Usart驱动框架分析

在充分使用HAL方式操作usart周边模块后，我们对ST的HAL驱动框架有了一些直观的感觉，显然，HAL架构将周边模块进行了很好的代码包装，并形成了对周边功能模块的规范化驱动编写格式，有必要对这种驱动框架进行回顾，深入理解。

HAL对任一周边模块的驱动，一般包括2到3个文件构成：n指型号，如4，7，mmm指模块名称，如usart

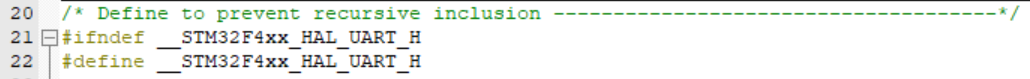
Stm32Fnxxx\_hal\_mmm.h //模块实体抽象，数据存储，动作函数定义

Stm32Fnxxx\_hal\_mmm.c //动作函数实现

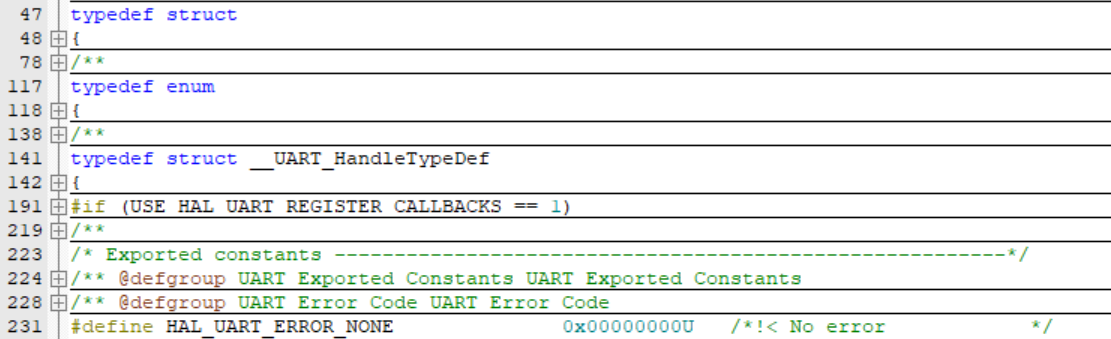
Stm32Fnxxx\_hal\_mmm\_ex.c //特殊功能实现，可选

软件模型建立，主要是通过Stm32Fnxxx\_hal\_mmm.h文件定义。

一．整体定义(以stm32f401re为例)



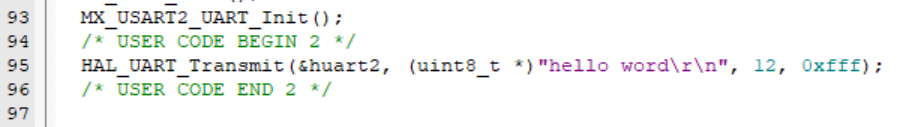
上述条件编译宏，可以防止多次该头文件被递归循环包括。

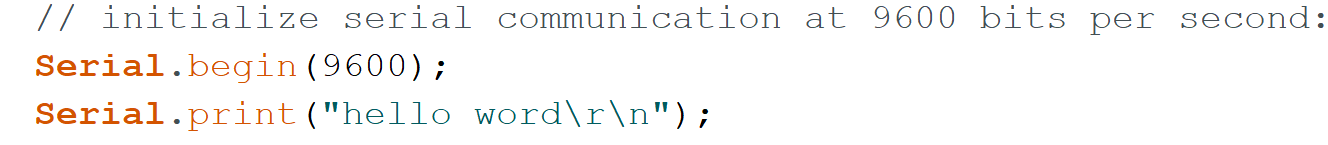
二．使用数据结构描述模块

一般来说，C语言基本数据类型很难描述复杂模块行为，因此，采用C语言的面向对象编程技术。这一技术主要要点是使用struct来组织模块数据结构，使用结构体内的函数指针成员来组织模块的行为动作。

实际上，上述两种技术是C++面向对象的C语言实现方法。下面，我们以一个C++类来说明面向对象模块模型抽象的强大。

首先，我们对比一下使用stm32 C语言和arduino C++描述的uart打印”hello word”的区别：





上图为stm32 C语言，arduino C++描述。

两者初始化相似，打印输出区别很大。C语言擅长过程描述，但需要通过特定硬件时，通过调用HAL\_UART\_Transmit函数的第一个参数传递实现；C++语言可以将UART硬件的属性（比如波特率，启停，每字节比特数等）和功能（输出，输入，新字符到达提醒等）整合到一个整体中。当需要使用这个硬件模块的功能时，使用Serial.功能形式调用。称描述硬件模块的Serial为对象（object）。基本这种面向对象的编程方法，程序员仅需要记忆软件工程中有多少类型的对象，它们的属性和能提供的功能函数即可实现项目设计。

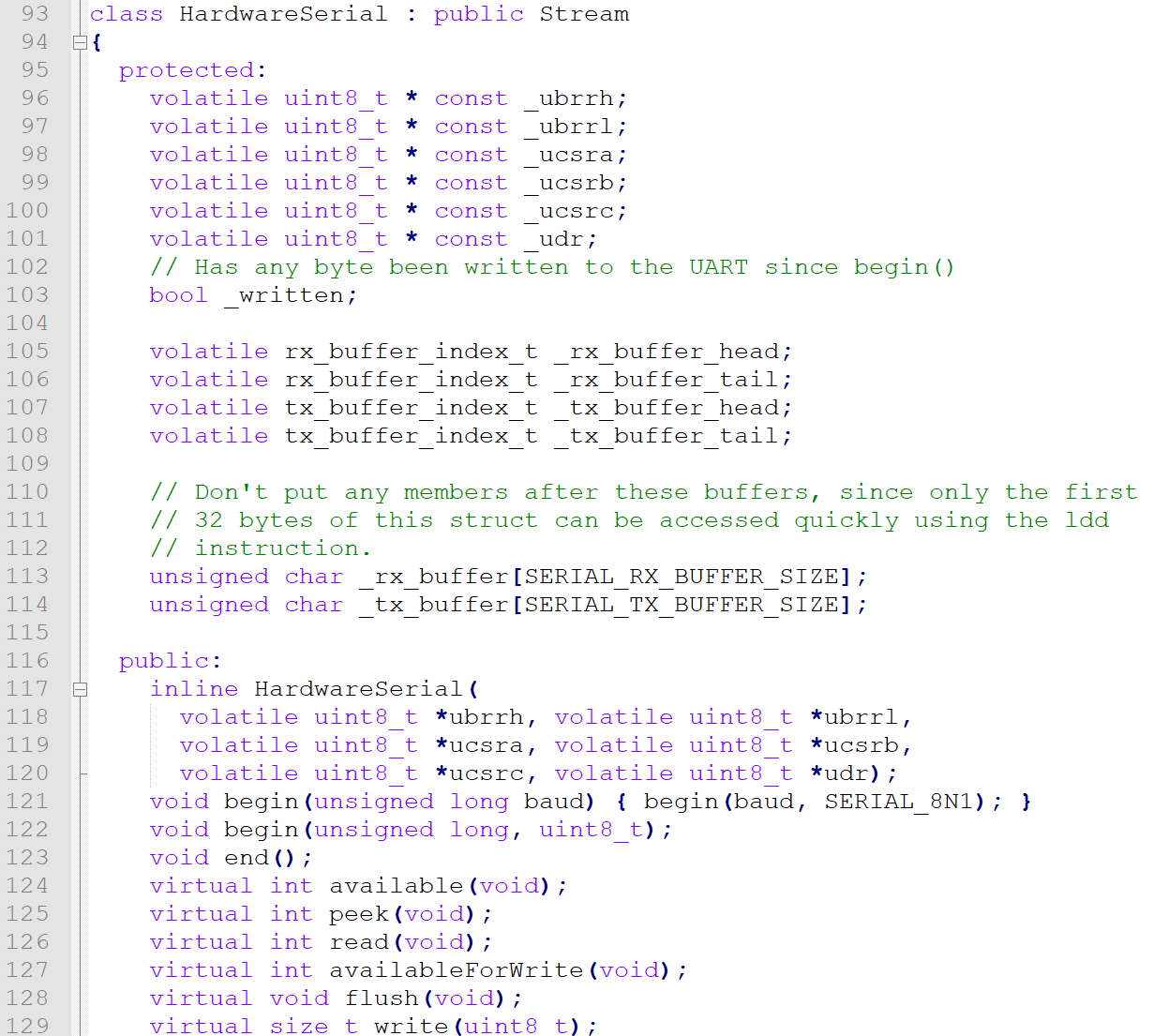
上述C++的面向对象软件建模程序设计方法，符合人类的思维习惯，这类似于“张三(.)说（print）（“hello word”）”，这里“张三”是对象，张三作为一个人当然他有能力（功能）说话，那么这个“说”动词就是张三这个对象的功能。类比上述实例：

张三 说 hello word

Serial print hello word

将对象，功能和内容组合起来，就形成了Serial.print(“hello word”)。其中”.”相当于对象”的”介词。

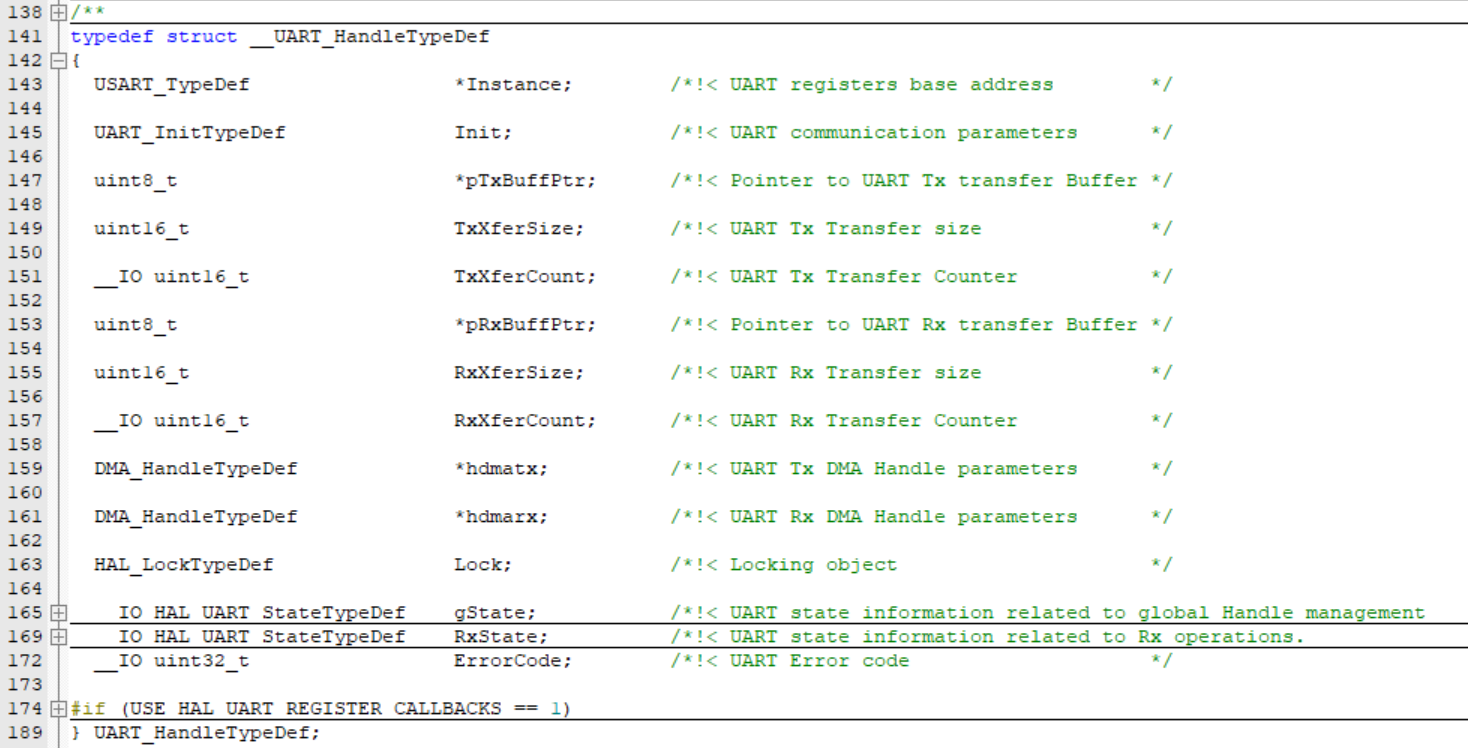
下面，我们简单看一下使用面向对象C++软件建模的Serial是如何设计的



C++软件建模的对象抽象称为“类（class）”，类是对象的抽象，即一种数据类型或结构。类的实例即为对象。93行通过关键词class hardwareSerial申明了新的数据类型，这等同于C语言的struct。但不同于C语言的struct只能组合基本数据类型，如int, float等，类可以组合基本类型和函数，如126行。通过将基本数据和函数组合在一起，实现了数据和功能（动作）的整合，这相似于描述一个人，他的属性，如高矮胖瘦，头发颜色，资产等，这些可以通过数据来说明；那么人的可能行为，如通过资产购买一台车辆，则通过函数执行。当然通过行为，可能对他的属性（数据）产生影响，也可能对其它对象产生影响，如车辆占用了其它对象的车位。这是面向对象设计的基本原理和思想。

另外，对象抽象对数据和行为进行了类别划分，即通过关键词，public, private, protected等对对象自身和其它对象访问数据的权限进行区别。一般来说对象自身可以访问类的所有权限数据，而其它对象仅能访问public权限的数据。这称为数据保护。另外，对象类抽象还规定了对象类间的关系，称为继承和派生，这样新类可以重复使用继承或派生源类的属性和功能，这好像是儿子具有一些父亲和母亲的外貌特征。如93行class hardwareSerial后面的“:public Stream”。

这样看起来，C++软件建模的对象类抽象的确是很强大了，但万事都有两面性，对象类抽象在提供强大功能的同时，需要更多的存储空间和更多一些的计算资源，这在存储和计算资源都有限的嵌入式系统中，有可能是致命的问题。通过近20年的嵌入式硬件发展和实践，现在嵌入式系统主要的软件建模和开发环境以C语言为主，但也将一些对象类抽象的思想改进基本C语言的发展，称为面向对象C语言。下面，我们来深入看下ST如何使用面向对象C语言建模uart硬件。



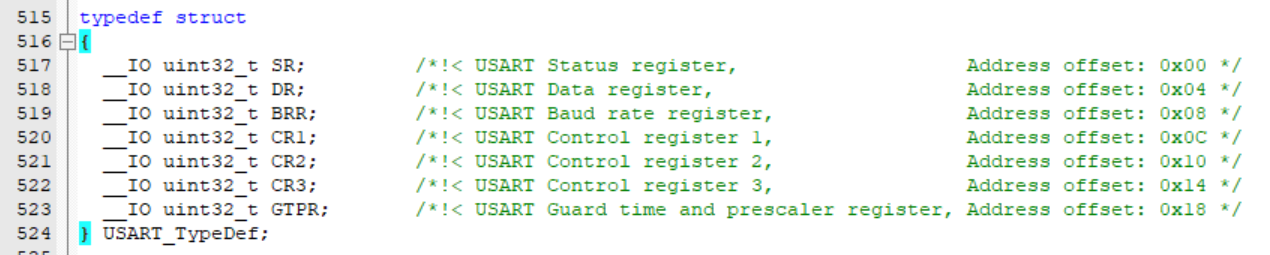
在前述Stm32Fnxxx\_hal\_mmm.h文件中通过struct定义了一个组合新数据类型”\_UART\_HandleTypeDef”，并通过typedef给这个结构体新数据类型起了一个别名“UART\_HandleTypeDef”。

这个结构体相较于C++的类（class）只包含了数据，没有函数，并且对数据没有访问权限分类，即所有的数据外部都是可以访问的。这当然在安全性上相较于对象类有一定的缺陷。

这个结构体分为六个子功能区：

|  |  |  |
| --- | --- | --- |
| 序号 | 功能区（行号） | 功能 |
| 1 | 143 | 硬件模块寄存器，代表这个模块实例 |
| 2 | 145 | 硬件模块寄存器初始化参数 |
| 3 | 147-157 | 收发数据缓冲区 |
| 4 | 159-161 | 协助模块句柄指针 |
| 5 | 163 | 互斥锁定 |
| 6 | 165-169 | 状态机 |

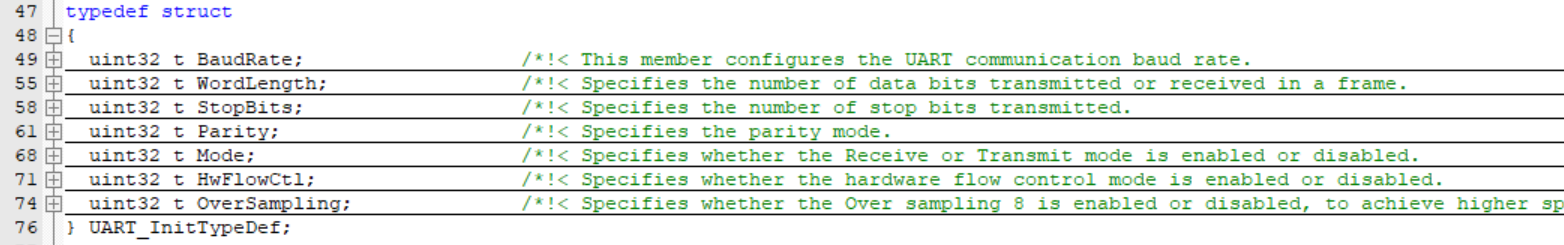
143行的USART\_T ypeDef如下(以stm32f401re为例)



其中,\_IO是#define \_\_IO volatile，指明这个变量不可被编译器优化，即指定的地址是确定不变的。这些32bit的变量，对应了硬件模块的储存器，软件通过读写这些储存器操作硬件模块，注意这里定义的32bit变量，SR，DR，CR等是相对这个模块基地址的偏移量，是控制这个模块所有功能的储存器。这些储存器的具体定义可参考对应MCU的数据手册（datasheet）中对应模块的储存器说明。

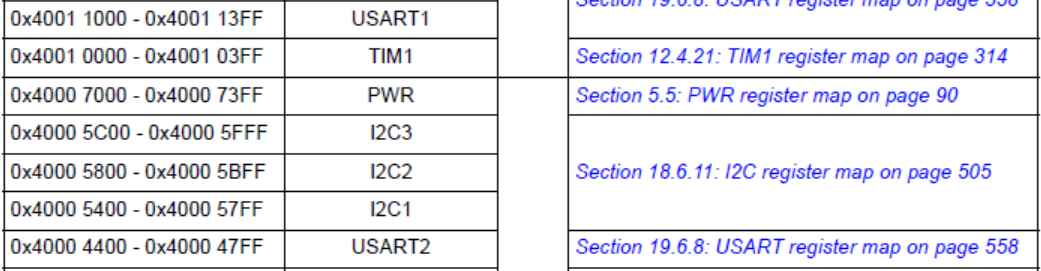
这里还有一个有趣的问题，我们通过struct定义的多个uint32\_t 的变量是否能“准确”地对应到实际硬件的寄存器地址？比如，571行能否写为\_IO uint8\_t SR? 或者说，我们需要理解为什么所有的偏移子功能储存器要定义为uint32\_t? 为回答这个问题，需要理解struct成员变量在内存中的映射问题。首先，struct中的变量是顺序连续内存映射分布，因此，上述SR，DR，BRR…他们的地址偏移（offset）刚好是4，内存按字节组织，即4字节一个储存器，即32bits。那么如果第571行修改为“\_IO uint8\_t SR”，那么是否DR储存器在0x01? 这个实际上与编译器有关，针对MCU的编译器来说，存在一个所谓struct的“内存对齐”的概念。如上述“\_IO uint8\_t SR”定义，DR仍然在0x04。进一步地，存在两行“\_IO uint8\_t SR，\_IO uint8\_t fake1”到“\_IO uint8\_t SR，\_IO uint8\_t fake1，…\_IO uint8\_t fake3”，DR仍然在0x04。而fake1地址为0x01, fake2地址为0x02, fake3地址为0x03。相关进一步的理解，可以自行查询struct “内存对齐”知识。

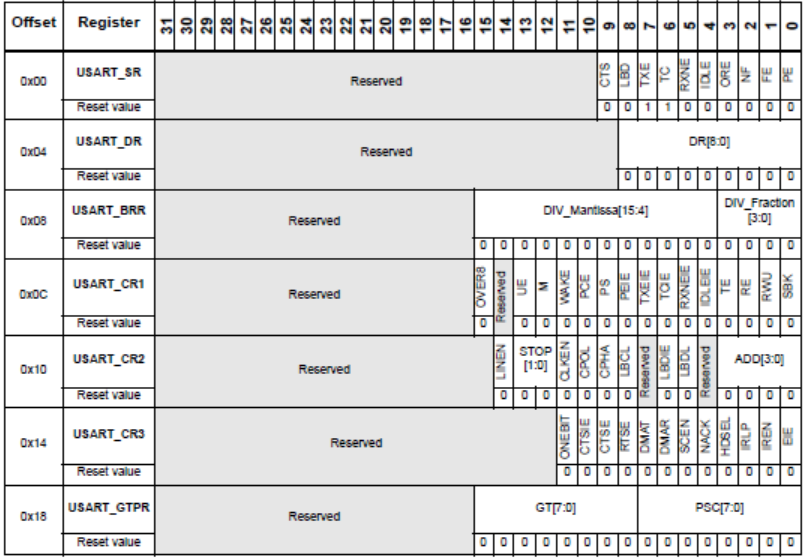
下一个是UART\_InitTypeDef，定义如下



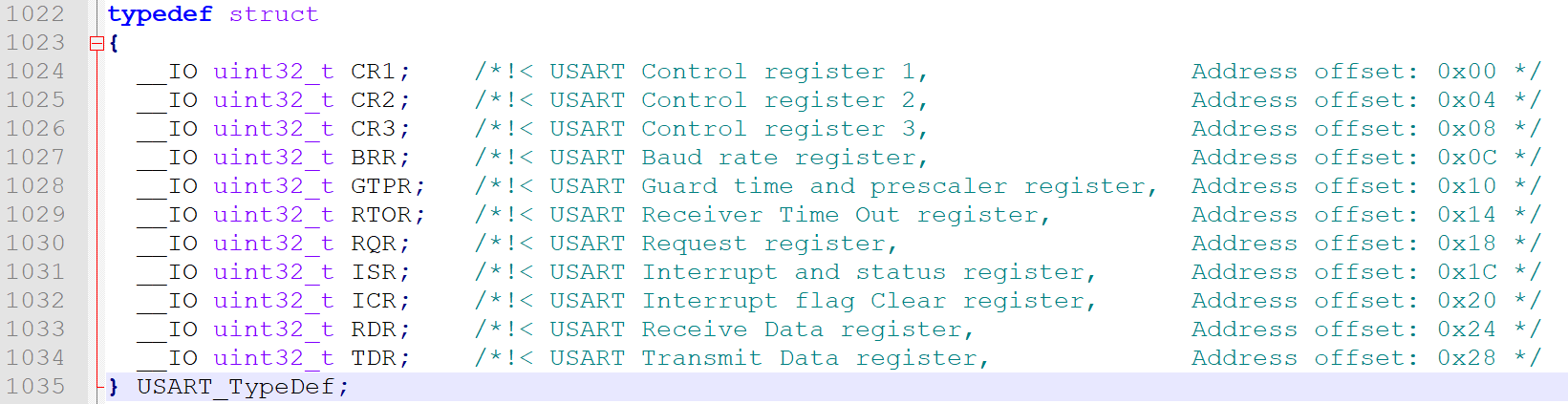
这里包含的是这个模块的配置参数，没有内存映射分布问题，可以乱序组织。

另外，对比\_\_UART\_HandleTypeDef定义的USART\_TypeDef和UART\_InitTypeDef成员变量，第一个为\*Instance，第二个为Init。储存器内存储的是地址，init储存的是配置参数。这样只要对Instance=0x4001 1000(以stm32f401re为例)即可完成其它偏移储存器对准了。

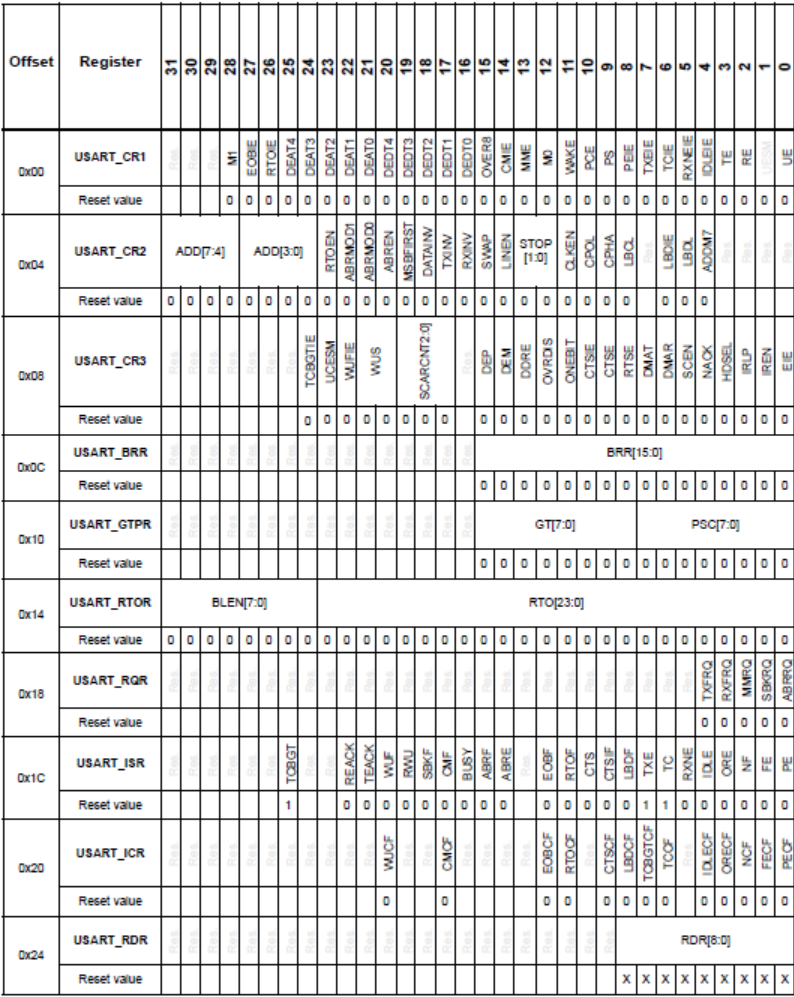




而如果使用的MCU是stm32F769,那么它的USART\_TypeDef为如下







对比stm32F401re和stm32F769ig的内存映射，可见他们的USART子储存器分布和数量都不同。

三．以功能为核心，整合其它硬件模块协同工作

通过一二节分析，一个基本的USART子模块可以实现一些基本的功能了，比如通过CR1,2,3对该模块的参数进行配置，通过USART\_RDR接收一个8bits的字符，等。但实际的应用需求来说，会关注更高层次的功能，比如，通过前面实验过的DMA方式批量传输数据。为了达到上述功能，单一的USART子模块需要与其它子模块进行功能整合。

一种整合的思路是这些功能由用户（程序员）来实施，即通过各子模块功能的整合来实现更高层次的应用。

另一种整合是思路是ST将这些子模块功能整合成新的类型定义，提供给用户，如果用户通过CUBEMX对这些多个子模块进行了配置（初始化），那么通过ST HAL框架即可方便使用定义好的高层功能函数。

另外，为了解决特定功能的C语言编写的函数的重载问题，使用了\_weak关键词的技术手段。

前述UART\_HandleTypeDef类型定义里的第159和161行，DMA\_HandleTypeDef即是将DMA子模块句柄整合到UART\_HandleTypeDef类型中来，如果需要将UART的数据进行DMA传输，那么通过配置好的DMA子模块来协合完成。反之，如果不需要DMA，那么第159和161行指针指向NULL来表示。

同样，为了实现多字节字符接收与传输，UART\_HandleTypeDef类型定义里的第147和153行包含了两个缓冲区用于中断或DMA方式数据的缓存。

除上述功能外， ST HAL还考虑了硬件访问的互斥问题，即引入第161行的HAL\_LockTypeDef，这个类型变量Lock的作用是保证UART在一个时间内仅能被一个硬件读取。

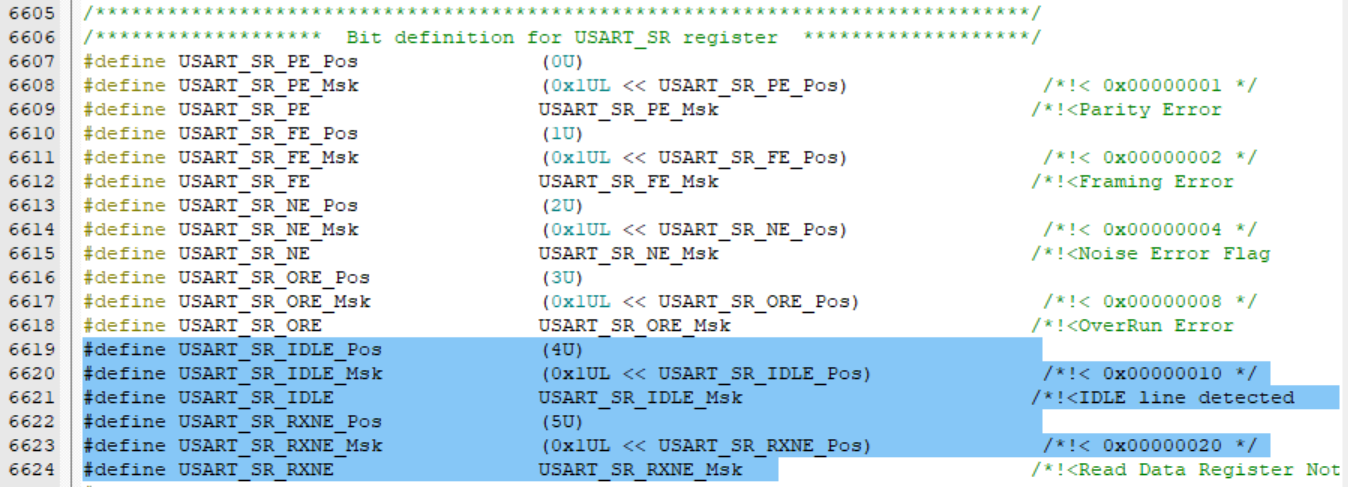
最后，还增加了两个状态变量用于描述USART工作状态。

综上四种功能的整合，DMA的协作算是硬件子模块间的整合，其它如缓冲和状态机等都是软件模块。实际上，可以这样说，ST的HAL固件库实现：1）硬件储存器的内存映射（USART\_TypeDef）；2）以应用为核心的多模块集成；3）使用1，2两点，以应用为核心的功能函数定义（.h中进行export函数定义; .c中实现功能函数）。

回顾前述的C与C++建模区别，ST的HAL固件库没有将功能函数包装到UART\_HandleTypeDef类型中，而是使用功能函数接收UART\_HandleTypeDef类型的指针的方案，比如HAL\_StatusTypeDef HAL\_UART\_Transmit\_DMA(UART\_HandleTypeDef \*huart, uint8\_t \*pData, uint16\_t Size)。

四．使用宏定义对32位寄存器内比特进行操作

为了对32位宽度的寄存器内的特定位进行比特操作，使用如下示的技巧



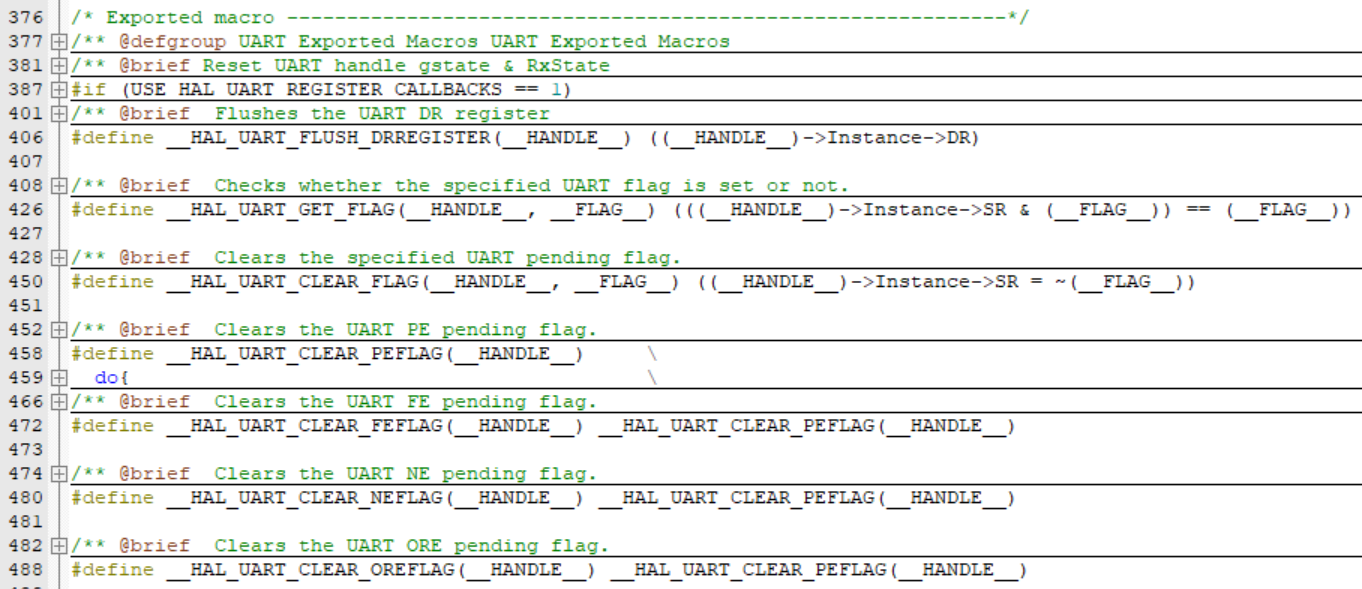
上图以stm32F401re为例，通过将0x1UL向左移动Pos位确定特定比特位为‘1’而其它位为全‘0’。这里还有一个问题，为什么要将‘1’强制定义为‘0x1UL’? UL是Unsigned Long的意思，它的长度刚好是32bits。

那么如何对一个32bits长的储存器使用上述位定义进行位操作？

1. 判断存在特定比特位，如USART\_SR\_RXNE位，(isrflags & USART\_SR\_RXNE) != RESET)，其中isrflags就是USART的SR 32bits宽度储存器的缓存版本。RESET宏就是32bits宽度的‘0’。
2. 设置，清除等比特位操作方法：stm32f4xx.h  
   

例如：CLEAR\_BIT(huart->Instance->CR1, USART\_CR1\_PEIE)，CR1是32bits宽度储存器，USART\_CR1\_PEIE宏是仅一位的32bits宽宏定义。

1. ST HAL对特定模块的设置，清除等比特位操作方法：stm32f4xxx\_hal\_uart.h



显然，这些宏定义是针对USART模块的通过位操作的包装，但使用更方便。宏格式为\_HAL\_UART\_XXX\_XXX。

五．对片外模块的驱动方法

上述一至四节，主要以USART子模块为例进行了深入的ST HAL驱动构架分析。本节讨论如何应用ST HAL驱动构架来实现对片外模块的驱动。

本节以MCP23S17芯片为例，有关MCP23S17的详细文档，可参考<https://www.microchip.com.cn/newcommunity/>中对应中英文数据手册。

简单地说，这是个通过SPI总线转换为16个（分为两个8位 bank）可配置IO的芯片。

下图是SPI接口

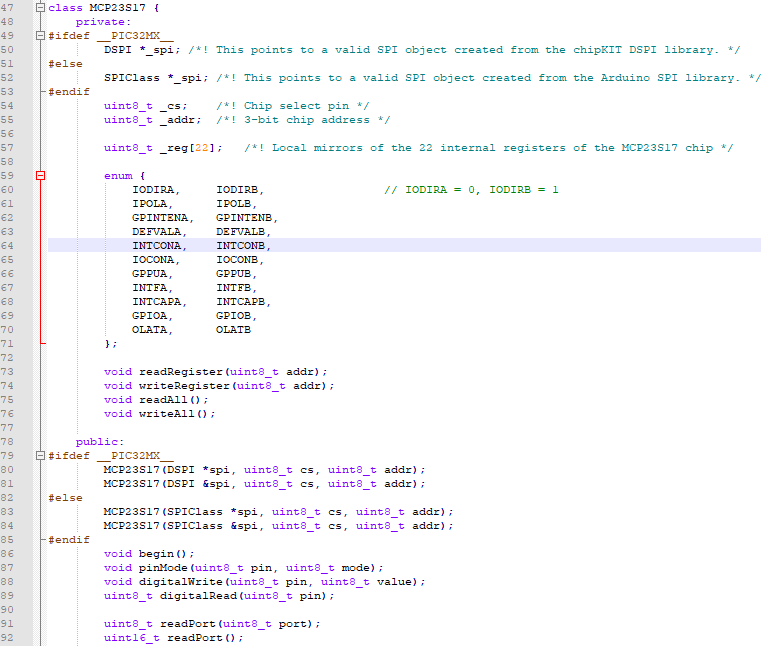
R/W位，0：写；1：读

每储存器长度为一字节8bits，因此，任意一次读写需要3字节24bits。

相比于MCU片内模块，片外模块的储存器无法通过硬件地址内存映射的方法进行寻址，所有的读写操作都必须通过SPI接口。

下面，我们先看一下基于arduino的C++面向对象对MCP23S17的软件封装，这来至于通过github.com检索“MCP23S17 arduino”关键词获得源代码。





比较重要的定义是第52行SPIClass \*\_spi，第57行uint8\_t \_reg[22]和第59-70行。因为片外模块必须通过SPI接口进行操作，因此，MCP23S17软件模块需要包含一个SPI句柄指针。其次，MCP23S17内部储存器数据也在软件模块中存储了一份，以方便根据这些储存器值组织动作函数。最后，由IOCON.BANK=0时，MCP23S17储存器地址连续顺序排列，因此，可以使用第59-70行的枚举变量来表示不同功能的储存器地址。

通过上述分析，可以参考UART\_HandleTypeDef类型定义方法，定义MCP23S17软件模块的类型句柄：



相关的实现这里篇幅有限就不作详细讨论了。