GEC6818 倒车雷达项目总结文档

一、项目概要

本项目基于 GEC6818 开发板，通过设计实现一个整合性的“倒车雷达系统”，包括以下核心部分：

- Linux 字符设备驱动编写（包括注册、创建设备节点、GPIO 操作等）
- 超声波传感器 HC-SR04 测距功能
- 蜂鸣器距离报警提示
- 按键控制启停逻辑（K2 启动，K6 停止）
- LED 根据测距结果显示“接近程度”

二、各传感器/设备原理与驱动详解

1. 超声波测距传感器 HC-SR04

⚙️**硬件原理:**

- **Trig（触发端）**：
- 由 CPU 控制输出，发送一个不低于 10 微秒的高电平；
- 模块检测到此脉冲后，自动发射 8 个周期的 40kHz 超声波。
- **Echo（接收端）**：
- 模块接收回波后将 Echo 引脚拉高；
- Echo 高电平持续的时间即为“超声波从发射到返回”的总时间。

🧮**引脚说明:**

- Trig：GPIO D19（输出），低 -> 高 -> 低（至少10微秒）
- Echo：GPIO D15（输入），读取高电平持续时间

🧮**电平逻辑:**

- Trig 发送：**高电平触发**（最少10us）
- Echo 回响：**高电平持续时间代表距离**

📐**驱动实现要点:**

```
// 触发 20us 高电平
gpio_set_value(sr04_gpios[0].gpio, 1);
udelay(20);
gpio_set_value(sr04_gpios[0].gpio, 0);
```

```
// 等待 Echo 高电平，并计时
while (gpio_get_value(ECHO) == 0) wait++;
while (gpio_get_value(ECHO) == 1) time++;
```

- Echo 检测中加入 timeout 机制，防止长时间阻塞
- 计算距离：`distance = t × 0.153f`（单位为厘米）

📌特点：

- Echo 信号为 **高电平表示有效回波**；
- 低电平时为无数据/等待状态；
- 距离越近，高电平时间越短。

---

2. 蜂鸣器（Buzzer）

⚙️工作原理：

- 控制引脚为 GPIO 输出，**低电平蜂鸣器响，** 高电平关闭；
- 使用 `write()` 接口写入 `0` / `1` 实现控制。

驱动控制策略：

- 距离越近，蜂鸣频率越快：

距离(cm)	蜂鸣节奏
⩽1	100ms 间隔闪烁
⩽2	300ms
⩽3	500ms
⩽4	900ms
>4	不响

📌特性：

- 使用 `/dev/mybeep` 控制
- 用户态传递 1 表示响，0 表示关闭
- 通过 `usleep()` 控制间隔节奏

---

3. 按键控制（K2 与 K6）

⚙️按键工作方式：

- 按键为 GPIO 输入，**未按下为高电平，按下为低电平**（上拉输入）；
- 读取返回的值为多按键状态组成的一个字节。

驱动按键位分配：

- `K2`：使用 `val_key & 0x01`
- `K6`：使用 `val_key & 0x02`

用户态逻辑：

```
if ((val_key & 0x01) && !(last_key & 0x01)) // K2
if ((val_key & 0x02) && !(last_key & 0x02)) // K6
```

📌**特性：**

- 使用 `/dev/mykey` 读取按键值
- 轮询方式检测按键状态
- 通过 last_key 防抖和状态判断

---

4. LED 指示器

⚙️**工作原理：**

- **低电平亮灯，高电平灭灯**；（驱动中 GPIO 输出为 0 点亮）
- 控制 GPIO 输出电平，模拟 4 段距离状态。

GPIO 分配：

- D7：PAD_GPIO_E + 13
- D8：PAD_GPIO_C + 17
- D9：PAD_GPIO_C + 8
- D10：PAD_GPIO_C + 7

写入格式：

- `led_buf[0] = 1;`（控制标志）
- `led_buf[1] = n;`（点亮前 n 个灯）

逻辑关系：

距离(cm)	LED 亮灯个数
⩽1	4
⩽2	3
⩽3	2
⩽4	1
>4	0

驱动关键：

```
for (i = 0; i < 4; i++) gpio_set_value(..., 1); // 全灭
for (i = 0; i < kbuf[1]; i++) gpio_set_value(..., 0); // 点亮前 n 个
```

八、字符设备与 GPIO 封装机制详解

✂️字符设备框架原理

Linux 字符设备驱动通过如下步骤完成设备注册与用户交互：

- `alloc_chrdev_region()`：申请设备号
- `cdev_init()` + `cdev_add()`：注册字符设备结构体
- `class_create()` + `device_create()`：自动创建设备节点（如 /dev/myled）
- `file_operations`：定义 `.open`、`.read`、`.write` 等系统调用

✒️封装 GPIO 的方式

Linux 提供 GPIO API 来进行统一封装：

- `gpio_request()`：申请 GPIO 控制权
- `gpio_direction_output()`：设置为输出
- `gpio_set_value(gpio, value)`：设置电平（1=高电平，0=低电平）
- `gpio_get_value(gpio)`：读取输入电平
- `gpio_free()`：释放 GPIO

驱动层通过这些接口控制硬件，无需用户空间干预。用户只需 `write(fd, data, len)` 即可控制底层 GPIO。

🛏️`gpio_set_value(gpio, value)` 详解

这是一个用于设置 GPIO 输出引脚电平的函数，其作用为：

```
void gpio_set_value(unsigned gpio, int value);
```

- `gpio`：GPIO 引脚编号
- `value`：写入的电平值
- `1`：高电平（电压为 VDD）
- `0`：低电平（接地）

⚠️注意：GPIO 必须已设置为输出方向，才能使用 `gpio_set_value()`。

LED 低电平点亮的实现原理： LED 一端接 VCC，另一端连接 GPIO 口，如果 GPIO 输出低电平，则形成电流回路，LED 点亮。

🌂GPIO 方向详解与示例

在 Linux 驱动开发中，**GPIO 方向（Direction）**非常关键，表示该 GPIO 是用来输入数据还是输出信号。

GPIO 方向	含义	典型用途
输出（Output）	通过 `gpio_set_value()` 设置高低电平，控制外设动作	控制 LED、蜂鸣器、超声波 Trig 触发信号等
输入（Input）	通过 `gpio_get_value()` 读取当前电平，接收外部信号	读取按键状态、超声波 Echo 回响信号等

为什么要设置 GPIO 方向?

GPIO 是双向引脚，不设置方向时，驱动无法判断该引脚是用来“发送信号”还是“接收信号”，

- 如果错误设置，可能导致硬件损坏；
- 逻辑混乱导致功能失效。

设置 GPIO 方向的关键函数

```
c
int gpio_direction_output(unsigned gpio, int value);
```

复制编辑

`int gpio_direction_output(unsigned gpio, int value);`

- 设置指定 GPIO 为输出方向；
- `value` 指定初始电平（0 或 1）；

```
c
int gpio_direction_input(unsigned gpio);
```

复制编辑

`int gpio_direction_input(unsigned gpio);`

- 设置指定 GPIO 为输入方向；

这两个函数需要在 `gpio_request()` 成功之后调用。

代码示例：LED 点亮流程

假设 LED 接法是 VCC 连接一端，GPIO 连接另一端。

```
c
// 设置 LED GPIO 为输出，初始状态为高电平（LED 灭）
```

```
gpio_direction_output(led_gpio_num, 1);

// 点亮 LED（输出低电平）
gpio_set_value(led_gpio_num, 0);

// 熄灭 LED（输出高电平）
gpio_set_value(led_gpio_num, 1);
```

复制编辑

```
// 设置 LED GPIO 为输出，初始状态为高电平（LED 灭） gpio_direction_output(led_gpio_num, 1); // 点亮 LED（输出低电平） gpio_set_value(led_gpio_num, 0); // 熄灭 LED（输出高电平） gpio_set_value(led_gpio_num, 1);
```

解释：

- 高电平时 GPIO 与 VCC 同电位，无电压差，LED 熄灭；
- 低电平时 GPIO 拉低电压，形成电流回路，LED 点亮。

这样设置 GPIO 方向，结合 `gpio_set_value()` 和 `gpio_get_value()`，能正确控制和读取硬件信号，保证驱动稳定运行。

是否需要我继续为文档添加用户态应用部分代码分析与封装细节？