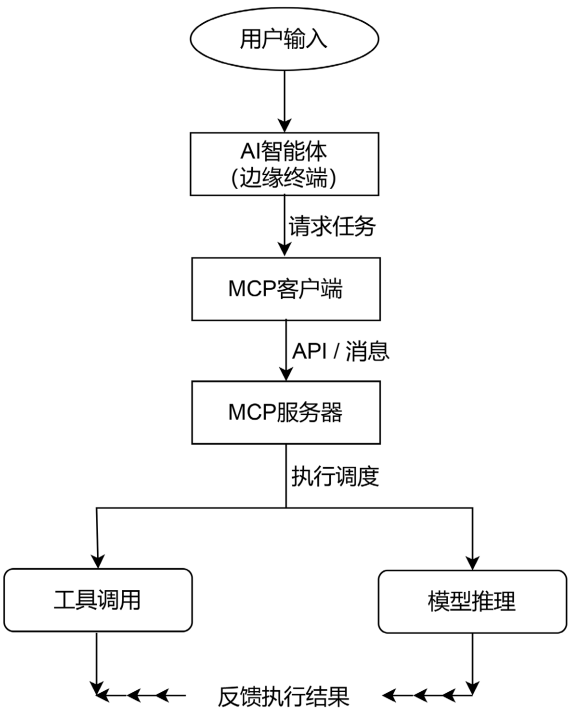

Figure 8. Collaboration mechanism between the AI agent and MCP
图 8. AI 智能体与 MCP 的协同机制

5.4. n8n 的主要功能

1) n8n 是一款开源的低代码自动化工作流引擎，其主要功能如表 6 所示。

Table 6. Main functions of n8n
表 6. n8n 主要功能

功能模块	功能描述
可视化任务流程设计	通过图形界面拖拽节点，构建自动化流程，无需编码
多协议接口集成	原生支持 HTTP、MQTT、WebSocket、数据库、文件系统等接口
Webhook 监听与触发	可作为外部系统任务入口点，接收来自 AI Agent 或 MCP 的请求
逻辑判断与条件控制	支持 if/switch/merge 等分支逻辑，完成任务路径动态选择
定时与事件驱动执行	可根据时间、事件或传感器状态自动执行任务流
插件与扩展机制	支持自定义 Function 节点(JS 脚本)，扩展逻辑处理能力

2) n8n 的作用

在本 AIoT 系统架构中，n8n 的核心作用可概括为以下几点：

- 任务执行器：根据 MCP Server 下发的计划任务，逐步执行各类工具控制、环境数据获取与状态反馈；
- 工具调用调度层：封装调用设备控制 API、模型推理服务、外部天气服务等操作，使任务执行与 AI Agent/MCP 解耦；
- 条件驱动的自动化引擎：根据实时环境或外部信息动态决定执行路径，如：白天拉开窗帘，夜间调暗灯光；
- 任务反馈通道：将任务执行结果上传至云平台(如 ThingsBoard)或返回给 AI Agent，完成“行动–感知–反馈”闭环。

3) 与 AI Agent 和 MCP 的协同关系

n8n 与 AI Agent 和 MCP 构成了 AIoT 系统中的感知–推理–执行三层协同结构，其具体关系如表 7 所示。

Table 7. The collaborative relationship between n8n, AI Agent, and MCP
表 7. n8n 与 AI Agent 和 MCP 的协同关系

层级	模块	作用
感知/交互层	AI Agent	接收用户输入(语音/图形界面)、获取环境数据、提出任务意图
推理/调度层	MCP Server	分析任务目标，调用本地/云端模型进行推理与任务规划，生成执行计划
执行/反馈层	n8n	根据计划调用设备控制工具与 API，实现动作执行与状态同步

6. AI 智能体与 MCP 驱动的个性化客房舒适控制：智慧酒店应用实例

6.1. 应用背景

在智慧酒店场景中，不同客人入住不同房型时，对环境舒适度的需求具有较强的个性化。系统需根据用户偏好、客房实时环境数据、酒店地理位置及室外气候进行综合判断，实现如“调整房间舒适度”这类模糊语义指令的智能响应。

6.2. 场景示例

- 酒店地点：杭州(CN-Hangzhou)
- 当前时间：夏季，室外温度 34℃，湿度 78%

- 客房号：Room 102
- 用户：外籍商务客人，偏好干爽、静音环境

6.3. 用户请求

用户通过床头屏/语音助手输入指令：
“请把房间调得舒服一点。”

6.4. 系统模块组成

酒店实例的系统模块组成如表 8 所示。

Table 8. System module composition of the hotel use case
表 8. 酒店实例的系统模块组成

模块	功能说明
AI 智能体	接收用户指令，提取意图，调用 MCP 客户端
MCP 客户端	打包任务，上传至 MCP 服务器
MCP 服务器	任务规划，调用外部气候信息，生成执行计划
n8n	工作流引擎，编排工具调用、条件判断、执行流程自动化
IoT 执行设备	空调、除湿机、窗帘、灯光，通过 MQTT/OPC UA 控制

6.5. n8n 实现流程(结构图)

酒店实例的 n8n 流程如图 9 所示。

Figure 9. n8n workflow of the hotel use case
图 9. 酒店实例的 n8n 流程

6.6. 实例任务数据传递结构

来自 AI Agent 的请求(Webhook 输入)：

```
{
  "task": "adjust_room_comfort",
  "room_id": "102",
  "location": "CN-Guangzhou",
  "user_pref": {
    "temperature": 25,
    "humidity": "dry",
    "light": "soft"
  }
}
```

6.7. 外部气候信息查询节点

GET https://api.weatherapi.com/v1/current.json?key=API_KEY&q=CN-Hangzhou

返回示例：

```
{
  "location": { "name": "Hangzhou" },
  "current": {
    "temp_c": 34.2,
    "humidity": 78,
    "is_day": 0
  }
}
```

6.8. 生成执行计划(Function 节点逻辑)

```
const weather = items[0].json.current;
const plan = [];
if (weather.temp_c > 28) {
  plan.push({ device: "ac", action: "set_temperature", value: 25 });
}
if (weather.humidity > 65) {
  plan.push({ device: "dehumidifier", action: "on" });
}
if (!weather.is_day) {
  plan.push({ device: "curtains", action: "close" });
  plan.push({ device: "light", action: "dim", value: 30 });
}
return plan.map(p => ({ json: p }));
```

6.9. 控制指令节点(HTTP/MQTT)

示例：

POST http://iot-gateway.local/device/control

```
{
  "device": "ac",
  "command": "set_temperature",
  "value": 25,
  "room_id": "102"
}
```

6.10. Agent 回传反馈内容节点

```
{
  "message": "房间温度已设为 25℃，除湿设备已开启，灯光已调暗。祝您入住愉快！"
}
```

6.11. 智能特性总结

酒店实例的智能特性如表 9 所示。

Table 9. Intelligent features of the hotel use case
表 9. 酒店实例的智能特性

智能特性	实现机制
个性化调节	用户偏好 + 当前房间数据
场景感知(夜晚)	天气 API 中的 is_day
环境适应	实时气候推理 + 动作规划
多工具协同控制	n8n 流程中串行 + 条件控制
云边协同	本地 Agent 交互 + 云端 MCP 知识调用

该实例展示了在智慧酒店环境中，AI Agent 与 MCP 的协同配合如何通过 n8n 实现面向 AGI 风格的“任务理解–情境推理–自动执行”闭环流程。系统不仅能响应自然语言的模糊请求，还能动态适应城市气候、夜晚时段等条件，体现了 AIoT 智能体系统的高度扩展性与泛化能力。

7. 总结及展望

7.1. 总结

本文围绕面向 AGI (通用人工智能)时代的 IoT 智能体系统，提出并实现了一种开源、可扩展、模块化的 AIoT 系统架构。该架构以 AI Agent 与 MCP (多组件任务规划器)协同机制为核心，融合边缘计算、本地/云端推理、自动化流程引擎(n8n)与 IoT 控制协议(如 MQTT、OPC UA)等关键技术，具备以下显著优势：

- 感知–推理–执行闭环：实现从用户意图输入到物理动作输出的完整智能体工作流；
- 任务解耦与模块协同：AI Agent 负责交互与触发，MCP 负责任务规划与工具调度，二者职责明确、解耦部署；

- 多模态融合与动态推理：系统可集成环境感知、位置与气候数据，动态选择本地/云端模型，提升系统智能响应能力；
- 开源与低耦合：各模块接口统一，基于开源技术构建，便于复用与工程扩展；
- 应用落地性强：通过智慧家庭与智慧酒店实例，验证架构在真实场景下的实用性与可部署性。

通过上述研究与实践，本文验证了构建面向 AGI 能力的 AIoT 系统在当前技术条件下的可行路径，为实现具备泛化理解与复杂任务自主执行能力的嵌入式人工智能系统奠定了基础。

7.2. 展望

尽管本系统已在架构与功能上初步实现 AGI 智能体的核心特征，但仍存在值得进一步研究与优化的方向：

- 本地 LLM 模型的轻量化与定制化。面向边缘设备 LLM (如 TinyGPT、Phi-2)的部署仍面临资源瓶颈，后续需探索蒸馏、量化、微调等模型优化技术以提升推理性能与本地自主能力。
- 任务规划器智能化升级。当前任务规划基于规则与轻量 LLM 混合方式，未来可引入更强的推理规划模型(如 Plan-GPT、AutoGPT)，增强系统在多轮任务理解与未知任务应对上的泛化能力。
- 多 Agent 协同与分布式智能体调度。当多个 AI Agent 部署在同一建筑或系统中，如何实现分布式 MCP 协同与任务抢占、冲突解决将成为重要的研究方向。
- 安全与隐私保护机制。在多 Agent 与云平台协同过程中，需进一步加强身份认证、数据加密与边缘隐私计算，确保系统运行的可靠性与合规性。
- 行业级部署与开源推广。后续将计划以 Docker + n8n + LLM 为基础封装系统的核心能力，发布开源框架、服务智慧园区、养老照护、智能制造等 AIoT 垂直领域。

参考文献

[1] 吴薇, 吕亚欣. 服务国外市场的 AIoT 开源方案及其应用[J]. 单片机与嵌入式系统应用, 2022, 22(9): 4-9.

[2] Santos, M.G.D., Ameyed, D., Petrillo, F., Jaafar, F. and Cheriet, M. (2020) Internet of Things Architectures: A Comparative Study. arXiv:2004.12936.

[3] 琚子晗, 白贺, 杨喜童. 基于 Freertos 与 ARM 的智能探索机器人系统设计与实现[J]. 机械工程师, 2021(6): 37-39+42.

[4] ThingsBoard Inc. (2023) Things Board Open-Source IoT Platform Documentation. https://thingsboard.io/docs/

[5] Masterman, T., Besen, S., Sawtell, M. and Chao, A. (2024) The Landscape of Emerging AI Agent Architectures for Reasoning, Planning, and Tool Calling: A Survey. arXiv:2404.11584.

[6] Krishnan, N. (2025) Advancing Multi-Agent Systems Through Model Context Protocol: Architecture, Implementation, and Applications. arXiv:2504.21030.

[7] Hou, X., Zhao, Y., Wang, S. and Wang, H. (2025) Model Context Protocol (MCP): Landscape, Security Threats, and Future Research Directions. arXiv:2503.23278.

[8] n8n GmbH (2024) n8n Workflow Automation Documentation. https://docs.n8n.io

Embedded Technology and Intelligent Systems 嵌入式技术与智能系统, 2025, 2(2), 115-122
Published Online April 2025 in Hans. https://www.hanspub.org/journal/etis
https://doi.org/10.12677/etis.2025.22009

Hans汉斯

基于FPGA和OV6946的微型3D内窥镜的实现

高　健

超威半导体，上海

收稿日期：2025年4月1日；录用日期：2025年4月23日；发布日期：2025年4月30日

摘　要

内窥镜泛指经自然腔道或人工孔道进入体内，并对体内器官或结构进行直接观察和对疾病进行诊断的医疗设备，一般由光学镜头、冷光源、光导纤维、图像传感器以及机械装置等构成。文章介绍了一款基于两片图像传感器和FPGA组成的微型3D内窥镜方案，其可以三维成像，提供更好的空间显示效果，已广泛应用于外科微创手术中。

关键词

微创，3D内窥镜，OV6946，FPGA

Implementation of a 3D Miniature Endoscope Based on FPGA and OV6946

Jian Gao

AMD, Shanghai

Received: Apr. 1st, 2025; accepted: Apr. 23rd, 2025; published: Apr. 30th, 2025

Abstract

Endoscopes generally refer to medical devices that enter the body through natural cavities or artificial channels to directly observe internal organs or structures and diagnose diseases. They are usually composed of optical lenses, cold light sources, optical fibers, image sensors, and mechanical devices. This article introduces a miniature 3D endoscope solution based on two image sensors and FPGA. It can achieve three-dimensional imaging, thus providing better spatial display effects, and is widely applicable to minimally invasive surgical procedures.

文章引用: 高健. 基于 FPGA 和 OV6946 的微型 3D 内窥镜的实现[J]. 嵌入式技术与智能系统, 2025, 2(2): 115-122.
DOI: 10.12677/etis.2025.22009

Keywords

Minimally Invasive Surgery, 3D Endoscope, OV6946, FPGA

1. 引言

微型内窥镜一般工作在操作空间较为狭小的受限环境中，广泛应用于医疗、航天、半导体和安防等领域。在一些比如微创手术或者神经外科手术等场合，往往需要更为精确的立体视觉定位，而常见的内窥镜一般采用单个镜头，只能做平面成像，难以提供准确的三维信息。

本方案基于两片微型图像传感器和 FPGA 组成双目视觉方案，前端器械端通过柔性材料连接，具有体积小、使用灵活、成像质量高，以及可以手持等特点。

2. 系统总体设计方案

2.1. 系统介绍

FPGA 是 Field Programmable Gate Array 的缩写，全称是现场可编程门阵列，是一种高度可配置的集成电路，允许用户在制造后通过硬件描述语言对其进行编程，以实现特定功能。FPGA 具有并行处理能力强、高吞吐量和低延迟的优势，广泛应用于信号处理、图像处理、工业控制等领域。

光学镜头采用 OV6946 芯片，此芯片是 OmniVision 公司面向医疗市场推出的小型成像元件[1]，大小仅有 0.9 × 0.9 mm，而图像分辨率可以达到 400 × 400@30fps，并且片上集成 ISP 单元，可以实现 BLC、AEC/AGC 和 MWB 等图像处理算法。即使加上集成 LED 照明，在封装后可以做到大小 1.2 mm 左右的模组，这使得其很适合做内窥镜方向的应用。由于 OV6946 本身输出的图像是模拟信号，一般的使用需要配合 OV426 芯片桥接[2]，才能输出 DVP 数字信号。

本方案集成两片 OV6946 芯片，其输出的模拟图像信号通过 OV426 转接，并通过 DVP 接口连接至 FPGA，FPGA 与电脑通过 EZUSB FX3 模块连接[3]，以传输实时图像。

2.2. 技术特点

1) 体积小，适用于对空间要求极高的场合。

2) 双目组成的三维成像视觉方案，可以提供更好的立体视觉效果。

3) 利用 FPGA 的逻辑触发对两个图像传感器同步，可以实现两个 Pixel 以内的帧同步。这在立体成像同步上有一定的优势。

4) 无需复杂的双目视频算法，无需对镜头校准，使用 3D 头盔左右模式即可显示立体图像。

5) 两种硬件的形态，既支持连接至 PC，通过 PC 外接头盔显示的传统方案，也支持直接输出至 3D 头盔的嵌入式方式。

3. 硬件实现方案

3.1. 机械结构

硬件上，本方案采用两组 OV6946 + OV426 芯片，两片 OV6946 加上 LED 照明组成前端的器件端，

后端电路板集成 FPGA 和两片 OV426 芯片，前后端使用 FPC 软排线连接，可以保证器械的灵活性。PCB 上使用 FPGA 采集原始 BAYER 格式的图像，并在内部集成 ISP 处理单元，处理后的两幅图像合成一幅左右模式的图，通过 HDMI 输出至显示器，配合 3D 头盔，可以实现 3D 成像的方案。并且，PCB 上集成了 EZUSB FX3 模块，输出标准的 UVC 格式图像，可以连接到 PC 端，使用常见的软件比如 VLC 实时显示 3D 图像。

器械前端采用双 OV6946 的结构，两个模组并排地连接固定，在周围集成 LED 照明单元。两个模组输出的信号和 LED 照明电源同时经过 FPC 软排线连接到后面电路板。FPC 的长度可以定制，能支持大约两米的长度，供常用的医学手术使用。

3.2. 电路结构

前端的两组镜头通过 FPC 软排线连接到后面的电路板。电路板上集成了 FPGA，除此之外，还集成了两片 DDR3 X16 颗粒，组成 32 位 DRAM。一颗 HDMI 芯片提供可选的 HDMI 输出，可以外接显示器。一颗 EZUSB FX3 芯片，可以连接 USB 3.0 到 PC。

整体电路结构图如图 1 所示：

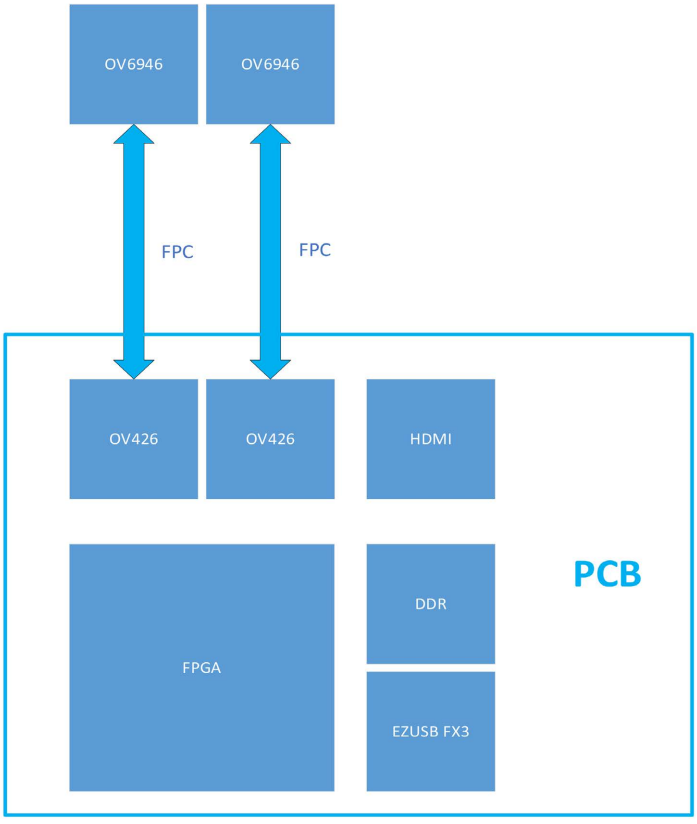

Figure 1. Overall hardware architecture
图 1. 整体硬件结构图

3.3. FPGA 选型与简介

FPGA 采用 AMD 公司的产品 XC7Z020CLG400-2，这是 ZYNQ-7000 系列的可编程片上系统(SoC)，它集成了 FPGA 和双核 ARM Cortex-A9 处理器，适用于嵌入式计算、信号处理、工业自动化等高性能应

用。其主要特点有：

1) FPGA + ARM 处理器：片上集成了 FPGA 逻辑和 ARM Cortex-A9，适用于嵌入式计算和硬件加速。

2) 高性能 DSP 计算：内置 220 个 DSP 计算单元，适用于信号处理、图像处理和 AI 加速。

3) 灵活的 I/O 设计：支持多种接口(USB、PCIe、UART、SPI、I2C、CAN、SDIO)。

4) 丰富存储选项：NOR/NAND Flash、SD/eMMC 存储。

5) 低功耗设计：适用于嵌入式和电池供电系统。

3.4. 3D 头盔显示模块

3D 头盔显示主要通过一系列硬件和显示技术来呈现沉浸式的三维图像和交互体验。其核心原理是通过双眼视差三维成像，具体来说，3D 头盔通常会配备两个显示屏(一个为每只眼睛提供图像)或者采用单一显示屏，通过光学分离技术(如镜头)让每只眼睛看到不同的图像。这种方法模仿了人类眼睛的视差效应：每只眼睛看到稍微不同的视角，通过大脑合成产生深度感知。

同时，当左右眼看到的图像稍有不同(基于摄像机的立体图像)，大脑会将这些图像结合起来，产生三维的深度感知，从而获得身临其境的视觉体验。

在选型上，3D 显示头盔只需要选用常见的可以通过 HDMI 从 PC 串联的设备即可。PC 将显示的图像实时串流至头盔，同时还可以将平面的图像输出至外接的显示器，供人员查看。

将 3D 头盔打开左右的 3D 模式后，人眼靠近头盔，即可观察到三维图像。

4. FPGA 实现方案

4.1. FPGA 整体实现框架

FPGA 系统框架如图 2 所示。所有模块都提供 AXI4-Lite Slave 接口，挂载到 FPGA 的 AXI 总线上，以便可以通过 ZYNQ-7000 芯片的 ARM 处理器在线读写内部寄存器，从而修改各模块配置。

整体的 FPGA 设计模块图如图 2 所示：

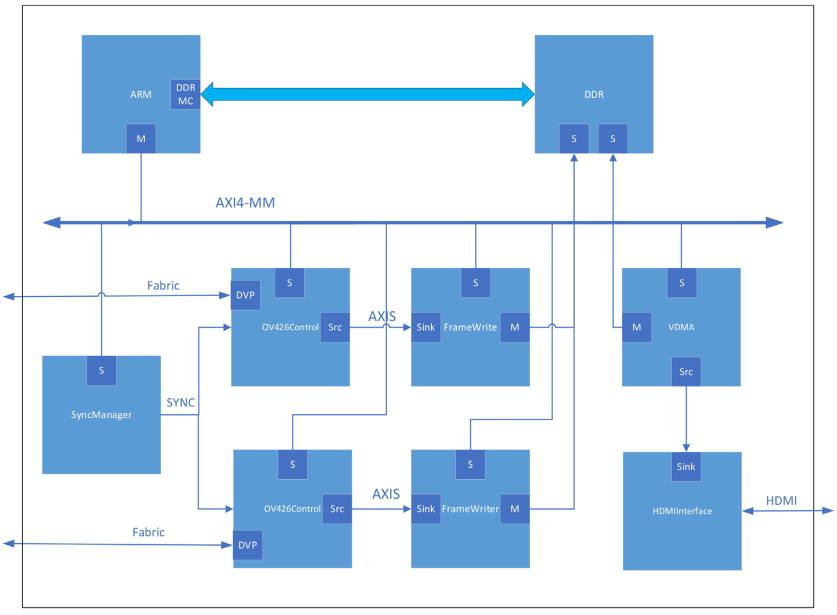

Figure 2. FPGA block diagram
图 2. FPGA 结构图

4.2. OV426 控制模块

FPGA 内部实现两个 OV426 模块，通过 10 位 DVP 接口分别与两个 OV426 模块连接，传输实时图像。模块内 SCCB 模块，用于初始化和在线修改 CMOS 的配置，比如白平衡、开窗大小等参数。系统异常通过中断引脚 INT_o 向 ARM 汇报。

需要说明的是，传统的相机同步方案一般是基于单片机和 ASIC，为了实现帧同步，在上层软件处理的“软”同步方法，此方案缺点是无法保证原始图像的同步。而本方案基于 FPGA，在两个 OV426 模块的外部分别增加了 SYNC_i 同步的逻辑，由外部统一出发，这可以在硬件的层面达到了原始图像的“硬”同步。实验结果表明，本方案可以达到 2 个 Pixel 以内的帧同步。

OV426 模块设计框图如图 3 所示：

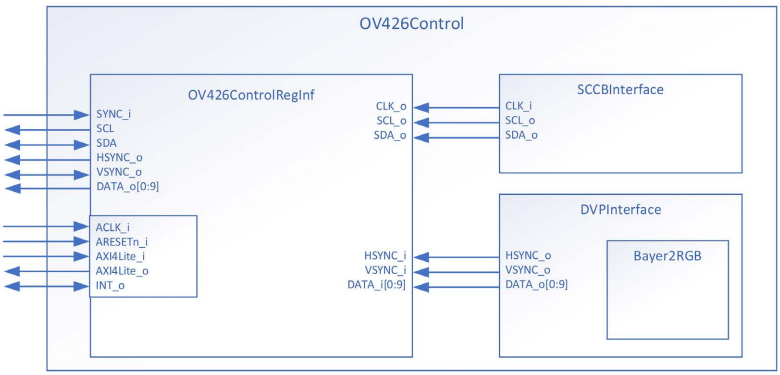

Figure 3. OV426 block diagram
图 3. OV426 模块结构图

4.3. EZUSB FX3 控制模块

上位机与 FPGA 的通信采用 EZUSB FX3 模块，其为英飞凌公司推出的灵活的 USB 控制器，提供 5 Gbps 的理论传输速率，实测速度大于 260 Mbps。可以传输 1080p@30Hz 的视频。对于两组 400 × 400@30fps 的图像，实际带宽完全没问题。

FPGA 内部实现 EZUSB FX3 模块，通过 10 位并口与 EZUSB FX3 模块连接，传输实时图像。模块内实现状态机、监测与管理数据传输模式、缓冲区满、数据有效等系统状态。

EZUSB FX3 模块设计框图如图 4 所示：

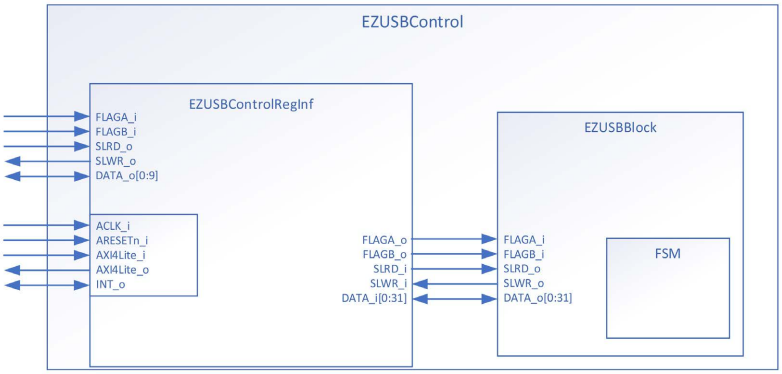

Figure 4. EZUSB block diagram
图 4. EZUSB 模块结构图

5. 软件实现

5.1. EZUSB FX3 固件的实现

EZ-USB FX3 支持多种工作模式，以满足不同的应用需求。这些模式可以通过 FX3 的配置进行灵活切换，适应不同的硬件和数据传输需求。常见的工作模式包括：

1) 主机模式。FX3 作为 USB 主机与外部 USB 设备进行通信。

2) 外设模式。FX3 作为 USB 外设连接到上游 USB 主机。

3) 数据传输模式。支持 USB 3.0 模式，实现高带宽的设备应用，如高清视频采集和流式传输。

4) UVC (USB Video Class)外设模式。FX3 充当 USB 视频设备，通过 USB 接口传输视频流。

5) GPIF (General Programmable Interface)模式。能够与外设进行高带宽、低延迟的数据传输。

6) 同步时钟模式。允许它在与其他外设同步时钟进行数据交换时进行精确的时序控制。

7) 异步传输模式。通过该模式与外设进行数据传输，适用于大多数基于流的传输应用。

在本应用下，通过修改 FX3 的固件，使其工作在数据传输模式下，EZUSB FX3 做 bulk 实时数据采集，单次块传输大小为 16 KB。

5.2. 上位机驱动实现

为了减少图像显示的延时，提高系统的实时性能，图像的采集、处理和显示采用多线程异步的架构，数据通过线程安全的环形队列 ConcurrentQueue 缓冲，采用生产－消费者模式，数据采集线程作为生产者，将采集到的图像放入缓冲队列，并工作在最高优先级，以保证帧的完整性，达到系统最大的可靠性。图像处理线程作为消费者，从缓冲队列读取图像帧，并处理后 dispatch 到 GUI 线程，这样可以保证用户界面的流畅性。

上位机软件与 FPGA 数据流遵循自定义的帧同步协议，保证每帧数据的完整有效性。

上位机软件的架构如图 5 所示：

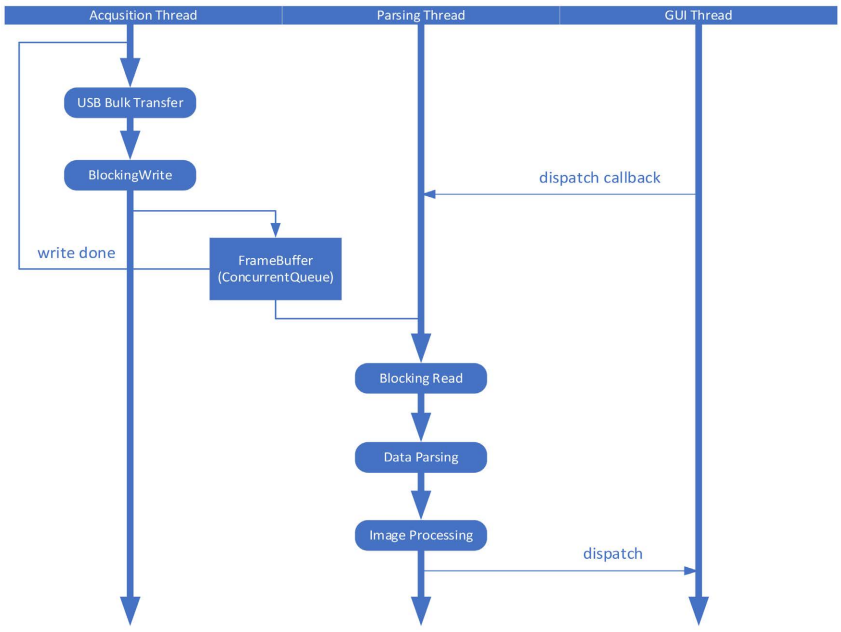

Figure 5. Software architecture
图 5. 软件架构图

5.3. 上位机图形界面

同时，基于上章节中介绍的驱动，在 PC 端开发基于 QT 的 GUI 用户应用程序，用于实时把两个图像传感器的实时图像在经过实时处理之后，分左右两端在显示器上全屏显示，并将原尺寸 400 × 400 的图像被分别压缩成两幅 200 × 400 的图像，并在左右两端显示。

实际的成像效果图如图 6 所示：

Figure 6. Realtime image picture
图 6. 实时成像范图

6. 完成情况

截至投稿前，已完成所有的机械和软、硬件工作，并已应用于实际的模拟实验中。

图 7 展示 3D 内窥镜在模拟远程操作器械端时的实时成像系统。图中用红色部分标注了 3D 内窥镜本体部分和 3D 头盔。

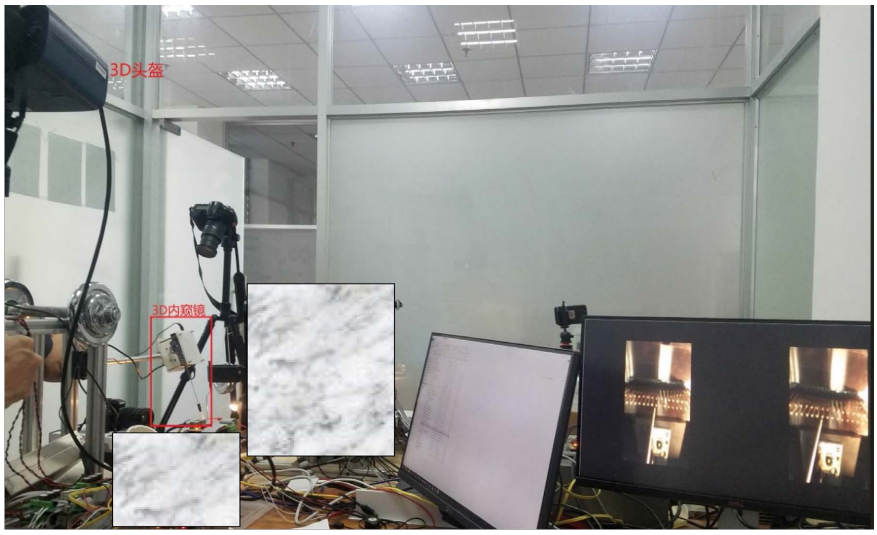

Figure 7. Finished product picture
图 7. 成品图

7. 未来展望

未来主要在以下几个方面考虑微型 3D 内窥镜在 FPGA 上的系统优化[4]-[6]。

7.1. FPGA 集成图像处理

1) 立体匹配：FPGA 可加速基于双目或多目立体视觉的深度计算，提高 3D 成像的实时性。
2) 图像去噪与增强：可在 FPGA 内部集成图像滤波器，以增强对比度，优化内窥图像质量。

7.2. 数据压缩

1) 实时数据压缩：使用 FPGA 加速 JPEG、JPEG2000、H.264/265 等压缩算法，减少数据带宽需求。
2) 高速接口支持：如 MIPI、LVDS 等接口的 CMOS，提高内窥镜系统的数据传输速率。

7.3. AI 与图像分析

目标检测与分割：基于 FPGA 实现神经网络(如 CNN)加速，用于病变检测、组织分类等。

参考文献

[1] OV6946 Product Brief. https://www.ovt.com/products/ov6946/
[2] OV426 Product Brief. https://www.ovt.com/products/ov426/
[3] Infineon Technologies. AN75779——如何使用 EZ-USB FX3 在 USB 视频类别(UVC)框架内实现图像传感器连接[Z].
[4] 陈梦. 小口径工业内窥镜成像系统设计与研究[D]: [硕士学位论文]. 天津: 天津大学, 2019.
[5] 王子浩. 基于 ZYNQ 的超高清内窥镜视频采集与传输系统设计[D]: [硕士学位论文]. 大连: 大连理工大学, 2023.
[6] 张朝财, 张薇, 周德峰, 韦晓孝, 万新军. 基于短基线双目内窥镜成像系统的 3D 视频标定与校正[J]. 光学仪器, 2023, 45(3): 30-36.

Embedded Technology and Intelligent Systems 嵌入式技术与智能系统, 2025, 2(2), 123-132
Published Online April 2025 in Hans. https://www.hanspub.org/journal/etis
https://doi.org/10.12677/etis.2025.22010

基于SCDAYU800A开发板的OpenHarmony移植与适配研究

王　剑[1]，闻　飞[2]，孙庆生[1]

[1]长江大学计算机科学学院，湖北　荆州
[2]江苏润开鸿数字科技有限公司，江苏　南京

收稿日期：2025年4月1日；录用日期：2025年4月23日；发布日期：2025年4月30日

摘　要

文章详细介绍了针对润开鸿鸿锐开发板SCDAYU800A的OpenHarmony操作系统的移植适配过程。润开鸿鸿锐开发板SCDAYU800A基于平头哥高性能RISC-V架构曳影TH1520芯片，具备强大的AI算力和丰富的功能接口。通过对OpenHarmony的适配，实现了从产品定义、工具链适配到内核移植、驱动适配等关键环节的完整流程。并重点探讨了RISC-V架构下针对标准系统Linux内核的移植以及触摸屏驱动的开发，为基于RISC-V架构的嵌入式系统研究提供了实践参考。

关键词

SCDAYU800A开发板，OpenHarmony，RISC-V架构，移植适配，触摸屏驱动

Research on the Porting and Adaptation of OpenHarmony Based on the SCDAYU800A Development Board

Jian Wang[1], Fei Wen[2], Qingsheng Sun[1]

[1]School of Computer Science, Yangtze University, Jingzhou Hubei
[2]Jiangsu Runkaihong Digital Technology Co., Ltd., Nanjing Jiangsu

Received: Apr. 1st, 2025; accepted: Apr. 23rd, 2025; published: Apr. 30th, 2025

Abstract

This paper provides a detailed introduction to the porting and adaptation process of the OpenHarmony operating system for the RunKaiHong Hongrui development board SCDAYU800A. The SCDAYU800

文章引用: 王剑, 闻飞, 孙庆生. 基于 SCDAYU800A 开发板的 OpenHarmony 移植与适配研究[J]. 嵌入式技术与智能系统, 2025, 2(2): 123-132. DOI: 10.12677/etis.2025.22010

development board, based on the Pingtouge high-performance RISC-V architecture Yieying TH1520 chip, features powerful AI computing capabilities and a rich set of functional interfaces. By adapting to OpenHarmony, the entire process from product definition to toolchain adaptation, kernel transplantation, and driver adaptation has been successfully implemented. The paper focuses on the adaptation of the transplantation of the Linux kernel under the RISC-V architecture, and the development of the touchscreen driver, providing practical references for embedded system development based on the RISC-V architecture.

Keywords

SCDAYU800 Development Board, OpenHarmony, RISC-V Architecture, Porting and Adaptation, Touchscreen Driver

1. 引言

随着 RISC-V 指令集架构的快速发展，基于该架构的计算机系统在工业控制、智能设备等领域得到了广泛应用[1]-[3]。润开鸿鸿锐开发板 SCDAYU800A 作为一款高性能的 RISC-V 开发平台，具备强大的 AI 算力和丰富的功能接口，适用于多种应用场景。OpenHarmony 是一个由华为公司贡献给开放原子基金会(OpenAtom Foundation)的开源项目。它是一个全场景分布式操作系统，旨在为各种智能设备提供统一的操作体验。本文将重点探讨如何将 OpenHarmony 4.1 Release 版本移植到润开鸿鸿锐开发板 SCDAYU800A 上，并实现系统的稳定运行。

2. 润开鸿鸿锐开发板 SCDAYU800A 硬件特性

润开鸿鸿锐开发板 SCDAYU800A 具备平头哥高性能 RISC-V 架构曳影 TH1520 芯片，集成 4 核高性能 RISC-V 处理器玄铁 C910 架构。润开鸿鸿锐开发板 SCDAYU800A 的外观如图 1 所示。

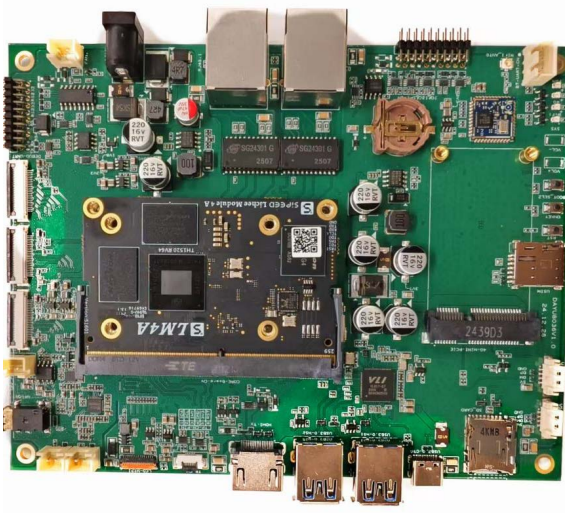

Figure 1. Runkaihong Hongrui development board SCDAYU800A appearance
图 1. 润开鸿鸿锐开发板 SCDAYU800A 的外观

该开发板具备以下硬件特性：

1) AI 算力达 4TOPs，支持蓝牙、音频、视频和摄像头等功能。

2) 支持多种视频输入输出接口，如 MIPI、HDMI 等。

3) 提供丰富的扩展接口，如 GPIO、SPI、I2C 等。

4) 适用于工控平板、智慧大屏、智能 NVR、信息发布系统、云终端、车载中控等多种场景。

图 2 显示了润开鸿鸿锐开发板 SCDAYU800A Mipi 屏幕安装的方法。

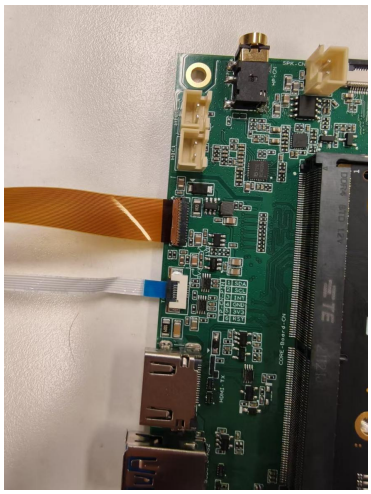

Figure 2. Runkaihong Hongrui development board SCDAYU800A Mipi screen installation
图 2. 润开鸿鸿锐开发板 SCDAYU800A Mipi 屏幕安装

3. OpenHarmony 移植流程

移植就是把程序从一个运行环境转移到另一个运行环境。在主机－开发机的交叉模式下，即是把主机上的程序下载到目标机上运行。图 3 显示了 OpenHarmony 移植到 SCDAYU800 的流程情况。OpenHarmony 版本选择必须大于 3.2 Beta2 以上版本。

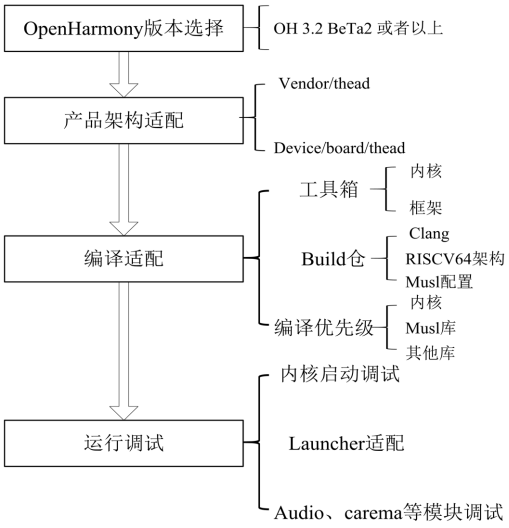

Figure 3. Flowchart of porting OpenHarmony to SCDAYU800
图 3. OpenHarmony 移植到 SCDAYU800 的流程图

OpenHarmony 的移植过程包括产品定义、工具链适配、musl 库适配、内核移植、init 启动子系统移植、显示适配等关键环节[4]。以下是移植步骤。

3.1. 产品定义

在 OpenHarmony 的移植过程中，产品定义是基础步骤。针对 OpenHarmony 标准系统，适配新的产品的方法是首先要在 vendor、device/board 和 device/soc 三个目录下创建产品相关的三个目录 vendor/hihope/${product_name}、device/board/hihope/${product_name}和 device/soc/${chip_product}/${chip}，以 SCDAYU800A 为例，在当前产品中 product_name 设置为 dayu800，chip_product 设置为 thead，chip 设置为 th1520。

vendor/hihope/dayu800 目录主要存放厂家资料以及产品的配置文件，包括描述产品的 config.json、产品的 hcs 配置文件以及其他配置文件等。

device/board/hihope/dayu800 目录主要用于存放开发板相关的文件，包括外设驱动、启动参数、内核编译相关文件、烧录相关文件、uboot 相关文件以及升级相关文件。

device/soc/thead/th1520 目录主要用于存放和芯片 SoC 相关的文件和库，这些文件只会因芯片 soc 改变才会修改，而不会因为开发板变化而进行修改。

快速适配这三个仓的方法是学习能编译通过的产品，例如 rk3568 产品，然后将其拷贝一份，并重命名为当前产品名，然后按照以下步骤根据自己产品的特性进行逐步修改。根据上述分析，首先需要在//vendor/hihope 目录下创建 dayu800 产品目录，并创建 config.json 文件，用于描述产品所使用的 SOC 及所需的子系统。配置文件内容如下[5]：

```
"product_name": "dayu800",
"device_company": "thead",
"device_build_path": "device/board/hihope/dayu800",
"target_cpu": "RISCV64",
"type": "standard",
"version": "3.0",
"board": "dayu800",
"api_version": 11,
"enable_ramdisk": true,
"build_selinux": false,
"build_seccomp": false,
"inherit": [
  "vendor/hihope/dayu800/rich.json",
  "vendor/hihope/dayu800/chipset_common.json"
],
"subsystems": [
  {
    "subsystem": "security",
    "components": [
      {
        "component": "selinux",
```

```
            "features": []
          }
        ]
      }
      // 其他子系统配置
    ]
}
```

该部分根据 vendor 仓的定义，在//device/board/hihope 目录下创建一个 dayu800 的产品，然后创建一个 ohos.build 文件定义产品的 subsystem 以及 BUILD.gn 定义产品的编译项目。

在//device/soc 下根据芯片公司名称创建一个 thead 的仓，存放 th1520 soc 相关的代码和闭源库，通过//device/board/hihope/BUILD.gn 添加模块进行关联编译。该仓下的 ohos.build 文件描述的是设备子系统，这里使用的是 device_th1520，相关名称需要将拷贝过来的文件修改为当前设备的子系统名。同时需要将 ohos.build 的 module_list 下的模块及子模块中涉及的 part_name 和 subsystem_name 全部修改为 device_th1520。

至此，可以通过使用如下命令，启动 dayu800 的构建：

./build.sh --product-name dayu800

3.2. 工具链适配

因为官方的 OpenHarmony 不支持 RISCV64 架构，因此在工具链架构的配置上相对来说配置起来比较困难，首先解决的是 build 仓中的架构适配，这部分仿照 ARM64 配置的部分进行相应的 RISCV64 架构添加，同时对于 musl 的编译配置根据 RISCV64 架构进行对应添加，内核的编译部分根据平头哥提供的编译工具链，使用 llvm 会有相关私有指令集无法编译[6]。

3.3. musl 库适配

musl 库是一个轻量级、高性能的 C 标准库(libc)实现，专为 Linux 系统设计。它旨在提供标准兼容性、代码简洁性和高效性，尤其适合嵌入式系统、容器化环境(如 Docker)或需要高度可移植性的场景。这里首先编译 musl 库是因为该库依赖的模块相对最少，能最快测试 RISCV64 架构编译配置结果是否符合要求，该部分的编译主要涉及到 build 仓的配置、musl 仓的 RISCV64 架构配置，以及 musl 库中有架构的代码，这一块部分从 glibc 中获取。

3.4. 内核移植

内核移植是 OpenHarmony 移植过程中的核心环节。润开鸿鸿锐开发板 SCDAYU800A 采用平头哥提供的 Linux 内核，适配 OpenHarmony 特性。该部分的移植既要对 Linux 内核本身很熟悉，同时也要对鸿蒙的内核特性宏配置很熟悉。本内核移植工作的主要思路是：

第一步是配置内核的编译流程脚本。

内核的编译是 device/board/hihope/dayu800/kernel/BUILD.gn 中 kernel 模块调用对应目录下的 build_kernel.sh 脚本进行编译的。该脚本的核心思想是首先拷贝内核源码到 out/kernel/src_tmp/linux-5.10 路径下，给内核源码中打上 HDF (Hardware Driver Foundation)的补丁，然后在内核源码中创建 vendor 路径的软链接(主要是设备树 dts 和产品的外设驱动)，然后将编译的 config 文件拷贝到 arch/arm64/config 路径下，将产品的 dts 文件拷贝到 vendor/arch/arm64/boot/dts/hihope 路径下，接下来将外设驱动链接到内核

中，编译内核。最后拷贝编译成功的镜像到 out/dayu800/packages/phone/images 中。

第二步是将产品的设备树 dts 完整地移植过来，然后编译内核并解决相应的编译问题。

第三步是将架构代码即 arch/arm64 下的代码尽可能地替换成产品 linux 中的内容，然后编译内核并解决相应的编译问题。

在 device/board/hihope/dayu800/kernel 目录下进行内核编译。除内核以外的代码编译采用的是 OpenHarmony 版本自身的 llvm 工具链。

这里只列出核心编译命令。

```
# 首次编译请先执行下面命令
./build/prebuilts_download.sh
# 全量代码编译
./build.sh --product-name dayu800 --gn-args full_mini_debug=false --ccache
# 单模块编译
# module_name 举例："kernel:kernel"，表示编译 kernel 目录下的 kernel 模块，所有后面的 kernel 是 module_name
./build.sh --product-name dayu800 --ccache --build-target module
# 内核模块编译
./build.sh --product-name dayu800 --ccache --build-target kernel
```

编译成功有如下信息打印

```
[OHOS INFO] c overall build overlap rate: 1.05
[OHOS INFO]
[OHOS INFO]
[OHOS INFO] dayu800 build success
[OHOS INFO] cost time: 0:45:57
=====build  successful=====
2025-01-18 13:06:52
++++++++++++++++++++++++++++++++++++++++
```

将编译后的内核打包成 boot.ext4 镜像文件。

第四步是将产品 linux 的 config文件内的宏移植过来。

第五步是运行内核，并根据运行时问题进行逐个解决。

3.5. init 启动子系统移植

在成功适配 musl 库和内核后，需要适配和启动第一个 init 进程及其初始化配置文件。主要工作包括：

1) 编译适配 startup_init 模块。
2) 配置启动参数，确保系统能够正常启动。

3.6. 显示适配

显示适配是系统启动的最后一步，主要包括以下内容：

1) 在 shell 中运行 bootanimation，确保启动动画正常显示。
2) 调试 OpenHarmony 的 Launcher 模块，完成系统启动。
3) 该模块涉及的仓为 graphic_graphic_2d，代码路径为 foundation/graphic/graphic_2d，该模块是针对

当前产品而适配。

4. 触摸屏驱动开发

触摸屏驱动是润开鸿鸿锐开发板 SCDAYU800A 的重要组成部分，其开发过程基于 HDF 驱动管理框架中的 Input 模型。触摸屏的驱动被放置在//drivers/hdf_core/framework/model/input/driver/touchscreen 目录中。移植触摸屏驱动主要工作是向系统注册 ChipDevice 模型实例[7]。触摸屏驱动的主要工作包括：

1) 对触摸屏驱动 IC 进行上电、配置硬件管脚并初始化其状态。

2) 注册中断、配置通信接口(I2C 或 SPI)。

3) 设定 Input 相关配置、下载及更新固件等操作。

图 4 显示了润开鸿鸿锐开发板 SCDAYU800A 触摸屏的常用管脚。

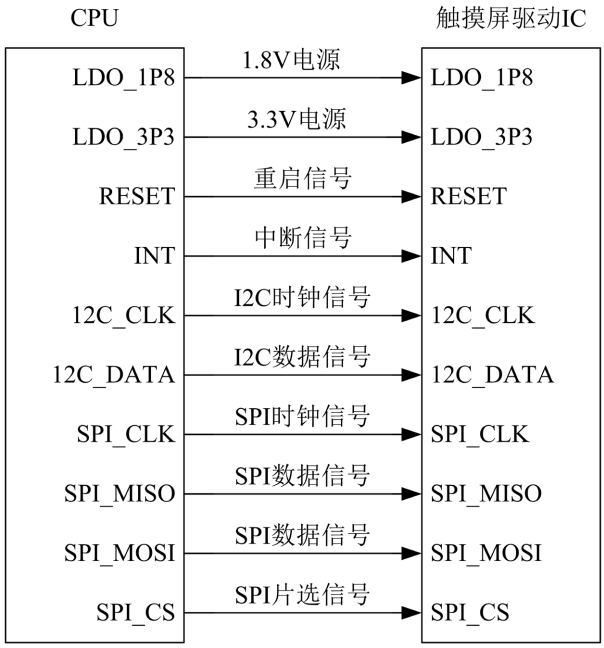

Figure 4. Common pins of the development board SCDAYU800A touch screen
图 4. 开发板 SCDAYU800A 触摸屏的常用管脚

润开鸿鸿锐开发板 SCDAYU800A 触摸屏驱动的开发过程分为以下三个步骤，这里仅列出部分代码。

4.1. 设备描述配置

在 OpenHarmony 中，配置(Configuration)通常指对系统、硬件或应用程序的参数、选项和功能进行定制和调整的过程，以满足特定设备或场景的需求。在 HDF 驱动框架下，需要在配置文件中注册驱动信息。设备描述配置主要包含 Input 驱动模型各模块层级信息配置文件路径为：vendor/hihope/dayu800/hdf_config/khdf/device_info/device_info.hcs，HDF 框架依据该配置信息实现对 Input 模型各模块的依次加载。配置内容如下：

```
input :: host {
    hostName = "input_host";
    priority = 100;
    device_input_manager :: device {
```

```
            device0 :: deviceNode {
                policy = 2;              // 向外发布服务
                priority = 100;       // 加载优先级
                preload = 0;             // 加载该驱动
                permission = 0660;
                moduleName = "HDF_INPUT_MANAGER";
                serviceName = "hdf_input_host";
                deviceMatchAttr = "";
            }
        }
        // 其他设备配置
    }
```

4.2. 板级配置及器件私有配置

板级配置及器件私有配置文件路径为 vendor/hihope/dayu800/hdf_config/khdf/input/input_config.hcs，配置内容如下：

```
    root {
        input_config {
            touchConfig {
                touch0 {
                    boardConfig {
                        match_attr = "touch_device1";
                        inputAttr {
                            inputType = 0;                  // 0 代表触摸屏
                            solutionX = 800;
                            solutionY = 1280;
                            devName = "main_touch";    // 设备名称
                        }
                        // 其他配置
                    }
                }
            }
        }
    }
```

4.3. 添加器件驱动

在器件驱动中，主要实现平台预留的差异化接口。以数据获取及解析为例，润开鸿鸿锐开发板 SCDAYU800A 使用了 gt911 触摸屏，代码路径为：drivers/hdf_core/framework/model/input/driver/touchscreen/touch_gt911.c，部分代码如下：

```
static void ParsePointData(ChipDevice *device, FrameData *frame, uint8_t *buf, uint8_t pointNum)
```

```
{
    int32_t resX = device->driver->boardCfg->attr.resolutionX;
    int32_t resY = device->driver->boardCfg->attr.resolutionY;
    // 数据解析逻辑
//器件私有配置解析
//器件设备注册到平台驱动
//调用 Input HDI 接口
}
```

图 5 显示了调用 Input HDI 的步骤。

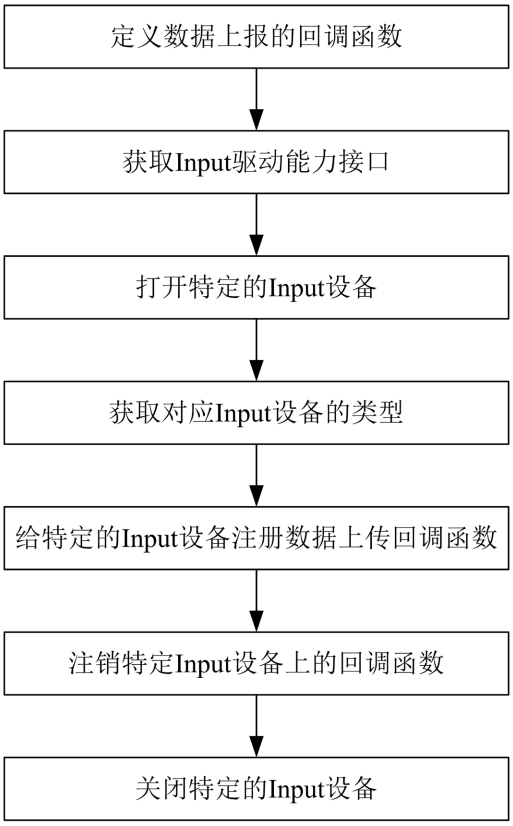

Figure 5. Steps to call Input HDI
图 5. 调用 Input HDI 的步骤

5. 结论

本文介绍了润开鸿鸿锐开发板 SCDAYU800A 的 OpenHarmony 操作系统的移植适配过程。在分析 OpenHarmony 标准系统的前提下，成功实现了 OpenHarmony 在润开鸿鸿锐开发板 SCDAYU800A 上的稳定运行。该研究为基于 RISC-V 架构的单板机系统开发提供了实践参考，具有一定的应用价值。

参考文献

[1] RISC-V 架构手册[EB/OL]. https://riscv.org/, 2024-12-11.

[2] 刘畅, 武延军, 吴敬征, 赵琛. RISC-V 指令集架构研究综述[J]. 软件学报, 2021, 32(12): 3992-4024.

[3] 林金龙, 何小庆. 深入理解 RISC-V 程序开发[M]. 北京: 航空航天大学出版社, 2021.

[4] OpenHarmony 官方文档[EB/OL]. https://www.openharmony.cn/docs/zh-cn/overview/, 2024-12-13.

[5] 润开鸿鸿锐开发板 SCDAYU800A 技术手册[EB/OL]. https://www.hoperun.com/, 2024-12-10.

[6] HDF 驱动开发框架[EB/OL]. OpenHarmony 开发者社区. https://www.openharmony.cn/docs/zh-cn/overview/, 2024-12-12.

[7] ITOPEN/DAYU800 [EB/OL]. https://gitee.com/itopen/dayu800, 2025-01-23.

Call for Papers

Embedded Technology and Intelligent Systems

嵌入式技术与智能系统

国际中文期刊征文启事

https://www.hanspub.org/journal/etis

ISSN: 3065-1220

《嵌入式技术与智能系统》是一本开放获取、关注集成传统嵌入式技术与新兴智能系统的前沿研究最新进展的国际中文期刊，期刊特别注重软件算法、芯片设计与硬件实施的协同进展，以及理论研究与工程实践的紧密结合，面向学术界学者、产业界专家与工程师、学生及技术爱好者，关注中国领先产业集群的广阔发展潜力。本期刊强调发表原创性、创新性及具有实用价值的研究成果。 该期刊由汉斯出版社出版，全球发行，现诚邀相关领域的学者投稿。

投稿领域：

人工智能技术-边缘计算-端侧智能和大模型嵌入式应用
GPT-行业GPT以及GPT在嵌入式及智能系统研发中的应用
信息物理融合系统(CPS)-物联网技术-感知计算和无线传感网-泛在电力物联网-智能电表-储能技术-智能输变电
嵌入式系统结构-嵌入式操作系统与中间件-Linux、安卓和开源鸿蒙应用
实时操作系统-虚拟化和容器技术-混合关键系统
嵌入式软件形式化建模-软件测试和仿真-功能安全技术
嵌入式软件云原生技术-CI/CD和DevOpt-微服务
软硬件协同设计-开源指令集和开源芯片-RISC-V产业生态
嵌入式SoC技术--MCU 创新与生态-FPGA/DSP技术和应用
AI芯片和算法-存储技术-GPU 技术-视觉芯片及嵌入式显控应用
CAN和工业总线技术-时间敏感系统-电机控制-PLC和工业PC
无线通信技术-WiFi/蓝牙/Mesh/蜂窝/5G网络-物联网安全-低功耗设计
嵌入式系统课程改革-物联网和AI教学研究-职业教育-企业人才培养
嵌入式智能系统应用（智能家居、可穿戴设备、机器人、医疗电子、汽车电子和航空航天等）

征文要求及注意事项：

1. 稿件务求主题新颖、论点明确、论据可靠、数字准确、文字精炼、逻辑严谨、文字通顺，具有科学性、先进性和实用性；
2. 稿件必须为中文,且须加有英文标题、作者信息、摘要、关键词和规范的参考文献列表；
3. 稿件请采用WORD排版，包括所有的文字、表格、图表、附注及参考文献；
4. 从稿件成功投递之日起，在2个月内请勿重复投递至其他刊物。本刊不发表已公开发表过的论文。文章严禁抄袭，否则后果自负；
5. 本刊采用同行评审的方式，审稿周期一般为5~14日。

欲了解更多信息请登录 https://www.hanspub.org/journal/etis　　联系邮箱：etis@hanspub.org

Hans汉斯

嵌入式技术与智能系统

主编：何立民　北京航空航天大学教授

主办：汉斯出版社　珠海昊谷电子科技有限公司

编辑：《嵌入式技术与智能系统》编委会

网址：https://www.hanspub.org/journal/etis

电子邮箱：etis@hanspub.org

出版：汉斯出版社