AG32 GPIO 控制与上下拉实现

AG32 (AGRV2K) 的 GPIO 控制完整指南:信号模型、寄存器、SDK API、上下拉实现、输入/输出/中断,以及工程级注意事项。

---

1. AG32 GPIO 信号模型

AG32 内部 GPIO 信号命名格式是 `GPIOx_y`,其中:
- `x` = GPIO 组 (0~9,共 10 组)
- `y` = bit 编号 (0~7,每组 8 bit)
- 寄存器基址: `GPIO0_BASE = 0x40014000`,每组 +0x1000

关键概念:GPIO 信号 ≠ 物理引脚。 `GPIO4_1` 是 MCU 内部一个信号名,需要在 VE 文件里绑到具体物理引脚(如 `PIN_34`)才有用。

```
# my_board.ve
GPIO4_1 PIN_34    # 把 GPIO4_1 信号绑到物理引脚 34
```

---

2. GPIO 寄存器(GPIO_TypeDef)

```
typedef struct {
  __IO uint32_t GpioDATA[256];  // 0x000-0x3FC, 位带别名
  __IO uint32_t GpioDIR;        // 0x400, 方向: 1=output, 0=input
  __IO uint32_t GpioIS;         // 0x404, 中断: 1=电平 0=边沿
  __IO uint32_t GpioIBE;        // 0x408, 中断: 1=双边沿
  __IO uint32_t GpioIEV;        // 0x40C, 中断: 1=高/上升 0=低/下降
  __IO uint32_t GpioIE;         // 0x410, 中断屏蔽 (1=enable)
  __I  uint32_t GpioRIS;        // 0x414, 原始中断状态
  __I  uint32_t GpioMIS;        // 0x418, 屏蔽后中断状态
  __O  uint32_t GpioIC;         // 0x41C, 写 1 清中断
  __IO uint32_t GpioAFSEL;      // 0x420, 1=AF 模式 0=IO 模式
} GPIO_TypeDef;
```

2.1 寄存器地址(各组)

GPIO 组	基址
GPIO0	0x40014000
GPIO1	0x40015000
GPIO2	0x40016000
GPIO3	0x40017000
GPIO4	0x40018000
GPIO5	0x40019000
GPIO6	0x4001A000
GPIO7	0x4001B000
GPIO8	0x4001C000
GPIO9	0x4001D000

2.2 GpioDATA 位带别名(关键)

`GpioDATA[0..255]` 是**位带别名**,写 `GpioDATA[bits]` 等价于"操作 `bits` 中为 1 的位":
- 写 `0xFF` = 把 `bits` 中所有位拉高
- 写 `0x00` = 把 `bits` 中所有位拉低
- 写 `0xAB` = 把 `bits` 中对应 1 的位设为 0xAB 对应位的值
- 读 `GpioDATA[bits]` = 读 `bits` 中所有位的当前值

位带别名的妙处:写操作只影响 `bits` 中为 1 的位,其他位保持不变。不需要 read-modify-write 就能原子地改单 bit。

```
// 例子:设 bit 0 高,bit 1 低,其他 bit 不动
GPIO0->GpioDATA[0x03] = 0x01;  // bit 0 写 1,bit 1 写 0
```

---

3. SDK GPIO API(`gpio.h`)

所有 GPIO 操作都通过 `gpio.h` 的 inline 函数,**不要**直接读写寄存器(可读性差、易错)。

3.1 时钟使能(必须先做)

```
#include "system.h"

SYS_EnableAPBClock(APB_MASK_GPIO0);  // GPIO0 时钟
SYS_EnableAPBClock(APB_MASK_GPIO4);  // GPIO4
```

不使能时钟就操作寄存器,寄存器读不到值,操作没效果。

3.2 基础 IO 操作

```
// 方向
GPIO_SetOutput(GPIO0, GPIO_BIT0);    // 设 bit 0 为输出
GPIO_SetInput(GPIO0, GPIO_BIT0);     // 设 bit 0 为输入

// 输出
GPIO_SetHigh(GPIO0, GPIO_BIT0);      // 拉高 (output 1)
GPIO_SetLow(GPIO0, GPIO_BIT0);       // 拉低 (output 0)
GPIO_SetValue(GPIO0, GPIO_BIT0, 1);  // 拉高
GPIO_Toggle(GPIO0, GPIO_BIT0);        // 翻转

// 输入读取
uint8_t v = GPIO_GetValue(GPIO0, GPIO_BIT0);  // 读 0 或 1
```

多 bit 同时操作:

```
GPIO_SetOutput(GPIO4, GPIO_BIT0 | GPIO_BIT1);  // bit 0/1 都设 output
GPIO_SetHigh(GPIO4, GPIO_BIT0 | GPIO_BIT1);    // bit 0/1 都拉高
```

3.3 复用模式(AF)

当引脚要给外设用(UART / SPI / I2C 等),必须切到 AF 模式,否则外设无输出。

```
// 启用 UART0 复用 (SDK 自动操作 GPIO AFSEL)
GPIO_AF_ENABLE(GPIO_AF_PIN(UART, 0, UARTRXD));
GPIO_AF_ENABLE(GPIO_AF_PIN(UART, 0, UARTTXD));

// 关掉 AF (回到普通 GPIO 模式)
GPIO_AF_DISABLE(GPIO_AF_PIN(UART, 0, UARTTXD));
```

`GPIO_AF_PIN(UART, 0, UARTTXD)` **宏** 内部自动算出:
- 哪个 GPIO group
- 哪个 bit
- AFSEL 寄存器要写什么

不要手动找 AF 寄存器,直接用 `GPIO_AF_ENABLE` / `GPIO_AF_DISABLE` 。

3.4 中断

```
// 1. 清中断(防止上电残留)
GPIO_ClearInt(GPIO2, GPIO_BIT0);

// 2. 配置触发方式
GPIO_IntConfig(GPIO2, GPIO_BIT0, GPIO_INTMODE_FALLEDGE);
// GPIO_INTMODE 5 种:
//   LOWLEVEL  - 低电平触发
//   HIGHLEVEL - 高电平触发
//   FALLEDGE  - 下降沿
//   RISEEDGE  - 上升沿
//   BOTHEDGE  - 双边沿

// 3. 启用该 bit 的中断
GPIO_EnableInt(GPIO2, GPIO_BIT0);

// 4. 在 PLIC 注册 ISR
INT_EnableIRQ(GPIO2_IRQn, 1);  // priority 1

// 5. ISR handler — SDK 默认符号 GPIO<x>_isr
void GPIO2_isr(void) {
  GPIO_ClearInt(GPIO2, GPIO_BIT0);  // 必须在 ISR 入口清中断
  // ... 你的处理
}
```

注意:
- ISR 里必须**先清中断**(`GPIO_ClearInt`),否则 PLIC 会一直触发
- ISR 名字格式 `GPIOx_isr` 是 SDK weak 符号,直接重定义就覆盖
- 优先级 1 是最高有效优先级(数越小越高,看 `interrupt.h` 的 `MIN_IRQ_PRIORITY`)

---

4. 上下拉实现(硬件级)

AG32 上下拉是 **引脚级硬件特性**,在 Quartus P&R 阶段配置,**运行时不能改**。SDK 运行时函数 `GPIO_SetInput / GPIO_SetOutput` 只设方向,**不**改上下拉。

4.1 配置位置

通过 VE 文件的 **.asf**(Altera Settings File)在 Quartus 编译时配置:

```
# my_board.asf
puts "DESIGN ASF -- GPIO pull-up/down"

# 弱上拉 (推荐 5-100 kΩ,功耗低)
```

```
set_instance_assignment -name WEAK_PULL_UP_RESISTOR ON -to PIN_16
set_instance_assignment -name WEAK_PULL_UP_RESISTOR ON -to PIN_23

# 弱下拉
set_instance_assignment -name WEAK_PULL_DOWN_RESISTOR ON -to PIN_26

# 驱动电流(可选,默认 8mA,LED 驱动可加大到 16mA)
set_instance_assignment -name CURRENT_STRENGTH 16MA -to PIN_31
```

4.2 上下拉与输入模式的关系

GPIO 方向	上下拉是否生效
Input (读 GPIO 值)	✅ 生效
Output (驱动 GPIO 值)	❌ 不生效(输出驱动)
AF (外设复用)	⚠ 取决于外设,部分场景仍有效

关键:即使引脚是 Input 模式,**没设上拉/下拉**的话,引脚是浮空,读到电平是随机的(EMI / 静电干扰)。**按键输入**、**开关检测**、**空闲状态确认** 都需要上下拉。

4.3 工程集成

platformio.ini 设 `design.asf = ./my_board.asf`, `pio run -t buildlogic` 时 Supra 编译器会读 asf 配置:

```
[setup_logic]
logic_ve     = my_board.ve
design.asf   = ./my_board.asf
logic_device = AGRV2KL100
```

注意改 ASF 后必须 `pio run -t logic_clean` + `pio run -t buildlogic` :sconsign cache 不跟踪 ASF 修改。

4.4 验证上下拉生效

读 pin 状态:

```
uint8_t v = GPIO_GetValue(GPIO, GPIO_BIT);  // 高 = 上拉生效,低 = 下拉生效
```

配置	没接外设时读到
Pull-up	1 (高)
Pull-down	0 (低)
浮空	随机

debug 推荐:在 main loop 里**周期性读 input pin**,打印 `pull-up OK` / `pull-down OK` / `FAIL`。如果读到跟预期相反,说明 ASF 配置没生效(看 buildlog 里 `Processing my_board.asf` 是否出现)。

---

5. GPIO 实战模式

5.1 输出模式(LED / 驱动外部电路)

```
// 1. 使能时钟
SYS_EnableAPBClock(APB_MASK_GPIO4);

// 2. 设方向 output
GPIO_SetOutput(GPIO4, GPIO_BIT0 | GPIO_BIT1 | GPIO_BIT2 | GPIO_BIT3);

// 3. 写值
GPIO_SetHigh(GPIO4, GPIO_BIT0 | GPIO_BIT1);  // LED1+2 灭 (active-low)
GPIO_SetLow(GPIO4, GPIO_BIT0);                // LED1 亮

// 4. 翻转
GPIO_Toggle(GPIO4, GPIO_BIT0);
```

active-low LED 用法:GPIO 输出 0 = 亮,1 = 灭。

5.2 输入模式(按键 / 开关检测)

```
// 1. 使能时钟
SYS_EnableAPBClock(APB_MASK_GPIO0);

// 2. 设方向 input
GPIO_SetInput(GPIO0, GPIO_BIT0);

// 3. 在 ASF 里设上拉(否则浮空)
set_instance_assignment -name WEAK_PULL_UP_RESISTOR ON -to PIN_16

// 4. 读
uint8_t pressed = GPIO_GetValue(GPIO0, GPIO_BIT0) == 0;  // active-low 按下
```

active-low 按键:按键一端接 MCU,另一端接 GND。没按时引脚被上拉拉高,按下时拉低 → 读到 0 = 按下。

5.3 中断模式(低功耗场景)

```
// pin_init()
SYS_EnableAPBClock(APB_MASK_GPIO2);
GPIO_SetInput(GPIO2, GPIO_BIT0);
GPIO_ClearInt(GPIO2, GPIO_BIT0);
GPIO_IntConfig(GPIO2, GPIO_BIT0, GPIO_INTMODE_FALLEDGE);  // 下降沿
INT_EnableIRQ(GPIO2_IRQn, 1);
GPIO_EnableInt(GPIO2, GPIO_BIT0);

// 中断服务程序
void GPIO2_isr(void) {
  GPIO_ClearInt(GPIO2, GPIO_BIT0);  // 必须先清
  // 处理
}
```

5.4 流水灯 demo

```
// 4 LED active-low,主循环 500ms 切换
static const uint8_t led_bits[] = { GPIO_BIT0, GPIO_BIT1, GPIO_BIT2, GPIO_BIT3 };
static const uint8_t led_mask  = GPIO_BIT0 | GPIO_BIT1 | GPIO_BIT2 | GPIO_BIT3;

void led_init(void) {
  SYS_EnableAPBClock(APB_MASK_GPIO4);
  GPIO_SetOutput(GPIO4, led_mask);
  GPIO_SetHigh(GPIO4, led_mask);  // 全灭
}

void led_waterfall(uint8_t phase) {
  for (uint8_t i = 0; i < 4; i++) {
    if (i == phase) GPIO_SetLow(GPIO4, led_bits[i]);   // ON
    else            GPIO_SetHigh(GPIO4, led_bits[i]);  // OFF
  }
}

// main
while (1) {
  led_waterfall(phase);
  phase = (phase + 1) & 0x03;
  UTIL_IdleUs(500e3);  // 500ms
}
```

---

6. 常见错误

6.1 GPIO 操作没效果

症状: `GPIO_SetHigh` 后读 pin 还是 0。

排查:
1. **没使能时钟** — `SYS_EnableAPBClock(APB_MASK_GPIOx)` 必须调
2. **方向设错** — Output 模式才能拉高拉低,Input 模式读不到
3. **VE 没绑引脚** — 寄存器操作改了,但 bit 对应的物理引脚没绑,外部看不到
4. **VE 路由了 AF 模式** — GPIO_AFSEL 寄存器被 SDK 改了,被外设占用

debug:

```
// 读 GpioDIR,看是不是 output
uint32_t dir = GPIO4->GpioDIR;
printf("GPIO4 DIR = 0x%X\n", dir);  // bit 0/1/2/3 应该是 1

// 读 GpioDATA,看实际值
uint32_t val = GPIO4->GpioDATA[0x0F];
printf("GPIO4 DATA = 0x%X\n", val);
```

6.2 上拉/下拉不生效

症状:Input pin 读到 0(应该上拉读到 1),或读到 1(应该下拉读到 0)。

排查:
1. **ASF 配置没生效** — 跑 `pio run -t logic_clean` + `pio run -t buildlogic`,看 buildlog 里 `Processing my_board.asf` 出现
2. **asf 文件最后一行有换行符** — 没换行的话最后一条 `set_instance_assignment` 不生效
3. **引脚 VE 绑错** — `set_instance_assignment -to PIN_x` 中的 PIN_x 跟 VE 里信号绑的引脚要一致
4. **外部干扰** — 拉一根线到 VCC / GND 验证,排除外部干扰

6.3 中断一直触发

症状:ISR 一秒触发几十次,看起来像"幽灵触发"。

排查:
1. **没清中断** — ISR 入口必须 `GPIO_ClearInt`,否则 PLIC 一直 pending
2. **中断模式设错** — `GPIO_INTMODE_FALLEDGE` 但没设上拉,引脚浮空,电平在高低之间漂移触发假中断
3. **多个 bit 共享一个 IRQ** — `GPIO2_IRQn` 是整个 GPIO2 组的中断, `GpioRIS` 寄存器有具体哪个 bit 触发。ISR 里**只清触发的那一个 bit**,别 `ClearInt` 整个 group

6.4 编译错误 `APB_MASK_GPIOGPIO_IDX undeclared`

症状:用了 `PERIPHERAL_ENABLE(GPIO, GPIO_IDX)` 宏,SDK `peripherals.h` 字符串拼接问题。

修法:**直接调** `SYS_EnableAPBClock(APB_MASK_GPIO4)` ,不要用 `PERIPHERAL_ENABLE(GPIO, IDX)` 宏。

---

7. SDK API 速查

```
// 时钟
void SYS_EnableAPBClock(uint32_t mask);     // APB_MASK_GPIOx 之类
void SYS_DisableAPBClock(uint32_t mask);

// 方向
void GPIO_SetOutput(GPIO_TypeDef *gpio, uint8_t bits);
void GPIO_SetInput(GPIO_TypeDef *gpio, uint8_t bits);

// 输出
void GPIO_SetHigh(GPIO_TypeDef *gpio, uint8_t bits);
void GPIO_SetLow(GPIO_TypeDef *gpio, uint8_t bits);
void GPIO_SetValue(GPIO_TypeDef *gpio, uint8_t bits, uint8_t value);
void GPIO_Toggle(GPIO_TypeDef *gpio, uint8_t bits);

// 输入
uint8_t GPIO_GetValue(GPIO_TypeDef *gpio, uint8_t bits);

// AF
void GPIO_AF_ENABLE(uint32_t pin);   // pin = GPIO_AF_PIN(UART, 0, UARTTXD)
void GPIO_AF_DISABLE(uint32_t pin);

// 中断
void GPIO_ClearInt(GPIO_TypeDef *gpio, uint8_t bits);
void GPIO_IntConfig(GPIO_TypeDef *gpio, uint8_t bits, uint32_t mode);
void GPIO_EnableInt(GPIO_TypeDef *gpio, uint8_t bits);
void GPIO_DisableInt(GPIO_TypeDef *gpio, uint8_t bits);
uint8_t GPIO_IsRawIntActive(GPIO_TypeDef *gpio, uint8_t bits);
uint8_t GPIO_IsMaskedIntActive(GPIO_TypeDef *gpio, uint8_t bits);

// 模式常量
GPIO_INTMODE_LOWLEVEL
GPIO_INTMODE_HIGHLEVEL
GPIO_INTMODE_FALLEDGE
GPIO_INTMODE_RISEEDGE
GPIO_INTMODE_BOTHEDGE
```

---

8. 关联文档

- 新建工程前必看-10个检查点.md — VE / ASF / board 配置
- 使用VE配置引脚-非自定义logic.md — 完整 VE-only 烧录流程
- 芯片型号对应board和引脚数对照表.md — 选 board
- SKILL.md — 入口