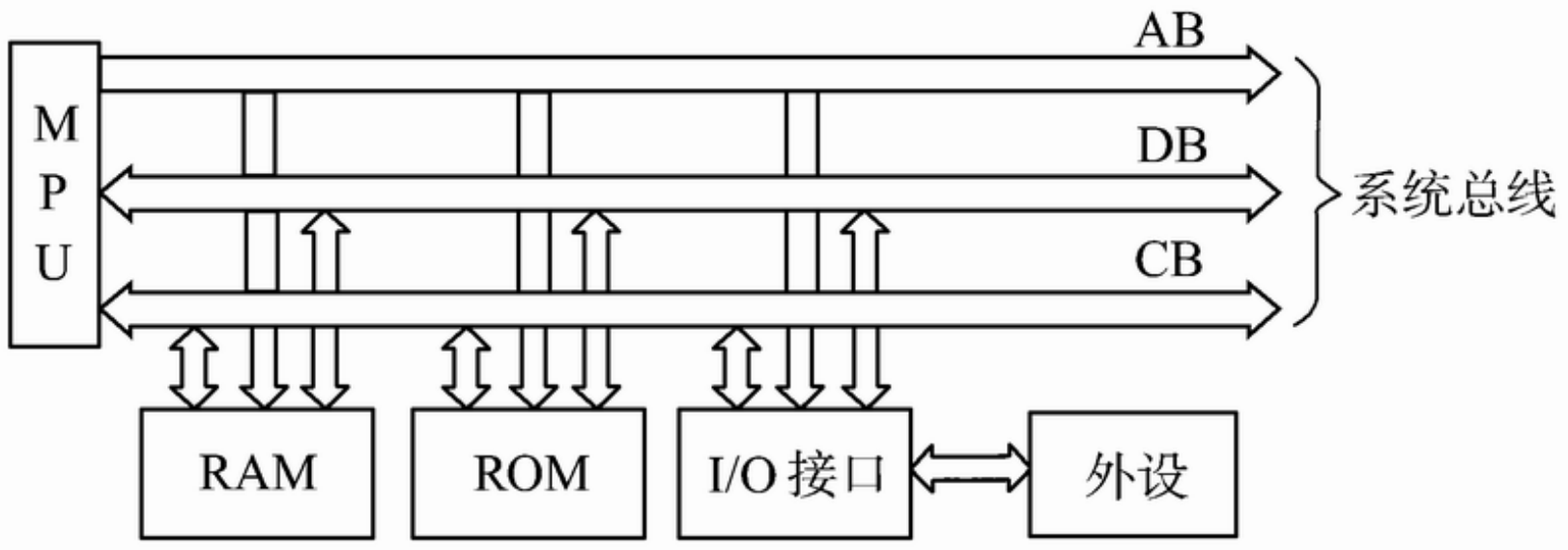
微机原理与接口技术

第八章 接口技术

第八章 接口技术

* 第一节 I/O接口概述
* 第二节 CPU与外设传送数据方式
* 第三节 并行口与串行口
* 第四节 GPIO 模块
* 第五节 定时器/计数器
* 第六节 数模转换器和模数转换器

第一节 I/O接口概述



* 接口（interface）： 是介于微处理器和外设之间的一种缓冲电路,是CPU与外部设备交换信息交换的中转站

第一节 I/O接口概述

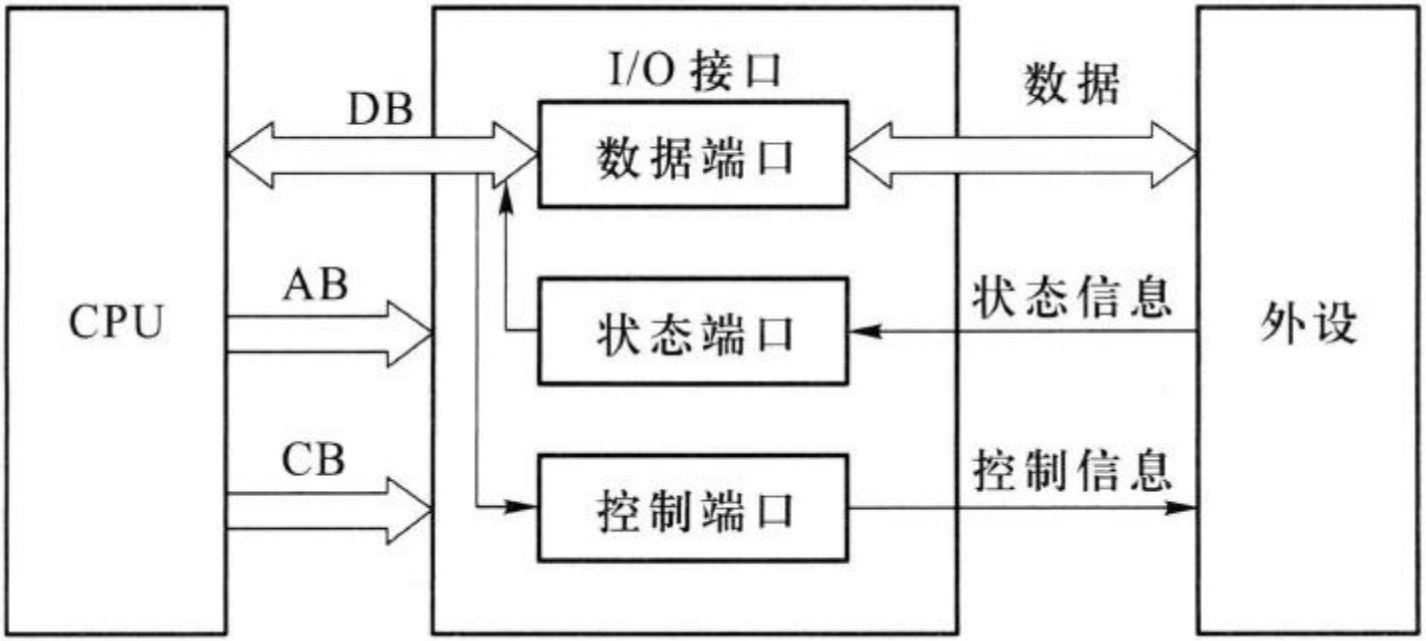
* I/O接口电路的功能:
  * 数据的寄存和缓冲功能
  * 设备选择功能
  * 在微机与外部设备间传送控制和状态信息
  * 信号转换功能
  * 具备时序控制

第一节 I/O接口概述

* CPU与I/O设备之间的信息通常包括：
  数据信息，状态信息和控制信息

1. 数据信息：数字量、模拟量、开关量
2. 状态信息：状态信息一般是外设通过接口送到CPU的；主要是反映外设的工作状态；
3. 控制信息：控制信息一般是CPU通过接口电路传送给外部设备的，主要用来控制外部设备的动作；

第一节 I/O接口概述



I/O 接口示意图

* 系统总线包含有三种不同功能的总线，即数据总线DB（Data Bus）、地址总线AB（Address Bus）和控制总线CB（Control Bus）

*

数据总线DB用于传送数据信息。它既可以把CPU的数据传送到存储器或I/O接口等其它部件，也可以将其它部件的数据传送到CPU。需要指出的是，数据的含义是广义的，它可以是真正的**数据，也可以指令代码或状态信息，有时可以是一个控制信息**，因此，在实际工作中，数据总线上传送的并不一定仅仅是真正意义上的数据。

*

地址总线AB是专门用来传送地址的，地址只能从CPU传向外部存储器或I/O端口。地址总线的位数决定了CPU可直接寻址的内存空间大小，若地址总线为n位，则可寻址空间为2^n字节。

*

控制总线CB用来传送控制信号和时序信号。控制信号中，有的是微处理器送往存储器和I/O接口电路的，如读/写信号，片选信号、中断响应信号等；也有是其它部件反馈给CPU的，比如：中断申请信号、复位信号、总线请求信号、限备就绪信号等。因此，控制总线的传送方向由具体控制信号而定，控制总线的位数要根据系统的实际控制需要而定。实际上控制总线的具体情况主要取决于CPU。

第一节 I/O接口概述

- 为了区分以上所述的三种信息，在接口部件中都包含一组寄存器；CPU和外设交换信息时，将根据三种不同的信息，对不同的寄存器进行读写；
- 这些寄存器就称为端口（PORT）；一个外设往往需要几个端口地址；不同的端口中存放不同的信息；
- 端口主要有三类，状态口，命令口和数据口
- CPU需要访问外设时，寻址的是端口，而不是笼统的外设；

第一节 I/O接口概述

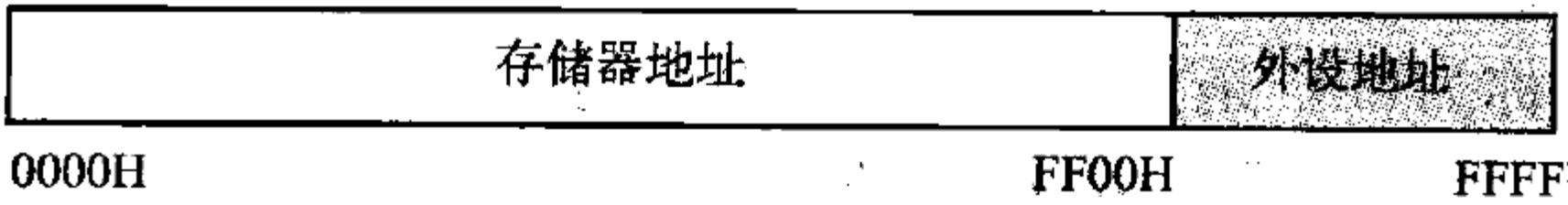
* I/O接口：指CPU和外设间的I/O接口芯片和电路
* I/O端口：常指在接口电路中用以完成某种信息传送，并可由编程人员通过端口地址进行读写的寄存器。
* 一个外设需要一个接口，但一个接口可以有不止一个端口。
* 每个端口对应一个地址，CPU通过不同的地址的访问来区别状态信息，数据信息和命令信息。

第一节 I/O接口概述

* 端口的编址方式

1. 存储器统一编址方式（存储器映像）

  * 在存储空间中划出一部分给外设端口，对端口当作存储器单元一样进行访问，不设置专门的I/O指令；
  * 优点：可以直接使用访问存储器的各种指令访问外设端口，使用方便；不需要专门的输入/输出指令；端口地址安排灵活，数量不限。
  * 缺点：使存储器容量变小等；

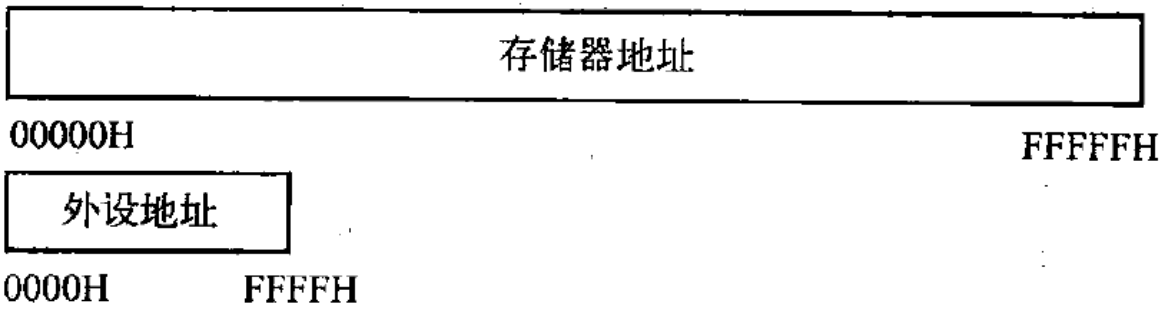


第一节 I/O接口概述

* 端口的编址方式

2. 独立编址方式（I/O映像）

  * 端口地址单独编址，而不和存储空间合在一起；设置专门的I/O指令来访问端口；
  * 特点：外设地址和内存地址是可以重复的；必须区分发出的地址是存储器地址还是外设地址。需要专用的I/O指令。



第二节 CPU与外设传送数据方式

一、无条件传送方式

* 在传送信息时，已知外设是“准备好”的；所以CPU在和外设交换数据时，不用查询外设的状态，直接进行输入/输出；
* 一般用于：
  * 外设是简单设备
  * 外设工作速度足够快
  * 两次数据传送的间隔时间足够长
* 接口电路中一般有输入缓冲器/输出锁存器（数据端口），由地址译码的输出信号来选通它们。

第二节 CPU与外设传送数据方式

二、查询传送方式

* 这种方式，CPU要遵循“先查询，后传送”的原则，保证只有在外设已经是在“准备好”状态，才开始传送数据；
* 接口电路中至少需要两个端口：状态端口和数据端口
* 查询式传送的一般流程
  * 先从状态口读入状态字；
  * 如果状态是“准备好”，开始传送；
  * 如果状态是“没有准备好”，则继续查询，直到“准备好”，开始传送；

第二节 CPU与外设传送数据方式

二、查询传送方式

* 在编写查询式传送程序时，要先确定两个问题：
  * 状态信号的位置：需要规定状态信号是在寄存器中的第几位；
  * 状态信号的有效电平：即是高电平表示准备好，还是低电平表示外设准备好；
* 优点：CPU和外设之间可以很好地配合工作；
* 缺点：CPU要长期地查询外设的状态，查询实际上就是一种等待；CPU长期的等待会影响CPU的工作效率；

第二节 CPU与外设传送数据方式

三、中断传送方式

* 当外设要输入/输出时（即外设已把待输入数据存放在数据输入寄存器，或输出时外设已把上一个数据输出，输出寄存器已空），向CPU发中断请求；
* CPU收到中断请求后，中断当前的工作，为外设服务，服务结束(输入或输出)后，继续原来的工作；
* 中断方式允许CPU与外设(甚至多个外设)同时工作，克服了查询方式的缺点；

第二节 CPU与外设传送数据方式

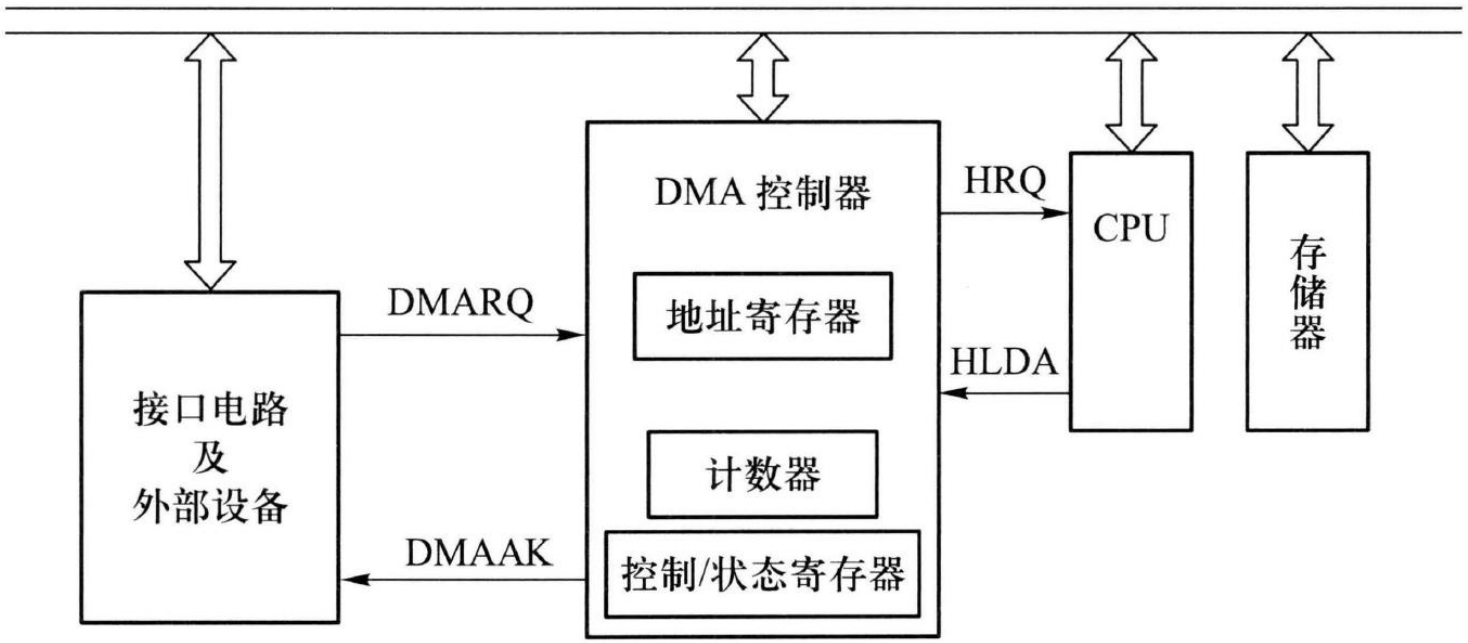
四、直接存储器存取方式（DMA）

* 中断方式仍是CPU通过程序来传送，每次传送一字节；为了要传送这一字节需要将保护断点，保护现场等多条指令执行一遍；
* 对一个高速I/O设备，成组交换数据的情况，中断方式显得不合适；
* DMA方式是由硬件(DMA控制器)来控制数据从外设到存储器的直接传送，而不通过CPU；
* DMA方式下，要求CPU让出这些总线，即要求CPU相应的引脚输出为高阻状态，系统总线由DMA控制器接管这些，产生相应的总线信号；

第二节 CPU与外设传送数据方式

四、直接存储器存取方式（DMA）

* DMA 传送的工作原理：



第二节 CPU与外设传送数据方式

四、直接存储器存取方式（DMA）

DMA 控制器是能在存储器和外部设备之间实现直接而高速地传送数据的一种专用处理器。它应具有独立访问存储器的能力，能取代CPU，提供存储器地址和必要的读写控制信号，将数据总线上的信息写入存储器或从存储器读出，为此，要求DMA 控制器具有独立控制三总线来访问存储器和I/O 端口的能力。

第三节 并行口与串行口

I/O接口的“本质”是电路，这些电路包括：

- I/O端口寄存器
- 地址译码电路
- 传送方式控制电路
- 并-串/串-并变换等转换电路

接口部件，使用本身的寄存器来实现各种接口，这些寄存器就是I/O端口。

接口部件的寄存器---CPU当作RAM单元访问

第三节 并行口与串行口

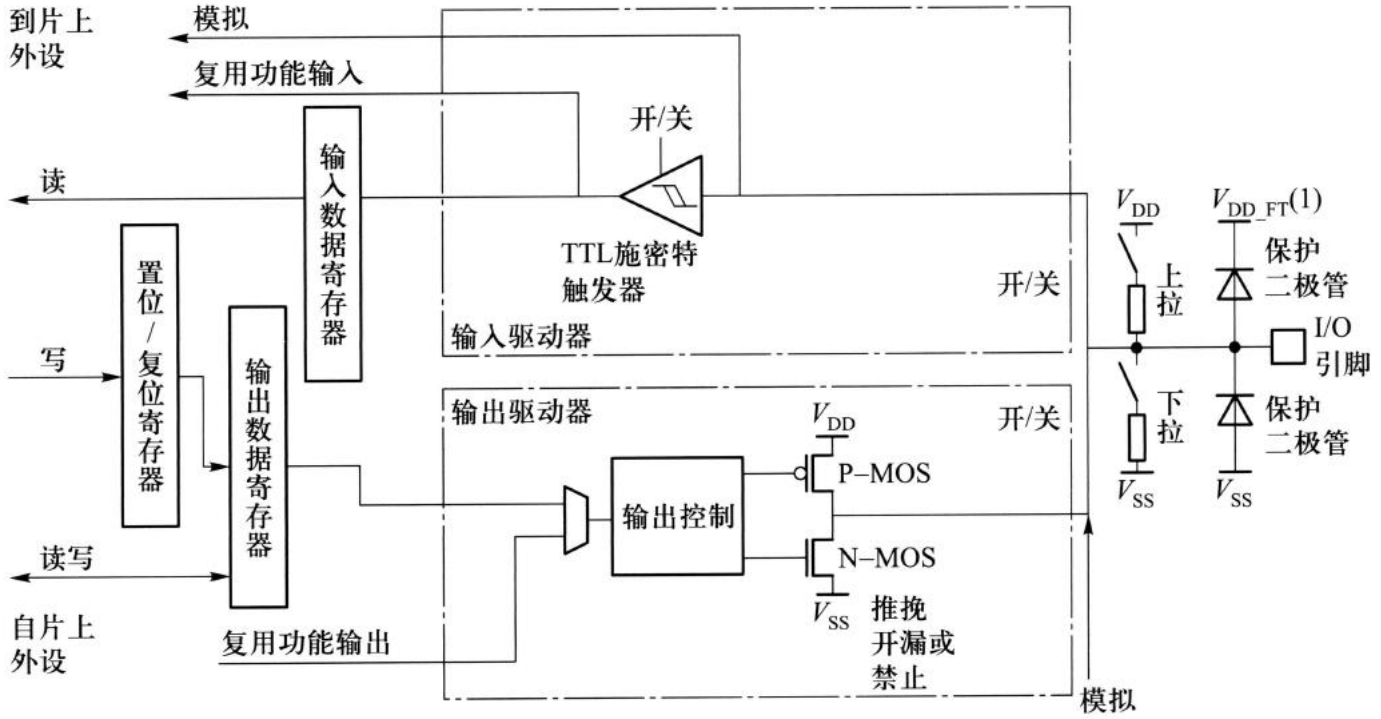
* 并行I/O接口：以并行方式和CPU传送I/O接口，以并行方式和外设交换数据。

* 串行口：数据和控制信息是一位接一位串行按顺序的传送的。因为计算机系统内部的数据是并行传送的，所以要串行传送数据的话，需要进行并---串/串---并变换。

第四节 GPIO 模块

GPIO的基本特性

GPIO的基本结构



第四节 GPIO 模块

GPIO 引脚的工作模式:

(1) 输入模式（上拉／下拉／浮空）

(2) 输出模式（推挽／开漏、上拉／下拉／浮空）

(3) 复用功能（推挽／开湍、上拉／下拉／浮空）

(4) 模拟输入输出模式

第四节 GPIO 模块

STM32H7xxd的GPIO寄存器

- 4个32-bit 配置寄存器(MODER,OTYPER, OSPEEDR PUPDR),
- 2个32-bit 数据寄存器(IDR,ODR)
- 1个32-bit 位操作寄存器 (BSRR).
- 1个32-bit 锁存寄存器 (LCKR)
- 2个32-bit 复用功能选择 (AFRH , AFRL).

第四节 GPIO 模块

GPIO 寄存器：

* 每个GPIO 端口都有4 个32 位存储器映射的配置寄存器，可以用来对GPIO 端口的每个I/O引脚进行设置：

(1) GPIOx_MODER 寄存器用于选择I/O方向及工作模式：输入、输出、AF 、模拟；

(2) GPIOx_OTYPER 寄存器用于选择输出类型：推挽或开漏；

(3) GPIOx _ OSPEEDR 寄存器用于选择输出速度： 2MHz 、25 MHz 、50MHz 、100 MHz ，输出速度是指I/O引脚支持的高、低电平状态的最高切换频率；

(4) GPIOx _ PUPDR 寄存器用于选择：上拉／下拉；

第四节 GPIO 模块

GPIO 寄存器：

* 每个GPIO 端口都具有2个16 位**数据寄存器**：

(1) GPIOx_IDR 寄存器：通过I/O引脚输入的数据存储到输入数据寄存器，它是一个只读寄存器；

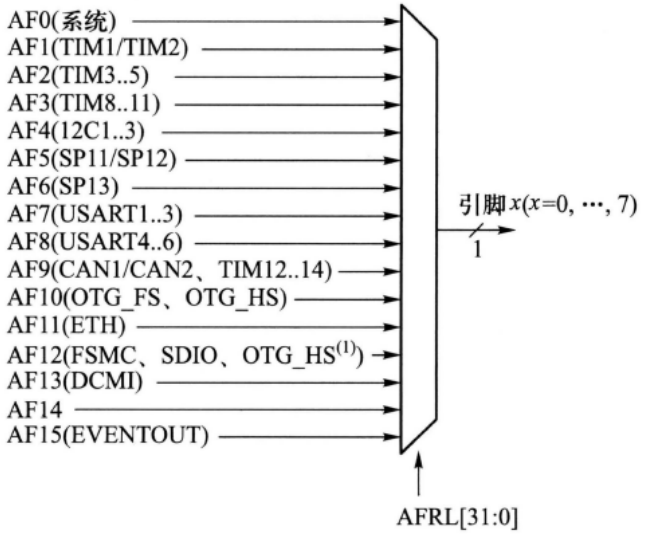
(2) GPIOx_ODR 寄存器：用于存储输出数据，可对其进行读写访问。写入该寄存器的数据直接控制I/O引脚输出高、低电平。

第四节 GPIO 模块

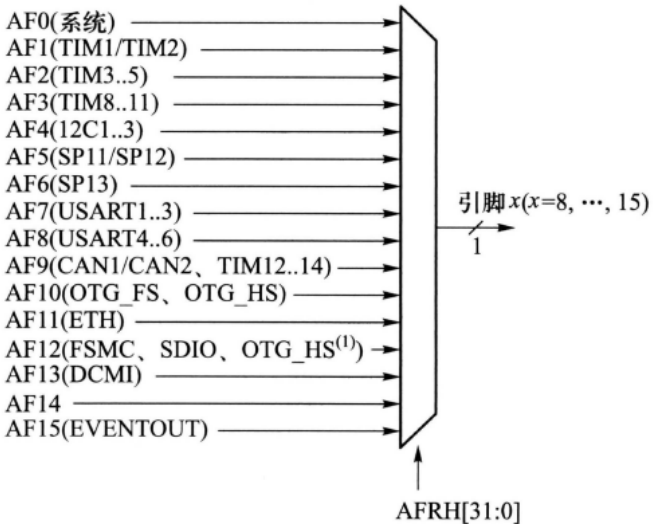
GPIO 寄存器：

* 每个GPIO 端口还有2 个32 位复用功能寄存器GPIOx_AFRL 和GPIOx_AFRH ，当I/O引脚配置为复用功能工作模式时，通过对这两个寄存器编程，可以使得I/O引脚与可用的16个复用功能引脚中的一个连通。

对于引脚0到引脚7，GPIOx_AFRL[31:0]寄存器会选择专用的复用功能



对于引脚8到引脚15，GPIOx_AFRH[31:0]寄存器会选择专用的复用功能



第四节 GPIO 模块

GPIO 寄存器：

* **STM32的I/O接口很多，每个I/O接口都对应着一个时钟。为了降低功耗，当芯片上电时这些时钟都是关闭的。当I/O接口工作时，必须将其对应的时钟使能。STM32 所有外I/O接口的时钟都是由复位时钟控制器(RCC) 来管理的。**

位 8：0：GPIOx 时钟使能，由软件置 1 和清零

0：禁止 GPIOx 时钟

1：使能 GPIOx 时钟

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reser-ved	OTGHSULPIEN	OTGHSEN	ETHMACPTPEN	ETHMACRXEN	ETHMACTXEN	ETHMACEN	Reserved		DMA2EN	DMA1EN	CCMDATARAMEN	Rex.	BKPSRAMEN	Reserved	
	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
Reserved			CRCEN	Reserved			GPIOIEN	GPIOHEN	GPIOGEN	GPIOFEN	GPIOEEN	GPIODEN	GPIOCEN	GPIOBEN	GPIOAEN
			rw				rw	rw	rw	rw	rw	rw	rw	rw	rw

第四节 异步串行通信接口

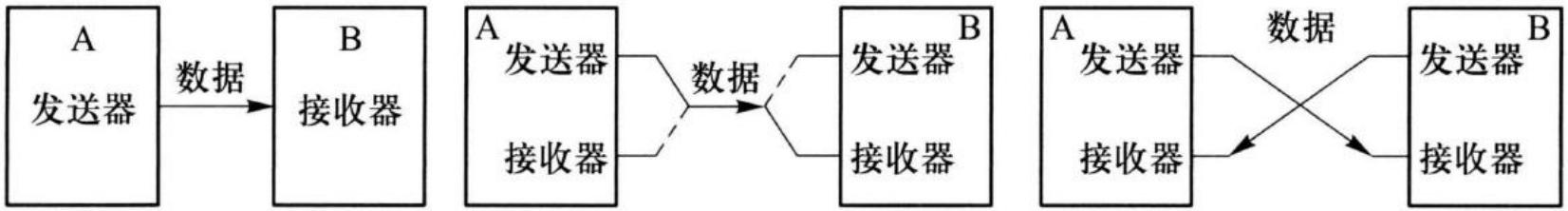
* **串行通信的特点：**
  * 一根线上既要传数据，又要传控制信息
  * 信息格式有固定的要求
  * 串行通信中对信息的逻辑定义与TTL不兼容，要进行逻辑电平转换

（TTL电平标准：
输出 L： <0.8V ，H：>2.4V；输入 L：<1.2V ，H：>2.0V
RS232标准：
逻辑1的电平为-3～-15V，逻辑0的电平为+3～+15V，注意电平的定义反相了）

* **串行通信的分类：**
  * 异步，发送和接收双方使用各自的时钟
  * 同步

第四节 异步串行通信接口

* 数据传送方式



(a) 单工方式　(b) 半双工方式　(c) 全双工方式

第四节 异步串行通信接口

* 起止式异步通信协议

(1) 1 位起始位，规定为低电平0，是字符开始的标志；

(2) 5~8 位数据位，规定低位在前，高位在后；

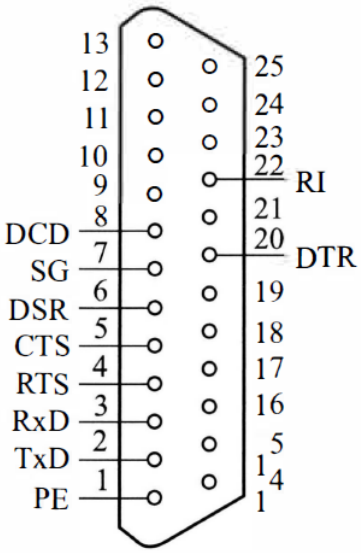
(3) 0 位或1 位奇偶校验位；

(4) 1 位、1.5 位或2 位停止位，规定为高电平“1”，是字符结束的标志。

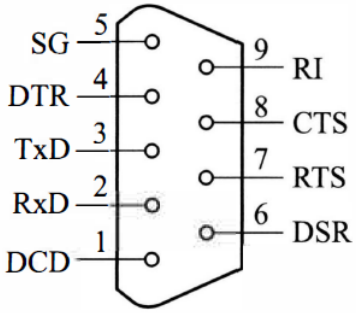
字符与字符之间的空闲位用高电平“1”填充。

第四节 异步串行通信接口

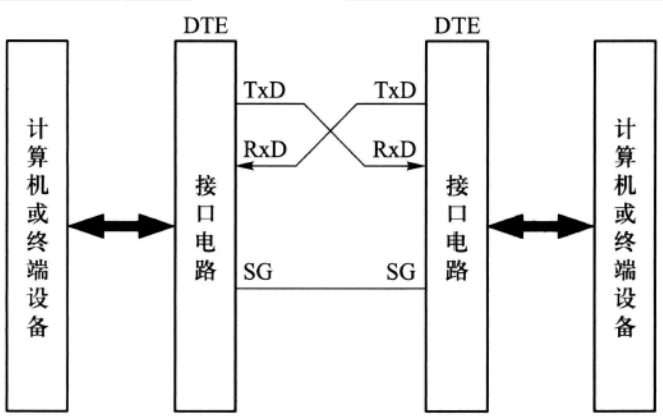
* RS-232C 接口标准



DB－25



DB－9



引脚号（9针）	引脚号（25针）	信号	方向	功能
3	2	TxD	输出	发送数据
2	3	RxD	输入	接收数据
7	4	RTS	输出	发送请求
8	5	CTS	输入	发送清除
6	6	DSR	输入	数据通信设备（DCE）就绪
5	7	SG		信号公共参考地
1	8	DCD	输入	数据载波检测
4	20	DTR	输出	数据终端设备（DTE）就绪
9	22	RI	输入	振铃指示

第四节 异步串行通信接口

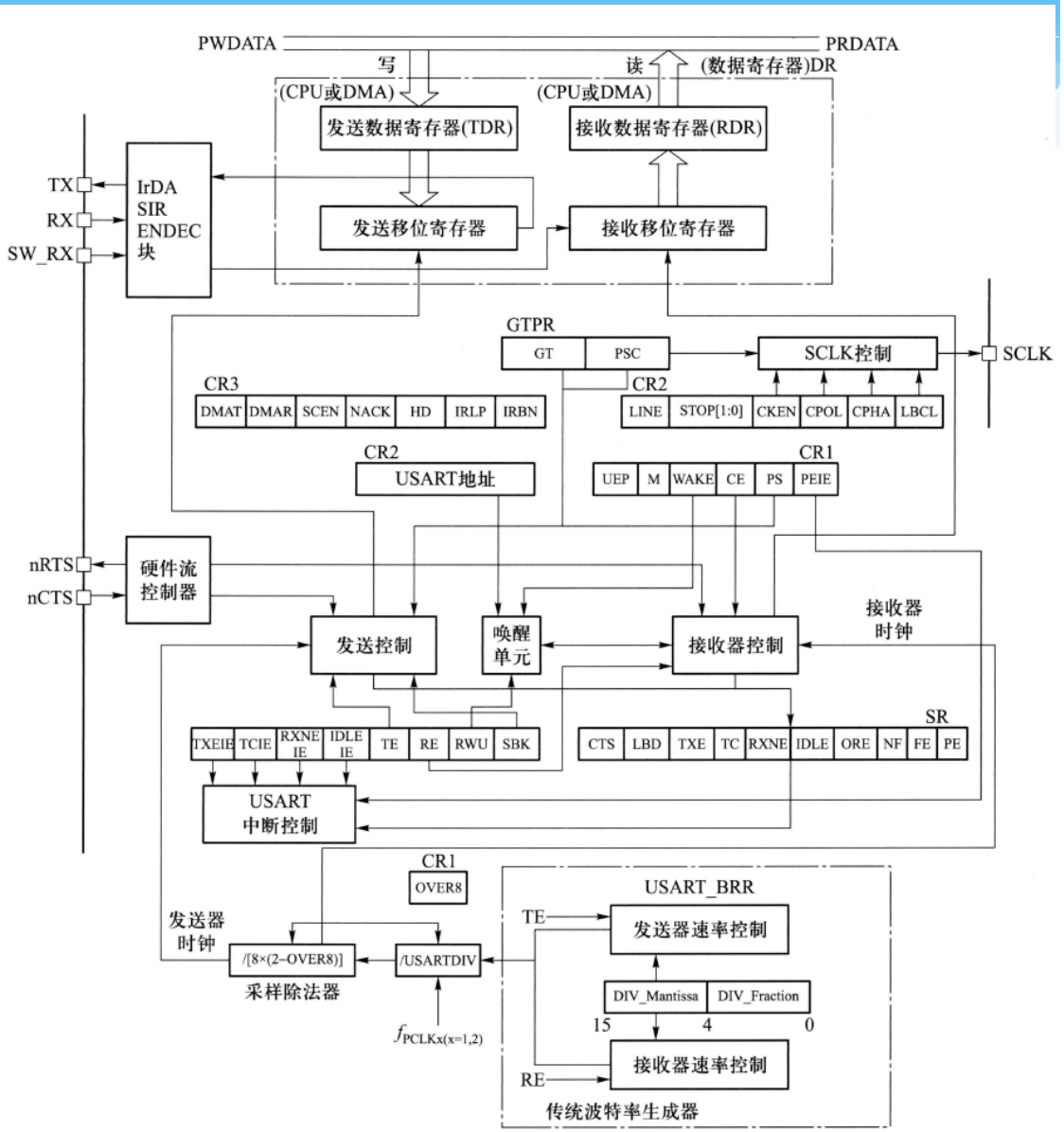
STM32F407的USART模块

TX： 发送数据输出引脚。

RX： 接收数据输入引脚。

PWDATA： 需要发送的并行数据，通过PWDATA 写入发送数据寄存器TDR 。

PRDATA： 接收到的并行数据保存在接收数据寄存器RDR 中，应用程序通过PRDATA读取接收到的数据。



第四节 异步串行通信接口

STM32F407的USART模块

波特率寄存器(USART_BRR)

位15: 4　DIV_Mantissa [11: 0]： 这12 个位用于定义USART 除数(USARTDIV) 的尾数.

位3 : 0　DIV_Fraction [3: 0]： 这4 个位用于定义USART 除数(USARTDIV) 的小数.

f_{ck}为外设的时钟，分频因子USARTDIV是一个存放在寄存器USART_BRR 中的无符号定点数。

例如：假设计算得到的分频因子是27.75，则DIV_Mantissa=0d27=0x1B，小数部分的计算0.75=12/16，所以DIV_Fraction=0xC，于是USART_BRR=0x1BC。

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
DIV_Mantissa[11:0]												DIV_Fraction[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

图 8-3-17　波特率寄存器（USART_BRR）

串口波特率的计算公式如下：

$$T_x/R_x \text{ 波特率} = \frac{f_{CK}}{8 \times (2-OVER8) \times USARTDIV}$$

第四节 异步串行通信接口

STM32F407的USART模块

例 8–11 异步通信双方约定数据格式采用一位起始位、七位数据位、一位偶校验位和一位停止位；波特率选择 1 200 b/s，波特率因子为 16。试确定收 / 发时钟频率、每秒传送的字符数、传输效率，画出传输字符 “E” 的波形图。

解：

（1）收、发时钟频率应为

$$1\,200 \times 16\ \text{Hz} = 19.2\ \text{kHz}$$

（2）每秒传送的字符数为

$$1\,200/(1+7+1+1)\text{个} = 120\text{个}$$

（3）传输效率为

$$7/(1+7+1+1) = 70\%$$

（4）传输字符 “E” 的波形图如图 8–3–4 所示，其中 E 的 ASCII 码为 45H，校验位为 1。

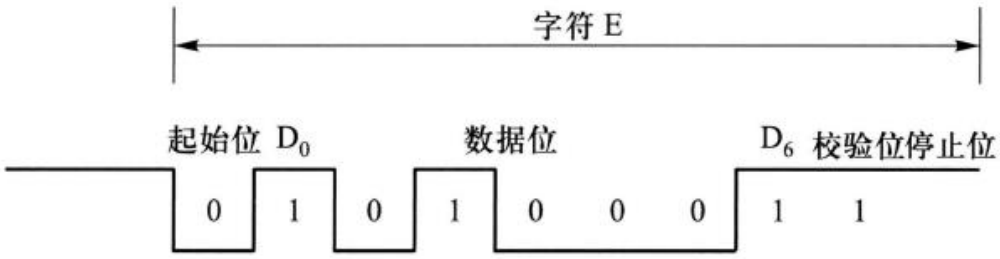


图 8–3–4 传输字符 E 的波形图

第四节 异步串行通信接口

STM32F407的USART模块

例 8–11 异步通信双方约定数据格式采用一位起始位、七位数据位、一位偶校验位和一位停止位；波特率选择 1 200 b/s，波特率因子为 16。试确定收 / 发时钟频率、每秒传送的字符数、传输效率，画出传输字符 “E” 的波形图。

解：

（1）收、发时钟频率应为

$$1\,200 \times 16\ \text{Hz} = 19.2\ \text{kHz}$$

（2）每秒传送的字符数为

$$1\,200/(1+7+1+1)\text{个} = 120\text{个}$$

（3）传输效率为

$$7/(1+7+1+1) = 70\%$$

（4）传输字符 “E” 的波形图如图 8–3–4 所示，其中 E 的 ASCII 码为 45H，校验位为 1。

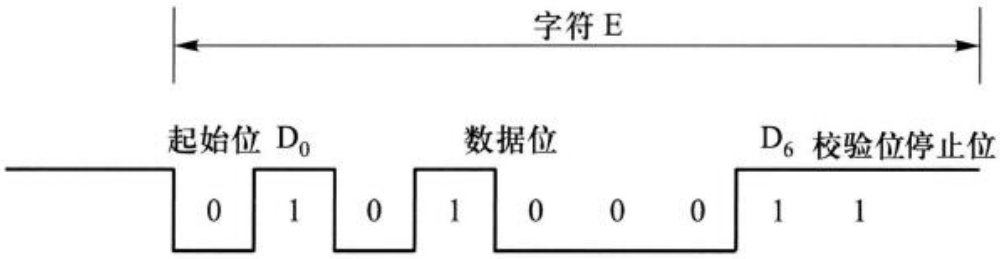


图 8–3–4 传输字符 E 的波形图

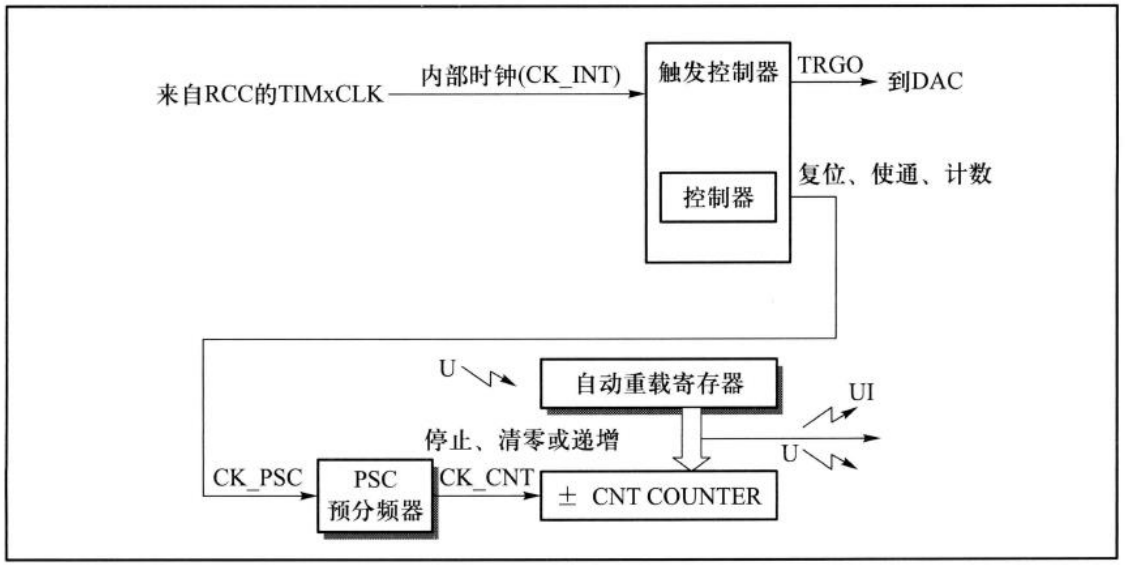
第五节定时器／计数器

1、基本定时器

通过对一个16 位自动重载递增计数器(TIMx_ARR) 和一个16 位预分频器(TIMx_PSC) 编程进行配置。

当定时器开始工作时，计数器CNT_COUNTER在计数时钟CK_CNT 的驱动下，计数值从0 开始不断递增（向上计数），当计数值与TIMx_ARR 寄存器的数值相等时，称计数器达到上溢值，定时器自动产生更新事件，完成一次计数过程。

CK_CNT=CK_PSC/ (TIMx_PSC+ 1)



第五节定时器／计数器

1、基本定时器

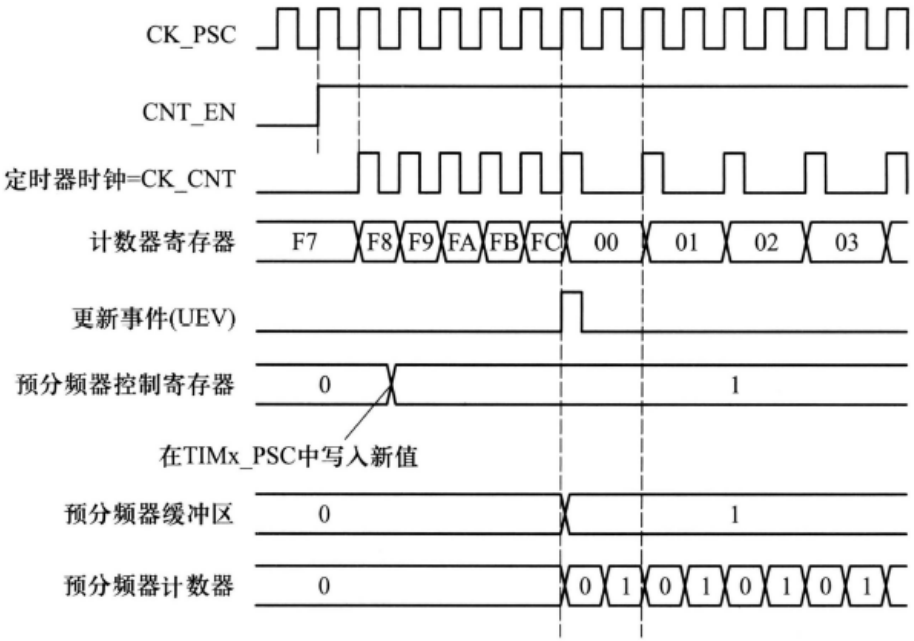


图 8-4-2　预分频器分频由 1 变为 2 时的计数器时序图

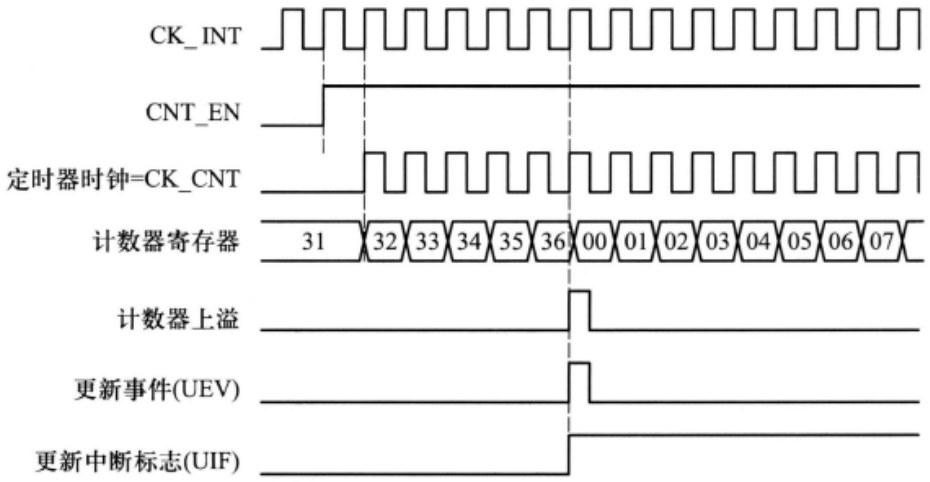


图 8-4-3　TIMx_ARR=0x36 时计数器工作时序图

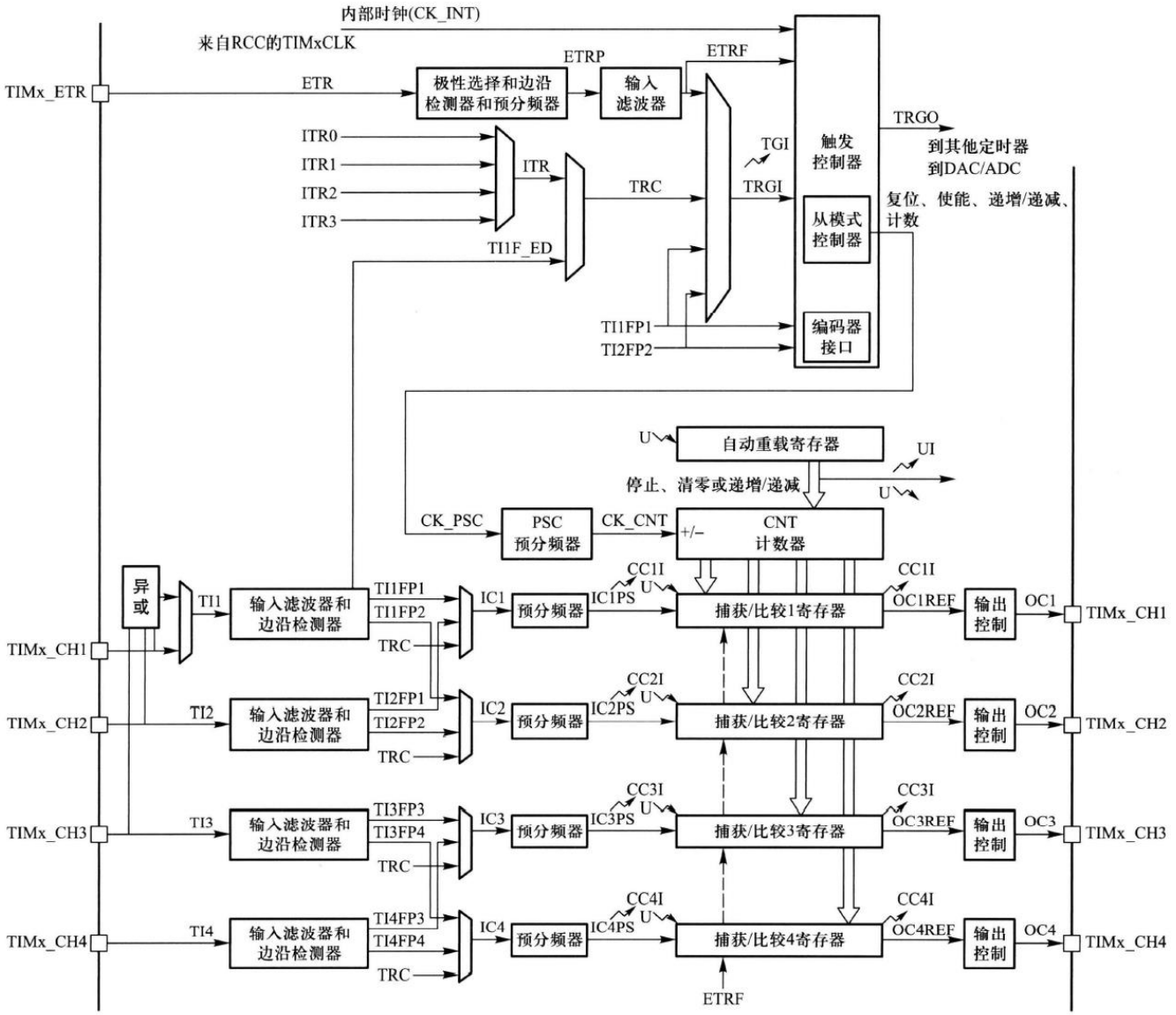
第五节定时器／计数器

2、通用定时器

每个定时器具有4个独立通道：

TIMx_CH1~TIMx_CH4

支持输入捕获、输出比较、脉冲宽度调制PWM 生成以及单脉冲模式输出



时钟来源：

- 内部时钟(CK_INT);
- 外部时钟模式l：外部输入引脚Tix;
- 外部时钟模式2：外部触发模式(ETR);
- 内部触发输入(ITRx)：使用一个定时器作为另一个定时器的预分频器;

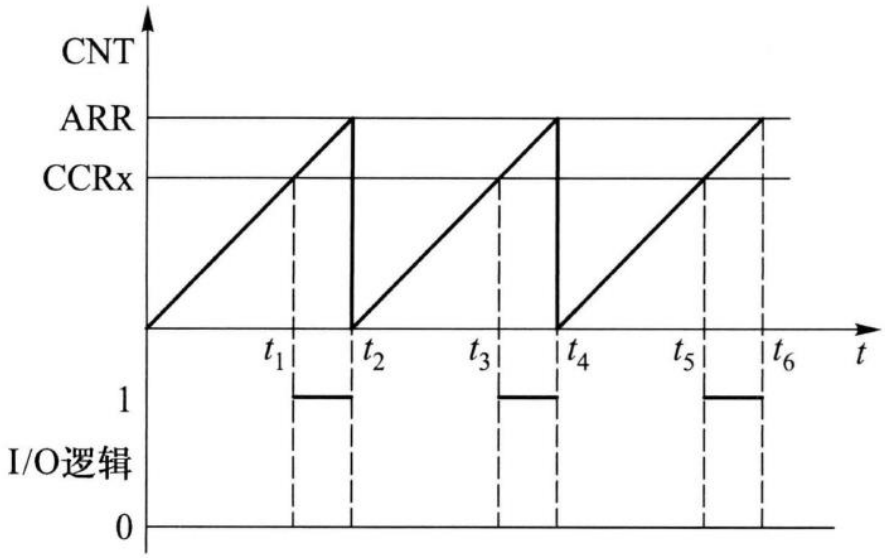
第五节定时器／计数器

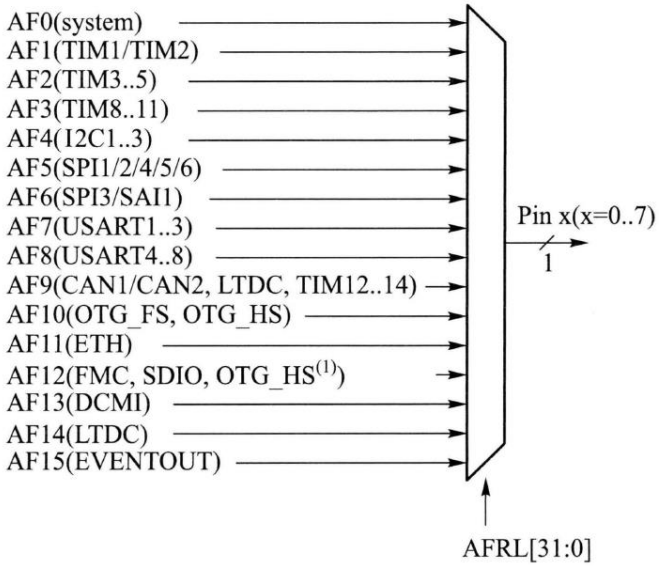
2、通用定时器

（1）递增、递减、递增／递减计数

（2）脉冲宽度调制PWM (pulse width modulation)

当计数值CNT 小于捕获/比较寄存器CCR 的设置值时(t<t1)，输出低电平；当计数值CNT 继续递增并大于等于CCR 的设置值时(t>t1)，输出为高电平，占空比为CCR/ (ARR+1)。





若通过GPIO 复用功能低位寄存器GPIOB_AFRL将PBx 引脚复用到AF2，可实现从PBx 引脚输出TIM3 PWM 脉冲信号。

第五节定时器／计数器

3、高级控制定时器

1、高级控制定时器可用于多种用途，支持输入捕获、输出比较、PWM生成（边沿和中心对齐模式）以及单脉冲模式输出，具有可编程死区(dead time)的互补输出；

2、高级控制定时器具有重复计数器TIMx_RCR ，仅在给定数目的计数器周期后，才更新定时器寄存器，即每当发生N+l（N为TIMx_RCR寄存器的值）个计数器上溢或下溢事件时，影子寄存器才从预装寄存器中拷贝数值。

* H7的高分辨率定时器High-resolution
* 低功耗定时器

39 High-Resolution Timer (HRTIM)

40 Advanced-control timers (TIM1/TIM8)

41 General-purpose timers (TIM2/TIM3/TIM4/TIM5)

42 General-purpose timers (TIM12/TIM13/TIM14)

43 General-purpose timers (TIM15/TIM16/TIM17)

44 Basic timers (TIM6/TIM7)

45 Low-power timer (LPTIM)

第五节定时器／计数器

4、定时器模块的寄存器

对定时器的编程主要通过配置相关的寄存器来完成：

表 8-4-2　定时器寄存器地址映射

定时器	寄存器地址映射	定时器	寄存器地址映射
TIM1	0x40010000~0x400103FF	TIM8	0x40010400~0x400107FF
TIM2	0x40000000~0x400003FF	TIM9	0x40014000~0x400143FF
TIM3	0x40000400~0x400007FF	TIM10	0x40014400~0x400147FF
TIM4	0x40000800~0x40000BFF	TIM11	0x40014800~0x40014BFF
TIM5	0x40000C00~0x40000FFF	TIM12	0x40001800~0x40001BFF
TIM6	0x40001000~0x400013FF	TIM13	0x40001C00~0x40001FFF
TIM7	0x40001400~0x400017FF	TIM14	0x40002000~0x400023FF

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved						CKD[1:0]		ARPE	CMS		DIR	OPM	URS	UDIS	CEN
						rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

图 8-4-9　控制寄存器 1 TIMx_CR1

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	TDE	Res.	CC4DE	CC3DE	CC2DE	CC1DE	UDE	Res.	TIE	Res	CC4IE	CC3IE	CC2IE	CC1IE	UIE
	rw		rw	rw	rw	rw	rw		rw		rw	rw	rw	rw	rw

图 8-4-10　DMA/ 中断使能寄存器 TIMx_DIER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNT[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

图 8-4-11　计数器 TIMx_CNT

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PSC[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

图 8-4-12　预分频器 TIMx_PSC

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ARR[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

图 8-4-13　自动重装载寄存器 TIMx_ARR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OC2CE	OC2M[2:0]			OC2PE	OC2FE	CC2S[1:0]		OC1CE	OC1M[2:0]			OC1PE	OC1FE	CC1S[1:0]	
IC2F[3:0]				IC2PSC[1:0]				IC1F[3:0]				IC1PSC[1:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

图 8-4-14　TIMx 捕获 / 比较模式寄存器 1 TIMx_CCMR1

第五节定时器／计数器

5、定时器编程示例

（1）定时功能

定时器的定时周期主要通过对 TIMx_PSC 和 TIMx_ARR 两个寄存器编程来控制。例如需要一个周期为 1 s 的定时器，假设预分频器的输入时钟 CK_PSC 为 90 MHz。首先，设置预分频器 TIMx_PSC 的值为 9000-1，则分频系数等于 9000，分频后输出的计数时钟 CK_CNT 频率为 10 KHz，周期为 100 μs。然后，设置 TIMx_ARR 寄存器的值为 9999。定时器计数器 TIMx_CNT 从 0 开始计数，当计数值等于 9999 时生成更新事件，此时总共计数 10000 次。这样，定时器以 100μs 为周期执行了 10000 次计数，产生了 1 s 的定时周期。定时器计数周期 Tperiod 的计算公式如下：

$$T_{\text{period}}=\frac{(TIMx\_ARR+1)\times(TIMx\_PSC+1)}{CK\_PSC}=\frac{(9999+1)\times(8999+1)}{90\ \text{MHz}}=1\ \text{s}$$

定时器编程一般包括下面几个步骤：1）如果允许定时器以中断方式工作，则需设置中断优先级分组；2）为了节省功耗，STM32F04 相关外设默认处于关闭状态。为了使用定时器，必须设置 RCC_APB1/APB2（reset clock controller_advanced peripheral bus1/2）外设时钟使能寄存器中与定时器对应的比特位，从而开启定时器的时钟；3）对定时器进行配置，主要包括设置预分频寄存器 TIMx_PSC 和自动重载寄存器 TIMx_ARR，配置控制寄存器 TIMx_CR 选择计数模式和分频因子等；4）设置定时器相关的中断，除了设置 NVIC 定时器中断优先级并允许内核响应定时器中断外，还需设置 TIMx_DIER 寄存器允许计数器在上溢 / 下溢时生成更新事件中断；5）最后通过配置控制寄存器 TIMx_CR 使能定时器，定时器开始工作。下面通过例 8-16 汇编程序说明定时器的初始化过程。

第六节 数模转换器和模数转换器

1、概述

在系统中完成模拟信号转换成数字信号的装置称为模数转换器(A/D 转换器）；反之，完成数字信号转换成模拟信号的装置称为数模转换器(D / A 转换器）。

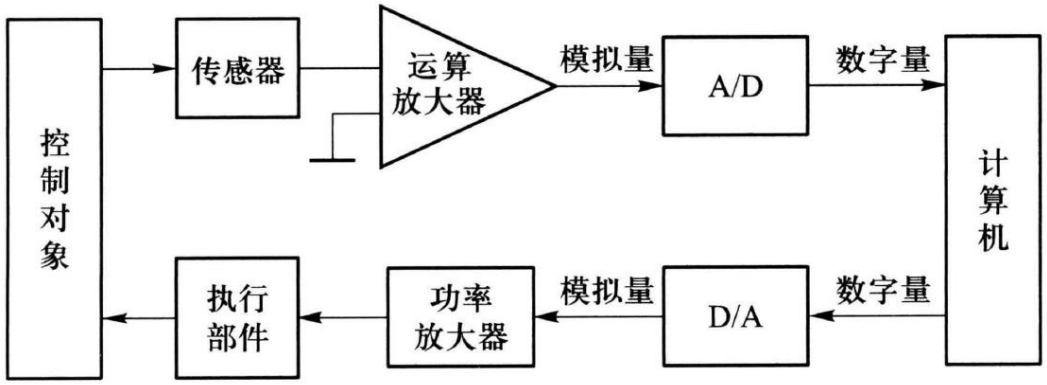


图 8-5-1　一个包含 A/D 和 D/A 转换环节的实时控制系统

第六节 数模转换器和模数转换器

2、数模转换器

D/A 转换器的主要参数:

(1) 分辨率：如4 位DAC ，其分辨率为$(1/2^4-1)$

(2) 输出范围：0-V_{REF+} ，满量程FS (full scale) = V_{REF+}

(3) 转换精度：绝对精度和相对精度(如0.5*LSB*)

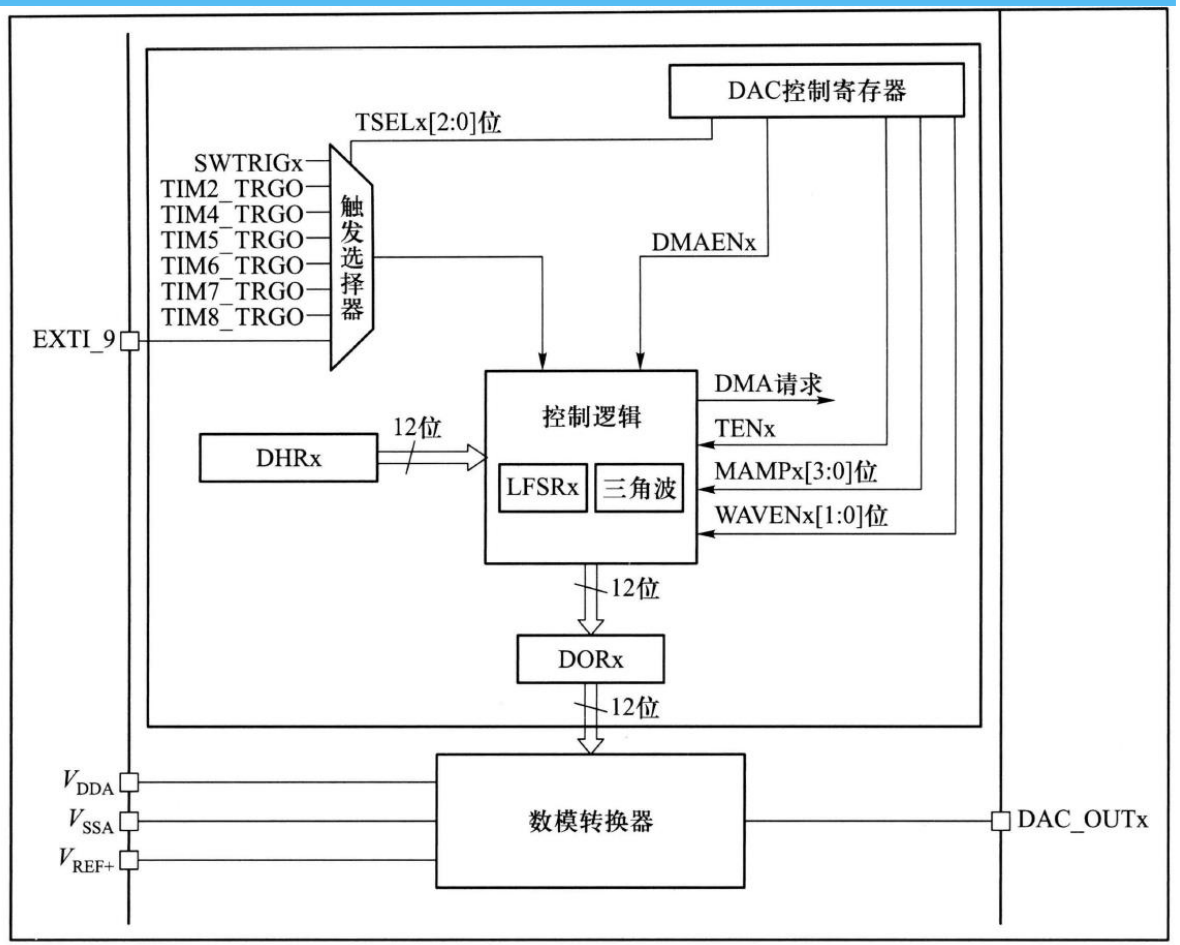
(4) 转换速率：模拟输出电压的最大变化速度，单位为V/μS

(5) 线性误差:实际转换特性与理想转换特性之间的偏差

第六节 数模转换器和模数转换器

2、数模转换器

DORx无法直接写入，任何数据都必须通过加载DHRx寄存器才能传输到DAC通道x 。



名称	信号类型	备注
V_{REF+}	正模拟参考电压输入	DAC 高 / 正参考电压，1.8 V ≤ V_{REF+} ≤ V_{DDA}
V_{DDA}	模拟电源输入	模拟电源
V_{SSA}	模拟电源接地输入	模拟电源接地
DAC_OUTx	模拟输出信号	DAC 通道 x 模拟输出

第六节 数模转换器和模数转换器

3、模数转换器

A / D转换器的主要参数:

(1) 转换精度：ADC 的实际输出接近理想输出的精确程度。通常用数字量的最低有效位(LSB) 来表示。

(2) 转换率：完成一次A/D 转换所需要的时间的倒数。

例如完成一次A/D转换需要的时间是100ns，那么转换率为10MHz，有时也标为10MSPS，即每秒转换1000 万次。

(3) 分辨率:

表明了能够分辨最小的量化信号的能力，通常用位数来表示。对于一个实现N位二进制转换的ADC来说，它能分辨的最小量化信号的能力为2^N位，所以它的分辨率为2^N。

第六节 数模转换器和模数转换器

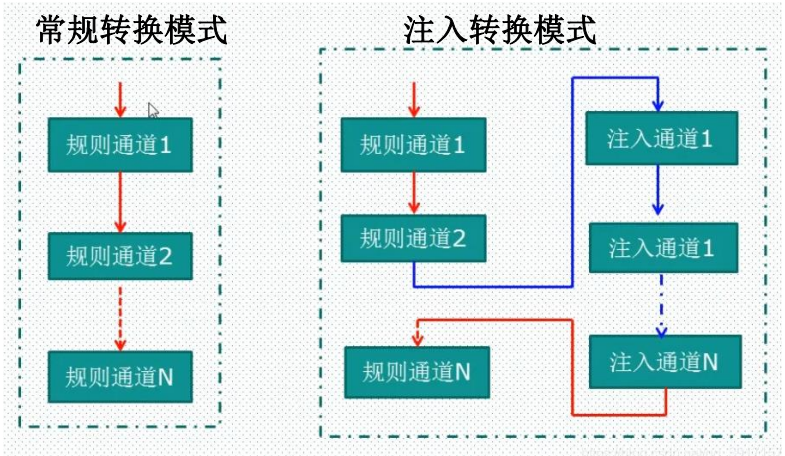
3、模数转换器

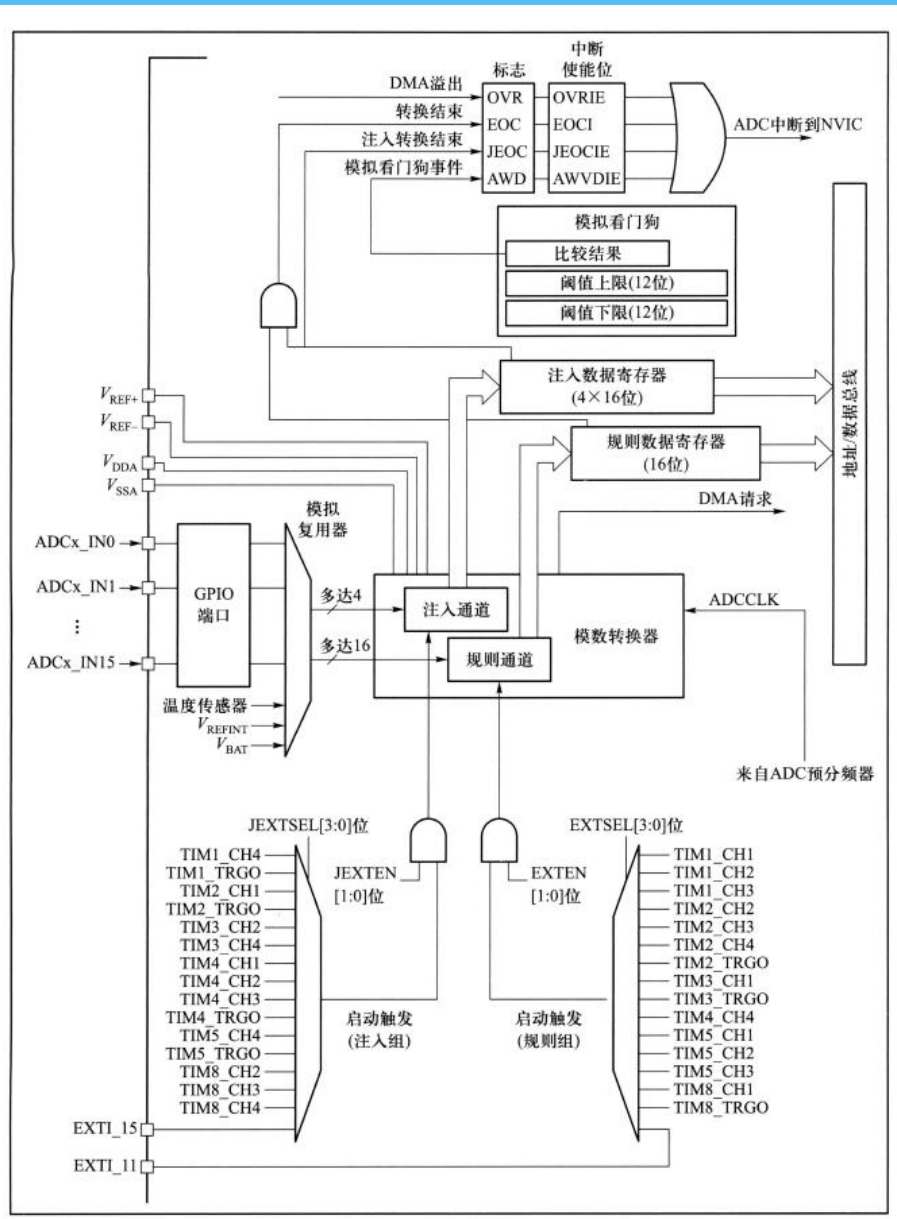
A / D 转换可在单次、连续、扫描或不连续采样模式下进行。ADC的结果存储在一个左对齐或右对齐的16 位数据寄存器中。ADC具有模拟看门狗特性，允许应用检测输入电压是否超过了用户自定义的阈值上限或下限。

第六节 数模转换器和模数转换器

3、模数转换器

名称	信号类型	备注
V_{REF+}	正模拟参考电压输入	ADC 高 / 正参考电压，1.8 V ≤ V_{REF+} ≤ V_{DDA}
V_{DDA}	模拟电源输入	模拟电源电压等于 V_{DD} 全速运行时，2.4 V ≤ V_{DDA} ≤ V_{DD}（3.6 V） 低速运行时，1.8 V ≤ V_{DDA} ≤ V_{DD}（3.6 V）
V_{REF-}	负模拟参考电压输入	ADC 低 / 负参考电压，V_{REF-} = V_{SSA}
V_{SSA}	模拟电源接地输入	模拟电源接地电压等于 V_{SS}
ADCx_IN[15:0]	模拟输入信号	16 个模拟输入通道





第六节 数模转换器和模数转换器

3、模数转换器

转换模式:

1) 单次转换模式

ADC 执行一次转换。

2) 连续转换模式

ADC 结束一个转换后立即启动一个新的转换。

3) 模拟看门狗

如果ADC转换的模拟电压低于阈值下限或高于阈值上限,则模拟看门狗状态位会置1。

4) 扫描模式

ADC可以在此模式下扫描一组选中的模拟通道。为组中的每个通道都执行一次转换。每次转换结束后，会自动转换该组中的下一个通道。

5) 不连续采样模式

该模式可用于转换含有n个转换的短序列，该短序列是上述扫描模式中选择的转换序列的一部分。出现外部触发时，将启动接下来n个转换，直到序列中的所有转换均完成为止。

例题解释

（2）脉冲宽度调制 PWM 功能

例 8-17 要求定时器 TIM3 产生频率为 100 Hz、占空比为 20% 的脉冲信号，并通过并行接口 GPIOB PB4 引脚输出生成的脉冲信号。

根据题目要求，首先配置 PWM 脉冲频率。设置 TIM3_ARR=99 和 TIM3_PSC=8399（设内部时钟 CK_INT 为 84 MHz），则 TIM3 脉冲信号周期 =（99+1）（8399+1）/84 MHz=10 ms，即信号频率为 100 Hz。然后，配置 PWM 占空比。设置捕获 / 比较寄存器 TIM3_CCR1=20，并配置捕获 / 比较使能寄存器 TIM3_CCER，选择通道 CH1 的输出极性为低电平有效（注意，此处的信号极性设置与图 8-4-7 示例相反）。因此，当计数值 CNT 小于 20 时，脉冲信号输出高电平（无效状态）；而计数值大于 20 时，脉冲信号输出低电平（有效状态），即信号占空比 =CCR/（ARR+1）=20/100=20%。最后，将 TIM3 产生的脉冲信号输出到 GPIOB 的 PB4 引脚。先配置复用功能低位寄存器 GPIOB_AFRL，将 TIM3 输出连接到并口引脚 PB4，再配置端口模式寄存器 GPIOB_MODER，将引脚 PB4 设置为复用功能（alternate function），并为 PB4 选择模式和速率等参数。

下面给出基于库函数的相应的 C 语言程序代码。

```
1   // 例 8-17：利用 PWM 生成频率为 100 Hz 占空比为 20% 的脉冲信号
2   int main（void）
3   {
4
5   TIM3_PWM_Configure（100-1，8400-1）；  // 计数时钟为 84M/8400=10 kHz，自动重载寄存器
6   // 为 100，因此 PWM 频率为 10 k/100 Hz=100 Hz
7   TIM_SetCompare1（TIM3，20）；       //PWM 占空比为 20/100=20%
8   }
9   /*********************************
10  功能：PWM 初始化
```

```
11  参数：arr 自动重装载值，psc 预分频值
12  返回值：无
13  ********************************/
14  void TIM3_PWM_Configure(uint16_t arr, uint16_t psc)
15  {
16      TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure; //TIM 时基初始化结构定义
17      TIM_OCInitTypeDef  TIM_OCInitStructure;          //TIM 输出比较初始化结构定义
18      GPIO_InitTypeDef  GPIO_InitStructure;            //GPIO 初始化结构定义
19
20      RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE); // 开启 GPIOB 时钟
21      RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3, ENABLE);  // 开启 TIM3 时钟
22
23      GPIO_PinAFConfig(GPIOB, GPIO_PinSource4, GPIO_AF_TIM3); //TIM3 输出复用到引
24    脚 PB4
25      GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4    // 选择 PB4
26      GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF; //PB4 为复用功能模式
27      GPIO_InitStructure.GPIO_OType = GPIO_OType_PP; //PB4 为输出推挽模式
28      GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100 MHz; // PB4 输出为高速模式
29      GPIO_Init(GPIOB, &GPIO_InitStructure);      // 配置并口 GPIOB
30
31      TIM_TimeBaseStructure.TIM_Period = arr;   // 定时器初值为 99+1
32      TIM_TimeBaseStructure.TIM_Prescaler = psc; // 定时器预分频系数为 8399+1
33      TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up; // 递增计数
34      TIM_TimeBaseStructure.TIM_ClockDivision = TIM_CKD_DIV1; // 默认内部时钟
35      TIM_TimeBaseInit(TIM3, &TIM_TimeBaseStructure); // 执行定时器 TIM3 初始化
36
37  // 通道 CH1 选择 PWM1 模式。当定时器选择递增计数时，若满足 TIM3_CNT<TIM3_CCR1，
38  // 则通道 CH1 为有效状态（本例中选择有效状态为低电平），否则无效
39  TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;
40  // 使能通道 CH1
41  TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
42  // 输出极性为低电平有效。本例中，当 CNT 计数值小于 CCR1=20 时，PWM 输出高电平；
43  // 当 CNT 计数值为 20~100 时，PWM 输出低电平。因此信号占空比为 20/100=20%
44      TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_Low;
45      // 初始化 TIM3 输出比较功能
46      TIM_OC1Init(TIM3, &TIM_OCInitStructure);
47
48      TIM_Cmd(TIM3, ENABLE);              // 使能 TIM3 定时器
49  }
```

4. A/D 转换器应用举例

例 8–19 通过程序配置可以实现 ADC 模块规则通道的单次转换功能。

解：

```
void  Adc_Init ( void )
{
        ADC_InitTypeDef ADC_InitStructure;
        GPIO_InitTypeDef GPIO_InitStructure;
        RCC_APB2PeriphClockCmd ( RCC_APB2Periph_GPIOA|RCC_APB2Periph_ADC1, ENABLE );
      // 使能 ADC1 通道时钟
        RCC_ADCCLKConfig ( RCC_PCLK2_Div6 ); // 设置 ADC 分频因子
        GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1;
        GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN; // 模拟输入引脚
        GPIO_Init ( GPIOA, &GPIO_InitStructure );

        ADC_DeInit ( ADC1 ); // 复位 ADC1,将外设 ADC1 的全部寄存器重设为缺省值
        ADC_InitStructure.ADC_Mode = ADC_Mode_Independent; // ADC1 和 ADC2 工作在独立模式
        ADC_InitStructure.ADC_ScanConvMode = DISABLE; // 单通道模式
        ADC_InitStructure.ADC_ContinuousConvMode = DISABLE; // 单次转换模式
        ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None; // 转换由软件触发启动
        ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right; //ADC 数据右对齐
        ADC_InitStructure.ADC_NbrOfChannel = 1; // 顺序进行规则转换的 ADC 通道的数目
        ADC_Init ( ADC1, &ADC_InitStructure ); // 初始化外设 ADCx 的寄存器
        ADC_Cmd ( ADC1, ENABLE ); // 使能指定的 ADC1
        ADC_ResetCalibration ( ADC1 ); // 使能复位校准
        while ( ADC_GetResetCalibrationStatus ( ADC1 )); // 等待复位校准结束
        ADC_StartCalibration ( ADC1 ); // 开启 AD 校准
        while ( ADC_GetCalibrationStatus ( ADC1 )); // 等待校准结束
      }

      // 获得 ADC 值
      //ch: 通道值 0~3
      u16 Get_Adc ( u8 ch )
      {
        // 设置指定 ADC 的规则组通道,一个序列,采样时间
        ADC_RegularChannelConfig ( ADC1, ch, 1, ADC_SampleTime_239Cycles5 ); //ADC1, ADC 通道,采样时
间为 239.5 周期
        ADC_SoftwareStartConvCmd ( ADC1, ENABLE ); // 使能指定的 ADC1 的软件转换启动功能
        while ( !ADC_GetFlagStatus ( ADC1, ADC_FLAG_EOC )); // 等待转换结束
        return ADC_GetConversionValue ( ADC1 ); // 返回最近一次 ADC1 规则组的转换结果
      }

      // 多次测量,得到平均结构,提高测量值的准确度
      u16 Get_Adc_Average ( u8 ch, u8 times )
      {
        u32 temp_val=0;
        u8 t;
        for ( t=0; t<times; t++ )
        {
            temp_val+=Get_Adc ( ch );
            delay_ms ( 5 );
        }
        return temp_val/times;
      }
```