

# STM32 缓存的对齐问题

## 前言

在我们对 STM32 进行编程的时候,都会用到变量,因为我们的 MCU 是 32 位的,所以在申请变量的时候,就会存在变量长度不一致,需要对齐的问题.这个变量长度对齐的问题,小则可以只是影响代码执行的效率,大则会出现系统 hard-fault 的问题.下面我们将详细的解说这个问题.

## 不对齐的后果

### 效率

需要对变量进行对齐的一个重要原因是因为 MCU 访问数据的效率问题.例如,如果一个变量的地址没有对齐,比如在 0x00000002,则 MCU 如果需要读取它的值,则需要进行两次访问.如果在 0x00000003 的话,则需要三次访问,然后整合读取的数据.而如果数据是对齐的话,例如 0x00000004,那么只需要一次访问就可以了.但是现在的 IDE 都比较智能,就算你强制让一个变量放到基数或者其他非对齐的地址上时,他是会报错的.

```
struct stu my_stu __attribute__((at(0x20001001))) = {0};
```

编译报错：

```
Error: L6984E: AT section main.o(.ARM.__AT_0x20001001) has required base address 0x20001001 which is not aligned to section alignment 0x00000008.  
Finished: 0 information, 0 warning and 1 error messages.
```

### 空间

变量地址对齐了,就会带来空间的浪费,但这个一般体现在结构体上.由下面的例子我们可以看出,就算一样的结构体,不同的顺序,会带来不一样的总长度.在 Struce B 中,char 虽然物理上是 2 个 byte,但是由于对齐的原因,加上他紧随的变量和他的变量不一致,所以他需要额外的 2 个 byte 来使结构体内部对齐.所以在定义结构体的变量时,我们一般最好做到同样的顺序,即使类型从小到大排序,或着从大到小来排序.这样可以带来空间上的节省.最直接的方式就是你可以用 sizeof() 这个函数来查看结构体的大小

```
//以下这种结构体的排序长度为 8 个 byte.  
typedef struct  
{  
    int a;    //4  
    char b;   //2  
    short c;  //2  
}Struce_A;
```

Watch 1			Memory 1	
Name	Value	Type	Address:	0x20001200
my_stu	0x20001000 &my_stu	struct stu	0x20001200:	01 00 00 00 02 00 FF FF
k	0x00000000	int	0x20001213:	D4 E1 C8 BA DE B6 D1 A2
s_a	0x20001200 &s_a	struct <u...	0x20001226:	EA CC 2E 60 0F 46 77 AD
a	0x00000001	int	0x20001239:	CA 73 21 DC 20 0D 49 92
b	0x02	char	0x2000124C:	11 91 2D 7E D6 AB B4 D2
c	0xFFFF	short	0x2000125F:	C9 61 D5 84 05 C9 00 16

```
//以下这种排序是 12 个 BYTE 的
typedef struct
{
    char b; //4
    int a; //4
    short c; //4
}Struce_A;
```

Watch 1			Memory 1	
Name	Value	Type	Address:	0x20001200
my_stu	0x20001000 &my_stu	struct stu	0x20001200:	01 00 00 00 02 00 00 00 FF FF 00 00
k	0x00000000	int	0x20001213:	D4 E1 C8 BA DE B6 D1 A2 8F 41 C0 C3
s_a	0x20001200 &s_a	struct <u...	0x20001226:	EA CC 2E 60 0F 46 77 AD 0A 04 EA CC
b	0x01	char	0x20001239:	CA 73 21 DC 20 0D 49 92 64 EA C1 34
a	0x00000002	int	0x2000124C:	11 91 2D 7E D6 AB B4 D2 70 DC EC D5
c	0xFFFF	short	0x2000125F:	C9 61 D5 84 05 C9 00 16 61 A9 16 49

## 综合

通过上述,一般以为只要按照最好的排序方式,结构体就会对齐了.其实并不是这样的,他只会对不同类型的变量之间做对齐,如果如下面的结构体.

```
typedef struct
{
    Uint8_t a1; //1
    uint8_t a2[5]; //7
    uint32_t a3[2]; //4
}Struce_A;
```

因为第一个和第一个变量的类型是一致的,所以系统会合并,所以 a1+a2 的大小才是 8byte.如果如下,就会对齐了,但是长度就会编程 16byte.

```
typedef struct
{
    Uint32_t a1;    //4
    uint8_t a2[5];  //8
    uint32_t a3[2]; //4
}Struce_A;
```

在第一个例子里面会存在效率的问题,如果在一些低功耗上,或者对算法实时性要求较高的地方,就会是个不小的问题.为什么?因为 **a2** 没有对齐.这会带来效率上的问题,下面我们通过实际代码来测试这个效率差.

## 案例与总结

### 案例 1 :

//首先定义变量,为了更好的查看里面的数据,我将其放到一个固定的地址.

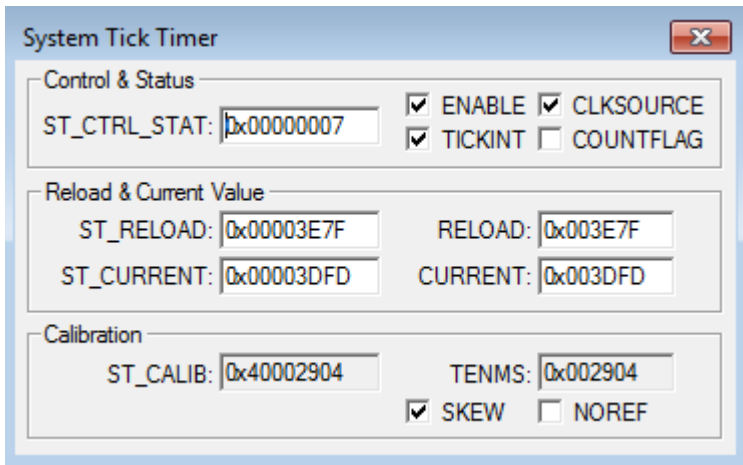
```
struct stu{
    uint32_t a1;
    uint8_t a2[5];
    uint32_t a3;
};
struct stu my_stu __attribute__((at(0x20001000))) = {0};
```

Watch 1			Memory 1	
Name	Value	Type	Address: 0x20001000	
my_stu	0x20001000 &my_stu	struct stu		
a1	0x00000001	unsigned...		
a2	0x20001004 " □"	unsigned...		
[0]	0x02	unsigned...	0x20001000:	01 00 00 00 02 03 04 00 00 00 00 00 00 00
[1]	0x03	unsigned...	0x20001013:	E8 13 44 54 69 B2 FE 6F 9E F5 E8 7F 97
[2]	0x04	unsigned...	0x20001026:	60 84 59 D1 3A 6A 4E 6A 90 59 9E 94 18
[3]	0x00	unsigned...	0x20001039:	76 39 EF A4 8A 68 12 CD D1 40 F1 D7 CB
			0x2000104C:	AC 86 45 40 E8 DD CD FB 03 46 86 B0 4B
			0x2000105F:	95 97 76 E9 64 F4 AD E8 1E 70 29 B9 1E
			0x20001072:	03 C2 D5 2B 5B 4C 8E F8 FD A2 26 4D 15
			0x20001085:	2F 4B 57 00 C5 2F 75 C2 40 40 12 F0 05

通过一个 for 循环来检测他的运行时间.

```
for(i = 0;i < 1000;i++)
{
    my_stu.a2[0] += x1;
    if( my_stu.a2[0] > 200)
        my_stu.a2[0] = 0;
}
```

这串代码所需的 tick  
运行之前:0x3D3F



System Tick Timer

Control & Status

ST\_CTRL\_STAT: 0x00000007

☒ ENABLE ☒ CLKSOURCE

☒ TICKINT ☐ COUNTFLAG

Reload & Current Value

ST\_RELOAD: 0x00003E7F RELOAD: 0x003E7F

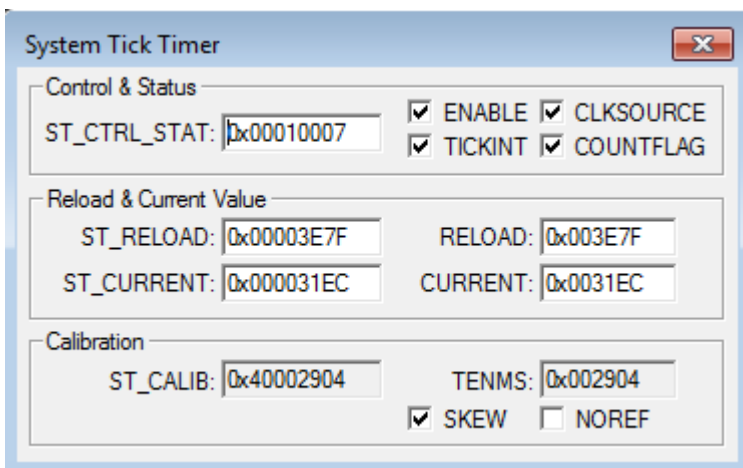
ST\_CURRENT: 0x00003DFD CURRENT: 0x003DFD

Calibration

ST\_CALIB: 0x40002904 TENMS: 0x002904

☒ SKEW ☐ NOREF

运行之后:0x31EC



System Tick Timer

Control & Status

ST\_CTRL\_STAT: 0x00010007

☒ ENABLE ☒ CLKSOURCE

☒ TICKINT ☒ COUNTFLAG

Reload & Current Value

ST\_RELOAD: 0x00003E7F RELOAD: 0x003E7F

ST\_CURRENT: 0x000031EC CURRENT: 0x0031EC

Calibration

ST\_CALIB: 0x40002904 TENMS: 0x002904

☒ SKEW ☐ NOREF

可以看出这段代码所需时间为:  $0x3D3F - 0x31EC = 0x0B53$

## 案例 2 :

//这里我们将 a1 改成 uint8\_t,这种方式就是很多 C 语言教材里面推荐的方式,结构体的排序由小到大排序,后面我们看看结果.

```
struct stu{
    uint8_t a1;
    uint8_t a2[5];
    uint32_t a3;
};
```

有下图我们可以看到 a1 和 a2 之间并不像案例 1 之间有填补.

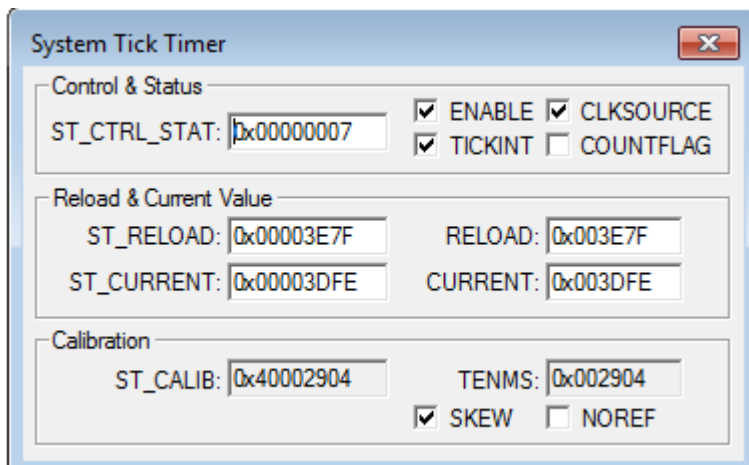
Watch 1			Memory 1	
Name	Value	Type	Address: 0X20001000	
my_stu	0x20001000 &my_stu	struct stu	0x20001000: 01 02 03 04 05 00 0	
a1	0x01	unsigned...	0x20001013: 00 00 00 00 00 00 0	
a2	0x20001001 " □  "	unsigned...	0x20001026: 00 00 00 00 00 00 0	
[0]	0x02	unsigned...	0x20001039: 00 00 00 00 00 00 0	
[1]	0x03	unsigned...	0x2000104C: 00 00 00 00 00 00 0	
[2]	0x04	unsigned...	0x2000105F: 00 00 00 00 00 00 0	
[3]	0x05	unsigned...	0x20001072: 00 00 00 00 00 00 0	

同样通过一个 for 循环来检测他的运行时间.

```
for(i = 0; i < 1000; i++)
{
    my_stu.a2[0] += x1;
    if( my_stu.a2[0] > 200)
        my_stu.a2[0] = 0;
}
```

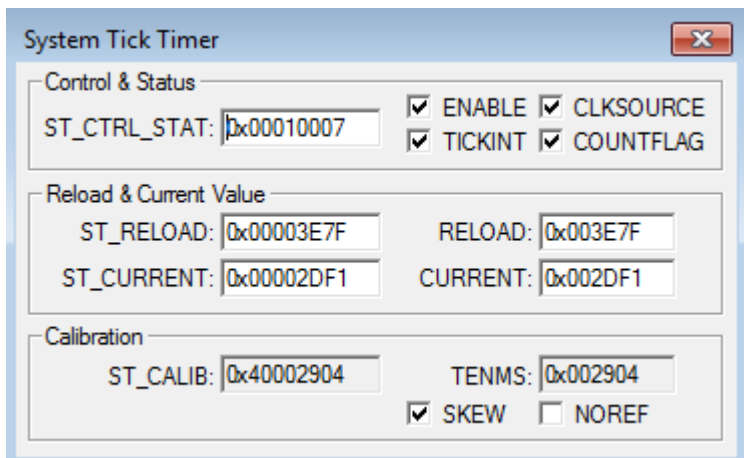
这串代码所需的 tick

运行之前:0x3D3F



The image shows the 'System Tick Timer' configuration window. It has three sections: 'Control & Status', 'Reload & Current Value', and 'Calibration'. In 'Control & Status', 'ENABLE', 'CLKSOURCE', and 'TICKINT' are checked, while 'COUNTFLAG' is unchecked. 'ST\_CTRL\_STAT' is set to 0x00000007. In 'Reload & Current Value', 'ST\_RELOAD' is 0x00003E7F, 'RELOAD' is 0x003E7F, 'ST\_CURRENT' is 0x00003DFE, and 'CURRENT' is 0x003DFE. In 'Calibration', 'ST\_CALIB' is 0x40002904, 'TENMS' is 0x002904, and 'SKEW' is checked while 'NOREF' is unchecked.

运行之后:



The image shows a 'System Tick Timer' configuration window. It has three main sections: 'Control & Status', 'Reload & Current Value', and 'Calibration'. In 'Control & Status', 'ST\_CTRL\_STAT' is set to 0x00010007, and 'ENABLE', 'CLKSOURCE', 'TICKINT', and 'COUNTFLAG' are all checked. In 'Reload & Current Value', 'ST\_RELOAD' is 0x00003E7F, 'RELOAD' is 0x003E7F, 'ST\_CURRENT' is 0x00002DF1, and 'CURRENT' is 0x002DF1. In 'Calibration', 'ST\_CALIB' is 0x40002904, 'TENMS' is 0x002904, and 'SKEW' is checked while 'NOREF' is unchecked.

可以看出这段代码所需时间为:  $0x3D3F - 0x2DF1 = 0x0F4E$

#### 总结：

方案	运行前	运行后	所需时间	
案例一	0x3D3F	0x31EC	0x0B53	35.1%
案例二	0x3D3F	0x2DF1	0x0F4E	

可以看出,简单的三行代码,在是否对齐下,增加 35.1%的时间,如果在一些复杂度要求较高的应用,低功耗的应用中,这将会是一个很大的问题.

#### 建议：

所以如果在一些比较大或者运用的比较多的结构体,我们可以人为的错开不同的类型变量,因为就算你外部添加了强制的对齐命令,在现有的 IDE 里面(KEIL 5.20),暂时还没有对同种类型的变量进行对齐.如果基本都是同种类型的变量,建议可以嵌套结构体来避开.这样错开了,剩下的就有系统帮你完成对齐动作.

### **重要通知 – 请仔细阅读**

意法半导体公司及其子公司（“ST”）保留随时对ST 产品和/ 或本文档进行变更、更正、增强、修改和改进的权利，恕不另行通知。买方在订货之前应获取关于ST 产品的最新信息。ST 产品的销售依照订单确认时的相关ST 销售条款。

买方自行负责对ST 产品的选择和使用， ST 概不承担与应用协助或买方产品设计相关的任何责任。

ST 不对任何知识产权进行任何明示或默示的授权或许可。

转售的ST 产品如有不同于此处提供的信息的规定，将导致ST 针对该产品授予的任何保证失效。

ST 和ST 徽标是ST 的商标。所有其他产品或服务名称均为其各自所有者的财产。

本文档中的信息取代本文档所有早期版本中提供的信息。