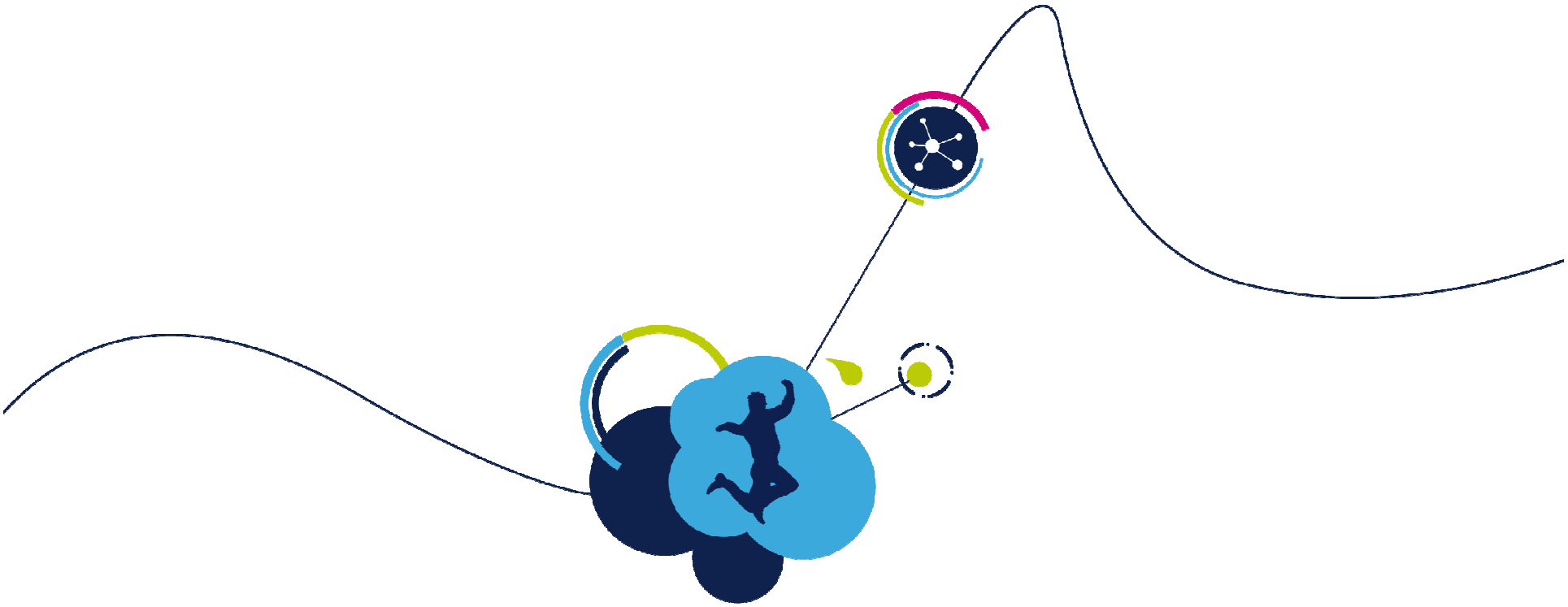


高速缓存Cache

life.augmented

高速缓存(Cache)介绍

- 什么是高速缓存？
  - 高速存储器块，包含地址信息（通常称作TAG）和相关联的数据。
  - 目的是提高对存储器的平均访问速度
- 高速缓存的应用基于下面两个程序的局部性：
  - 空间局部性：如果一个存储器的位置被访问，那么将来它附近的位置也会被访问。比如顺序执行代码，或者使用一个数据结构
  - 时间局部性：被访问过一次的存储器位置，接下来会被多次引用。比如：循环
- 缓存行（cache line)
  - 逻辑上的一组存储器位置。内存交换数据的最小粒度
- 缓存命中（cache hit)，要访问的数据/指令已经存在缓存里
- 缓存缺失（cache miss)，要访问的数据/指令不在缓存里

高速缓存介绍

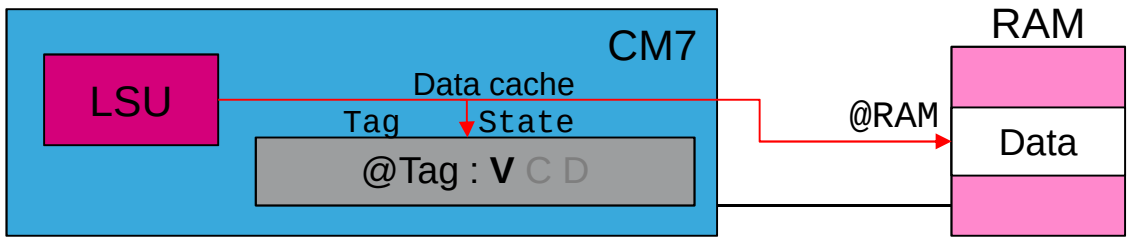
- 处理器需要访问某个可缓存的存储器位置时，先检查缓存内是否已经存在该位置的内容
- 如果缓存命中，则直接从缓存读出。如果缓存缺失，则从存储器里读出，同时放入缓存。
- 缓存分配（Cache Allocation)
  - 缓存缺失（Cache miss）时，在缓存中发现一个位置，并把新的缓存数据存到这个位置。
  - 读分配（Read-allocate）：在进行读操作发生缓存缺失时，进行缓存分配。所有可缓存的存储器都是读分配。
  - 写分配（Write-allocate)：在进行写操作发生缓存缺失时，进行缓存分配。
- 高速缓存的应用带来一致性问题
  - 程序员不能控制对存储器的访问时机
  - 同一个数据被保存在多个物理位置

Cache术语

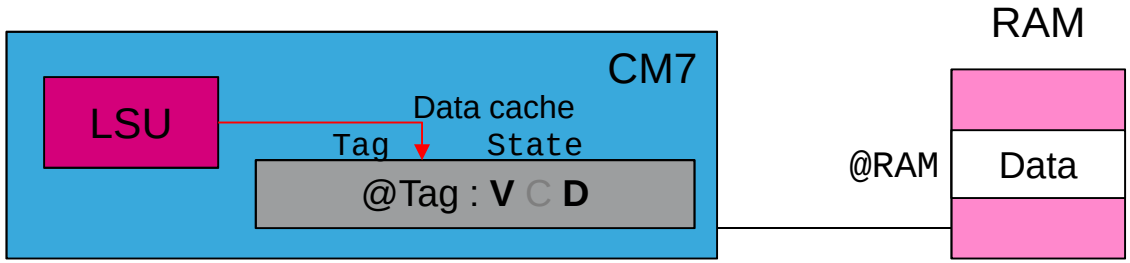
- 驱逐（Eviction）:
  - 从缓存中移除一个缓存行（cache line)，为新的数据腾位置的过程Write of an entire cache line on the bus
  - 发生于当一个“dirty”的缓存行被新的缓存行替代时
  - Dirty是标记那些需要写回到存储器中的缓存数据。
- 回写(Write-back):
  - 写数据时，只更新缓存，然后将缓存行标记为“dirty”
  - 当这个缓存行被新的缓存行替换时，再将数据写到存储器中
- 透写(Write through):
  - 写数据时同时更新缓存和二级存储
  - 缓存行不被标记为“dirty”

缓存策略

- 内部L1缓存策略描述:
  1. 透写 (读或者写分配)
     - 数据同时写到缓存和下一级存储器中



  2. 回写（读或者写分配）
     - 数据只写到缓存



life.augmented

Cortex-M7高速缓存

- 采用哈佛结构，实现最优性能
- 指令缓存和数据缓存都是可选的，分开配置大小(STM32F7中各4kB)
- 只在AXIM接口有缓存 (TCMs和AHBP没有)
- 两路相联指令高速缓存, 四路相联数据高速缓存
- 对ARMv7E-M系统架构的要求
  - 增加Cache维护操作，增加新的相关寄存器.
  - 使用Cache会对系统软件有影响

Cortex-M7高速缓存

- 全面支持下列缓存属性
  - 透写(Write Through), 没有写分配(write allocate)
  - WBRA:回写(Write-Back),没有写分配(WBRA)
  - WBWA:回写(Write-Back), 写分配(write allocate)
- 回写方式有利于优化性能
- 没有对一致性的硬件支持
  - 共享内存不支持缓存，以保持和二级存储的一致性.
- 所有的cache ECC都在内部进行（STM32F7不支持ECC）

Cache的一致性

- 没有对一致性的硬件支持
- 两种可选方案
  - 所有的共享存储器都定义为共享属性
    - 这些区域将默认不被缓存到D-Cache
    - 所有的操作都直接针对2级存储器（内部Flash，外部存储器），性能降低
    - 因为缓存对这些区域是透明的，写软件更容易
  - 通过软件进行cache的维护
    - Cortex-M7 的写操作要是全局可见的
      - 使用透写属性（通过MPU设置）
      - 使用SIWT@CACR (Shared = Write Through)
      - 通过指令清D-cache，然后所有更新位置禁止D-Cache操作
    - 其他主设备的写操作要对Cortex-M7可见
      - 比如作废Cortex-M7 Dache中数据.

缓存和存储器属性

- 存储器的属性由MPU进行设定
- 这些属性同样也对AXIM/Cache接口上的存储器系统有影响
- 属性
  - 共享
    - 仅用于当多个master共享同一块存储空间时 (一致性保证)
    - 默认**对数据的存取不在D-Cache里做缓存**
    - **对I-cache没有影响**– 共享的区域还是会被缓存
  - 分配策略
    - 写分配- 整体性能最佳
    - 读分配 – 具体例子 (e.g memset())
    - 透写 – 性能上比回写低，但对安全关键代码很有用
    - 对 I-Cache而言, 所有的分配策略都被当做读分配 (没有写指令的操作)
  - 存储器类型
    - Normal – 可以被缓存 (peripheral on cacheability/shared attributes)
    - DEV/SO – 仅用于Devices- 不能被缓存.

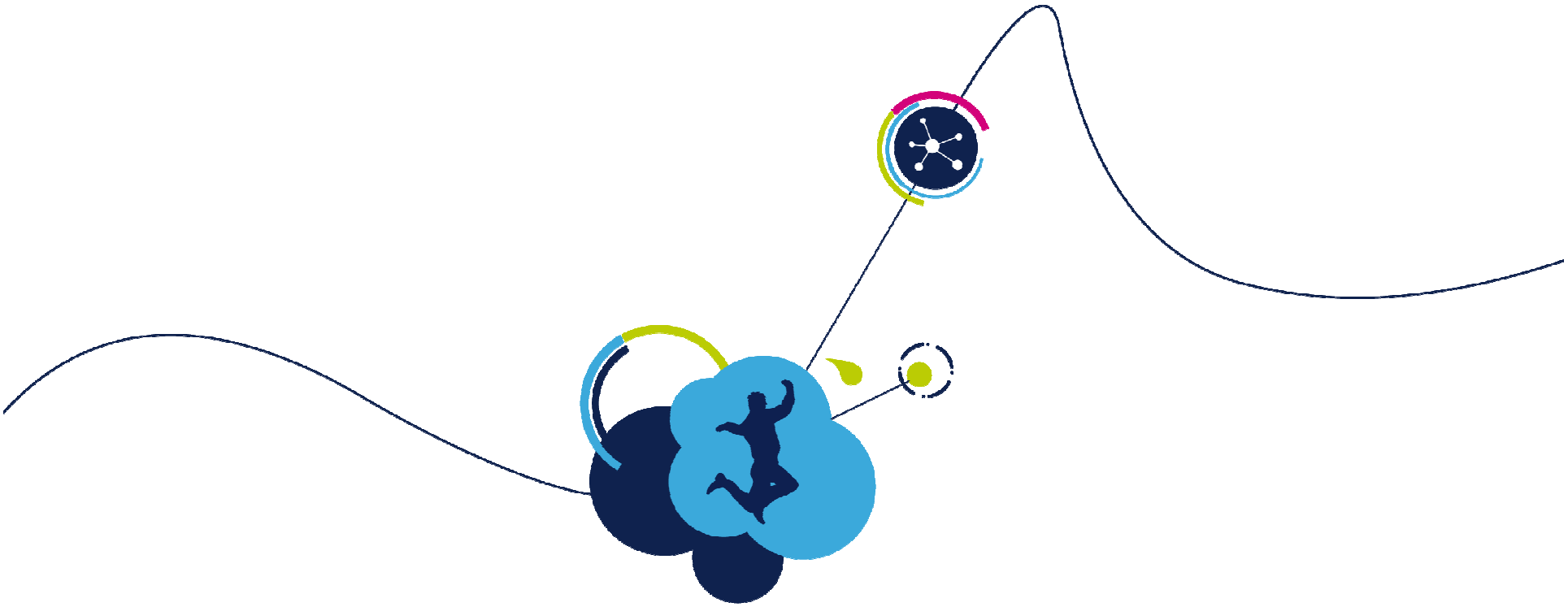
存储器默认映射和属性

地址范围	名称	存储器类型	XN	Cache	描述
0x0000 0000 0x1FFF FFFF	Code	Normal		WT	Typically ROM or flash memory. Memory required from address 0x0 to support the vector table for system boot code on reset
0x2000 0000 0x3FFF FFFF	SRAM	Normal	-	WBWA	SRAM region typically used for on-chip RAM
0x4000 0000 0x5FFF FFFF	Peripheral	Device	XN	-	On chip peripheral address space
0x6000 0000 0x7FFF FFFF	RAM	Normal	-	WBWA	Memory with write-back, write allocate cache attribute for L2/L3 cache support
0x8000 0000 0x9FFF FFFF	RAM	Normal	-	WT	Memory with write-through cache attribute
0xA000 0000 0xBFFF FFFF	Device	Device, Shareable	XN	-	Shared device space
0xC000 0000 0xDFFF FFFF	Device	Device, non-shareable	XN	-	Non-shared device space
0xE000 0000 0xE00F FFFF	PPB	Strongly-Ordered	XN	-	1MB region reserved as the PPB. This supports key resources, including the System Control Space and debug features.
0xE010 0000 0xFFFF FFFF	Vendor_SYS	Device	XN	-	Vendor system region

life.augmented

初始化和使能L1 Caches

- 上电（power-on)复位时，在使能前，I-cache和D-cache必须全部被作废（invalidate)
  - Cache行的标签(Tag)中的valid位
  - 如果不这样做，可能会引发程序不可预测的行为
  - 注意：软复位时，如果复位前RAM里的值是可靠的，可以不用做这一步
- I-Cache:
  - 通过 ICIALLU 寄存器来作废整个I-Cache
- D-Cache:
  - 建议遍历整个D-Cache并作废
- 为了保证数据的一致性，必须在除能D-cache之前对其进行清理
  - 仅在使用回写策略时需要
  - 如果不这么做就可能会丢失数据

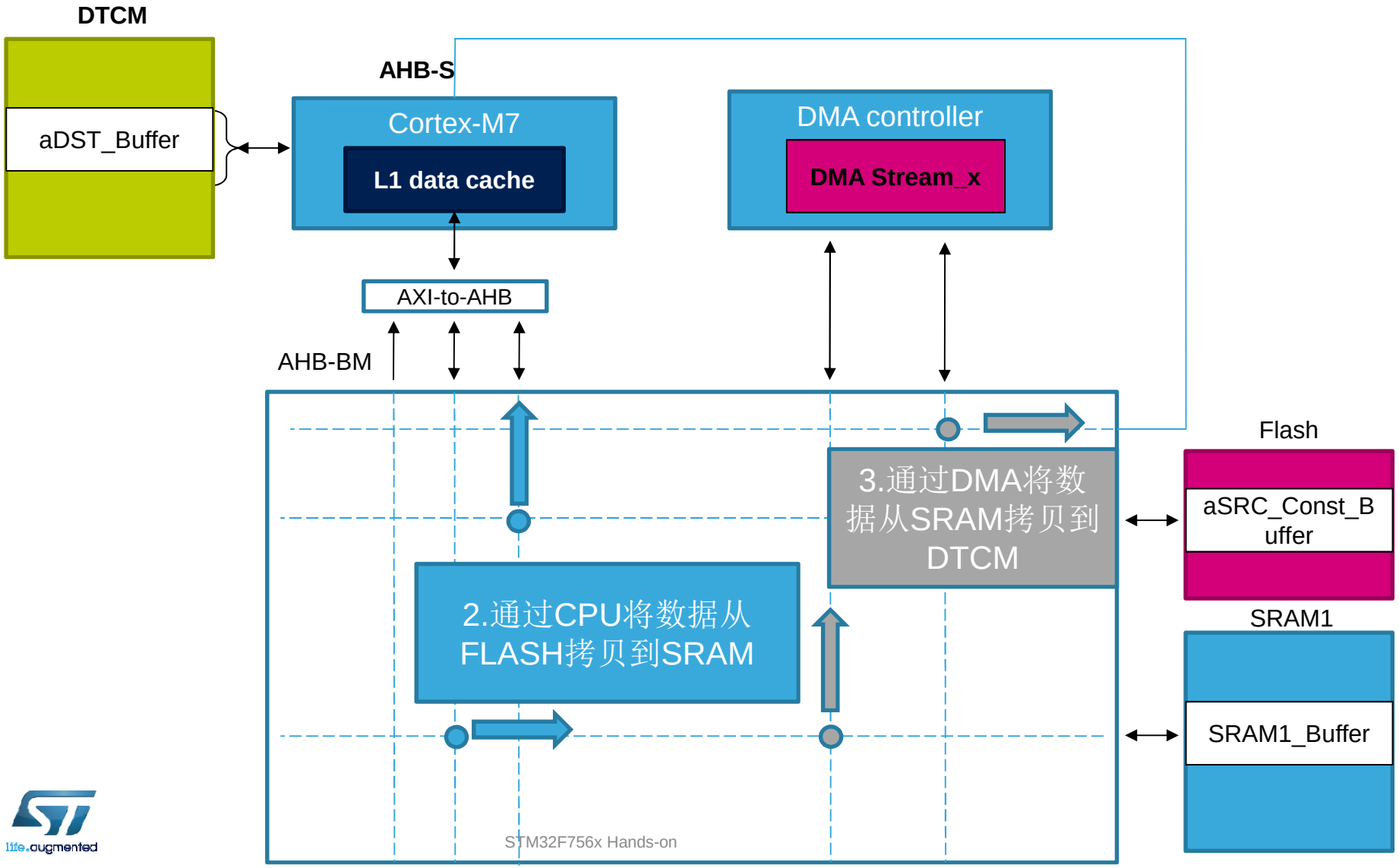


Cache一致性举例

程序描述

- 1.首先将地址0x20010000（SRAM1）处开始的128字节初始化为0x55.
- 2.将Flash中的128字节的常量数组aSRC_Const_Buffer拷贝到SRAM1地址0x20010000 (pBuffer).
- 3.配置并使能DMA，通过DMA将数据从SRAM1的地址0x20010000处拷贝到DTCM RAM中的数组 aDST_Buffer中.
  - DMA传输完成后，LED1 亮.
- 4.将Flash中的数组aSRC_Const_Buffer与DMA读出的数组aDST_Buffer进行比较
  - 如果不一样则LED2 亮.
  - 一样则LED3 亮

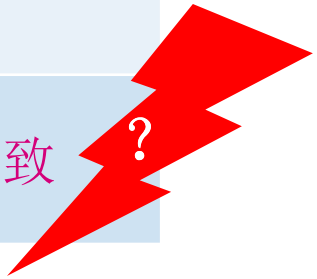
数据传输路径


DTCM
aDST_Buffer
AHB-S
Cortex-M7
L1 data cache
DMA controller
DMA Stream_x
AXI-to-AHB
AHB-BM
3.通过DMA将数据从SRAM拷贝到DTCM
2.通过CPU将数据从FLASH拷贝到SRAM
Flash
aSRC_Const_Buffer
SRAM1
SRAM1_Buffer

实验1

Cortex-M7 数据一致性– Cache关 vs Cache开

Cache 关	Cache 开
默认关	/* Enable D-Cache */ SCB_EnableDCache();
LED1 &LED3 亮	LED1 &LED2 亮
Flash与DTCM的数据一致	Flash与DTCM的数据不一致



CMSIS定义了开关Cache的函数：
SCB_EnableDCache（）， SCB_EnableICache （）

Cortex-M7 数据一致性——执行清Cache操作

实验2

- 使能D-cache

使能D-Cache

```
/* Fill 128 bytes with 0x55 pattern */
memset((uint8_t*)SRAM1_ADDRESS_START, 0x55, sizeof(aSRC_Const_Buffer));

/* Step-1 : Enable Data cache (uncomment following line to enable Data Cache */
/* Enable Data cache */
 SCB_EnableDCache();
```

- 在对可缓存区域进行写操作后，执行清Cache的操作。

2.通过CPU将数据从FLASH拷贝到SRAM

```
for (counter = 0; counter < (sizeof(aSRC_Const_Buffer)/4); counter++)
{
  *pBuffer++ = aSRC_Const_Buffer[counter];
}
```

```
/* Step-2 : Perform cache clean operation by software (uncomment following line to
 * ensure coherency by software) */
```

执行清Cache操作

```
/* Clean Data Cache */
SCB_CleanDCache();
```

- 所有的dirty line都被回写到SRAM1， 保证了cache和二级存储之间数据的一致性
- LED1 &LED3灯亮
- Flash与DTCM中的数据一致

Cortex-M7 数据一致性——透写属性

实验3

- 使能MPU，修改SRAM1的MPU属性从回写（write-back)改为透写（write-through）。

1. 在打开D-Cache之前把MPU配置好

2. MPU要配置为Write-Through透写模式

```
/* Fill 128 bytes with 0x55 pattern */
memset((uint8_t*)SRAM1_ADDRESS_START, 0x55, sizeof(aSRC_Const_Buffer));


/* Step-3 : Enable MPU and change SRAM region attribute (uncomment following line to
* set write-through policy on SRAM) */
 MPU_Config();

/* Step-1 : Enable Data cache (uncomment following line to enable Data Cache */
/* Enable Data cache */
 SCB_EnableDCache();

/* Step-4 : Force all CPU accesses to be write-through (uncomment following line to
* set all CPU accesses to cachable region as write-through) */

 /*__FORCE_WRITE_THROUGH(); */

for (counter = 0; counter < (sizeof(aSRC_Const_Buffer)/4); counter++)
{
      *pBuffer++ = aSRC_Const_Buffer[counter];
}
```

- 所有的dirty line都被回写到SRAM1， 保证了cache和二级存储之间数据的一致性
- Flash与DTCM中的数据一致，LED1 &LED3灯亮

Cortex-M7 数据一致性——SIWT@CACR

实验4

- 使能MPU，SRAM1的MPU属性设置成回写（write-back)。

1. 在打开D-Cache之前把设置SIWT位

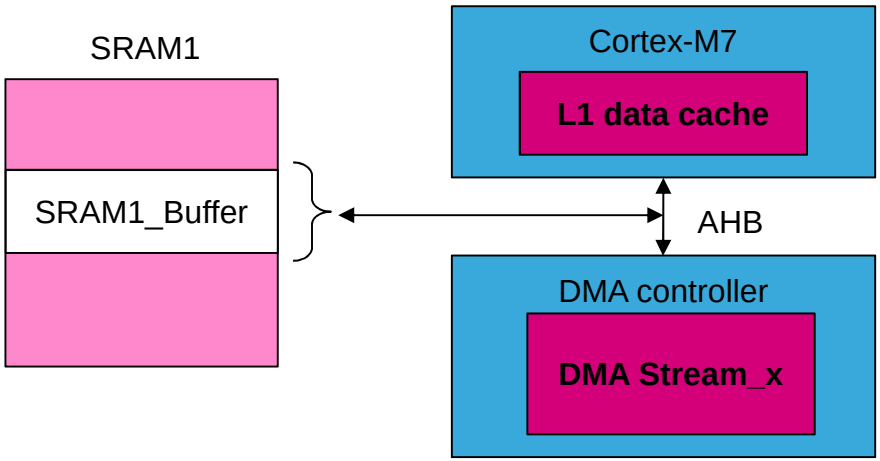
2.不管 MPU如何配置，都按照Write-Through透写模式对待

```
/* Fill 128 bytes with 0x55 pattern */
memset((uint8_t*)SRAM1_ADDRESS_START, 0x55, sizeof(aSRC_Const_Buffer));


/* Step-3 : Enable MPU and change SRAM region attribute */
 MPU_Config();


/* Step-4 : Force all CPU accesses to be write-through (uncomment following line to
* set all CPU accesses to cachable region as write-through) */

 __FORCE_WRITE_THROUGH();

/* Step-1 : Enable Data cache (uncomment following line to enable Data Cache */
/* Enable Data cache */
 SCB_EnableDCache();

for (counter = 0; counter < (sizeof(aSRC_Const_Buffer)/4); counter++)
{
      *pBuffer++ = aSRC_Const_Buffer[counter];
}
```

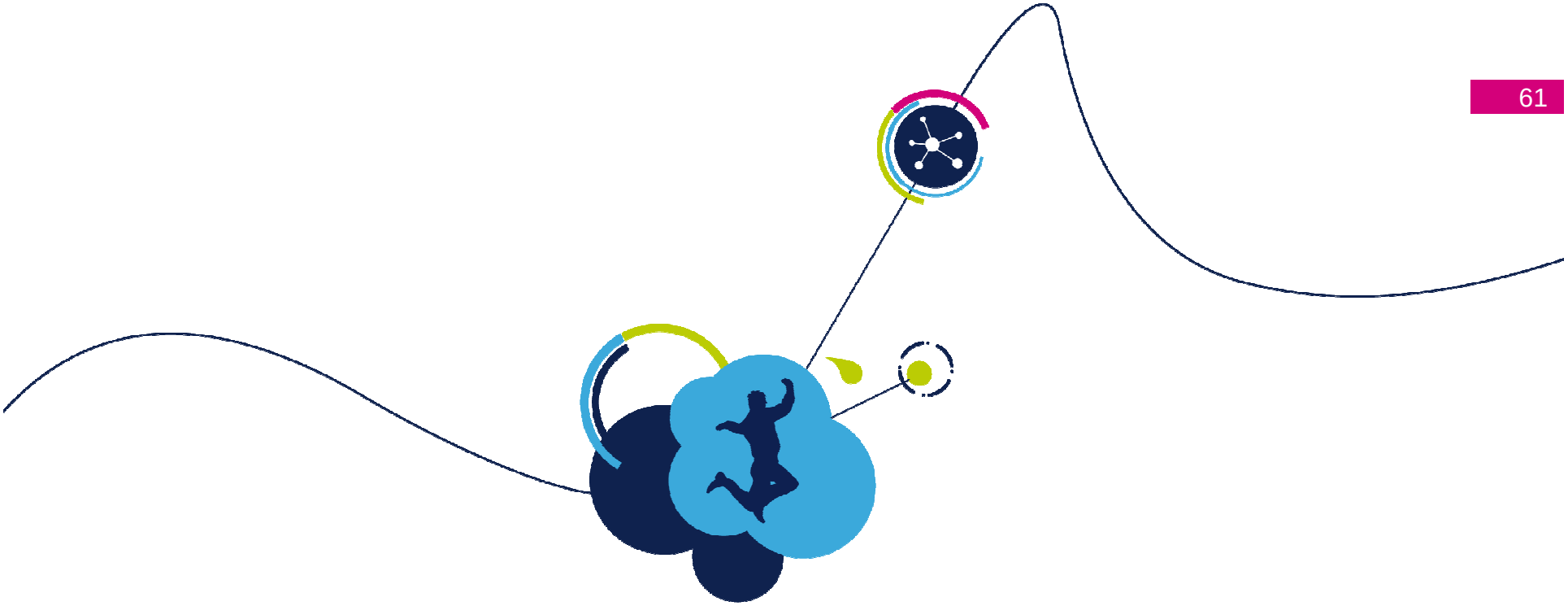
- 所有的dirty line都被回写到SRAM1， 保证了cache和二级存储之间数据的一致性
- Flash与DTCM中的数据一致，LED1 &LED3灯亮

Cortex-M7与DMA之间的数据一致性

- 前面的例子中，Cortex-M7内核与DMA共享SRAM1中的一个数据buffer:

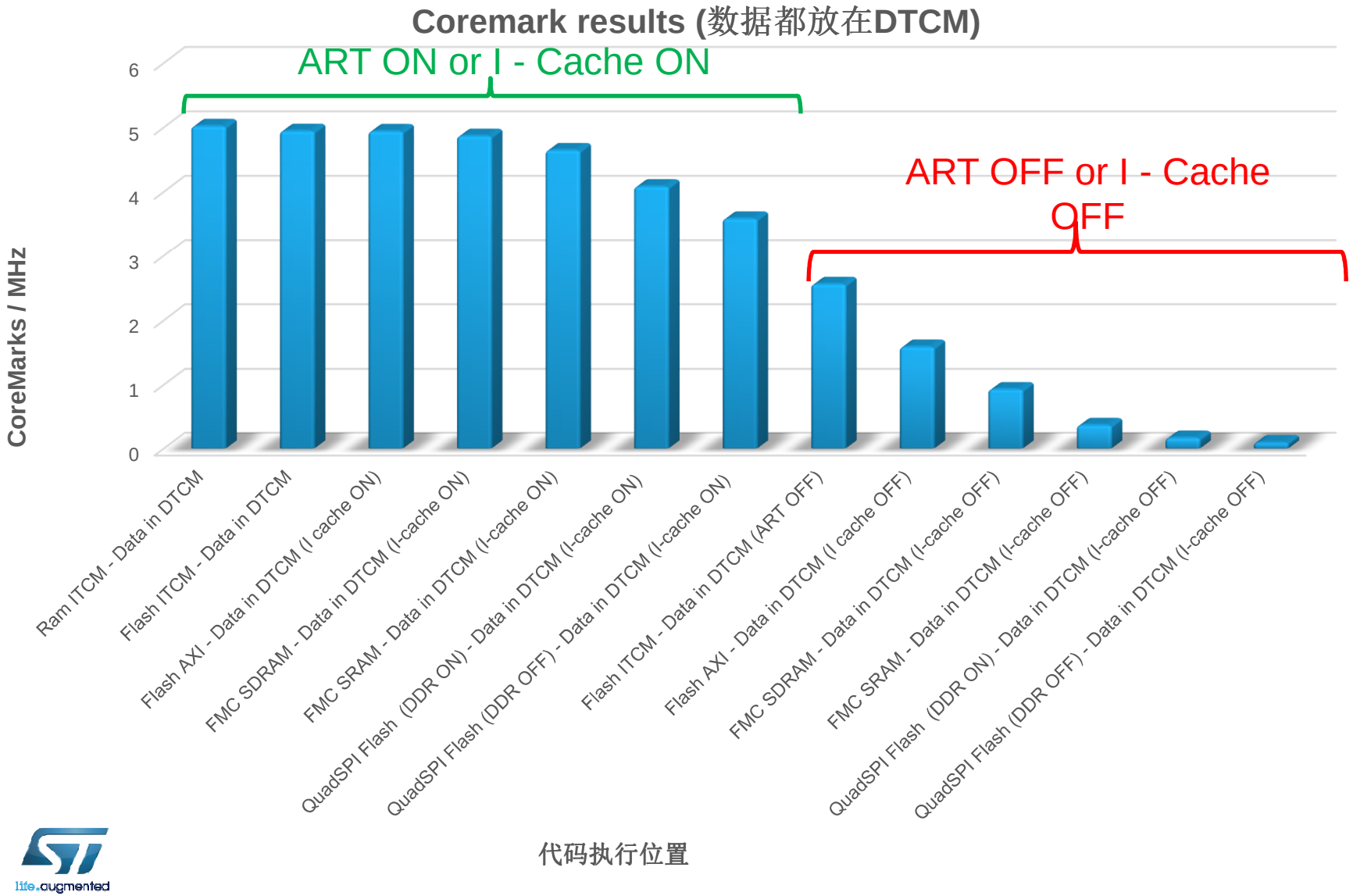


- 内核与DMA之间的数据一致性通过下面3种方法得到了保证:
  1. 通过置位SIWT@CACR 禁止对SRAM1进行cache
  2. 通过软件关闭D-Cache功能，或者及时进行清Cache的操作，将数据回写到存储器中
  3. 打开Cache功能，当把SRAM1设置为透写属性
  - 在这个例子中DMA的目标buffer不能被缓存（在DTCM中），所以没有数据一致性的问题。但数据不一致的问题同样有可能发生在DMA的目标buffer上，如果它是一个可被缓存的共享buffer的话.



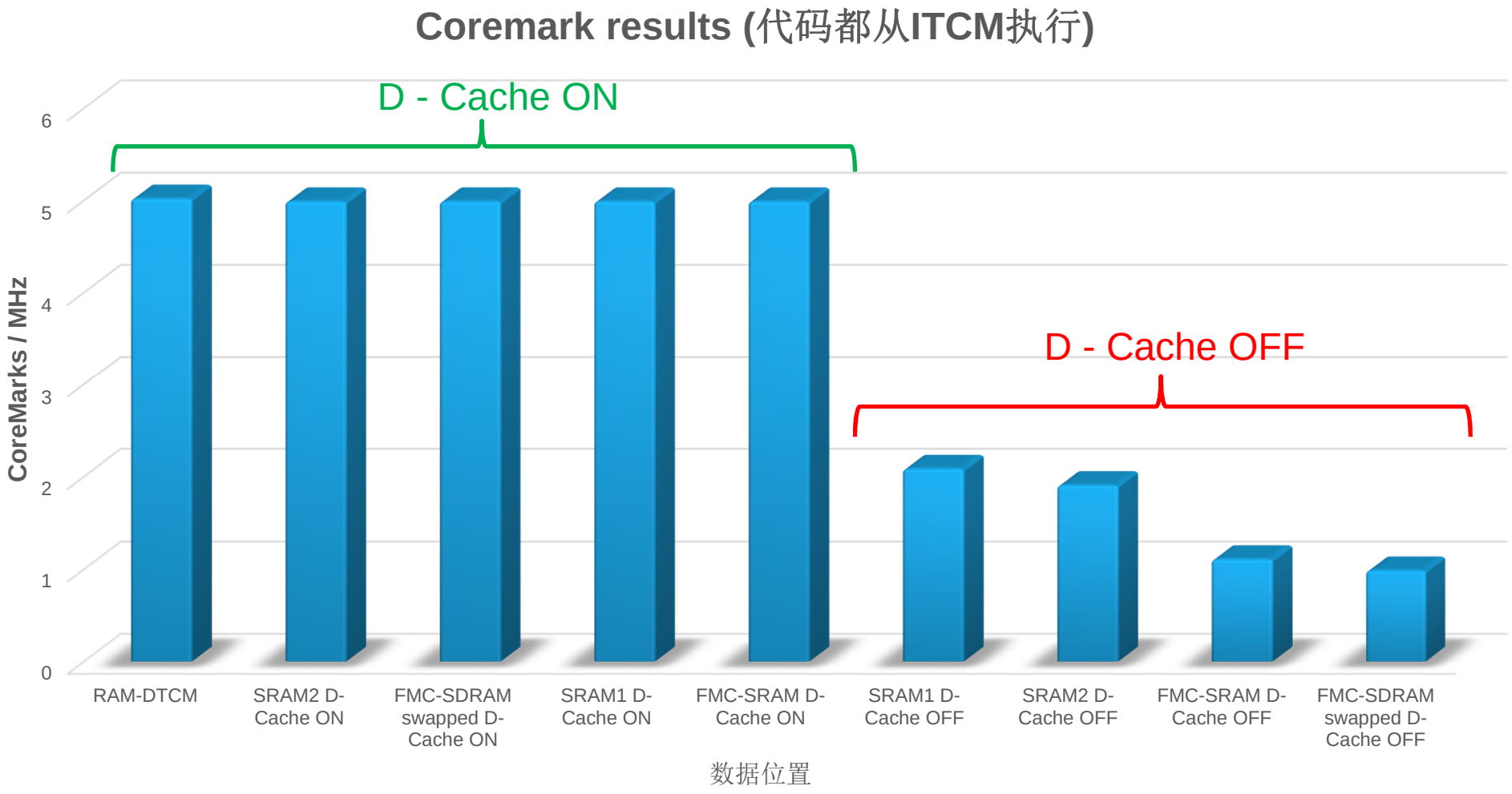
STM32F756x 性能

Coremark 基准

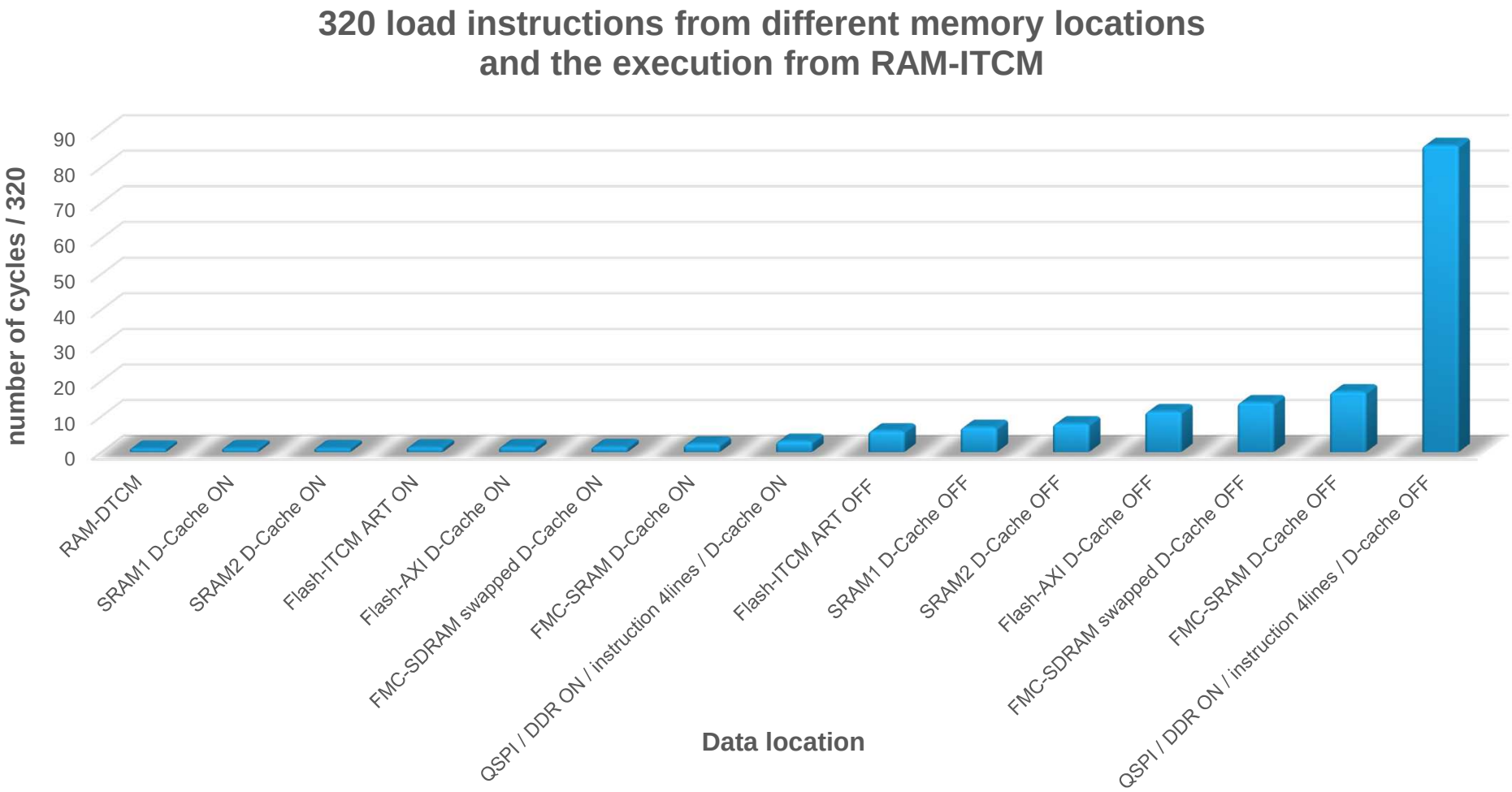


life.augmented

Coremark 基准


Coremark results (代码都从ITCM执行)
D - Cache ON
D - Cache OFF
CoreMarks / MHz
0
1
2
3
4
5
6
RAM-DTCM
SRAM2 D-Cache ON
FMC-SDRAM swapped D-Cache ON
SRAM1 D-Cache ON
FMC-SRAM D-Cache ON
SRAM1 D-Cache OFF
SRAM2 D-Cache OFF
FMC-SRAM D-Cache OFF
FMC-SDRAM swapped D-Cache OFF
数据位置

数据加载性能



life.augmented

CONFIDENTIALITY OBLIGATIONS:

THIS DOCUMENT CONTAINS SENSITIVE INFORMATION.

IT IS CLASSIFIED “MICROCONTROLLERS, MEMORIES & SECURE MCUs (MMS) RESTRICTED AND ITS DISTRIBUTION IS SUBMITTED TO ST/MMS AUTHORIZATION

AT ALL TIMES YOU SHOULD COMPLY WITH THE FOLLOWING SECURITY RULES:

- DO NOT COPY OR REPRODUCE ALL OR PART OF THIS DOCUMENT
- KEEP THIS DOCUMENT LOCKED AWAY
- FURTHER COPIES CAN BE PROVIDED ON A “NEED TO KNOW BASIS”, PLEASE CONTACT YOUR LOCAL ST SALES OFFICE OR DOCUMENT WRITTER.

Information furnished is believed to be accurate and reliable. However, STMicroelectronics assumes no responsibility for the consequences of use of such information nor for any infringement of patents or other rights of third parties which may result from its use. No license is granted by implication or otherwise under any patent or patent rights of STMicroelectronics. Specifications mentioned in this publication are subject to change without notice. This publication supersedes and replaces all information previously supplied. STMicroelectronics products are not authorized for use as critical components in life support devices or systems without express written approval of STMicroelectronics.

The ST logo is registered trademark of STMicroelectronics

All other names are the property of their respective owners

© 2015 STMicroelectronics - All Rights Reserved

STMicroelectronics group of companies Australia - Brazil - Canada - China - Finland - France - Germany - Hong Kong - India - Israel - Italy - Japan - Malaysia - Malta - Morocco - Singapore - Spain - Sweden - Switzerland - United Kingdom - United States.

www.st.com

life.augmented