Application Note

ZHCAFM7 – July 2025

TF2838xD 多核处理器间数据快速交互的实现

Strong Zhang, Peter Ban

摘要

在多核处理器系统中，各内核之间的数据传输的及时性和准确性，对实现实时的控制系统十分重要。在 2838xD 中，由于 CPU2 和 CPU1.CLA1 中不存在共享的 RAM 空间，因此 CPU2 和 CPU1.CLA1 之间难以实现快速的数据交互。本文分析了传统的利用 IPC 的方法实现数据交互的局限性，利用 28388D 现有的外设资源，提出了一种“IO 触发 + DMA 搬运”的方法，实现 CPU2 和 CPU1.CLA1 间数据实时的快速交互。同样地，此方法可以适用于与 2838xD 具有相似架构的 F28P65 处理器上，实现多核处理器间数据的快速交互。

内容

1 TMS320F2838xD 的介绍和功能框图 2
2 通过 IPC 模块实现 CPU2 和 CPU1.CLA1 间的数据传输的局限性 2
3 利用“IO 触发+DMA 搬运”实现 CPU2 和 CPU1.CLA1 数据传输的原理 4
4 “IO 触发+DMA 搬运”方法的验验证 6
4.1 时序说明及代码实现 6
4.2 实验环境搭建 8
4.3 时序波形验证 9
5 总结 10
References 10

Figures

Figure 1. TMS320F28385D 功能框图，CPU1.CLA1 和 CPU2 之间不存在共享的 RAM 空间 2
Figure 2. 数据流，通过 IPC 模块实现 CPU2 和 CPU1.CLA1 之间数据交互 3
Figure 3. IPC 模块实现 CPU2 和 CPU1.CLA1 之间数据交互 4
Figure 4. 使用 IPC 方法的 CPU1 的开销 4
Figure 5. “IO 触发+DMA 搬运”实现 CPU2 和 CPU1.CLA1 之间数据交互 5
Figure 6. “IO 触发+DMA 搬运”实现 CPU2 和 CPU1.CLA1 之间数据交互 6
Figure 7. IO 触发 + DMA 搬运的时序图 6
Figure 8. 实验环境，TMDSCNCD28388D + TMDSHSECDOCK 9
Figure 9. IO 触发 + DMA 搬运，实验波形 9
Figure 10. IO 触发 + DMA 搬运，实验时序 10

1 TMS320F2838xD 的介绍和功能框图

TMS320F2838xD 是多核架构的 C2000 高性能系列的产品，集成两个 200MHz 的 C28 核，两个 200MHz 的 CLA 核，以及一个 ARM 核，可以实现多轴的带 Ethercat 的电机控制。

在双轴伺服控制系统中，一个应用场景是用 CPU1 做 Ethercat 控制，CPU1.CLA 和 CPU2 分别做不同轴的电机控制，由于编码器解码要用到 CLB 和 SPI 模块，因此对于两个轴的编码器信号的接收都由 CPU2 统一处理， CPU2 收到编码器的数据后需要快速的传输给 CPU1.CLA，确保电机能够实现精准的实时控制 。

图 1 所示为 TMS320F28388D 的功能框图。CPU2 和 CPU1 之间有共享的 Message RAM, 通过 IPC 模块发送同步触发指令，可以快速的实现数据交互，但 CPU1.CLA1 和 CPU2 间不存在共享的 RAM 空间， CPU1.CLA1 和 CPU2 不能通过各自的数据总线访问相同的存储空间，因此在 CPU1.CLA1 和 CPU2 之间难以实现数据的快速交互。

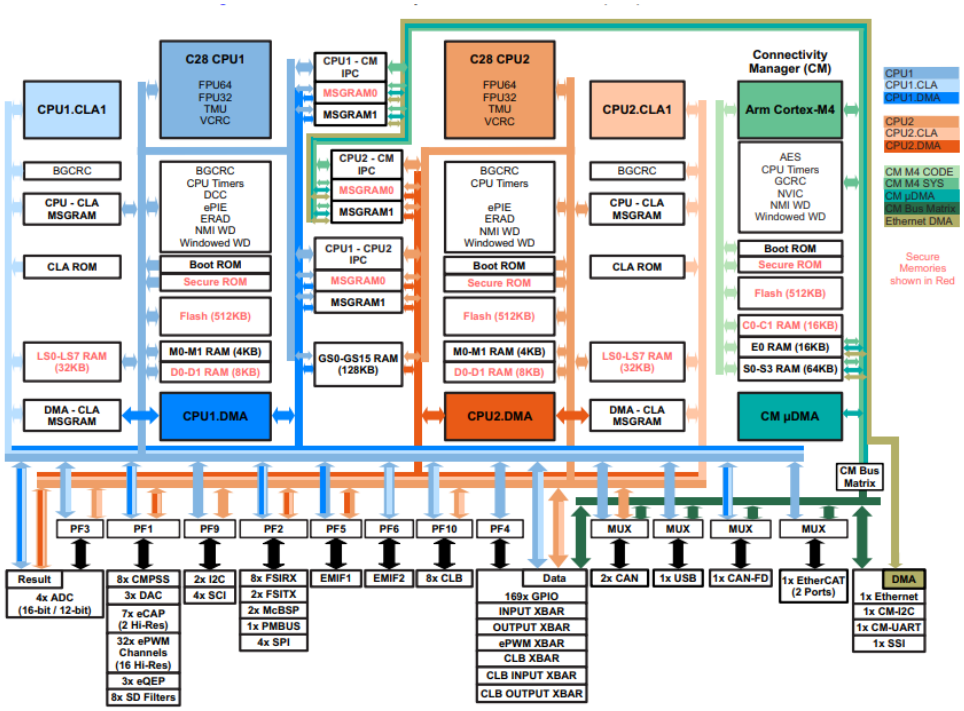

Figure 1. TMS320F28388D 功能框图，CPU1.CLA1 和 CPU2 之间不存在共享的 RAM 空间

2 通过 IPC 模块实现 CPU2 和 CPU1.CLA1 间的数据传输的局限性

由于 CPU2 和 CPU1.CLA1 之间不存在的共享 RAM 空间，传统的方法是通过 IPC 模块实现 CPU2 和 CPU1.CLA1 的数据交互。

首先，CPU2 可以通过 IPC 模块(CPU1-CPU2 IPC)将数据传输到 MSGRAMx/GSx RAM 中，之后 CPU1 可以通过 CPU1.DMA 模块通过 DMA 搬运的方式将数据从 MSGRAMx/GSx RAM 中搬运到 DMA-CLA1 MSGRAM 中，最后 CPU1.CLA1 通过访问 DMA-CLA1 MSGRAM 将数据读出，并在 CLA1 中进行后续的处理。

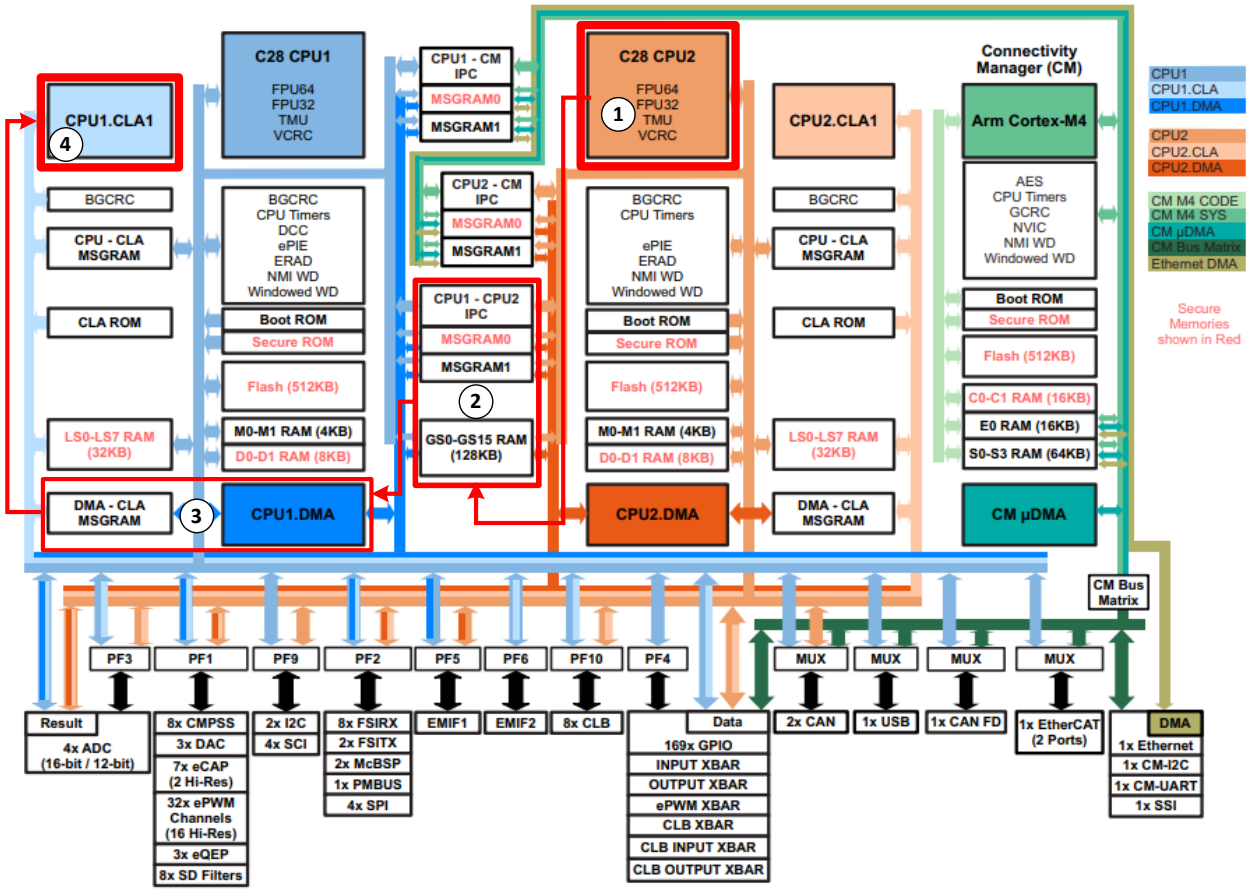

Figure 2.　数据流，通过 IPC 模块实现 CPU2 和 CPU1.CLA1 之间数据交互

图 3 为 IPC 模块实现 CPU2 和 CPU1.CLA1 间数据传输的具体过程：

1. CPU2 首先将把需要传递给 CLA1 的数据写入 MSG RAMx/GSx RAM 中。
2. 在 CPU2 中向 C2TOC1IPCSET 寄存器中的写 1，将 C2TOC1IPCFLG 置 1，由于 CPU2 的 C2TOC1IPCFLG 寄存器和 CPU1 的 C2TOC1IPCSTS 是同一个物理地址上的寄存器， 因此 CPU1 中的 C2TOC1IPCSTS 寄存器也置 1，同时触发了 CPU1 产生了 IPC 的中断，告知 CPU1 数据已准备好，可以进行下一步操作。
3. CPU1 检测到 C2TOC1IPCSTS 寄存器置 1。
4. CPU1 开始从 MSG RAMx/GSx RAM 中读取数据，同时启动 DMA 任务，通过 CPU1 的 DMA 功能，将数据从 MSG RAMx/GSx RAM 搬运到 DMA-CLA1 MSGRAM 中。

5. CPU1 将 C2TOC1IPCACK 寄存器 置 1，以确认它已收到数据。 随后清除了 C2TOC1IPCFLG 和 C2TOC1IPCSTS 中的标志位，完成本次数据传输流程。

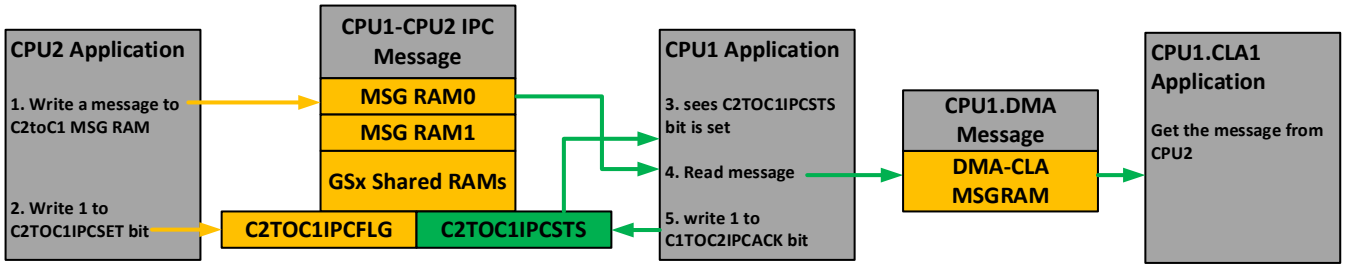

Figure 3. IPC 模块实现 CPU2 和 CPU1.CLA1 之间数据交互

基于上述分析，由于 CPU2 和 CPU1 间的数据交互是通过 IPC 模块实现的，在数据交互过程中，CPU1 需要完成上述过程的第三步到第五步，这一过程增加了 CPU1 的开销。因此在 CPU1 处理对时间敏感的任务或者 CPU1 的开销很大时，通过 IPC 模块实现 CPU2 和 CPU1.CLA1 间数据传输的方式存在一定的局限。

如图 4 所示， CPU1 完成第三步到第五步的时间，CPU 的开销大约花费 2.93us。

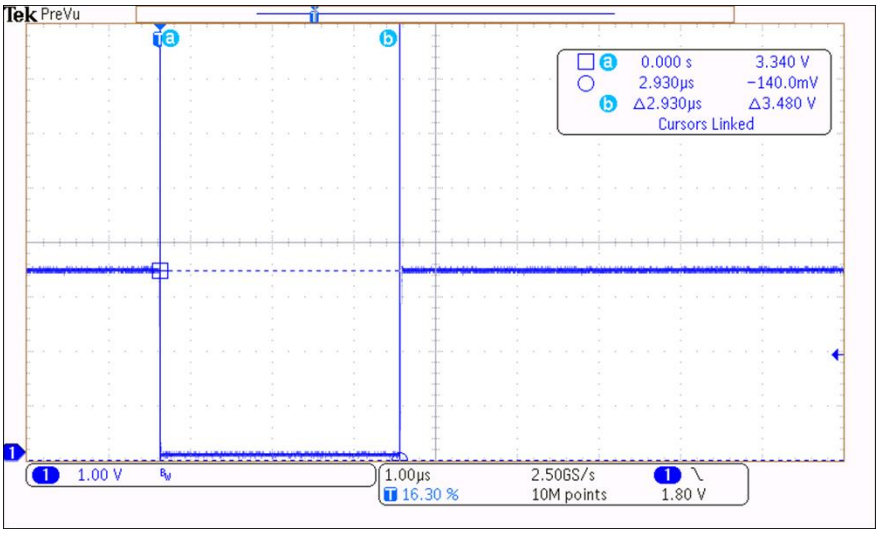

Figure 4. 使用 IPC 方法的 CPU1 的开销

3 利用“IO 触发+DMA 搬运”实现 CPU2 和 CPU1.CLA1 数据传输的原理

为了实现更快速的数据交互，提出了一种利用“IO 触发+DMA 搬运”实现 CPU2 和 CPU1.CLA1 间的数据传输的方法。该方法通过 IO 口触发 DMA 搬运实现了 CPU2 和 CPU1.CLA1 间的数据传输，并且不占用 CPU1 的开销。

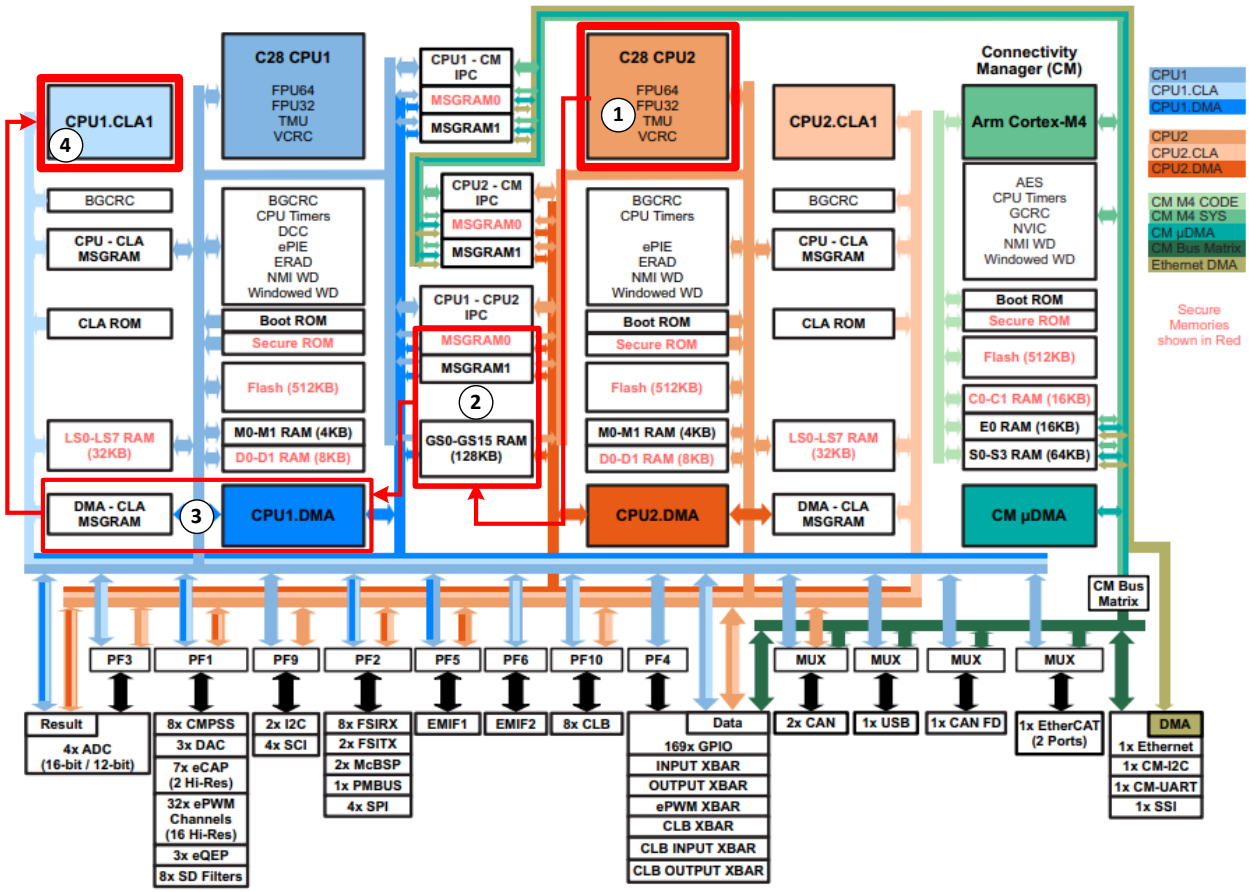

Figure 5. “IO 触发+DMA 搬运”实现 CPU2 和 CPU1.CLA1 之间数据交互

首先 CPU2 会将需要传输的数据写入 GSx RAM 中，并且在每次数据写入结束后，在 CPU2 中会将一个 GPIO 口的电平翻转，通过该 GPIO 口的翻转作为 CPU1.DMA 的触发源，触发 CPU1.DMA 的 DMA 搬运任务，通过 CPU1.DMA 将 GSx RAM 数据搬运到 DMA-CLA MSGRAM 中，最后 CPU1.CLA1 从 DMA-CLA MSGRAM 中将数据读出，并在 CLA1 中进行后续的处理，与 IPC 的方案相比减少了 CPU1 的参与，实现更快的数据传输。

图 6 为“IO 口触发+DMA 搬运”的流程，具体如下，

1. CPU2 首先将把需要传递给 CLA1 的数据写入 MSG RAMx 或者 GSx RAM 中。
2. CPU2 在数据写入完成后将一个 IO 口翻转，用作 CPU1.DMA 的触发源。
3. CPU1.DMA 被触发后，将 MSG RAMx/GSx RAM 中的数据搬运到 DMA-CLA1 MSGRAM
4. CPU1.CLA1 可以直接读取 DMA-CLA1 MSGRAM 中的数据，并进行后续的运算。

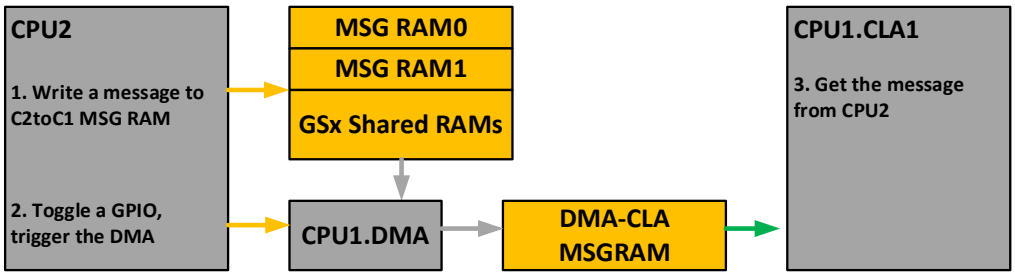

Figure 6. “IO 触发+DMA 搬运”实现 CPU2 和 CPU1.CLA1 之间数据交互

4 “IO 触发+DMA 搬运”方法的验验证

为了更好的说明本方案的实现，设计了以下实验对利用“IO 触发+DMA 搬运”实现 CPU2 和 CPU1.CLA1 间的数据传输方法进行验证，图 7 所示为此演示的时序说明。为了更好的理解数据传输过程中的各个阶段，额外加入了一个 GPIO3 提示各阶段任务的开始和结束。

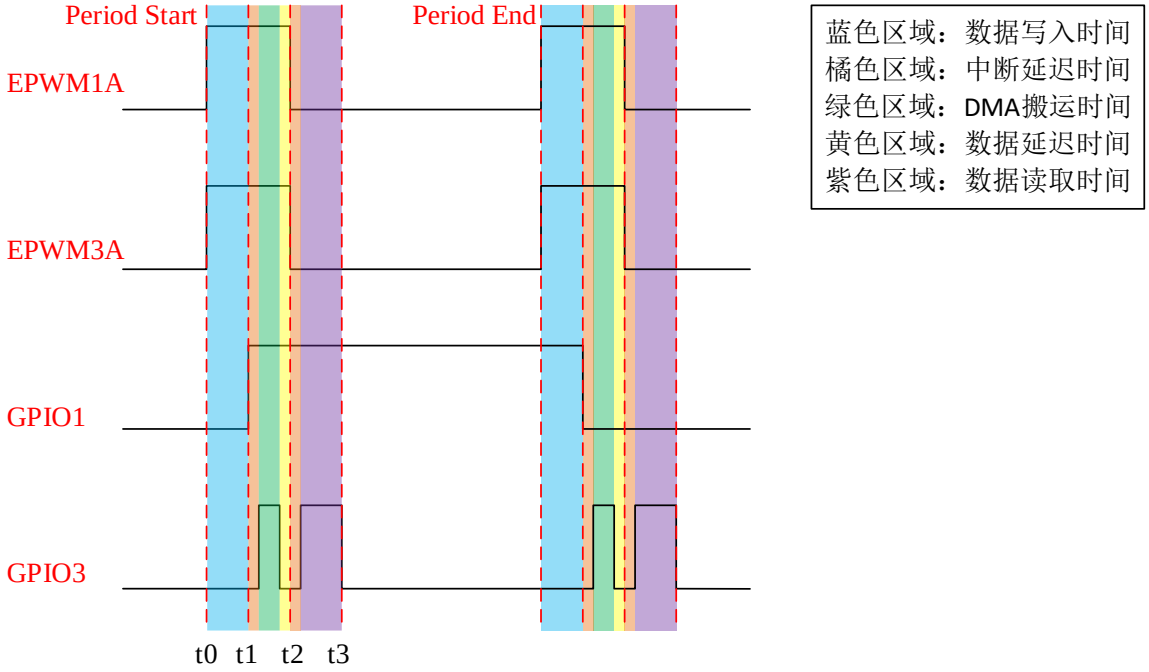

Figure 7. IO 触发 + DMA 搬运的时序图

4.1 时序说明及代码实现

1. t0 时刻，EPWM1A 的上升沿触发中断写任务，将数据写入 GS0RAM 中，并在写任务的结束将 GPIO1 的电平翻转。

```
//
__interrupt void INT_myEPWM1_ISR(void)
{
    //GPIO_togglePin(myGPIO9);

    //
    // Initialize the data buffers
    //
    uint16_t i;
    for(i = 0; i < 16; i++)
    {
        sourceData[i] = i;
    }

    //
    //toggle GPIO
    //
    GPIO_togglePin(myGPIO1);

    Debugger1 = 1;

    //
    // Clear INT flag for this timer
    //
    EPWM_clearEventTriggerInterruptFlag(EPWM1_BASE);
```

2. t1 时刻，由于 GPIO1 的电平翻转会产生一个外部中断，该中断可以作为 DMA 搬运的触发源。

```
//*****************************************************************************
//
// INTERRUPT Configurations
//
//*****************************************************************************
void INTERRUPT_init(){

    // Interrupt Setings for INT_myCLA1
    Interrupt_register(INT_myCLA1, &INT_myCLA1_ISR);
    Interrupt_enable(INT_myCLA1);

    // Interrupt Setings for INT_myDMA0
    Interrupt_register(INT_myDMA0, &INT_myDMA0_ISR);
    Interrupt_enable(INT_myDMA0);

    // Interrupt Setings for INT_myEPWM3
    Interrupt_register(INT_myEPWM3, &INT_myEPWM3_ISR);
    Interrupt_enable(INT_myEPWM3);

    // Interrupt Setings for INT_myGPIO1_XINT
    Interrupt_register(INT_myGPIO1_XINT, &INT_myGPIO1_XINT_ISR);
    Interrupt_enable(INT_myGPIO1_XINT);
}
//*****************************************************************************
//
// DMA Configurations
//
//*****************************************************************************
void DMA_init(){
    DMA_initController();
    myDMA0_init();
}

void myDMA0_init(){
    DMA_setEmulationMode(DMA_EMULATION_STOP);
    DMA_configAddresses(myDMA0_BASE, 5888, 53248);
    DMA_configBurst(myDMA0_BASE, 9U, 1, 1);
    DMA_configTransfer(myDMA0_BASE, 3U, 1, 1);
    DMA_configWrap(myDMA0_BASE, 65535U, 0, 65535U, 0);
    DMA_configMode(myDMA0_BASE, DMA_TRIGGER_XINT1, DMA_CFG_ONESHOT_ENABLE | DMA_CFG_CONTINUOUS_ENABLE | DMA_CFG_SIZE_16BIT);
    DMA_setInterruptMode(myDMA0_BASE, DMA_INT_AT_END);
    DMA_enableInterrupt(myDMA0_BASE);
    DMA_disableOverrunInterrupt(myDMA0_BASE);
    DMA_enableTrigger(myDMA0_BASE);
    DMA_startChannel(myDMA0_BASE);
}
//*****************************************************************************
```

3. t2 时刻，EPWM3A 的下降沿触发 CLA 任务，CPU1.CLA1 读取数据，t3 时刻，数据读取结束。

```
58 //---------------------------------------------------------------------------
59 //
60 // Task 1 - Read Task in CLA
61 //
62 //---------------------------------------------------------------------------
63 __attribute__((interrupt)) void Cla1Task1 ( void )
64 {
65     //__mdebugstop();
66
67     //
68     // Read the Data
69     //
70     uint16_t i=0;
71     uint16_t j=0;
72     for( i = 0; i < 16; i++ )
73     {
74         claData[i] = rData[j];
75         j++;
76     }
77     debugger_cla = 5;
78 }
79
```

4.2 实验环境搭建

根据上述的实验设计，利用 TI 的开发板 TMDSCNCD28388D 和 TMDSHSECDOCK 进行了实验验证。

实验设备：

1. TMDSCNCD28388D — F28388D evaluation module for C2000™ MCU controlCARD™
2. TMDSHSECDOCK — HSEC180 controlCARD baseboard docking station

接线：

1. PIN49-EPWM1A
2. PIN51-GPIO1
3. PIN50-EPWM3A
4. PIN55-GPIO3

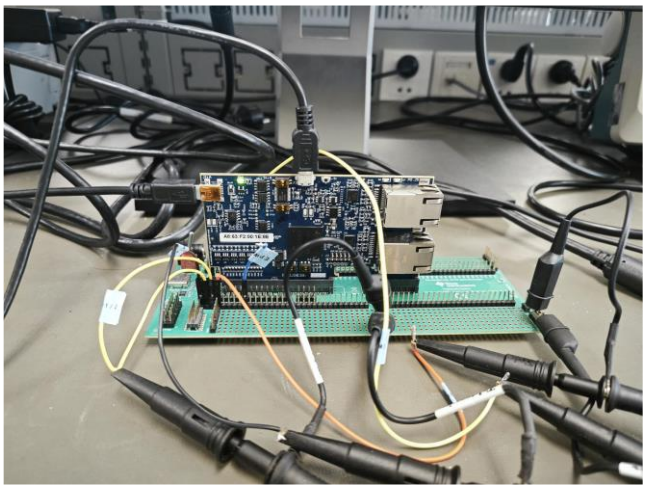

Figure 8.　实验环境，TMDSCNCD28388D + TMDSHSECDOCK

4.3　时序波形验证

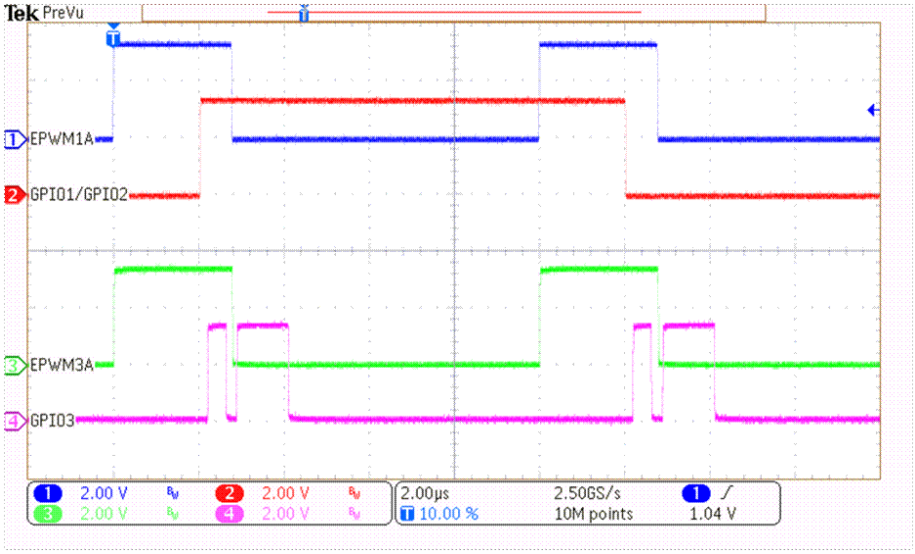

Figure 9.　IO 触发 + DMA 搬运，实验波形

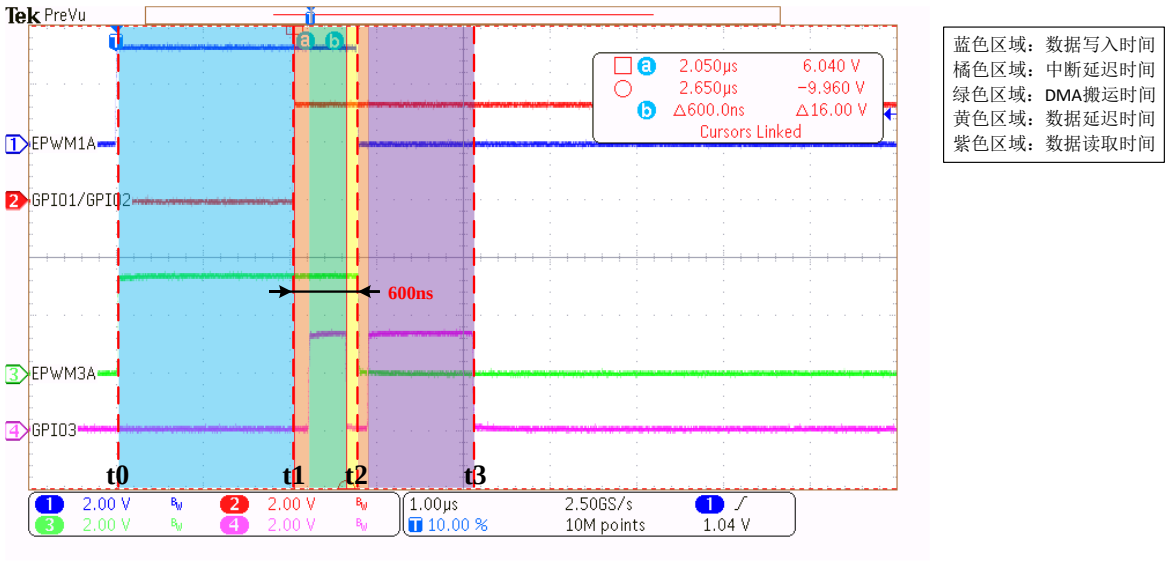

Figure 10. IO 触发 + DMA 搬运，实验时序

图 10 所示为利用“IO 触发+DMA 搬运”实现 CPU2 和 CPU1.CLA1 间的数据传输的时序验证波形。从数据写入结束(t1 时刻)到 DMA 搬运结束(t2 时刻)，大约花费 600ns。与 IPC 的方式相比在不增加 CPU1 开销的条件下，满足系统数据传输的快速性要求。

5 总结

针对 28388D 中出现的 CPU2 和 CLA1 间数据快速交互的需求，本方法利用现有的芯片外设，创新性的提出了一种利用 GPIO 口触发 DMA 中断的方法，有效解决了 CPU2 和 CPU1.CLA1 间数据交互困难，数据交互慢的痛点。利用现有资源实现了客户的需求。所提思路不仅可以应用在 28388D/28385D，同时也可以解决具有类似架构的多核 MCU 间的数据交互问题。

References

1. *TMS320F2838x Real-Time Microcontrollers With Connectivity Manager TRM (Rev. F)*

重要通知和免责声明

TI“按原样”提供技术和可靠性数据（包括数据表）、设计资源（包括参考设计）、应用或其他设计建议、网络工具、安全信息和其他资源，不保证没有瑕疵且不做出任何明示或暗示的担保，包括但不限于对适销性、某特定用途方面的适用性或不侵犯任何第三方知识产权的暗示担保。

这些资源可供使用 TI 产品进行设计的熟练开发人员使用。您将自行承担以下全部责任：(1) 针对您的应用选择合适的 TI 产品，(2) 设计、验证并测试您的应用，(3) 确保您的应用满足相应标准以及任何其他功能安全、信息安全、监管或其他要求。

这些资源如有变更，恕不另行通知。TI 授权您仅可将这些资源用于研发本资源所述的 TI 产品的相关应用。 严禁以其他方式对这些资源进行复制或展示。您无权使用任何其他 TI 知识产权或任何第三方知识产权。您应全额赔偿因在这些资源的使用中对 TI 及其代表造成的任何索赔、损害、成本、损失和债务，TI 对此概不负责。

TI 提供的产品受 TI 的销售条款或 ti.com 上其他适用条款/TI 产品随附的其他适用条款的约束。TI 提供这些资源并不会扩展或以其他方式更改 TI 针对 TI 产品发布的适用的担保或担保免责声明。

TI 反对并拒绝您可能提出的任何其他或不同的条款。

邮寄地址：Texas Instruments, Post Office Box 655303, Dallas, Texas 75265