CH32H417 DMA 性能分析

V1.0

说明

CH32H417 是沁恒新推出的双核产品。CH32H417 的 DMA 相较与之前产品，增加了请求复用器和双缓冲功能，文章对增加功能进行了介绍。分析了 CPU 和 DMA 操作相同 SRAM 的仲裁优先级，DMA 传输速率。

适用范围

适用范围	系列
通用 MCU	CH32H417

目录

表格索引

图片索引

第1章 CH32H417 DMA 功能

1.1 DMA 基本概念

DMA(Direct Memory Access，直接存储器访问)可以在外设和寄存器之间或存储器和存储器之间直接进行数据传输。

DMA 数据传输过程中不需要 CPU 的干预，节省了 CPU 资源。相比传统数据传输方式需要 CPU 全程参与，提高了系统的数据传输效率和芯片性能。

1.2 CH32 DMA 对比

CH32H417 是沁恒最近推出的双核产品，相对于之前 CH32V30X, CH32V20X,CH32V00X 系列产品 DMA 略有不同。

表 1-1 DMA 功能对比

MCU	CH32H41X MCU	CH32V30X MCU
DMA 通道数量	DMA1/DAM2(16 channels)	DMA1/DMA2(18 channels)
数据传输数量	0-65535	0-65535
仲裁优先级	最高/高/中/低	最高/高/中/低
循环传输模式	支持	支持
DMA 请求复用器	支持	不支持
双缓冲模式	支持	不支持
数据传输位宽	8/16/32/256	8/16/32

CH32H417 相比于 CH32V30X 系列芯片增加了 256 bit 位宽和请求复用器，支持双循环模式。当 DMA 配置中选择不同的搬运位宽，搬运地址也要相应对齐。例如： DMA 选择 256bit 搬运位宽，搬运地址要 32 字节对齐。

1.3 DMA 请求复用器

CH32H417 之前产品的每个 DMA 通道都直接连接专用的 DMA 请求。外设请求只能触发相应的 DMA 通道，一定程度上限制了 DMA 的使用。DMAMUX 请求复用器可重新配置芯片的外设和 DMA 控制器之间的请求线。通过请求复用器外设请求可以使用 DMA 任意通道。

CH32H417 DMA提供16 个通道，其中DMAMUX 通道1 到8 与DMA1 通道1 到8 相连，DMAMUX 通道9 到16 与DMA2 通道1 到8 相连。

在外设触发DMA的传输应用中，CH32H417相比于CH32V30X需要按照DMAMAX输入资源分配表分配外设请求。

表 1-2 DMAMAX 输入资源分配表

DMA 请求输入	外设	DMA 请求输入	外设	DMA 请求输入	外设
1	TIM1_CH1	42	TIM9_CH4	83	I3C_TX
2	TIM1_CH2	43	TIM9_UP	84	I3X_RX
3	TIM1_CH3	44	TIM9_TRIG	85	USART1_TX
4	TIM1_CH4	45	TIM10_CH1	86	USART1_RX
5	TIM1_UP	46	TIM10_CH2	87	USART2_TX
6	TIM1_COM	47	TIM10_CH3	88	USART2_RX
7	TIM1_TRIG	48	TIM10_CH4	89	USART3_TX
8	TIM2_CH1	49	TIM10_UP	90	USART3_RX
9	TIM2_CH2	50	TIM10_TRIG	91	USART4_TX
10	TIM2_CH3	51	TIM11_CH1	92	USART4_RX
11	TIM2_CH4	52	TIM11_CH2	93	USART5_TX
12	TIM2_UP	53	TIM11_CH3	94	USART5_RX
13	TIM2_TRIG	54	TIM11_CH4	95	USART6_TX
14	TIM3_CH1	55	TIM11_UP	96	USART6_RX
15	TIM3_CH2	56	TIM11_TRIG	97	USART7_TX
16	TIM3_CH3	57	TIM12_CH1	98	USART7_RX
17	TIM3_CH4	58	TIM12_CH2	99	USART8_TX
18	TIM3_UP	59	TIM12_CH3	100	USART8_RX
19	TIM3_TRIG	60	TIM12_CH4	101	SWPMI_TX
20	TIM4_CH1	61	TIM12_UP	102	SWPMI_RX
21	TIM4_CH2	62	TIM12_TRIG	103	DAC1
22	TIM4_CH3	63	SPI1_TX	104	DAC2
23	TIM4_CH4	64	SPI1_RX	105	保留
24	TIM4_UP	65	SPI2_TX	106	保留
25	TIM4_TRIG	66	SPI2_RX	107	DFSDM_DMA0
26	TIM5_CH1	67	SPI3_TX	108	DFSDM_DMA1
27	TIM5_CH2	68	SPI3_RX	109	保留
28	TIM5_CH3	69	SPI4_TX	110	保留
29	TIM5_CH4	70	SPI4_RX	111	SDIO
30	TIM5_UP	71	QSPI1_DMA	112	SAI_A_TX
31	TIM5_TRIG	72	QSPI2_DMA	113	SAI_A_RX
32	TIM8_CH1	73	I2C1_TX	114	SAI_B_TX
33	TIM8_CH2	74	I2C1_RX	115	SAI_B_RX
34	TIM8_CH3	75	I2C2_TX	116	保留
35	TIM8_CH4	76	I2C2_RX	117	保留
36	TIM8_UP	77	I2C3_TX	118	保留
37	TIM8_COM	78	I2C3_RX	119	保留
38	TIM8_TRIG	79	I2C4_TX	120	ADC1
39	TIM9_CH1	80	I2C4_RX	121	ADC2
40	TIM9_CH2	81	I3C_RS	122	TIM6_UP
41	TIM9_CH3	82	I3C_TC	123	TIM7_UP

示例中使用 TIM1 的更新事件触发 DMA，在 DMA 的配置后，需要操作 DMAMUX 相关寄存器，分配 TIM1 更新事件到 DMA1_Channel1.根据 DMAMUX 输入资源分配表，TIM1_UP 事件的 DMA 请求输入序号为 5，配置 DMAMUX1_4_CFGR[0:6]或直接调用 DMA_MuxChannelConfig()函数。

图 1-1 TIM1_UP 触发 DMA1_Channel1 示例配置

```
void TIM1_DMA_Init(DMA_Channel_TypeDef *DMA_CHx, u32 ppadr, u32 memadr, u16 bufsize)
{
    DMA_InitTypeDef DMA_InitStructure = {0};

    RCC_HBPeriphClockCmd(RCC_HBPeriph_DMA1, ENABLE);
    RCC_HB2PeriphClockCmd(RCC_HB2Periph_AFIO, ENABLE);

    NVIC_SetPriority(DMA1_Channel1_IRQn,0);
    NVIC_EnableIRQ(DMA1_Channel1_IRQn);

    DMA_DeInit(DMA_CHx);
    DMA_InitStructure.DMA_PeripheralBaseAddr = ppadr;
    DMA_InitStructure.DMA_Memory0BaseAddr = memadr;
    DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralDST;
    DMA_InitStructure.DMA_BufferSize = bufsize;
    DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
    DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
    DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_HalfWord;
    DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_HalfWord;
    DMA_InitStructure.DMA_Mode = DMA_Mode_Circular;
    DMA_InitStructure.DMA_Priority = DMA_Priority_VeryHigh;
    DMA_InitStructure.DMA_M2M = DMA_M2M_Disable;

    DMA_Init(DMA_CHx, &DMA_InitStructure);
    DMA_Cmd(DMA_CHx, ENABLE);

}

void Hardware(void)
{
    printf("TIM_DMA\r\n");
    TIM1_PWMOut_Init(100 - 1, 10000 - 1, 60);
    TIM1_DMA_Init(DMA1_Channel1, (u32)TIM1_CH2CVR_ADDRESS, (u32)pbuf, 3);
    DMA_MuxChannelConfig(DMA_MuxChannel1,5);
    TIM_Cmd(TIM1, ENABLE);
    TIM_CtrlPWMOutputs(TIM1, ENABLE);
    while(1)
    {
    }
}
```

DMAMUX配置

1.4 DMA 双缓冲功能

CH32H417 DMA所有通道新增双缓冲功能。通过设置DMA_CFGRx寄存器的DOUBLE_MODE位置1，使能双缓冲模式，自动进入循环模式，并在完成一次循环后自动切换存储器地（DMA_MADDRx和DMA_M1ADDRx间交替切换）。

每次循环结束时 DMA 控制器都从一个存储器目标交换为另一个存储器目标，这样，软件在处理一个存储器区域的同时，DMA 传输还可以填充或使用第二个存储器区域。双缓冲区通道可双向工作，如下表所示。

表 1-3 双缓冲模式下的源和目标地址寄存器

DMA_CFGRx.DIR	方向	源地址	目标地址
0	从外设读	DMA_PADDRx	DMA_MADDRx/DMA_M1ADDRx
1	从存储器读	DMA_MADDRx/DMA_M1ADDRx	DMA_PADDRx

双缓冲模式主要应用于外设与存储器之间进行数据传输，禁止应用于存储器到存储器模式。双缓冲模式有两个存储器地址（DMA_MADDRx和DMA_M1ADDRx），可通过配置DMAy_CFGRx[16]来确定从哪

一个存储器地址（默认DMA_MADDRx）开始传输。DMAy_CFGRx[16]在通道使能后，此位相当于状态标志位，用于指示具体的存储器地址。

图 1-2 DMAy_CFGRx[16]寄存器

16	FLAG_CUR_MEM	RW	存储器地址选择设置，双缓冲模式时硬件会自动切换： 1：M1ADDR（使用 DMA_M1ADDRx 指针寻址）； 0：MADDR（使用 DMA_MADDRx 指针寻址）。 *注：该位仅在双缓冲模式下使用，在通道使能后，此位相当于一个状态标志，用于指示作为当前目标的存储区。*	0

DoubleBuffer_DMA 示例通过 DMA1_Channel1 把 pbuf[3]和 pbuf1[3]中的数据传输到 TIM1_CH2CVR，用于修改定时器的占空比。

图 1-3 DoubleBuffer_DMA 示例配置

```
void TIM1_DMA_Init(DMA_Channel_TypeDef *DMA_CHx, u32 ppadr, u32 memadr,u32 mem1adr, u16 bufsize)
{
    DMA_InitTypeDef DMA_InitStructure = {0};

    RCC_HBPeriphClockCmd(RCC_HBPeriph_DMA1, ENABLE);
    RCC_HB2PeriphClockCmd(RCC_HB2Periph_AFIO, ENABLE);

    NVIC_SetPriority(DMA1_Channel1_IRQn,0);
    NVIC_EnableIRQ(DMA1_Channel1_IRQn);

    DMA_DeInit(DMA_CHx);
    DMA_InitStructure.DMA_PeripheralBaseAddr = ppadr;
    DMA_InitStructure.DMA_Memory0BaseAddr = memadr;      → pbuf 地址
    DMA_InitStructure.DMA_Memory1BaseAddr = mem1adr;     → pbuf1地址
    DMA_InitStructure.DMA_DoubleBuffer_StartMemory = DMA_DoubleBufferMode_Memory_1;
    DMA_InitStructure.DMA_BufferMode = DMA_DoubleBufferMode;
    DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralDST;
    DMA_InitStructure.DMA_BufferSize = bufsize;
    DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
    DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
    DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_HalfWord;
    DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_HalfWord;
    DMA_InitStructure.DMA_Mode = DMA_Mode_Circular;
    DMA_InitStructure.DMA_Priority = DMA_Priority_VeryHigh;
    DMA_InitStructure.DMA_M2M = DMA_M2M_Disable;

    DMA_Init(DMA_CHx, &DMA_InitStructure);
    DMA_Cmd(DMA_CHx, ENABLE);
}
```

选择双缓冲模式

双缓冲模式，从Memory1地址开始传输数据

1.5 外设专用 DMA

HSADC ，SDMMC， USB 等外设有外设内部专用 DMA。前面章节提到的 DMA 是通用 DMA。外设专用 DMA 和通用 DMA 的根本区别在于设计目的，所属权和控制方法。

专用 DMA：直接集成在特定外设内部（如 ETH ，USB， SDMM， DVP 等），它是该外设的一个专属部件。外设无需仲裁，自主控制数据传输。

通用 DMA：外设发出请求，DMA 数据传输。当多个请求同时对一个 DMA 通道发出请求，DMA 通道需要仲裁。

表 1-4 专用 DMA 功能对比表

特性	专用 DMA	通用 DMA
集成位置	集成在外设控制器内部	独立模块
归属权	外设专署	系统共享
设计目的	为特定告诉外设优化	为多个外设提供 DMA 功能
工作方式	外设自主控制数据传输	外设请求 DAM 数据传输
仲裁需求	无需仲裁	需仲裁（多个外设竞争）
性能	低延迟	较高延迟（仲裁与共享通道）
CPU 负担	极低（仅初始化和结尾）	较低（但需配置 DMA 控制寄存器）

专用 DMA 在使用过程中也需要数据对齐。

1. CH32H417 USBHS 专用 DMA 缓冲区地址需要 4 字节对齐
2. CH32H417 ETH 专用 DMA 收发描述符队列和缓冲区都需要保证它们的起始地址在 32 字节对齐
3. CH32H417 DVP 专用 DMA 缓冲区地址需要 32 字节对齐
4. CH32H417 USBPD 专用 DMA 缓冲区地址需要 4 字节对齐
5. CH32H417 SerDes 专用 DMA 缓冲区地址在接收模式下需要 16 字节对齐，发送模式下不需要 16 字节对齐
6. CH32H417 HSADC 专用 DMA 缓冲区地址需要 32 字节对齐
7. CH32H417 SDMMC 专用 DMA 缓冲区地址需要 16 字节对齐

图 1-4 SDMMC 缓冲区地址定义

```
21    __attribute__((aligned(16))) u8 buf[512];
22    __attribute__((aligned(16))) u8 Readbuf[512];
```

定义SDMMC缓存区

第2章 CH32H417 DMA 性能

2.1 CH32H417 SRAM 分配

CH32H417 内置总容量 896K 字节 SRAM。SRAM 分为 3 块：128KB 的 ITCM 代码区、256KB 的 DTCM 数据区、剩余 512KB 共享代码和数据区。

其中 ITCM 和 DTCM 在 TCM 总线上，TCM 总线速率和 RISC_V5 速率相同。

共享代码区和数据区挂载在 HB 总线上，HB 总线速率和 RISC_V3 速率相同。

512KB 共享区可配置为 RISC-V3F 的零等待代码区和数据区，建议以 128KB 为单位根据需要灵活分配。

128KB 的 ITCM 和 256KB 的 DTCM 合计 384KB 均可以配置为 RISC-V5F 的代码区，其中 DTCM 作为代码区在发生跳转时会增加 1 个时钟等待。

RISC-V3F 可以按 HCLK 时钟 2 个等待访问 ITCM 或 DTCM。

RISC-V5F 和 RISC-V3F 可以按 HCLK 时钟零等待访问 512KB 共享区。

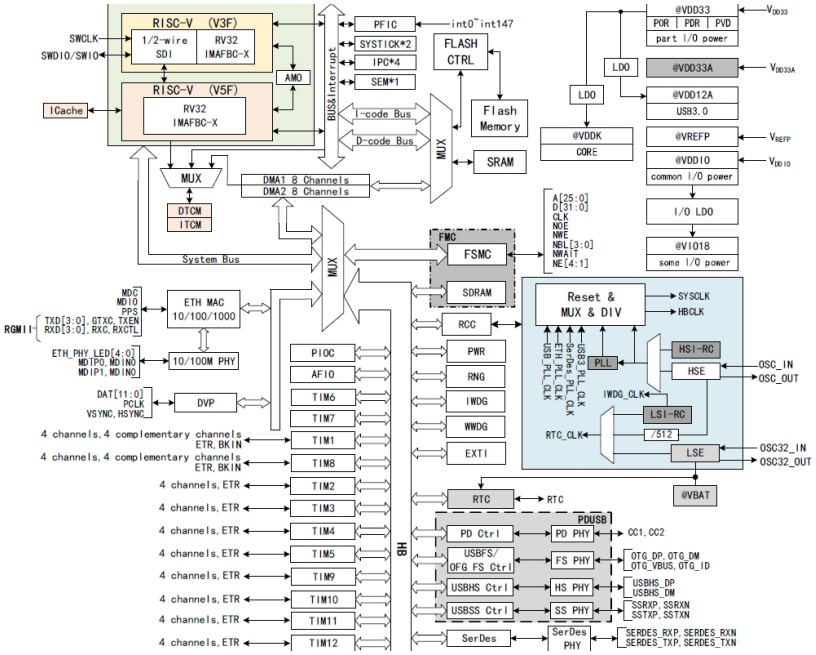
沁恒官方例程中 ITCM 和 DTCM 分配给 RISC_V5，512KB 共享区域部分分配给 RISC_V3。

RISC_V5 例程中使用的变量将会存储在 DTCM 区域，RISC_V3 例程中使用的变量将会存储在共享数据区。

2.2 DMA 传输速率

在评估 DMA 性能时，需要考虑多个因素，其中 DMA 数据的传输速度是一个关键指标。DMA 传输的速度直接受限于 DMA 访问的源地址和目的地址在总线上的位置。SRAM 到 SRAM 的 DMA 传输非常快就是因为都挂载在 HB 高速总线上。CH32H417 的大多数外设也是直接挂载在 HB 总线上，所以通过 DMA 在 SRAM 和外设之间传输速度和 SRAM 之间传输数据相同。CH32V30X 系列芯片 SRAM 挂载在 HB 总线上，外设挂载在 PB1/PB2 总线上，DMA 传输外设与 SRAM 之间的数据必须经过 PB-HB 桥-HB-SRAM ，路径较长会限制 DMA 的传输速度。CH32H417 DTCM 挂载在 TCM 总线上，CH32H417 例程变量存放在 DTCM。DMA 挂载在 HB 总线上。

图 2-1 CH32H417 系统框图



CH32H417 的 RISC_V5 通过 DMA 对 DTCM 进行数据传输时，DMA 在读/写 DTCM 的过程中相比与传输 SRAM 数据传输会多等待一个 HB 周期。

2.3 DMA/CPU 仲裁优先级

当 DMA 和内核同时访问同一块 SRAM 时，会产生仲裁，确定访问的优先级。

CH32H417 的 DTCM 按 128K 分成 2 块，当 DMA，RISC_V5，RISC_V3 同时操作同一块 128K DTCM 时，根据 MEMARY_CFGR[18:19]优先级设置仲裁访问 DTCM。

图 2-2 DTCM 存储器访问优先级

[19:18]	dtcm_rr_mode[1:0]	MRW	DTCM 存储器访问优先级模式： 00：固定优先级，其他请求（DMA，CO_lsu，CO_ifu）高于 C1 请求； 01：固定优先级，C1 请求高于其他请求（DMA，CO_lsu，CO_ifu）； 10：轮询优先级，一通道最多连续获得一段时间的操作优先权，超时后切换到另一通道，当获得优先权的通道无操作请求而令一通道有请求时优先权切换到另一通道并重新计时，优先时长配置见 TCM_RRDUTY_CFGR 寄存器。 11：强制轮询优先级，一通道强制获得一段时间的操作优先权，超时后切换到另一通道，优先时长配置见 TCM_RRDUTY_CFGR 寄存器。	0x3

CH32H417 的共享 RAM，按 128K 分成 4 块。当 DMA，RISC_V5，RISC_V3 同时操作同一块 128K SRAM 时，通过总线仲裁来访问 SRAM。访问 SRAM 的仲裁规则是固定的，不可配置修改的。

仲裁规则：

DMA1 > DMA2 > RISC_V3 > RISC_V5

赢得仲裁的主设备（例如 DMA）可以访问 SRAM，而其他设备（例如 RISC_V3）将会临时挂起，等待 DMA 访问完成，当 DMA 访问 SRAM 结束后，挂起设备再操作 SRAM。内核挂起只是插入等待周期，导致指令执行稍慢，不会产生异常或错误。整个过程由硬件自动完成。

注：用户要避免同时操作同一块 SRAM，例如内核在累加 SRAM 中数组时，DMA 会传输改变 SRAM，导致内核最终累加数据错误。

解决方案：

1. 使用双缓冲区功能，避免内核和 DMA 操作同一个缓冲区
2. 确保内存区域不重合。为 DMA 和内核分配完全独立区域，根本上避免冲突

内核使用 SRAM 时关闭 DMA 功能，使用结束后开启 DMA 功能。只适用于实时性要求不高的场景，使用时要十分小心。

历史版本

更新内容

日期	版本	变更内容
2025/9/22	V1.0	初版发行

声明

任何未经允许擅自更改本产品手册中的内容与南京沁恒微电子股份有限公司无关。

南京沁恒微电子股份有限公司所提供的说明文档只作为相关产品的使用参考，不包含任何对特殊使用目的的担保。南京沁恒微电子股份有限公司保留更改和升级本产品手册以及手册中涉及的产品或软件的权利。

参考手册中可能包含少量由于疏忽造成的错误。已发现的会定期勘误，并在再版中更新和避免出现此类错误。