3.3 具体单板的 GPIO 操作方法

请使用 GIT 下载文档后，看图 3.1 红框所示目录中各板子对应的文档及图片。

网盘中相同名字的目录下也有对应的视频。

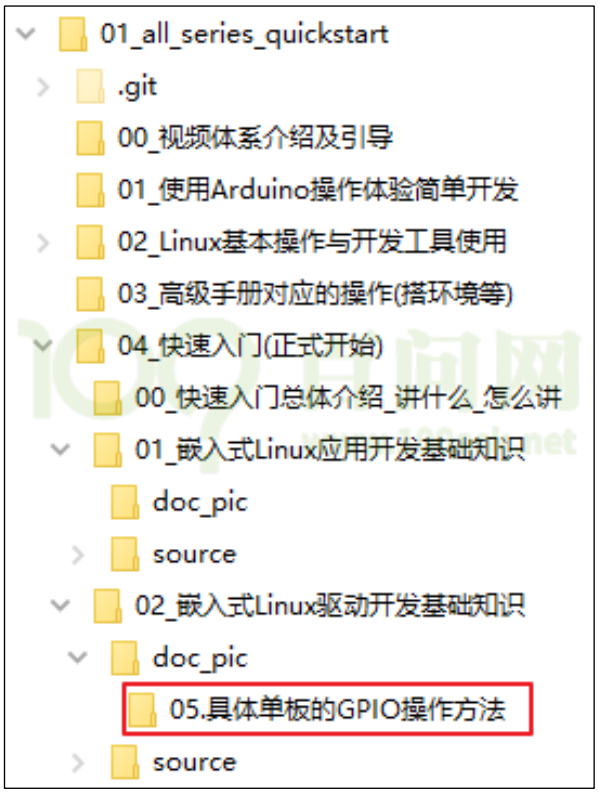


图 3.1 GPIO 操作文档在 git 仓库所在位置

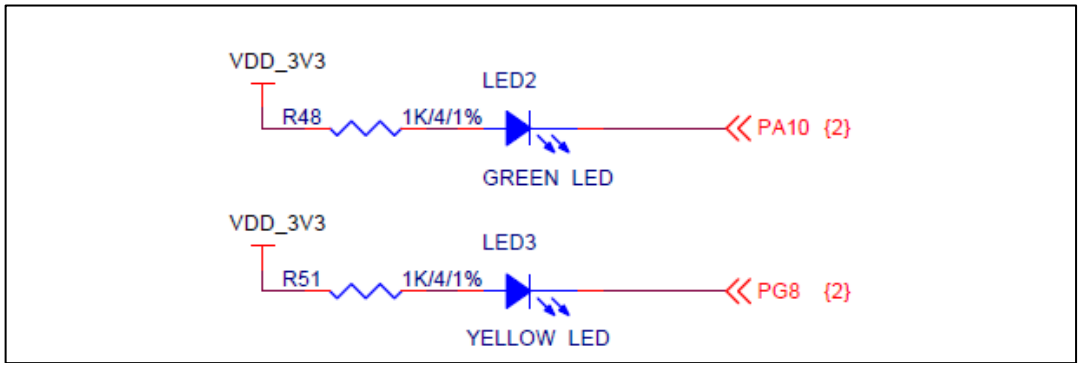
为方便学习，在本文档中也把上述 GIT 目录中的文档添加进来了。

3.4 STM32MP157 GPIO 操作方法

RCC: Reset and clock control (复位和时钟控制)

GPIO: General-purpose input/output，通用的输入输出口

如下图所示，我们需要操作 LED2 引脚标号为 PA10 LED3 引脚标号为 PG8。



3.4.1 STM32MP157 的 GPIO 模块结构

参考资料：CPU 开发手册 DM00327695.pdf《13: General-purpose I/Os(GPIO)》。

GPIO 寄存器有多种，包括功能寄存器和 GPIO 时钟寄存器，下面进行具体介绍，如下为引脚对应的表。

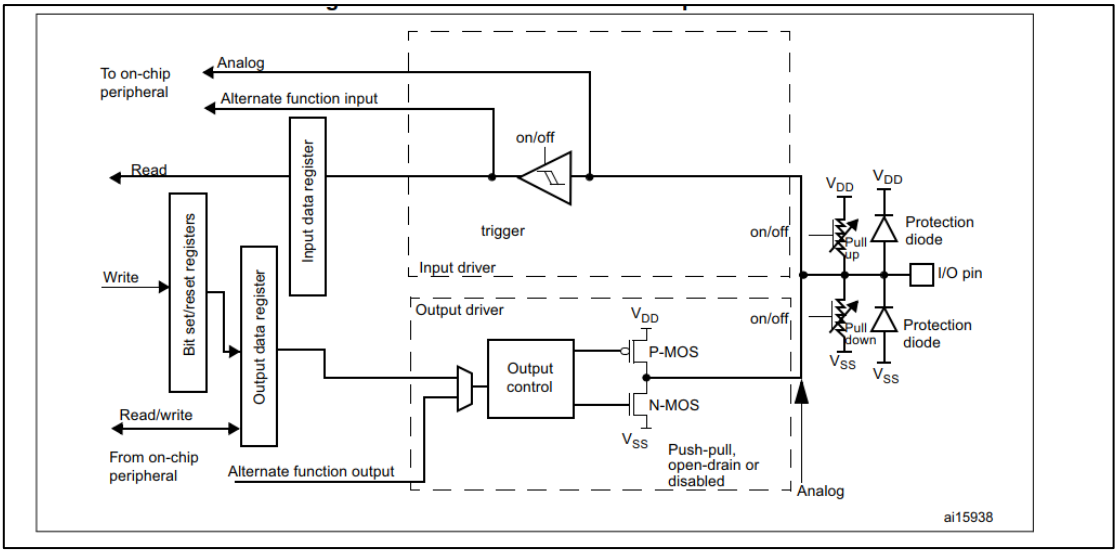
```
GPIOA (16 pins) <--对应--> gpiochip0
GPIOB (16 pins) <--对应--> gpiochip1
GPIOC (16 pins) <--对应--> gpiochip2
GPIOD (16 pins) <--对应--> gpiochip3
GPIOE (16 pins) <--对应--> gpiochip4
GPIOF (16 pins) <--对应--> gpiochip5
GPIOG (16 pins) <--对应--> gpiochip6
GPIOH (16 pins) <--对应--> gpiochip7
GPIOI (16 pins) <--对应--> gpiochip8
          ....
GPIOZ (8 pins)  <--对应--> gpiochip9
```

由于 GPIO 的功能多种多样，我们需要根据实际功能来设置 GPIO 引脚的工作模式。对于 STM32MP157 来说,每一个 GPIO 端口有四个 32 位的配置寄存器（GPIOx_MODER, GPIOx_OTYPER, GPIOx_OSPEEDR 和 GPIOx_PUPDR），两个 32 位的数据寄存器(GPIOx_IDR 和 GPIOx_ODR)，一个 32 位的设置/复位寄存器(GPIOx_BSRR)。此外，所有的 GPIO 都有一个 32 位的锁定寄存器（GPIOx_LCKR）和两个 32 位的多功能选择寄存器（GPIOx_AFRH and GPIOx_AFRL）。此外，还有 GPIO 外设时钟控制寄存器。

通过编程寄存器，我们可以设置 GPIO 为不同的模式，包括以下几种。

- 输入悬空
- 输入上拉
- 输入下拉
- 模拟输入
- OD 输出，支持上拉或者下拉
- Push-pull 输出，支持上拉或者下拉
- 多功能 push-pull 输出，支持上拉或者下拉
- 多功能 OD 输出，支持上拉或者下拉

下图为 GPIO 的基本结构简图，可以看到支持的各种不同模式和对应的寄存器，我们需要根据实际的功能来设置输入输出功能。



每个 GPIO 端口的基地址不同，具体的对应可以通过查看手册《dm00327659.pdf》的 memory map 章节来查看，下面红框内是本次实验使用到的 GPIOA 和 GPIOG 的基地址。

		0x5000A000 - 0x5000A3FF	1 KB	GPIOI	*GPIO registers*
		0x50009400 - 0x50009FFF	3 KB	Reserved	-
		0x50009000 - 0x500093FF	1 KB	GPIOH	*GPIO registers*
		0x50008400 - 0x50008FFF	3 KB	Reserved	-
		0x50008000 - 0x500083FF	1 KB	GPIOG	*GPIO registers*
		0x50007400 - 0x50007FFF	3 KB	Reserved	-
	AHB4	0x50007000 - 0x500073FF	1 KB	GPIOF	*GPIO registers*
		0x50006400 - 0x50006FFF	3 KB	Reserved	-
		0x50006000 - 0x500063FF	1 KB	GPIOE	*GPIO registers*
		0x50005400 - 0x50005FFF	3 KB	Reserved	-
		0x50005000 - 0x500053FF	1 KB	GPIOD	*GPIO registers*
		0x50004400 - 0x50004FFF	3 KB	Reserved	-
		0x50004000 - 0x500043FF	1 KB	GPIOC	*GPIO registers*
		0x50003400 - 0x50003FFF	3 KB	Reserved	-
		0x50003000 - 0x500033FF	1 KB	GPIOB	*GPIO registers*
		0x50002400 - 0x50002FFF	3 KB	Reserved	-
		0x50002000 - 0x500023FF	1 KB	GPIOA	*GPIO registers*
		0x50001400 - 0x50001FFF	3 KB	Reserved	-

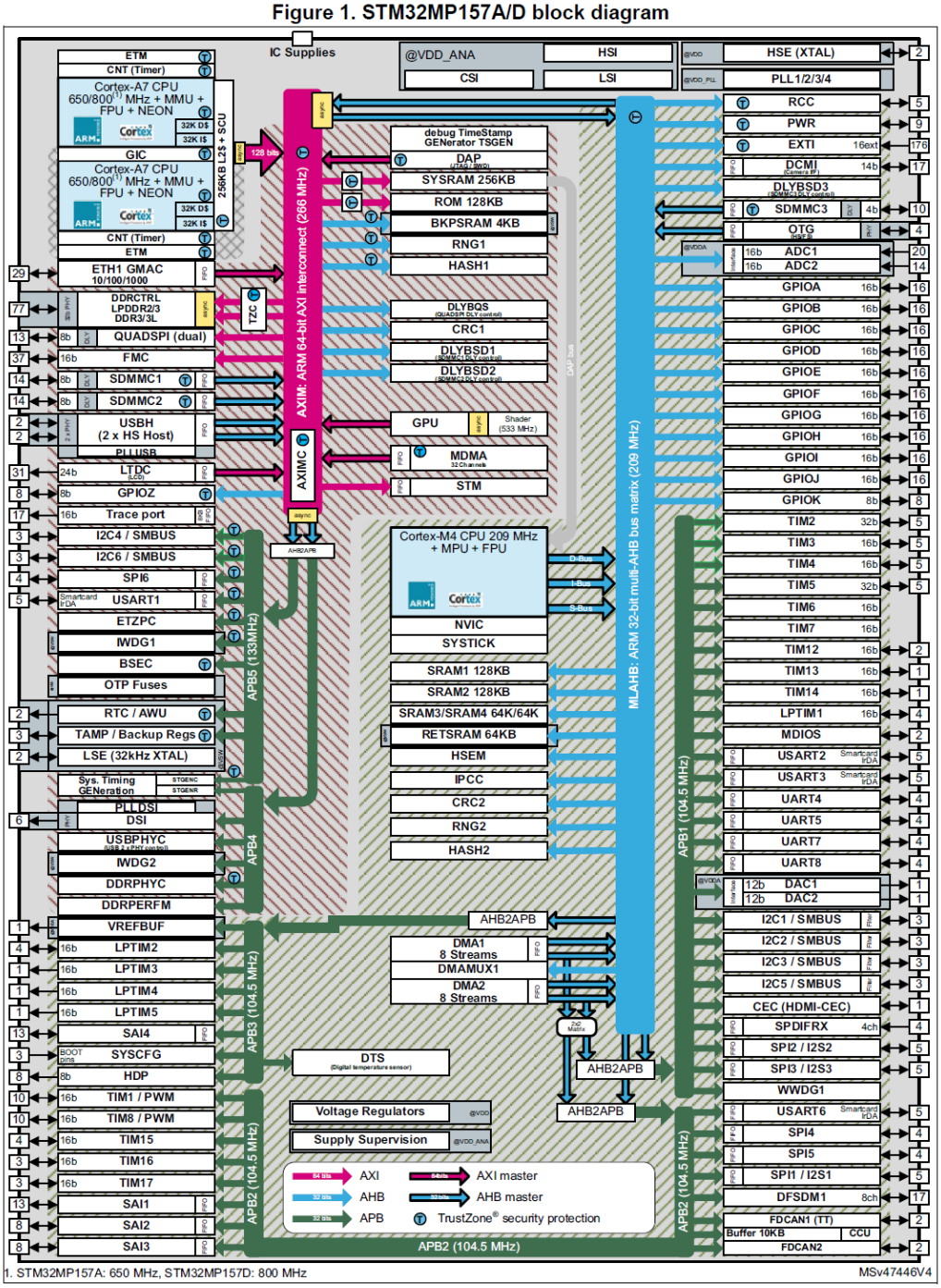
GPIO 的使用涉及到多个寄存器，包括 RCC 时钟 GPIOx_MODER、GPIOx_IDR 和 GPIOx_ODR 等。

3.4.2 RCC 用于设置是否向 GPIO 模块提供时钟

参考资料：

- CPU 数据手册 DS12504 Rev 4.pdf《2 Description – Figure 1.STM32MP157A block diagram》
- CPU 开发手册 DM00327695.pdf《10 Reset and clock control (RCC)》

Figure 1. STM32MP157A/D block diagram



GPIO 外设的时钟来源各自不同，具体的可以从手册当中可以看到，详见《dm00327659.pdf》RCC 章节 Table56。下图为截图，其中 GPIOA-K 的时钟来源为 hclk4，GPIOZ 的时钟来源为 hclk5。因此为了使用 GPIO，我们需要使能锁相环和外设 GPIO 各自对应的时钟。

FDCAN	Kernel	←	A	100	FDCANSRC	**0**	hse_ker_ck
						1	pll3_q_ck
						2	pll4_q_ck
						3	pll4_r_ck
	Bus	←	-	104.5	-	-	pclk2
FMC	Kernel	←	A	266	FMCSRC	**0**	aclk
						1	pll3_r_ck
						2	pll4_p_ck
						3	per_ck
	Bus	←	-	266	-	-	hclk6
							aclk
GPIOA-K	Bus	←	-	209	-	-	hclk4
GPIOZ	Bus	←	-	266	-	-	hclk5
GPU	Kernel	←	-	533	-	-	pll2_q_ck
	Bus	←	-	266	-	-	hclk6
							aclk
HASH1	Bus	←	-	266	-	-	hclk5
HASH2	Bus	←	-	209	-	-	hclk3

接下来我们要找到配置锁相环 PLL4 时钟源的寄存器，我们需要将其使能，并等待完成锁定。查看 CPU 开发手册 DM00327695.pdf《10 Reset and clock control (RCC)》

10.7.42 RCC PLL4 Control Register (RCC_PLL4CR)

Address offset: 0x894

Reset value: 0x0000 0000

This register is used to control the PLL4.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	DIVREN	DIVQEN	DIVPEN	Res.	Res.	PLL4RDY	PLLON
									rw	rw	rw			r	rw

由于 STM32MP157 包含 cortex-m4 和 cortex-A7 两部分，并且 GPIO 可以分别由 MCU 和 APU 来控制，因此 GPIO 的时钟使能也分成两种。这里选择针对 APU 的进行介绍说明。

下图为针对 APU 的 GPIOA 至 K 的时钟使能寄存器，低 11 位有效。为了使用 GPIOA 和 GPIOG，我们需要将对应的位设置为 1。

10.7.155 RCC AHB4 Periph. Enable For MPU Set Register (RCC_MP_AHB4ENSETR)

Address offset: 0xA28

Reset value: 0x0000 0000

This register is used to set the peripheral clock enable bit of the corresponding peripheral to '1'. It shall be used to allocate a peripheral to the MPU. Writing '0' has no effect, reading will return the effective values of the corresponding bits. Writing a '1' sets the corresponding bit to '1'.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	GPIOKEN	GPIOJEN	GPIOIEN	GPIOHEN	GPIOGEN	GPIOFEN	GPIOEEN	GPIODEN	GPIOCEN	GPIOBEN	GPIOAEN
					rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs

3.4.3 GPIOx_MODER 配置 GPIO 模式

GPIOx_MODER 用于配置 GPIO 的模式，包括输入、通用输出、多功能和模拟共四种模式。该寄存器共 32 位,，涉及 16 个 GPIO，每个 GPIO 对应 2 位。GPIOx_MODER 的各位定义如下，我们在这里分别选择 00 和 01 两种，各自对应输入和输出模式。（上电默认为输入悬空模式）。其中 00 对应输入功能，01 对应输出功能。

13.4.1 GPIO port mode register (GPIOx_MODER) (x = A to K, Z)

Address offset:0x00

Reset value: 0xFFFF FFFF

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MODER7[1:0]		MODER6[1:0]		MODER5[1:0]		MODER4[1:0]		MODER3[1:0]		MODER2[1:0]		MODER1[1:0]		MODER0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **MODER[15:0][1:0]:** Port x configuration I/O pin y (y = 15 to 0)
These bits are written by software to configure the I/O mode.
00: Input mode
01: General purpose output mode
10: Alternate function mode
11: Analog mode

我们需要设置 PA10 对应 GPIOA10 其中 GPIOA_MODER 寄存器基地址为 0x50002000 由于每个 GOIO 引脚对应两个 bit 位，所以我们需要操作 bit[21:20]设置为 01 输出模式

PG8 对应 GPIOG8 其中 GPIOA_MODER 寄存器基地址为 0x50008000 由于每个 GOIO 引脚对应两个 bit 位，所以我们需要操作 bit[17:16]设置为 01 输出模式

3.4.4 GPIOx_OTYPER 配置 GPIO 输出

GPIOx_OTYPER 寄存器用于配置输出类型，可以设置为输出为 OD 或者 push-pull 两种结构。该寄存器共 32 位，低 16 位对应 16 个 GPIO，通过设置位的高低来选择 OD 或者 push-pull 的输出类型。

11: Analog mode

13.4.2 GPIO port output type register (GPIOx_OTYPER) (x = A to K, Z)

Address offset: 0x04

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OT15	OT14	OT13	OT12	OT11	OT10	OT9	OT8	OT7	OT6	OT5	OT4	OT3	OT2	OT1	OT0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **OT[15:0]:** Port x configuration I/O pin y (y = 15 to 0)
These bits are written by software to configure the I/O output type.
0: Output push-pull (reset state)
1: Output open-drain

3.5 GPIOx_IDR 设置输入 GPIO

GPIOx_IDR 是 GPIO 输入数据寄存器，用于存储外部输入的数据。可以通过读取该寄存器的值来判断外部输入电平的高低。具体的寄存器定义如下图所示，只使用低 16 位

13.4.5 GPIO port input data register (GPIOx_IDR) (x = A to K, Z)

Address offset: 0x10

Reset value: 0x0000 XXXX

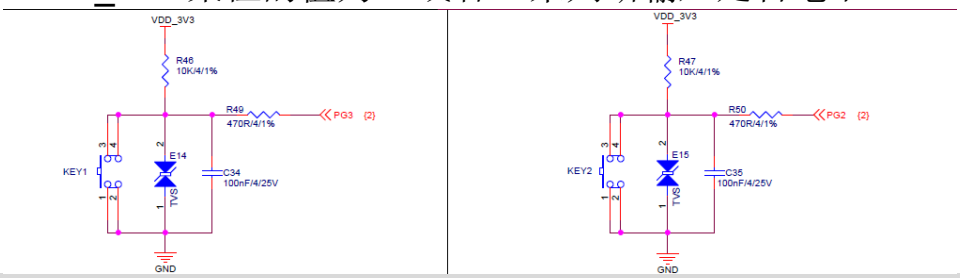
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IDR15	IDR14	IDR13	IDR12	IDR11	IDR10	IDR9	IDR8	IDR7	IDR6	IDR5	IDR4	IDR3	IDR2	IDR1	IDR0
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **IDR[15:0]:** Port x input data I/O pin y (y = 15 to 0)
These bits are read-only. They contain the input value of the corresponding I/O port.

3.5.1 读 GPIO

1. 设置 RCC_PLL4CR 使能 hclk4 使用的时钟。
2. 设置 RCC_MP_AHB4ENSETR 使能 GPIOA 外设时钟。
3. 设置 GPIOA_MODER 中某位为 1，把该引脚设置为输入功能。
4. 读 GPIOG_IDR 某位的值为 1 或者 0 来判断输入是否电平。




```
/* 使能 PLL4 */
/* RCC_PLL4CR 地址：0x50000000 + 0x894   */
pReg = (volatile unsigned int *)(0x50000000 + 0x894);
*pReg |= (1<<0);
while(*pReg & (1<<1) == 0);
/* for A7 (0x50000000 + 0xA28) */
gpio_clk = (volatile unsigned int *)(0x50000000 + 0xA28);
*gpio_clk |= (1<<6);//GPIOG 的外设时钟

//GPIOG 相关的寄存器地址
GPIOG_MODER  = (volatile unsigned int *)(0x50008000);
GPIOG_ODR    = (volatile unsigned int *)(0x50008014);
//设置 PG8 为输出    ，PG2 和 PG3 为输入
val = *GPIOG_MODER;
val &= ~((3<<4)|(3<<6));
unsigned int key_val=0;
```

```
unsigned int val;
val = *GPIOG_IDR;//读取 GPIOG 的输入值
 if(( val&0x0004)==0)//如果按键 KEY2，PG2 被按下，则输出 1
 {
    key_val = 1;
 }
else if((val &0x0008)==0) //如果按键 KEY1，PG3 被按下，则输出 1
{
    key_val = 2;
}
else//如果没有按键按下，则输出 0
{
    key_val = 0;
}
return key_val;//输出检测到的按键值
```

3.5.2 写 GPIO

1 设置 RCC_PLL4CR 使能 hclk4 使用的时钟。

2 设置 RCC_MP_AHB4ENSETR 使能 GPIOA 外设时钟。

3 设置 GPIOA_MODER 中某位为 1，把该引脚设置为输出功能。

4 写 GPIOA_ODR 某位的值为 1 或者 0，让其输出高电平或低电平。

```
/* 使能 PLL4 */
//锁相环 PLL4 初始
/* RCC_PLL4CR 地址 0x50000000 + 0x894*/
pll_clk = (volatile unsigned int *)(0x50000000 + 0x894);
*pll_clk |= (1<<0);//使能锁相环 PLL4
while(*pll_clk & (1<<1) == 0);//等待锁相环完成锁频

/*GPIOA 和 GPIOG 外设初始化
/* for A7 (0x50000000 + 0xA28) */
gpio_clk = (volatile unsigned int *)(0x50000000 + 0xA28);
*gpio_clk |= (1<<0);//使能 GPIOA 的外设时钟

unsigned int val;
//GPIOA 相关的寄存器地址
    GPIOA_MODER= (volatile unsigned int *)(0x50002000);
    GPIOA_ODR  = (volatile unsigned int *)(0x50002014);
//设置 PA10 为输出
    val = *GPIOA_MODER;
    val &= ~(3 << 20);
    val |= (1 << 20);
    *GPIOA_MODER = val;
//设置 PA10 输出低电平
    val = *GPIOA_ODR;
    val &= ~(1 << 10);
    *GPIOA_ODR = val;
```