

第八章 定时器

主讲人：漆强

电子科技大学

ytqiqiang@163.com

本章内容



定时器概述



HAL库外设模块的设计方法



定时器的定时/计数功能



定时器的PWM输出功能



定时器的输入捕获功能

教学目标



了解定时器的功能和特点



了解HAL库的外设模块设计方法



熟练掌握定时器的定时/计数功能



熟练掌握定时器的PWM输出及输入捕获功能



电子科技大学
University of Electronic Science and Technology of China

8.1 定时器概述

1 基本概念

基本概念

定时器和计数器的区别

- ❑ **定时器**是对周期固定的脉冲信号进行计数，
如MCU内部的外设时钟(APB)。

周期固定或者不固定
- ❑ **计数器**是对**周期不确定**的脉冲信号进行计数，
如MCU的I/O引脚所引入的外部脉冲信号。
- ❑ **结论：**定时器和计数器本质上都是**计数器**，
定时器是计数器的一种特例。



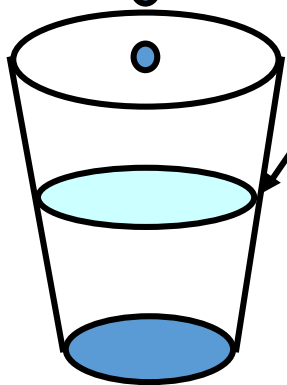
滴水实例

定时器和计数器的区别

假如有一个水容器

1000滴水刚好装满

计数终值



初始时已经装入500滴水

计数初值

计数的概念

问：还需滴入多少滴水才能将其装满？

答：还需滴入500滴水才能将其装满，第501滴水时容器溢出。

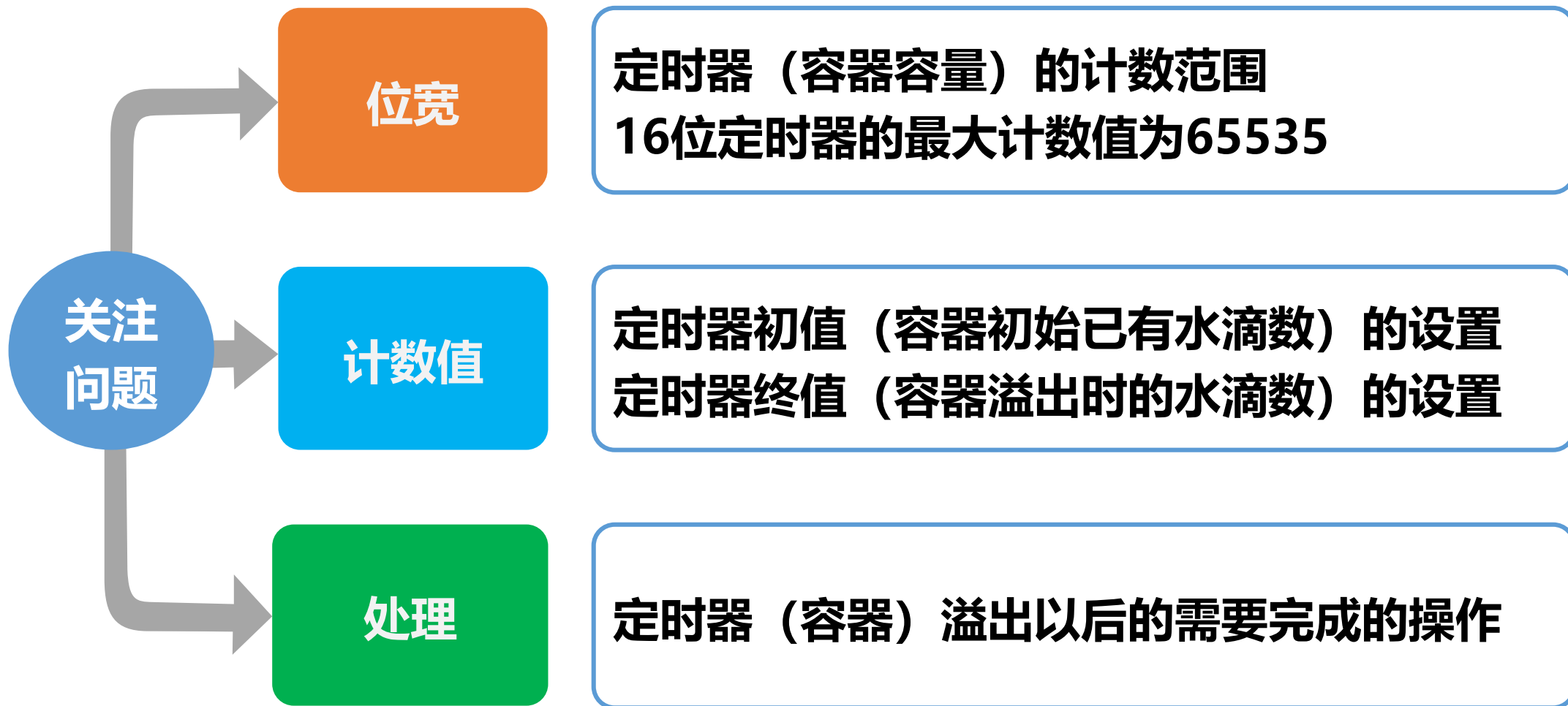
问：如果每秒滴入1滴水，还需多长时间才能将其装满？

答：还需500秒时间才能将其装满，第501秒时容器溢出。

定时的概念

关注问题

定时器使用所关注的三个问题



两个概念

定时器模式的两个概念

时钟频率

在定时器模式下，送入定时器的周期性时钟信号的频率

以STM32F411微控制器为例，送入定时器的时钟频率为100M

计数时间

在定时器模式下，计数单元记一次数所花费的时间，它是时钟频率的倒数

假设计数单元在1s内计数1000000次，则计数时间为1us

定时时间计算公式

$$\text{定时时间} = \text{计数值} * \text{计数时间}$$

$$\text{定时时间} = \text{计数值} / \text{时钟频率}$$

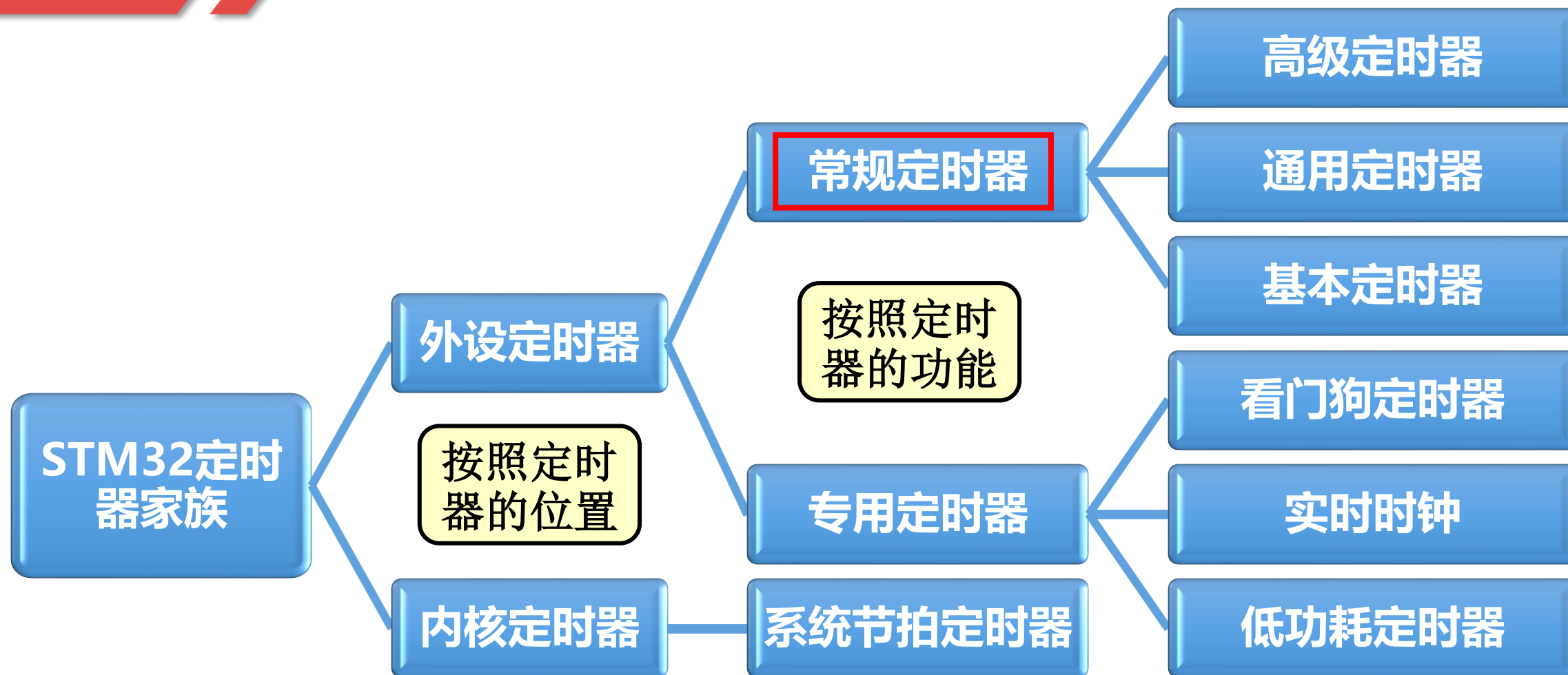


电子科技大学
University of Electronic Science and Technology of China

2 STM32微控制器的定时器概述

分类

STM32定时器家族



常规定时器

常规定时器分类

基本定时器

几乎没有任何输入/输出通道，常用作时基，实现基本的定时/计数功能

通用定时器

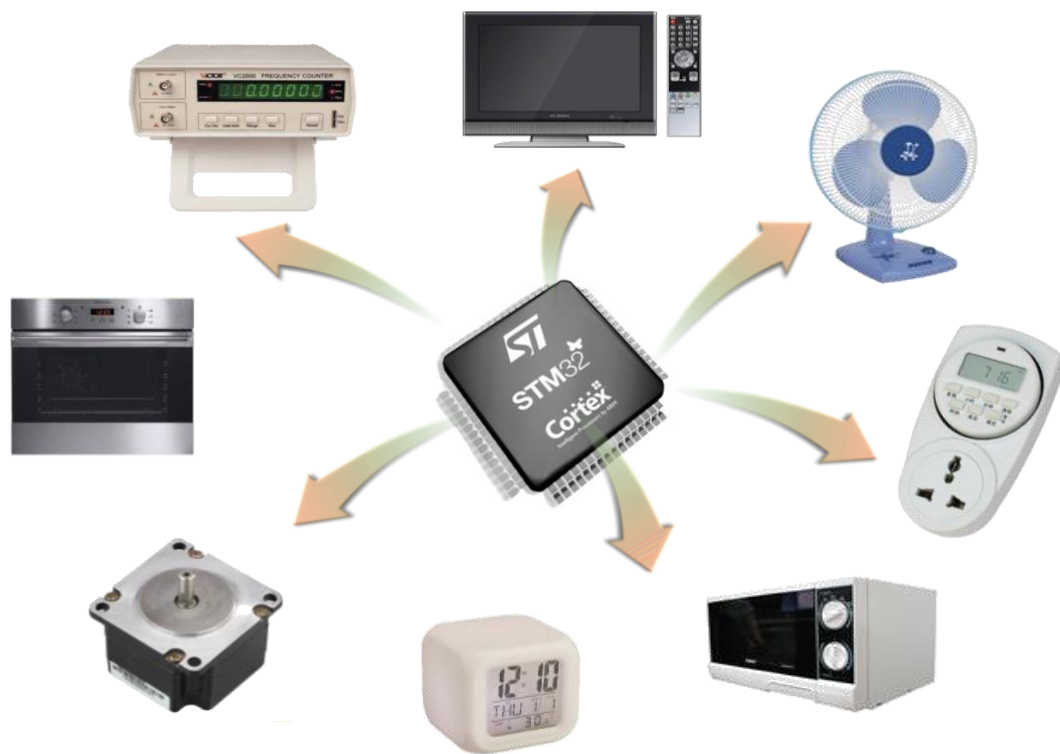
具备多路独立的捕获和比较通道，可以完成定时/计数、输入捕获、输出比较等功能

高级定时器

除具备通用定时器的功能外，还具备带死区控制的互补信号输出、紧急刹车关断输入等功能，可用于电机控制和数字电源设计

STM32
F411

- ◆ 1 个 16 位 高级定时器
- ◆ 2 个 32 位 通用定时器
- ◆ 5 个 16 位 通用定时器



应用:

- 数字频率计
- 智能家用电器
- 定时控制设备
- 电机驱动

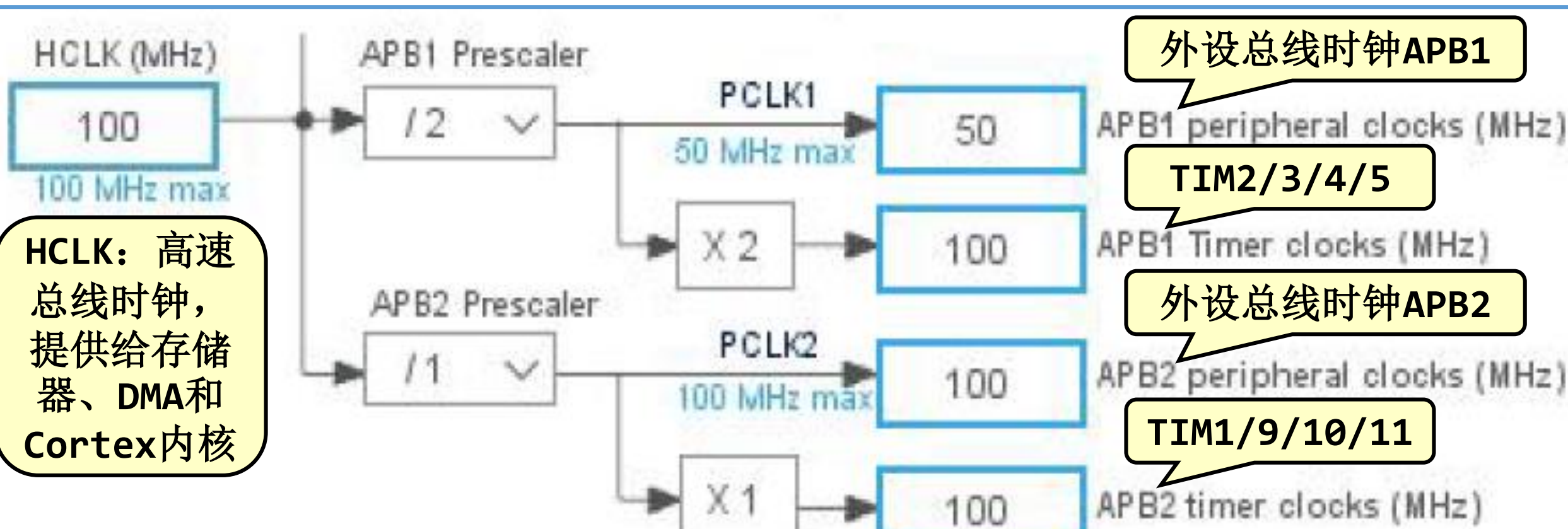
详细特性

STM32F411定时器的详细特性

类型	名称	位数	计数方式	预分频系数	产生DMA请求	捕获/比较通道	互补输出	时钟频率	挂接总线
高级定时器	TIM1	16	递增/递减/中心对齐	1 ~ 65536	是	4	支持	100M	APB2
通用定时器	TIM2/ TIM5	32	递增/递减/中心对齐	1 ~ 65536	是	4	不支持	100M	APB1
	TIM3/ TIM4	16	递增/递减/中心对齐	1 ~ 65536	是	4	不支持	100M	APB1
	TIM9	16	递增	1 ~ 65536	否	2	不支持	100M	APB2
	TIM10/ TIM11	16	递增	1 ~ 65536	否	1	不支持	100M	APB2

时钟频率

定时器的时钟频率



注意：外设总线时钟和定时器时钟并不完全一致，APB1总线时钟为50M，APB2总线时钟为100M，而这两个外设总线所挂接的定时器时钟均为100M。

主要功能

定时器

定时计数

计数内部时钟，即定时器模式
计数外部脉冲，即计数器模式

输出比较

PWM输出
电平翻转
单脉冲输出
强制输出

输入捕获

捕获时保存定时器的当前计数值
捕获时，可选择触发捕获中断
触发捕获的信号边沿类型可选择
(上升沿，下降沿，双边沿)



电子科技大学
University of Electronic Science and Technology of China

8.2 HAL库外设模块设计方法

简单外设的设计方法

简单外设

GPIO外设使用引脚初始化数据类型GPIO_InitTypeDef来描述GPIO引脚的属性：引脚编号、工作模式、输出速度等

复杂外设

定时器外设具有三类功能：

- 定时/计数功能
- 输出比较功能
- 输入捕获功能

每一类功能都需要单独的初始化数据类型

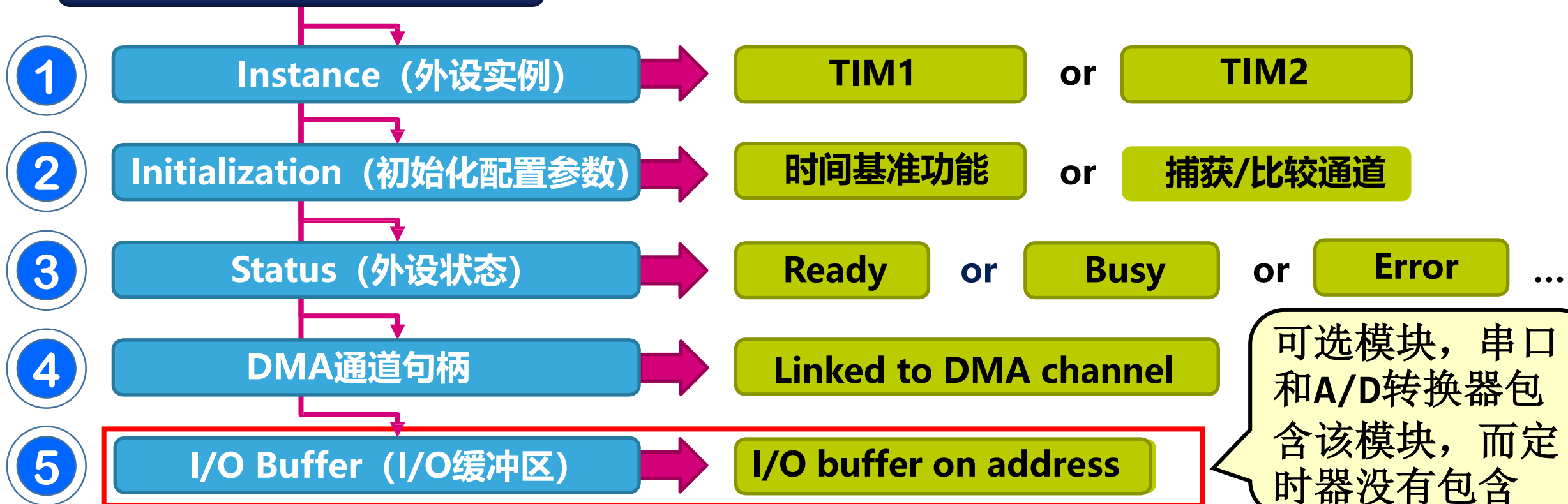
三种基本设计方法



外设句柄

外设句柄数据类型的组成

Handle structure



可选模块，串口和A/D转换器包含该模块，而定时器没有包含

定时器句柄

定时器句柄结构的组成

```
1. typedef struct
2. {
3.     TIM_TypeDef          *Instance;           // 定时器寄存器的基地址定义
4.     TIM_Base_InitTypeDef  Init;               // 时基功能的配置参数定义
5.     HAL_TIM_ActiveChannel Channel;            // 捕获/比较通道的定义
6.     DMA_HandleTypeDef     *hdma[7];           // DMA通道句柄定义
7.     HAL_LockTypeDef        Lock;              // 保护锁类型定义
8.     __IO HAL_TIM_StateTypeDef State;          // 定时器运行状态定义
9. } TIM_HandleTypeDef;
```

定时器实例，如TIM1和TIM2等

定时器句柄

定时器句柄结构的组成

```
1. typedef struct
2. {
3.     TIM_TypeDef                *Instance; // 定时器寄存器的基地址定义
4.     TIM_Base_InitTypeDef      Init;      // 时基功能的配置参数定义
5.     HAL_TIM_ActiveChannel      Channel;  // 捕获/比较通道的定义
6.     DMA_HandleTypeDef         *hdma[7];  // DMA通道句柄定义
7.     HAL_LockTypeDef           Lock;      // 保护锁类型定义
8.     __IO HAL_TIM_StateTypeDef State;    // 定时器运行状态定义
9. } TIM_HandleTypeDef;
```

定时器的初始化配置参数

定时器句柄

定时器句柄结构的组成

```
1. typedef struct
2. {
3.     TIM_TypeDef                *Instance; // 定时器寄存器的基地址定义
4.     TIM_Base_InitTypeDef      Init;      // 时基功能的配置参数定义
5.     HAL_TIM_ActiveChannel      Channel;   // 捕获/比较通道的定义
6.     DMA_HandleTypeDef          *hdma[7];  // DMA通道句柄定义
7.     HAL_LockTypeDef            Lock;       // 保护锁类型定义
8.     __IO HAL_TIM_StateTypeDef  State;     // 定时器运行状态定义
9. } TIM_HandleTypeDef;
```

DMA通道句柄

定时器句柄

定时器句柄结构的组成

```
1. typedef struct
2. {
3.     TIM_TypeDef          *Instance;      // 定时器寄存器的基地址定义
4.     TIM_Base_InitTypeDef  Init;          // 时基功能的配置参数定义
5.     HAL_TIM_ActiveChannel Channel;
6.     DMA_HandleTypeDef     *hdma[7];
7.     HAL_LockTypeDef       Lock;          // 保护锁类型定义
8.     __IO HAL_TIM_StateTypeDef State;     // 定时器运行状态定义
9. } TIM_HandleTypeDef;
```

保护锁是HAL库中提供的一种安全机制，以避免对外设的并发访问

外设状态

外设编程模型

三种外设编程模型

特点1: 以后缀区分编程模型

特点2: 入口参数均为外设句柄的指针

编程
模型

轮询方式

```
HAL_TIM_Base_Start(TIM_HandleTypeDef *htim);  
HAL_TIM_Base_Stop(TIM_HandleTypeDef *htim);
```

中断方式

```
HAL_TIM_Base_Start_IT(TIM_HandleTypeDef *htim);  
HAL_TIM_Base_Stop_IT(TIM_HandleTypeDef *htim);
```

DMA方式

```
HAL_TIM_Base_Start_DMA(TIM_HandleTypeDef  
*htim, uint32_t *pData, uint16_t Length);
```

四种类型的通用接口函数

初始化函数

根据用户配置参数完成外设的初始化操作

I/O操作函数

与外设进行数据交互，包括轮询、中断和DMA三种编程模型

控制函数

动态配置外设参数

状态函数

用于获取外设的运行状态以及出错信息

扩展接口函数的设计

设计目的

- ① 兼顾STM32各产品系列的特有功能和扩展性能
- ② 兼顾同一个产品系列中不同芯片的特有功能

`stm32f4xx_hal_tim.c`

设计方法

单独定义后续为ex的文件
`stm32fxxx_hal_ppp_ex.c`
`stm32fxxx_hal_ppp_ex.h`

`stm32f4xx_hal_tim_ex.c`



电子科技大学
University of Electronic Science and Technology of China

8.3 定时器的定时/计数功能



电子科技大学
University of Electronic Science and Technology of China

1 时基单元

Timer

时钟源

内部时钟CK_INT

外部输入引脚CHx

外部触发输入ETR

内部触发信号ITRx

定时模式

计数模式

时钟源选择

内部时钟：来自外设总线APB提供的时钟
CK_INT(定时器时钟TIM_CLK)

外部时钟模式1：捕获/比较通道CH1/CH2

时基单元的预分频时钟CK_PSC

计数模式

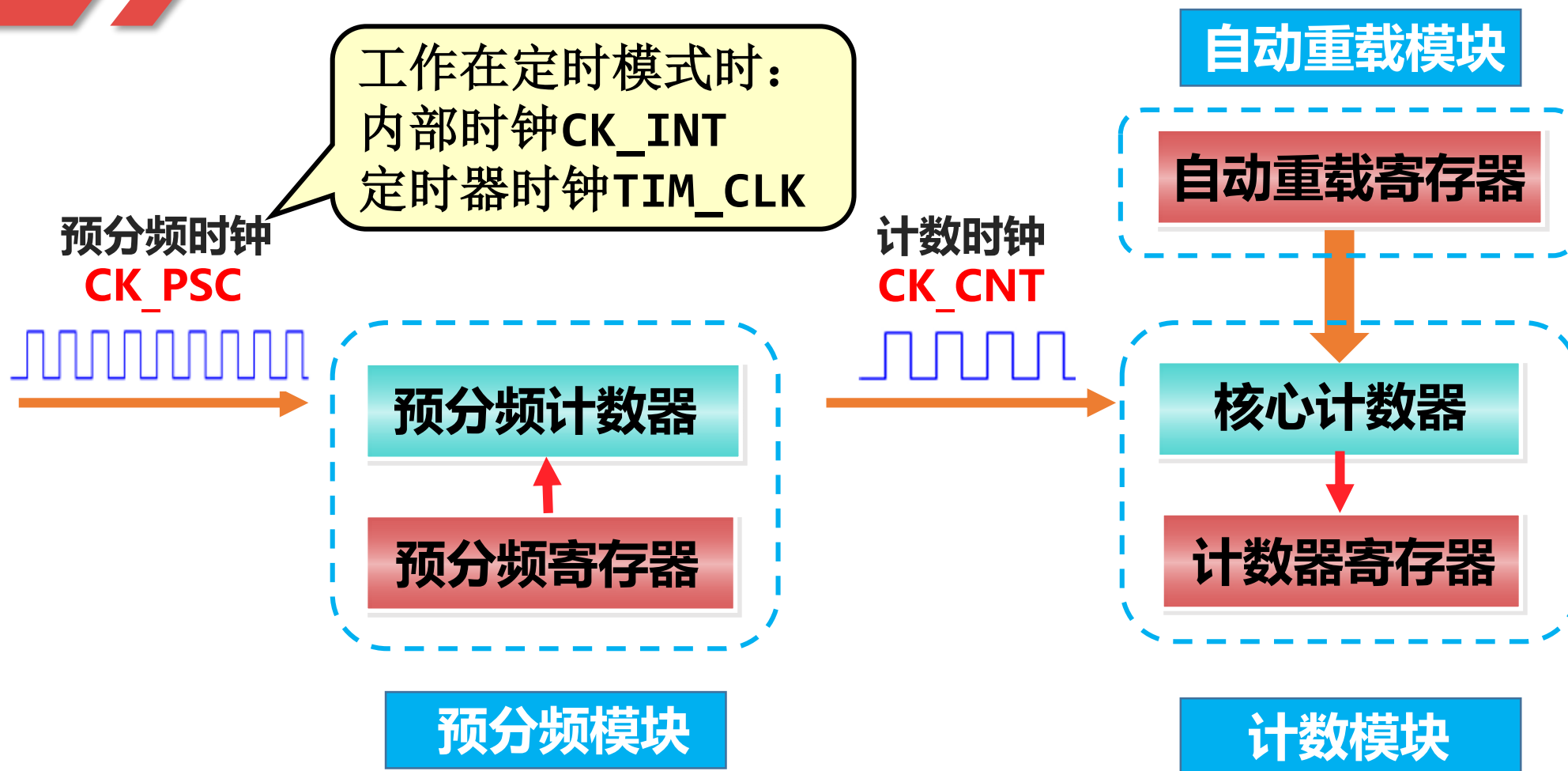
外部时钟模式2：外部触发引脚ETR

内部触发输入：使用一个定时器作为另一个
定时器的预分频器

从模式

时基单元

时基单元功能框架



预分频模块

预分频模块

x表示定时器的编号
如TIM1或TIM2

预分频计数器



对预分频时钟CK_PSC进行分频

预分频寄存器



TIMx_PSC: 设置预分频系数PSC

作用1: 扩大定时器的定时范围

作用2: 获取精确的计数时钟

TIM_CLK为100MHz, 16位定时器的最大定时时间为0.65ms, 经过预分频以后, 可以降低计数时钟CK_CNT的频率

TIM_CLK为72MHz, 可以通过设置预分频寄存器, 使计数时钟CK_CNT变为1MHz, 从而获得精确的计数时钟

预分频模块工作原理

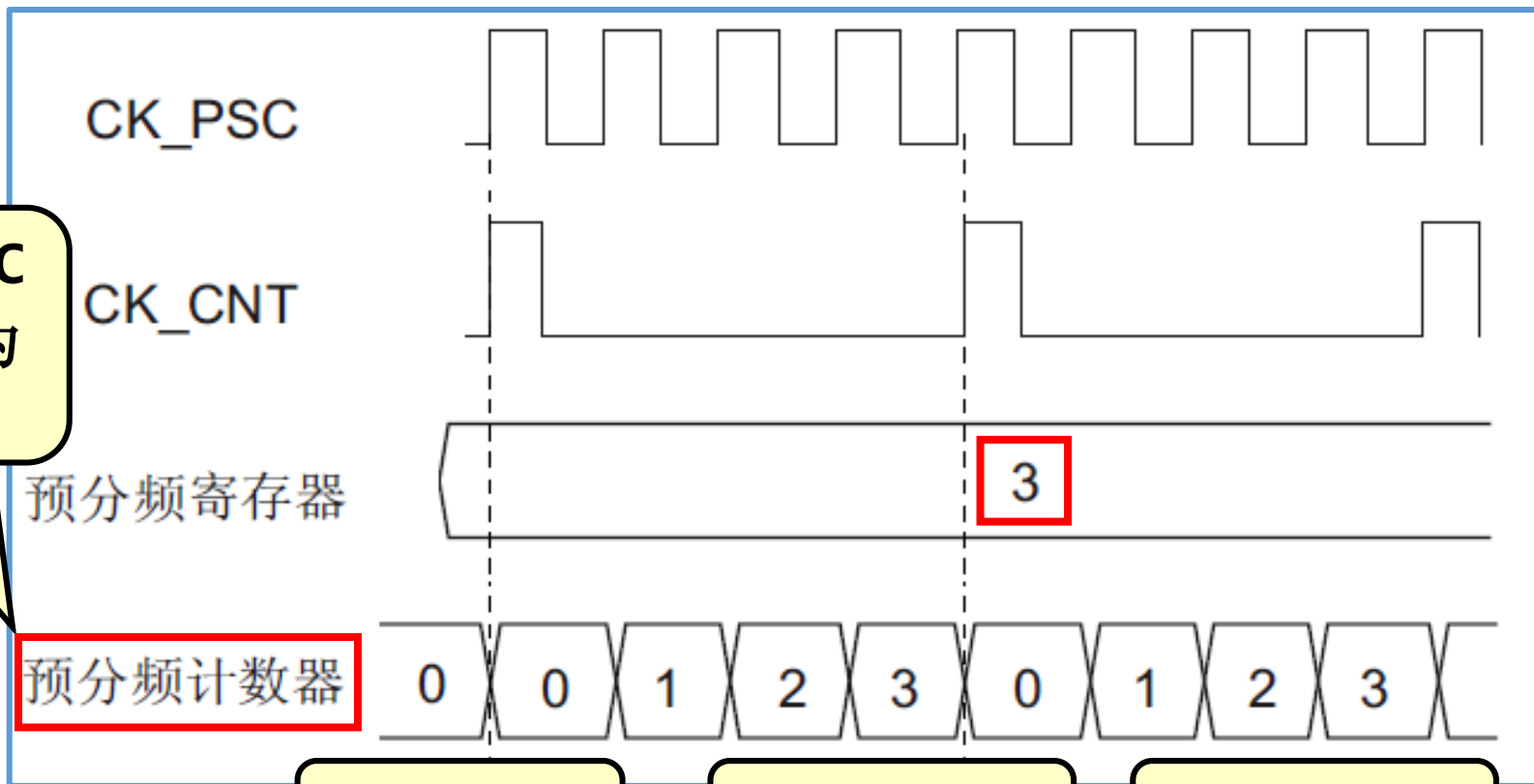
工作原理

定时器启动后，预分频计数器的初值为0，预分频时钟CK_PSC每来一个时钟，预分频计数器的值就加1。当计数值等于预分频寄存器所设定的预分频系数PSC时，预分频计数器的值将清零，开始下一轮计数。

预分频模块

预分频时序图

从0计数到PSC
实际计数值为
PSC+1



假设预分频
系数PSC=3

计数时钟

预分频时钟

预分频系数

分频公式

$$CK_CNT = CK_PSC / (PSC + 1)$$

计数模块

计数模块

核心计数器



对计数时钟CK_CNT进行二次计数

计数器寄存器



TIMx_CNT: 存放核心计数器运行时的当前计数值

自动重载模块

自动重载模块

自动重载模块由自动重载寄存器TIMx_ARR组成

递增计数模式



TIMx_ARR的值作为核心计数器的计数终值

递减计数模式



TIMx_ARR的值作为核心计数器的计数初值

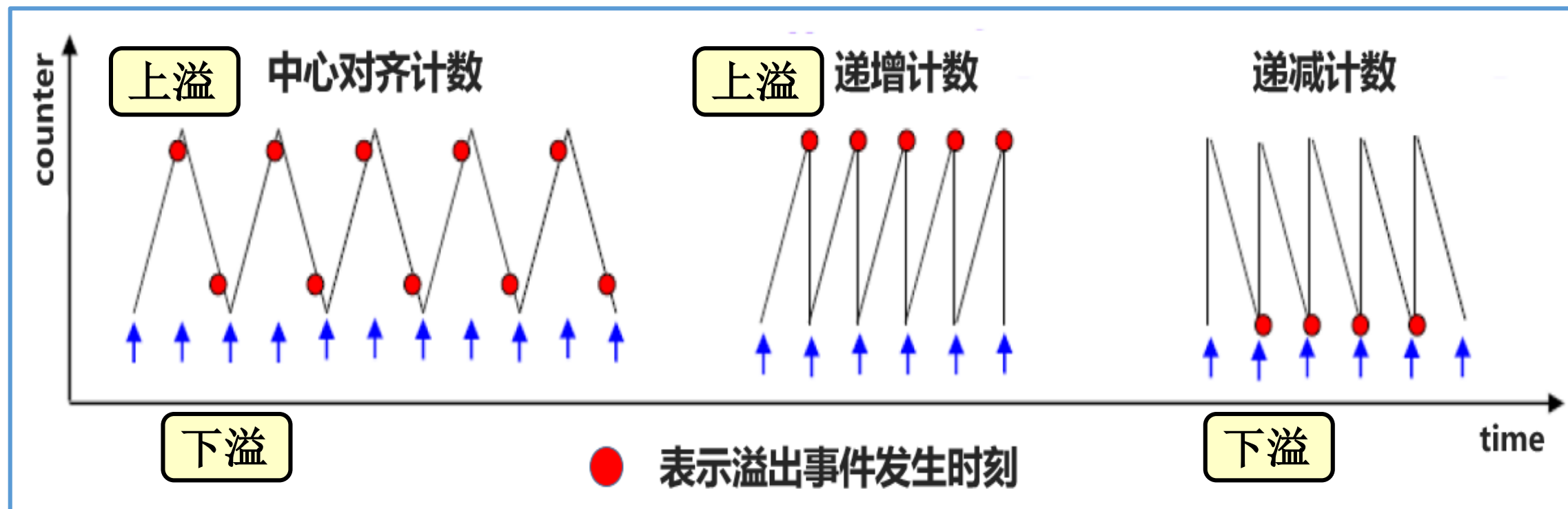
计数模式

定时器的三种计数模式

$0 \rightarrow \text{ARR}-1$
 $\text{ARR} \rightarrow 1$

$0 \rightarrow \text{ARR}$

$\text{ARR} \rightarrow 0$



计数模式

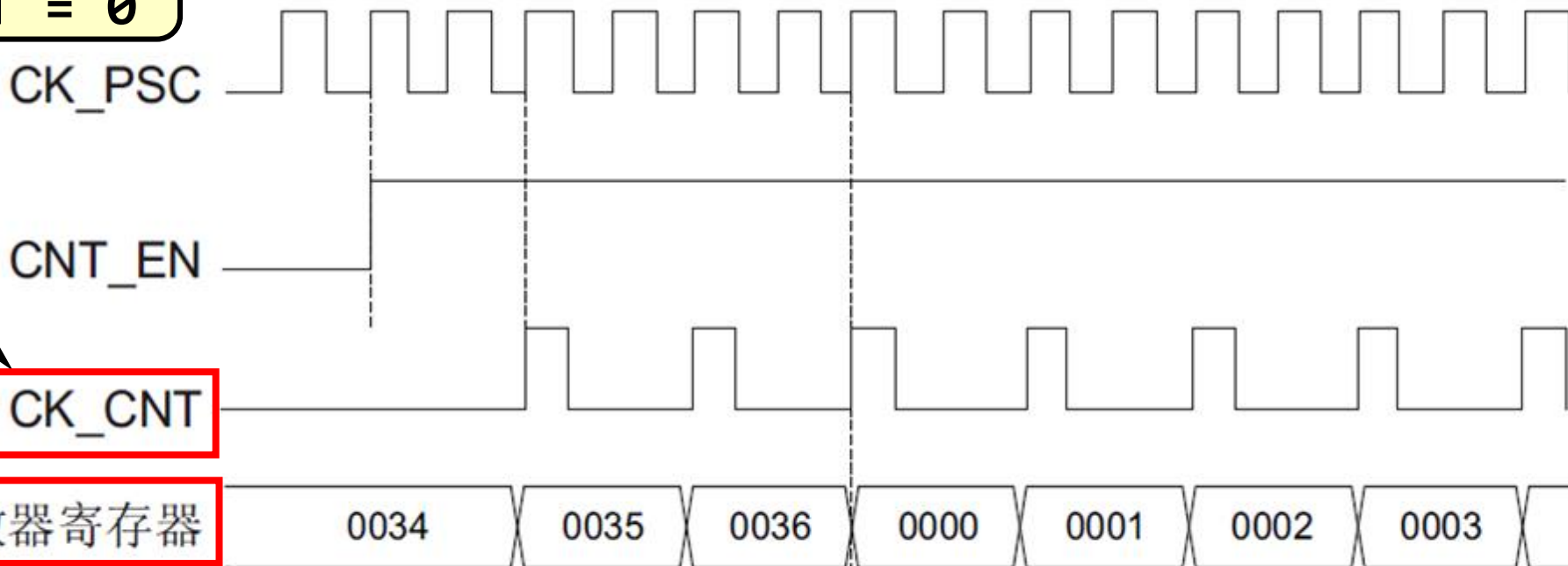
溢出值CNT与自动重载值ARR的关系列表

计数模式	计数器溢出值	计数器重载值
递增计数	$CNT = ARR$	$CNT = 0$
递减计数	$CNT = 0$	$CNT = ARR$
中心对齐计数	$CNT = ARR - 1$	$CNT = ARR$
	$CNT = 1$	$CNT = 0$

定时器时序图

PSC = 1, ARR = 36
递增计数, CNT = 0

PSC = 1
对CK_PSC
两分频



计数器寄存器

从0计数到ARR
实际计数值为
ARR+1

计数器上溢

更新事件 (UEV)

更新中断标志 (UIF)

上溢事件

更新中断/定时中断

定时时间公式

定时器的定时时间公式

$$\text{定时时间} = \text{计数值} * \text{计数时间}$$
$$\text{定时时间} = \text{计数值} / \text{时钟频率}$$

计数时钟
CK_CNT

$$\text{时钟频率} = \text{TIM_CLK} / (\text{PSC} + 1)$$

$$\text{计数值} = \text{ARR} + 1$$

定时器时钟，
等于预分频
时钟CK_PSC

自动重载值

预分频系数

$$T(s) = \frac{(\text{ARR} + 1) * (\text{PSC} + 1)}{\text{TIM_CLK}(\text{Hz})}$$

定时器时钟

相关寄存器



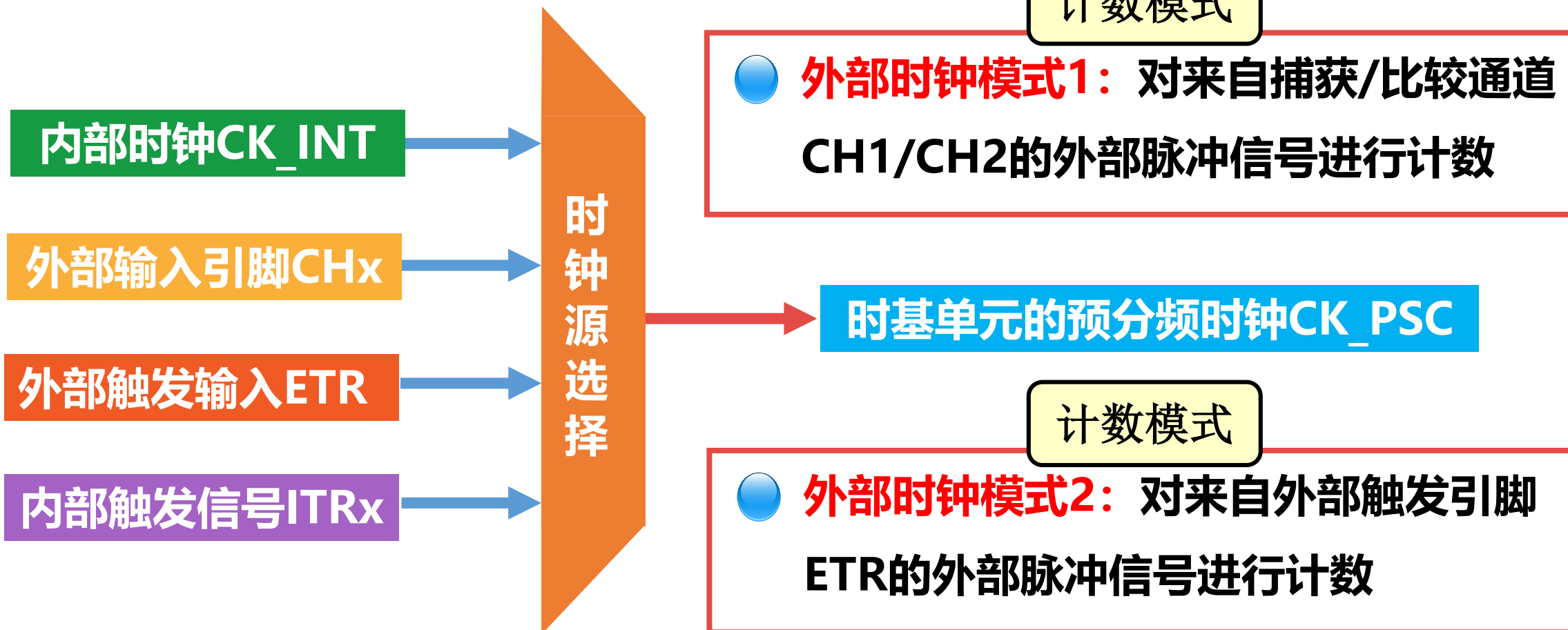


电子科技大学
University of Electronic Science and Technology of China

2 外部脉冲计数

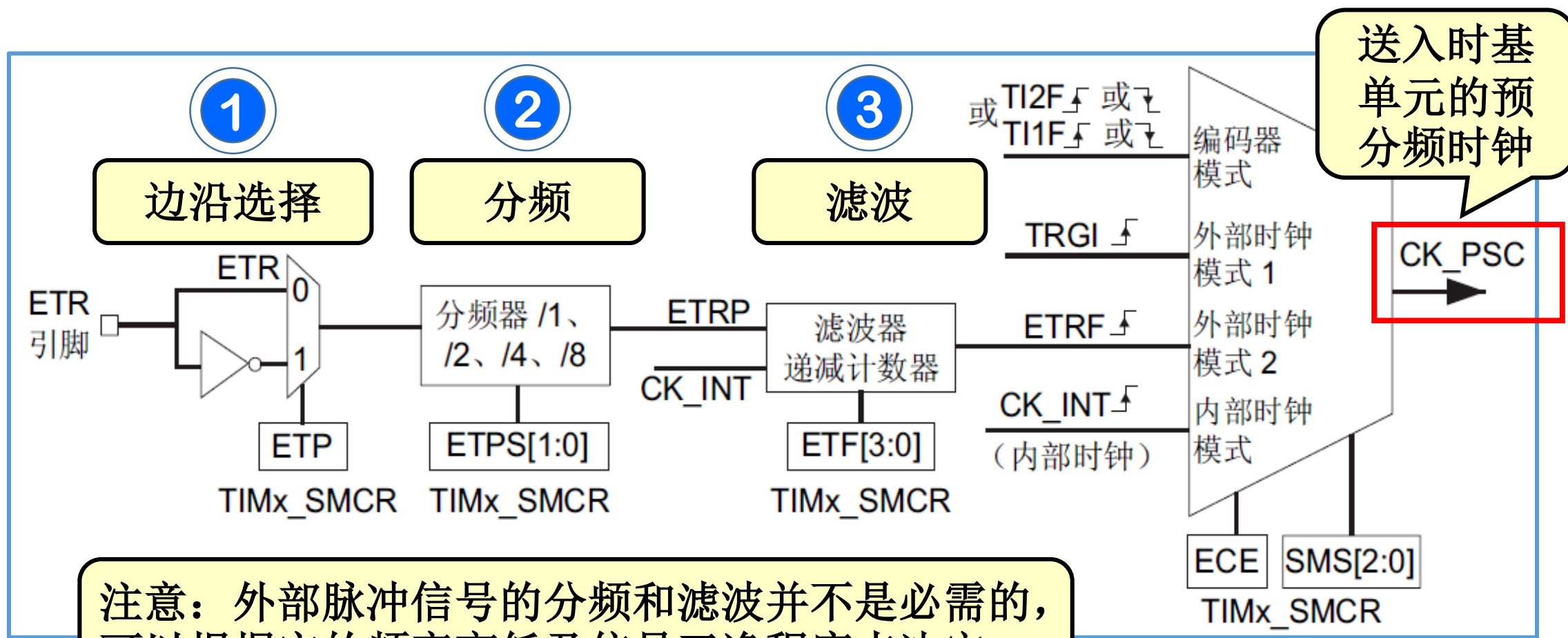
时钟源

时基单元的四钟时钟源



内部结构

外部触发引脚ETR的内部结构



注意：外部脉冲信号的分频和滤波并不是必需的，可以根据它的频率高低及信号干净程度来决定

STM32F411芯片的ETR引脚

查询芯片的数据手册
Datasheet

定时器的ETR引脚	对应GPIO引脚
TIM1_ETR	PA12
TIM2_ETR	PA0/PA5/PA15
TIM3_ETR	PD2



电子科技大学
University of Electronic Science and Technology of China

3 定时/计数功能的数据类型和接口函数

时基单元

时基单元初始化类型

```
1. typedef struct
```

```
2. {
```

```
3.     uint32_t Prescaler;
```

```
4.     uint32_t CounterMode;
```

```
5.     uint32_t Period;
```

```
6.     uint32_t ClockDivision;
```

```
7.     uint32_t RepetitionCounter;
```

```
8.     uint32_t AutoReloadPreload;
```

```
9. }
```

```
10. TIM_Base_InitTypeDef;
```

结构体类型，包括6个成员变量

// 表示预分频系数 PSC，即 TIMx_PSC 寄存器的内容

// 设置计数模式

// 表示自动重载值 ARR，即 TIMx_ARR 寄存器的内容

// 设置定时器时钟 TIM_CLK 分频值，用于输入信号滤波

// 表示重复定时器的值，只针对高级定时器

// 设置自动重载寄存器 TIMx_ARR 内容的生效时刻

Period的值不能设置为0，
否则定时器将不会启动

成员变量ClockDivision的取值范围

宏常量定义	含义
TIM_CLOCKDIVISION_DIV1	对定时器时钟TIM_CLK进行1分频
TIM_CLOCKDIVISION_DIV2	对定时器时钟TIM_CLK进行2分频
TIM_CLOCKDIVISION_DIV4	对定时器时钟TIM_CLK进行4分频

主要用于输入信号的滤波，一般使用默认值：1分频

成员变量CounterMode的取值范围

宏常量定义	含义
TIM_COUNTERMODE_UP	递增计数模式
TIM_COUNTERMODE_DOWN	递减计数模式
TIM_COUNTERMODE_CENTERALIGNED1	中心对齐计数模式1
TIM_COUNTERMODE_CENTERALIGNED2	中心对齐计数模式2
TIM_COUNTERMODE_CENTERALIGNED3	中心对齐计数模式3

三种中心对齐计数模式的区别主要是输出比较中断标志位的设置方式

成员变量AutoReloadPreload的取值范围

宏常量定义	含义
TIM_AUTORELOAD_PRELOAD_DISABLE	预装载功能关闭
TIM_AUTORELOAD_PRELOAD_ENABLE	预装载功能开启

1. 用于设置自动重载寄存器TIMx_ARR的预装载功能，即自动重装寄存器的内容是更新事件产生时写入有效，还是立即写入有效；
2. 预装载功能在多个定时器同时输出信号时比较有用，可以确保多个定时器的输出信号在同一个时刻变化，实现同步输出；
3. 单个定时器输出时，一般不开启预装载功能。

时基初始化

1 时基单元初始化函数：HAL_TIM_Base_Init

接口函数：HAL_TIM_Base_Init

函数原型	HAL_StatusTypeDef HAL_TIM_Base_Init (TIM_HandleTypeDef *htim)
功能描述	按照定时器句柄中指定的参数初始化定时器时基单元
入口参数	htim：定时器句柄的地址
返回值	HAL状态值
注意事项	1. 该函数将调用MCU底层初始化函数HAL_TIM_Base_MspInit完成引脚、时钟和中断的设置 2. 该函数由CubeMX自动生成

2 轮询模式启动函数：HAL_TIM_Base_Start

接口函数：HAL_TIM_Base_Start

函数原型	HAL_StatusTypeDef HAL_TIM_Base_Start (TIM_HandleTypeDef *htim)
功能描述	在轮询方式下启动定时器运行
入口参数	htim：定时器句柄的地址
返回值	HAL状态值
注意事项	<ol style="list-style-type: none">1. 该函数在定时器初始化完成之后调用2. 函数需要由用户调用，用于轮询方式下启动定时器运行

3 中断模式启动函数：HAL_TIM_Base_Start_IT

接口函数：HAL_TIM_Base_Start_IT

函数原型	HAL_StatusTypeDef HAL_TIM_Base_Start_IT (TIM_HandleTypeDef *htim)
功能描述	使能定时器的更新中断，并启动定时器运行
入口参数	htim：定时器句柄的地址
返回值	HAL状态值
注意事项	<ol style="list-style-type: none">1. 该函数在定时器初始化完成之后调用2. 函数需要由用户调用，用于使能定时器的更新中断，并启动定时器运行3. 启动前需要调用宏函数__HAL_TIM_CLEAR_IT来清除更新中断标志

4 定时器中断通用处理函数HAL_TIM_IRQHandler

接口函数：HAL_TIM_IRQHandler

函数原型	void HAL_TIM_IRQHandler(TIM_HandleTypeDef *htim);
功能描述	作为所有定时器中断发生后的通用处理函数
入口参数	htim：定时器句柄的地址
返回值	无
注意事项	<ol style="list-style-type: none">1. 函数内部先判断中断类型，并清除对应的中断标志，最后调用回调函数完成中断处理2. 该函数由CubeMX自动生成

5 定时器更新中断回调函数HAL_TIM_PeriodElapsedCallback

接口函数：HAL_TIM_PeriodElapsedCallback

函数原型	void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
功能描述	回调函数，用于处理所有定时器的更新中断，用户在该函数内编写实际的任务处理程序
入口参数	htim：定时器句柄的地址
返回值	无
注意事项	<ol style="list-style-type: none">1. 该函数由定时器中断通用处理函数HAL_TIM_IRQHandler调用，完成所有定时器的更新中断的任务处理2. 函数内部需要根据定时器句柄的实例来判断是哪一个定时器产生的本次更新中断3. 函数由用户根据具体的处理任务编写

6 计数值读取函数 `__HAL_TIM_GET_COUNTER`

带参数的宏

```
#define __HAL_TIM_GET_COUNTER(__HANDLE__) ((__HANDLE__)->Instance->CNT)
```

定时器句柄的地址

该函数通过直接访问计数器寄存器
`TIMx_CNT`来获取计数器的当前计数值

7 定时器中断标志清除函数 `__HAL_TIM_CLEAR_IT`

带参数的宏

定时器句柄
的地址

定时器中断标志

```
#define __HAL_TIM_CLEAR_IT(__HANDLE__, __INTERRUPT__)\n((__HANDLE__)->Instance->SR = ~(__INTERRUPT__))
```

常用中断标志

宏常量定义	含义
<code>TIM_IT_UPDATE</code>	更新中断标志
<code>TIM_IT_CC1</code>	通道1的捕获/比较中断标志
<code>TIM_IT_CC2</code>	通道2的捕获/比较中断标志
<code>TIM_IT_CC3</code>	通道3的捕获/比较中断标志
<code>TIM_IT_CC4</code>	通道4的捕获/比较中断标志



电子科技大学
University of Electronic Science and Technology of China

4 基础任务：定时闪烁指示灯

定时闪烁指示灯

01 任务目标

掌握CubeMX软件配置定时器实现基本定时的方法

02 任务内容

利用定时器10，控制Nucleo开发板上的指示灯LD2，每隔1s闪烁

任务分析

定时时间计算公式

$$T(s) = \frac{(ARR + 1) * (PSC + 1)}{TIM_CLK(Hz)}$$

分析:

1. 本任务使用定时器10，挂接在APB2总线，时钟频率为100MHz；
2. 任务要求的定时时间为1s，由于TIM_CLK为100MHz，可以假设PSC为9999，根据公式计算可得ARR的值也为9999。注意实际的取值要减1。

注意：PSC和ARR参数的选取，以不超过它们的计数范围为准。TIMx_PSC为16位寄存器，最大预分频系数为65536。TIMx_ARR寄存器的位数由定时器位数决定：16位定时器，ARR的最大值为65535；32位定时器，ARR的最大值为4294967295。

引脚分配

步骤二：引脚分配

设置指示灯引脚PA5

设置引脚为
输出模式

外设配置

步骤三：外设配置

PA5 Configuration :

配置引脚PA5的初始化参数

GPIO output level

初始电平

Low

GPIO mode

引脚模式

Output Push Pull

GPIO Pull-up/Pull-down

上/下拉电阻

No pull-up and no pull-down

Maximum output speed

引脚速度

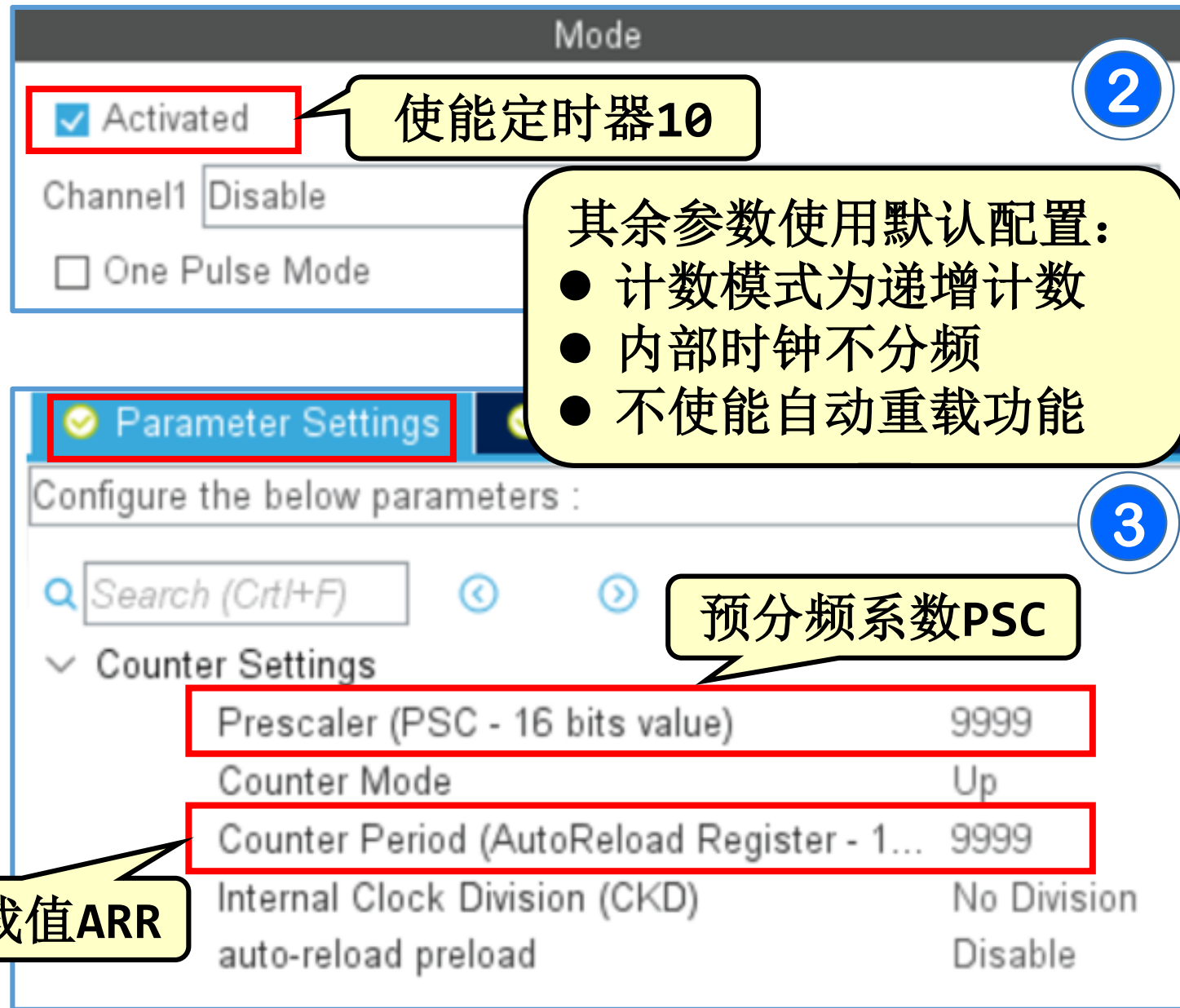
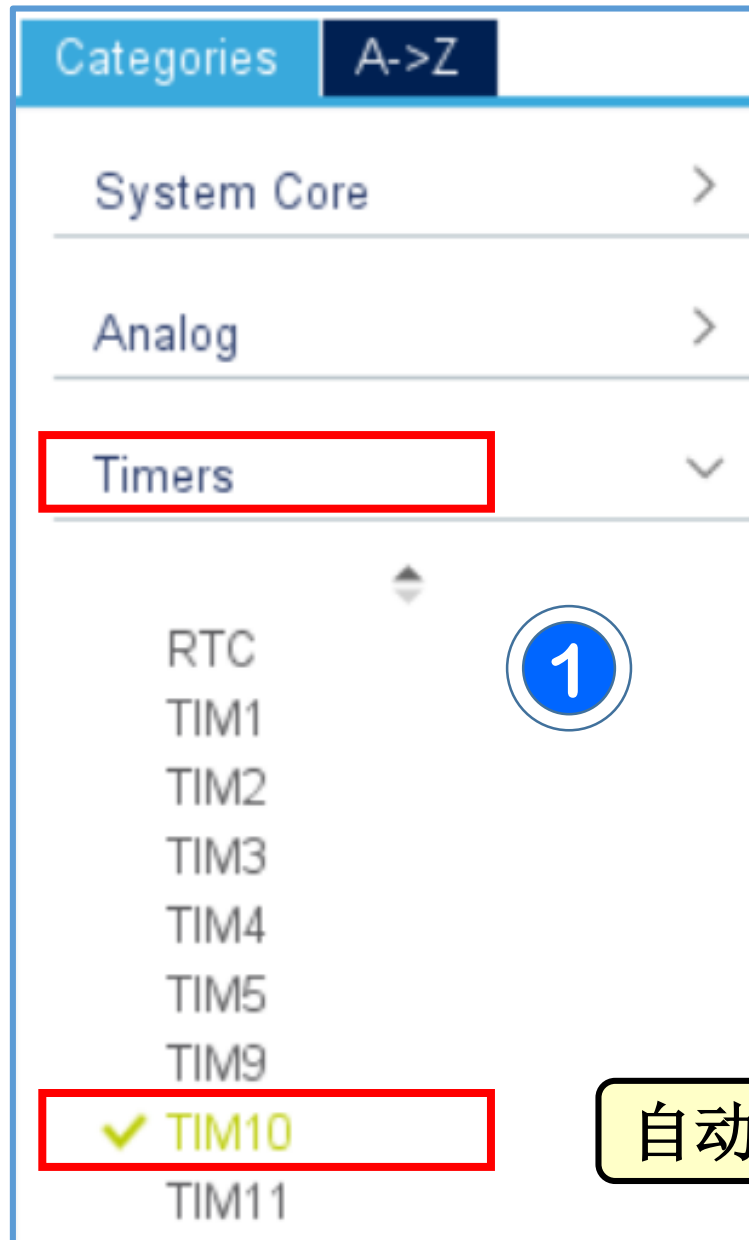
Low

User Label

引脚名称

LD2

定时器10的配置




```

1. /* Private variables ----- 主程序代码 ----- CubeMX自动生成 */
2. TIM_HandleTypeDef htim10; // 定时器10的句柄htim10
3. /* Private function prototypes -- ----- */
4. static void MX_TIM10_Init(void); // 定时器10初始化函数声明
5. int main(void)
6. {
7.     HAL_Init(); // HAL初始化
8.     MX_TIM10_Init(); // 定时器10初始化
9.     /* USER CODE BEGIN 2 */
10.    // 清除定时器初始化过程中的更新中断标志，避免定时器一启动就进入中断
11.    __HAL_TIM_CLEAR_IT(&htim10, TIM_IT_UPDATE);
12.    // 使能定时器10更新中断并启动定时器
13.    HAL_TIM_Base_Start_IT(&htim10);
14.    /* USER CODE END 2 */
15.}

```

步骤六：程序编写

添加用户初始化代码

定时器更新中断回调函数

```
1. /* USER CODE BEGIN 4 */
2. /*
3.  * @brief Period elapsed callback in non-blocking mode
4.  * @param htim TIM handle
5.  * @retval None
6.  */
7. void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
8. {
9.     if( htim ->Instance==TIM10 )           // 判断发生更新中断的定时器
10.    {
11.        HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin); // 翻转LD2状态
12.    }
13. }
14. /* USER CODE END 4 */
```

使用句柄的实例来判断发生更新中断的定时器

定时中断流程

定时器10递增计数，从0开始记到自动重载值ARR时，产生计数器上溢事件，触发更新中断

在启动文件中根据中断类型号找到对应的中断服务程序TIM1_UP_TIM10_IRQHandler

在中断服务程序中再调用定时器通用处理函数HAL_TIM_IRQHandler

在函数内部先判断中断类型，并清除对应的中断标志，然后调用更新中断回调函数HAL_TIM_PeriodElapsedCallback
完成具体的任务处理

定时器初始化函数

```
1. static void MX_TIM10_Init(void)
2. {
3.     htim10.Instance = TIM10;
4.     htim10.Init.Prescaler = 9999;
5.     htim10.Init.CounterMode = TIM_COUNTERMODE_UP;
6.     htim10.Init.Period = 9999;
7.     htim10.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
8.     htim10.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
9.     if (HAL_TIM_Base_Init(&htim10) != HAL_OK)
10.    {
11.        Error_Handler();
12.    }
13.}
```

由CubeMX自动生成

本任务为基本定时功能，只进行了时基单元的初始化，初始化参数根据用户在CubeMX中的设置决定

时基单元初始化函数

练习任务

1 任务一：软件计数器

如果不使用预分频寄存器，只使用核心计数器，如何实现1s的定时中断？

2 任务二：定时中断处理按键

对照第六章中采用状态机思想所设计的按键处理程序，完成10ms定时中断的设置。



电子科技大学
University of Electronic Science and Technology of China

5 进阶任务：测量外部脉冲个数

基础任务

测量外部脉冲的个数

01

任务目标

掌握CubeMX软件配置定时器实现基本计数的方法

02

任务内容

利用Nucleo开发板上的按键B1来触发外部脉冲，按键每按下一次，就利用PA1引脚发送一个周期2ms左右的脉冲，送到定时器2的外部触发引脚ETR（PA0）进行计数，并将计数结果通过串口发送到PC上显示。

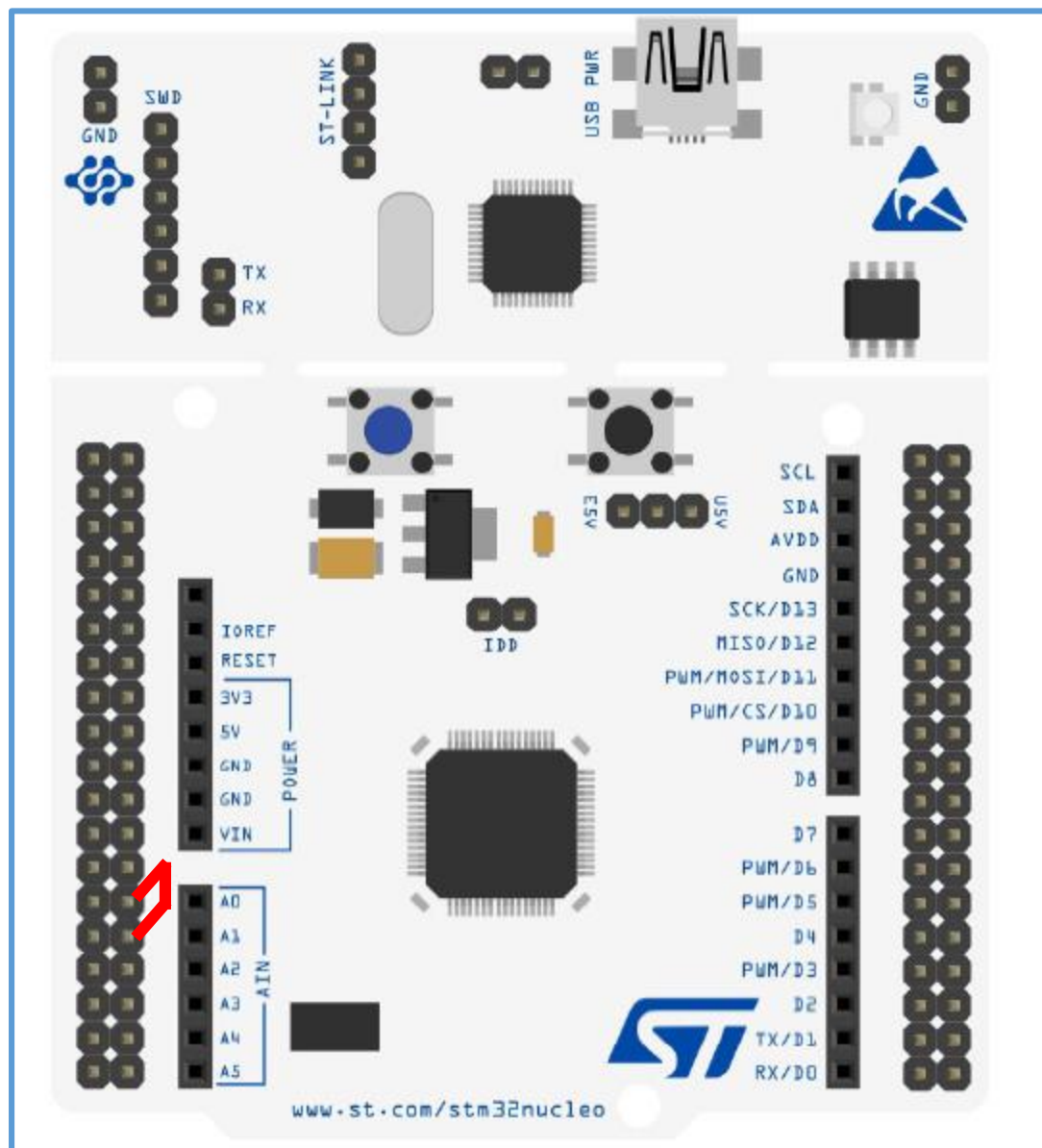
设计思路

设计思路

1. 按键的读取采用状态机设计思想，使用定时器10产生10ms的定时中断，相关配置和用户代码可以参考第六章中的介绍；
2. 利用杜邦线将引脚PA0和PA1连接，PA1引脚配置为输出模式，通过按键触发来发送脉冲；
3. PA0引脚配置为定时器2的外部触发引脚，用来测量PA1引脚发送的脉冲；
4. 定义一个全局变量Result，用于存放外部脉冲的计数值；
5. 配置Nucelo开发板的串口2，用于串口数据发送。

硬件连接图

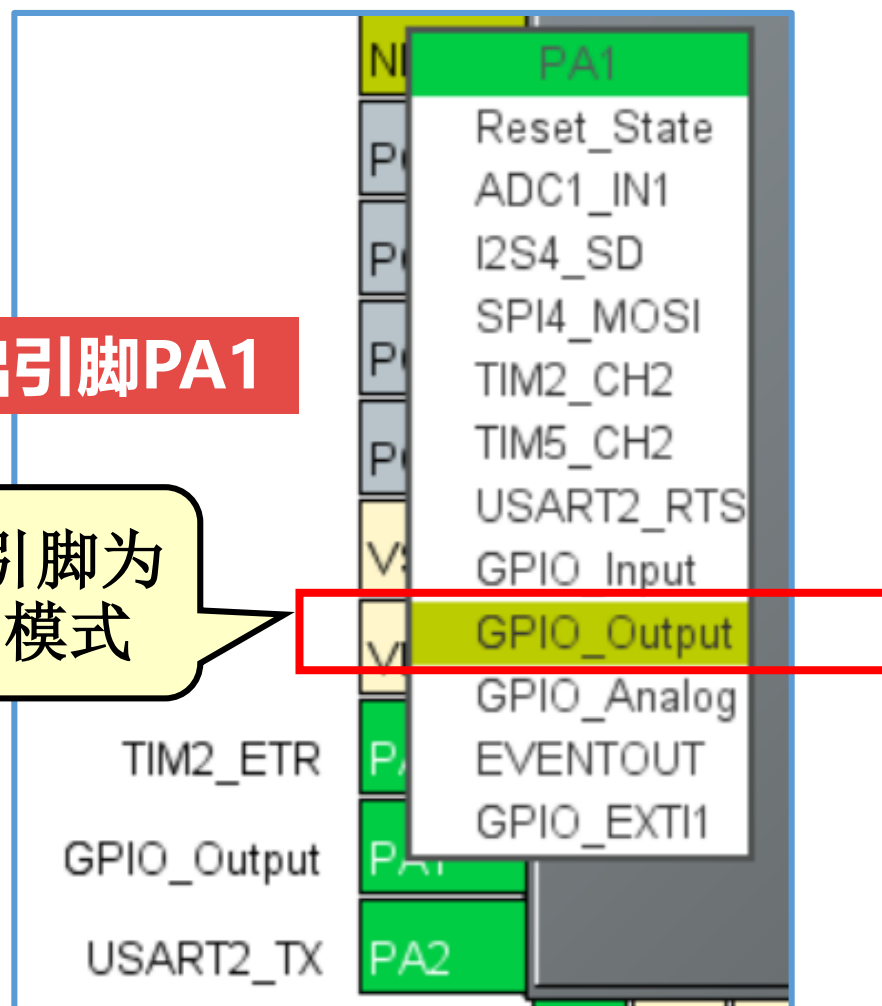
利用杜邦线将PA0
和PA1连接，连接
示意图如图所示



引脚分配

步骤二：引脚分配

设置脉冲输出引脚PA1

设置引脚为
输出模式

外设配置

步骤三：外设配置

PA1 Configuration :

配置引脚PA1的初始化参数

GPIO output level

初始电平

Low

GPIO mode

引脚模式

Output Push Pull

GPIO Pull-up/Pull-down

上/下拉电阻

No pull-up and no pull-down

Maximum output speed

引脚速度

Low

User Label

引脚名称

PULSE

配置定时器2的时钟源

Categories A->Z

System Core >

Analog >

Timers v

RTC

TIM1

TIM2

TIM3

TIM4

TIM5

TIM9

TIM10

TIM11

1

TIM2 Mode and Configuration

Mode 2

Slave Mode Disable

Trigger Source Disable

Clock Source ETR2

Channel1 Disable

Channel2 Disable

Channel3 Disable

Channel4 Disable

选择定时器2的时钟源为外部触发引脚ETR2

配置定时器2的初始化参数

✓ NVIC Settings		✓ DMA Settings		✓ GPIO Settings	
✓ Parameter Settings		✓ User Constants			
Configure the below parameters :					
Search (Ctrl+F) ⏪ ⏩ ⓘ					
✓ Counter Settings					
Prescaler (PSC - 16 bits value)		0			
Counter Mode		Up			
Counter Period (AutoReload ...)		65535			
Internal Clock Division (CKD)		No Division			
auto-reload preload		Disable			
✓ Clock					
Clock Filter (4 bits value)		0			
Clock Polarity		non inverted			
Clock Prescaler		Prescaler not used			

将自动重载值ARR配置为一个相对较大的计数终值，以避免计数溢出

外部脉冲信号采用默认配置：

- 不使用滤波
- 不进行脉冲信号反相
- 不进行脉冲信号分频

Categories

A->Z

Connectivity

I2C1

I2C2

I2C3

SDIO

SPI1

SPI2

SPI3

SPI4

SPI5

USART1

✓ USART2

USART6

USB_OTG_FS

串口外设配置

Mode

Mode Asynchronous

Hardware Flow Control (RS232) Disable

Configuration

Reset Configuration

✓ NVIC Settings

✓ DMA Settings

✓ GPIO Settings

✓ Parameter Settings

✓ User Constants

Configure the below parameters :

Search (Ctrl+F)

Basic Parameters

Baud Rate

115200 Bits/s

Word Length

8 Bits (including Parity)

Parity

None

Stop Bits

1

选择异步模式
无硬件流控

设置串口通信参数
波特率115200，8位数据位
无奇偶校验，1位停止位

步骤六：程序编写

用户变量定义

```
1. /* USER CODE BEGIN PV */
2. KEY_STATE KeyState = KEY_CHECK; // 按键状态，初值为按键检测状态
3. uint8_t KeyFlag     = 0;         // 按键值有效标志，1：有效；0：无效
4. uint8_t Result      = 0;         // 存放计数值
5. /* USER CODE END PV */
```

用户初始化代码

```
1. /* USER CODE BEGIN 2 */
2. HAL_TIM_Base_Start_IT(&htim10); // 启动定时器10，用于按键检测
3. HAL_TIM_Base_Start(&htim2);     // 启动定时器2，用于外部脉冲计数
4. printf(" Timer count function test: \n"); // 发送提示信息
5. /* USER CODE END 2 */
```

用户应用代码

```
1.  /* USER CODE BEGIN 3 */
2.      if(KeyFlag == 1)                                // 检测按键标志
3.      {
4.          KeyFlag = 0;                                // 清除标志
5.          // 发送一个周期为2ms左右的脉冲
6.          HAL_GPIO_WritePin(GPIOA, PULSE_Pin, GPIO_PIN_SET);
7.          HAL_Delay(1);
8.          HAL_GPIO_WritePin(GPIOA, PULSE_Pin, GPIO_PIN_RESET);
9.          HAL_Delay(1);
10.         Result  = __HAL_TIM_GET_COUNTER(&htim2); // 读取计数值
11.         printf(" Count = %d.\n",Result);         // 发送到PC
12.     }
13. }
14. /* USER CODE END 3 */
```


添加头文件

main.h

```
/* Private includes ----- */
```

```
/* USER CODE BEGIN Includes */
```

```
#include "stdio.h"           // 包含标准输入输出模块的头文件
```

```
/* USER CODE END Includes */
```

重定义fputc函数

```
/* USER CODE BEGIN 4 */
```

```
/*  
*****
```

```
* 函 数 名: fputc
```

```
* 功能说明: 重定义fputc函数, 以便使用printf函数从串口打印输出
```

```
* 形 参: 无
```

```
* 返 回 值: 无
```

```
*****
```

```
int fputc(int ch, FILE *f)
```

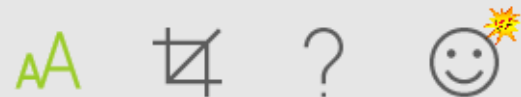
```
{
```

```
    HAL_UART_Transmit(&huart2, (uint8_t *)&ch, 1, HAL_MAX_DELAY);
```

```
    return ch;
```

```
}
```

```
/* USER CODE END 4 */
```



设置串口参数
与MCU端一致

1

串口号: COM6 ✓
波特率: 115200
数据位: 8
校验位: None
停止位: One

```
Timer count function test:  
count = 1.  
count = 2.  
count = 3.  
count = 4.
```

关闭串口

打开串口

3

接收区设置.

☐ 接收并保存到文件

☐ 十六进制显示

☐ 暂停接收显示

☐ 自动断帧

☐ 接收脚本

选择ASCII码显示

2

下载方法

串口调试助手可以在微软应用商店中搜索关键字“串口调试助手”来进行下载

? 20

Add Timesta

发送: 0

接收: 77

复位计数

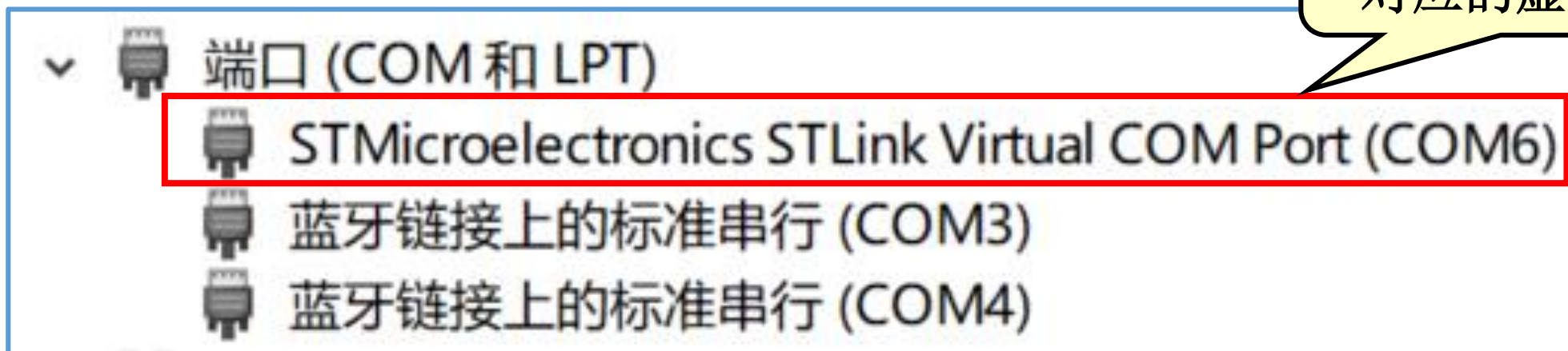


查找串口号

查找Nucleo开发板的虚拟串口号

查找虚拟串口号

将Nucleo开发板与电脑连接-> 右键“我的电脑”
-> 属性 -> 设备管理器 -> 端口(COM 和 LPT)





ST-Link仿真器所
对应的虚拟串口

MDK的调试方法

方法一：利用Watch窗口查看变量

在代码编辑窗口中，选择Result变量，鼠标右键加入到Watch窗口。然后进入调试界面，全速运行程序。此时，用户每按键一次，Result的值就会加1。



Name	Value	Type
 Result	0x02	uchar
<Enter expression>		

实时显示Result
变量的值

MDK的调试方法

方法二：查看寄存器

1. 进入调试界面后，在MDK的菜单栏中依次选择：
Peripherals -> System Viewer
-> TIM -> TIM2
将出现定时器2的寄存器窗口；
2. 全速运行程序。此时，用户每按键一次，CNT的值就会加1。



TIM2	
Property	Value
CCMR1_Input	0
CCMR2_Ou...	0
CCMR2_Input	0
CCER	0
CNT	0x00000002
PSC	0
ARR	0x0000FFFF

计数器寄存器
TIM_CNT，实
时显示定时器的
当前计数值



电子科技大学
University of Electronic Science and Technology of China

6 挑战任务：设计电子时钟

挑战任务

设计电子时钟

01

任务目标

掌握CubeMX软件配置定时器实现基本定时的方法

02

任务内容

设计电子时钟，从00:00:00开始计时，并将计时信息通过Nucleo开发板上的串口UART2发送到PC进行显示

设计思路

设计思路

1. 定义一个结构体变量，包含时、分、秒三个成员变量。设置定时器产生1s的更新中断，在更新中断回调函数中，修改时、分、秒的值。主程序在while循环中每隔1s调用printf函数将时间信息发送到PC显示；
2. 使用定时器10产生1s的更新中断，设置PSC为9999，ARR为9999，相关配置方法可以参考基础任务；
3. 配置串口2为异步模式，串口2配置参数为：波特率115200，8位数据位，1位停止位，无奇偶校验，相关配置方法可以参考进阶任务。

步骤六：程序编写

```
1.  /* USER CODE BEGIN PTD */
2.  // 定义时钟结构体类型，包含时、分、秒 3 个成员变量
3.  typedef struct
4.  {
5.      uint8_t hour;
6.      uint8_t minute;
7.      uint8_t second;
8.  }CLOCK_TypeDef;
9.  /* USER CODE END PTD */
```

用户数据类型定义

```
10. /* USER CODE BEGIN PV */
```

用户变量定义

```
11. CLOCK_TypeDef clock = {0} ;
```

// 定义时钟结构体变量，初始化为0

```
12. /* USER CODE END PV */
```

```
15. /* USER CODE BEGIN 2 */
```

用户初始化代码

```
16. // 清除更新中断标志，避免定时器一启动就进入中断
```

```
17. __HAL_TIM_CLEAR_IT(&htim10, TIM_IT_UPDATE);
```

```
18. HAL_TIM_Base_Start_IT(&htim10); // 使能定时器10 更新中断并启动定时器
```

```
19. /* USER CODE END 2 */
```

```
25. /* USER CODE BEGIN 3 */
```

用户应用代码

```
26. // 每隔1s 显示时间信息
```

```
27. printf("Time:%02d:%02d:%02d.\r\n",clock.hour,clock.minute,clock.second);
```

```
28. HAL_Delay(1000);
```

```
29. }
```

```
30. /* USER CODE END 3 */
```

输出占两个字符宽度，不足补0

```
1. void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
2. {
3.     if( htim->Instance == TIM10 ) // 判断中断源
4.     {
5.         clock.second++;           // 依次修改时、分、秒变量
6.         if( clock.second == 60)
7.         {
8.             clock.second = 0;
9.             clock.minute++;
10.            if( clock.minute == 60 )
11.            {
12.                clock.minute = 0;
13.                clock.hour++;
14.                if( clock.hour == 24)
15.                {
16.                    clock.hour = 0;
17.                }
18.            }
19.        }
20.    }
21. }
```

定时器更新中断回调函数

添加位置

USER CODE BEGIN 4
USER CODE END 4

程序运行结果

AA



?



数据位：

8

校验位：

None

停止位：

One

关闭串口

接收区设置.

☐ 接收并保存到文件☐ 十六进制显示☐ 暂停接收显示☐ 自动断帧

?

20

☐ 接收脚本

Add Timesta



保存数据

清空数据

发送区设置.

Time: 00:00:00.

Time: 00:00:01.

Time: 00:00:02.

Time: 00:00:03.

Time: 00:00:04.

Time: 00:00:05.



发送：0

接收：108

复位计数



电子科技大学
University of Electronic Science and Technology of China

8.4 定时器的PWM输出功能

1 PWM工作原理

概述

基本特性

脉冲宽度调制 (PWM) 是一种对模拟信号电平进行数字编码的方法。广泛应用于电机控制、灯光的亮度调节、功率控制等领域。



脉宽调制动画



脉宽调制的实质是修改高电平的持续时间

两个参数

PWM信号的两个基本参数

周期 (Period)

一个完整PWM波形所持续的时间

占空比 (Duty)

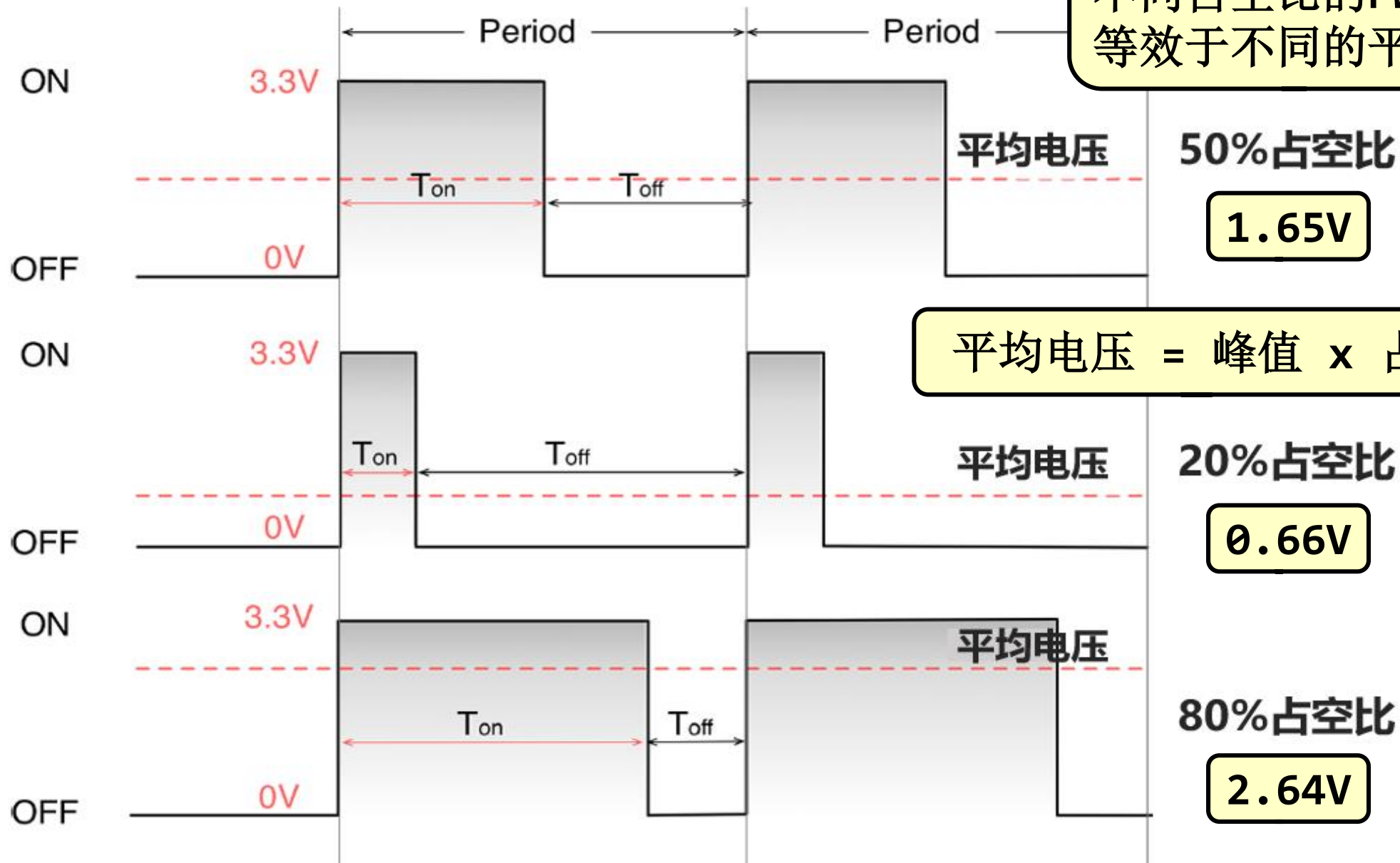
高电平持续时间 (Ton) 与周期时间 (Period) 的比值

占空比计算公式

$$\text{Duty} = (T_{\text{on}} / \text{Period}) \times 100\%$$

PWM信号的电压调节原理

电压调节的原理
不同占空比的PWM信号
等效于不同的平均电压



$$\text{平均电压} = \text{峰值} \times \text{占空比}$$

定时器通道结构

时基单元工作于定时模式，
预分频时钟CK_PSC等于定
时器时钟TIMx_CLK

时基单元

CK_PSC



预分频寄存器

CK_CNT



计数器寄存器

自动重载寄存器

每个定时器具备1~4个独
立的通道，各个通道具有
独立的输入捕获单元、捕
获/比较寄存器和输出比
较单元，但共享同一个时
基单元

x表示定时器编号，每个通
道有对应的GPIO引脚作为
通道的输入/输出引脚

通道引脚

TIMx_CH

输入滤波
边沿检测预分
频器

捕获/比较寄存器

输出
控制

通道引脚

TIMx_CH

输入捕获单元

每个通道可以选择作为输入捕获或
者输出比较功能，但是只能二选一

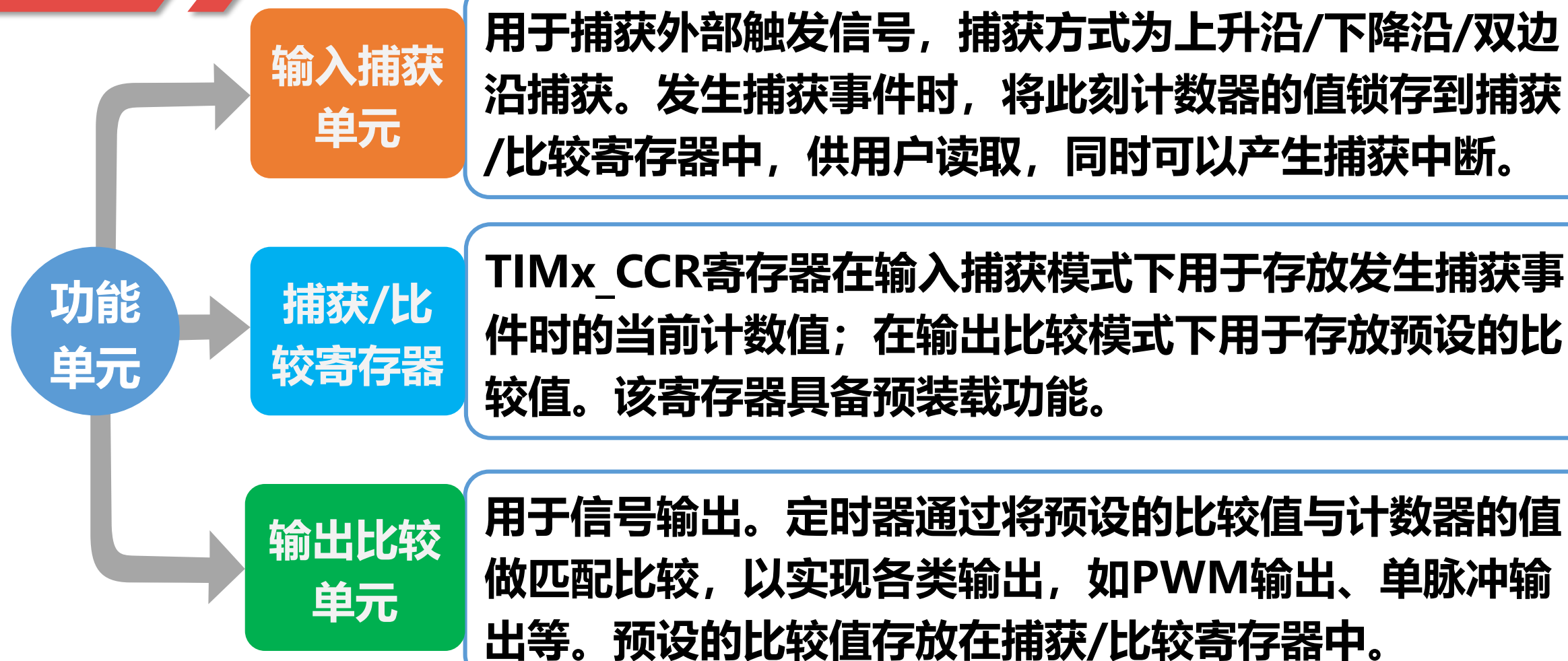
输出比较单元

STM32F411定时器2的通道引脚

查询芯片的数据手册
Datasheet

定时器2的通道引脚	对应GPIO引脚
TIM2_CH1	PA0/PA5/PA15
TIM2_CH2	PA1/PB3
TIM2_CH3	PA2/PB10
TIM2_CH4	PA3

功能单元作用



工作原理

PWM输出的工作原理

■ 初始输出高电平

■ 匹配CCR时输出为低

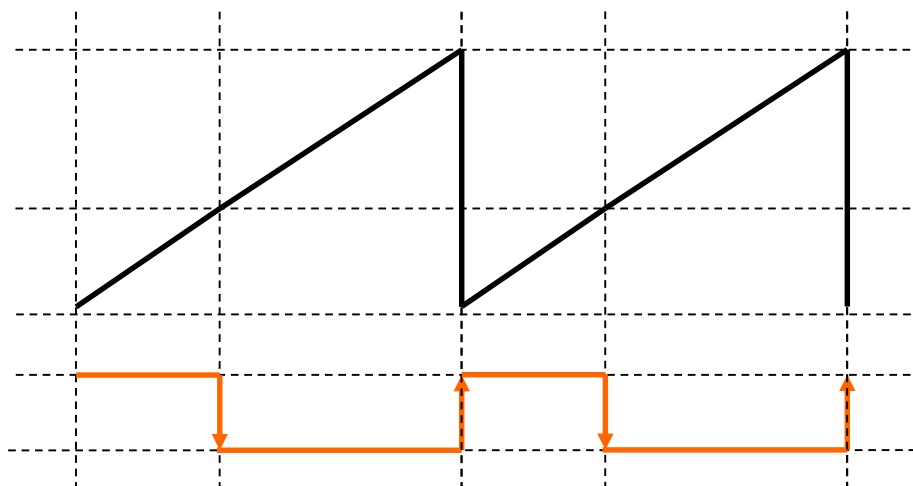
■ 匹配ARR时输出为高

自动重载寄存器ARR

捕获/比较寄存器CCR

计数器寄存器CNT

通道CHx输出波形



控制PWM信号的周期

控制PWM信号的占空比

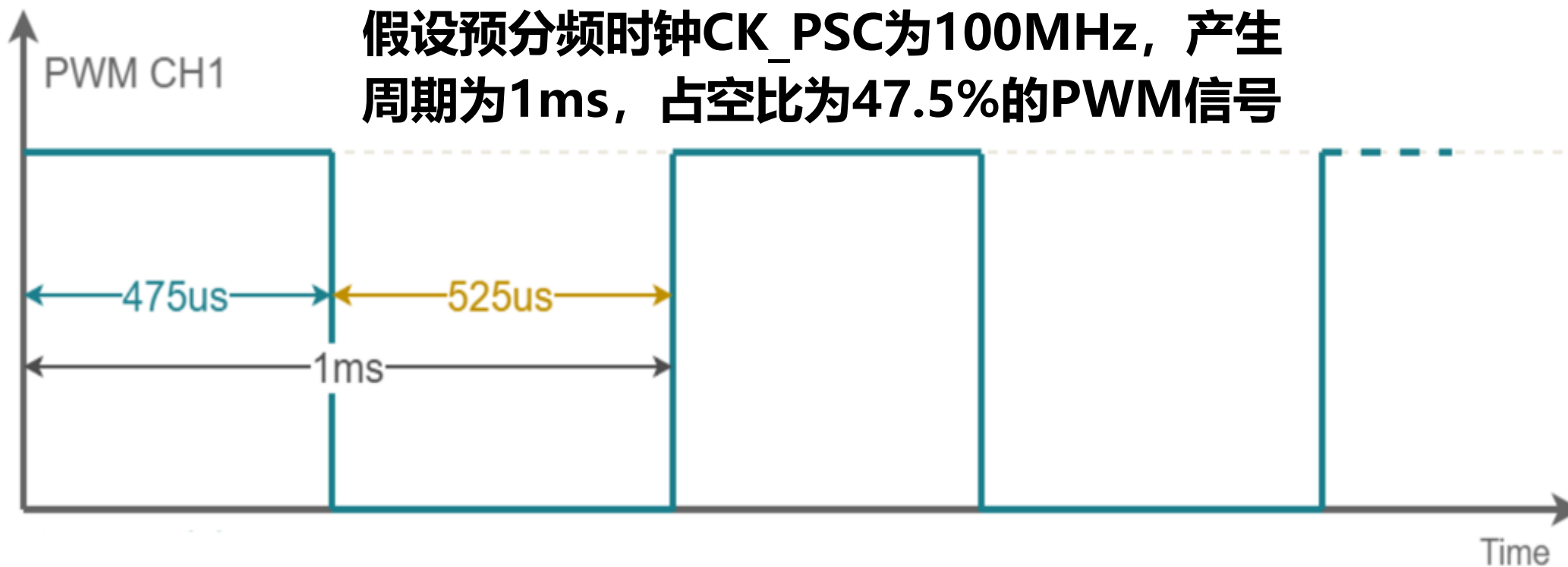
输出PWM信号

参数计算公式

$$\text{Period(s)} = (\text{ARR} + 1) \times (\text{PSC} + 1) / \text{TIMx_CLK}$$

$$\text{Duty} = (\text{CRR} / (\text{ARR} + 1)) \times 100\%$$

应用实例

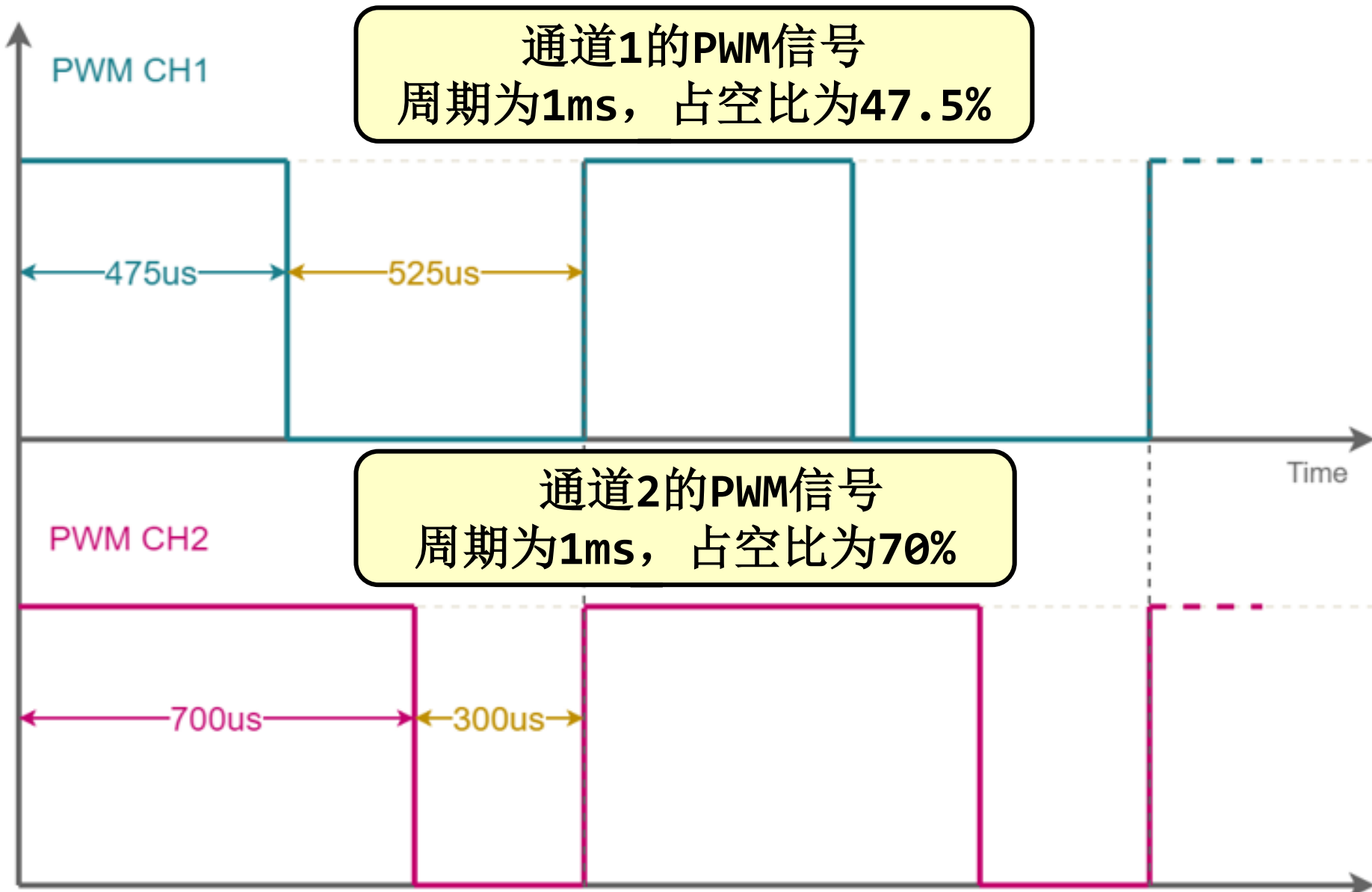


Period = 1ms, 可以设置 PSC = 99, ARR = 999

Duty = 47.5%, 则 CRR = 475

多通道输出

定时器的每个通道都可以输出PWM信号，对于同一个定时器而言，它的多个通道共享同一个自动重载寄存器，因此可以输出**占空比不同，但周期相同**的PWM信号。





电子科技大学
University of Electronic Science and Technology of China

2 PWM功能的数据类型和接口函数

输出比较单元初始化类型

结构体类型，包括7个成员变量，与PWM相关的4个

```
1. typedef struct
```

```
2. {
```

```
3.     uint32_t OCMode;
```

// 设置定时器输出比较模式

```
4.     uint32_t Pulse;
```

// 表示捕获/比较值 CCR，即 TIMx_CCR 寄存器的内容

```
5.     uint32_t OCPolarity;
```

// 设置通道输出有效电平的极性

```
6.     uint32_t OCNPolarity;
```

// 设置互补通道输出有效电平的极性（用于互补输出）

```
7.     uint32_t OCFastMode;
```

// 快速输出模式使能，该参数只对 PWM 输出有效

```
8.     uint32_t OCIdleState;
```

// 设置空闲状态下通道输出的电平状态（用于互补输出）

```
9.     uint32_t OCNIdleState;
```

// 设置空闲状态下互补通道输出的电平状态（用于互补输出）

```
10. } TIM_OC_InitTypeDef;
```

输出比较模式

成员变量OCMode的取值范围

宏常量定义	含义
TIM_OC_MODE_TIMING	输出比较冻结模式，匹配时无通道输出
TIM_OC_MODE_ACTIVE	匹配时设置通道输出为有效电平
TIM_OC_MODE_INACTIVE	匹配时设置通道输出为无效电平
TIM_OC_MODE_TOGGLE	匹配时设置通道输出电平翻转
TIM_OC_MODE_PWM1	PWM输出模式1
TIM_OC_MODE_PWM2	PWM输出模式2
TIM_OC_MODE_FORCED_ACTIVE	不进行匹配，强制通道输出为有效电平
TIM_OC_MODE_FORCED_INACTIVE	不进行匹配，强制通道输出为无效电平

成员变量OCPolarity的取值范围

宏常量定义	含义
TIM_OCPOLARITY_HIGH	输出有效电平为高电平
TIM_OCPOLARITY_LOW	输出有效电平为低电平

PWM模式

PWM输出的两种模式

PWM模式1

递增计数时，当TIMx_CNT（当前计数值） $<$ TIMx_CCR（捕获/比较值）时，通道输出为**有效电平**，否则为无效电平。递减计数模式则刚好相反。

PWM模式2

递增计数时，当TIMx_CNT（当前计数值） $<$ TIMx_CCR（捕获/比较值）时，通道输出为**无效电平**，否则为有效电平。递减计数模式则刚好相反。

递增计数，高电平有效时：

- PWM1模式下的CCR用于控制高电平持续的时间
- PWM2模式下的CCR用于控制低电平持续的时间

总结：互补输出

成员变量OCFastMode的取值范围

宏常量定义	含义
TIM_OCFAST_DISABLE	不使能快速输出模式
TIM_OCFAST_ENABLE	使能快速输出模式

可以加快触发输入事件对通道输出的影响，默认配置为不使能

1 定时器PWM输出启动函数：HAL_TIM_PWM_Start

接口函数：HAL_TIM_PWM_Start

函数原型	HAL_StatusTypeDef HAL_TIM_PWM_Start (TIM_HandleTypeDef *htim, uint32_t Channel)
功能描述	在轮询方式下启动PWM信号输出
入口参数1	htim：定时器句柄的地址
入口参数2	Channel：定时器通道号，取值范围是TIM_CHANNEL_1 ~ TIM_CHANNEL_4
返回值	HAL状态值
注意事项	1. 该函数在定时器初始化完成之后调用 2. 函数需要由用户调用，用于启动定时器的指定通道输出PWM信号

2 定时器比较/捕获寄存器设置函数：__HAL_TIM_SET_COMPARE

接口函数：__HAL_TIM_SET_COMPARE	
函数原型	__HAL_TIM_SET_COMPARE (__HANDLE__, __CHANNEL__, __COMPARE__)
功能描述	设置捕获/比较寄存器TIMx_CCR的值。在PWM输出时，用于改变PWM信号的占空比
参数1	__HANDLE__ ：定时器句柄的地址
参数2	__CHANNEL__ ：定时器通道号，取值范围是TIM_CHANNEL_1 ~ TIM_CHANNEL_4
参数3	__COMPARE__ ：写入捕获/比较寄存器TIMx_CCR的值
返回值	无
注意事项	1. 该函数是宏函数，进行宏替换，不发生函数调用 2. 函数需要由用户调用，用于PWM输出时，改变PWM信号的占空比



电子科技大学
University of Electronic Science and Technology of China

3 基础任务：输出PWM信号

基础任务

输出PWM信号

01

任务目标

掌握CubeMX软件配置定时器输出PWM信号的方法。

02

任务内容

产生周期为200ms，占空比为50%的PWM信号来控制Nucleo开发板上的用户指示灯LD2。

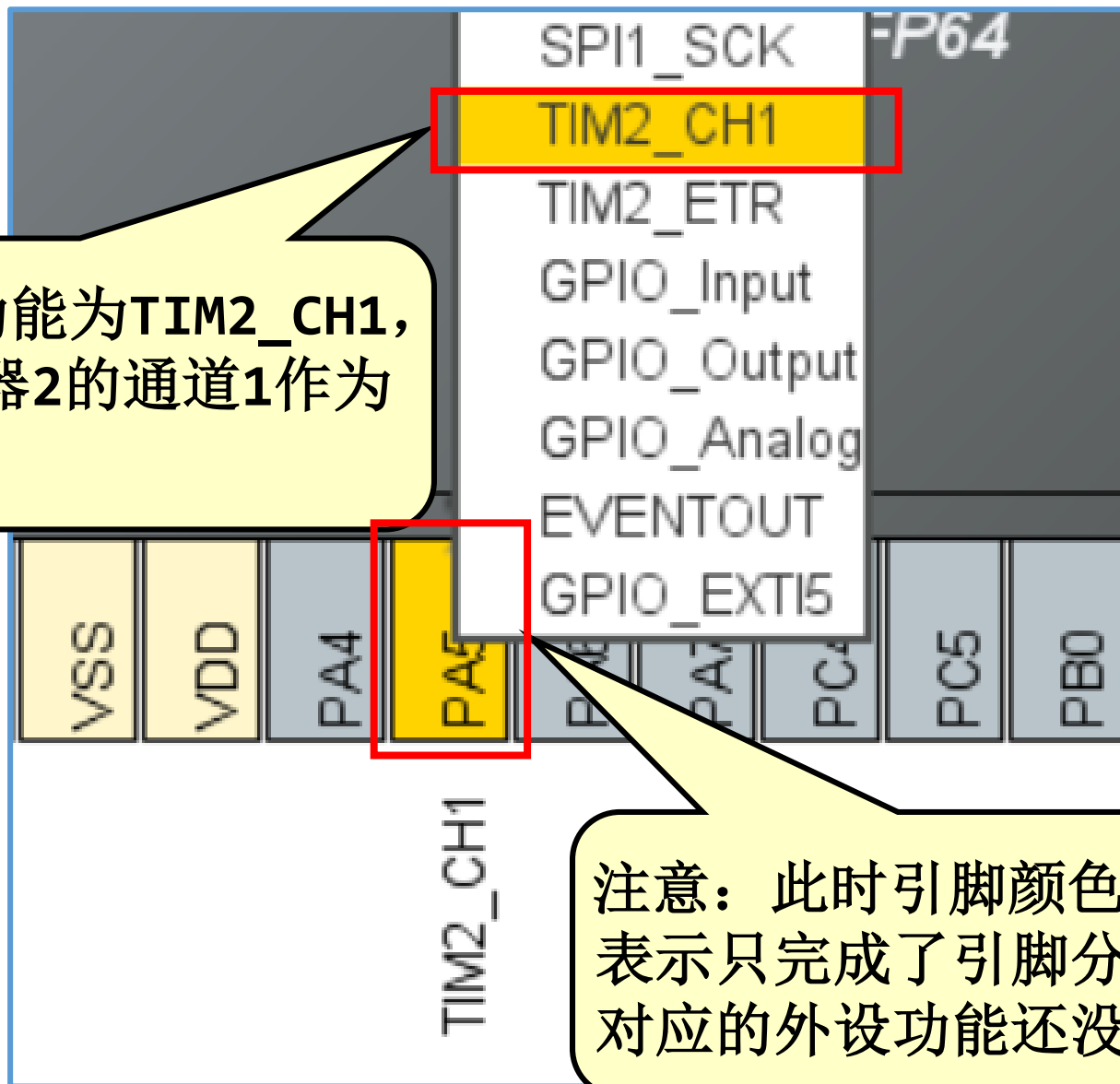
设计思路

设计思路

1. 引脚PA5的复用功能中与定时器输出功能相关的是TIM2_CH1, 因此使用定时器2的通道1输出PWM信号;
2. 使用定时器2, 挂接在APB2总线上, 定时器时钟TIM2_CLK为100MHz;
3. PWM周期为200ms, 可以假设PSC为9999, 根据公式可以计算出ARR的值为1999;
4. 占空比为50%, 则CCR为1000。

引脚分配

选择PA5引脚功能为TIM2_CH1，
表示利用定时器2的通道1作为
PWM输出

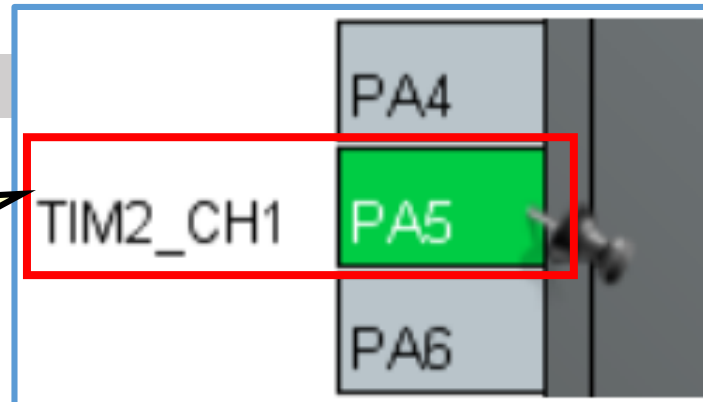


注意：此时引脚颜色为橙色，
表示只完成了引脚分配，但
对应的外设功能还没有使能

Timer

外设配置

引脚PA5变为亮绿色，
表示完成PWM引脚配置



Categories A->Z

Timers

- RTC
- TIM1
- ✓ TIM2
- TIM3
- TIM4
- TIM5

Mode

Slave Mode Disable

Trigger Source

Clock Source Internal Clock

Channel1 PWM Generation CH1

Channel2 Disable

Channel3 Disable

选择时钟源为内部时钟

选择通道1为PWM输出模式

外设配置

配置时基单元

Parameter Settings

Counter Settings

Prescaler (PSC - 16 bits ... 9999

Counter Mode Up

Counter Period (AutoRelo... 1999

Internal Clock Division (C... No Division

auto-reload preload Disable

设置预分频系数PSC为9999，自动重载值ARR为1999，表示PWM信号的周期为200ms

其他参数采用默认配置：

- 使用PWM1模式
- 使能CCR寄存器的预装载功能
- 关闭快速输出模式
- 输出有效电平为高电平

配置PWM输出通道

P

PWM Generation

Mode PWM mode 1

Pulse (32 bits value) 1000

Output compare preload Enable

Fast Mode Disable

CH Polarity High

设置捕获/比较值CRR为1000
表示PWM信号的占空比为50%

注意：在大多数情况下，选择开启CCR寄存器的预装载功能，让占空比的变化在下一个PWM信号周期才生效

步骤六：程序编写

main.c

```
/* USER CODE BEGIN 2 */
```

```
//启动定时器2的通道1输出周期为200ms，占空比为50%的PWM信号
```

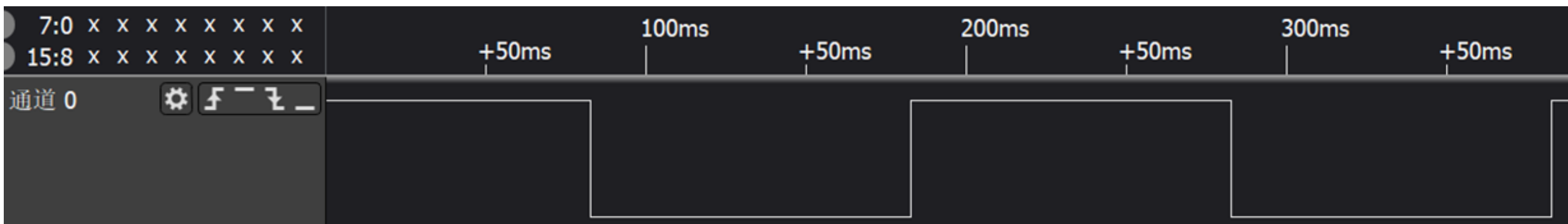
```
HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1);
```

```
/* USER CODE END 2 */
```

运行结果

程序运行结果

指示灯LD2每隔100ms闪烁一次，通过逻辑分析仪可以得到如下波形：



PWM信号参数:

▼ 测量

脉宽: 100.02444ms
周期: 200.04887ms
占空比: 49.999975%
频率: 4.99877855Hz



电子科技大学
University of Electronic Science and Technology of China

4 进阶任务：实现呼吸灯

进阶任务

实现呼吸灯

01

任务目标

掌握CubeMX软件配置定时器输出PWM信号的方法。

02

任务内容

利用人眼的视觉暂留效果，扫描频率需大于50Hz，以避免闪烁

利用PWM信号控制Nucleo开发板上的指示灯LD2。设置PWM周期为20ms，占空比从0%开始，步进为20%。递增至100%后，又从0%开始，并重复整个过程。占空比修改的时间间隔为100ms。

设计思路

设计思路

1. 定义2个变量：占空比Duty和步进值Step。占空比从0%逐次步进到100%，步进比例为20%，步进时间间隔为100ms。PWM信号的周期为20ms，定时器2的定时器时钟TIM2_CLK为100MHz，根据公式可以设置预分频系数PSC为9999，自动重载值ARR为199；
2. 占空比从0%开始，因此写入捕获/比较寄存器CCR的初值为0，然后在while循环中调用宏函数HAL_TIM_SET_COMPARE修改CCR的内容，从0开始，逐渐增加到200，步进值为40。并重复该过程。

外设配置

配置定时器2

配置时基单元

配置PWM输出通道

Parameter Settings

Counter Settings

Prescaler (PSC - 16 bits ... 9999

Counter Mode Up

Counter Period (AutoRelo... 199

Internal Clock Division (C... No Division

auto-reload preload Disable

设置预分频系数PSC为9999，自动重载值ARR为199，表示PWM信号的周期为20ms

Parameter Settings

PWM Generation Channel 1

Mode PWM mode 1

Pulse (32 bits value) 0

Output compare preload Enable

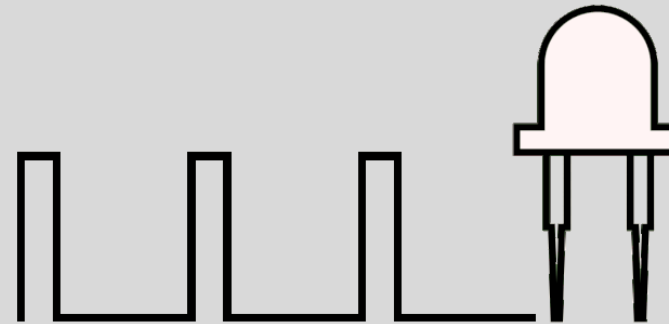
Fast Mode Disable

CH Polarity High

设置捕获/比较值CRR为0，表示PWM信号最初的占空比为0%

程序编写

```
1. /* USER CODE BEGIN PV */
2. uint16_t Duty = 0;           // 占空比
3. uint16_t Step = 40;          // 步进值
4. /* USER CODE END PV */
5. /* USER CODE BEGIN 2 */
6. HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1); //启动定时器2的通道1输出PWM信号
7. /* USER CODE END 2 */
8. while (1)
9. {
10. /* USER CODE BEGIN 3 */
11. for( Duty = 0 ; Duty <= 200 ; Duty = Duty + Step )
12. {
13.     __HAL_TIM_SET_COMPARE(&htim2,TIM_CHANNEL_1,Duty); // 修改占空比
14.     HAL_Delay(100); // 延时100ms
15. }
16. }
17. /* USER CODE END 3 */
```



练习任务

1

任务：实现双向呼吸灯

修改程序，实现指示灯从暗到亮又从亮到暗的渐变，并重复该过程。

提示：在while循环增加一个for循环，循环初值为200，终值为0，步进值为-40。



电子科技大学
University of Electronic Science and Technology of China

8.5 定时器的输入捕获功能

1 输入捕获功能概述

用途与原理

输入捕获功能的用途和工作原理

用途

用于测量信号的参数，比如周期和频率。

工作原理

在输入捕获模式下，当捕获单元捕捉到外部信号的有效边沿（上升沿/下降沿/双边沿）时，将计数器的当前值锁存到捕获/比较寄存器TIMx_CCR，供用户读取。

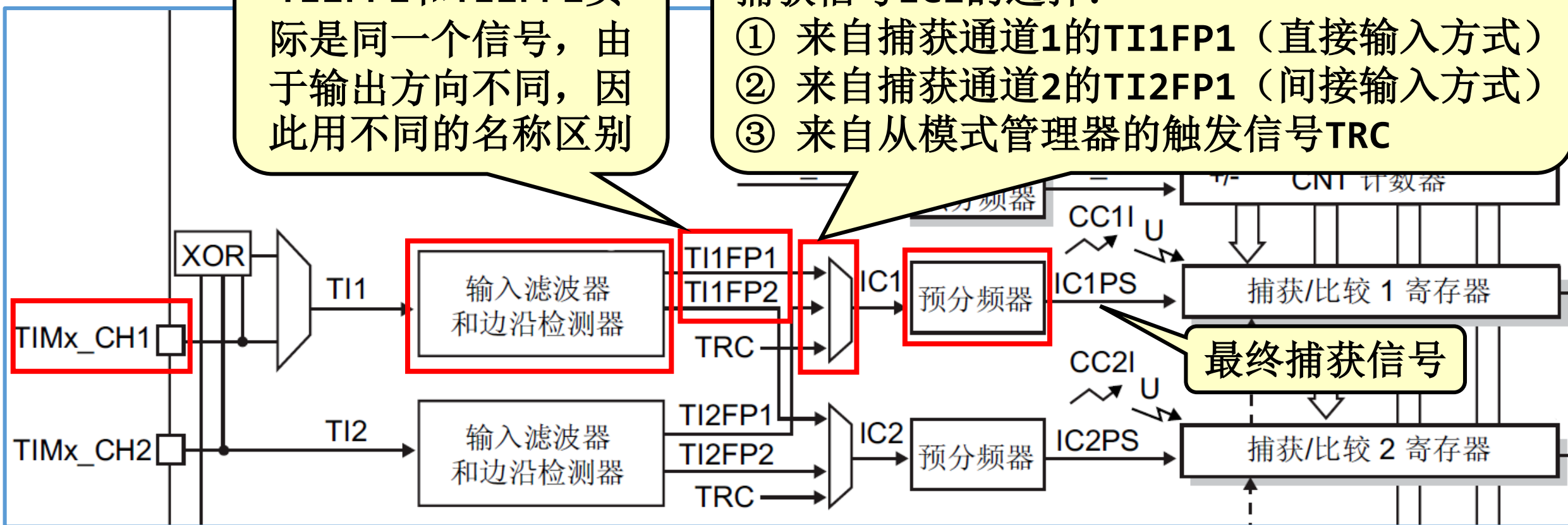
捕获通道结构

捕获通道的内部结构

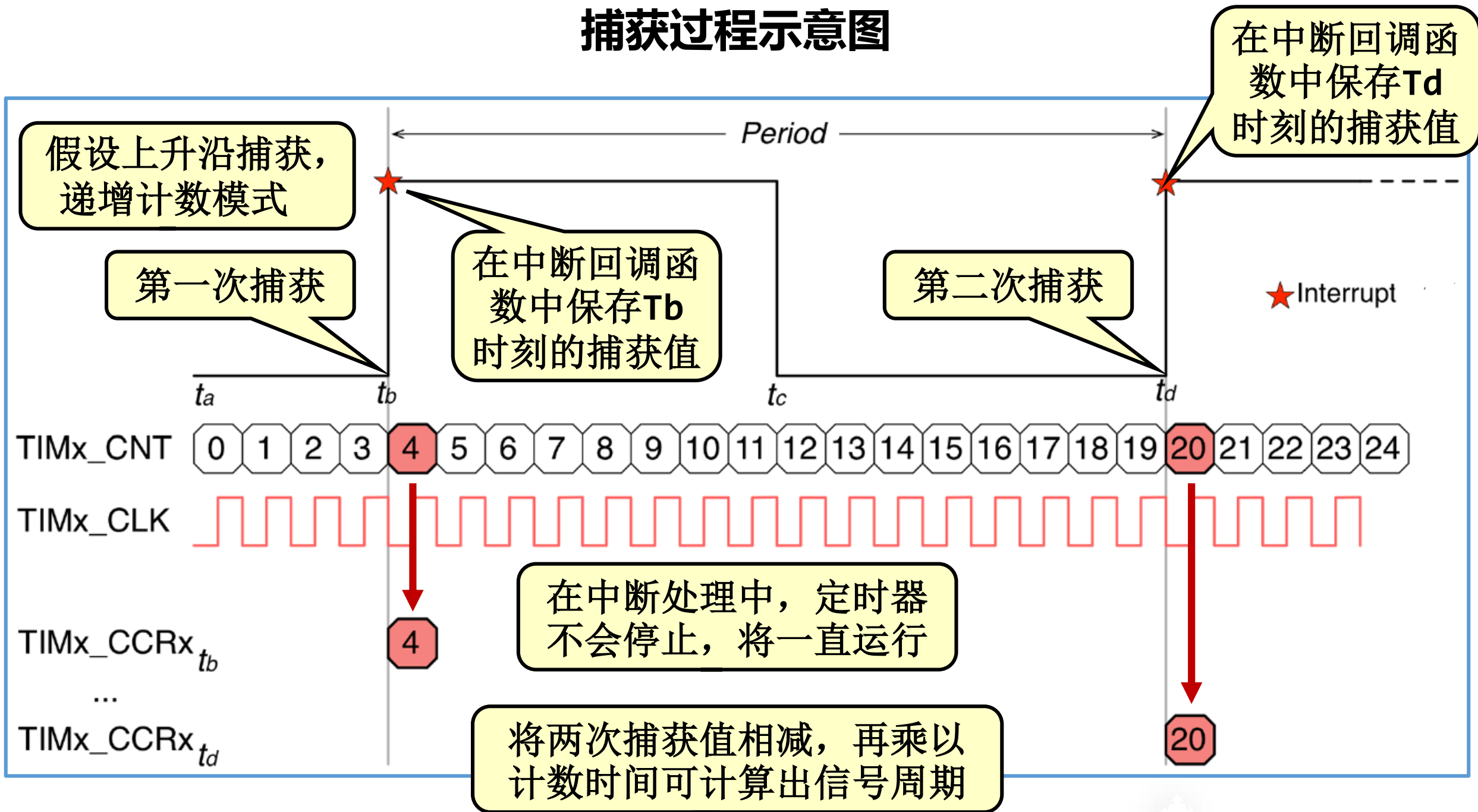
TI1FP1和TI1FP2实际是同一个信号，由于输出方向不同，因此用不同的名称区别

捕获信号IC1的选择：

- ① 来自捕获通道1的TI1FP1（直接输入方式）
- ② 来自捕获通道2的TI2FP1（间接输入方式）
- ③ 来自从模式管理器的触发信号TRC



捕获过程示意图



计算公式

信号参数计算公式

捕获差值

定时器时钟

预分频系数

$$\text{Period(s)} = \text{Diff} / \text{TIMx_CLK} \times (\text{PSC} + 1)$$

$$\text{Freq(Hz)} = \text{TIMx_CLK} / (\text{Diff} \times (\text{PSC} + 1))$$

当待测信号不大于定时器的一个完整计数周期（从0到ARR）时。假设两次连续的捕获值分别为CCR_{x_1}和CCR_{x_2}，则捕获差值可以按照如下方法计算：

- 如果 CCR_{x_1} < CCR_{x_2}：捕获差值 = CCR_{x_2} - CCR_{x_1}
- 如果 CCR_{x_1} > CCR_{x_2}：捕获差值 = (ARR + 1 - CCR_{x_1}) + CCR_{x_2}

注意：如果待测信号大于定时器的一个完整计数周期，则需要结合定时器的更新中断次数来计算捕获差值。



电子科技大学
University of Electronic Science and Technology of China

2 输入捕获功能的数据类型和接口函数

输入捕获单元初始化类型

```
1. typedef struct
```

```
2. {
```

```
3.     uint32_t ICPolarity;
```

```
4.     uint32_t ICSelection;
```

```
5.     uint32_t ICPrescaler;
```

```
6.     uint32_t ICFilter;
```

```
7. } TIM_IC_InitTypeDef;
```

结构体类型，包括4个成员变量

// 设置输入信号的捕获边沿

// 选择输入捕获通道

// 设置输入信号的预分频系数，取值为 1/2/4/8

// 设置输入信号的滤波器长度，取值范围 0x0~0xF

成员变量ICPolarity的取值范围

宏常量定义	含义
TIM_ICPOLARITY_RISING	上升沿捕获
TIM_ICPOLARITY_FALLING	下降沿捕获
TIM_ICPOLARITY_BOTHEDGE	双边沿捕获

成员变量ICSelection的取值范围

宏常量定义	含义
TIM_ICSELECTION_DIRECTT1	直接输入模式： 捕获通道CH1和CH2分别与IC1和IC2连接 捕获通道CH3和CH4分别与IC3和IC4连接
TIM_ICSELECTION_DIRECTT2	间接输入模式： 捕获通道CH1和CH2分别与IC2和IC1连接 捕获通道CH3和CH4分别与IC4和IC3连接
TIM_ICSELECTION_TRC	选择从模式管理器的触发信号TRC 作为捕获信号

成员变量ICPrescaler的取值范围

宏常量定义	含义
TIM_ICPSC_DIV1	检测到输入信号的每1个有效边沿触发1次捕获
TIM_ICPSC_DIV2	检测到输入信号的每2个有效边沿触发1次捕获
TIM_ICPSC_DIV3	检测到输入信号的每4个有效边沿触发1次捕获
TIM_ICPSC_DIV4	检测到输入信号的每8个有效边沿触发1次捕获

1 输入捕获启动函数：HAL_TIM_IC_Start_IT

接口函数：HAL_TIM_IC_Start_IT

函数原型	uint32_t HAL_TIM_IC_Start_IT (TIM_HandleTypeDef *htim, uint32_t Channel)
功能描述	用于在中断方式下启动定时器的输入捕获功能
入口参数1	*htim: 定时器句柄的地址
入口参数2	Channel: 定时器通道号, 取值范围是TIM_CHANNEL_1 ~ TIM_CHANNEL_4
返回值	HAL状态值
注意事项	1. 该函数在定时器初始化完成之后调用 2. 函数需要由用户调用, 用于使能定时器的捕获中断, 并启动定时器运行

2 输入捕获停止函数：HAL_TIM_IC_Stop_IT

接口函数：HAL_TIM_IC_Stop_IT	
函数原型	HAL_StatusTypeDef HAL_TIM_IC_Stop_IT (TIM_HandleTypeDef *htim, uint32_t Channel)
功能描述	用于停止中断模式下的定时器输入捕获功能
入口参数1	*htim: 定时器句柄的地址
入口参数2	Channel: 定时器通道号, 取值范围是TIM_CHANNEL_1 ~ TIM_CHANNEL_4
返回值	HAL状态值
注意事项	1. 该函数需要由用户调用, 用于禁止捕获中断, 关闭输入捕获通道, 停止定时器运行

3 输入捕获中断回调函数HAL_TIM_IC_CaptureCallback

接口函数：HAL_TIM_IC_CaptureCallback

函数原型	void HAL_TIM_IC_CaptureCallback (TIM_HandleTypeDef *htim)
功能描述	用于处理所有定时器的输入捕获中断，用户在该函数内编写实际的任务处理程序
入口参数	*htim：定时器句柄的地址
返回值	无
注意事项	<ol style="list-style-type: none">1. 该函数由定时器中断通用处理函数HAL_TIM_IRQHandler调用，完成所有定时器输入捕获中断的任务处理2. 函数内部需要根据定时器句柄的实例来判断是哪一个定时器的哪一个通道产生的本次输入捕获中断3. 函数由用户根据具体的处理任务编写

4 捕获值读取函数：HAL_TIM_ReadCapturedValue

接口函数：HAL_TIM_ReadCapturedValue

函数原型	uint32_t HAL_TIM_IC_Start_IT (TIM_HandleTypeDef *htim, uint32_t Channel)
功能描述	用于读取捕获/比较寄存器TIMx_CCR的值，即捕获值
入口参数1	*htim：定时器句柄的地址
入口参数2	Channel：定时器通道号，取值范围是TIM_CHANNEL_1 ~ TIM_CHANNEL_4
返回值	捕获值
注意事项	1. 函数需要由用户调用，用于读取发生捕获时的捕获值

5 捕获边沿设置函数：__HAL_TIM_SET_CAPTUREPOLARITY

接口函数：__HAL_TIM_SET_CAPTUREPOLARITY	
函数原型	__HAL_TIM_SET_CAPTUREPOLARITY(__HANDLE__, __CHANNEL__, __POLARITY__
功能描述	用于设置输入信号的捕获边沿
参数1	__HANDLE__ ： 定时器句柄的地址
参数2	__CHANNEL__ ： 定时器通道号，取值范围是TIM_CHANNEL_1 ~ TIM_CHANNEL_4
参数3	__POLARITY__ ： 捕获边沿，取值范围是 TIM_INPUTCHANNELPOLARITY_RISING/FALLING/BOTHEDGE
返回值	无
注意事项	1. 该函数是宏函数，进行宏替换，不发生函数调用 2. 函数需要由用户调用，用于设置输入信号的有效捕获边沿

3 基础任务：信号测量

基础任务

信号测量

01

任务目标

掌握CubeMX软件配置定时器输入捕获功能的方法。

02

任务内容

利用定时器2的通道1 (对应引脚PA0)来测量一个外部脉冲信号的周期和频率，外部脉冲信号利用引脚PA6输入。

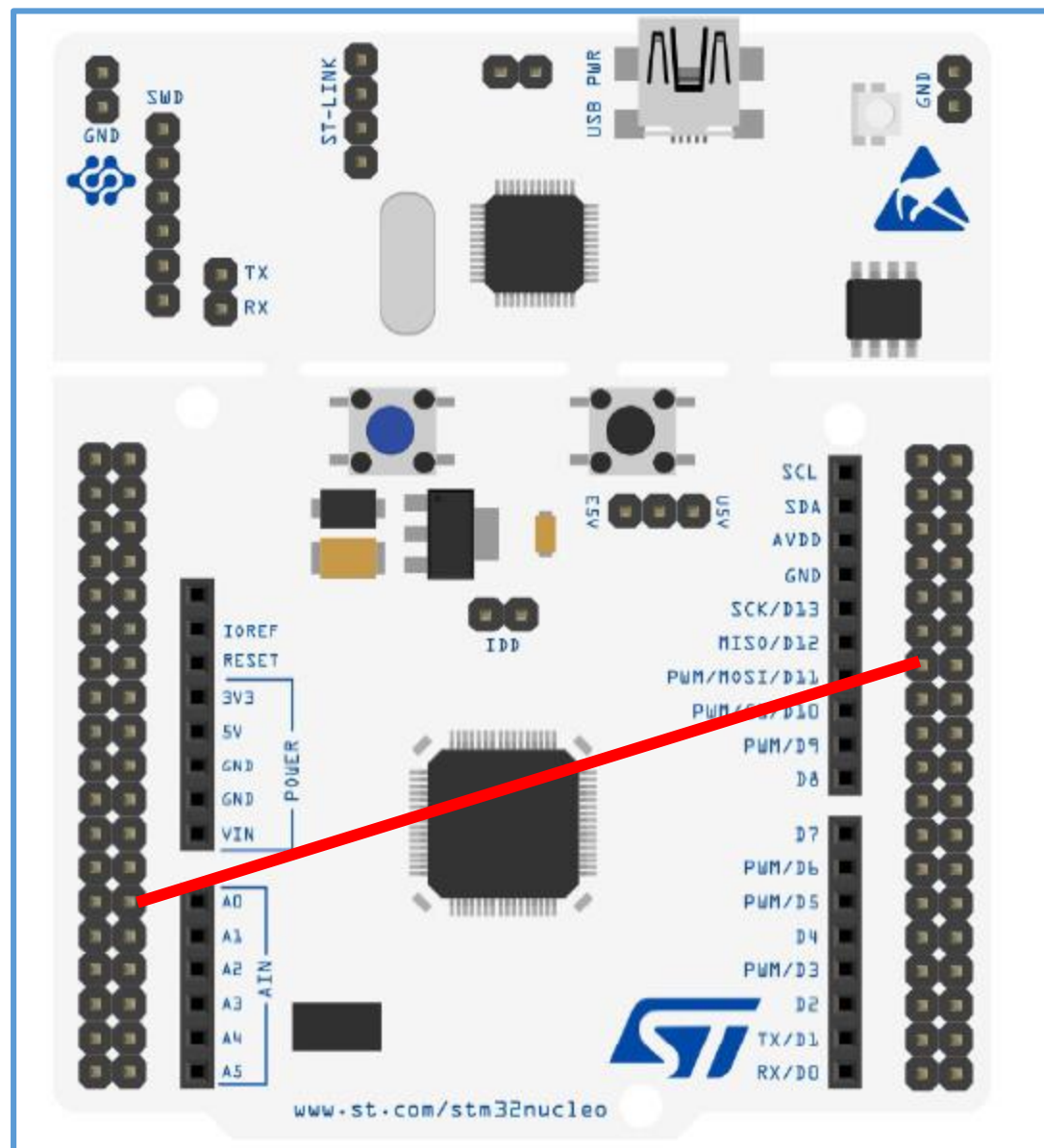
设计思路

设计思路

1. 配置定时器2的通道1为输入捕获模式，采用上升沿触发。不设置预分频系数，计数时钟CK_CNT的频率为100MHz；
2. 设置引脚PA6为定时器3的通道1（TIM3_CH1），利用定时器3的PWM输出功能，输出一个频率为100KHz，占空比50%的方波；
3. 采用前后台编程模式。在输入捕获中断的回调函数中，设置测量完成标志，主程序中检测该标志，一旦置位则计算信号的周期和频率，并清除标志位。

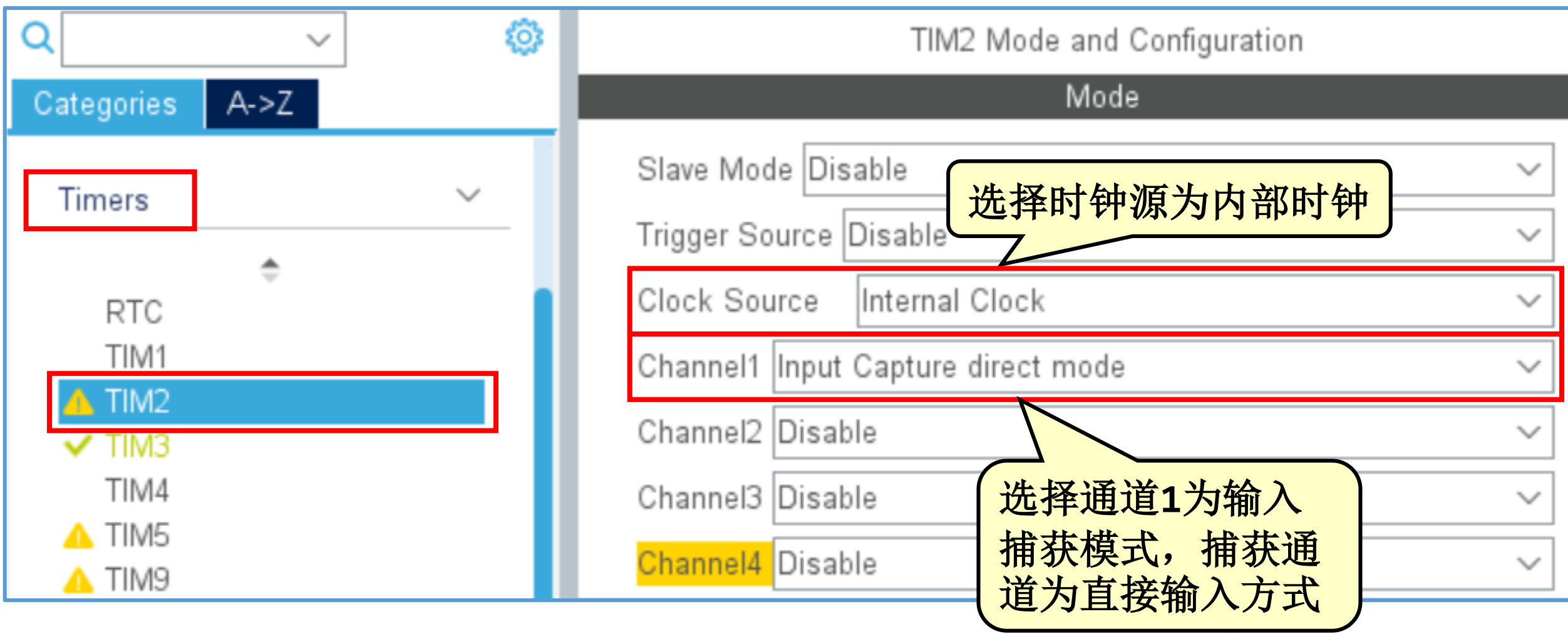
硬件连接图

利用杜邦线将PA6
和PA0连接，连接
示意图如图所示



外设配置

配置定时器2



The screenshot displays the STM32CubeMX configuration interface for the TIM2 timer. The left sidebar shows the 'Timers' category selected, with TIM2 highlighted. The main panel shows the 'TIM2 Mode and Configuration' settings.

Configuration Details:

- Slave Mode:** Disable
- Trigger Source:** Disable
- Clock Source:** Internal Clock (Selected)
- Channel1:** Input Capture direct mode (Selected)
- Channel2:** Disable
- Channel3:** Disable
- Channel4:** Disable

Callouts:

- 选择时钟源为内部时钟 (Select internal clock as clock source)
- 选择通道1为输入捕获模式，捕获通道为直接输入方式 (Select channel 1 as input capture mode, capture channel as direct input mode)

外设配置

配置定时器2

配置时基单元

预分频系数PSC为0，即计数时钟CK_CNT为100MHz，便于扩大测量频率的范围

Counter Settings

Prescaler (PSC - 16 bits ... 0

Counter Mode Up

Counter Period (AutoRelo... 0xFFFFFFFF

Internal Clock Division (C... No Division

auto-reload preload Disable

设置自动重载值ARR为0xFFFFFFFF，即最长的计数周期，便于扩大测量频率的脉宽

配置输入捕获通道

Parameter Settings

Input Capture Channel 1

Polarity Selection Rising Edge

IC Selection Direct

Prescaler Division Ratio No division

Input Filter (4 bits value) 0

采用默认配置参数

- 捕获有效边沿为上升沿
- 捕获通道为直接输入方式
- 捕获信号不分频
- 捕获信号不进行滤波

中断使能

配置定时器2

✓ NVIC Settings	✓ DMA Settings	✓ GPIO Settings	
✓ Parameter Settings	✓ User Constants		
NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priority
TIM2 global interrupt	✓	0	0

使能定时器2的全局中断
中断优先级使用默认值

外设配置

配置定时器3

The screenshot displays the STM32CubeMX configuration interface for the TIM3 timer. The left sidebar shows the 'Timers' category selected, with TIM3 highlighted. The right pane shows the 'TIM3 Mode and Configuration' settings.

TIM3 Mode and Configuration	
Mode	
Slave Mode	Disable
Trigger Source	Disable
Clock Source	Internal Clock
Channel1	PWM Generation CH1
Channel2	Disable
Channel3	Disable
Channel4	Disable

Callouts in the image:

- 选择时钟源为内部时钟 (Select internal clock as clock source) - points to Clock Source: Internal Clock
- 选择通道1为PWM输出 (Select channel 1 as PWM output) - points to Channel1: PWM Generation CH1

外设配置

配置定时器3

配置时基单元

配置PWM输出通道

Parameter Settings

Counter Settings

Prescaler (PSC - 16 bits ... 99

Counter Mode Up

Counter Period (AutoRelo... 9

Internal Clock Division (C... No Division

auto-reload preload Disable

设置预分频系数PSC为99，自动重载值ARR为9，表示PWM信号的周期为10us

Parameter Settings

PWM Generation Channel 1

Mode PWM mode 1

Pulse (16 bits value) 5

Output compare preload Enable

Fast Mode Disable

CH Polarity High

设置捕获/比较值CRR为5，表示PWM信号的占空比为50%

程序编写

用户变量定义

```
1. /* USER CODE BEGIN PV */
2. uint32_t Diff          = 0 ; // 存放捕获差值
3. uint8_t  MeasureFlag   = 0 ; // 测量完成标志: 0表示未完成, 1表示完成
4. uint8_t  CapIndex       = 0 ; // 捕获指示: 0表示没有开始捕获, 1表示完成一次捕获
5. uint32_t CapVal1        = 0 ; // 存放第一次捕获值
6. uint32_t CapVal2        = 0 ; // 存放第二次捕获值
7. /* USER CODE END PV */
```

用户初始化代码

```
1. /* USER CODE BEGIN 2 */
2. printf ("Timer Capture Function Test: \n"); // 发送提示信息
3. HAL_TIM_IC_Start_IT(&htim2, TIM_CHANNEL_1); // 启动定时器2通道1的输入捕获功能
4. HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_1);    // 启动定时器3通道1输出PWM信号
5. /* USER CODE END 2 */
```


用户应用代码

```
1.  if( MeasureFlag == 1)           // 判断测量标志是否置位
2.  {
3.      if(CapVal2 >= CapVal1)       // 两次捕获在同一个计数周期内
4.      {
5.          Diff = CapVal2 - CapVal1;
6.      }
7.      else                         // 两次捕获不在同一个计数周期内
8.      {
9.          Diff = ((0xFFFFFFFF + 1 - CapVal1) + CapVal2) ;
10.     }
11.     printf ("Period      is: %.4fms\r\n",Period/100000.0); // 计算信号周期
12.     printf ("Frequency is: %dHz\r\n",100000000/Period); // 计算信号频率
13.     printf ("/*****\r\n");
14.     MeasureFlag = 0; // 清除测量完成标志
15.     HAL_Delay(1000);
16.     HAL_TIM_IC_Start_IT(&htim2, TIM_CHANNEL_1); // 启动下一次捕获
17. }
```

添加位置

USER CODE BEGIN 3

USER CODE END 3

捕获中断回调函数

```
1. void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim)
2. {
3.     if( htim->Instance == TIM2 )    // 判断发生捕获中断的定时器
4.     {
5.         if( htim->Channel == HAL_TIM_ACTIVE_CHANNEL_1 ) // 判断发生捕获中断的通道
6.         {
7.             if(CapIndex == 0)        // 存放第一次捕获时的CCR 值
8.             {
9.                 CapVal1 = HAL_TIM_ReadCapturedValue(htim, TIM_CHANNEL_1);
10.                CapIndex = 1 ;        // 修改捕获指示
11.            }
```

添加位置

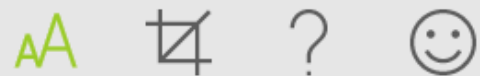
USER CODE BEGIN 4

USER CODE END 4

捕获中断回调函数

```
12.     else if(CapIndex == 1)           // 存放第二次捕获时的 CCR 值
13.     {
14.         CapVal2 = HAL_TIM_ReadCapturedValue(htim, TIM_CHANNEL_1);
15.         HAL_TIM_IC_Stop_IT(htim, TIM_CHANNEL_1); // 暂停捕获
16.         CapIndex    = 0;               // 重置捕获指示
17.         MeasureFlag = 1;               // 置位测量完成标志
18.     }
19.     else
20.     {
21.         Error_Handler();               // 出错指示
22.     }
23. }
24. }
25. }
```

程序运行结果



串口号：



COM15



波特率：



115200



数据位：

8



校验位：

None



停止位：

One



关闭串口

接收区设置.



接收并保存到文件



十六进制显示



暂停接收显示



自动断帧

?

20



接收脚本



Add Timesta



Timer Capture Function Test:

Period is: 0.0100ms

Frequency is: 100000Hz

/*****/

Period is: 0.0100ms

Frequency is: 100000Hz

/*****/



发送：0

接收：3674

复位计数

