RMW(Read-Modify-Write)对 STM32F7xx 内核运行速度的影响

前言

在客户使用 STM32F7xx（Cortex-M7 内核）实际测试中，会发现同等主频下 STM32F4xx（Cortex-M4）执行同样一段简单程序在时间上要快于 STM32F7xx，这个会影响到客户切换到 STM32F7xx 的信心，也对 ST 以及 ARM 宣传上 Cortex-M7 内核执行时间远快于 Cortex-M4 内核的说法提出质疑，本文将针对具体案例说明这一情况的产生以及解决办法。

问题描述

客户测试复杂程序运行时间，比如同样 180MHz 主频下，STM32F7xx 执行 Coremark 测试程序时间远小于 STM32F4xx 的执行时间；也就是 STM32F7xx 的性能更佳，运算执行效率更好。但当客户程序顺序执行程序，尤其是简单程序时发现 STM32F7xx 执行时间大于 STM32F4xx 的执行时间，比如运行下面的同样的测试代码，就有明显差距：

```
volatile uint16_t  i;
static volatile uint16_t j = 0;
i = 0;
while(i<300)
{
  i++;
}

if(j < 100)
{
  j ++;
}
else
{
  j = 0;
}
```

为方便量化时间，使用 Timer2 计数方式对这段时间进行计数，Timer2 运行在 90MHz，向上计数，Test_Counter 数据用于输出计数数值，增加后代码如下：

```
volatile uint16_t  i;
static volatile uint16_t j = 0;

TIM2->CNT = 0;
__HAL_TIM_ENABLE(&htim2);

i = 0;
while(i<300)
{
  i++;
}
```

```
if(j < 100)
{
  j ++;
}
else
{
  j = 0;
}

__HAL_TIM_DISABLE(&htim2);
Test_Counter = __HAL_TIM_GET_COUNTER(&htim2);
```

通过上面的修改后测试下来，Test_Counter 数据分别为：

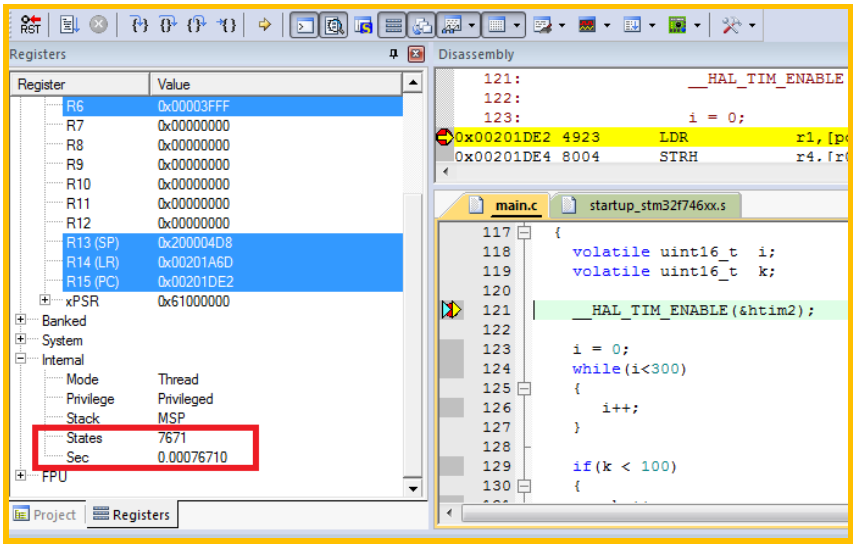
STM32F446 数据为 1543

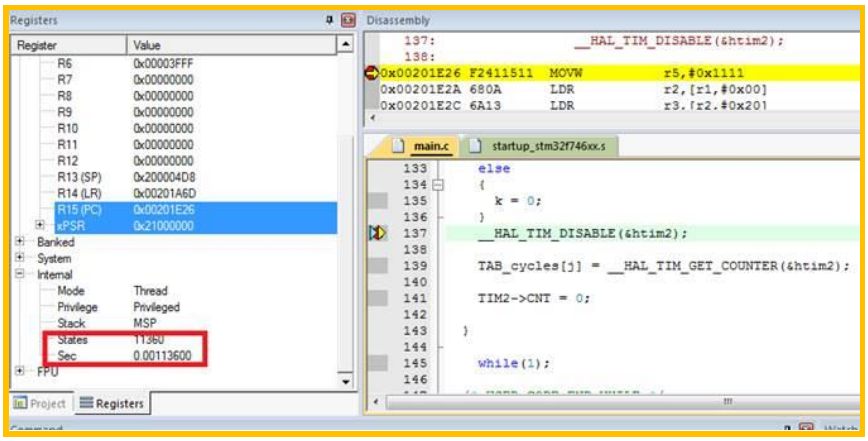
STM32F746 数据为 1836

如果使用 Keil 自带的 States cycles 计算方法得到如下数据，后面会按照这个来计算执行时间数据。

STM32F446 数据为 3009

STM32F746 数据为 3635





产生上述问题的原因：

上面的测试都是在使用了 Cache 以及 ART 加速方法测得，如果针对 STM32F7xx 的性能优化可以参考 AN4667 "STM32F7 Series system architecture and performance"这篇应用文档的描述，本例已经对文档描述部分做过优化，但问题依然是 STM32F7xx 速度慢于 STM32F4xx。两颗芯片运行同样代码，比较两颗芯片汇编代码也是相同的：

```
LDRH        r2,[sp,#0x00]
ADDS        r2,r2,#1
STRH        r2,[sp,#0x00]
LDRH        r2,[sp,#0x00]
CMP         r2,r3
BCC         0x00000128
```

通过查看 ARM Cortex-M7 内核文档发现下面描述：

13.2.5 Performance impact

In an error-free system, the major performance impact is the cost of the read-modify-write scheme for non-full stores in the data side. If a store buffer slot does not contain at least a full 32-bit word, it must read the word to be able to compute the check bits. This can occur because software only writes to an area of memory with byte or halfword store instructions. The data can then be written in the RAM. This additional read can have a negative impact on performance because it prevents the slot from being used for another write.

The buffering and outstanding capabilities of the memory system mask part of the additional read, and it is negligible for most codes. However, it is recommended that you use as few cacheable STRB and STRH instructions as possible to reduce the performance impact.

反映到本例中发现定义的 i 数据为 16-bit 数据，同样也在汇编代码上发现了 STRB 这个汇编代码；这样在 RMW（read-modify-write）机制下，当定义为 byte 以及 half-word 数据时将有一个先读取数据，修改后再写入数据的过程，这个读取-修改-写入的过程正是能够影响到内核执行效率的问题点，如果定义为 32-bit 就避免了这个问题的发生。

问题解决

按照文档说明，我们将 16-bit 定义数据，改为 32-bit 的定义数据，即：

```
volatile uint32_t  i;
static volatile uint16_t j = 0;
```

生成的汇编代码如下：

```
LDR         r0,[sp,#0x00]
ADDS        r0,r0,#1
STR         r0,[sp,#0x00]
CMP         r0,r1
BCC         0x08001F28
```

测试下来结果如下：

STM32F446 数据为 2102

STM32F746 数据为 1807

可以看到不管是 STM32F4xx 还是 STM32F7xx，当数据定义为 32-bit 时都有显著的速度提升，当然 STM32F7xx 的提升更加明显，同样测试条件下 STM32F7xx 执行时间小于 STM32F4xx 的执行时间。

深入内核修改

因为 32-bit 数据定义会增加内存，并且有时候定义为 byte 或 halfword 更方便，还需要提升速度的话我们看到同样是内核文件给出的说明，可以将 RMW 机制屏蔽掉：

Table 3-9 CM7_ITCMCR and CM7_DTCMCR bit assignments

Bits	Name	Type	Function
[31:7]	-	-	Reserved.
[6:3]	SZ	RO	TCM size. Indicates the size of the relevant TCM: 0b0000 No TCM implemented. 0b0011 4KB. 0b0100 8KB. 0b0101 16KB. 0b0110 32KB. 0b0111 64KB. 0b1000 128KB. 0b1001 256KB. 0b1010 512KB. 0b1011 1MB. 0b1100 2MB. 0b1101 4MB. 0b1110 8MB. 0b1111 16MB. All other encodings are reserved. The reset value is derived from the **CFGITCMSZ** and **CFGDTCMSZ** pins.
[2]	RETEN	RW	Retry phase enable. When enabled the processor guarantees to honor the retry output on the corresponding TCM interface, re-executing the instruction which carried out the TCM access. **0** Retry phase disabled. **1** Retry phase enabled. The reset value is derived from the **INITRETRYEN** pin. The retry functionality can be used together with external logic to support error detection and correction in the TCM
[1]	RMW	RW	*Read-Modify-Write* (RMW) enable. Indicates that all writes to TCM, that are not the full width of the TCM RAM, use a RMW sequence: **0** RMW disabled. **1** RMW enabled. The reset value is derived from the **INITRMWEN** pin. The RMW functionality can be used together with external logic to support error detection and correction in the TCM.

实际上就是对 CM7_ITCMCR 寄存器的第 1 位写 0， 即可以在程序中有下面的操作：

```
__IO uint32_t * DTCM_CR = (uint32_t*)(0xE000EF94);
* DTCM_CR &= 0xFFFFFFFD; /* Disable read-modify-write */
```

禁止 RMW 后测试下来数据如下：

16-bit 定义数据 STM32F746 测试 cycles 数据为 3022

32-bit 定义数据 STM32F746 测试 cycles 数据为 1808

可以对比上面的测试数据也可以看到当禁止 RMW 后 STM32F7xx 性能也是优于 STM32F4xx 的。

具体测试数据如下：

Environnement: KEIL 5.20, system-tick used to measure the cycles number

			F756	nb cycles	Comment
i: 16-bit	R-M-W ON	RAM TCM DTCM	O3 size	3635	-> halfword access **with read-modify-write Enabled**
			O3 speed	2736	
	R-M-W OFF	RAM TCM DTCM	O3 size	3022	-> halfword access **with read-modify-write Disabled**
			O3 speed	2137	
i: 32-bit	R-M-W ON	RAM TCM DTCM	O3 size	1807	-> **Word access**. Regardless if read-modify-write is enabled or disabled, the performance is the same for the two configs and better than F4
			O3 speed	470	
	R-M-W OFF	RAM TCM DTCM	O3 size	1808	
			O3 speed	467	

	F446	nb cycles
i: 16-bit	O3 size	3009
	O3 speed	2577
i: 32-bit	O3 size	2102
	O3 speed	1355

结论：

需要提升 STM32F7xx 执行时间，发挥出最大效能时，请参考 AN4667，同时需要注意 RMW 对内核性能发挥的影响。

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。