

DWT 是什么

使用

DWT相关寄存器

DEMCR

DWT_CYCCNT

CYCCNTENA

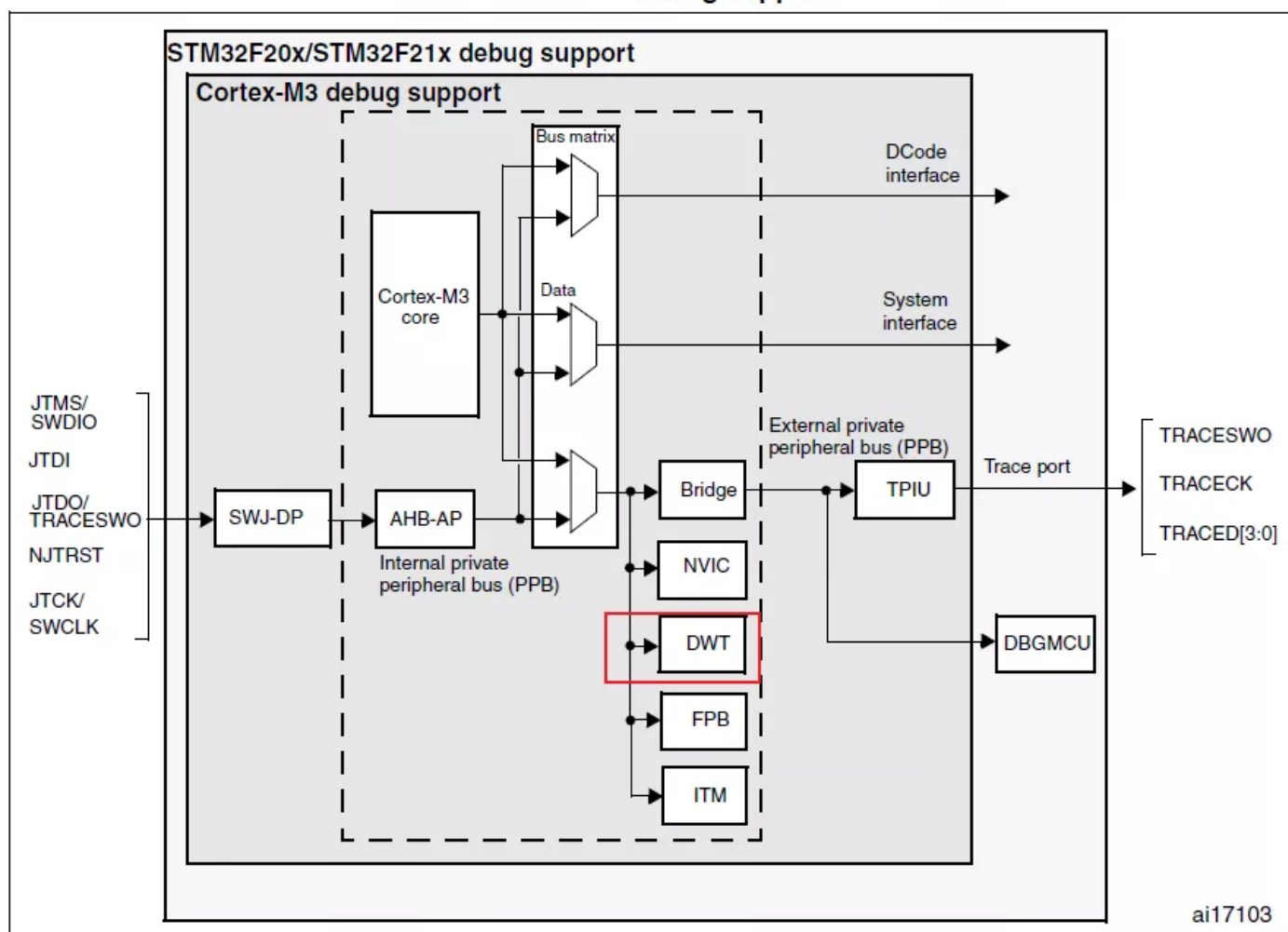
相关实现函数（HAL库）

参考

DWT 是什么

在Cortex-M里面有一个外设叫DWT(DataWatchpoint andTrace)，是用于系统调试及跟踪，DWT的中文名字应该是：数据观察点触发。在STM32用户手册的第32章节Debugsupport (DBG)有如下框图。

Figure 417. Block diagram of STM32 MCU and Cortex®-M3-level debug support



ai17103

33.13 DWT（数据观察点触发）

DWT 单元由四个比较器组成。这些比较器可配置为：

- 硬件观察点或
- ETM 的触发或
- PC 采样器或
- 数据地址采样器

DWT 还能以某种方式提供一些概要信息。为此，可访问某些计数器以获得以下数据：

- 时钟周期
- 分支指令
- 存取单元操作
- 睡眠周期
- CPI（每条指令的执行时间）
- 中断开销

它有一个32位的寄存器叫CYCCNT，它是一个向上的计数器，记录的是内核时钟运行的个数，内核时钟跳动一次，该计数器就加1，精度非常高，决定内核的频率是多少。

例如F103系列，内核时钟是72M，那精度就是 $1/72M = 14ns$ ，而程序的运行时间都是微秒级别的，所以14ns的精度是远远够的。最长能记录的时间为： $60s=2$ 的32次方/72000000(假设内核频率为72M，内核跳一次的时间大概为 $1/72M=14ns$)，。

而如果是H7这种400M主频的芯片，那它的计时精度高达2.5ns ($1/400000000 = 2.5$)，而如果是 i.MX RT1052这种高速的处理器，最长能记录的时间为： $8.13s=2$ 的32次方/528000000 (假设内核频率为528M，内核跳一次的时间大概为 $1/528M=1.9ns$)。当CYCCNT溢出之后，会清0重新开始向上计数。

33.14.2 时间戳数据包、同步和溢出数据包

时间戳数据包会对时间戳信息、通用控制和同步进行编码。它使用 21 位时间戳计数器（带可能的预分频器），该计数器在每次发送时间戳数据包时复位。此计数器可由 CPU 时钟或 SWV 时钟驱动。

同步数据包由 6 个字节组成，等于 0x80_00_00_00_00_00，以 00 00 00 00 00 80 形式（首先发送 LSB）发送到 TPIU。

同步数据包是时间戳数据包的控制信号。它在每次触发 DWT 时发送。

为此，DWT 必须配置为触发 ITM：必须将 DWT 控制寄存器的位 CYCCNTENA（位 0）置 1。此外，还必须将 ITM 跟踪控制寄存器的位 2 (SYNCENA) 置 1。

注意：如果未将 SYNENA 位置 1，DWT 产生给 TPIU 的同步触发，将仅发送 TPIU 同步数据包，不发送 ITM 同步数据包。

溢出数据包是一种特殊的时间戳数据包，用于指示已写入数据但 FIFO 已满。

表 233. 主要的 ITM 寄存器





地址	寄存器	详细信息
@E0000FB0	ITM 锁定访问	写入 0xC5ACCE55 以解锁对其它 ITM 寄存器的写访问
@E0000E80	ITM 跟踪控制	位 31-24 = 始终为 0
		位 23 = 忙碌
		位 22-16 = 7 位 ATB ID，用于标识跟踪数据源
		位 15-10 = 始终为 0
		位 9:8 = TSPrescale = 时间戳预分频器
		位 7-5 = 保留
		位 4 = SWOENA = 使能 SWV 行为（由 SWV 时钟驱动时间戳计数器）
		位 3 = DWTENA: 使能 DWT 激励
		位 2 = SYNCENA: 此位必须置为 1 以使能 DWT，进而生成同步触发，这样 TPIU 便可发送同步数据包
		位 1 = TSENA（时间戳使能）
@E0000E40	ITM 跟踪特权	位 0 = ITMENA: ITM 的全局使能位
		位 3: 置 1 以使能跟踪端口 31:24
		位 2: 置 1 以使能跟踪端口 23:16
		位 1: 置 1 以使能跟踪端口 15:8
		位 0: 置 1 以使能跟踪端口 7:0

使用

想要使用DWT的CYCCNT步骤：

1. 先使能DWT外设，这个由另外内核调试寄存器DEMCR的位24控制，写1使能
2. 使能CYCCNT寄存器之前，先清0。
3. 使能CYCCNT寄存器，这个由DWT的CYCCNTENA 控制，也就是DWT控制寄存器的位0控制，写1使能

要实现延时的功能，总共涉及到三个寄存器：DEMCR、DWT_CTRL、DWT_CYCCNT，分别用于开启DWT功能、开启CYCCNT及获得系统时钟计数值。

-  ttf1100: 使用DWT 是否要修改SYSTICK 3 月前 [回复](#) ... 
-  小葛学飞控 回复： 不需要，这个代码m3 m4 m7 内核是通用的，移植到不同的单片机上只要让dwt时钟频率等于系统时钟就行，因为这个是内核资源不是单片机资源，更换单片机并不影响内核
3 天前 [回复](#) ... 

DWT相关寄存器

DEMCR

想要使能DWT外设，需要由另外的内核调试寄存器DEMCR的位24控制，写1使能（划重点啦，要考试！！）。

DEMCR的地址是0xE000 EDFC

表 15.2 调试及监视器控制寄存器 DEMCR (地址：0xE000_EDFC)

位段	名称	类型	复位值	描述
24	TRCENA	RW	0*	跟踪系统使能位。在使用 DWT, ETM, ITM 和 TPIU 前，必须先设置此位
23:20	保留			
19	MON_REQ	RW	0	1=调试监视器异常不是由硬件调试事件触发，而是由软件手工悬起的
18	MON_STEP	RW	0	让处理器单步执行，在 MON_EN=1 时有效
17	MON_PEND	RW	0	悬起监视器异常请求，内核将在优先级允许时响应
16	MON_EN	RW	0	使能调试监视器异常
15:11	保留			
10	VC_HARDERR	RW	0*	发生硬 fault 时停机调试
9	VC_INTERR	RW	0*	指令/异常服务错误时停机调试
8	VC_BUSERR	RW	0*	发生总线 fault 时停机调试
7	VC_STATERR	RW	0*	发生用法 fault 时停机调试
6	VC_CHKERR	RW	0*	发生用法 fault 使能的检查错误时停机调试（如未对齐，除数为零）
5	VC_NOCERR	RW	0*	发生用法 fault 之无处理器错误时停机调试
4	VC_MMERR	RW	0*	发生存储器管理 fault 时停机调试
3:1	保留			

<https://blog.csdn.net/jiejie MCU>

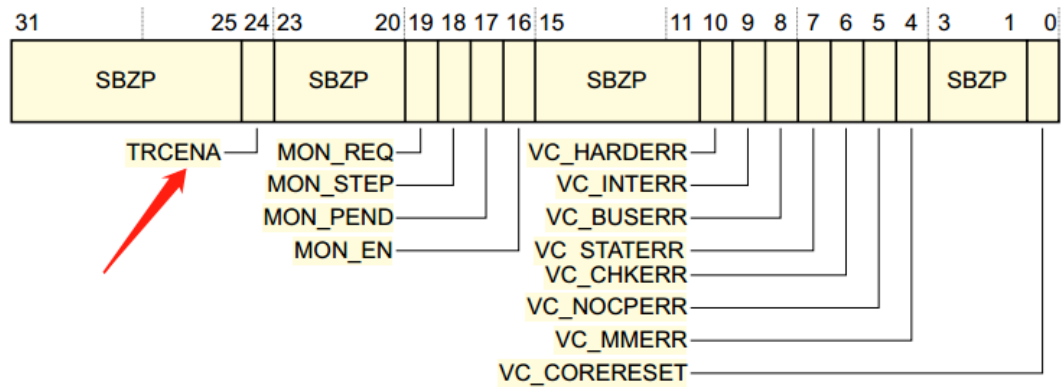


Figure 10-3 Debug Exception and Monitor Control Register bit assignments

Table 10-4 shows the bit functions of the Debug Exception and Monitor Control Register.

Table 10-4 Debug Exception and Monitor Control Register

Bits	Type	Field	Function
[31:25]	-	-	Reserved, SBZP
[24]	Read/write	TRCENA	<p>This bit must be set to 1 to enable use of the trace and debug blocks:</p> <ul style="list-style-type: none"> • <i>Data Watchpoint and Trace (DWT)</i> • <i>Instrumentation Trace Macrocell (ITM)</i> • <i>Embedded Trace Macrocell (ETM)</i> • <i>Trace Port Interface Unit (TPIU).</i> <p>This enables control of power usage unless tracing is required. The application can enable this, for ITM use, or use by a debugger.</p>

Note

If no debug or trace components are present in the implementation then it is not possible to set TRCENA.

<https://blog.csdn.net/fiejiecmu>

DWT_CYCCNT

使能DWT_CYCCNT寄存器之前，先清0。

其基地址是**0xE0001004**，复位默认值是0，可读写类型。所以往**0xE0001004**这个地址写就将DWT_CYCCNT清0了。

```

/* core_cm4.h 中的代码 */
/**
 \brief Structure type to access the Data Watchpoint and Trace Register
 (DWT).
 */
typedef struct
{
    __IOM uint32_t CTRL;           /*!< Offset: 0x000 (R/W) Control
Register */
    __IOM uint32_t CYCCNT;        /*!< Offset: 0x004 (R/W) Cycle
Count Register */

```

__IOM uint32_t CPICNT;	/*!< Offset: 0x008 (R/W) CPI Count
Register */	
__IOM uint32_t EXCCNT;	/*!< Offset: 0x00C (R/W) Exception
Overhead Count Register */	
__IOM uint32_t SLEEPcnt;	/*!< Offset: 0x010 (R/W) Sleep
Count Register */	
__IOM uint32_t LSUCNT;	/*!< Offset: 0x014 (R/W) LSU Count
Register */	
__IOM uint32_t FOLDcnt;	/*!< Offset: 0x018 (R/W) Folded-
instruction Count Register */	
__IM uint32_t PCSR;	/*!< Offset: 0x01C (R/) Program
Counter Sample Register */	
__IOM uint32_t COMP0;	/*!< Offset: 0x020 (R/W) Comparator
Register 0 */	
__IOM uint32_t MASK0;	/*!< Offset: 0x024 (R/W) Mask
Register 0 */	
__IOM uint32_t FUNCTION0;	/*!< Offset: 0x028 (R/W) Function
Register 0 */	
uint32_t RESERVED0[1U];	
__IOM uint32_t COMP1;	/*!< Offset: 0x030 (R/W) Comparator
Register 1 */	
__IOM uint32_t MASK1;	/*!< Offset: 0x034 (R/W) Mask
Register 1 */	
__IOM uint32_t FUNCTION1;	/*!< Offset: 0x038 (R/W) Function
Register 1 */	
uint32_t RESERVED1[1U];	
__IOM uint32_t COMP2;	/*!< Offset: 0x040 (R/W) Comparator
Register 2 */	
__IOM uint32_t MASK2;	/*!< Offset: 0x044 (R/W) Mask
Register 2 */	
__IOM uint32_t FUNCTION2;	/*!< Offset: 0x048 (R/W) Function
Register 2 */	
uint32_t RESERVED2[1U];	
__IOM uint32_t COMP3;	/*!< Offset: 0x050 (R/W) Comparator
Register 3 */	
__IOM uint32_t MASK3;	/*!< Offset: 0x054 (R/W) Mask
Register 3 */	
__IOM uint32_t FUNCTION3;	/*!< Offset: 0x058 (R/W) Function
Register 3 */	
} DWT_Type;	

9.3 DWT Programmers Model

Table 9-1 lists the DWT registers. Depending on the implementation of your processor, some of these registers might not be present. Any register that is configured as not present reads as zero.

Table 9-1 DWT register summary

Address	Name	Type	Reset	Description
0xE0001000	DWT_CTRL	RW	See ^a	Control Register
0xE0001004	DWT_CYCCNT	RW	0x00000000	Cycle Count Register
0xE0001008	DWT_CPICNT	RW	-	CPI Count Register
0xE000100C	DWT_EXCCNT	RW	-	Exception Overhead Count Register
0xE0001010	DWT_SLEPCNT	RW	-	Sleep Count Register
0xE0001014	DWT_LSUCNT	RW	-	LSU Count Register
0xE0001018	DWT_FOLDCNT	RW	-	Folded-instruction Count Register
0xE000101C	DWT_PCSR	RO	-	Program Counter Sample Register
0xE0001020	DWT_COMP0	RW	-	Comparator Register0
0xE0001024	DWT_MASK0	RW	-	Mask Register0
0xE0001028	DWT_FUNCTION0	RW	0x00000000	Function Register0
0xE0001030	DWT_COMP1	RW	-	Comparator Register1
0xE0001034	DWT_MASK1	RW	-	Mask Register1
0xE0001038	DWT_FUNCTION1	RW	0x00000000	Function Register1

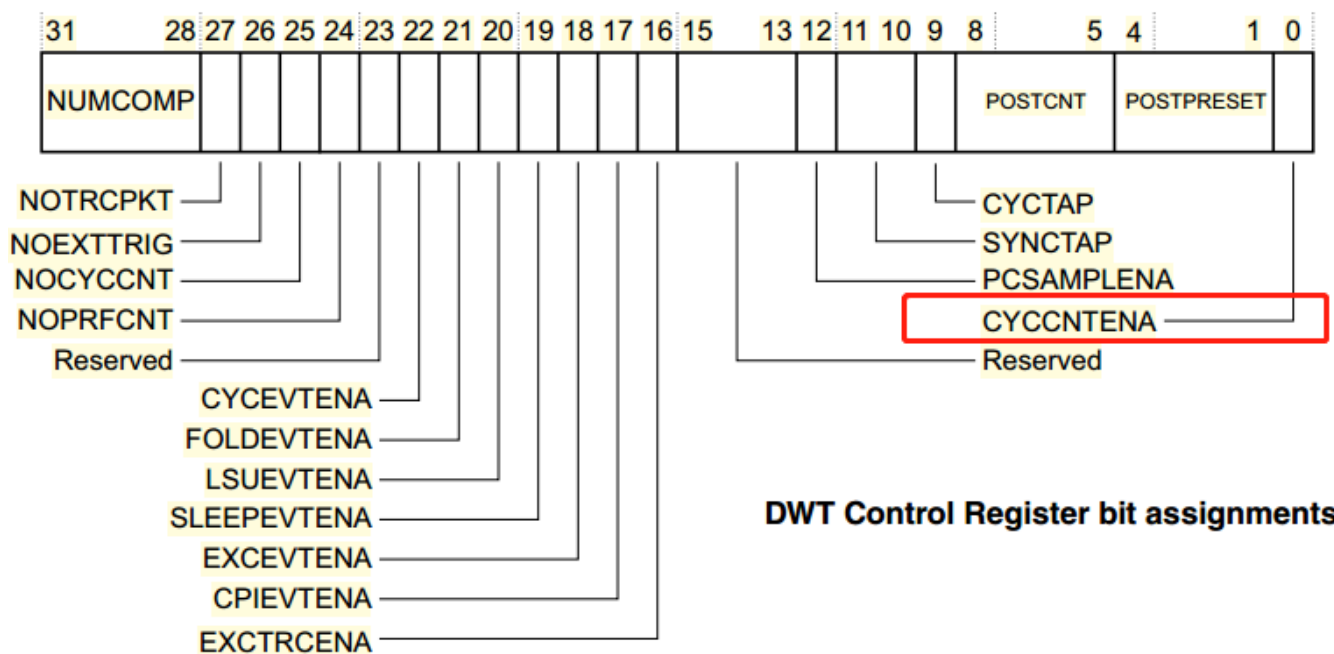
来源自：《F4内核编程手册》

CYCCNTENA

CYCCNTENA Enable the CYCCNT counter. If not enabled, the counter does not count and no event is

generated for PS sampling or CYCCNTENA. In normal use, the debugger must initialize the CYCCNT counter to 0.

它是DWT控制寄存器的第一位，写1使能，则启用CYCCNT计数器，否则CYCCNT计数器将不会工作。



相关实现函数（HAL库）

```
#include "bsp_dwt.h"

/*
 * 1. 先使能DWT外设，这个由另外内核调试寄存器DEMCR的位24控制，写1使能
 * 2. 使能CYCCNT寄存器之前，先清0。
 * 3. 使能CYCCNT寄存器，这个由DWT的CYCCNTENA 控制，也就是DWT控制寄存器的位0控制，写1使能
 */

//寄存器基地址
#define DWT_CR          *(uint32_t*)0xE0001000
#define DWT_CYCCNT      *(uint32_t*)0xE0001004
#define DEM_CR          *(uint32_t*)0xE000EDFC

//定义需使能位
#define DEM_CR_TRCENA    (1<<24)
#define DWT_CR_CYCCNTENA (1<<0)

uint32_t CPU_FREQ_Hz; //系统时钟 单位 MHz
static uint32_t CYCCNT_RountCount;
static float DWT_Timeline;

void DWT_reset_tick(void)
```

```

{
    //系统时钟 单位 MHz
    CPU_FREQ_Hz = HAL_RCC_GetHCLKFreq() / (1000 * 1000); //GET_CPU_ClkFreq()
    HAL_RCC_GetHCLKFreq()
    return;
}

void DWT_init(void)
{
    /* 使能DWT外设 */
    CoreDebug->DEMCR |= CoreDebug_DEMCR_TRCENA_Msk;

    /* DWT CYCCNT寄存器计数清0 */
    DWT->CYCCNT = (uint32_t) 0u;

    /* 使能Cortex-M DWT CYCCNT寄存器 */
    DWT->CTRL |= DWT_CTRL_CYCCNTENA_Msk;

    //HAL_RCC_GetHCLKFreq(); //16000000UL
    DWT_reset_tick();
    CYCCNT_RountCount = 0;
    DWT_Timeline = 0;
}

/*
 * 主频168MHz的情况下，32位计数器计满是 $2^{32}/168000000 = 25.565$ 秒
 * 建议使用本函数做延迟的话，延迟在1秒以下。
 * 两个32位无符号数相减，获取的结果再赋值给32位无符号数依然可以正确的获取差值。
 * 假如A,B,C都是32位无符号数。
 * 如果A > B 那么A - B = C，这个很好理解，完全没有问题
 * 如果A < B 那么A - B = C，C的数值就是 $0xFFFFFFFF - B + A + 1$ 。这一点要特别注意，正好用于本函数。
 */
void DWT_delay_us(uint32_t us)
{
    uint32_t tStart;

    us *= CPU_FREQ_Hz; // 需要延时的节拍数
    tStart = DWT->CYCCNT;

    //两个无符号的数相减，如果前者比后者小，会发生什么问题？这样是否就达不到准确延时的目的了？是否需要考虑溢出的情况？
    while ((DWT->CYCCNT - tStart) < us); /* 求减过程中，如果发生第一次32位计数器重新计数，依然可以正确计算 */
}

void DWT_delay_ms(uint32_t ms)
{

```

```
    DWT_delay_us(1000 * ms);  
}  
  
uint32_t DWT_get_time(void)  
{  
    return (DWT->CYCCNT);  
}
```

参考

主要去看《F4内核编程手册》

[STM32 DWT](#)

[STM32HAL库微秒延时函数的实现---DWT和SysTick](#)

[52. DWT—内核定时器](#)

[延时功能进化论\(合集\)](#)