

中南大学FYT战队 电控组培训讲义

主讲人：黄杰

学习要点

1. 简单了解寄存器的基本作用
2. 学会配置STM32标准库开发环境
3. 了解STM32不同编程开发环境的区别
4. 熟练掌握HAL库配置开发

学前准备

1. 芯片标准库包
2. Keil MDK
3. STM32CubeMX
4. 芯片对应参考手册

学习内容

一、操作寄存器方式点灯

看原理图，找到LED对应引脚GPIO，例如PF9

新建工程

打开KEIL、stm32f4xx中文参考手册，在Keil里新建project，文档中新建User 文件夹，新建main.c文件。

输入如下代码，作为框架：

```
int main(void)
{

    while(1)
    {

    }

}
```

打开时钟

打开stm32f4xx中文参考手册（53页），发现GPIOF 挂载在AHB1总线

0x4002 4000 - 0x4002 4FFF	BKPSRAM	AHB1	
0x4002 3C00 - 0x4002 3FFF	Flash 接口寄存器		第 3.8 节: Flash 接口寄存器
0x4002 3800 - 0x4002 3BFF	RCC		第 171 页的第 6.3.32 节: RCC
0x4002 3000 - 0x4002 33FF	CRC		第 85 页的第 4.4.4 节: CRC 寄
0x4002 2000 - 0x4002 23FF	GPIOI		第 192 页的第 7.4.11 节: GPIC
0x4002 1C00 - 0x4002 1FFF	GPIOH		
0x4002 1800 - 0x4002 1BFF	GPIOG		
0x4002 1400 - 0x4002 17FF	GPIOF		
0x4002 1000 - 0x4002 13FF	GPIOE		
0x4002 0C00 - 0x4002 0FFF	GPIOD		
0x4002 0800 - 0x4002 0BFF	GPIOC		
0x4002 0400 - 0x4002 07FF	GPIOB		
0x4002 0000 - 0x4002 03FF	GPIOA		

打开至AHB1的寄存器地址表（查找【RCC_AHB1ENR】）：

6.3.12 RCC_AHB1 外设时钟使能寄存器 (RCC_AHB1ENR)

RCC AHB1 peripheral clock enable register

偏移地址: 0x30

复位值: 0x0010 0000

访问：无等待周期，按字、半字和字节访问。

[illegible]

上

► 替换为

keil 中书写代码:

```
int main(void)
{
    RCC_AHB1ENR |= 1<<5;
    while(1)
    {

    }
}
```

设置GPIO

配置端口模式寄存器GPIOx_MODER为输出模式，ctrl+f 寻找【GPIOx_MODER】

7.4.1 GPIO 端口模式寄存器 (GPIOx_MODER) (x = A..I)

GPIO port mode register

偏移地址: 0x00

复位值:

- 0xA800 0000 (端口 A)
- 0x0000 0280 (端口 B)
- 0x0000 0000 (其它端口)

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]	
r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w
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]	
r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w

位 2y:2y+1 **MODERy[1:0]**: 端口 x 配置位 (Port x configuration bits) (y = 0..15)

这些位通过软件写入，用于配置 I/O 方向模式。

- 00: 输入 (复位状态)
- 01: 通用输出模式
- 10: 复用功能模式
- 11: 模拟模式

由手册53页，可以看到GPIOF的基地址为0x40021400，在这里我们要使用的是PF9引脚，因此配置GPIOx_MODER时，它的偏移地址就相对于GPIOF的基地址而言。

在这里我们是需要将19和18位中的数据赋值为0和1，代表着将PF9选择为输出模式。同时为了不影响其他位的数值，先将19和18位这两位清零，然后或上01。

```

int main(void)
{
    RCC_AHB1ENR |= 1<<5;
    GPIOF_MODER &= ~(0x03<<2 * 9);
    GPIOF_MODER |= (0x01<<2 * 9);
    while(1)
    {

    }
}

```

接下来选择输出模式，找到GPIO 端口输出类型寄存器 (GPIOx_OTYPER)。

7.4.2 GPIO 端口输出类型寄存器 (GPIOx_OTYPER) (x = A..I)

GPIO port output type register

偏移地址: 0x04

复位值: 0x0000 0000

GPIOF_OTYPER寄存器地址计算:
GPIOF基地址+偏移地址
0x40021400+0x04

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
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
r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w

将该位置0，为PF9引脚选择推挽输出

位 31:16 保留，必须保持复位值。

位 15:0 OTy[1:0]: 端口 x 配置位 (Port x configuration bits) (y = 0..15)

这些位通过软件写入，用于配置 I/O 端口的输出类型。

0: 输出推挽 (复位状态)

1: 输出开漏

```

RCC_AHB1ENR |= 1<<5;

GPIOF_MODER &= ~(0x03<<2 * 9);
GPIOF_MODER |= (0x01<<2 * 9);

GPIOF_OTYPER &= ~(0x01<<9);

```

引脚PF9选择推挽输出。

接下来是GPIO端口输出速度寄存器，GPIO 端口输出速度寄存器 (GPIOx_OSPEEDR) (x = A..I)。其实这里端口输出速度对于我们点亮led灯没有什么实际用处，配不配置都行，不过在这里我们还是给它配置一个50MHz的输出速度。

```

int main(void)
{
    RCC_AHB1ENR |= 1<<5;
    GPIOF_MODER &= ~(0x03<<2 * 9);
    GPIOF_MODER |= (0x01<<2 * 9);
    GPIOF_OTYPER &= ~(0x01<<9);
    GPIOF_OSPEEDR &= ~(0x03<<2*9);
    GPIOF_OSPEEDR |= (0x02<<2*9);
    while(1)
    {}
}

```

配置上拉或下拉，GPIO 端口上拉/下拉寄存器 (GPIOx_PUPDR) (x = A..I)。在这里我们选择配置为no pull/no down输出：

```

GPIOF_PUPDR &= ~(0x03<<2*9);
GPIOF_PUPDR |= (0x00<<2*9);

```

最后终于到我们的数据输出寄存器了，GPIO 端口输出数据寄存器 (GPIOx_ODR) (x = A..I)，将该寄存器中的第九位数值赋0，即可让GPIO输出高电平，即LED亮（有些板子LED电路需要GPIO输出低电平）。

```

GPIOF_ODR &= ~(0x01<<9);

```

最终代码：

```

int main(void)
{
    RCC_AHB1ENR |= 1<<5;

    GPIOF_MODER &= ~(0x03<<2 * 9);
    GPIOF_MODER |= (0x01<<2 * 9);

    GPIOF_OTYPER &= ~(0x01<<9);

    GPIOF_OSPEEDR &= ~(0x03<<2*9);
    GPIOF_OSPEEDR |= (0x02<<2*9);

    GPIOF_PUPDR &= ~(0x03<<2*9);
    GPIOF_PUPDR |= (0x00<<2*9);
}

```

```

GPIOF_ODR &= ~(0x01<<9);
while(1)
{

}
}

```

宏定义：

```

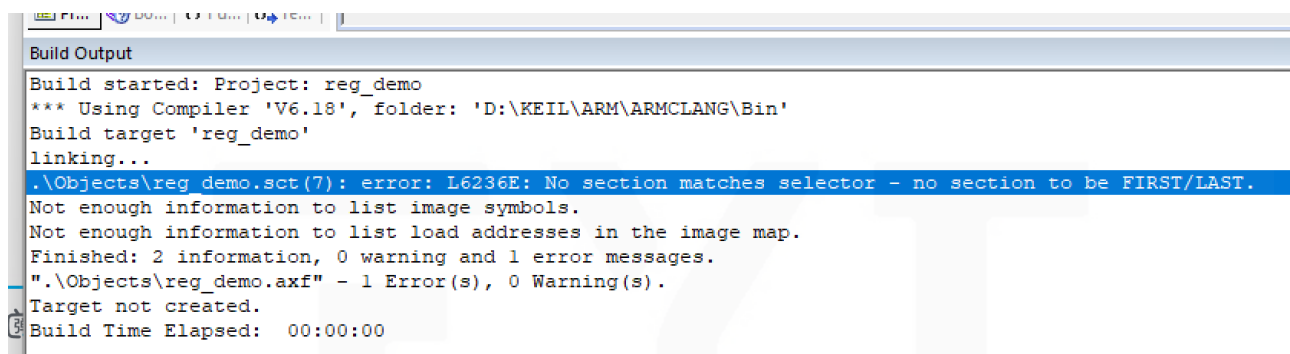
#define RCC_BASEADDR      0x40023800
#define RCC_AHB1ENR      *((volatile unsigned int *)
(RCC_BASEADDR+0x30))

#define GPIOF_BASEADDR    0x40021400

#define GPIOF_MODER      *((volatile unsigned int *)
(GPIOF_BASEADDR+0x00))
#define GPIOF_OTYPER     *((volatile unsigned int *)
(GPIOF_BASEADDR+0x04))
#define GPIOF_OSPEEDR    *((volatile unsigned int *)
(GPIOF_BASEADDR+0x08))
#define GPIOF_PUPDR      *((volatile unsigned int *)
(GPIOF_BASEADDR+0x0C))
#define GPIOF_ODR        *((volatile unsigned int *)
(GPIOF_BASEADDR+0x014))

```

代码编译后出现报错：



```

Build Output
Build started: Project: reg_demo
*** Using Compiler 'V6.18', folder: 'D:\KEIL\ARM\ARMCLANG\Bin'
Build target 'reg_demo'
linking...
.\Objects\reg_demo.sct(7): error: L6236E: No section matches selector - no section to be FIRST/LAST.
Not enough information to list image symbols.
Not enough information to list load addresses in the image map.
Finished: 2 information, 0 warning and 1 error messages.
".\Objects\reg_demo.axf" - 1 Error(s), 0 Warning(s).
Target not created.
Build Time Elapsed: 00:00:00

```

这下需要汇编代码。打开STM32F4xx_DSP_StdPeriph_Lib_V1.9.0 文件夹，打开路径：.\STM32F4xx_DSP_StdPeriph_Lib_V1.9.0\Libraries\CMSIS\Device\ST\STM32F4xx\Source\Templates\arm 在文件夹中找到启动汇编文件startup_stm32f40xx.s，

将其添加至project中，汇编文件会启动main函数与SystemInit函数。

```
Reset_Handler    PROC
                 EXPORT Reset_Handler            [WEAK]
                 IMPORT SystemInit
                 IMPORT __main

                 LDR    R0, =SystemInit
                 BLX    R0
                 LDR    R0, =__main
                 BX     R0
                 ENDP
```

顺便写个SystemInit 骗过编译器，最终代码如下：

```
#define RCC_BASEADDR    0x40023800
#define RCC_AHB1ENR    *((volatile unsigned int *)
(RCC_BASEADDR+0x30))

#define GPIOF_BASEADDR  0x40021400

#define GPIOF_MODER      *((volatile unsigned int *)
(GPIOF_BASEADDR+0x00))
#define GPIOF_OTYPER     *((volatile unsigned int *)
(GPIOF_BASEADDR+0x04))
#define GPIOF_OSPEEDR    *((volatile unsigned int *)
(GPIOF_BASEADDR+0x08))
#define GPIOF_PUPDR      *((volatile unsigned int *)
(GPIOF_BASEADDR+0x0C))
#define GPIOF_ODR        *((volatile unsigned int *)
(GPIOF_BASEADDR+0x014))

void SystemInit(void)
{

}

int main(void)
{
    RCC_AHB1ENR |= 1<<5;

    GPIOF_MODER &= ~(0x03<<2 * 9);
```

```

GPIOF_MODER |= (0x01<<2 * 9);

GPIOF_OTYPER &= ~(0x01<<9);

GPIOF_OSPEEDR &= ~(0x03<<2*9);
GPIOF_OSPEEDR |= (0x02<<2*9);

GPIOF_PUPDR &= ~(0x03<<2*9);
GPIOF_PUPDR |= (0x00<<2*9);

GPIOF_ODR &= ~(0x01<<9);
while(1)
{

}
}

```

编译成功。

没错，上面就是不依赖任何库，通过直接操作寄存器来实现的点亮第一个LED（话说翻看手册看得有点眼花，这是正常的，因为寄存器实在是太多了.....）

而且有没有发现，上面的代码基本上是对一堆数字（地址）进行操作，一旦离开手册，你又如何记得那些数字代表什么呢？纵使记忆力惊人，这全部记下来也是不现实的，那么接下来我们将它包装一下，让它看起来人性化一点：那就有下面所讲的标准库/HAL库.....

二、标准库开发环境配置及点灯

STM32标准库开发环境配置

过程略 ----- 上课培训再讲，在下面放一个标准库目录树

```

|demo
|├Doc
|├Libraries
| |├CMSIS
| | |├Device
| | | |├ST
| | | |├STM32F4xx
| | | |├Include

```



```
| | | | stm32f4xx.h
| | | | system_stm32f4xx.h
| | | |
| | | | └Source
| | | |     └Templates
| | | |         | system_stm32f4xx.c
| | | |         |
| | | |         └arm
| | | |             startup_stm32f40xx.s
| | | |
| | | | └Include
| | | |     arm_common_tables.h
| | | |     ...
| | | |     core_sc300.h
| | | |
| | | | └STM32F4xx_StdPeriph_Driver
| | | |     └inc
| | | |         | misc.h
| | | |         | stm32f4xx_adc.h
| | | |         | stm32f4xx_can.h
| | | |         | ...
| | | |         | stm32f4xx_tim.h
| | | |         | stm32f4xx_usart.h
| | | |         | stm32f4xx_wwdg.h
| | | |         |
| | | |         └src
| | | |             misc.c
| | | |             stm32f4xx_adc.c
| | | |             stm32f4xx_can.c
| | | |             ...
| | | |             stm32f4xx_usart.c
| | | |             stm32f4xx_wwdg.c
| | | |
| | | | └Listings
| | | | └Objects
| | | |     demo.hex
| | | |
| | | | └User
| | | |     └inc
| | | |         | stm32f4xx_conf.h
| | | |         | stm32f4xx_it.h
| | | |         |
| | | |         └src
```

```
main.c
stm32f4xx_it.c
```

使用标准库点灯

```
#include "stm32f4xx.h"
int main(void)
{
    //声明一个结构体变量
    GPIO_InitTypeDef GPIO_InitStructure;
    //使能GPIO外设时钟
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOF, ENABLE);
    //定义一个结构体
    GPIO_InitStructure.GPIO_Pin=GPIO_Pin_9 | GPIO_Pin_10; //连接
    LED的引脚
    GPIO_InitStructure.GPIO_Mode=GPIO_Mode_OUT; //输出模
    式
    GPIO_InitStructure.GPIO_Speed=GPIO_Speed_100MHz;
    //100MHz
    GPIO_InitStructure.GPIO_OType=GPIO_OType_PP; //推挽
    GPIO_InitStructure.GPIO_PuPd=GPIO_PuPd_UP; //上拉
    GPIO_Init(GPIOF, &GPIO_InitStructure);
    while(1)
    {
    }
}
```

三、STM32CubeMX HAL库方式点灯

务必熟练掌握

CubeMX配置教程

过程略 ----- 上课培训再讲



FYT