中南大学FYT战队 电控组培训讲义

主讲人：黄杰

学习要点

1. 初识stm32
2. 了解软件基础使用方法
3. 了解标准库与hal库
4. 熟练掌握GPIO的引脚模式
5. 了解外部中断

学前准备

- 安装keil与stm32cubemx，以及自己开发板对应固件库
- 购买开发板，下载器

学习内容

一、初识stm32

1、了解命名规则

STM32系列专为要求高性能、低成本、低功耗的嵌入式应用设计的**ARM Cortex®-M0，M0+，M3, M4和M7内核**，按内核架构分为不同产品：**主流产品**（STM32F0、STM32F1、STM32F3）、**超低功耗产品**（STM32L0、STM32L1、STM32L4、STM32L4+）、**高性能产品**（STM32F2、STM32F4、STM32F7、STM32H7）。

命名规则： STM32型号的说明：以STM32F103RBT6这个型号的芯片为例，该型号的组成为7个部分，其命名规则如下：

1	STM32	STM32代表ARM Cortex-M内核的32位微控制器。
2	F	F代表芯片子系列。
3	103	103代表增强型系列。
4	R	R这一项代表引脚数，其中T代表36脚，C代表48脚，R代表64脚，V代表100脚，Z代表144脚，I代表176脚。
5	B	B这一项代表内嵌Flash容量，其中6代表32K字节Flash，8代表64K字节Flash，B代表128K字节Flash，C代表256K字节Flash，D代表384K字节Flash，E代表512K字节Flash，G代表1M字节Flash。
6	T	T这一项代表封装，其中H代表BGA封装，T代表LQFP封装，U代表VFQFPN封装。
7	6	6这一项代表工作温度范围，其中6代表-40——85℃，7代表-40——105℃。

2、了解硬件电路设计

STM32的大体情况了解完了之后，那么就是了解如何设计一块STM32的板子，虽然说，市面上的成品的开发板五花八门，实际做项目用的核心板全为战队自行设计

不作复杂的要求，最基础的最小系统板，起码应该知道它的组成部分，这个也是我对大二和大三同学面试时考察过的问题

这里以51单片机为例：那就是 单片机、电源电路、时钟电路以及复位电路。

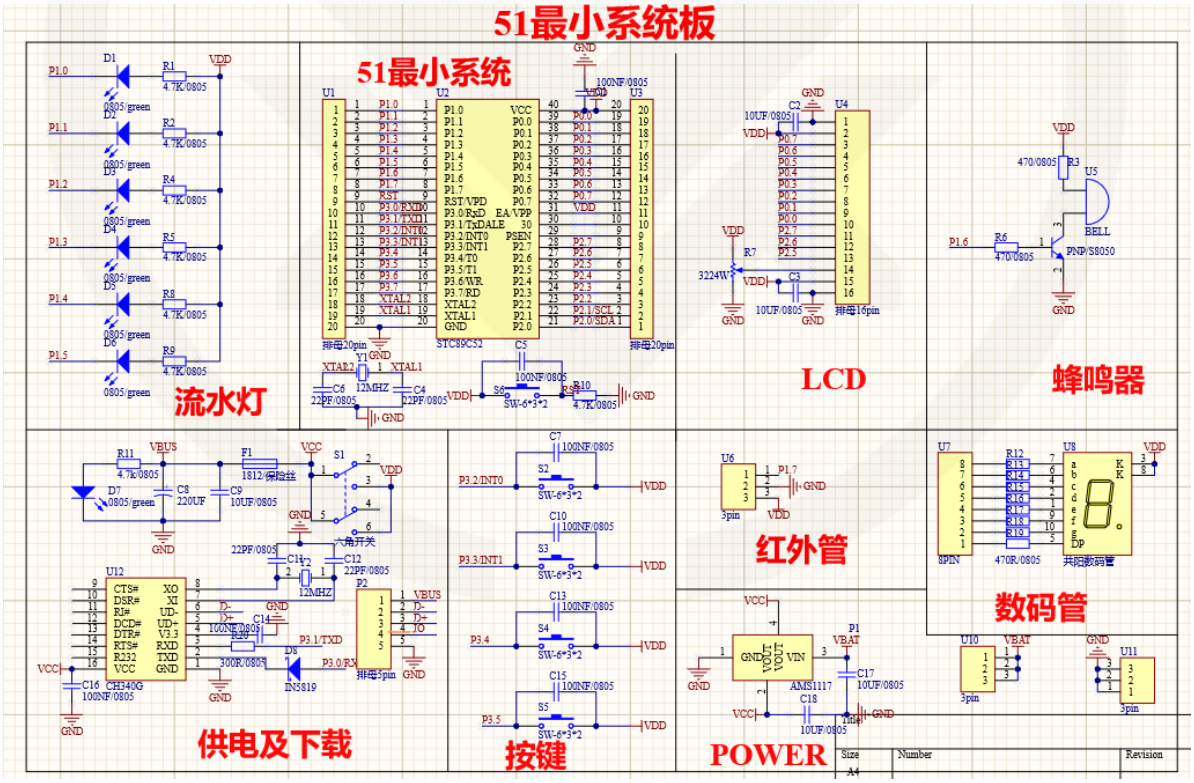
单片机是一种集成电路芯片。它采用超大规模技术将具有数据处理能力的微处理器（CPU）、存储器（含程序存储器ROM和数据存储器RAM）、输入、输出接口电路（I/O接口）集成在同一块芯片上，构成一个即小巧又很完善的计算机硬件系统，在单片机程序的控制下能准确、迅速、高效地完成程序设计者事先规定的任务。所以说，一片单片机芯片就具有了组成计算机的全部功能。

复位电路：由电容串联电阻构成，由图并结合“电容电压不能突变”的性质，可以知道，当系统一上电，RST脚将会出现高电平，并且，这个高电平持续的时间由电路的RC值来决定。典型的51单片机当RST脚的高电平持续两个机器周期以上就将复位，所以，适当组合RC的取值就可以保证可靠的复位。

时钟电路：单片机外部接上振荡器（也可以是内部振荡器）提供高频脉冲经过分频处理后，成为单片机内部时钟信号，作为片内各部件协调工作的控制信号。作用是来配合外部晶体实现振荡的电路，这样可以为单片机提供运行时钟。以MCS—5l单片机为例随明：MCS—51单片机为12个时钟周期执行一条指令。也就是说单片机运行一条指令，必须要用12个时钟周期。没有这个时钟，单片机就跑不起来了，也没有办法定时和进行和时间有关的操作。 时钟电路是微型计算机的心脏，它控制着计算机的二个节奏。CPU就是通过复杂的时序电路完成不同的指令功能的。

电源电路：用于给单片机供电，一个稳定的电源是单片机正常工作的前提，因此，单片机附近经常会有电容用于滤波。

下面时电子科技制作时战队设计并辅以教学的51板原理图



3、stm32编程

STM32单片机的成功，和ST的软件生态也是有很大关系的。 早期的STM32编程主要以库函数（标准固件库）和寄存器两种方式进行的。库函数编程比较方便，调用ST官方提供的函数，即可完成相应功能，效率也还可以接受。寄存器方式则是直接操作STM32的相应寄存器的数据，效率极高，但是难度也很大，编程时需要查阅芯片参考手册，多用于嵌入式老手。但随着单片机性能的不断提升，对于效率又不太看重，所以，很多人基本就是选择库函数了。

后来，ST为了降低编程的门槛，推出了HAL库，Hardware Abstraction Layer（硬件抽象层），说白了就是减少硬件细节，提高移植性，降低编程门槛。又配合STM32CUBEMX软件，直接就可以初始化一个工程。大大降低编程难度，本次教程就是介绍：HAL库配合CUBEMX配置一些常用外设的初始化，直观感受STM32编程，用最短时间入门STM32

当然了，HAL库虽然简单直观，但我们也不能忽视硬件底层的学习，不能不求甚解。

二、GPIO介绍

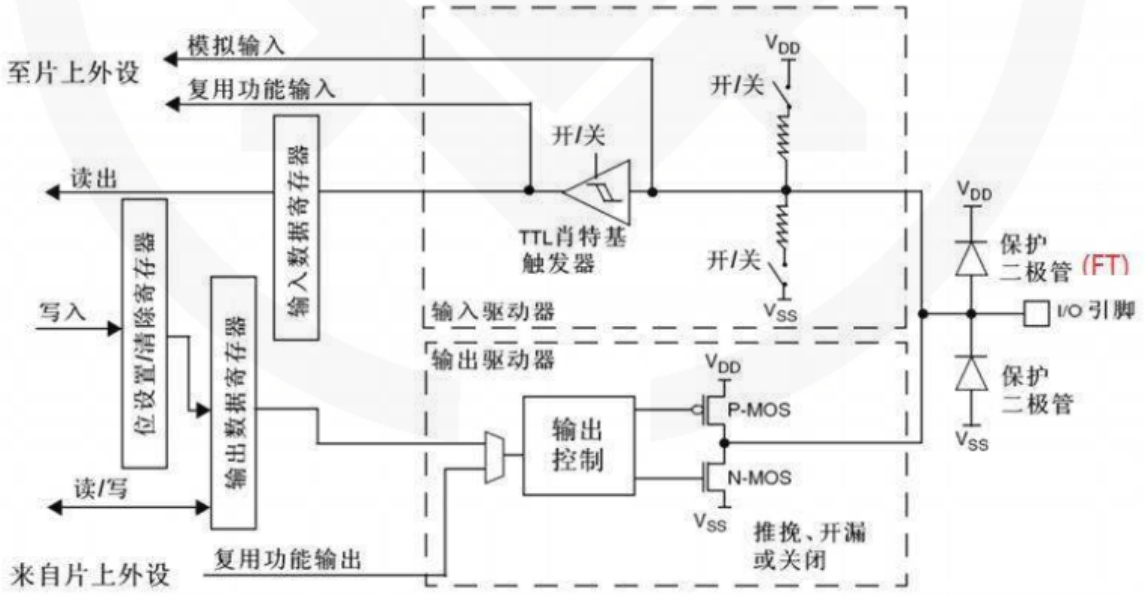
GPIO是通用输入/输出端口的简称，是STM32可控制的引脚。GPIO的引脚与外部硬件设备连接，可实现与外部通讯、控制外部硬件或者采集外部硬件数据的功能。

STM32F103ZET6芯片为144脚芯片，包括7个通用目的的输入/输出口（GPIO）组，分别为GPIOA、GPIOB、GPIOC、GPIOD、GPIOE、GPIOF、GPIOG，同时每组GPIO口组有16个GPIO口。通常简略称为PAx、PBx、PCx、PDx、PEx、PFx、PGx，其中x为0-15。

STM32的大部分引脚除了当GPIO使用之外，还可以复用位外设功能引脚（比如串口），这部分在【STM32】STM32端口复用和重映射（AFIO辅助功能时钟） 中有详细的介绍

GPIO基本结构

每个GPIO内部都有这样的一个电路结构，这个结构在本文下面会具体介绍。



这边的电路图稍微提一下：

保护二极管：IO引脚上下两边两个二极管用于防止引脚外部过高、过低的电压输入。当引脚电压高于VDD时，上方的二极管导通；当引脚电压低于VSS时，下方的二极管导通，防止不正常电压引入芯片导致芯片烧毁。但是尽管如此，还是不能直接外接大功率器件，须加大功率及隔离电路驱动，防止烧坏芯片或者外接器件无法正常工作。

P-MOS管和N-MOS管：由P-MOS管和N-MOS管组成的单元电路使得GPIO具有“推挽输出”和“开漏输出”的模式。这里的电路会在下面很详细地分析到。

TTL肖特基触发器：信号经过触发器后，模拟信号转化为0和1的数字信号。但是，当GPIO引脚作为ADC采集电压的输入通道时，用其“模拟输入”功能，此时信号不再经过触发器进行TTL电平转换。ADC外设要采集到的原始的模拟信号。

STM32的GPIO工作方式

GPIO支持4种输入模式（浮空输入、上拉输入、下拉输入、模拟输入）和4种输出模式（开漏输出、开漏复用输出、推挽输出、推挽复用输出）。同时，GPIO还支持三种最大翻转速度（2MHz、10MHz、50MHz）。

每个I/O口可以自由编程，但I/O口寄存器必须按32位字被访问。

GPIO_Mode_AIN 模拟输入

GPIO_Mode_IN_FLOATING 浮空输入

GPIO_Mode_IPD 下拉输入

GPIO_Mode_IPU 上拉输入

GPIO_Mode_Out_OD 开漏输出

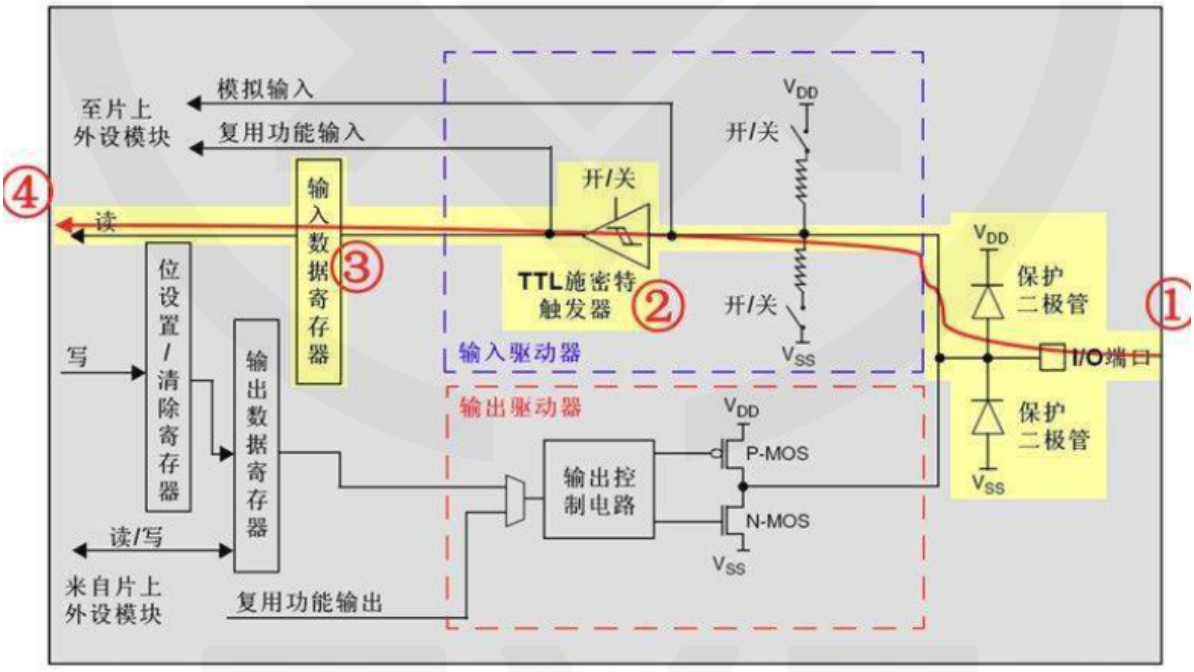
GPIO_Mode_Out_PP 推挽输出

GPIO_Mode_AF_OD 复用开漏输出

GPIO_Mode_AF_PP 复用推挽输出

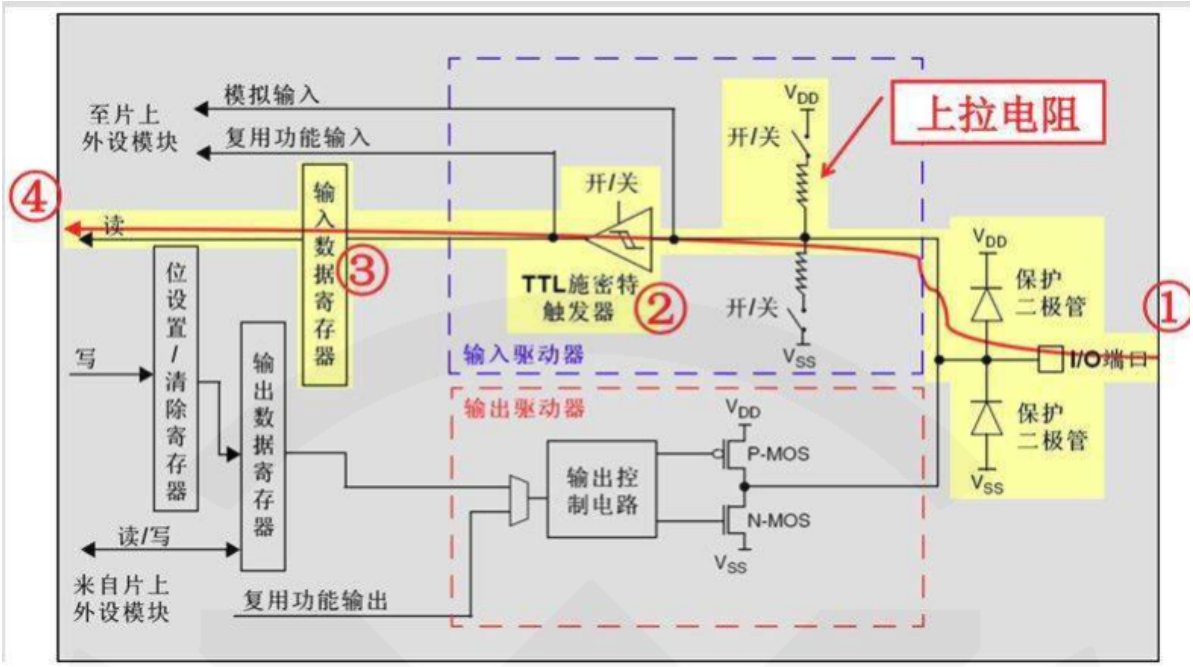
下面将具体介绍GPIO的这八种工作方式：

浮空输入模式



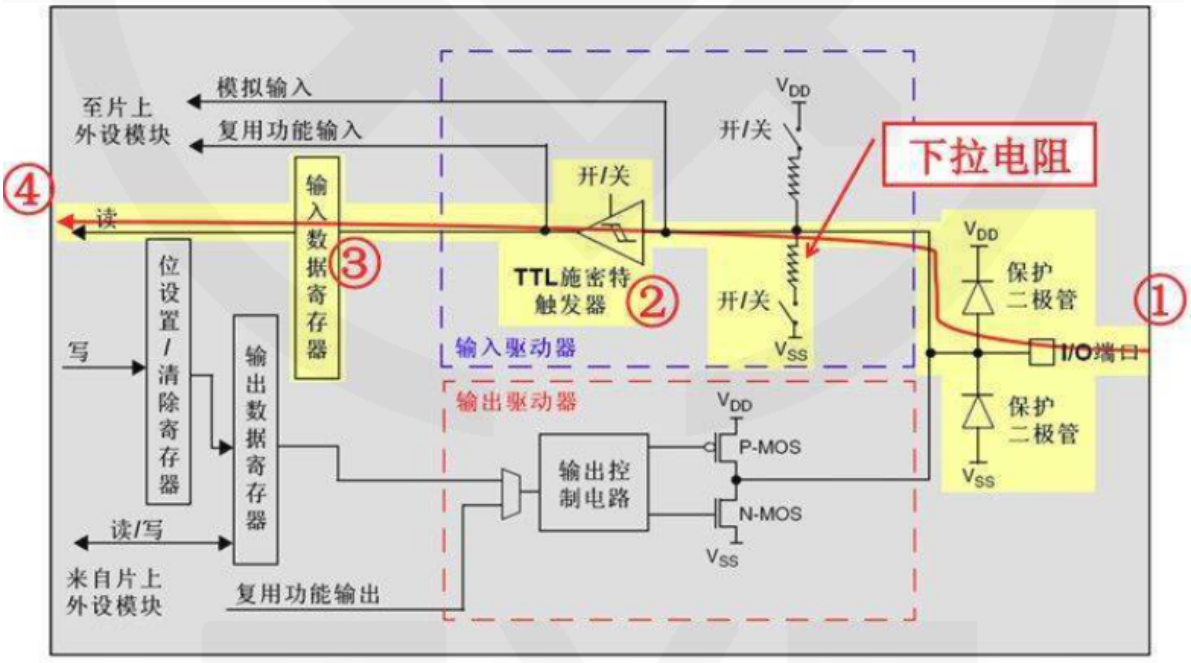
浮空输入模式下，I/O端口的电平信号直接进入输入数据寄存器。也就是说，I/O的电平状态是不确定的，完全由外部输入决定；如果在该引脚悬空（在无信号输入）的情况下，读取该端口的电平是不确定的。

上拉输入模式



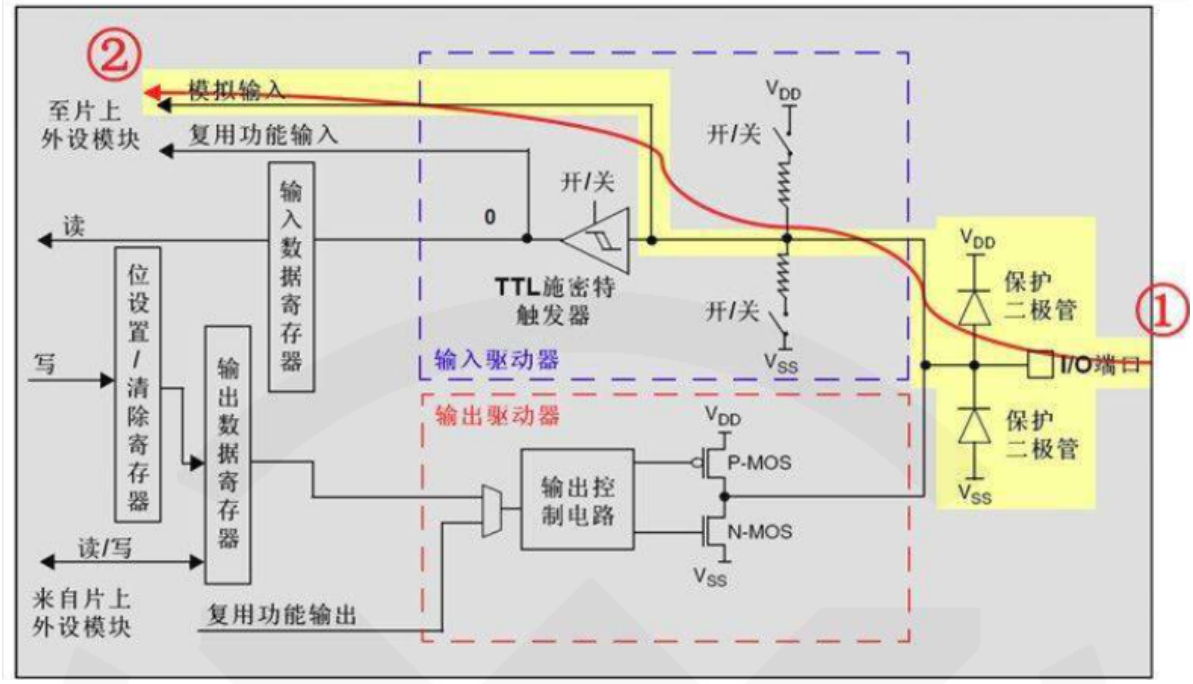
上拉输入模式下，I/O端口的电平信号直接进入输入数据寄存器。但是在I/O端口悬空（在无信号输入）的情况下，输入端的电平可以保持在高电平；并且在I/O端口输入为低电平的时候，输入端的电平也还是低电平。

下拉输入模式



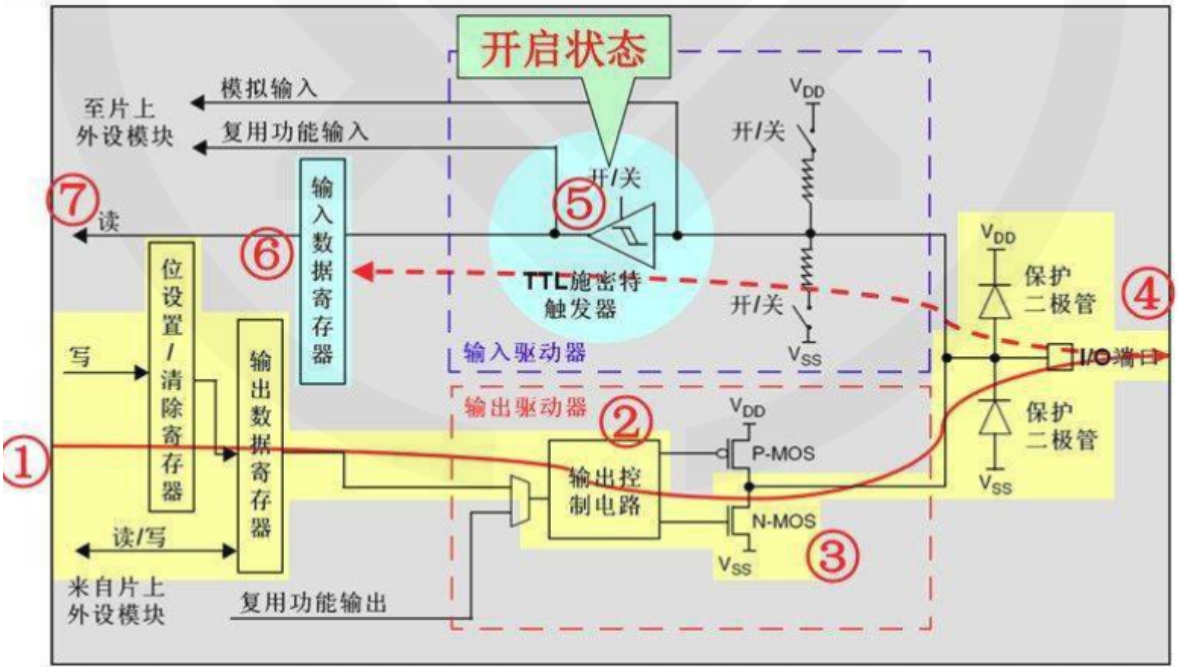
下拉输入模式下，I/O端口的电平信号直接进入输入数据寄存器。但是在I/O端口悬空（在无信号输入）的情况下，输入端的电平可以保持在低电平；并且在I/O端口输入为高电平的时候，输入端的电平也还是高电平。

模拟输入模式



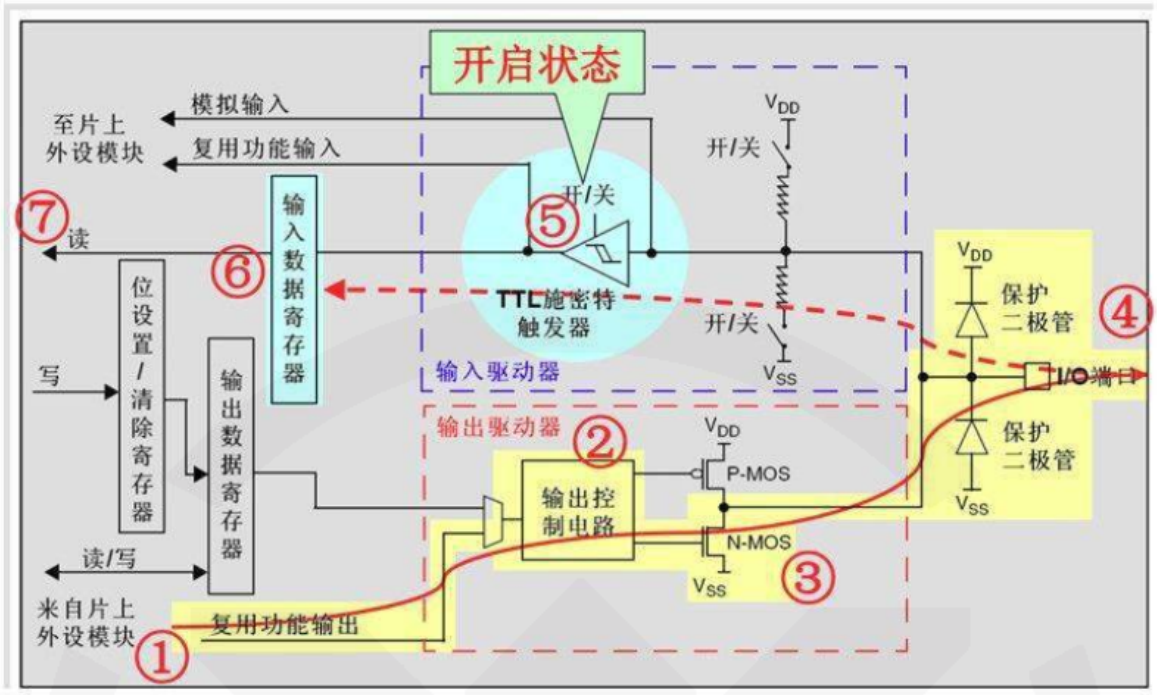
模拟输入模式下，I/O端口的模拟信号（电压信号，而非电平信号）直接模拟输入到片上外设模块，比如ADC模块等等。

开漏输出模式



开漏输出模式下，通过设置位设置/清除寄存器或者输出数据寄存器的值，途经N-MOS管，最终输出到I/O端口。这里要注意N-MOS管，当设置输出的值为高电平的时候，N-MOS管处于关闭状态，此时I/O端口的电平就不会由输出的高低电平决定，而是由I/O端口外部的上拉或者下拉决定；当设置输出的值为低电平的时候，N-MOS管处于开启状态，此时I/O端口的电平就是低电平。同时，I/O端口的电平也可以通过输入电路进行读取；注意，I/O端口的电平不一定是输出的电平。

开漏复用输出模式



开漏复用输出模式，与开漏输出模式很是类似。只是输出的高低电平的来源，不是让CPU直接写输出数据寄存器，取而代之利用片上外设模块的复用功能输出来决定的。

GPIO代码实现

1、标准库

初始化函数

```
void GPIO_Init(GPIO_TypeDef* GPIOx, GPIO_InitTypeDef* GPIO_InitStruct);
```

2个读取输入电平函数：

```
uint8_t GPIO_ReadInputDataBit(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);

uint16_t  GPIO_ReadInputData(GPIO_TypeDef* GPIOx);
```

2个读取输出电平函数：

```
uint8_t  GPIO_ReadOutputDataBit(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);

uint16_t  GPIO_ReadOutputData(GPIO_TypeDef* GPIOx);
```

4个设置输出电平函数：

```
void GPIO_SetBits(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);

void GPIO_ResetBits(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);

void GPIO_WriteBit(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin, BitAction BitVal);

void GPIO_Write(GPIO_TypeDef* GPIOx, uint16_t PortVal);
```

2、hal库

设置或清除选定的数据端口位

```
void HAL_GPIO_WritePin(GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin, GPIO_PinState
PinState)
```

设置或清除选定的数据端口位

```
void HAL_GPIO_TogglePin(GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin)
```

读取指定的输入端口引脚

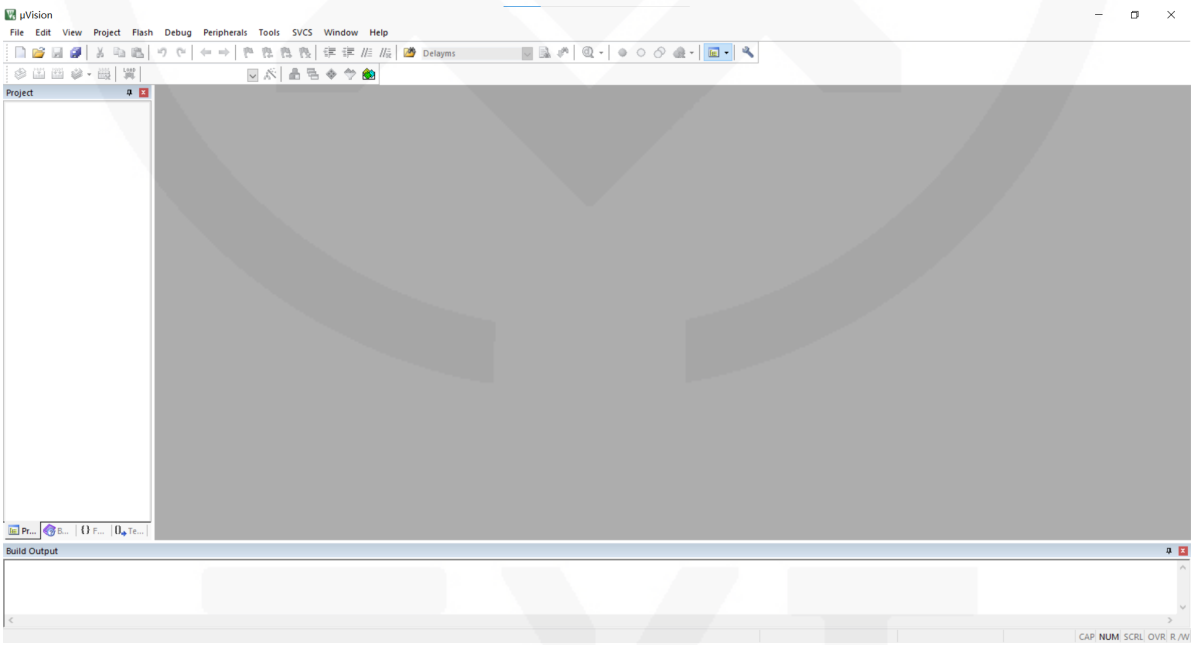
```
GPIO_PinState HAL_GPIO_ReadPin(GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin)
```

三、keil 简单介绍

Keil 提供了包括 C编译器、宏汇编、链接器、库管理和一个功能强大的仿真调试器等在内的完整开发方案，通过一个集成开发环境（μVision）将这些部分组合在一起。

在此说明，在这个阶段，我不太推荐大家去使用clion来搞咱们培训

下图是我的keil截图



四、EXTI介绍

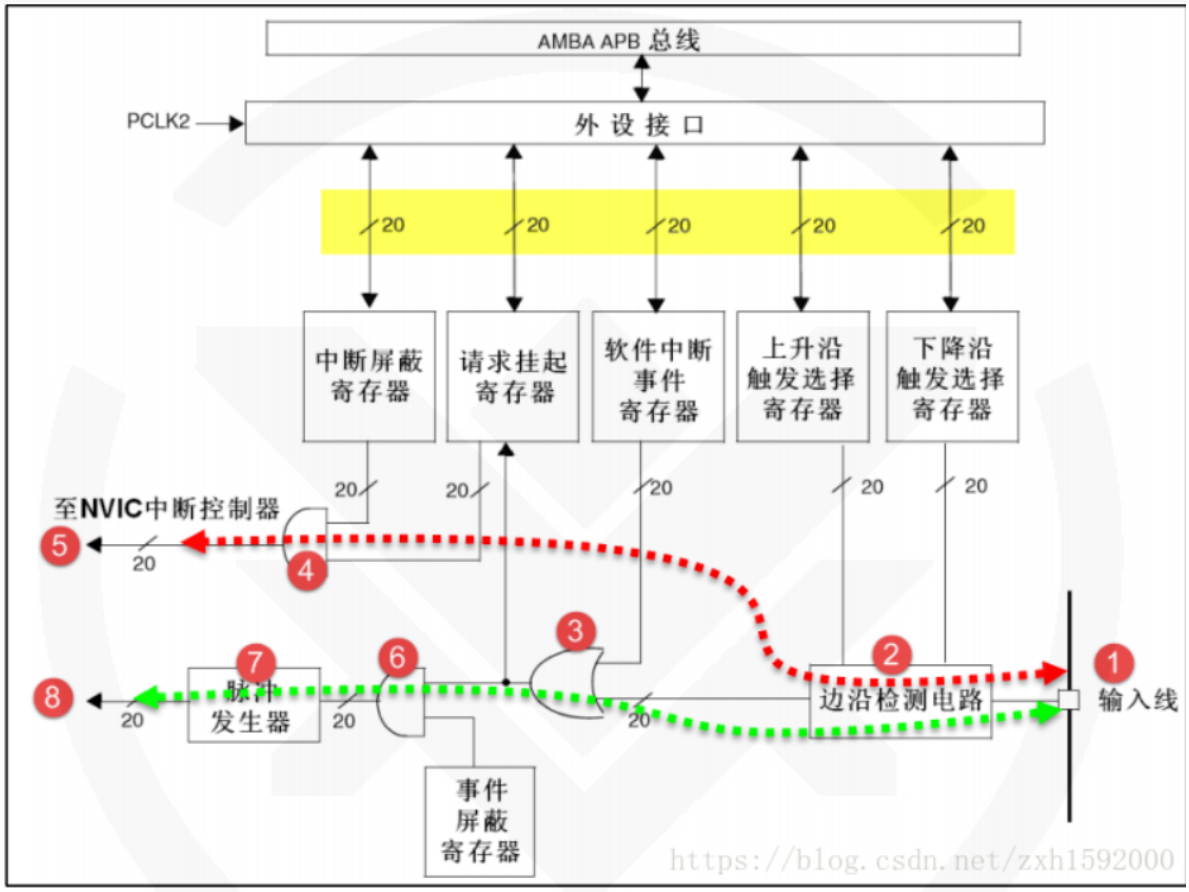
EXTI 简介

EXTI（External interrupt/event controller）—外部中断/事件控制器，管理了控制器的 20个中断/事件线。每个中断/事件线都对应有一个边沿检测器，可以实现输入信号的上升沿检测和下降沿的检测。EXTI可以实现对每个中断/事件线进行单独配置，可以单独配置为中断或者事件，以及触发事件的属性。

EXTI 功能框图

EXTI 的功能框图包含了 EXTI 最核心内容，掌握了功能框图，对 EXTI 就有一个整体的把握，在编程时思路就非常清晰。EXTI功能框图见图。

在图可以看到很多在信号线上打一个斜杠并标注“20”字样，这个表示在控制器内部类似的信号线路有 20 个，这与 EXTI 总共有 20 个中断/事件线是吻合的。所以我们只要明白其中一个的原理，那其他 19 个线路原理也就知道了



EXTI 可分为两大部分功能，一个是产生中断，另一个是产生事件，这两个功能从硬件上就有所不同。

首先我们来看图中红色虚线指示的电路流程。它是一个产生中断的线路，最终信号流入到 NVIC 控制器内。

编号 1 是输入线，EXTI 控制器有 19 个中断/事件输入线，这些输入线可以通过寄存器设置为任意一个 GPIO，也可以是一些外设的事件，这部分内容我们将在后面专门讲解。输入线一般是存在电平变化的信号。

编号 2 是一个边沿检测电路，它会根据上升沿触发选择寄存(EXTI_RTSR)和下降沿触发选择寄存器(EXTI_FTSR)对应位的设置来控制信号触发。边沿检测电路以输入线作为信号输入端，如果检测到有边沿跳变就输出有效信号 1 给编号 3 电路，否则输出无效信号0。而 EXTI_RTSR 和 EXTI_FTSR 两个寄存器可以控制器需要检测哪些类型的电平跳变过程，可以是只有上升沿触发、只有下降沿触发或者上升沿和下降沿都触发。

编号 3 电路实际就是一个或门电路，它一个输入来自编号 2 电路，另外一个输入来自软件中断事件寄存器(EXTI_SWIER)。EXTI_SWIER允许我们通过程序控制就可以启动中断/事件线，这在某些地方非常有用。我们知道或门的作用就是有 1 就为 1，所以这两个输入随便一个有有效信号 1就可以输出 1 给编号 4和编号 6电路。

编号 4 电路是一个与门电路，它一个输入是编号 3 电路，另外一个输入来自中断屏蔽寄存器(EXTI_IMR)。与门电路要求输入都为 1 才输出 1，导致的结果是如果 EXTI_IMR 设置为 0 时，那不管编号 3 电路的输出信号是 1 还是 0，最终编号 4 电路输出的信号都为 0；

如果EXTI_IMR设置为1时，最终编号4电路输出的信号才由编号3电路的输出信号决定，这样我们可以简单的控制 EXTI_IMR 来实现是否产生中断的目的。编号 4 电路输出的信号会被保存到挂起寄存器(EXTI_PR)内，如果确定编号 4 电路输出为 1 就会把 EXTI_PR 对应位置 1。

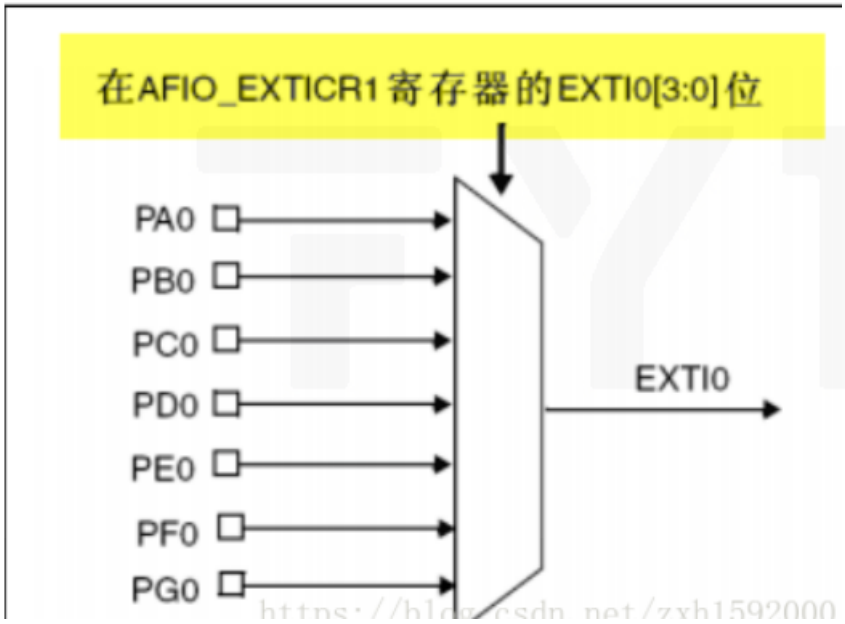
编号 5 是将 EXTI_PR 寄存器内容输出到 NVIC 内，从而实现系统中断事件控制。
接下来我们来看看绿色虚线指示的电路流程。它是一个产生事件的线路，最终输出一个脉冲信号。
产生事件线路是在编号3电路之后与中断线路有所不同，之前电路都是共用的。
编号6电路是一个与门，它一个输入来自编号 3 电路，另外一个输入来自事件屏蔽寄存器(EXTI_EMR)。
如果 EXTI_EMR设置为 0时，那不管编号 3电路的输出信号是 1还是 0，最终编号 6 电路输出的信号都为 0；如果 EXTI_EMR 设置为 1 时，最终编号 6 电路输出的信号才由编号 3 电路的输出信号决定，这样我们可以简单的控制 EXTI_EMR 来实现是否产生
事件的目的。
编号 7 是一个脉冲发生器电路，当它的输入端，即编号 6 电路的输出端，是一个有效信号 1 时就会产生一个脉冲；如果输入端是无效信号就不会输出脉冲。
编号 8 是一个脉冲信号，就是产生事件的线路最终的产物，这个脉冲信号可以给其他外设电路使用，比如定时器 TIM、模拟数字转换器 ADC等等，这样的脉冲信号一般用来触发 TIM 或者 ADC开始转换。
产生中断线路目的是把输入信号输入到 NVIC，进一步会运行中断服务函数，实现功能，这样是软件级的。而产生事件线路目的就是传输一个脉冲信号给其他外设使用，并且是电路级别的信号传输，属于硬件级的。
另外，EXTI是在 APB2总线上的，在编程时候需要注意到这点。

中断/事件线

EXTI有 20个中断/事件线，每个 GPIO都可以被设置为输入线，占用 EXTI0至 EXTI15，还有另外七根用于特定的外设事件。
4根特定外设中断/事件线由外设触发，具体用法参考《STM32F10X-中文参考手册》中对外设的具体说明。

中断/事件线	输入源
EXTI0	PX0(X 可为 A, B, C, D, E, F, G, H, I)
EXTI1	PX1(X 可为 A, B, C, D, E, F, G, H, I)
EXTI2	PX2(X 可为 A, B, C, D, E, F, G, H, I)
EXTI3	PX3(X 可为 A, B, C, D, E, F, G, H, I)
EXTI4	PX4(X 可为 A, B, C, D, E, F, G, H, I)
EXTI5	PX5(X 可为 A, B, C, D, E, F, G, H, I)
EXTI6	PX6(X 可为 A, B, C, D, E, F, G, H, I)
EXTI7	PX7(X 可为 A, B, C, D, E, F, G, H, I)
EXTI8	PX8(X 可为 A, B, C, D, E, F, G, H, I)
EXTI9	PX9(X 可为 A, B, C, D, E, F, G, H, I)
EXTI10	PX10(X 可为 A, B, C, D, E, F, G, H, I)
EXTI11	PX11(X 可为 A, B, C, D, E, F, G, H, I)
EXTI12	PX12(X 可为 A, B, C, D, E, F, G, H, I)
EXTI13	PX13(X 可为 A, B, C, D, E, F, G, H, I)
EXTI14	PX14(X 可为 A, B, C, D, E, F, G, H, I)
EXTI15	PX15(X 可为 A, B, C, D, E, F, G, H, I)
EXTI16	PVD 输出
EXTI17	RTC 闹钟事件
EXTI18	USB 唤醒事件
EXTI19	以太网唤醒事件（只适用互联型）

EXTI0至 EXTI15用于 GPIO，通过编程控制可以实现任意一个 GPIO作为 EXTI的输入源。由表可知，EXTI0 可以通过 AFIO 的外部中断配置寄存器 1(AFIO_EXTICR1)的EXTI0[3:0]位选择配置为 PA0、PB0、PC0、PD0、PE0、PF0、PG0、PH0 或者 PI0，其他 EXTI线(EXTI中断/事件线)使用配置都是类似的。



EXTI代码实现

中断回调函数

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin);
```

其他

关于按键防抖的问题：

- 软件防抖可以检测到电平延时一段时间再确认电平，延时时间一般为10-20ms
- 硬件防抖可以在按键上并联一个电容，一般为0.1uf

课后作业

- 通过按键轮询实现灯效加减
- 通过按键中断实现灯效模式切换

参考文献

STM32基础入门（一）——STM32概览 - 知乎 (zhihu.com)

GPIO口工作原理的超详细解释（附电路图） 输出 (sohu.com)

(31条消息) STM32系统学习——EXTI（外部中断）_Yuk丶的博客-CSDN博客exti中断

(31条消息) STM32-GPIO介绍_KevinFlyn的博客-CSDN博客stm32gpio是什么