第十一讲 STM32F103 嵌入式微处理器

§1 MCU组成模型和学习方法

§2 STM32F系列及命名方法

§3 STM32F103C8概述

§4 STM32F103的功能部件

§1 MCU组成模型和学习方法

§1.1 MCU的作用

① 名称：Micro Control Unit 微控制器

② 功能

通过CPU运算、部件的功能，实现在不同应用场合的检测、控制等功能；

③ 特点

以芯片的形式集成到嵌入式系统中，在完成指定功能的基础上，提供高可靠性、低成本、低功耗、体积小等特性（嵌入式系统与微机系统的差别）

浙江大学 ZheJiang University

§1.2 MCU的组成模型

① **组成：** CPU + FBs（Functional Blocks）

- CPU 核： 内部架构、位数、个数、主频等，如CM3
- FBs 功能部件：多个，能独立运行，但受CPU控制

② **内部结构**

CPU 和FBs通过内部三总线相互连接，即数据总线、地址总线和控制总线（包括控制信号和状态信号）

③ **两者的通信方式**

CPU为主，读写功能部件的内部寄存器

功能部件工作完成后向CPU发起中断请求

§1.3 MCU的主动学习方法

① 网上找资料

MCU是半导体公司设计和生产的，最新的权威资料可以从该公司的网站上获取；

② 用套件做实验

由于MCU有较高的复杂度，根据说明资料直接应用是一件困难的事。用套件来熟悉和验证MCU各部件及常见扩展硬件的编程使用方法，以增加感性认识，得到编程的参考模板；

③ 应用MCU

通过设计练习和实物制作等环节，掌握MCU的特性和应用方法。

浙江大学 ZheJiang University

§1.4 MCU的参考资料

① 数据手册(Data Sheet)

介绍某个MCU的组成和应用信息，如

STM32F103C8.pdf

② 用户手册(User Guider)

详细介绍一个MCU系列中的功能部件，如

STM32F103 Reference Manuals .pdf

③ 应用例子(App Notes)

简要介绍MCU及功能部件的应用参考，

ST公司的网站www.stmicroelectronics.com

④ 勘误表（Errata Sheet）

记录MCU的缺陷（偏离DS, UG中规定的功能）及补救方法，如STM32F103C8 errata sheet.pdf

§1.5 实验套件的作用

① 提供CPU核的部件、MCU中内部功能部件的实验代码；

② 提供MCU外部扩展部件的使用方法（代码例）；

③ 帮助同学掌握ARM (MCU) 的集成开发环境IDE；

④ 库函数以及辅助软件学习，降低嵌入式系统开发难度；

⑤ 可以作为积累个人开发平台资源的素材。

§1.6 功能部件的学习内容

① 硬件组成：包含哪些功能电路？

② 主要功能：该部件有哪几种功能？

③ 工作模式：实现某功能的运行方式，通常有多种，是如何选择的？

④ 内部寄存器的定义：控制寄存器、状态寄存器各位的含义，功能部件是让CPU读写这些内部寄存器，来选择运行模式，实现具体的功能；

⑤ 该部件对应的库函数有哪些？

浙江大学 ZheJiang University

§1.7 功能部件的学习顺序

第一步：RCC

第二步：GPIO

第三步：中断系统、DMA

第四步：Timers、（并列，按需选择）

A/D、D/A、

(UART、SPI、I2C)、USB、CAN、EMAC

FSMC

…

浙江大学 ZheJiang University

第十一讲 STM32F103 嵌入式微处理器

§1 MCU组成模型和学习方法

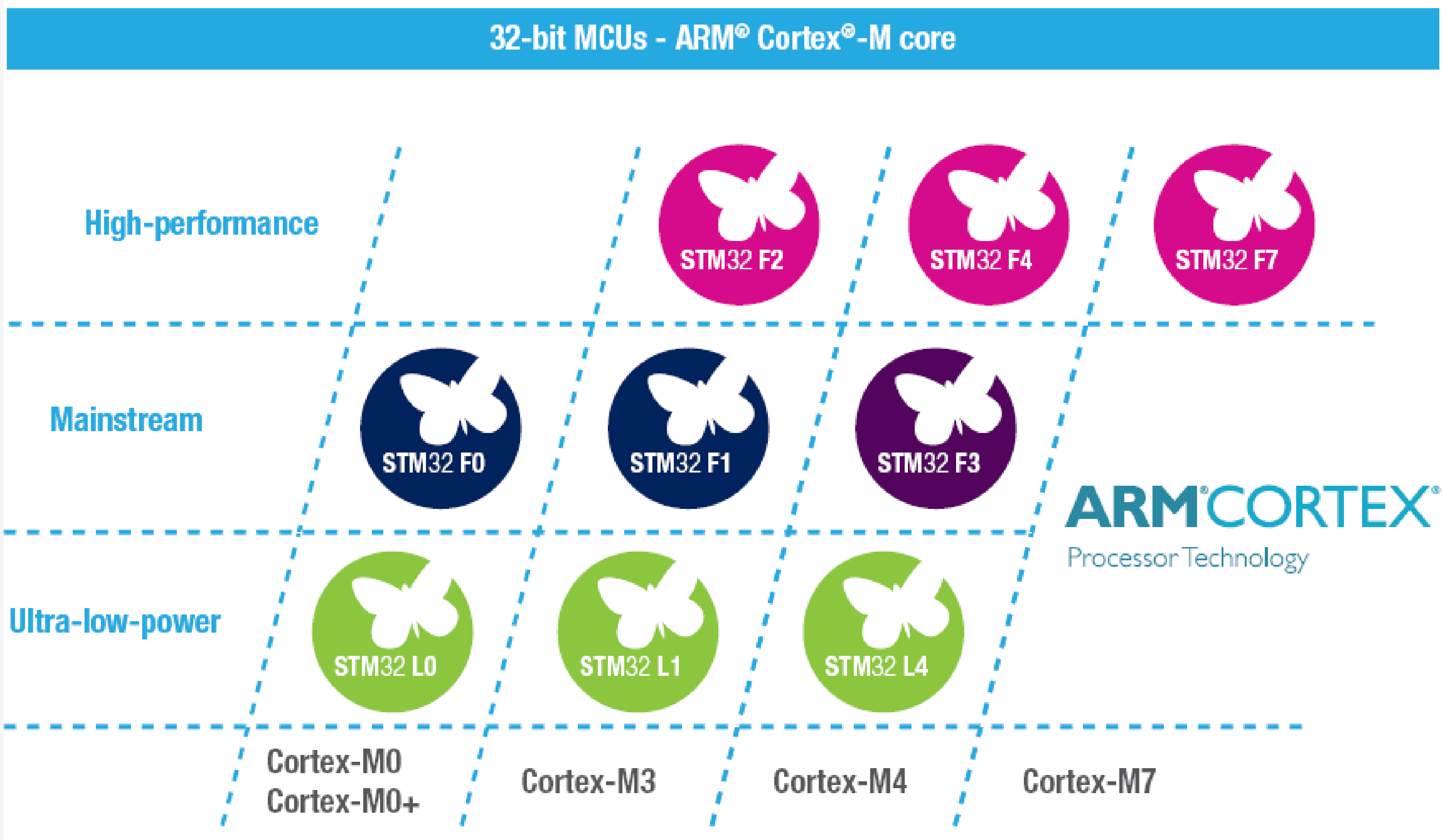
§2 STM32F系列及命名方法

§3 STM32F103C8概述

§4 STM32F103C8功能部件

浙江大学
ZheJiang University

§2.1 STM32F系列



http://www.stmicroelectronics.com

§2.2 STM32F1XX系列

- 主流MCU，用于工业、医疗和消费类市场的各种应用。
- 凭借该产品系列，意法半导体在全球ARM Cortex-M3微控制器领域处于领先地位，同时树立了嵌入式应用的里程碑。
- 该系列利用一流的外设和低功耗、低压操作实现了高性能，同时价格低、生态环境好。

§2.2 STM32F1XX系列

该系列包含五个产品线，它们的引脚、外设和软件均兼容。

Cortex®-M4 (DSP + FPU) - Up to 72 MHz

- -40 to +105 °C range
- USART, SPI, I^2C
- 16- and 32-bit timers
- Temperature sensor
- Up to 3 x 12-bit ADC
- Dual 12-bit DAC
- Low voltage 2.0 to 3.6 V (5 V tolerant I/Os)

STM32 F1 Product line	FCPU (MHz)	FLASH (bytes)	RAM (KB)	USB 2.0 FS	USB 2.0 FS OTG	FSMC	CAN 2.0B	3-phase MC timer	I^2S	SDIO	Ethernet IEEE1588	HDMI CEC
STM32F100 Value line	24	16 K to 512 K	4 to 32			•		•				•
STM32F101	36	16 K to 1 M	4 to 80			•						
STM32F102	42	16 K to 128 K	4 to 16	•								
STM32F103	72	16 K to 1 M	6 to 96	•		•	•	•	•	•		
STM32F105 STM32F107	72	64 K to 256 K	64		•	•	•	•	•		•	

浙江大学 ZheJiang University

§2.2 STM32F1XX系列

36脚	STM32F103T6	32k	10k
	STM32F103T8	64k	20k
48脚	STM32F103C6	32k	10k
	STM32F103C8	64k	20k
	STM32F103CB	128k	20k
64脚	STM32F103R6	32k	10k
	STM32F103R8	64k	20k
	STM32F103RB	128k	20k
	STM32F103RC	256k	48k
	STM32F103RD	384k	64k
	STM32F103RE	512k	64k
100脚	STM32F103V8	64k	20k
	STM32F103VB	128k	20k
	STM32F103VC	256k	48k
	STM32F103VD	384k	64k
	STM32F103VE	512k	64k
144脚	STM32F103ZC	256k	48k
	STM32F103ZD	384k	64k
	STM32F103ZE	512k	64k

§2.3 STM32F的命名方法

STM32 F 103 Z E T 6

① ② ③ ④ ⑤ ⑥ ⑦

① 产品系列名：固定为STM32

② 产品类型：F表示这是Flash产品

③ 产品子系列：103增强型，101基本型，105连接型

④ 管脚数目：

T=36脚、C=48脚、R=64脚、V=100脚、Z=144脚

⑤ 闪存存储器容量：

6=32K ,8=64K,B=128K,C=256K, D=384K,E=512K

⑥ 封装信息：H=BGA 、T=LQFP

⑦ 温度等级：6=-40~85 ℃，7=-40~105 ℃

（芯片的温度等级：商业级：0~70 ℃；工业级：-40~85 ℃；汽车级：-40~125 ℃；军工级-55~155 ℃）

浙江大学 ZheJiang University

第十一讲 STM32F103嵌入式微处理器

§3 STM32F103C8 概述

§3.1 STM32F103C8的特点

§3.2 STM32F103C8的组成

§3.3 RCC 复位和时钟控制

§3.4 上电启动过程

§3.5 中断系统

§3.6 DMA系统

§3.7 最小应用系统

浙江大學
ZheJiang University

§3.1 STM32F103C8的特点

Peripheral		STM32F103Tx		STM32F103Cx		STM32F103Rx		STM32F103Vx	
Flash - Kbytes		64	128	64	128	64	128	64	128
SRAM - Kbytes		20		20		20		20	
Timers	General-purpose	3		3		3		3	
	Advanced-control	1		1		1		1	
Communication	SPI	1		2		2		2	
	I^2C	1		2		2		2	
	USART	2		3		3		3	
	USB	1		1		1		1	
	CAN	1		1		1		1	
GPIOs		26		37		51		80	
12-bit synchronized ADC Number of channels		2 10 channels		2 10 channels		2 16 channels(1)		2 16 channels	
CPU frequency		72 MHz							
Operating voltage		2.0 to 3.6 V							
Operating temperatures		Ambient temperatures: -40 to +85 °C / -40 to +105 °C (see *Table 9*) Junction temperature: -40 to + 125 °C (see *Table 9*)							
Packages		VFQFPN36		LQFP48, UFQFPN48		LQFP64, TFBGA64		LQFP100, LFBGA100, UFBGA100	

§ 3.1 STM32F103C8的特点

- Mainstream Performance line, ARM Cortex-M3 MCU with 64 Kbytes Flash, 72 MHz CPU, motor control, USB and CAN
- 48 pins， without extended 32-bit bus and FSMC

浙江大学 ZheJiang University

§3.2 STM32F103C8的组成

1）CPU：Cortex-m3，72 MHz ；

2）功能部件

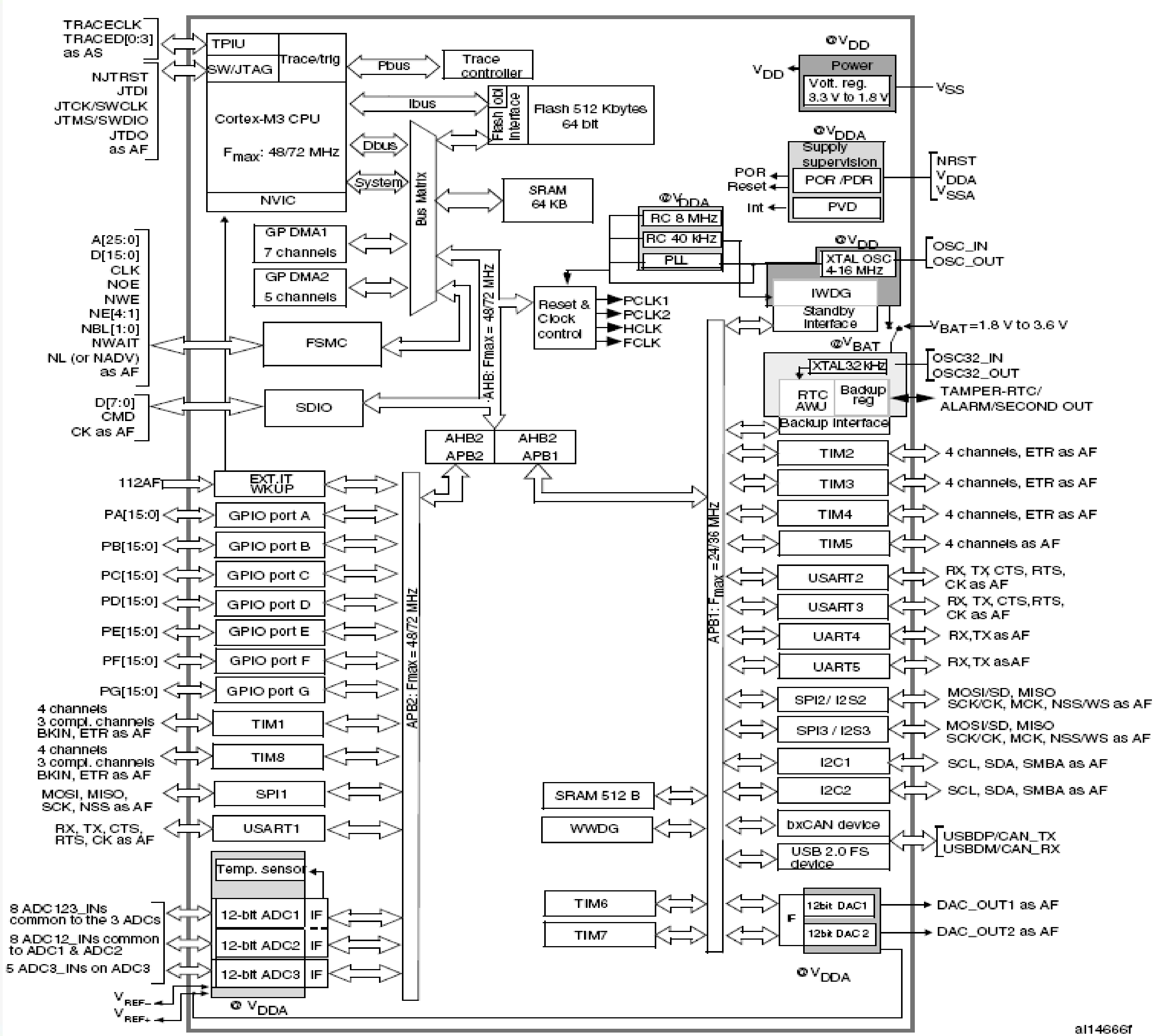
- Flash: 64KB, SRAM:20KB, FSMC
- 37 GPIO Pins with 16 Interrupt vectors, 5V-tolerant
- 9 communication interfaces

  2xI2C 、3xUSART、2xSPI、CAN 、USB 2.0
- 7 timers：

  3 x16-bit TIM with 4 chs， 1x16-bit for motor control,

  2x WDT，24-bit Systick TIM
- 2-ch 12-bit A/D
- 7-ch DMA, RTC with Vbat
- CRC calculation unit
- 96-bit unique ID

3）内部结构示意图


TRACECLK
TRACED[0:3]
as AS
NJTRST
JTDI
JTCK/SWCLK
JTMS/SWDIO
JTDO
as AF
TPIU
SW/JTAG
Trace/trig
Pbus
Trace controller
Cortex-M3 CPU
Fmax: 48/72 MHz
NVIC
Ibus
Dbus
System
Flash obl Interface
Flash 512 Kbytes
64 bit
Bus Matrix
SRAM
64 KB
@VDD
VDD
Power
Volt. reg.
3.3 V to 1.8 V
VSS
@VDDA
Supply supervision
POR /PDR
PVD
POR
Reset
Int
NRST
VDDA
VSSA
@VDDA
RC 8 MHz
RC 40 kHz
PLL
@VDD
XTAL OSC
4-16 MHz
OSC_IN
OSC_OUT
IWDG
Standby interface
VBAT=1.8 V to 3.6 V
@VBAT
XTAL32 kHz
OSC32_IN
OSC32_OUT
RTC
AWU
Backup reg
Backup interface
TAMPER-RTC/
ALARM/SECOND OUT
A[25:0]
D[15:0]
CLK
NOE
NWE
NE[4:1]
NBL[1:0]
NWAIT
NL (or NADV)
as AF
GP DMA1
7 channels
GP DMA2
5 channels
FSMC
D[7:0]
CMD
CK as AF
SDIO
AHB: Fmax = 48/72 MHz
Reset &
Clock
control
PCLK1
PCLK2
HCLK
FCLK
AHB2
APB2
AHB2
APB1
112AF
EXT.IT
WKUP
PA[15:0]
GPIO port A
PB[15:0]
GPIO port B
PC[15:0]
GPIO port C
PD[15:0]
GPIO port D
PE[15:0]
GPIO port E
PF[15:0]
GPIO port F
PG[15:0]
GPIO port G
4 channels
3 compl. channels
BKIN, ETR as AF
TIM1
4 channels
3 compl. channels
BKIN, ETR as AF
TIM8
MOSI, MISO,
SCK, NSS as AF
SPI1
RX, TX, CTS,
RTS, CK as AF
USART1
Temp. sensor
8 ADC123_INs
common to the 3 ADCs
12-bit ADC1
IF
8 ADC12_INs common
to ADC1 & ADC2
12-bit ADC2
IF
5 ADC3_INs on ADC3
12-bit ADC3
IF
VREF–
VREF+
@ VDDA
APB2: Fmax = 48/72 MHz
APB1: Fmax = 24/36 MHz
TIM2
4 channels, ETR as AF
TIM3
4 channels, ETR as AF
TIM4
4 channels, ETR as AF
TIM5
4 channels as AF
USART2
RX, TX, CTS, RTS,
CK as AF
USART3
RX, TX, CTS, RTS,
CK as AF
UART4
RX,TX as AF
UART5
RX, TX as AF
SPI2/ I2S2
MOSI/SD, MISO
SCK/CK, MCK, NSS/WS as AF
SPI3 / I2S3
MOSI/SD, MISO
SCK/CK, MCK, NSS/WS as AF
I2C1
SCL, SDA, SMBA as AF
I2C2
SCL, SDA, SMBA as AF
bxCAN device
USB 2.0 FS
device
USBDP/CAN_TX
USBDM/CAN_RX
SRAM 512 B
WWDG
TIM6
TIM7
IF
12bit DAC1
12bit DAC 2
DAC_OUT1 as AF
DAC_OUT2 as AF
@VDDA
ai14666f

§ 3.3 RCC(Reset & Clock Control)

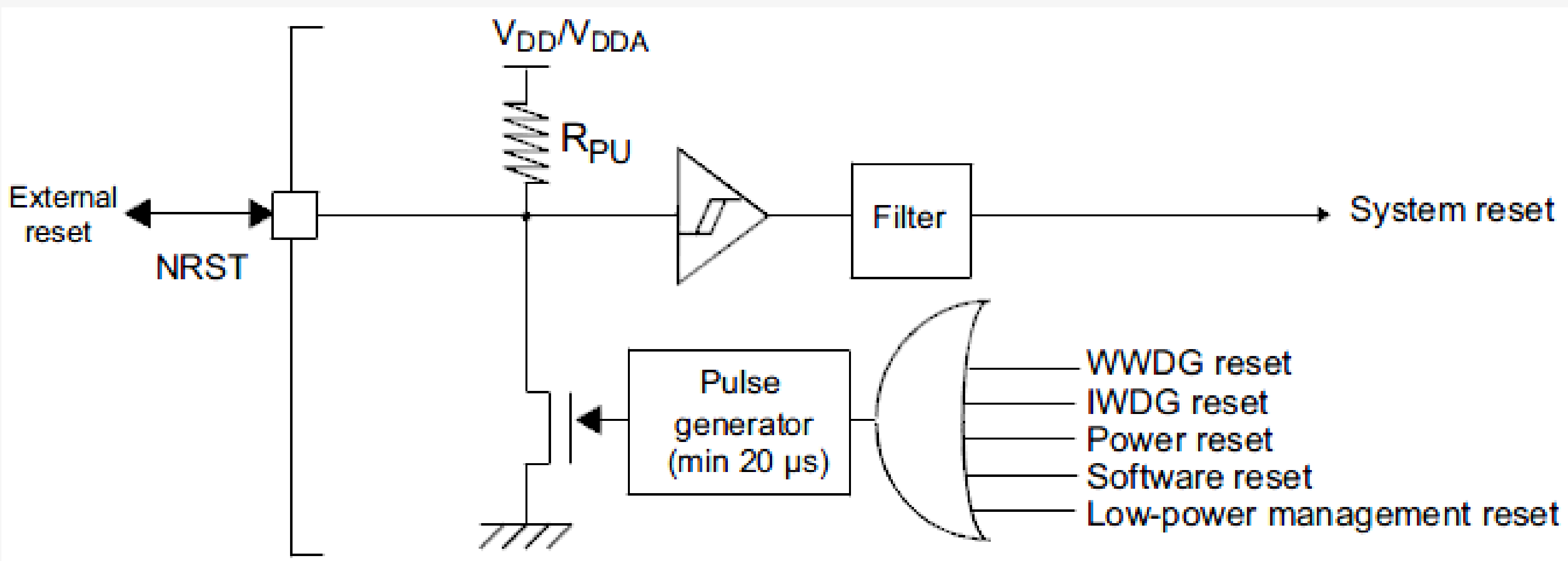
1）Chapter 8 of STM32F103 Reference Manual

§ 3.3 RCC(Reset & Clock Control)

2）多个复位源

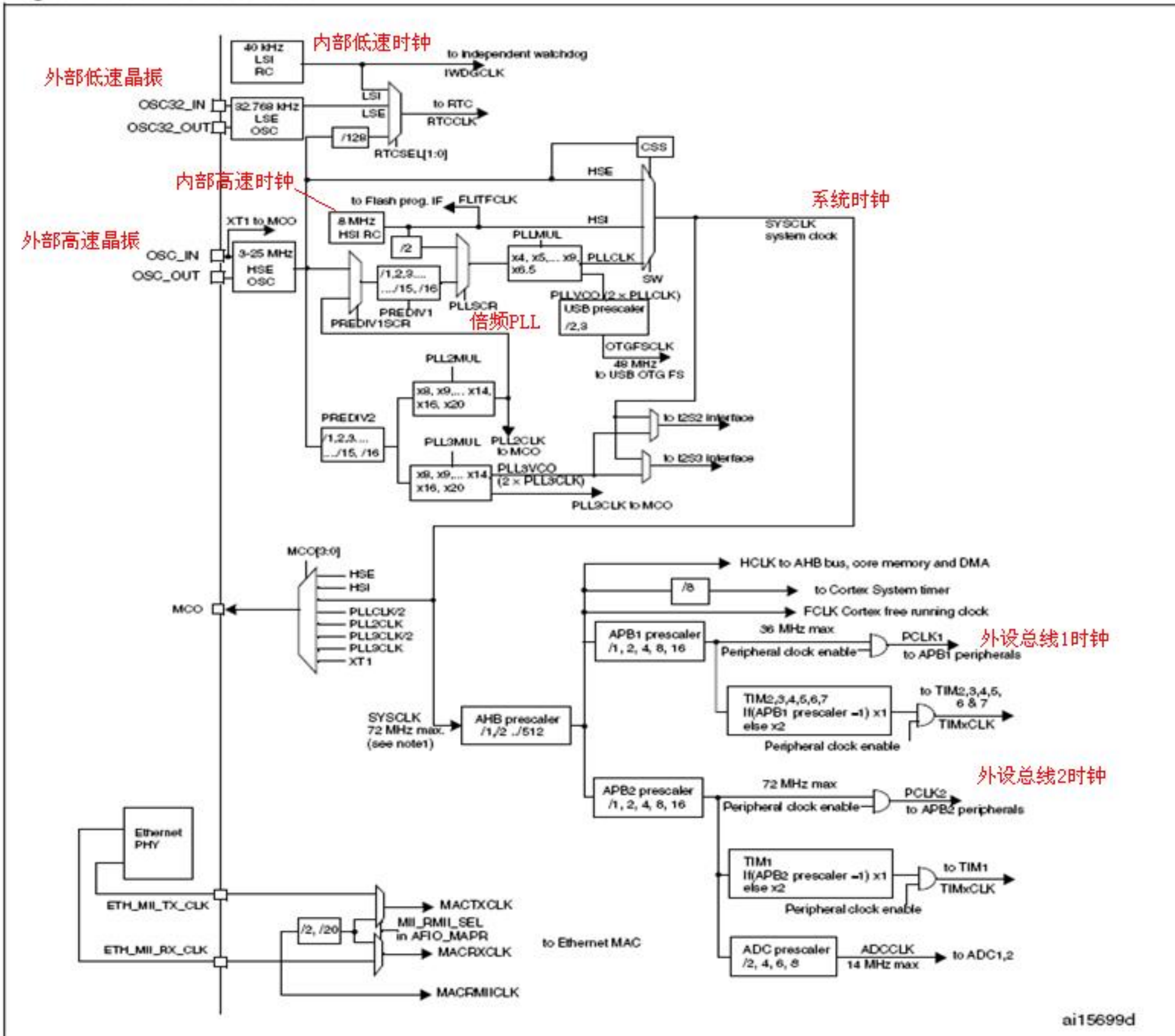
外部复位、上电复位、看门狗复位、软件复位、低功耗管理复位；

复位后，CPU从地址0x0000,0000处开始执行；内部寄存器为默认值或随机值。



3）时钟电路的组成

Figure 11. Clock tree



§ 3.3 RCC(Reset & Clock Control)

4）时钟电路的工作方式

- 为什么同时设置内、外两套时钟振荡电路？
- 为什么时钟电路都包含低频、高频两种时钟？
- 主时钟是如何设置的？

  内部锁相环PLL的作用：把时钟倍频到72MHz
- 外设总线1、外设总线2、定时器等工作时钟是如何设置的？
- 上电启动时，自动选用内部高速时钟；
- 上电后，通过软件（时钟初始化）来选择外部时钟等；
- 设置方式：根据寄存器的定义而设置。

浙江大学 ZheJiang University

5）RCC的寄存器

8.3.1 Clock control register (RCC_CR)

Address offset: 0x00

Reset value: 0x0000 XX83 where X is undefined.

Access: no wait state, word, half-word and byte access

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved		PLL3 RDY	PLL3 ON	PLL2 RDY	PLL2 ON	PLLRDY	PLLON	Reserved				CSSON	HSEBYP	HSERDY	HSEON
		r	rw	r	rw	r	rw					rw	rw	r	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
HSICAL[7:0]								HSITRIM[4:0]					Res.	HSIRDY	HSION
r	r	r	r	r	r	r	r	rw	rw	rw	rw	rw		r	rw

Bits 31:30 Reserved, must be kept at reset value.

Bit 29 **PLL3RDY**: PLL3 clock ready flag

Set by hardware to indicate that the PLL3 is locked.
0: PLL3 unlocked
1: PLL3 locked

Bit 28 **PLL3ON**: PLL3 enable

Set and cleared by software to enable PLL3.
Cleared by hardware when entering Stop or Standby mode.
0: PLL3 OFF
1: PLL3 ON

5）RCC的寄存器

Bit 27 **PLL2RDY**: PLL2 clock ready flag

Set by hardware to indicate that the PLL2 is locked.
0: PLL2 unlocked
1: PLL2 locked

Bit 26 **PLL2ON**: PLL2 enable

Set and cleared by software to enable PLL2.
Cleared by hardware when entering Stop or Standby mode. This bit can not be cleared if the PLL2 clock is used indirectly as system clock (i.e. it is used as PLL clock entry that is used as system clock).
0: PLL2 OFF
1: PLL2 ON

Bit 25 **PLLRDY**: PLL clock ready flag

Set by hardware to indicate that the PLL is locked.
0: PLL unlocked
1: PLL locked

Bit 24 **PLLON**: PLL enable

Set and cleared by software to enable PLL.
Cleared by hardware when entering Stop or Standby mode. This bit can not be reset if the PLL clock is used as system clock or is selected to become the system clock. Software must disable the USB OTG FS clock before clearing this bit.
0: PLL OFF
1: PLL ON

Bits 23:20 Reserved, must be kept at reset value.

Bit 19 **CSSON**: Clock security system enable

Set and cleared by software to enable the clock security system. When CSSON is set, the clock detector is enabled by hardware when the HSE oscillator is ready, and disabled by hardware if a HSE clock failure is detected.

0: Clock detector OFF

1: Clock detector ON (Clock detector ON if the HSE oscillator is ready, OFF if not)

Bit 18 **HSEBYP**: External high-speed clock bypass

Set and cleared by software to bypass the oscillator with an external clock. The external clock must be enabled with the HSEON bit set, to be used by the device. The HSEBYP bit can be written only if the HSE oscillator is disabled.

0: external 3-25 MHz oscillator not bypassed

1: external 3-25 MHz oscillator bypassed with external clock

Bit 17 **HSERDY**: External high-speed clock ready flag

Set by hardware to indicate that the HSE oscillator is stable. This bit needs 6 cycles of the HSE oscillator clock to fall down after HSEON reset.

0: HSE oscillator not ready

1: HSE oscillator ready

Bit 16 **HSEON**: HSE clock enable

Set and cleared by software.

Cleared by hardware to stop the HSE oscillator when entering Stop or Standby mode. This bit cannot be reset if the HSE oscillator is used directly or indirectly as the system clock.

0: HSE oscillator OFF

1: HSE oscillator ON

Bits 15:8 **HSICAL[7:0]**: Internal high-speed clock calibration

These bits are initialized automatically at startup.

5）RCC的寄存器

Bits 7:3 **HSITRIM[4:0]**: Internal high-speed clock trimming

These bits provide an additional user-programmable trimming value that is added to the HSICAL[7:0] bits. It can be programmed to adjust to variations in voltage and temperature that influence the frequency of the internal HSI RC.

The default value is 16, which, when added to the HSICAL value, should trim the HSI to 8 MHz ± 1%. The trimming step ($F_{hsitrim}$) is around 40 kHz between two consecutive HSICAL steps.

Bit 2 Reserved, must be kept at reset value.

Bit 1 **HSIRDY**: Internal high-speed clock ready flag

Set by hardware to indicate that internal 8 MHz RC oscillator is stable. After the HSION bit is cleared, HSIRDY goes low after 6 internal 8 MHz RC oscillator clock cycles.

0: Internal 8 MHz RC oscillator not ready

1: Internal 8 MHz RC oscillator ready

Bit 0 **HSION**: Internal high-speed clock enable

Set and cleared by software.

Set by hardware to force the internal 8 MHz RC oscillator ON when leaving Stop or Standby mode or in case of failure of the external 3-25 MHz oscillator used directly or indirectly as system clock. This bit can not be cleared if the internal 8 MHz RC is used directly or indirectly as system clock or is selected to become the system clock.

0: Internal 8 MHz RC oscillator OFF

1: Internal 8 MHz RC oscillator ON

6）RCC操作代码

```
typedef struct
{
    __IO uint32_t CR;
    __IO uint32_t CFGR;
    __IO uint32_t CIR;
    __IO uint32_t APB2RSTR;
    __IO uint32_t APB1RSTR;
    __IO uint32_t AHBENR;
    __IO uint32_t APB2ENR;
    __IO uint32_t APB1ENR;
    __IO uint32_t BDCR;
    __IO uint32_t CSR;
} RCC_TypeDef;

/********** GPIOB管脚的内存对应地址 ********/
#define PERIPH_BASE            ((uint32_t)0x40000000)
#define APB2PERIPH_BASE        (PERIPH_BASE + 0x10000)
#define GPIOF_BASE             (APB2PERIPH_BASE + 0x1c00)
#define GPIOF                ((GPIO_TypeDef *) GPIOF_BASE)

/************* RCC时钟 <*************/
#define AHBPERIPH_BASE         (PERIPH_BASE + 0x20000)
#define RCC_BASE               (AHBPERIPH_BASE + 0x1000)
#define RCC                  ((RCC_TypeDef *) RCC_BASE)
```

定义 RCC的寄存器，其每一位的含义

参见 技术手册的 8.3 RCC registers

6）RCC操作代码

```
/***** 以下是关于RCC时钟 详细请见《STM32F10XXX参考手册》6.3节RCC寄存器描述 *******/
 unsigned char sws = 0;
 RCC->CR |= 0X00010000; //使能外部高速时钟HSEON
 while(!(RCC->CR>>17));  //将RCC_CR寄存器的值右移17位， 等待HSERDY就绪， 即外部时钟就绪

/* 因为手册有要求APB1时钟频率不超过36MHZ， 而在STM32中最大为72MHZ */
/* 为了保证最大速度， 我们这里设置成2分频 */
/* 设置寄存器CFGR里的8-10位的值为100 */
// RCC->CFGR = 0x00000400;

/* 寄存器CFGR的18-21四个bit位配置成以下值,则PLL就会设置成对应的值:
0000: PLL 2倍频输出   1000: PLL 10倍频输出
0001: PLL 3倍频输出   1001: PLL 11倍频输出
0010: PLL 4倍频输出   1010: PLL 12倍频输出
0011: PLL 5倍频输出   1011: PLL 13倍频输出
0100: PLL 6倍频输出   1100: PLL 14倍频输出
0101: PLL 7倍频输出   1101: PLL 15倍频输出
0110: PLL 8倍频输出   1110: PLL 16倍频输出
0111: PLL 9倍频输出   1111: PLL 16倍频输出
我们在这里， 因为STM32神舟I号上的晶振是8MHZ的， 配置成9倍输出就能达到STM32最大72MHZ工作频率*/

RCC->CFGR |= 7<<18; //本例程希望设置成40MHZ的工作频率， 我们在这里尝试一下
                    //2右移动18位， 即0011使得PLL获得5倍频输出， 外部晶振是8MHZ
                    //乘以4就是40MHZ了
RCC->CFGR |= 1<<16; //PLLSRC设置成1， 使得HSE时钟作为PLL输入时钟
RCC->CR |= 1<<24; //将PLL使能
FLASH->ACR|=0x32;

while(!(RCC->CR>>25)); //监控寄存器CR的PLLRDY位， 等待PLL时钟就绪

RCC->CFGR |= 1<<1;  //将时钟切换寄存器配置成用PLL输出作为系统时钟

  while(sws != 0x2)  //等待CFGR寄存器的2， 3位为10， 系统正式切换到了PLL输出作为时钟
{
  sws = RCC->CFGR>>2;  // 将CFGR寄存器右移2位， 将2， 3位sws状态移出来，
                       // 详情请见《STM32F10XXX参考手册》 54页
  sws &= 0x3;       //这里的0x3为二进制的11， 这个whlie循环设计的一个算法， 为了判断sws是不是为10
}
```

§3.4 STM32F103的上电启动过程

① 选择启动模式

CPU复位后的第4个SYSCLK上升沿BOOT1/0引脚的电平被锁存。

BOOT1	BOOT0	启动模式	启动地址	作用
x	0	FLASH	0x0000,0000	正常运行程序
0	1	BootLoader	0x1FFF,F000	ISP, IAP功能
1	1	SRAM	0x2000,0000	调试小段代码

② 启动入口

在启动地址重映射、延时后，CPU从0x0000,0000获取堆栈栈顶的地址，并从0x0000,0004所指示的地址开始执行代码。

§3.4 STM32F103的上电启动过程

③ STM32F103在MDK中的启动代码

- 启动代码：MCU从“复位”到“开始执行main函数”（称为启动过程）中所需进行的初始化工作。包括堆栈、堆、中断向量、编译参数设置等，用汇编语言编写。
- 参见“STM32_keil_mdk启动代码发分析.txt”

§3.5 STM32F103的中断系统

一、NVIC嵌套中断向量控制器

1）STM32F1xx的NVIC（Nested vectored interrupt controller）管理68 个中断源(不包括 Cortex-M3内核的16个中断源)，有16个优先级可供选择；

2）中断向量表：即中断入口地址

Table 61. Vector table for connectivity line devices

Position	Priority	Type of priority	Acronym	Description	Address
-	-	-	-	Reserved	0x0000_0000
-	-3	fixed	Reset	Reset	0x0000_0004
-	-2	fixed	NMI	Non maskable interrupt. The RCC Clock Security System (CSS) is linked to the NMI vector.	0x0000_0008
-	-1	fixed	HardFault	All class of fault	0x0000_000C

§3.5 STM32F103的中断系统

2）中断向量表：即中断入口地址

Position	Priority	Type of priority	Acronym	Description	Address
-	0	settable	MemManage	Memory management	0x0000_0010
-	1	settable	BusFault	Pre-fetch fault, memory access fault	0x0000_0014
-	2	settable	UsageFault	Undefined instruction or illegal state	0x0000_0018
-	-	-	-	Reserved	0x0000_001C - 0x0000_002B
-	3	settable	SVCall	System service call via SWI instruction	0x0000_002C
-	4	settable	Debug Monitor	Debug Monitor	0x0000_0030
-	-	-	-	Reserved	0x0000_0034
-	5	settable	PendSV	Pendable request for system service	0x0000_0038
-	6	settable	SysTick	System tick timer	0x0000_003C

浙江大学 ZheJiang University

§3.5 STM32F103的中断系统

2）中断向量表：即中断入口地址

0	7	settable	WWDG	Window Watchdog interrupt	0x0000_0040
1	8	settable	PVD	PVD through EXTI Line detection interrupt	0x0000_0044
2	9	settable	TAMPER	Tamper interrupt	0x0000_0048
3	10	settable	RTC	RTC global interrupt	0x0000_004C
4	11	settable	FLASH	Flash global interrupt	0x0000_0050
5	12	settable	RCC	RCC global interrupt	0x0000_0054
6	13	settable	EXTI0	EXTI Line0 interrupt	0x0000_0058
7	14	settable	EXTI1	EXTI Line1 interrupt	0x0000_005C
⋮					
64	71	settable	CAN2_RX0	CAN2 RX0 interrupts	0x0000_0140
65	72	settable	CAN2_RX1	CAN2 RX1 interrupt	0x0000_0144
66	73	settable	CAN2_SCE	CAN2 SCE interrupt	0x0000_0148
67	74	settable	OTG_FS	USB On The Go FS global interrupt	0x0000_014C

浙江大学 ZheJiang University

§3.5 STM32F103的中断系统

二、外部中断/事件控制器（EXTI）

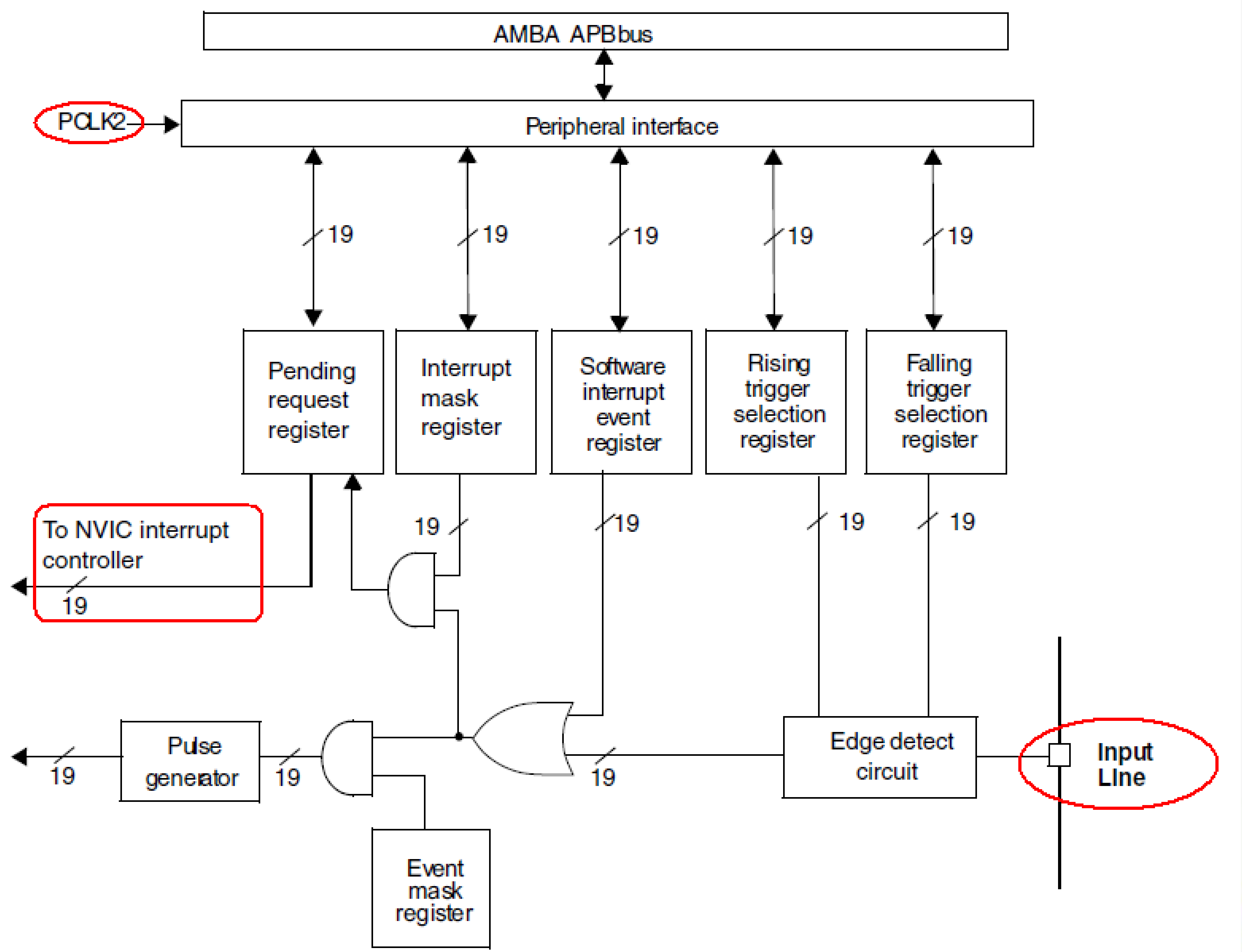
1）特点

- 19个外部中断源+1个软件中断
- 每个中断源相互独立、可屏蔽，有专属的状态指示位
- EXTI外部中断响应快，可低于一个APB2 时钟脉冲.
- EXTI可设置成中断或事件，中断触发条件可选择上升沿、下降沿、或上升下降沿；

浙江大学 ZheJiang University

§3.5 STM32F103的中断系统

2）EXTI控制器组成

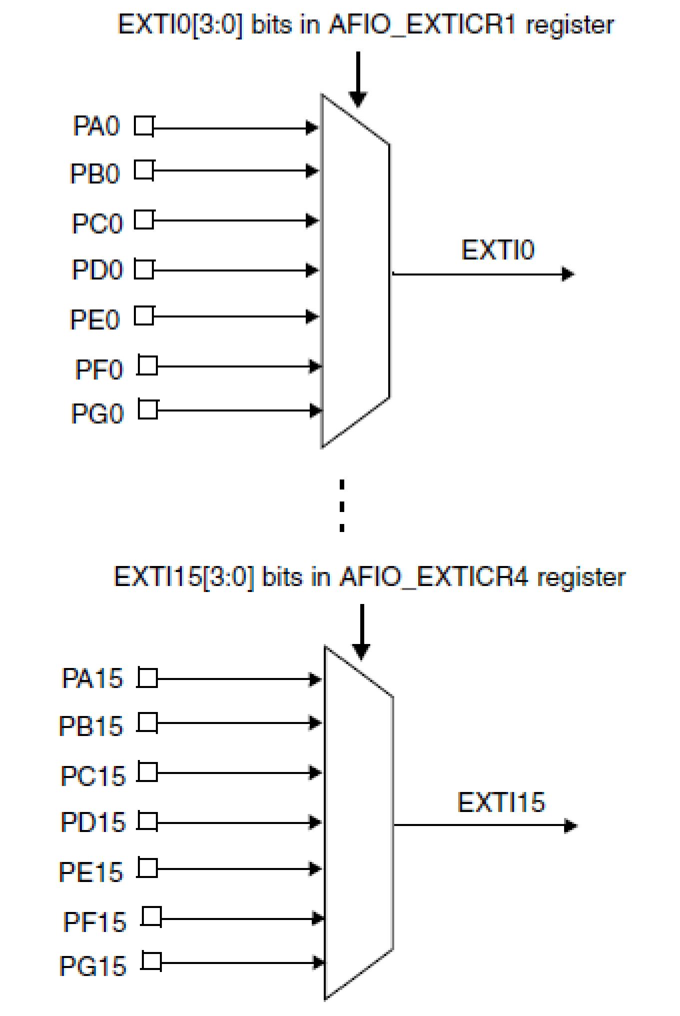


§3.5 STM32F103的中断系统

二、外部中断/事件控制器（EXTI）

3）16个GPIO中断的配置

第n个GPIO中断源EXTIn只能是MCU的所有GPIO端口的第n位中选择其中一个，通过AFIO_EXTICRx寄存器的EXTIn[3:0]来指定。



EXTI line 16 is connected to the PVD output
EXTI line 17 is connected to the RTC Alarm event
EXTI line 18 is connected to the USB Wakeup event
EXTI line 19 is connected to the Ethernet Wakeup event

§3.5 STM32F103的中断系统

三、EXTI控制器的寄存器

① EXTI_IMR：Interrupt mask register

bit19~bit0用于屏蔽 line 19~ line 0上的中断

② EXTI_EMR：Event mask register

bit19~bit0用于屏蔽 line 19~ line 0上的事件

③ EXTI_RTSR：Rising trigger selection register

bit19~bit0用于允许或禁止line 19~ line 0上的上升沿中断；

④ EXTI_FTSR：Falling trigger selection register

bit19~bit0用于允许或禁止line 19~ line 0上的下降沿中断；

⑤ EXTI_SWIER：Software interrupt event register，软件置第n位设置1，当EXTI_IMR(n)为使能，则EXTI_PR(n)置1，并产生中断；

⑥ EXTI_PR：Pending register，记录line 19~ line 0上有效事件发

浙江大学 ZheJiang University

四、中断的库函数

```
stm32f10x_exti.c
    EXTI_DeInit (void)
    EXTI_Init (EXTI_InitTypeDef* EXTI_InitStruct)
    EXTI_StructInit (EXTI_InitTypeDef* EXTI_InitStruct)
    EXTI_GenerateSWInterrupt (uint32_t EXTI_Line)
    EXTI_GetFlagStatus (uint32_t EXTI_Line)
    EXTI_ClearFlag (uint32_t EXTI_Line)
    EXTI_GetITStatus (uint32_t EXTI_Line)
    EXTI_ClearITPendingBit (uint32_t EXTI_Line)
```

§3.5 STM32F103的中断系统

五、中断编程要点

某个软件功能是由中断信号触发而执行的，通常用两部分的代码来实现该功能：

① 中断程序

安排在中断入口处的代码，完成信息记录及简单的处理，并设置信号标记；此处的代码力求简短，以提高整个系统的实时性；

② 中断响应程序

安排在主程序（后台程序）中，判别信号标记，如果被置位，则执行中断响应代码，完成其余的功能。

§ 3.6 STM32F103的DMA功能

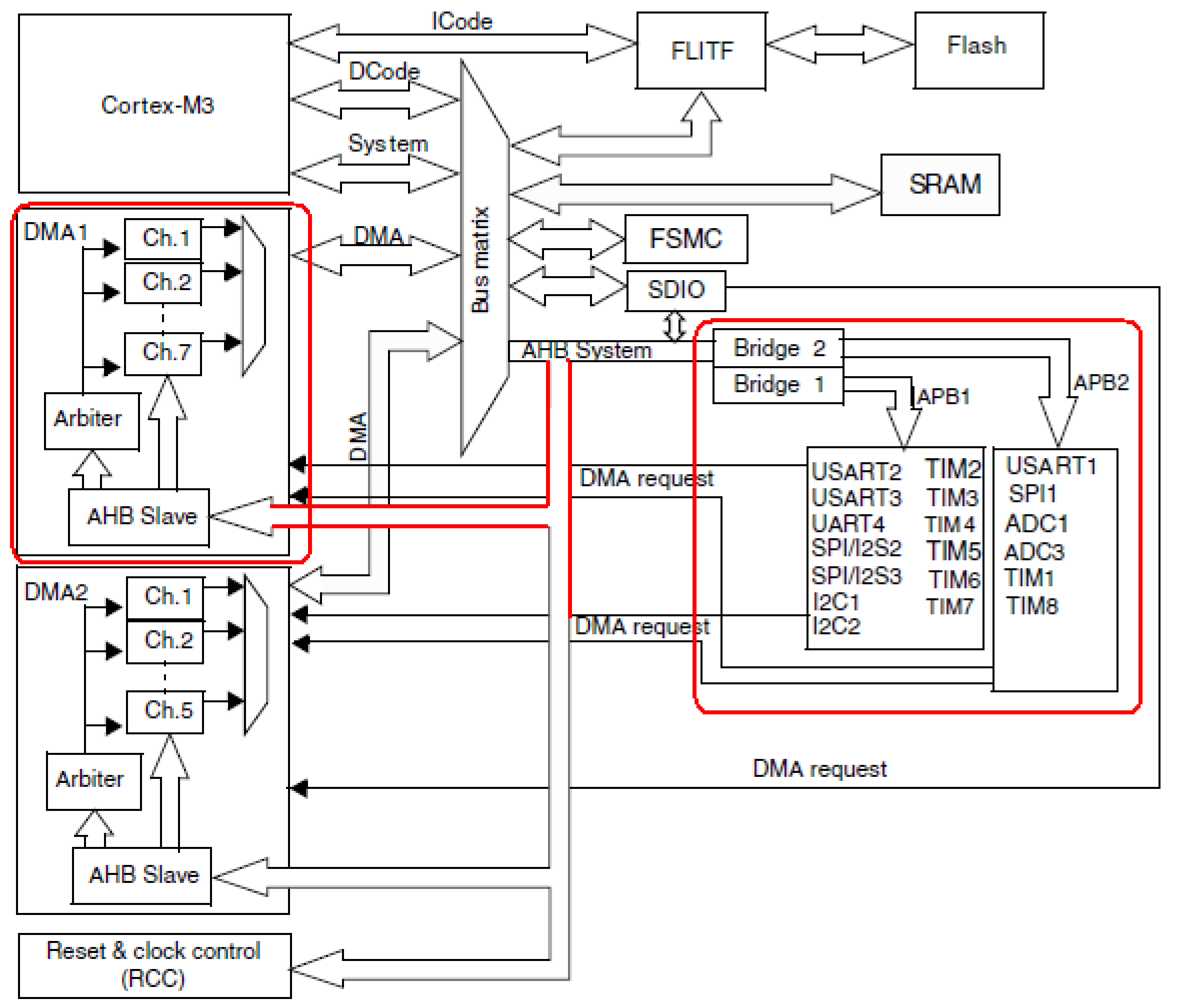
一、DMA特点

- DMA(Direct memory access，直接存储访问）存储器、外设（功能部件）之间的高速数据传输，无需CPU的干预；
- STM32F有2个DMA控制器，共管理12个DMA通道，其中DMA1有7个， DMA2有5个，但STM32F103C8中只有DMA1，7个DMA通道；
- 每个DMA通道具有4个可编程优先级（最高、高、中、低），连接专属的硬件需求源，软件可触发每个通道的DMA传送；
- 每个DMA通道为单向数据传输，2端可以是存储器—存储器、外设—存储器；
- 传送数据以字节、双字节、字为单位，长度可设置，最长64KB，传送方和接收方的数据长度必须一致；
- DMA能产生3个事件（传输到一半、传输完成、传输出错）标记，产生中断。

§ 3.6 STM32F103的DMA功能

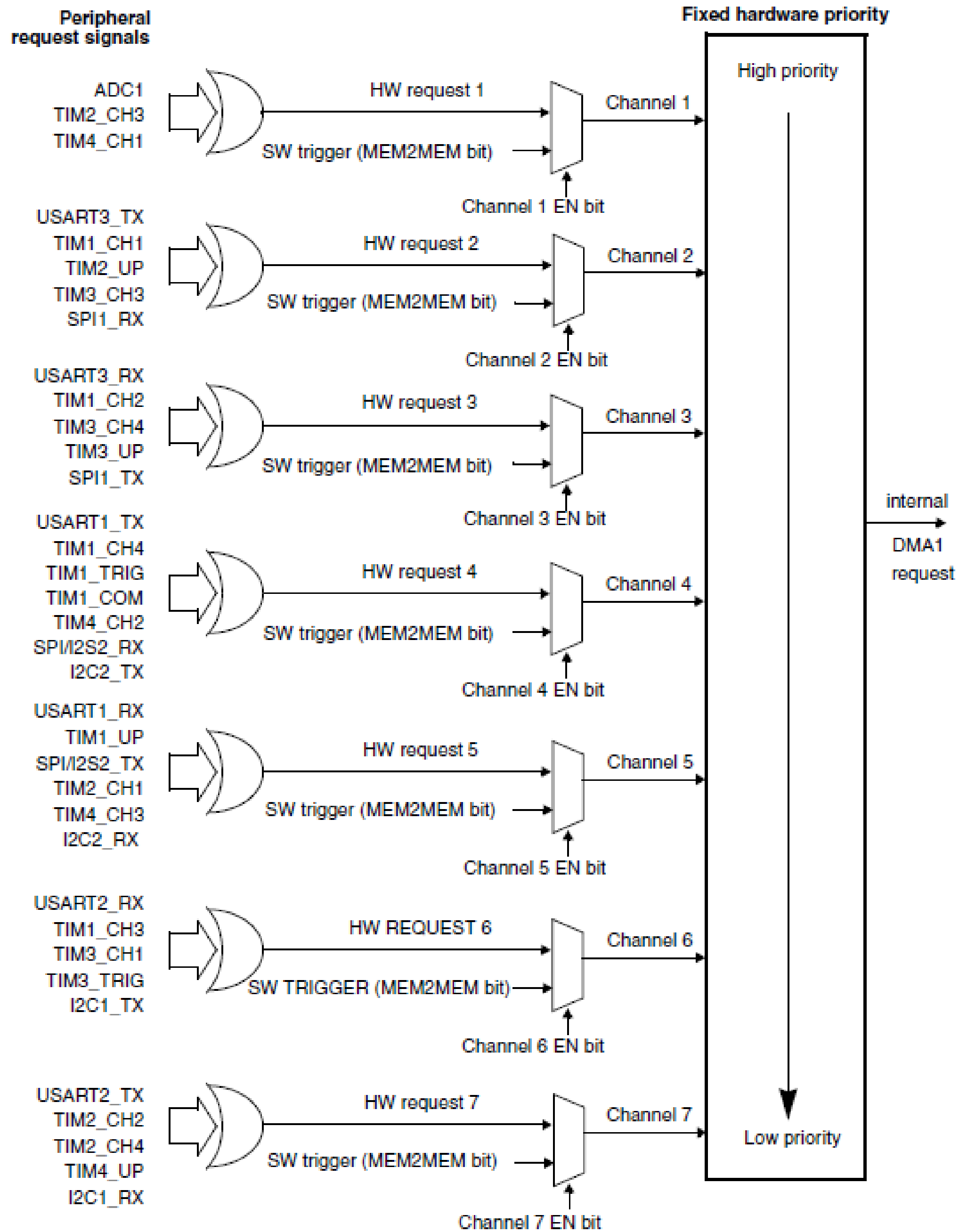
二、DMA组成

Figure 49. DMA block diagram in low-, medium- high- and XL-density devices



二、DMA组成

Figure 50. DMA1 request mapping



§ 3.6 STM32F103的DMA功能

二、DMA组成

Table 78. Summary of DMA1 requests for each channel

Peripherals	Channel 1	Channel 2	Channel 3	Channel 4	Channel 5	Channel 6	Channel 7
ADC1	ADC1	-	-	-	-	-	-
SPI/I²S	-	SPI1_RX	SPI1_TX	SPI2/I2S2_RX	SPI2/I2S2_TX	-	-
USART	-	USART3_TX	USART3_RX	USART1_TX	USART1_RX	USART2_RX	USART2_TX
I²C	-	-	-	I2C2_TX	I2C2_RX	I2C1_TX	I2C1_RX
TIM1	-	TIM1_CH1	-	TIM1_CH4 TIM1_TRIG TIM1_COM	TIM1_UP	TIM1_CH3	-
TIM2	TIM2_CH3	TIM2_UP	-	-	TIM2_CH1	-	TIM2_CH2 TIM2_CH4
TIM3	-	TIM3_CH3	TIM3_CH4 TIM3_UP	-	-	TIM3_CH1 TIM3_TRIG	-
TIM4	TIM4_CH1	-	-	TIM4_CH2	TIM4_CH3	-	TIM4_UP

浙江大学 ZheJiang University

§3.6 STM32F103的DMA功能

三、DMA工作模式

① P2M：外设→存储器，通常事件触发，或软件触发；

② M2M：存储器→存储器，无事件触发，只有软件触发；

③ CIRC：循环模式，连续数据→循环缓冲，传输计数器减为0时，会自动回复初值，继续DMA传输。

§3.6 STM32F103的DMA功能

四、P2M的工作过程

① 当数据源预定义的事件发生时，硬件自动产生X通道的MDA请求；

② DMA控制器接到该请求时，通过DMA通道的优先级比较，选择优先级最高的DMA请求；

③ 从CPU那边夺取内部数据总线的控制权，执行优先级最高而且已就绪的DMA通道的数据传送， 当DMA_CNDTRx 减为0，传送完毕，释放内部总线的控制权，产生DMA传输完毕的中断信号。

§ 3.6 STM32F103的DMA功能

五、主要寄存器

① DMA 状态寄存器(DMA_ISR)：描述7个通道的4种状态，传输异常、一半传输完成、全部传输完成、中断产生；

② DMA 设置寄存器(DMA_CCRx)：x=1…7，用于设置数据格式、M2M或CIRC模式启用、优先级、指针是否递增、中断使能、DMA使能等；

③ 存储器地址 (DMA_CMARx)：DMA传输的起点或终点；

④ 外设地址 (DMA_CPARx)：DMA传输的起点或终点；

⑤ 数据长度寄存器(DMA_CNDTRx)：小于65535

§3.6 STM32F103的DMA功能

六、库函数

使用库函数可以绕过寄存器的读写操作，提高编程效率，因为正确理解多个寄存器的每一位功能是一件很费时的工作。

```
stm32f10x_dma.c
  DMA_DeInit (DMA_Channel_TypeDef* DMAy_Channelx)
  DMA_Init (DMA_Channel_TypeDef* DMAy_Channelx, DMA_InitTypeDef* DMA_InitStruct)
  DMA_StructInit (DMA_InitTypeDef* DMA_InitStruct)
  DMA_Cmd (DMA_Channel_TypeDef* DMAy_Channelx, FunctionalState NewState)
  DMA_ITConfig (DMA_Channel_TypeDef* DMAy_Channelx, uint32_t DMA_IT,FunctionalState)
  DMA_SetCurrDataCounter (DMA_Channel_TypeDef* DMAy_Channelx, uint16_t DataNumber)
  DMA_GetCurrDataCounter (DMA_Channel_TypeDef* DMAy_Channelx)
  DMA_GetFlagStatus (uint32_t DMAy_FLAG)
  DMA_ClearFlag (uint32_t DMAy_FLAG)
  DMA_GetITStatus (uint32_t DMAy_IT)
  DMA_ClearITPendingBit (uint32_t DMAy_IT)
```

§3.6 STM32F103的DMA功能

七、应用例 用DMA通道6，把FLASH中32个字节传输到RAM中。

```
DMA_InitTypeDef  DMA_InitStruct;
int main(void)
{ RCC_Configuration( );
   NVIC_Configuration( );
   DMA_DeInit( DMA1_Channel6);
   DMA_InitStruct.MDA_PeripheralBaseAddr =(u32)DAT;// DAT[32]
   DMA_InitStruct.MDA_MemoryBaseAddr =(u32)RAM_BUF;// Target
   DMA_InitStruct.MDA_DIR = DMA_DIR_PeripheralSRC;
   DMA_InitStruct.MDA_BufferSize = 32;
    …
   DMA_InitStruct.MDA_Mode = DMA_Mode_Normal;
   DMA_InitStruct.MDA_Priority = DMA_Priority_High;
   DMA_InitStruct.MDA_M2M = DMA_M2M_Enable;
   DMA_Init(MDA_Channel6, &DMA_InitStruct); //初始化设置

   DMA_ITConfig (DMA_Channel6, DMA_IT_TC,ENABLE); //传输完成后中断
   DMA_Cmd(DMA_Channel6, ENABLE); //DMA6使能，对M2M传输，启动传输

   While( CurrDataCounterEnd !=0) {}; //等待传输结束， 中断程序中把该变量清零
}
```

§3.7 STM32F103C8的最小应用系统

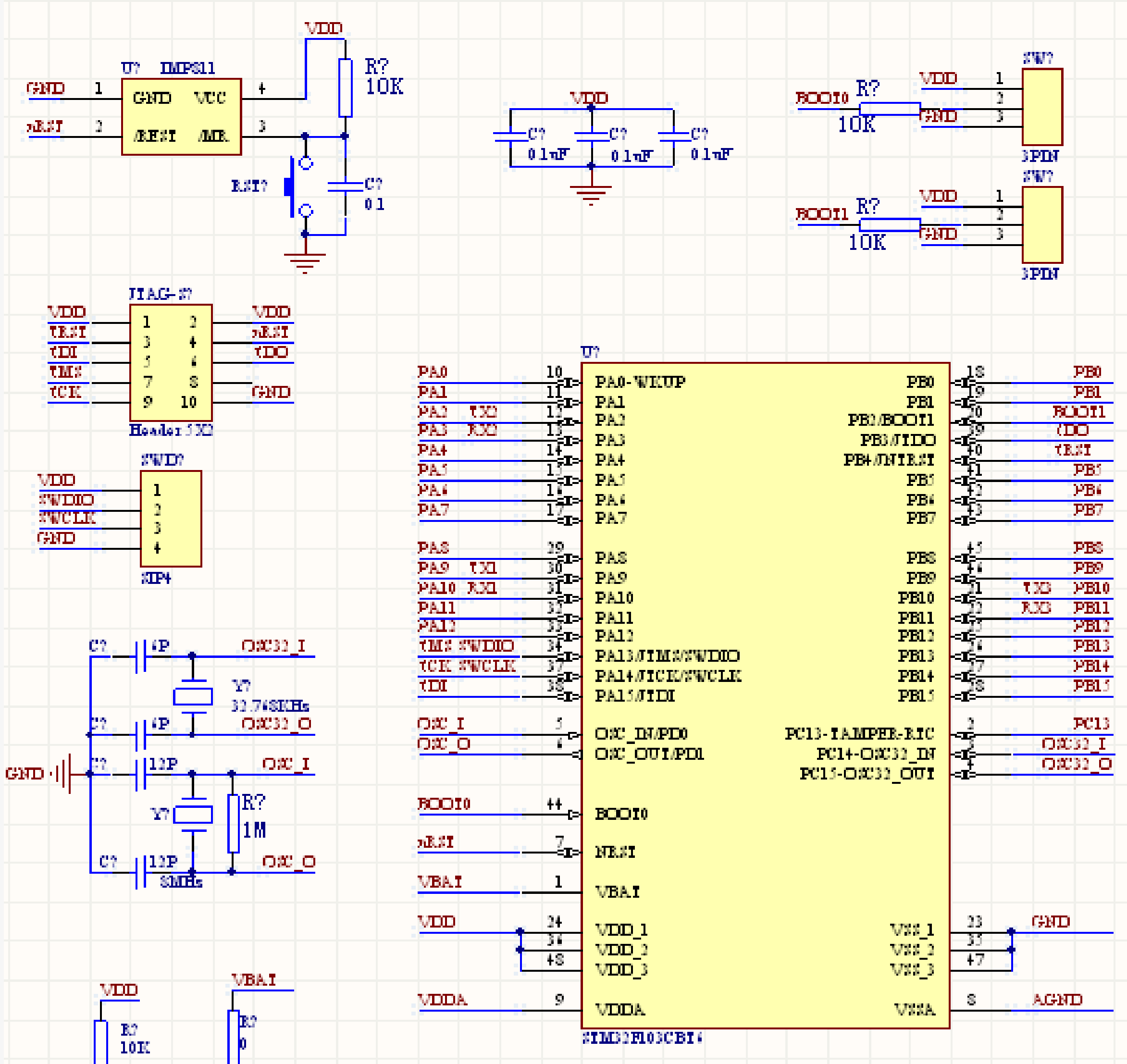
最小应用系统是指能让MCU运行的基本电路；

- STM32F103C 8

  芯片内包含CPU(Cortex-m3)，64KB FLASH，20KB SRAM,CAN, UART等；
- 调试接口：JTAG或SWD
- 复位电路
- 晶振：8MHz，32.768KHz

浙江大学 ZheJiang University

§3.7 STM32F103C8的最小应用系统



BOOT1,B0OT0		BOOT FROM
*	0	FLASH
0	1	SYSTEM BOOT
1	1	SRAM

第十一讲 STM32F103嵌入式微处理器

§1 MCU组成模型和学习方法

§2 STM32F系列及命名方法

§3 STM32F103C8概述

§4 STM32F103的功能部件

§4 STM32F103的功能部件

§4.1 GPIO

§4.2 Timer

§4.3 UART

§4.4 A/D

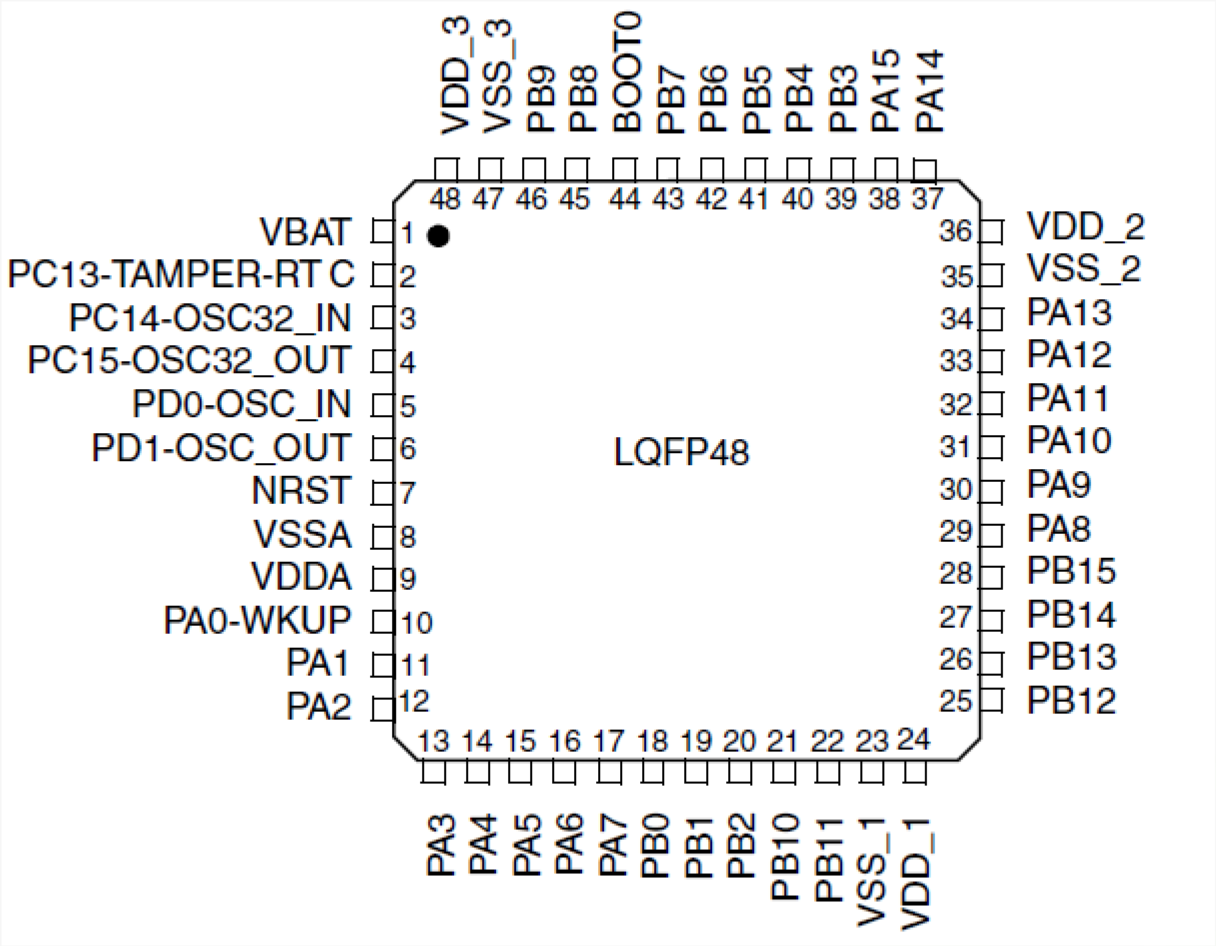
§4.5 D/A

浙江大學
ZheJiang University

§4.1 GPIO

一、引脚分布

STM32F103C8: 48引脚，PA0~15,PB0~15等



§4.1 GPIO

二、主要功能

① DI 数字量输入：高电平1，低电平0，输入中断、唤醒等；

② DO 数字量输出：高电平1，低电平0；

③ AF 其它功能：引脚复用，预定义的功能模块引脚；

三、工作模式

通过设置GPIO的寄存器来选择：

① Input floating、 Input pull-up、Input-pull-down

② Output open-drain、 Output push-pull

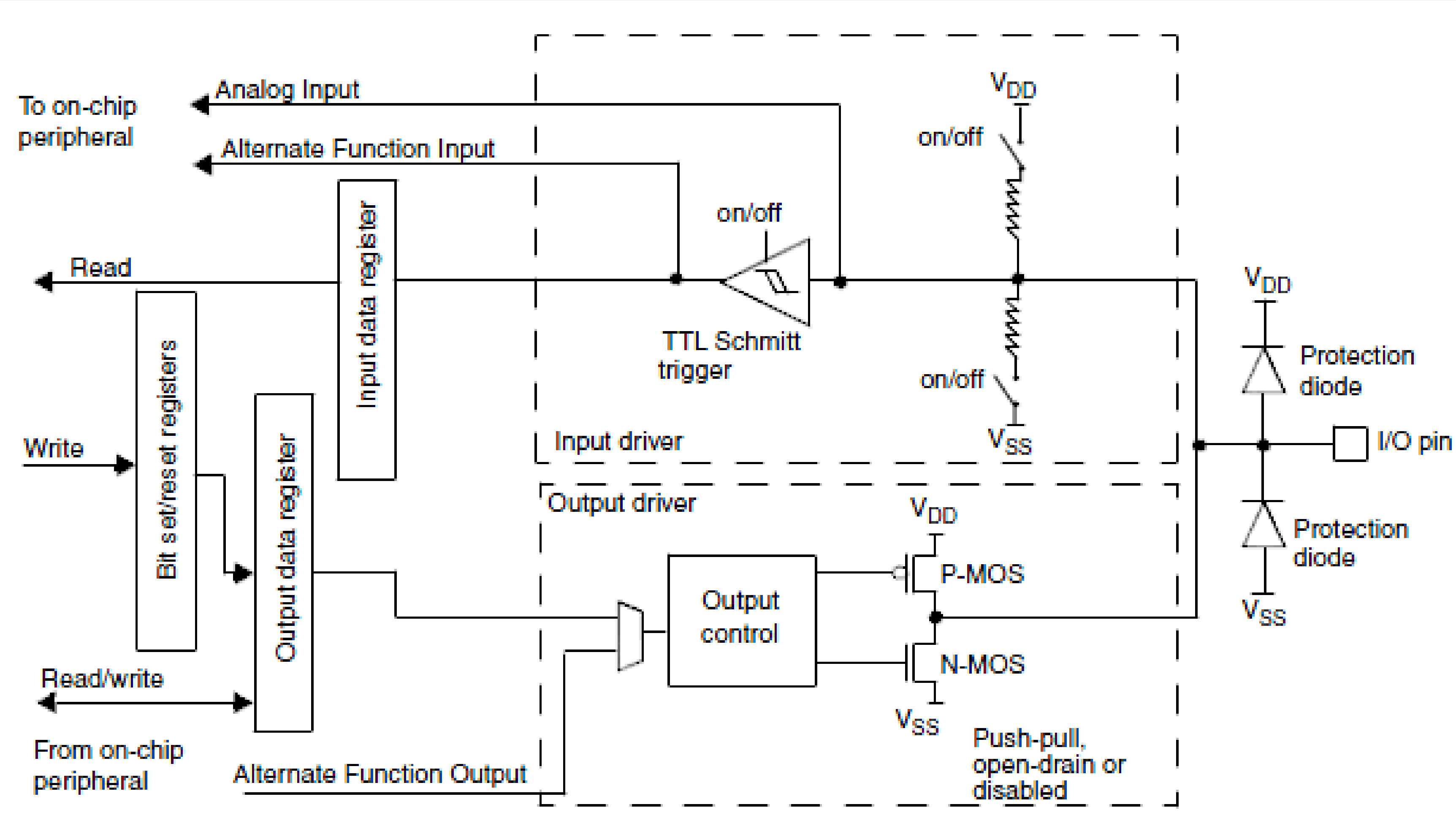
③ Analog

④ Alternate function push-pull、 open-drain

浙江大学 ZheJiang University

§ 4.1 GPIO

四、结构



To on-chip peripheral
Analog Input
Alternate Function Input
Read
Input data register
Bit set/reset registers
Write
Output data register
Read/write
From on-chip peripheral
Alternate Function Output
on/off
TTL Schmitt trigger
Input driver
V_DD
on/off
on/off
V_SS
Output driver
Output control
V_DD
P-MOS
N-MOS
V_SS
Push-pull, open-drain or disabled
V_DD
Protection diode
I/O pin
Protection diode
V_SS

§4.1 GPIO

五、寄存器：由 GPIO 、AFIO 两个寄存器组

Table 59. GPIO register map and reset values

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	GPIOx_CRL	CNF7[1:0]		MODE7[1:0]		CNF6[1:0]		MODE6[1:0]		CNF5[1:0]		MODE5[1:0]		CNF4[1:0]		MODE4[1:0]		CNF3[1:0]		MODE3[1:0]		CNF2[1:0]		MODE2[1:0]		CNF1[1:0]		MODE1[1:0]		CNF0[1:0]		MODE0[1:0]	
	Reset value	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0
0x04	GPIOx_CRH	CNF15[1:0]		MODE15[1:0]		CNF14[1:0]		MODE14[1:0]		CNF13[1:0]		MODE13[1:0]		CNF12[1:0]		MODE12[1:0]		CNF11[1:0]		MODE11[1:0]		CNF10[1:0]		MODE10[1:0]		CNF9[1:0]		MODE9[1:0]		CNF8[1:0]		MODE8[1:0]	
	Reset value	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0
0x08	GPIOx_IDR	Reserved																IDRy															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x0C	GPIOx_ODR	Reserved																ODRy															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x10	GPIOx_BSRR	BR[15:0]																BSR[15:0]															
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x14	GPIOx_BRR	Reserved																BR[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x18	GPIOx_LCKR	Reserved															LCKK	LCK[15:0]															
	Reset value																0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

§ 4.1 GPIO

六、库函数

嵌入式低层编程是通过操作寄存器来完成，非常繁琐；库函数把封装了寄存器操作，程序员可以不需要了解寄存器的细节。

.\StdPeriphDriver\src\stm32f10x_gpio.c

```
stm32f10x_gpio.c
    GPIO_DeInit (GPIO_TypeDef* GPIOx)
    GPIO_AFIODeInit (void)
    GPIO_Init (GPIO_TypeDef* GPIOx, GPIO_InitTypeDef* GPIO_InitStruct)
    GPIO_StructInit (GPIO_InitTypeDef* GPIO_InitStruct)
    GPIO_ReadInputDataBit (GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)
    GPIO_ReadInputData (GPIO_TypeDef* GPIOx)
    GPIO_ReadOutputDataBit (GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)
    GPIO_ReadOutputData (GPIO_TypeDef* GPIOx)
    GPIO_SetBits (GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)
    GPIO_ResetBits (GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)
    GPIO_WriteBit (GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin, BitAction BitVal)
    GPIO_Write (GPIO_TypeDef* GPIOx, uint16_t PortVal)
    GPIO_PinLockConfig (GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)
    GPIO_EventOutputConfig (uint8_t GPIO_PortSource, uint8_t GPIO_PinSource)
    GPIO_EventOutputCmd (FunctionalState NewState)
    GPIO_PinRemapConfig (uint32_t GPIO_Remap, FunctionalState NewState)
    GPIO_EXTILineConfig (uint8_t GPIO_PortSource, uint8_t GPIO_PinSource)
    GPIO_ETH_MediaInterfaceConfig (uint32_t GPIO_ETH_MediaInterface)
```

七、应用例

```
void LED_Init(void)
{
        GPIO_InitTypeDef  GPIO_InitStructure; //描述GPIO寄存器的结构
        GPIO_InitStructure.GPIO_Pin = PIN_LED; //选择LED控制的引脚_2
        GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz; //速度_3
        GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;//模式_4
        RCC_APB2PeriphClockCmd(RCC_LED, ENABLE); //给引脚加时钟_1
        GPIO_Init(GPIO_LED, &GPIO_InitStructure);  //初始化设置
}
```

.\8--ARM 例程\1 ARM例程\stm32例程\1，LED\流水灯\Users\Src\led.c

§ 4.1 GPIO

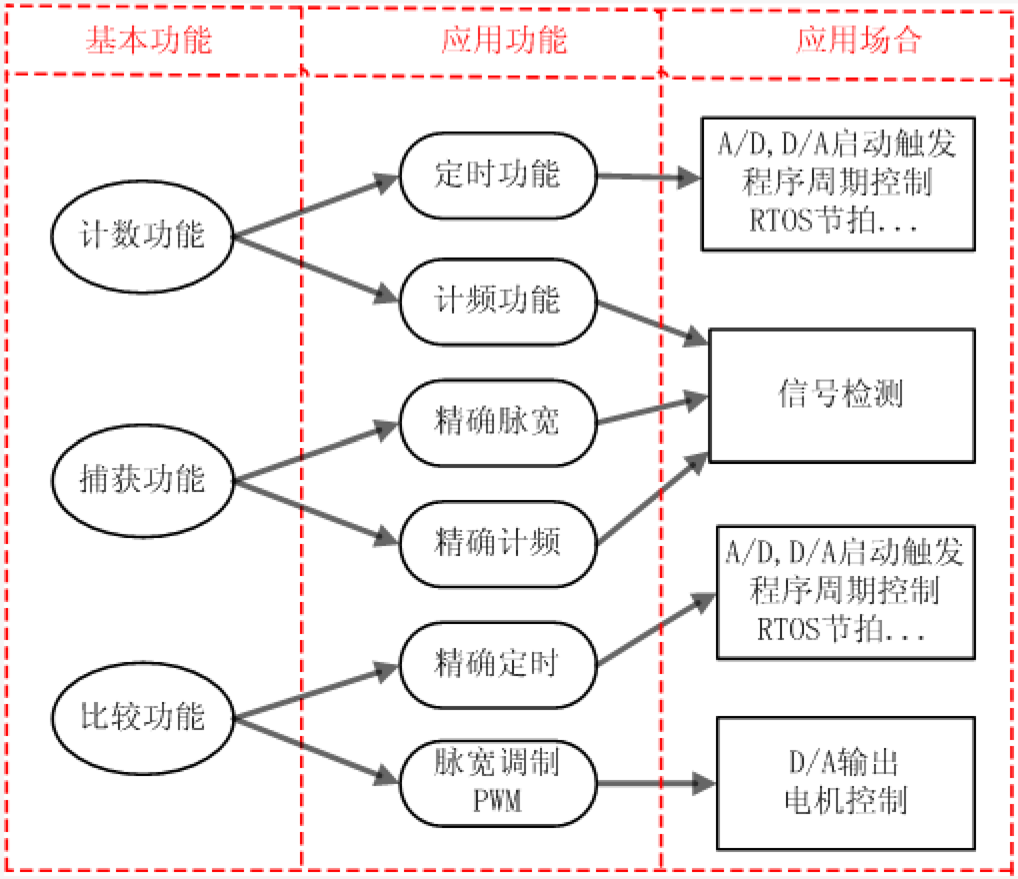
七、应用例

```
void LED_Sets(uint8_t data)
{
       uint16_t  setValue;

       setValue = GPIO_ReadOutputData(GPIO_LED); //调用API
        //((uint16_t)GPIOx->ODR)-->setValue

       setValue &= 0x00ff;
        setValue |= (uint16_t)data << 8;
       //data-->setValue

      GPIO_Write(GPIO_LED, setValue);              //调用API
       //setValue-->((uint16_t)GPIOx->ODR)       //等效的寄存器操作
}
```

.\8--ARM 例程\1 ARM例程\stm32例程\1，LED\流水灯\Users\Src\led.c

§ 4.2 Timer

一、定时器的功能和应用



二、STM32F103C8的定时器

1）4个通用16位定时器

- 具有4个比较、捕获通道；
- 计数模式：上升、下降、上升和下降；（对比51）
- T1 为增强型，带互补输出，紧急停止等功能，可用于电机控制
- 中断产生条件：定时器溢出、比较值相等、捕获引脚有指定的跳变

Table 4. Timer feature comparison

Timer	Counter resolution	Counter type	Prescaler factor	DMA request generation	Capture/compare channels	Complementary outputs
TIM1	16-bit	Up, down, up/down	Any integer between 1 and 65536	Yes	4	Yes
TIM2, TIM3, TIM4	16-bit	Up, down, up/down	Any integer between 1 and 65536	Yes	4	No

二、STM32F103C8的定时器

2）1个24位systick timer

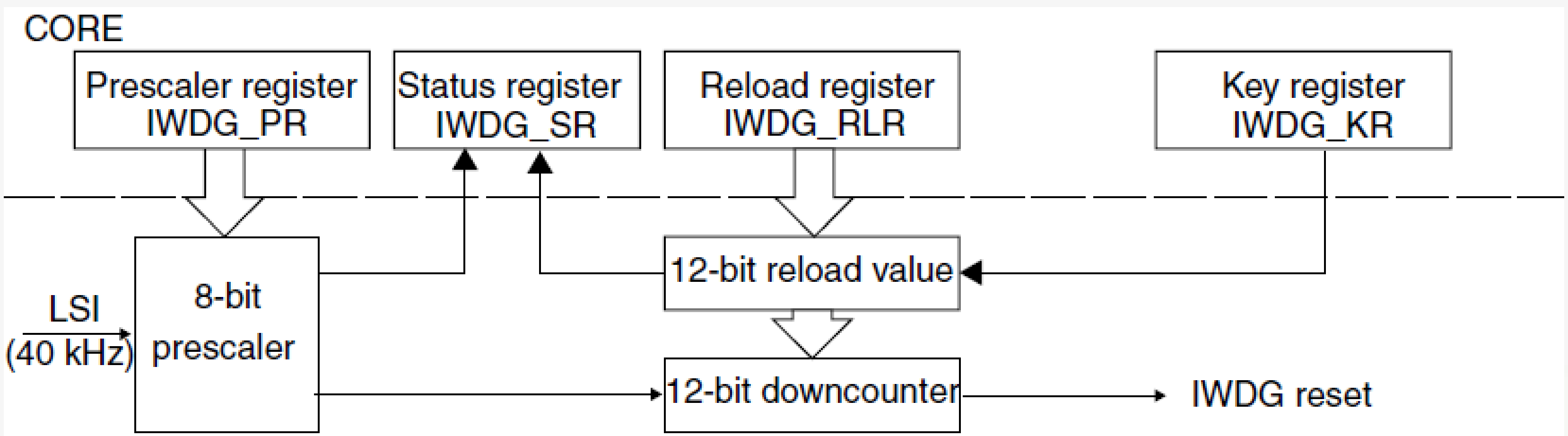
- 为OS配置，产生OS的任务扫描节拍；
- 也可用作普通的减计数器；
- 特征：
  - 24位减计数器；
  - 计数值可自动加载；
  - 计数值为0时产生中断，该中断可屏蔽；
  - 输入时钟可编程；

浙江大学 ZheJiang University

二、STM32F103C8的定时器

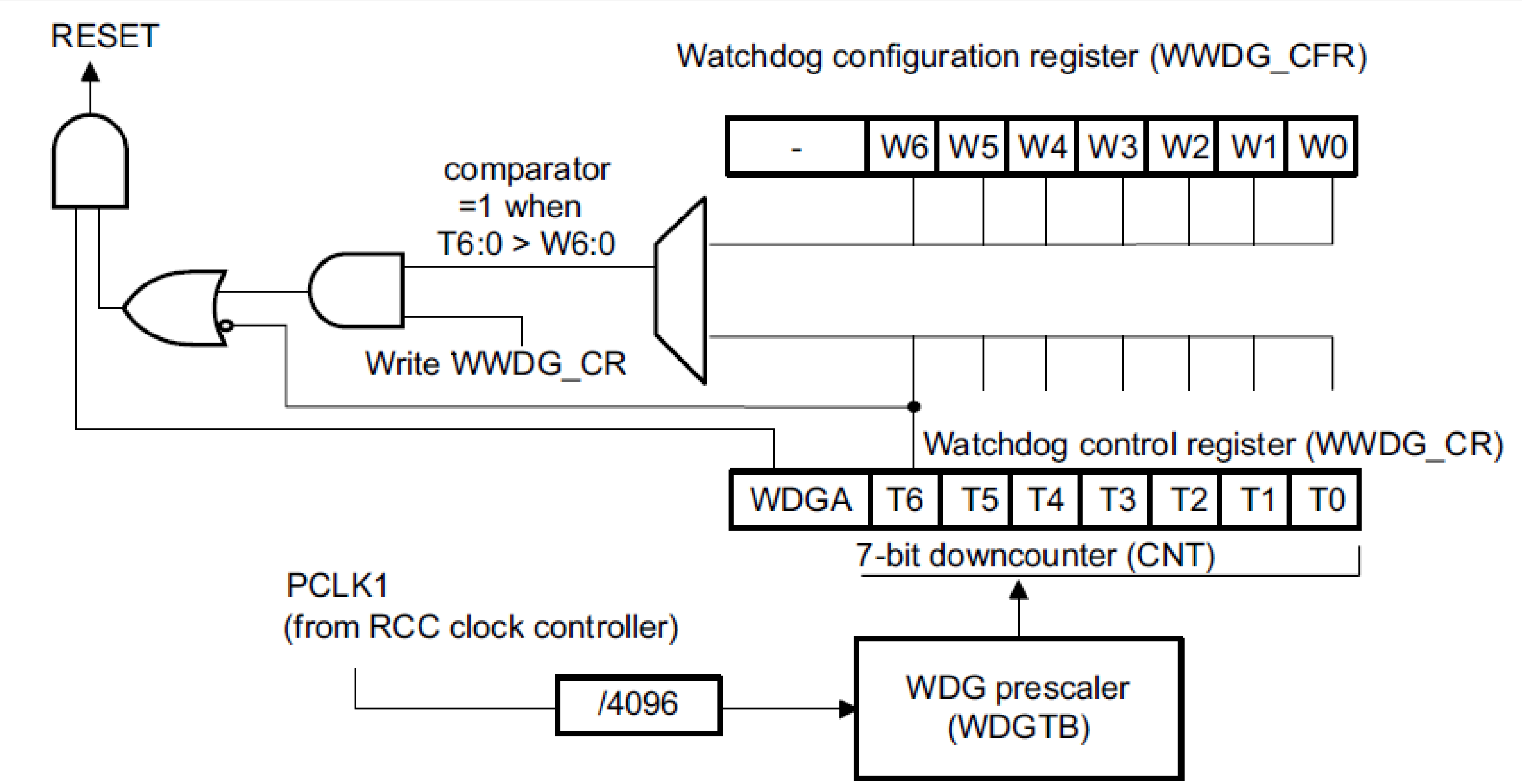
3）2个看门狗定时器

- ➢ 看门狗作用：溢出后自动复位MCU，防止程序异常跑飞
- ➢ IWDT独立看门狗定时器
- ● 8位预分频计数器+12位减计数器；
- ● 内部40kHz时钟源，独立于系统时钟；



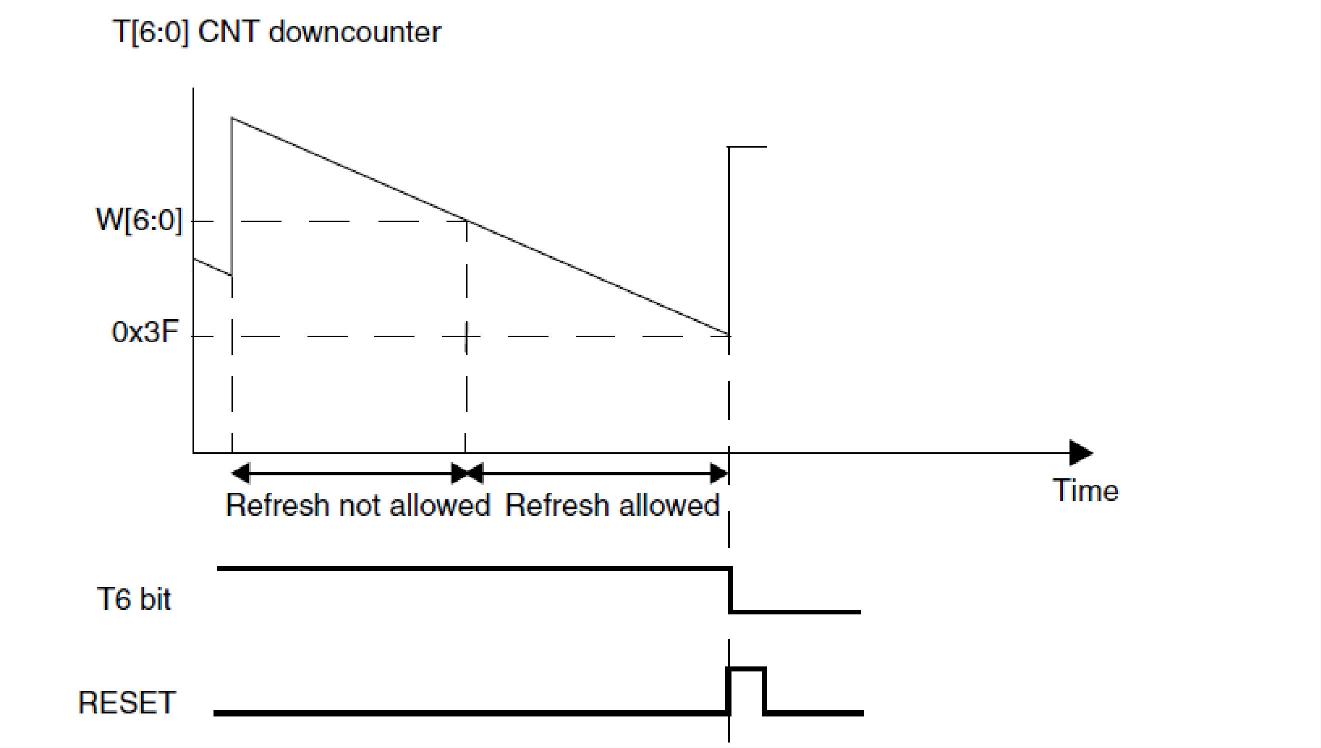
二、STM32F103C8的定时器

- ➢ WWDT窗口看门狗定时器：组成
- 7位减计数器；
- 时钟源为系统时钟；



二、STM32F103C8的定时器

- WWDT窗口看门狗定时器：工作过程

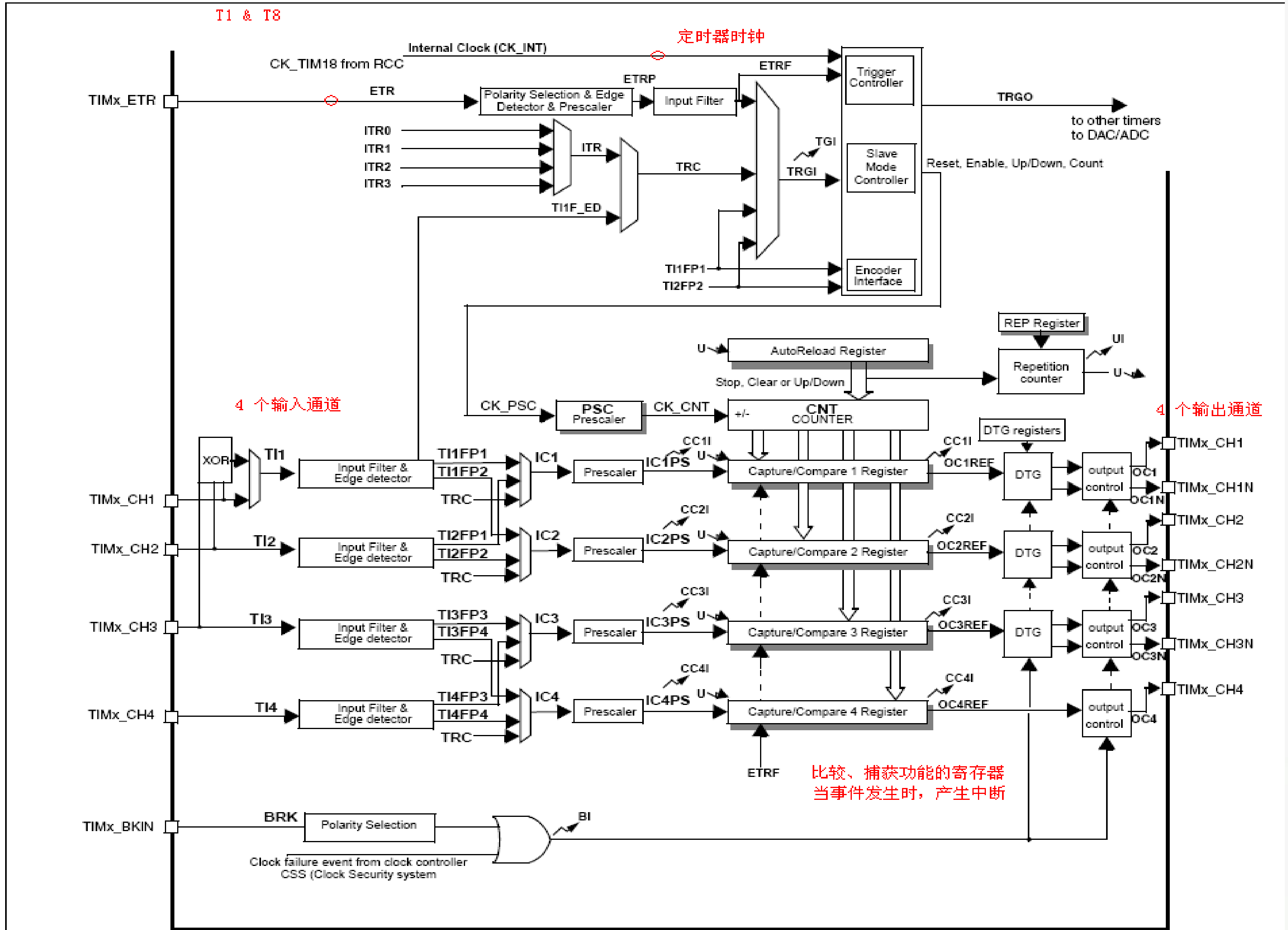


- 溢出间隔：

tWWDG = tPCLK1 × 4096 × 2 ^ WDGTB[1:0] × (T[5:0] + 1) (ms)

三、通用定时器——T1定时器的组成

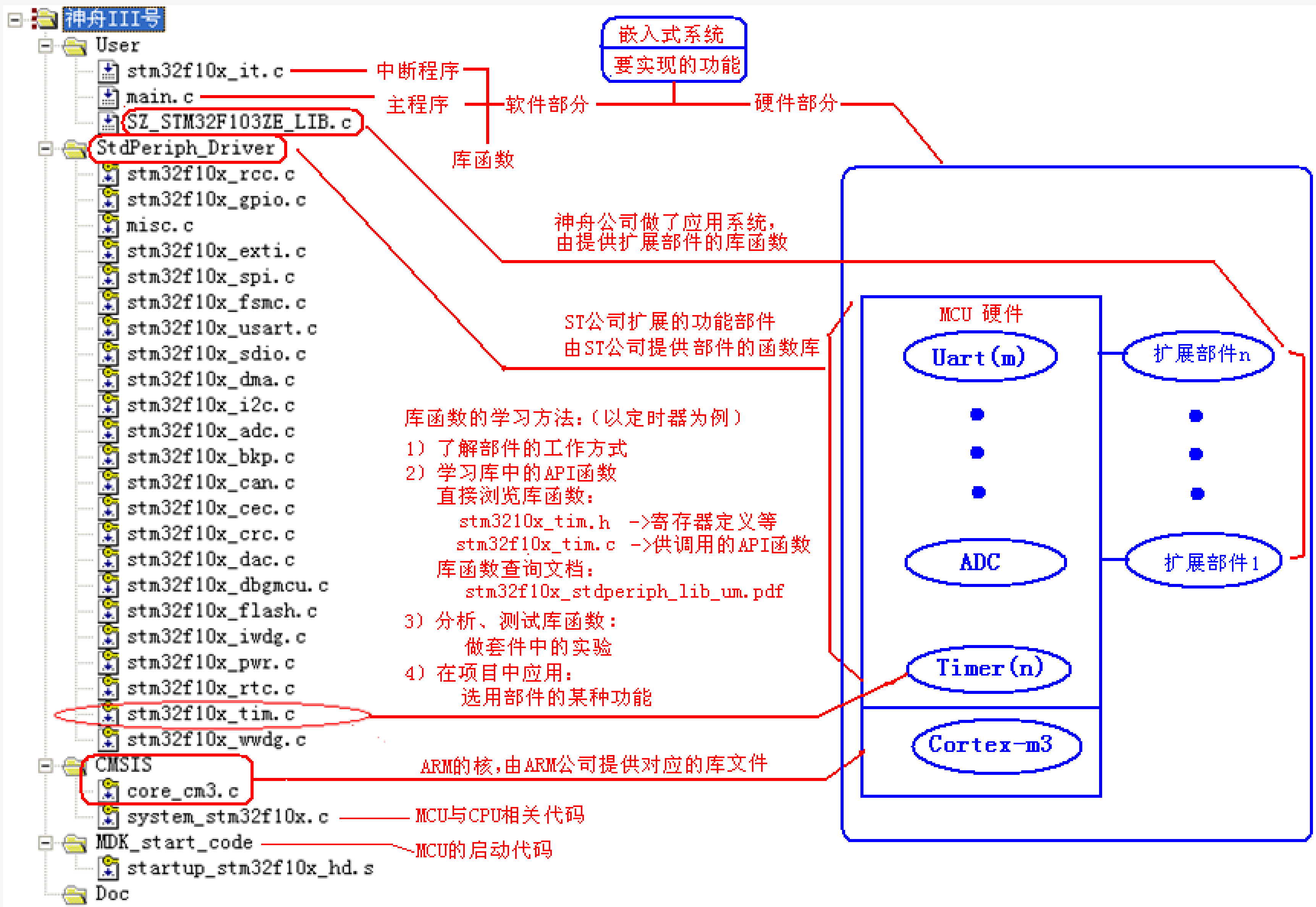
Figure 52. Advanced-control timer block diagram



三、通用定时器—T1定时器的寄存器

- 14 Advanced-control timers (TIM1&TIM8)
  - 14.1 TIM1&TIM8 introduction
  - 14.2 TIM1&TIM8 main features
  - 14.3 TIM1&TIM8 functional description
  - 14.4 TIM1&TIM8 registers
    - 14.4.1 TIM1&TIM8 control register 1 (TIMx_CR1)
    - 14.4.2 TIM1&TIM8 control register 2 (TIMx_CR2)
    - 14.4.3 TIM1&TIM8 slave mode control register (TIMx_SMCR)
    - 14.4.4 TIM1&TIM8 DMA/interrupt enable register (TIMx_DIER)
    - 14.4.5 TIM1&TIM8 status register (TIMx_SR)
    - 14.4.6 TIM1&TIM8 event generation register (TIMx_EGR)
    - 14.4.7 TIM1&TIM8 capture/compare mode register 1 (TIMx_CCMR1)
    - 14.4.8 TIM1&TIM8 capture/compare mode register 2 (TIMx_CCMR2)
    - 14.4.9 TIM1&TIM8 capture/compare enable register (TIMx_CCER)
    - 14.4.10 TIM1&TIM8 counter (TIMx_CNT)
    - 14.4.11 TIM1&TIM8 prescaler (TIMx_PSC)
    - 14.4.12 TIM1&TIM8 auto-reload register (TIMx_ARR)
    - 14.4.13 TIM1&TIM8 repetition counter register (TIMx_RCR)
    - 14.4.14 TIM1&TIM8 capture/compare register 1 (TIMx_CCR1)
    - 14.4.15 TIM1&TIM8 capture/compare register 2 (TIMx_CCR2)
    - 14.4.16 TIM1&TIM8 capture/compare register 3 (TIMx_CCR3)
    - 14.4.17 TIM1&TIM8 capture/compare register 4 (TIMx_CCR4)
    - 14.4.18 TIM1&TIM8 break and dead-time register (TIMx_BDTR)
    - 14.4.19 TIM1&TIM8 DMA control register (TIMx_DCR)
    - 14.4.20 TIM1&TIM8 DMA address for full transfer (TIMx_DMAR)
    - 14.4.21 TIM1&TIM8 register map

三、通用定时器——库函数文件



三、通用定时器——部分库函数

```
stm32f10x_tim.c
TIM_DeInit (TIM_TypeDef* TIMx)
TIM_TimeBaseInit (TIM_TypeDef* TIMx, TIM_TimeBaseInitTypeDef* TIM_TimeBaseInitStruct)
TIM_OC1Init (TIM_TypeDef* TIMx, TIM_OCInitTypeDef* TIM_OCInitStruct)
TIM_OC2Init (TIM_TypeDef* TIMx, TIM_OCInitTypeDef* TIM_OCInitStruct)
TIM_OC3Init (TIM_TypeDef* TIMx, TIM_OCInitTypeDef* TIM_OCInitStruct)
TIM_OC4Init (TIM_TypeDef* TIMx, TIM_OCInitTypeDef* TIM_OCInitStruct)
TIM_ICInit (TIM_TypeDef* TIMx, TIM_ICInitTypeDef* TIM_ICInitStruct)
TIM_PWMIConfig (TIM_TypeDef* TIMx, TIM_ICInitTypeDef* TIM_ICInitStruct)
TIM_BDTRConfig (TIM_TypeDef* TIMx, TIM_BDTRInitTypeDef *TIM_BDTRInitStruct)
TIM_TimeBaseStructInit (TIM_TimeBaseInitTypeDef* TIM_TimeBaseInitStruct)
TIM_OCStructInit (TIM_OCInitTypeDef* TIM_OCInitStruct)
TIM_ICStructInit (TIM_ICInitTypeDef* TIM_ICInitStruct)
TIM_BDTRStructInit (TIM_BDTRInitTypeDef* TIM_BDTRInitStruct)
TIM_Cmd (TIM_TypeDef* TIMx, FunctionalState NewState)
TIM_CtrlPWMOutputs (TIM_TypeDef* TIMx, FunctionalState NewState)
TIM_ITConfig (TIM_TypeDef* TIMx, uint16_t TIM_IT, FunctionalState NewState)
TIM_GenerateEvent (TIM_TypeDef* TIMx, uint16_t TIM_EventSource)
TIM_DMAConfig (TIM_TypeDef* TIMx, uint16_t TIM_DMABase, uint16_t TIM_DMABurstLength)
TIM_DMACmd (TIM_TypeDef* TIMx, uint16_t TIM_DMASource, FunctionalState NewState)
TIM_InternalClockConfig (TIM_TypeDef* TIMx)
TIM_ITRxExternalClockConfig (TIM_TypeDef* TIMx, uint16_t TIM_InputTriggerSource)
TIM_TIxExternalClockConfig (TIM_TypeDef* TIMx, uint16_t TIM_TIxExternalCLKSource, uint
TIM_ETRClockMode1Config (TIM_TypeDef* TIMx, uint16_t TIM_ExtTRGPrescaler, uint16_t TIM
TIM_ETRClockMode2Config (TIM_TypeDef* TIMx, uint16_t TIM_ExtTRGPrescaler, uint16_t TI
TIM_ETRConfig (TIM_TypeDef* TIMx, uint16_t TIM_ExtTRGPrescaler, uint16_t TIM_ExtTRGPol
TIM_PrescalerConfig (TIM_TypeDef* TIMx, uint16_t Prescaler, uint16_t TIM_PSCReloadMode
TIM_CounterModeConfig (TIM_TypeDef* TIMx, uint16_t TIM_CounterMode)
TIM_SelectInputTrigger (TIM_TypeDef* TIMx, uint16_t TIM_InputTriggerSource)
TIM_EncoderInterfaceConfig (TIM_TypeDef* TIMx, uint16_t TIM_EncoderMode, uint16_t TIM_
TIM_ForcedOC1Config (TIM_TypeDef* TIMx, uint16_t TIM_ForcedAction)
TIM_ForcedOC2Config (TIM_TypeDef* TIMx, uint16_t TIM_ForcedAction)
TIM_ForcedOC3Config (TIM_TypeDef* TIMx, uint16_t TIM_ForcedAction)
TIM_ForcedOC4Config (TIM_TypeDef* TIMx, uint16_t TIM_ForcedAction)
TIM_ARRPreloadConfig (TIM_TypeDef* TIMx, FunctionalState NewState)
```

四、库函数应用代码例

```c
/**--------------------------------------------------------------
  * @函数名 NVIC_TIM5Configuration
  * @功能   配置TIM5中断向量参数函数
  * @参数   无
  * @返回值 无
***-----------------------------------------------------------*/
static void NVIC_TIM5Configuration(void)  用户编写的函数
{
    NVIC_InitTypeDef NVIC_InitStructure;

    /* Set the Vector Table base address at 0x08000000 */
    //NVIC_SetVectorTable(NVIC_VectTab_FLASH, 0x0000);

    /* Enable the TIM5 gloabal Interrupt */  调用库函数前，设置结构参数
    NVIC_InitStructure.NVIC_IRQChannel = TIM5_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;

    NVIC_Init(&NVIC_InitStructure); 库函数
}
```

浙江大学 ZheJiang University

四、库函数应用代码例

```
void TIM5_Init(void) 用户函数
{
    TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure;

    /* TIM5 clock enable */
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM5, ENABLE); 使能TIM5的时钟
                                               TIM5 挂在外设总线1上
    /* ---------------------------------------                   ----
    TIM4 Configuration: Output Compare Timing Mode:
    TIM2CLK = 36 MHz, Prescaler = 7200, TIM2 counter clock = 7.2 MHz
    --------------------------------------------------------------- */

    /* Time base configuration */
    //这个就是自动装载的计数值，由于计数是从0开始的，计数10000次后为9999
    TIM_TimeBaseStructure.TIM_Period = (10000 - 1);
    // 这个就是预分频系数，当由于为0时表示不分频所以要减1
    TIM_TimeBaseStructure.TIM_Prescaler = (7200 - 1);
    // 高级应用本次不涉及。定义在定时器时钟(CK_INT)频率与数字滤波器(ETR,TIx)
    // 使用的采样频率之间的分频比例
    TIM_TimeBaseStructure.TIM_ClockDivision = 0;      库函数的输入参数设置
    //向上计数
    TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
    //初始化定时器5
    TIM_TimeBaseInit(TIM5, &TIM_TimeBaseStructure);

    /* Clear TIM5 update pending flag[清除TIM5溢出中断标志] */
    TIM_ClearITPendingBit(TIM5, TIM_IT_Update);

    /* TIM IT enable */ //打开溢出中断
    TIM_ITConfig(TIM5, TIM_IT_Update, ENABLE);              库函数

    /* TIM5 enable counter */
    TIM_Cmd(TIM5, ENABLE);  //计数器使能，开始工作

    /* 中断参数配置 */
    NVIC_TIM5Configuration(); 用户函数
}
```

四、库函数应用代码例

```
/**-----------------------------------------------------------
  * @函数名 TIM5_IRQHandler
  * @功能   TIM5中断处理函数，每秒中断一次
  * @参数   无
  * @返回值 无
***----------------------------------------------------------*/
void TIM5_IRQHandler(void)
{
    /* www.armjishu.com ARM技术论坛 */
    static u32 counter = 0;

    if (TIM_GetITStatus(TIM5, TIM_IT_Update) != RESET)
    {
        TIM_ClearITPendingBit(TIM5, TIM_IT_Update);
        /* LED1指示灯状态取反 */
        SZ_STM32_LEDToggle(LED1);

        /* armjishu.com提心您：不建议在中断中使用Printf，此示例只是演示。  */
        printf("\n\rarmjishu.com提示您：不建议在中断中使用Printf，此示例只是演示。\n\r");
        printf("ARMJISHU.COM-->TIM5:%d\n\r", counter++);
    }
}
```

§4 STM32F103的功能部件

§4.1 GPIO

§4.2 Timer

§4.3 UART

§4.4 A/D

§4.5 D/A

浙江大学
ZheJiang University

§4.3 USART

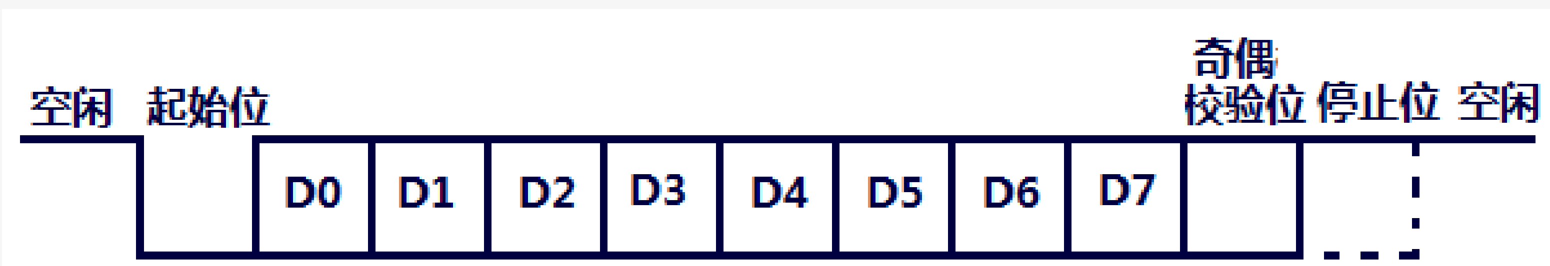
一、USART概述

- USART：Universal synchronous/asynchronous receiver transmitter，通用同步/异步收发器
- UART：通用异步收发器

1）异步串行的字节帧

UART通信是以字节帧为单位的，常用的字节帧：

1个起始位+8个数据位+1个校验位+1个停止位



§4.3 UART

2）通信参数

① 波特率：pbs，每秒多少位，即每一位的时间宽度；

② 数据区长度：8位（常用），或7位

③ 数据区顺序：低位在先（常用），或高位在先

④ 奇偶校验位：无、奇校验、偶校验

⑤ 停止位：1位、1.5位、2位

UART部件能按上述参数，自动完成一个字节帧的接收或发送；在每位宽度的中间连续采样3次，确定该位的数值；

异步通信体现在通信速度是事先约定的。

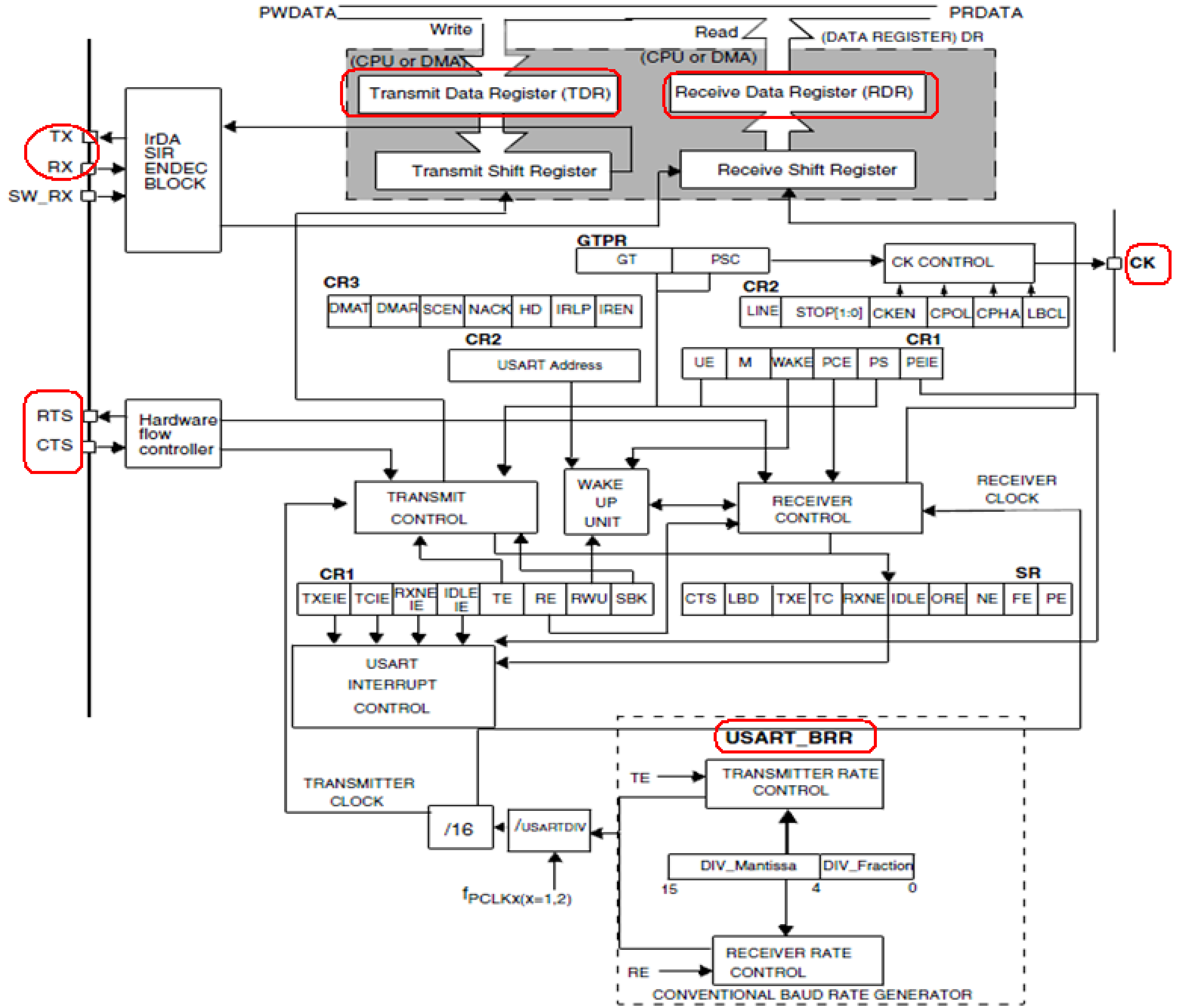
§4.3 UART

3）stm32f103C8的USART

3个USART，可使用DMA操作，有多种工作模式：

- UART：异步串行通信，用于RS232、RS485
- ISO7816：智能卡
- LIN： 串行通信，主/从功能，汽车和仪表中应用
- IrDA：红外数据通信

二、UART的组成


PWDATA
PRDATA
Write
Read
(DATA REGISTER) DR
(CPU or DMA)
(CPU or DMA)
Transmit Data Register (TDR)
Receive Data Register (RDR)
Transmit Shift Register
Receive Shift Register
TX
RX
SW_RX
IrDA SIR ENDEC BLOCK
GTPR
GT
PSC
CK CONTROL
CK
CR3
DMAT
DMAR
SCEN
NACK
HD
IRLP
IREN
CR2
LINE
STOP[1:0]
CKEN
CPOL
CPHA
LBCL
CR2
USART Address
CR1
UE
M
WAKE
PCE
PS
PEIE
RTS
CTS
Hardware flow controller
TRANSMIT CONTROL
WAKE UP UNIT
RECEIVER CONTROL
RECEIVER CLOCK
CR1
TXEIE
TCIE
RXNE IE
IDLE IE
TE
RE
RWU
SBK
SR
CTS
LBD
TXE
TC
RXNE
IDLE
ORE
NE
FE
PE
USART INTERRUPT CONTROL
USART_BRR
TE
TRANSMITTER RATE CONTROL
TRANSMITTER CLOCK
/16
/USARTDIV
DIV_Mantissa
DIV_Fraction
15
4
0
fPCLKx(x=1,2)
RECEIVER RATE CONTROL
RE
CONVENTIONAL BAUD RATE GENERATOR

§4.3 UART

三、UART的工作流程

① 接收过程：UART监听总线，有下跳变时，启动数据采样，一个字节的数据收到后，如果奇偶校验正确，则把数据存到接收寄存器中，置状接收态标志位，并向CPU申请中断，让CPU及时读取；

② 发送过程：UART的发送寄存器接收到CPU写入的数据，立刻启动，按设定的参数逐位发送，发送完毕，置状发送态标志位，并向CPU申请中断，告诉CPU可以发送下一个字节。

浙江大学 ZheJiang University

§4.3 UART

四、UART的寄存器

Table 198. USART register map and reset values

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	USART_SR	状态寄存器 Reserved																						CTS	LBD	TXE	TC	RXNE	IDLE	ORE	NE	FE	PE
	Reset value																							0	0	1	1	0	0	0	0	0	0
0x04	USART_DR	数据寄存器 接收寄存器：读出 发送寄存器：写入 Reserved																							DR[8:0]								
	Reset value																								0	0	0	0	0	0	0	0	0
0x08	USART_BRR	波特率寄存器 Reserved																DIV_Mantissa[15:4]												DIV_Fraction [3:0]			
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x0C	USART_CR1	控制寄存器1 Reserved																		UE	M	WAKE	PCE	PS	PEIE	TXEIE	TCIE	RXNEIE	IDLEIE	TE	RE	RWU	SBK
	Reset value																			0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x10	USART_CR2	控制寄存器2 Reserved																	LINEN	STOP[1:0]		CLKEN	CPOL	CPHA	LBCL	Reserved	LBDIE	LBDL	Reserved	ADD[3:0]			
	Reset value																		0	0	0	0	0	0	0		0	0		0	0	0	0
0x14	USART_CR3	控制寄存器3 Reserved																					CTSIE	CTSE	RTSE	DMAT	DMAR	SCEN	NACK	HDSEL	IRLP	IREN	EIE
	Reset value																						0	0	0	0	0	0	0	0	0	0	0
0x18	USART_GTPR	预分频寄存器 Reserved																GT[7:0]								PSC[7:0]							
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

五、库函数

```
stm32f10x_usart.c
    USART_DeInit (USART_TypeDef* USARTx)
    USART_Init (USART_TypeDef* USARTx, USART_InitTypeDef* USART_InitStruct)
    USART_StructInit (USART_InitTypeDef* USART_InitStruct)
    USART_ClockInit (USART_TypeDef* USARTx, USART_ClockInitTypeDef* USART_Clo
    USART_ClockStructInit (USART_ClockInitTypeDef* USART_ClockInitStruct)
    USART_Cmd (USART_TypeDef* USARTx, FunctionalState NewState)
    USART_ITConfig (USART_TypeDef* USARTx, uint16_t USART_IT, FunctionalState
    USART_DMACmd (USART_TypeDef* USARTx, uint16_t USART_DMAReq, FunctionalSta
    USART_SetAddress (USART_TypeDef* USARTx, uint8_t USART_Address)
    USART_WakeUpConfig (USART_TypeDef* USARTx, uint16_t USART_WakeUp)
    USART_ReceiverWakeUpCmd (USART_TypeDef* USARTx, FunctionalState NewState)
    USART_LINBreakDetectLengthConfig (USART_TypeDef* USARTx, uint16_t USART_L
    USART_LINCmd (USART_TypeDef* USARTx, FunctionalState NewState)
    USART_SendData (USART_TypeDef* USARTx, uint16_t Data)
    USART_ReceiveData (USART_TypeDef* USARTx)
    USART_SendBreak (USART_TypeDef* USARTx)
    USART_SetGuardTime (USART_TypeDef* USARTx, uint8_t USART_GuardTime)
    USART_SetPrescaler (USART_TypeDef* USARTx, uint8_t USART_Prescaler)
    USART_SmartCardCmd (USART_TypeDef* USARTx, FunctionalState NewState)
    USART_SmartCardNACKCmd (USART_TypeDef* USARTx, FunctionalState NewState)
    USART_HalfDuplexCmd (USART_TypeDef* USARTx, FunctionalState NewState)
    USART_OverSampling8Cmd (USART_TypeDef* USARTx, FunctionalState NewState)
    USART_OneBitMethodCmd (USART_TypeDef* USARTx, FunctionalState NewState)
    USART_IrDAConfig (USART_TypeDef* USARTx, uint16_t USART_IrDAMode)
    USART_IrDACmd (USART_TypeDef* USARTx, FunctionalState NewState)
    USART_GetFlagStatus (USART_TypeDef* USARTx, uint16_t USART_FLAG)
    USART_ClearFlag (USART_TypeDef* USARTx, uint16_t USART_FLAG)
    USART_GetITStatus (USART_TypeDef* USARTx, uint16_t USART_IT)
    USART_ClearITPendingBit (USART_TypeDef* USARTx, uint16_t USART_IT)
```

每种工作模式用若干个库函数来表达；

库函数说明：参见“STM32的函数说明（中文）.pdf”

六、程序例

功能：中断接收1个字节，马上发送该字节；

参见：STM32 实验18，串口通信；

```
void USART1_IRQHandler(void)
{
  u8 k;
  if(USART_GetITStatus(USART1,USART_IT_RXNE)!=RESET)//检查指定的USART中断发生与否
  {
    k=USART_ReceiveData(USART1);
    k++;
    USART_SendData(USART1,k);//通过外设USARTx发送单个数据
    //USART_ReceiveData(USART1)返回USARTx最近接收到的数据
    while(USART_GetFlagStatus(USART1,USART_FLAG_TXE)==RESET);
  }
}
```

§ 4.3 UART

六、程序例

```
void USARTINIT()        //通讯串口的配置
{
   USART_InitTypeDef  USART_InitStructure;

   USART_InitStructure.USART_BaudRate=9600;    //波特率设置为9600
   USART_InitStructure.USART_WordLength=USART_WordLength_8b;
   USART_InitStructure.USART_StopBits=USART_StopBits_1;
   USART_InitStructure.USART_Parity=USART_Parity_No;
   USART_InitStructure.USART_HardwareFlowControl=USART_HardwareFlowControl_None;
   USART_InitStructure.USART_Mode=USART_Mode_Rx|USART_Mode_Tx;

   USART_Init(USART1,&USART_InitStructure);

   USART_Cmd(USART1, ENABLE);
   USART_ITConfig(USART1, USART_IT_RXNE, ENABLE);//使能或者失能指定的USART中断 接收中断
   USART_ClearFlag(USART1,USART_FLAG_TC);//清除USARTx的待处理标志位
}
```

```
int main()
{
   RCCINIT();       //系统时钟的初始化
   GPIOINIT();      // 端口的初始化
   USARTINIT();     // 串口的配置及其初始化
   NVICINIT();     // 中断模式的初始化
   while(1);
}
```

§4 STMS32F103的功能部件

§4.1 GPIO

§4.2 Timer

§4.3 UART

§4.4 A/D

§4.5 D/A

§4.4 A/D转换器

一、A/D的概述

- STM32F103xx包含2个12位的逐次比较型ADC；
- 每个ADC有多达16个外部通道；
- ADC时钟是PCLK2经过预分频器得到；
- A/D转换时间1~1.5us；
- 带有自标定功能，可以减小电路漂移的影响；
- 附加的A/D值比较功能，可实现模拟量超限的自动报警；
- 有多种A/D转换的触发源；
- ADC工作电压：2.4 V to 3.6 V；
- 输入信号电压范围：VREF- ≤ VIN ≤ VREF+

浙江大学 ZheJiang University

§4.4 A/D转换器

二、工作方式

1）单次转换模式：在每个通道上，只执行一次转换；

2）连续转换模式：在每个通道上，执行连续转换；

3）扫描转换模式：在一组选定的模拟输入通道上自动转换；

4）启动A/D转换

软件命令、定时器(TIM1)产生的事件、外部触发和DMA触发；其中外部触发和DMA触发，允许应用程序同步AD转换和时钟的操作；

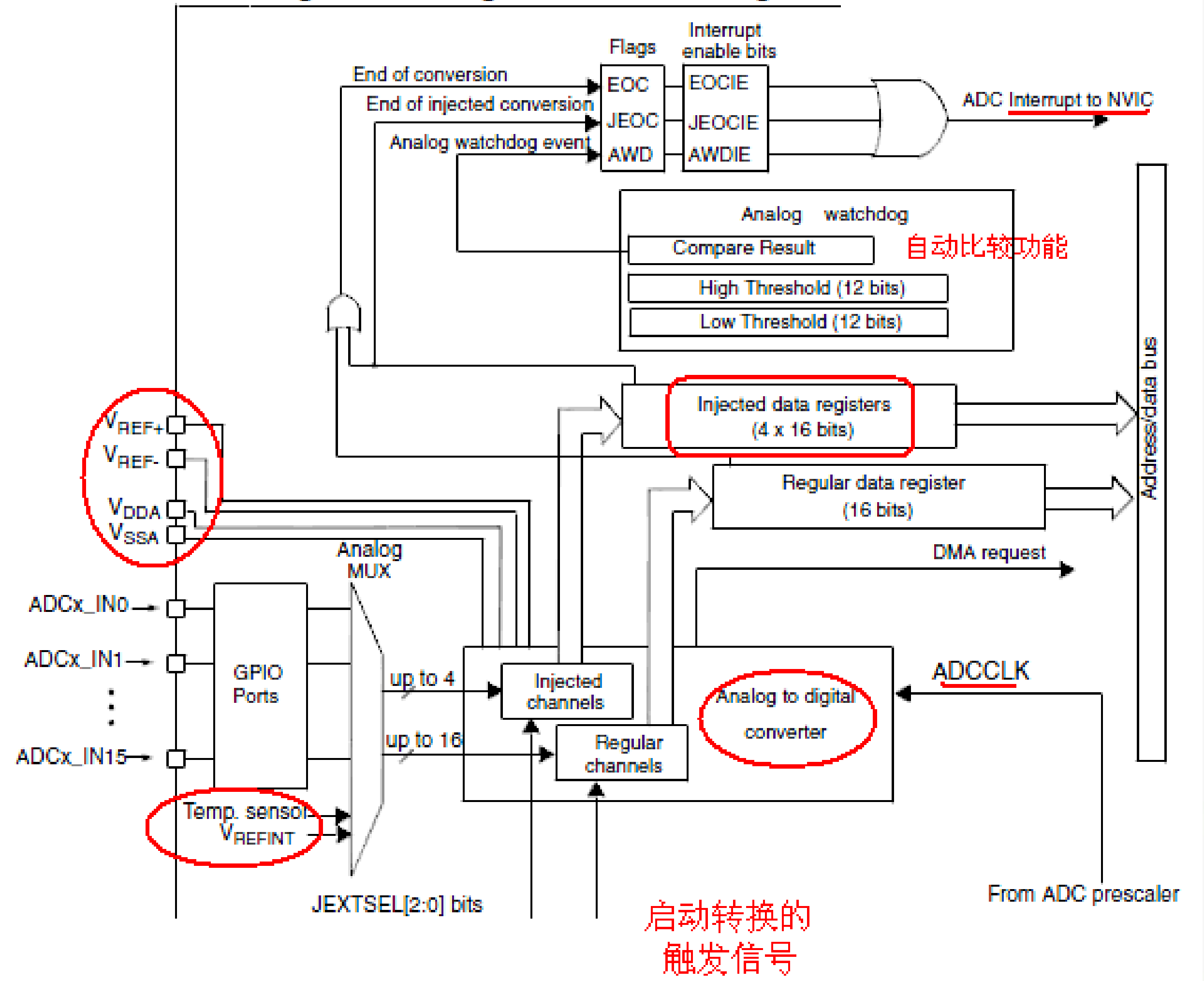
5）A/D转换结束后自动产生中断；

6）CPU通过查询状态位、中断响应、DMA方式获取A/D值。

§4.4 A/D转换器

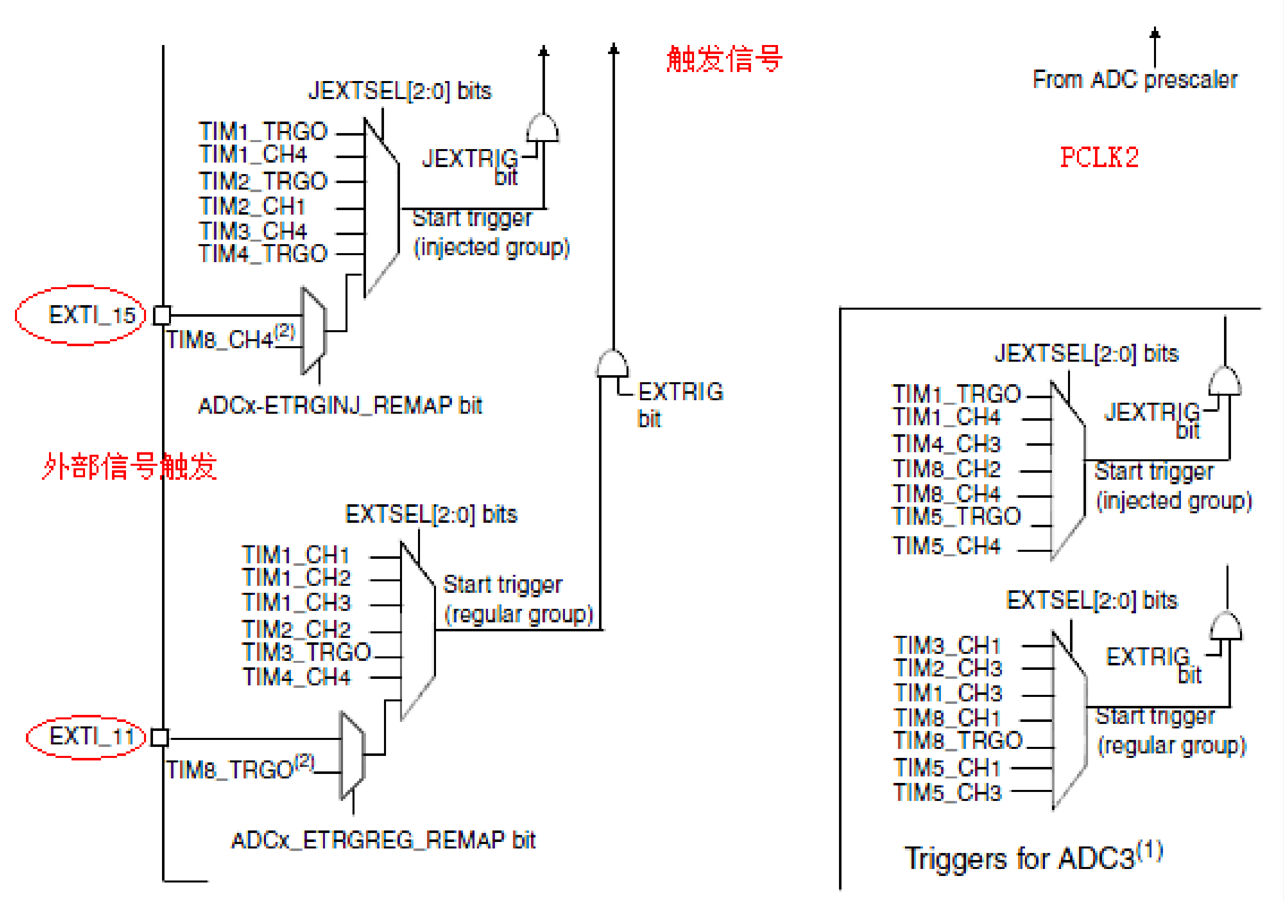
三、模块组成

Figure 22. Single ADC block diagram



§4.4 A/D转换器

三、模块组成



§ 4.4 A/D转换器

四、寄存器

- 11.12 ADC registers
  - 11.12.1 ADC status register (ADC_SR)
  - 11.12.2 ADC control register 1 (ADC_CR1)
  - 11.12.3 ADC control register 2 (ADC_CR2)
  - 11.12.4 ADC sample time register 1 (ADC_SMPR1)
  - 11.12.5 ADC sample time register 2 (ADC_SMPR2)
  - 11.12.6 ADC injected channel data offset register x (ADC_JOFRx) (x=1..4)
  - 11.12.7 ADC watchdog high threshold register (ADC_HTR)
  - 11.12.8 ADC watchdog low threshold register (ADC_LTR)
  - 11.12.9 ADC regular sequence register 1 (ADC_SQR1)
  - 11.12.10 ADC regular sequence register 2 (ADC_SQR2)
  - 11.12.11 ADC regular sequence register 3 (ADC_SQR3)
  - 11.12.12 ADC injected sequence register (ADC_JSQR)
  - 11.12.13 ADC injected data register x (ADC_JDRx) (x= 1..4)
  - 11.12.14 ADC regular data register (ADC_DR)

五、ADC库函数

```
stm32f10x_adc.c
    ADC_DeInit (ADC_TypeDef* ADCx)
    ADC_Init (ADC_TypeDef* ADCx, ADC_InitTypeDef* ADC_InitStruct)
    ADC_StructInit (ADC_InitTypeDef* ADC_InitStruct)
    ADC_Cmd (ADC_TypeDef* ADCx, FunctionalState NewState)
    ADC_DMACmd (ADC_TypeDef* ADCx, FunctionalState NewState)
    ADC_ITConfig (ADC_TypeDef* ADCx, uint16_t ADC_IT, FunctionalState NewState)
    ADC_ResetCalibration (ADC_TypeDef* ADCx)
    ADC_GetResetCalibrationStatus (ADC_TypeDef* ADCx)
    ADC_StartCalibration (ADC_TypeDef* ADCx)
    ADC_GetCalibrationStatus (ADC_TypeDef* ADCx)
    ADC_SoftwareStartConvCmd (ADC_TypeDef* ADCx, FunctionalState NewState)
    ADC_GetSoftwareStartConvStatus (ADC_TypeDef* ADCx)
    ADC_DiscModeChannelCountConfig (ADC_TypeDef* ADCx, uint8_t Number)
    ADC_DiscModeCmd (ADC_TypeDef* ADCx, FunctionalState NewState)
    ADC_RegularChannelConfig (ADC_TypeDef* ADCx, uint8_t ADC_Channel, uint8_t Rank, uint8_t ADC_SampleTime)
    ADC_ExternalTrigConvCmd (ADC_TypeDef* ADCx, FunctionalState NewState)
    ADC_GetConversionValue (ADC_TypeDef* ADCx)
    ADC_GetDualModeConversionValue (void)
    ADC_AutoInjectedConvCmd (ADC_TypeDef* ADCx, FunctionalState NewState)
    ADC_InjectedDiscModeCmd (ADC_TypeDef* ADCx, FunctionalState NewState)
    ADC_ExternalTrigInjectedConvConfig (ADC_TypeDef* ADCx, uint32_t ADC_ExternalTrigInjecConv)
    ADC_ExternalTrigInjectedConvCmd (ADC_TypeDef* ADCx, FunctionalState NewState)
    ADC_SoftwareStartInjectedConvCmd (ADC_TypeDef* ADCx, FunctionalState NewState)
    ADC_GetSoftwareStartInjectedConvCmdStatus (ADC_TypeDef* ADCx)
    ADC_InjectedChannelConfig (ADC_TypeDef* ADCx, uint8_t ADC_Channel, uint8_t Rank, uint8_t ADC_SampleTime)
    ADC_InjectedSequencerLengthConfig (ADC_TypeDef* ADCx, uint8_t Length)
    ADC_SetInjectedOffset (ADC_TypeDef* ADCx, uint8_t ADC_InjectedChannel, uint16_t Offset)
    ADC_GetInjectedConversionValue (ADC_TypeDef* ADCx, uint8_t ADC_InjectedChannel)
    ADC_AnalogWatchdogCmd (ADC_TypeDef* ADCx, uint32_t ADC_AnalogWatchdog)
    ADC_AnalogWatchdogThresholdsConfig (ADC_TypeDef* ADCx, uint16_t HighThreshold, uint16_t LowThreshold)
    ADC_AnalogWatchdogSingleChannelConfig (ADC_TypeDef* ADCx, uint8_t ADC_Channel)
    ADC_TempSensorVrefintCmd (FunctionalState NewState)
    ADC_GetFlagStatus (ADC_TypeDef* ADCx, uint8_t ADC_FLAG)
    ADC_ClearFlag (ADC_TypeDef* ADCx, uint8_t ADC_FLAG)
    ADC_GetITStatus (ADC_TypeDef* ADCx, uint16_t ADC_IT)
    ADC_ClearITPendingBit (ADC_TypeDef* ADCx, uint16_t ADC_IT)
```

六、ADC代码例

```
void RCCINIT_ADC()
{
  SystemInit();
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB,ENABLE);
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO,ENABLE);
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1,ENABLE);
  RCC_ADCCLKConfig(RCC_PCLK2_Div6);//12M  最大14M 设置ADC时钟（ADCCLK）
}

void ADCINIT_ADC()
{
  ADC_InitTypeDef ADC_InitStructure;
  ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;
  ADC_InitStructure.ADC_ScanConvMode = DISABLE;
  ADC_InitStructure.ADC_ContinuousConvMode = DISABLE;
  ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;
  ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
  ADC_InitStructure.ADC_NbrOfChannel = 1;
  ADC_Init(ADC1, &ADC_InitStructure);

  //设置指定ADC的规则组通道, 设置它们的转化顺序和采样时间
  ADC_RegularChannelConfig(ADC1,ADC_Channel_8,1,ADC_SampleTime_239Cycles5);

  ADC_Cmd(ADC1,ENABLE);
  ADC_ResetCalibration(ADC1);//重置指定的ADC的校准寄存器
  while(ADC_GetResetCalibrationStatus(ADC1));//获取ADC重置校准寄存器的状态

  ADC_StartCalibration(ADC1);//开始指定ADC的校准状态
  while(ADC_GetCalibrationStatus(ADC1));//获取指定ADC的校准程序
  ADC_SoftwareStartConvCmd(ADC1, ENABLE);//使能或者失能指定的ADC的软件转换启动功能
}
```

§4.4 A/D转换器

六、代码例

```
int main()
{
  u32 ad=0;
  u8 i;
  RCCINIT_PRINTF();     //初始化printf的系统时钟
  RCCINIT_ADC();        //初始化ADC的系统时钟
  GPIOINIT_ADC();       //初始化ADC的端口配置
  GPIOINIT_PRINTF();
  USARTINIT_PRINTF();   //printf串口的初始化配置
  NVICINIT_PRINTF();     //printf中断模式的初始化配置
  ADCINIT_ADC();
  while(1)
  {
    ad=0;
    for(i=0;i<50;i++)//读取50次的AD数值取其平均数较为准确
    {
      ADC_SoftwareStartConvCmd(ADC1, ENABLE);
      while(!ADC_GetFlagStatus(ADC1,ADC_FLAG_EOC));//转换结束标志位
      ad=ad+ADC_GetConversionValue(ADC1);//返回最近一次ADCx规则组的转换结果
    }
    ad=ad/50;
    printf("ad=%f\n",ad*3.3/4096);
    delay_ms(1000);
  }
}
```

§4 STM32F103的功能部件

§4.1 GPIO

§4.2 Timer

§4.3 UART

§4.4 A/D

§4.5 D/A

§4.5 D/A转换器

一、D/A的特点

- STM32F103C8 中无DAC；
- 2个独立的12位D/A，每个D/A有一个电压信号输出端；
- 能产生三角波、随机噪声波；
- 每个DAC具有DMA传送能力；
- 带外部触发信号，启动D/A转换；
- 外接参考电压VREF+（与ADC共用）输入，可提高DAC分辨率；
- 输出电压计算：Vout = DA /4096* (VREF+ — VREF-)

§4.5 D/A转换器

二、工作模式

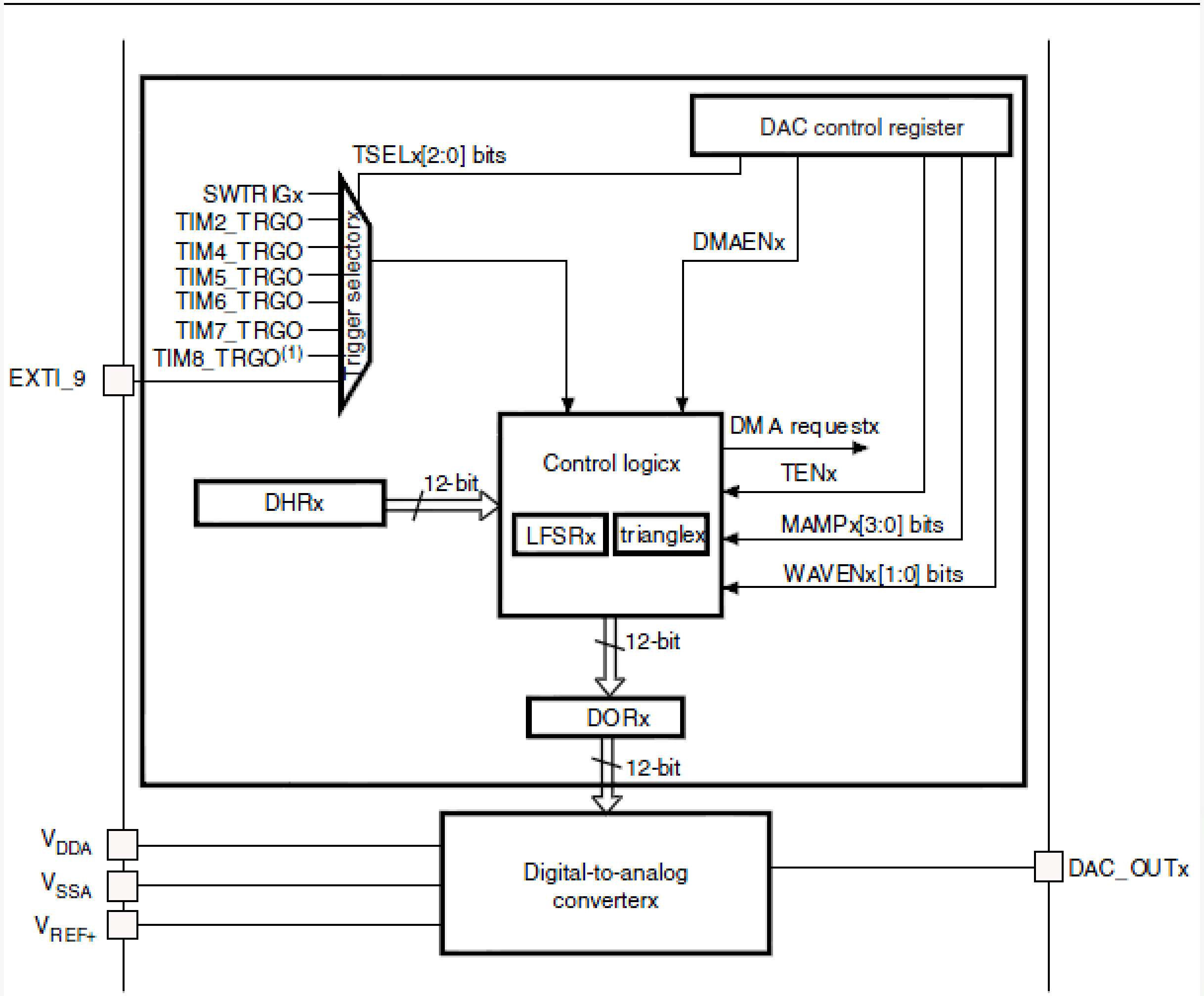
- 单通道模式和双通道模式；
- 双通道模式中，2个DAC可以独立运行，或同时转换；
- 每个DAC可选择8位或12位转换模式；
- 在12位转换模式中，数据可选择左对齐，或右对齐。

浙江大学 ZheJiang University

§4.5 D/A转换器

三、模块组成

Figure 40. DAC channel block diagram



四、DAC寄存器

Table 75. DAC register map

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	DAC_CR	Res.			DMAEN2	MAMP2[3:0]				WAVE2[2:0]		TSEL2[2:0]			TEN2	BOFF2	EN2	Res.			DMAEN1	MAMP1[3:0]				WAVE1[2:0]		TSEL1[2:0]			TEN1	BOFF1	EN1
	Reset value				0	0	0	0	0	0	0	0	0	0	0	0	0				0	0	0	0	0	0	0	0	0	0	0	0	0
0x04	DAC_SWTRIGR	Reserved																														SWTRIG2	SWTRIG1
	Reset value																															0	0
0x08	DAC_DHR12R1	Reserved																				DACC1DHR[11:0]											
	Reset value																					0	0	0	0	0	0	0	0	0	0	0	0
0x0C	DAC_DHR12L1	Reserved																DACC1DHR[11:0]												Reserved			
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0				
0x10	DAC_DHR8R1	Reserved																								DACC1DHR[7:0]							
	Reset value																									0	0	0	0	0	0	0	0
0x14	DAC_DHR12R2	Reserved																				DACC2DHR[11:0]											
	Reset value																					0	0	0	0	0	0	0	0	0	0	0	0
0x18	DAC_DHR12L2	Reserved																DACC2DHR[11:0]												Reserved			
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0				
0x1C	DAC_DHR8R2	Reserved																								DACC2DHR[7:0]							
																										0	0	0	0	0	0	0	0
0x20	DAC_DHR12RD	Reserved				DACC2DHR[11:0]												Reserved				DACC1DHR[11:0]											
	Reset value					0	0	0	0	0	0	0	0	0	0	0	0					0	0	0	0	0	0	0	0	0	0	0	0
0x24	DAC_DHR12LD	DACC2DHR[11:0]												Reserved				DACC1DHR[11:0]												Reserved			
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0					0	0	0	0	0	0	0	0	0	0	0	0				
0x28	DAC_DHR8RD	Reserved																DACC2DHR[7:0]								DACC1DHR[7:0]							
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x2C	DAC_DOR1	Reserved																				DACC1DOR[11:0]											
	Reset value																					0	0	0	0	0	0	0	0	0	0	0	0
0x30	DAC_DOR2	Reserved																				DACC2DOR[11:0]											
	Reset value																					0	0	0	0	0	0	0	0	0	0	0	0

§4.5 D/A转换器

五、库函数

```
stm32f10x_dac.c
  DAC_DeInit (void)
  DAC_Init (uint32_t DAC_Channel, DAC_InitTypeDef* DAC_InitStruct)
  DAC_StructInit (DAC_InitTypeDef* DAC_InitStruct)
  DAC_Cmd (uint32_t DAC_Channel, FunctionalState NewState)
  DAC_ITConfig (uint32_t DAC_Channel, uint32_t DAC_IT, FunctionalState NewState)
  DAC_DMACmd (uint32_t DAC_Channel, FunctionalState NewState)
  DAC_SoftwareTriggerCmd (uint32_t DAC_Channel, FunctionalState NewState)
  DAC_DualSoftwareTriggerCmd (FunctionalState NewState)
  DAC_WaveGenerationCmd (uint32_t DAC_Channel, uint32_t DAC_Wave,FunctionalState St)
  DAC_SetChannel1Data (uint32_t DAC_Align, uint16_t Data)
  DAC_SetChannel2Data (uint32_t DAC_Align, uint16_t Data)
  DAC_SetDualChannelData (uint32_t DAC_Align, uint16_t Data2, uint16_t Data1)
  DAC_GetDataOutputValue (uint32_t DAC_Channel)
  DAC_GetFlagStatus (uint32_t DAC_Channel, uint32_t DAC_FLAG)
  DAC_ClearFlag (uint32_t DAC_Channel, uint32_t DAC_FLAG)
  DAC_GetITStatus (uint32_t DAC_Channel, uint32_t DAC_IT)
  DAC_ClearITPendingBit (uint32_t DAC_Channel, uint32_t DAC_IT)
```

浙江大学
ZheJiang University

§4.5 D/A转换器

六、程序例

PA4 用作DA_OUT

此例中MCU是STM32F103ZE

```
void RCCINIT(void)
{
    SystemInit();//72m
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA,ENABLE);
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1,ENABLE);
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO,ENABLE);

  RCC_APB1PeriphClockCmd(RCC_APB1Periph_DAC,ENABLE);
}
```

```
void DACINIT(void)          //DAC初始化配置
{
   DAC_InitTypeDef DAC_InitStructure;

   DAC_InitStructure.DAC_Trigger=DAC_Trigger_None;//不使用出发功能
   DAC_InitStructure.DAC_WaveGeneration=DAC_WaveGeneration_None;//不使用三角波
   //屏蔽 幅值设置
   DAC_InitStructure.DAC_LFSRUnmask_TriangleAmplitude=DAC_LFSRUnmask_Bit0;
   //关闭缓存
   DAC_InitStructure.DAC_OutputBuffer=DAC_OutputBuffer_Disable;

   DAC_Init(DAC_Channel_1,&DAC_InitStructure);//初始化DAC通道1

   DAC_Cmd(DAC_Channel_1,ENABLE);//使能DAC1

   DAC_SetChannel1Data(DAC_Align_12b_R,0);//12位 右对齐 写0数据
}
```

§4.5 D/A转换器

六、程序例

```
int main()
{
  u8 i;
  float da;
  RCCINIT();
  GPIOINIT();
  NVICINIT();
  USARTINIT();
  DACINIT();
  while(1)
  {
    da=0;
    for(i=0;i<=10;i++)
    {
      da=i*400;
      //12位 右对齐 PA4 端口输出
      DAC_SetChannel1Data(DAC_Align_12b_R,da);
      printf("da=%fv\n",3.3*da/4096);
      delayms(1000);
      delayms(1000);
      delayms(1000);
      delayms(1000);
      delayms(1000);//间隔5秒输出一个电压
    }
  }
}
```

第十一讲 小结

§1 MCU组成模型和学习方法

§2 STM32F系列及命名方法

§3 STM32F103C8概述

§4 STM32F103的功能部件

MCU的学习内容

1）CPU：功能、组成、寻址方式和指令系统；

2）功能部件：功能、组成、工作方式、寄存器、库函数、典型应用场合和代码等；

3）其它常用功能部件将在后续章节、实验课上介绍，目标是掌握MCU功能部件的学习方法，从资料、到实验代码、应用测试，然后能举一反三。

浙江大学 ZheJiang University

实验作业

- 做一个控制系统，输入是温度/光敏传感器，输出是步进电机。
- 设定一个初始值（以当前温度/亮度为参考），如果传感器温度/亮度超过初始值，则电机正传；如果低于设定值，则电机反转
- 当前值和初始值差异越大，则电机转速越快。

致谢

谢谢！