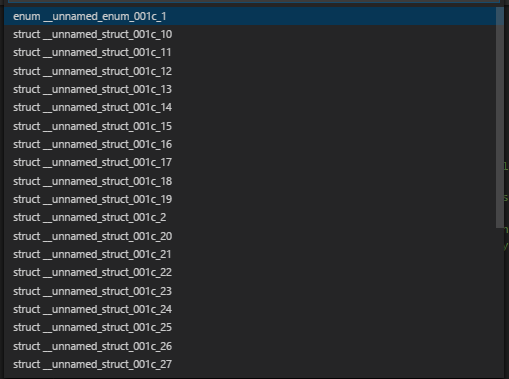
由于实际学习与研究开发需要，今天打算仔细分析一下stm32f103xe.h这个文件，方便熟悉stm的开发框架。首先从navigate中看到的全是这样的结构体，看不出个所以然来，还是开始硬啃代码吧~



/\*\* @addtogroup Configuration\_section\_for\_CMSIS

\* @{

\*/

/\*\*

\* @brief Configuration of the Cortex-M3 Processor and Core Peripherals

\*/

#define \_\_MPU\_PRESENT 0 /\*!< Other STM32 devices does not provide an MPU \*/

#define \_\_CM3\_REV 0x0200 /\*!< Core Revision r2p0 \*/

#define \_\_NVIC\_PRIO\_BITS 4 /\*!< STM32 uses 4 Bits for the Priority Levels \*/

#define \_\_Vendor\_SysTickConfig 0 /\*!< Set to 1 if different SysTick Config is used \*/

这里定义了一些一些硬件相关特性，第一个 \_\_MPU\_PRESENT 定义了是否有MPU【Memory Protection Unit：硬件保护单元，注意这里不是Microprocessor Unit的意思】，显然F1不带有MPU；

\_\_CM3\_REV 是Cortex-M3的核心版本，

后面两个是现在还不能理解的标志位，待补充。

typedef enum

{

/\*\*\*\*\*\* Cortex-M3 Processor Exceptions Numbers \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

NonMaskableInt\_IRQn = -14, /\*!< 2 Non Maskable Interrupt \*/

HardFault\_IRQn = -13, /\*!< 3 Cortex-M3 Hard Fault Interrupt \*/

。。。 。。。 。。。 。。。 。。。

DebugMonitor\_IRQn = -4, /\*!< 12 Cortex-M3 Debug Monitor Interrupt \*/

PendSV\_IRQn = -2, /\*!< 14 Cortex-M3 Pend SV Interrupt \*/

SysTick\_IRQn = -1, /\*!< 15 Cortex-M3 System Tick Interrupt \*/

/\*\*\*\*\*\* STM32 specific Interrupt Numbers \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

WWDG\_IRQn = 0, /\*!< Window WatchDog Interrupt \*/

PVD\_IRQn = 1, /\*!< PVD through EXTI Line detection Interrupt \*/

TAMPER\_IRQn = 2, /\*!< Tamper Interrupt \*/

RTC\_IRQn = 3, /\*!< RTC global Interrupt \*/

FLASH\_IRQn = 4, /\*!< FLASH global Interrupt \*/

RCC\_IRQn = 5, /\*!< RCC global Interrupt \*/

EXTI0\_IRQn = 6, /\*!< EXTI Line0 Interrupt \*/

EXTI1\_IRQn = 7, /\*!< EXTI Line1 Interrupt \*/

。。。 。。。 。。。 。。。 。。。

DMA2\_Channel4\_5\_IRQn = 59, /\*!< DMA2 Channel 4 and Channel 5 global Interrupt \*/

} IRQn\_Type;

中断表，不多解释。

#include "core\_cm3.h"

#include "system\_stm32f1xx.h"

#include <stdint.h>

引用了几个文件，稍后分析，有兴趣的可以看后续更新哦 （^ \_ ^）

/\*\*

\* @brief Analog to Digital Converter

\*/

typedef struct

{

\_\_IO uint32\_t SR;

\_\_IO uint32\_t CR1;

\_\_IO uint32\_t CR2;

… … … … …

\_\_IO uint32\_t JDR3;

\_\_IO uint32\_t JDR4;

\_\_IO uint32\_t DR;

} ADC\_TypeDef;

定义了ADC的标准封装类，还有很多类似的结构，总结一下就是：

|  |  |
| --- | --- |
| 名称 | 寄存器位数 |
| ADC\_TypeDef | 32位 |
| ADC\_Common\_TypeDef | 32位 |
| BKP\_TypeDef | 32位 |
| CAN\_TxMailBox\_TypeDef | 32位 |
| CAN\_FIFOMailBox\_TypeDef | 32位 |
| CAN\_FilterRegister\_TypeDef | 32位 |
| CAN\_TypeDef | 32位 |
| CRC\_TypeDef | 8、16、32混合 |
| DAC\_TypeDef | 32位 |
| DBGMCU\_TypeDef | 32位 |
| DMA\_Channel\_TypeDef | 32位 |
| DMA\_TypeDef | 32位 |
| EXTI\_TypeDef | 32位 |
| FLASH\_TypeDef | 32位 |
| OB\_TypeDef | 16位 |
| FSMC\_Bank1\_TypeDef | 32位 |
| FSMC\_Bank1E\_TypeDef | 32位 |
| FSMC\_Bank2\_3\_TypeDef | 32位 |
| FSMC\_Bank4\_TypeDef | 32位 |
| GPIO\_TypeDef | 32位 |
| AFIO\_TypeDef | 32位 |
| I2C\_TypeDef | 32位 |
| IWDG\_TypeDef | 32位 |
| PWR\_TypeDef | 32位 |
| RCC\_TypeDef | 32位 |
| RTC\_TypeDef | 32位 |
| SDIO\_TypeDef | 32位 |
| SPI\_TypeDef | 32位 |
| TIM\_TypeDef | 32位 |
| USART\_TypeDef | 32位 |
| USB\_TypeDef | 16位 |
| WWDG\_TypeDef | 32位 |

可以看出，STM32所有的资源基本都是通过这种统一的方式来管理的，这一点必须理解清楚。

然后，寄存器中多数都是32位 的，这点其实也不难理解，如果stm32还是像c51那样广泛使用8位寄存器，那我们不是学了个假的ARM？（~\_~）

#define FLASH\_BASE ((uint32\_t)0x08000000) /\*!< FLASH base address in the alias region \*/

#define FLASH\_BANK1\_END ((uint32\_t)0x0807FFFF) /\*!< FLASH END address of bank1 \*/

#define SRAM\_BASE ((uint32\_t)0x20000000) /\*!< SRAM base address in the alias region \*/

#define PERIPH\_BASE ((uint32\_t)0x40000000) /\*!< Peripheral base address in the alias region \*/

#define SRAM\_BB\_BASE ((uint32\_t)0x22000000) /\*!< SRAM base address in the bit-band region \*/

#define PERIPH\_BB\_BASE ((uint32\_t)0x42000000) /\*!< Peripheral base address in the bit-band region \*/

#define FSMC\_BASE ((uint32\_t)0x60000000) /\*!< FSMC base address \*/

#define FSMC\_R\_BASE ((uint32\_t)0xA0000000) /\*!< FSMC registers base address \*/

。。。 。。。 。。。 。。。 。。。 。。。 。。。 。。。

#define DBGMCU\_BASE ((uint32\_t)0xE0042000) /\*!< Debug MCU registers base address \*/

/\* USB device FS \*/

#define USB\_BASE (APB1PERIPH\_BASE + 0x00005C00) /\*!< USB\_IP Peripheral Registers base address \*/

#define USB\_PMAADDR (APB1PERIPH\_BASE + 0x00006000) /\*!< USB\_IP Packet Memory Area base address \*/

后面定义了这些设备的基地址，我们知道在Linux中万物皆文件，其实这不是没有依据的，因为所有的这些外设设备都是通过寄存器、内存来进行管理的，所以这里对同一类的设备归到一个基地址附近是十分合理的【方便统一管理嘛，这个设计在很多系统中都是类似的】。

/\*\* @addtogroup Peripheral\_declaration

\* @{

\*/

#define TIM2 ((TIM\_TypeDef \*) TIM2\_BASE)

。。。 。。。 。。。 。。。 。。。 。。。 。。。 。。。 。。。

#define RTC ((RTC\_TypeDef \*) RTC\_BASE)

#define WWDG ((WWDG\_TypeDef \*) WWDG\_BASE)

#define IWDG ((IWDG\_TypeDef \*) IWDG\_BASE)

#define BKP ((BKP\_TypeDef \*) BKP\_BASE)

#define GPIOA ((GPIO\_TypeDef \*) GPIOA\_BASE)

。。。 。。。 。。。 。。。 。。。 。。。 。。。 。。

#define GPIOG ((GPIO\_TypeDef \*) GPIOG\_BASE)

#define ADC1 ((ADC\_TypeDef \*) ADC1\_BASE)

#define DMA2 ((DMA\_TypeDef \*) DMA2\_BASE)

#define DMA1\_Channel1 ((DMA\_Channel\_TypeDef \*) DMA1\_Channel1\_BASE)

。。。 。。。 。。。 。。。 。。。 。。。 。。。 。。。 。。。

#define DMA2\_Channel5 ((DMA\_Channel\_TypeDef \*) DMA2\_Channel5\_BASE)

#define RCC ((RCC\_TypeDef \*) RCC\_BASE)

#define CRC ((CRC\_TypeDef \*) CRC\_BASE)

#define FLASH ((FLASH\_TypeDef \*) FLASH\_R\_BASE)

#define OB ((OB\_TypeDef \*) OB\_BASE)

#define FSMC\_Bank1 ((FSMC\_Bank1\_TypeDef \*) FSMC\_BANK1\_R\_BASE)

#define FSMC\_Bank1E ((FSMC\_Bank1E\_TypeDef \*) FSMC\_BANK1E\_R\_BASE)

#define FSMC\_Bank2\_3 ((FSMC\_Bank2\_3\_TypeDef \*) FSMC\_BANK2\_3\_R\_BASE)

#define FSMC\_Bank4 ((FSMC\_Bank4\_TypeDef \*) FSMC\_BANK4\_R\_BASE)

#define DBGMCU ((DBGMCU\_TypeDef \*) DBGMCU\_BASE)

这里就可以看到我们常用的硬件管理接口了，GPIOA，RCC什么的是不是灰常熟悉呀？没错，他们被定义在这里的，用的是刚刚定义的标准结构体实例化的，很明显，这是精心设计的一个系统，ARM或者说ST预先将各个寄存器按照结构体所表示的结构进行配置，然后连接到前面给定的基地址处，这里再将该地址通过类型强制 变为相应结构体，并实现 寄存器变化，硬件即进行相应响应，不得不佩服这种设计者的构思呀~

/\*\* @addtogroup Peripheral\_Registers\_Bits\_Definition

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\* Peripheral Registers\_Bits\_Definition

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\* CRC calculation unit (CRC)

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Bit definition for CRC\_DR register \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define CRC\_DR\_DR\_Pos (0U)

#define CRC\_DR\_DR\_Msk (0xFFFFFFFFU << CRC\_DR\_DR\_Pos) /\*!< 0xFFFFFFFF \*/

#define CRC\_DR\_DR CRC\_DR\_DR\_Msk /\*!< Data register bits \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Bit definition for CRC\_IDR register \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define CRC\_IDR\_IDR\_Pos (0U)

#define CRC\_IDR\_IDR\_Msk (0xFFU << CRC\_IDR\_IDR\_Pos) /\*!< 0x000000FF \*/

#define CRC\_IDR\_IDR CRC\_IDR\_IDR\_Msk /\*!< General-purpose 8-bit data register bits \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Bit definition for CRC\_CR register \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define CRC\_CR\_RESET\_Pos (0U)

#define CRC\_CR\_RESET\_Msk (0x1U << CRC\_CR\_RESET\_Pos) /\*!< 0x00000001 \*/

#define CRC\_CR\_RESET CRC\_CR\_RESET\_Msk /\*!< RESET bit \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\* Power Control

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Bit definition for PWR\_CR register \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define PWR\_CR\_LPDS\_Pos (0U)

#define PWR\_CR\_LPDS\_Msk (0x1U << PWR\_CR\_LPDS\_Pos) /\*!< 0x00000001 \*/

#define PWR\_CR\_LPDS PWR\_CR\_LPDS\_Msk /\*!< Low-Power Deepsleep \*/

。。。 。。。 。。。 。。。 。。。。 。。。。 。。。 。。。

后面以这种结构定义了大批的Peripheral Registers\_Bits\_Definition，也就是外围设备位，

在这里做一个统计：

CRC

Powe Control

Backup Registers

Reset and Clock Control

GPIO Registers

SystemTick

NVIC : Nested Vectored Interrupt Controller　(终于见到这个像是Nvidia缩写的单词全程了，内置中断向量控制器)

External Interrupt/Event Controlller 外部[中断/事件]控制器

DMA Controller DMA设备辅助管理器控制器

ADC：Analog to Digital Converter ADC转换器

DAC

Real-Time Clock 实时时钟

IWDG：Independent WATCHDOG 独立看门狗

WWDG：Window WATCHDOG 窗口看门狗？

FSMC：Flexible Static Memory Controller 【灵活的】静态内存控制器，也可理解为轻量级的

SDIO：SD Host Interface SD卡接口 原来这个SDIO就是指的sd卡呀？

USB Device FS 这个猜测的话就是USB文件系统设备咯

CAN：Controller Area Network ：局部控制网络，名字非常熟悉，实际一窍不通emmmm

SPI：Serial Peripheral Interface 同样是名字熟悉的SPI

I2C：Inter-Integrated Circuit Interface 同样熟悉的I2C。。

USART：Universal Synchronous Asynchronous Receiver Transmitter 通用同步/异步收发器 就是使用最广泛的RS232了

Debug MCU 貌似是调试接口相关的，比如JTAG？

Flash and Option Bytes Registers ： 从名字上看是Flash配置和系统配置位寄存器，后者可能是指跳线吧

漫长的寄存器位设定终于结束了。这么长篇的代码就差不多要结尾了。

/\*\* @addtogroup Exported\_macro

\* @{

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* ADC Instances \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define IS\_ADC\_ALL\_INSTANCE(INSTANCE) (((INSTANCE) == ADC1) || \

((INSTANCE) == ADC2) || \

((INSTANCE) == ADC3))

#define IS\_ADC\_MULTIMODE\_MASTER\_INSTANCE(INSTANCE) ((INSTANCE) == ADC1)

#define IS\_ADC\_COMMON\_INSTANCE(INSTANCE) ((INSTANCE) == ADC12\_COMMON)

#define IS\_ADC\_DMA\_CAPABILITY\_INSTANCE(INSTANCE) (((INSTANCE) == ADC1) || \

((INSTANCE) == ADC3))

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* CAN Instances \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define IS\_CAN\_ALL\_INSTANCE(INSTANCE) ((INSTANCE) == CAN1)

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* CRC Instances \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define IS\_CRC\_ALL\_INSTANCE(INSTANCE) ((INSTANCE) == CRC)

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* DAC Instances \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define IS\_DAC\_ALL\_INSTANCE(INSTANCE) ((INSTANCE) == DAC)

。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。

这里定义了一系列宏函数，用于判断所给参数是否为相关结构体的实例，后面应该会经常用到的。

好了，最后一段代码了哦~

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* For a painless codes migration between the STM32F1xx device product \*/

/\* lines, the aliases defined below are put in place to overcome the \*/

/\* differences in the interrupt handlers and IRQn definitions. \*/

/\* No need to update developed interrupt code when moving across \*/

/\* product lines within the same STM32F1 Family \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

又是一段贴心提示，大体意思是说：从不同产品线之间移植代码是一件烦心事，所以这里附上一段暖心代码，解决了在F1系列内部切换芯片的一些麻烦。

/\* Aliases for \_\_IRQn \*/

#define ADC1\_IRQn ADC1\_2\_IRQn

#define DMA2\_Channel4\_IRQn DMA2\_Channel4\_5\_IRQn

#define TIM1\_BRK\_TIM15\_IRQn TIM1\_BRK\_IRQn

#define TIM1\_BRK\_TIM9\_IRQn TIM1\_BRK\_IRQn

#define TIM9\_IRQn TIM1\_BRK\_IRQn

。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。

#define USB\_HP\_IRQn USB\_HP\_CAN1\_TX\_IRQn

#define USB\_LP\_IRQn USB\_LP\_CAN1\_RX0\_IRQn

#define CAN1\_RX0\_IRQn USB\_LP\_CAN1\_RX0\_IRQn

/\* Aliases for \_\_IRQHandler \*/

#define ADC1\_IRQHandler ADC1\_2\_IRQHandler

#define DMA2\_Channel4\_IRQHandler DMA2\_Channel4\_5\_IRQHandler

#define TIM1\_BRK\_TIM15\_IRQHandler TIM1\_BRK\_IRQHandler

#define TIM1\_BRK\_TIM9\_IRQHandler TIM1\_BRK\_IRQHandler

。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。

#define USB\_HP\_IRQHandler USB\_HP\_CAN1\_TX\_IRQHandler

#define USB\_LP\_IRQHandler USB\_LP\_CAN1\_RX0\_IRQHandler

#define CAN1\_RX0\_IRQHandler USB\_LP\_CAN1\_RX0\_IRQHandler

其实就是为中断变量的不同版本之间起了个别名，方便移植而已。

好了，stm32f103xe.h文件就分析到这里，收获还是不小的至少认识了很多英文名词嘛~