

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 过 IPC 模块实现 CPU2 和 CPU1.CLA1 间的数据传输的局限性.....	3
3 利用"IO 触发+DMA 搬运"实现 CPU2 和 CPU1.CLA1 数据传输的原理.....	5
4 "IO 触发+DMA 搬运"方法的验证.....	6
4.1 时序说明及代码实现.....	7
4.2 实验环境搭建.....	8
4.3 时序波形验证.....	9
5 总结.....	10
6 参考文献.....	10

插图清单

图 1-1. TMS320F28388D 功能框图，CPU1.CLA1 和 CPU2 之间不存在共享的 RAM 空间.....	2
图 2-1. 数据流，通过 IPC 模块实现 CPU2 和 CPU1.CLA1 之间数据交互.....	3
图 2-2. IPC 模块实现 CPU2 和 CPU1.CLA1 之间数据交互.....	4
图 2-3. 使用 IPC 方法的 CPU1 的开销.....	4
图 3-1. "IO 触发+DMA 搬运"实现 CPU2 和 CPU1.CLA1 之间数据交互.....	5
图 3-2. "IO 触发+DMA 搬运"实现 CPU2 和 CPU1.CLA1 之间数据交互.....	6
图 4-1. IO 触发 + DMA 搬运的时序图.....	6
图 4-2. 实验环境，TMDSCNCD28388D + TMDSHSECDCK.....	8
图 4-3. IO 触发 + DMA 搬运，实验波形.....	9
图 4-4. IO 触发 + DMA 搬运，实验时序.....	9

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 1-1 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。

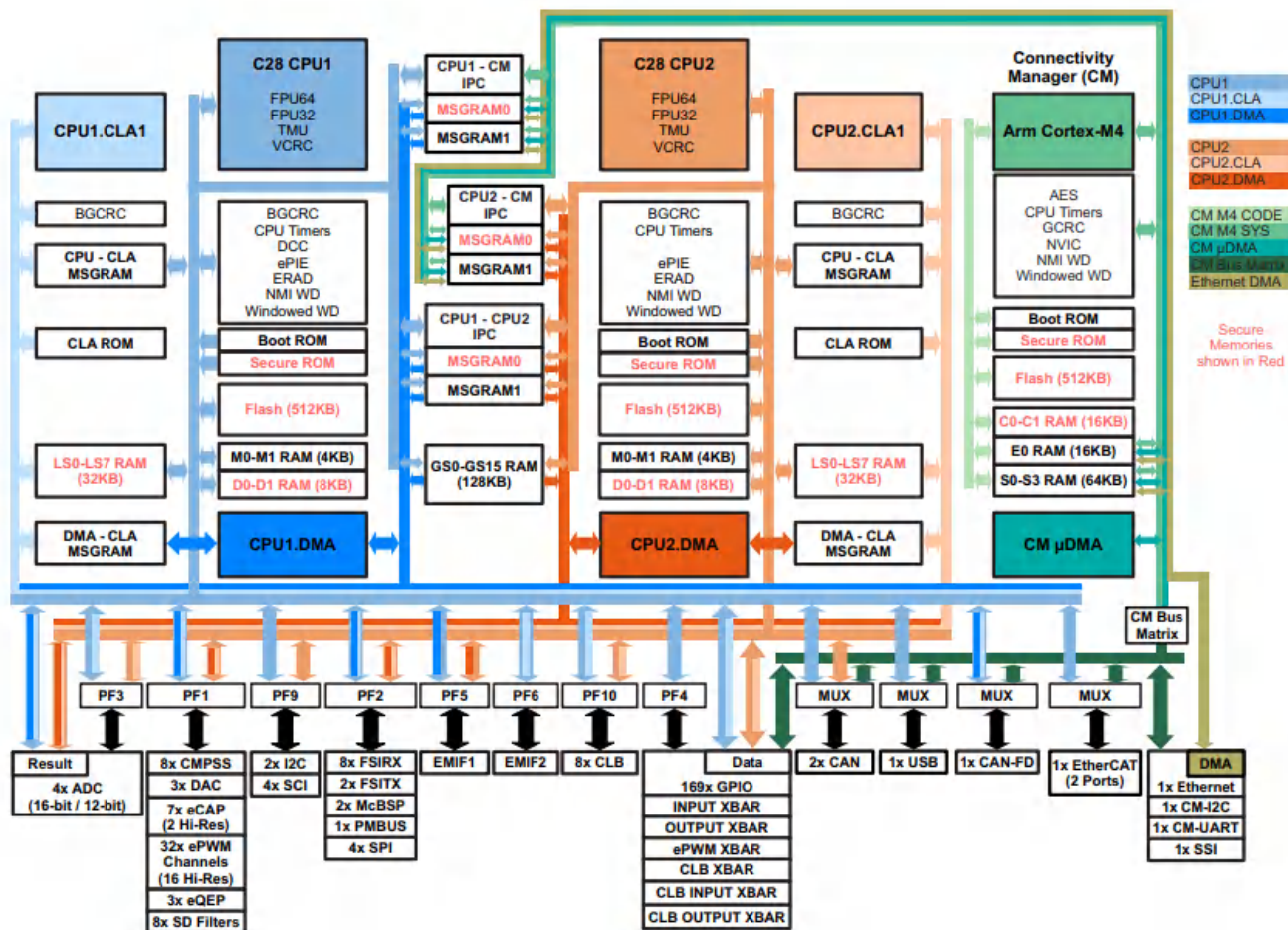


图 1-1. TMS320F28388D 功能框图，CPU1.CLA1 和 CPU2 之间不存在共享的 RAM 空间

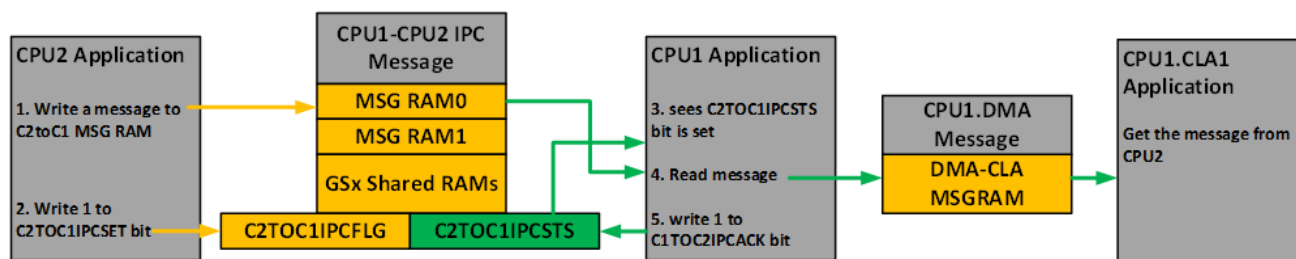


图 2-2. IPC 模块实现 CPU2 和 CPU1.CLA1 之间数据交互

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 2-3 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。

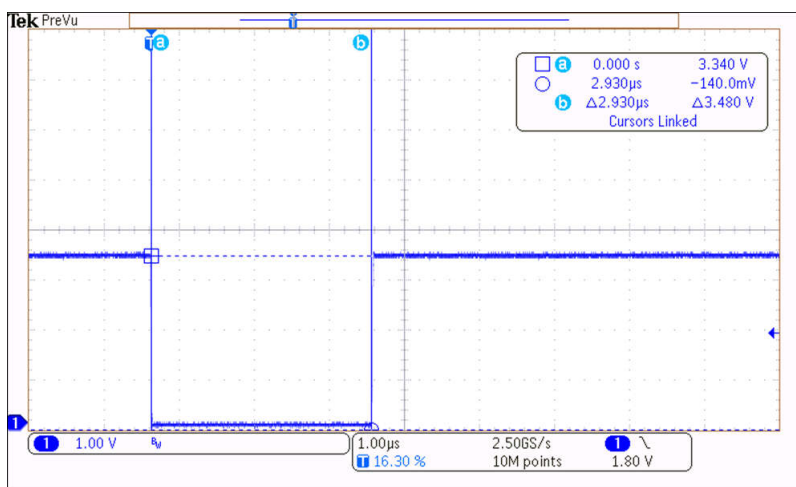


图 2-3. 使用 IPC 方法的 CPU1 的开销

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。

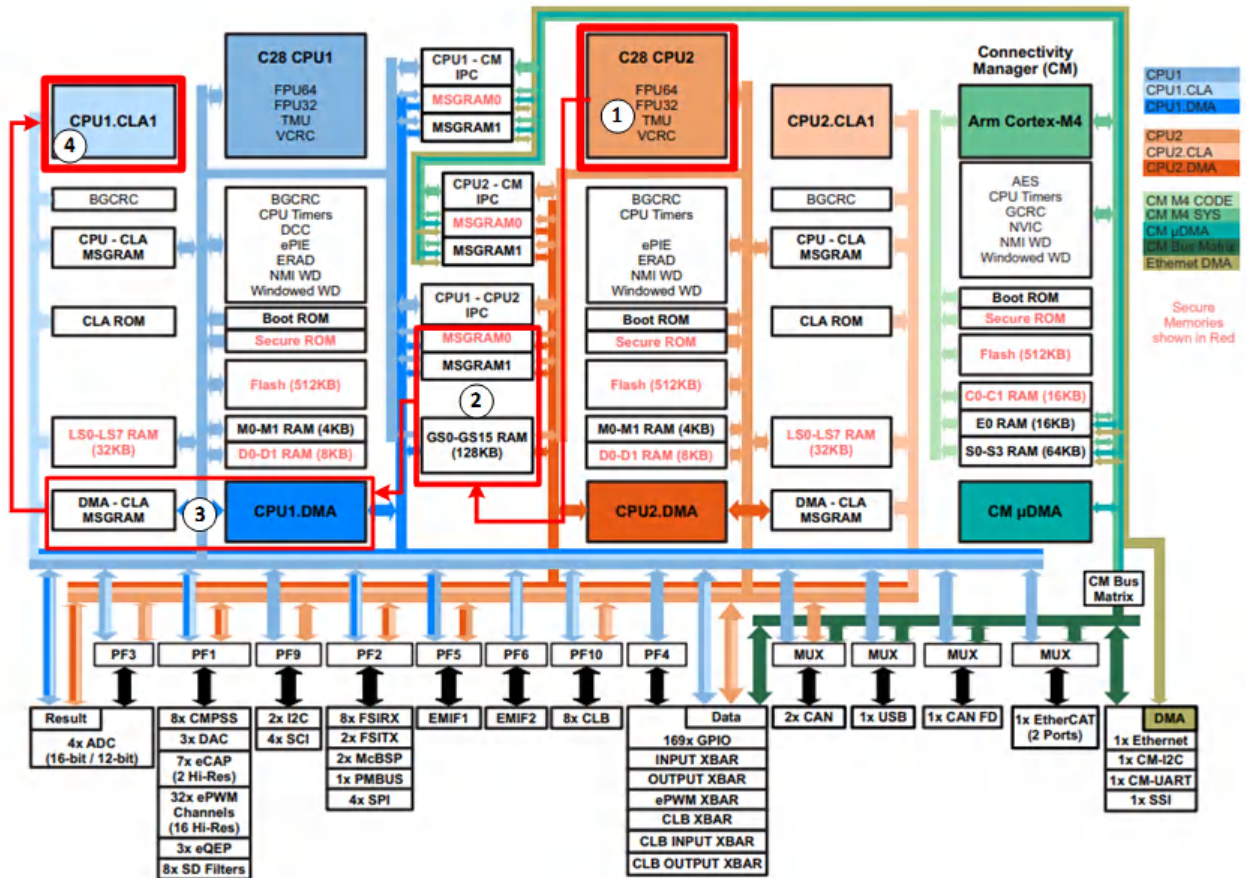


图 3-1. "IO 触发+DMA 搬运"实现 CPU2 和 CPU1.CLA1 之间数据交互

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 3-2 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。

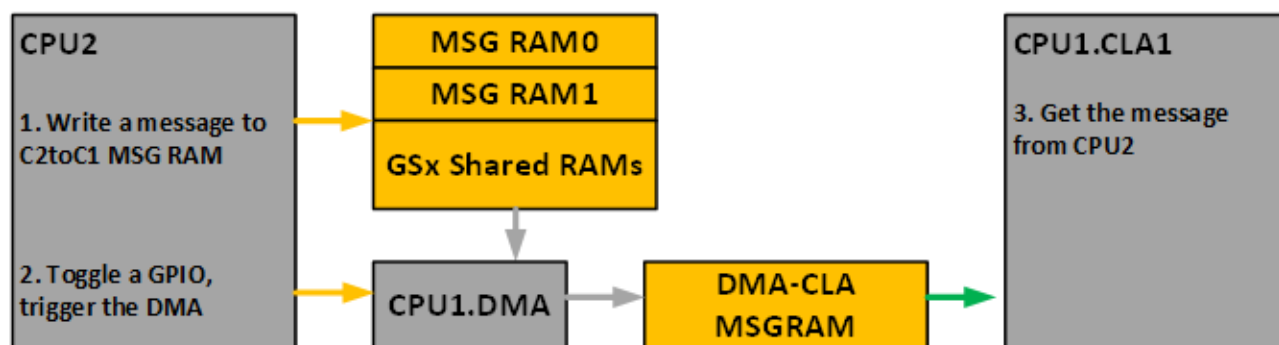


图 3-2. "IO 触发+DMA 搬运"实现 CPU2 和 CPU1.CLA1 之间数据交互

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 4-1 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。

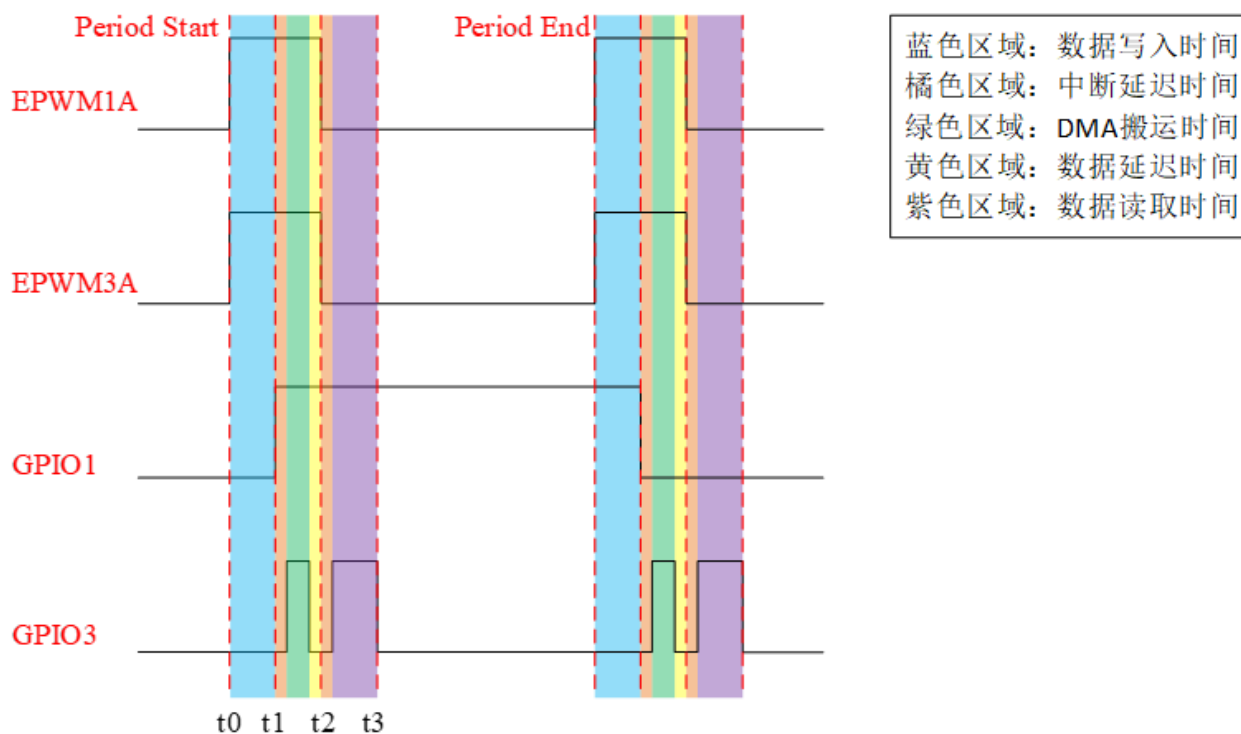


图 4-1. IO 触发 + DMA 搬运的时序图

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。

```

106 //
107 __interrupt void INT_myEPWM1_ISR(void)
108 {
109     //GPIO_togglePin(myGPIO9);
110
111     //
112     // Initialize the data buffers
113     //
114     uint16_t i;
115     for(i = 0; i < 16; i++)
116     {
117         sourceData[i] = i;
118     }
119
120     //
121     //toggle GPIO
122     //
123     GPIO_togglePin(myGPIO1);
124
125     Debugger1 = 1;
126
127     //
128     // Clear INT flag for this timer
129     //
130     EPWM_clearEventTriggerInterruptFlag(EPWM1_BASE);
131

```

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。

```

282 //*****
283 //
284 // INTERRUPT Configurations
285 //
286 //*****
287 void INTERRUPT_init(){
288
289     // Interrupt Settings for INT_myCLA1
290     Interrupt_register(INT_myCLA1, &INT_myCLA1_ISR);
291     Interrupt_enable(INT_myCLA1);
292
293     // Interrupt Settings for INT_myDMA0
294     Interrupt_register(INT_myDMA0, &INT_myDMA0_ISR);
295     Interrupt_enable(INT_myDMA0);
296
297     // Interrupt Settings for INT_myEPWM3
298     Interrupt_register(INT_myEPWM3, &INT_myEPWM3_ISR);
299     Interrupt_enable(INT_myEPWM3);
300
301     // Interrupt Settings for INT_myGPIO1_XINT
302     Interrupt_register(INT_myGPIO1_XINT, &INT_myGPIO1_XINT_ISR);
303     Interrupt_enable(INT_myGPIO1_XINT);
304 }
305 //*****
306 //
307 // DMA Configurations
308 //
309 //*****
310 void DMA_init(){
311     DMA_initController();
312     myDMA0_init();
313 }
314
315 void myDMA0_init(){
316     DMA_setEmulationMode(DMA_EMULATION_STOP);
317     DMA_configAddresses(myDMA0_BASE, 5888, 53248);
318     DMA_configBurst(myDMA0_BASE, 9U, 1, 1);
319     DMA_configTransfer(myDMA0_BASE, 3U, 1, 1);
320     DMA_configWrap(myDMA0_BASE, 65535U, 0, 65535U, 0);
321     DMA_configMode(myDMA0_BASE, DMA_TRIGGER_XINT1, DMA_CFG_ONESHOT_ENABLE | DMA_CFG_CONTINUOUS_ENABLE | DMA_CFG_SIZE_16BIT);
322     DMA_setInterruptMode(myDMA0_BASE, DMA_INT_AT_END);
323     DMA_enableInterrupt(myDMA0_BASE);
324     DMA_disableOverrunInterrupt(myDMA0_BASE);
325     DMA_enableTrigger(myDMA0_BASE);
326     DMA_startChannel(myDMA0_BASE);
327 }
328 //*****

```

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

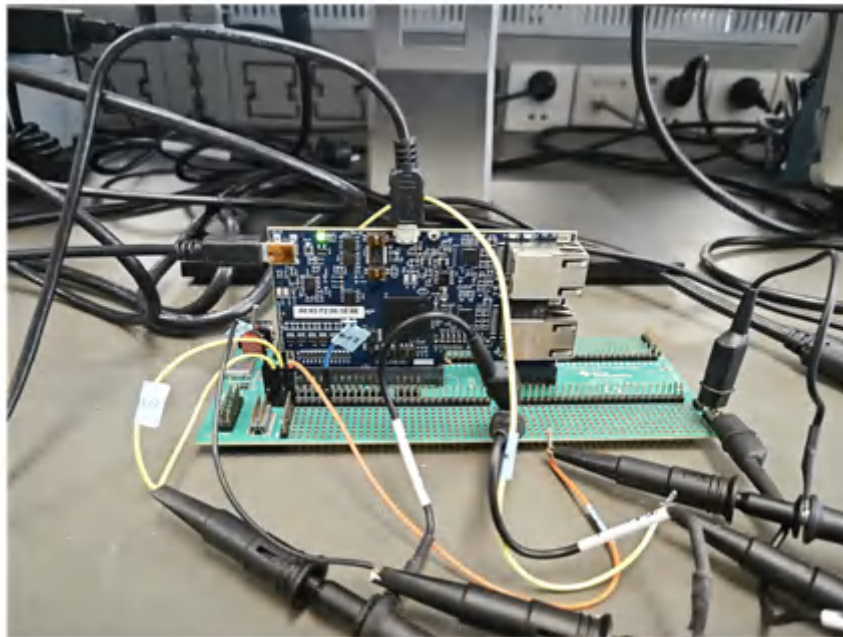


图 4-2. 实验环境，TMDSCNCD28388D + TMDSHSECDOCK

4.3 时序波形验证

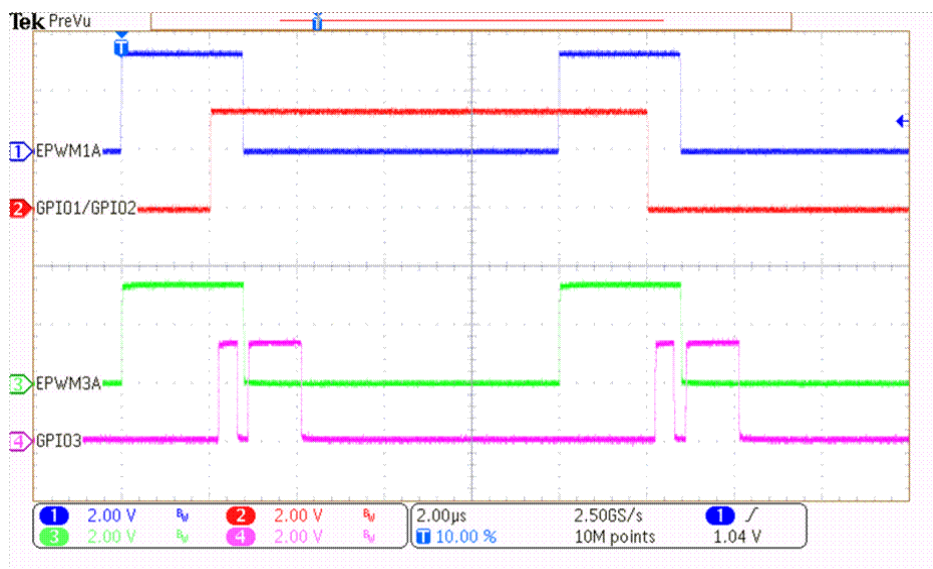


图 4-3. IO 触发 + DMA 搬运，实验波形

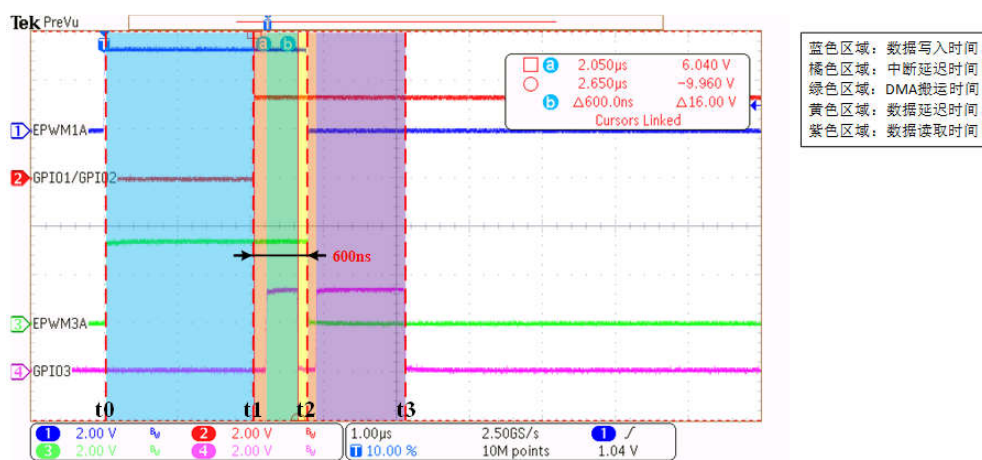


图 4-4. IO 触发 + DMA 搬运，实验时序

图 4-4 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(t_1 时刻)到 DMA 搬运结束(t_2 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。

6 参考文献

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任何明示或暗示的担保，包括但不限于对适销性、与某特定用途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安全、安保法规或其他要求。

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。对于因您对这些资源的使用而对 TI 及其代表造成的任何索赔、损害、成本、损失和债务，您将全额赔偿，TI 对此概不负责。

TI 提供的产品受 [TI 销售条款](#)、[TI 通用质量指南](#) 或 [ti.com](#) 上其他适用条款或 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。除非德州仪器 (TI) 明确将某产品指定为定制产品或客户特定产品，否则其产品均为按确定价格收入目录的标准通用器件。

TI 反对并拒绝您可能提出的任何其他或不同的条款。

版权所有 © 2026，德州仪器 (TI) 公司

最后更新日期：2025 年 10 月