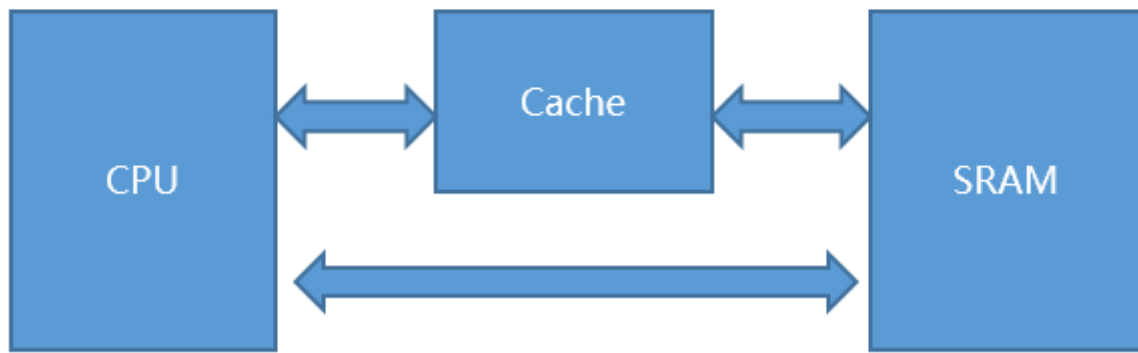


STM32解决Cache与DMA冲突的问题

为什么要开启Cache

虽然stm32h723的主频可达到550MHz,乍一看很高的亚子,但是实际的运行是伴随着数据交互的,CPU需要对RAM和Flash执行读写操作,这里面就涉及到了等待周期的概念(不深入解说了),往往读写RAM只能达到200MHz的速度,内核这叫要说话了:兄弟我550你200,差这么多玩鸡毛.就好比打比赛你负责代码,另外一个兄弟负责机械,你代码一天敲完了,机械的兄弟还在新建文件夹(不恰当的比喻).我只能说:what can i say? 内核就这么被RAM把速率给拖下来了.

为了解决这个问题,就有了Cache,Cache的速率可以到400MHz以上(具体看手册).我们直接看图.



CPU在数据交互过程中会先看一下Cache中是否已经开辟了对应数据地址的空间,如果已经开辟了那就是Cache命中;如果没开辟,那就是Cache miss.

当Cache命中的时候,CPU会直接与Cache进行数据交互(具体是否与RAM进行交互要看配置),读写速度就会达到和M7内核主频匹配的速率,这才是真正的高速.

当Cache miss的时候,会根据配置决定是否为此地址数据在Cache中开辟空间.

但是其实Cache只有16k的大小.实际Cache里面记载的数据地址会一直的变换.

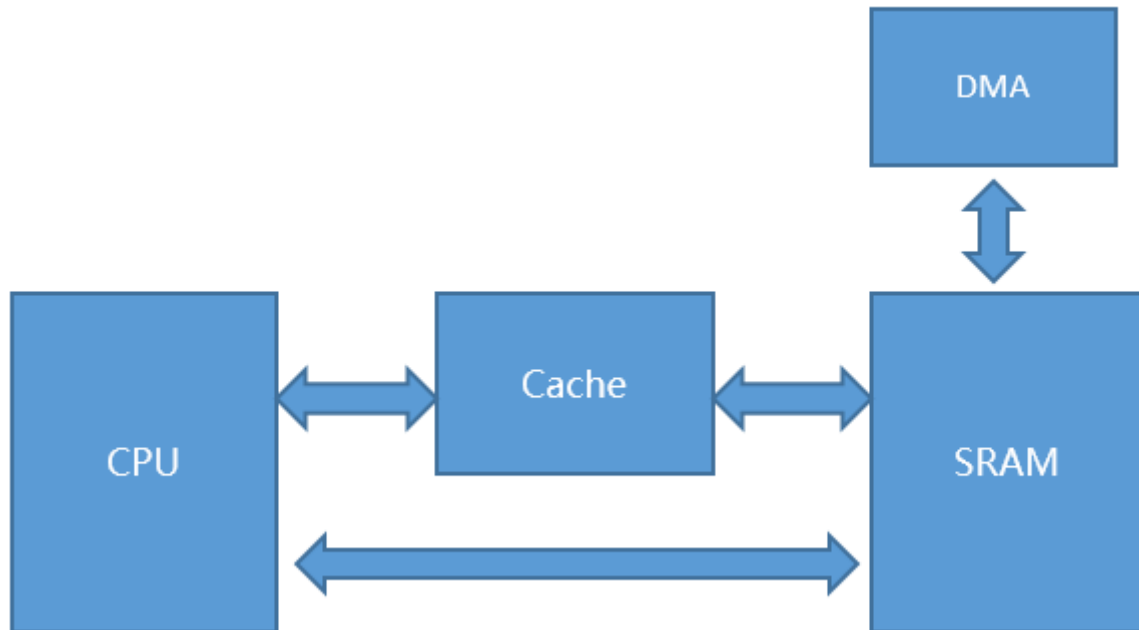
那这个时候有童鞋就要问了:你这有命中,又有miss,我代码数据量那么大你这只有16k,那我这命中率岂不是很低?

这涉及到 时间局部性和空间局部性,具体我不赘述了,本文主要讲解如何去避免Cache和DMA的冲突,事实上Cache的命中率是非常高的(有兴趣可以深入了解此文:

<https://blog.csdn.net/as480133937/article/details/123663197>).

怎么就冲突了

直接看图



可以看到, 当有Cache在的时候, CPU大概率在和Cache进行数据交互, DMA只能和SRAM进行交互, 所以我们大概就会发现, 外设的DR寄存器在疯狂的更新数据, 但是DMA的目标地址动都不动一下.

下面以MC02 ADC的例程举例

我在CubeMX中开启了Cache

Configure the below parameters :

⏪
⏩




- ✓ Speculation default mode Settings

Speculation default mode
 Disabled
- ✓ Cortex Interface Settings

CPU ICache
 Enabled

CPU DCache
 Enabled

可以看到Vbus变量是没有更新的

 vbus	0	float
 hdma_adc1	0x2400008C &hd...	struct __DMA_Ha...
 adc_val	0x24000000 adc_...	ushort[2]
<Enter expression>		

但是在寄存器里面看ADC的DR寄存器

ADC_DR	0x00001386
RDATA	0x00001386

是有值的, 很明显就是DMA出问题了

解决方案

这里直接说结论, 利用MPU配置直接禁用掉一段空间的Cache.

代码里面, DMA的目标地址是0x24000000(这个是正常的SRAM区域起始区域)

adc_val	0x24000000 adc_...	ushort[2]
[0]	0x0000	ushort
[1]	0x0000	ushort

我们直接配置MPU

<ul style="list-style-type: none"> <ul style="list-style-type: none"> Cortex Memory Protection Unit Control Settings <ul style="list-style-type: none"> MPU Control Mode Cortex Memory Protection Unit Region 0 Settings <ul style="list-style-type: none"> MPU Region MPU Region Base Address MPU Region Size MPU SubRegion Disable MPU TEX field level MPU Access Permission MPU Instruction Access MPU Shareability Permission MPU Cacheable Permission MPU Bufferable Permission 	Background Region Privileged accesses only + MPU Enabled during hard fault, NMI and FAULTMASK handlers Enabled 0x24000000 512B 0x0 level 1 ALL ACCESS PERMITTED ENABLE DISABLE DISABLE DISABLE
--	--

MPU有很多个区域, 我们仅需要配置其中一个区域用来放置DMA冲突, 其他的都可以配置成正常的读写.

解释一下参数

1. MPU Region: stm32H723有16个MPU管理区, 通过此配置来使能该区域
2. MPU Region Base Address: 此MPU区域的起始地址(只在SRAM有效)
3. MPU Region Size: MPU区域大小
4. MPU SubRegion Disable: 失能子区域, 共有八个子区域, 设置成0就可以全使能了(每一个bit对应一个子区域)
5. MPU Access Permisson : 此区域内存访问权限设置
6. MPU Instruction Access: 是否允许此区域执行代码

7. **MPU TEX field level** and **MPU Shareability Permission** and **MPU Cacheable Permission** and **MPU Bufferable Permission**. 这四个参数放到一起说, 就是来配置此区域的Cache. 看下图. C 为Cacheable Permission的配置, B为Bufferable Permission的配置, S为Shareability Permission的配置

TEX	C	B	S	Memory type	Shareability	Other attributes
0b000	0	0	x ⁽¹⁾	Strongly-ordered	Shareable	-
		1	x ⁽¹⁾	Device	Shareable	-
	1	0	0	Normal	Not shareable	Outer and inner write-through. No write allocate.
			1		Shareable	
		1	0	Normal	Not shareable	Outer and inner write-back. No write allocate.
			1		Shareable	
0b001	0	0	0	Normal	Not shareable	Outer and inner noncacheable.
			1		Shareable	
		1	x ⁽¹⁾	Reserved encoding		-
	1	0	x ⁽¹⁾	Implementation defined attributes.		-
		1	0	Normal	Not shareable	Outer and inner write-back. Write and read allocate.
			1		Shareable	

当TEX C B S 分别配置为 1 0 0 0(Level1 Disable Disable Disable)的时候 就是禁用掉Cache的配置, 与上图 [STM32解决Cache与DMA冲突的问题](#) 的配置一样.

此时再编译下载debug, 就会发现Vbus有值了

Name	Value	Type
vbus	2.77172804	float
hdma_adc1	0x2400008C &hd...	struct __DMA_Ha...
adc_val	0x24000000 adc_...	ushort[2]
<Enter expression>		

结语和建议

1. 此方案只供短暂项目所需. 建议在sct文件中单独开一个分区用来存放DMA搬运的数据, 然后单独配置一个MPU Region来对应. 非dma变量不需要存在此空间, 否则会Cache无效化(如果有这个例程需求我后面有空可以更一个)
2. 其他区域的MPU都需要去配置一下, 不然只会默认开启读的Cache

3. 还有更多MPU相关知识可以去看硬汉嵌入式的文档, 参考
<https://www.cnblogs.com/armfly/p/10947503.html>