Application Note

Implementation of Fast Data Interaction Between Multiple Cores in the TF2838xD Processor

Strong Zhang, Peter Ban

ABSTRACT

This document was translated from a simplified Chinese source. (ZHCAFM7)

In multi-core processor systems, the timeliness and accuracy of data transfers between cores are critical for realizing real-time control systems. In the 2838xD device, the absence of shared RAM space between CPU2 and CPU1.CLA1 poses a challenge for achieving fast data interaction between them. This paper analyzes the limitations of conventional IPC-based data transfer methods and, by leveraging the existing peripheral resources of the 28388D, proposes an "IO trigger + DMA transfer" approach to enable real-time, high-speed data interaction between CPU2 and CPU1.CLA1. Furthermore, this method is also applicable to processors with architectures similar to the 2838xD, such as the F28P65, facilitating fast inter-core data interaction in multi-core processors.

Table of Contents

List of Figures

Copyright © 2026 Texas Instruments Incorporated

1 Introduction and Functional Block Diagram of TMS320F2838xD

The TMS320F2838xD is a high-performance C2000 product featuring a multi-core architecture. It integrates two 200MHz C28 cores, two 200MHz CLA cores, and an ARM core, enabling multi-axis motor control with EtherCAT support.

In a dual-axis servo control system, one typical application scenario assigns CPU1 for EtherCAT control, while CPU1.CLA and CPU2 handle motor control for separate axes. Since encoder decoding requires the CLB and SPI modules, CPU2 is responsible for receiving encoder signals from both axes. After CPU2 receives the encoder data, it must rapidly transmit the data to CPU1.CLA to ensure precise real-time motor control.

Figure 1-1 shows the functional block diagram of the TMS320F28388D. Shared Message RAM exists between CPU2 and CPU1, allowing fast data interaction via IPC-triggered synchronization commands. However, there is no shared RAM space between CPU1.CLA1 and CPU2. As a result, CPU1.CLA1 and CPU2 cannot access the same memory space via their respective data buses, making rapid data interaction between CPU1.CLA1 and CPU2 difficult to achieve.

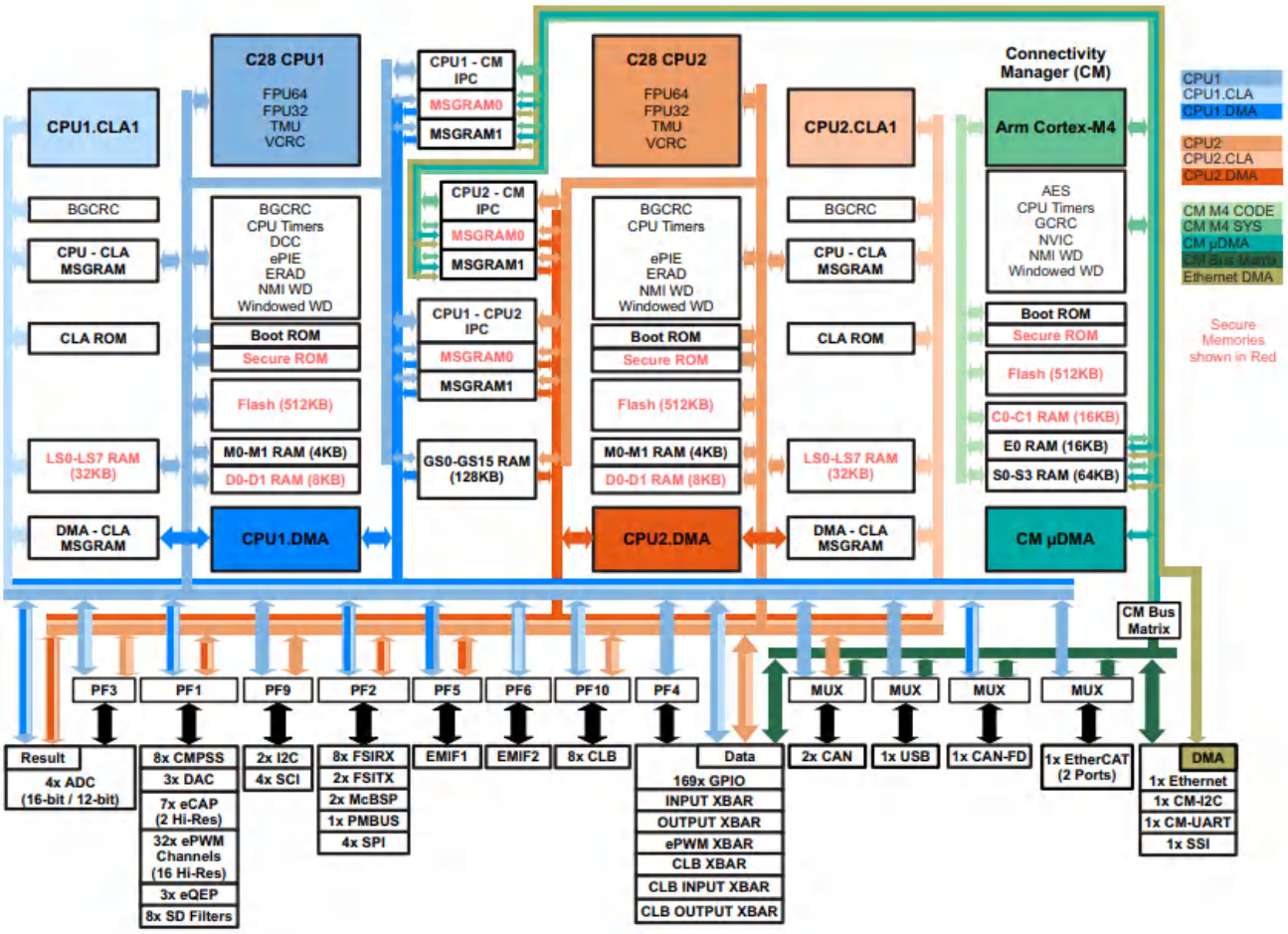

Figure 1-1. TMS320F28388D Functional Block Diagram — No Shared RAM Between CPU1.CLA1 and CPU2

Copyright © 2026 Texas Instruments Incorporated

2 Limitations of Data Transfer Between CPU2 and CPU1.CLA1 via the IPC Module

Since there is no shared RAM space between CPU2 and CPU1.CLA1, the conventional approach to enable data interaction between them is through the IPC module.

First, CPU2 can transfer data to MSGRAMx/GSx RAM via the IPC module (CPU1-CPU2 IPC). Then, CPU1 can use CPU1.DMA module to move the data from MSGRAMx/GSx RAM to the DMA-CLA1 MSGRAM. Finally, CPU1.CLA1 accesses the DMA-CLA1 MSGRAM to read the data and perform subsequent processing in CLA1.

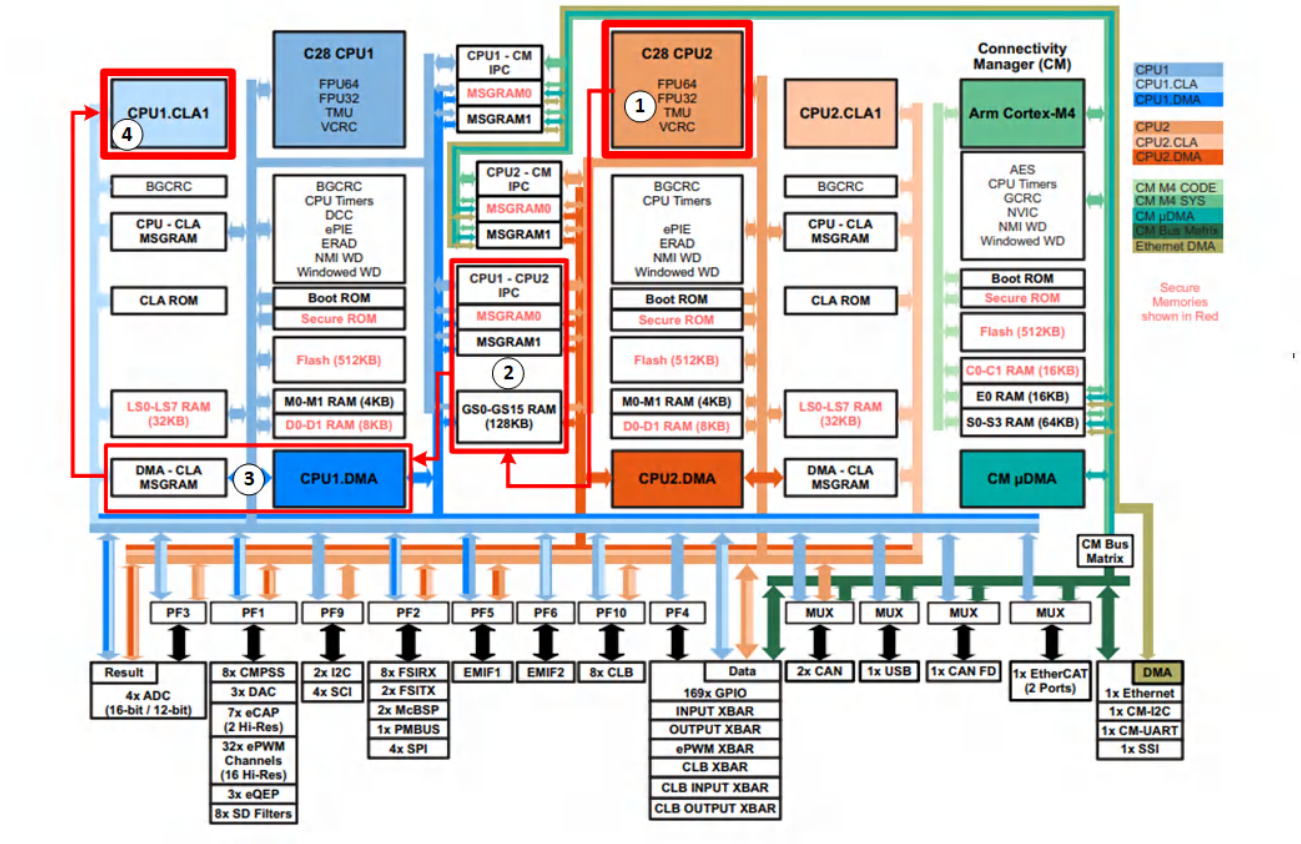

Figure 2-1. Data Flow: IPC-Based Data Interaction Between CPU2 and CPU1.CLA1

Figure 2-2 illustrates the specific procedure for data transfer between CPU2 and CPU1.CLA1 using the IPC module:

1. CPU2 first writes the data to be transferred to CLA1 into the MSG RAMx/GSx RAM.
2. CPU2 writes a 1 to the C2TOC1IPCSET register, setting the C2TOC1IPCFLG bit to 1. Since the C2TOC1IPCFLG register in CPU2 and the C2TOC1IPCSTS register in CPU1 map to the same physical address, the C2TOC1IPCSTS bit in CPU1 is also set to 1. This simultaneously triggers an IPC interrupt to CPU1, notifying CPU1 that the data is ready and that it can proceed to the next step.
3. CPU1 detects that the C2TOC1IPCSTS bit is set to 1.
4. CPU1 begins reading data from MSG RAMx/GSx RAM and simultaneously initiates a DMA task. Through CPU1's DMA function, the data is transferred from MSG RAMx/GSx RAM to the DMA-CLA1 MSGRAM.
5. CPU1 sets the C2TOC1IPCACK register to 1 to acknowledge receipt of the data. It then clears the flag bits in both C2TOC1IPCFLG and C2TOC1IPCSTS, completing the data transfer procedure.

Copyright © 2026 Texas Instruments Incorporated

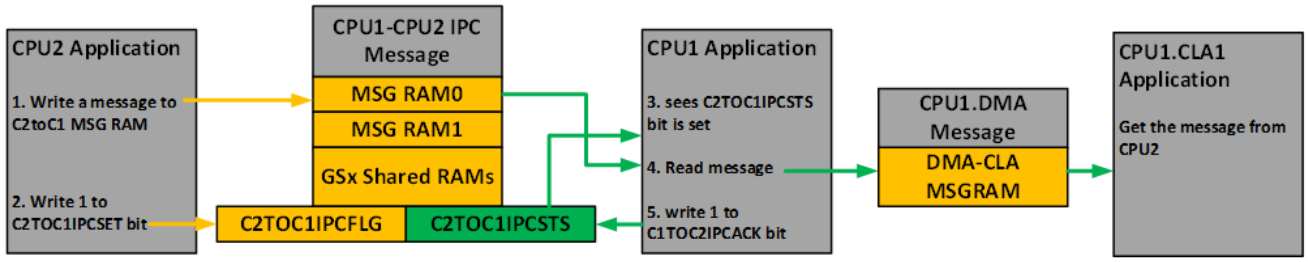

Figure 2-2. IPC-Based Data Interaction Between CPU2 and CPU1.CLA1

Based on the above analysis, since data interaction between CPU2 and CPU1 is implemented via the IPC module, CPU1 is required to perform steps 3 through 5 during the process, which adds significant overhead to CPU1. Therefore, when CPU1 is handling time-sensitive tasks or operating under heavy load, the IPC-based approach for data transfer between CPU2 and CPU1.CLA1 exhibits certain limitations.

As shown in Figure 2-3, the time required for CPU1 to complete steps 3 through 5 results in an overhead of approximately 2.93µs.

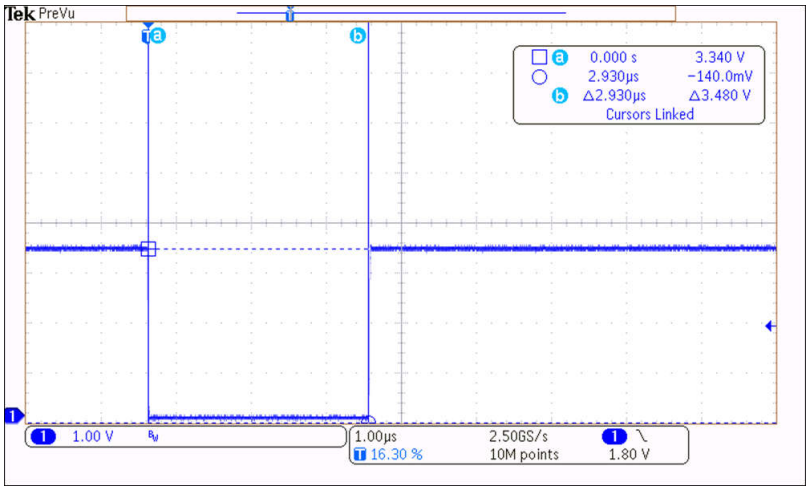

Figure 2-3. CPU1 Overhead Using the IPC Method

Copyright © 2026 Texas Instruments Incorporated

3 Principle of Data Transfer Between CPU2 and CPU1.CLA1 Using "IO Trigger + DMA Transfer"

To achieve faster data interaction, a method utilizing "IO trigger + DMA transfer" is proposed for data exchange between CPU2 and CPU1.CLA1. This approach enables data exchange between CPU2 and CPU1.CLA1 via IO-triggered DMA transfer, without imposing any overhead on CPU1.

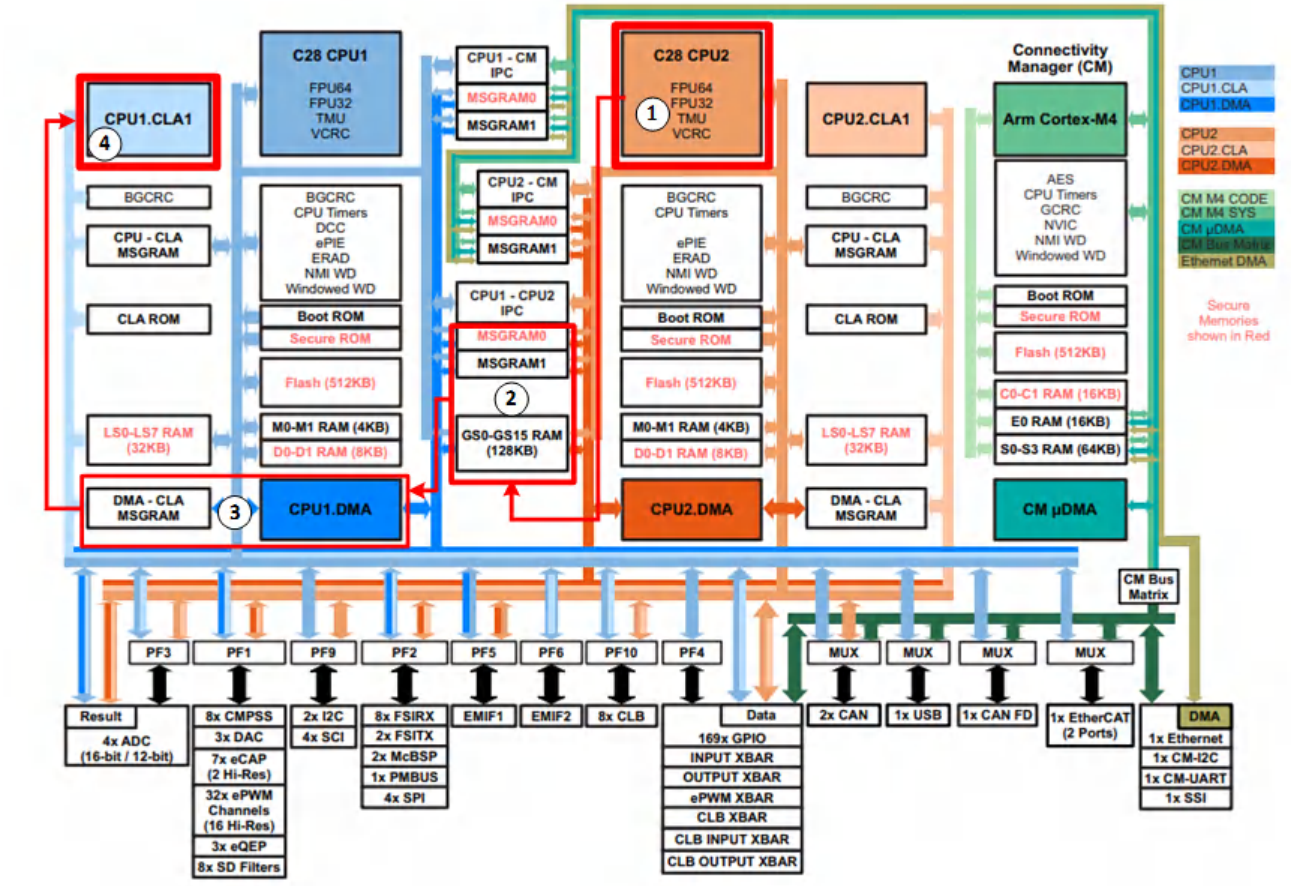

Figure 3-1. Data Interaction Between CPU2 and CPU1.CLA1 via "IO Trigger + DMA Transfer"

First, CPU2 writes the data to be transferred into GSx RAM. After each data write is completed, CPU2 toggles a GPIO pin. This GPIO toggle serves as the trigger source for CPU1.DMA, initiating the DMA transfer task of CPU1.DMA. Through CPU1.DMA, the data in GSx RAM is transferred to the DMA-CLA MSGRAM. Finally, CPU1.CLA1 reads the data from the DMA-CLA MSGRAM and performs subsequent processing in CLA1. Compared with the IPC-based approach, this method reduces CPU1's involvement and achieves faster data transfer.

Figure 3-2 illustrates the flowchart of the "IO trigger + DMA transfer" method, with the specific steps as follows:

1. CPU2 first writes the data to be transferred to CLA1 into MSG RAMx or GSx RAM.
2. After the data write is completed, CPU2 toggles an IO pin to serve as the trigger source for CPU1.DMA.
3. Upon being triggered, CPU1.DMA transfers the data from MSG RAMx/GSx RAM to the DMA-CLA1 MSGRAM
4. CPU1.CLA1 can directly read the data from the DMA-CLA1 MSGRAM and perform subsequent computations.

Copyright © 2026 Texas Instruments Incorporated

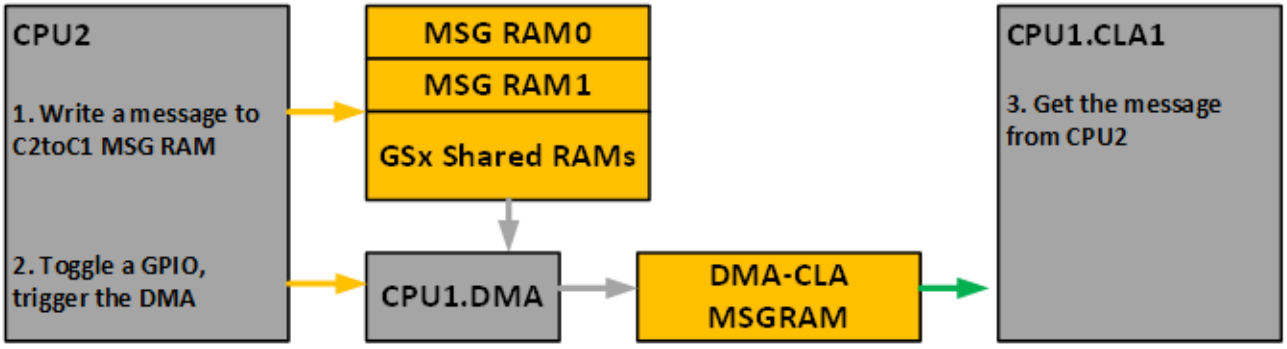

Figure 3-2. Data Interaction Between CPU2 and CPU1.CLA1 via "IO Trigger + DMA Transfer"

4 Verification of the "IO Trigger + DMA Transfer" Method

To better demonstrate the implementation of the proposed scheme, the following experiment was designed to validate the data transfer between CPU2 and CPU1.CLA1 using the "IO trigger + DMA transfer" method. The timing diagram of this demonstration is shown in Figure 4-1. To provide clearer insight into each stage of the data transfer process, an additional GPIO3 signal is introduced to indicate the start and end of each phase.

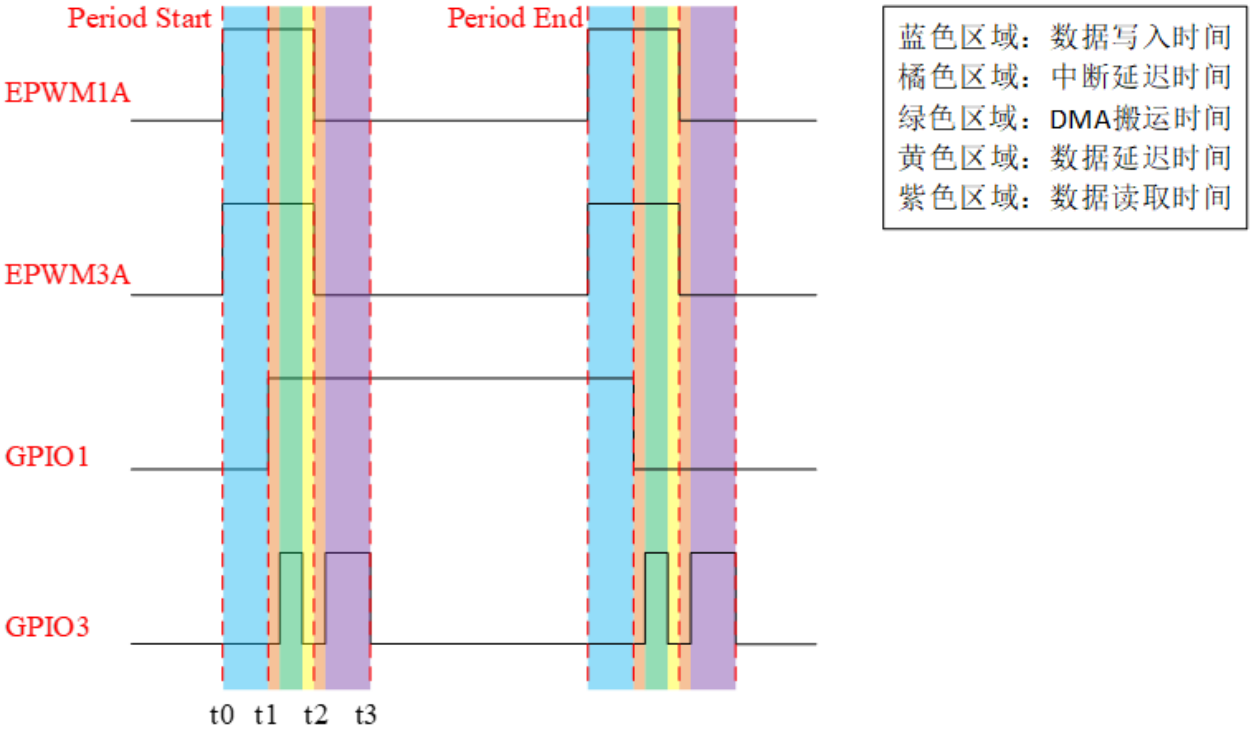

Figure 4-1. Timing Diagram of IO Trigger + DMA Transfer

Copyright © 2026 Texas Instruments Incorporated

4.1 Timing Description and Code Implementation

1. At time t0, the rising edge of EPWM1A triggers an interrupt service routine for the write task, which writes data into GS0RAM. At the end of this write task, the GPIO1 pin is toggled.

```
//
__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. At time t1, the toggle of GPIO1 generates an external interrupt, which serves as the trigger source for the DMA transfer.

```
//*****************************************************************************
//
// 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. At time t2, the falling edge of EPWM3A triggers the CLA task. CPU1.CLA1 then reads the data, and the read operation concludes at time t3.

Copyright © 2026 Texas Instruments Incorporated

```
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 Experimental Setup

Based on the aforementioned experimental design, verification was performed using TI's development boards: TMDSCNCD28388D and TMDSHSECDOCK.

Test Equipment:

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

Connections:

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

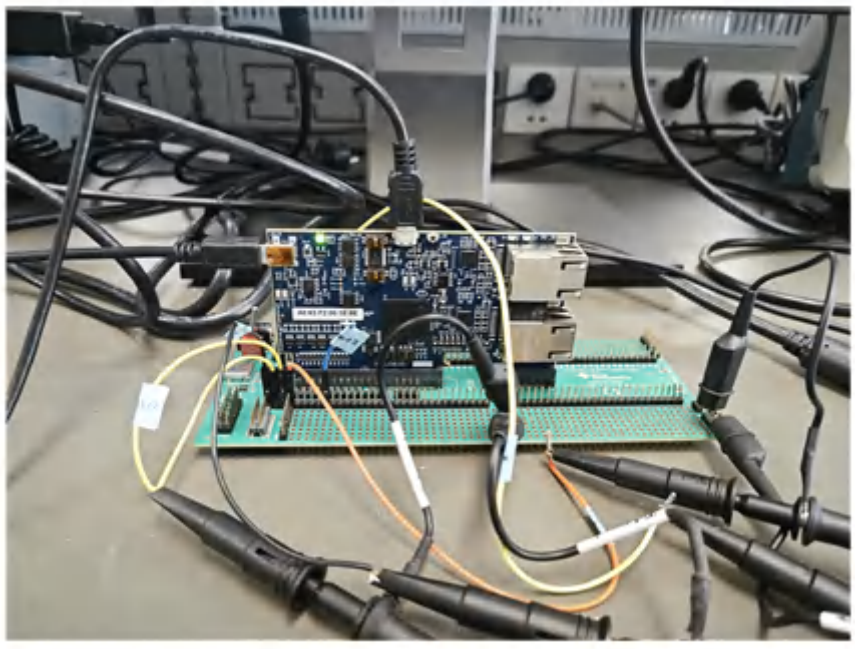

Figure 4-2. Experimental Environment: TMDSCNCD28388D + TMDSHSECDOCK

Copyright © 2026 Texas Instruments Incorporated

4.3 Timing Waveform Verification

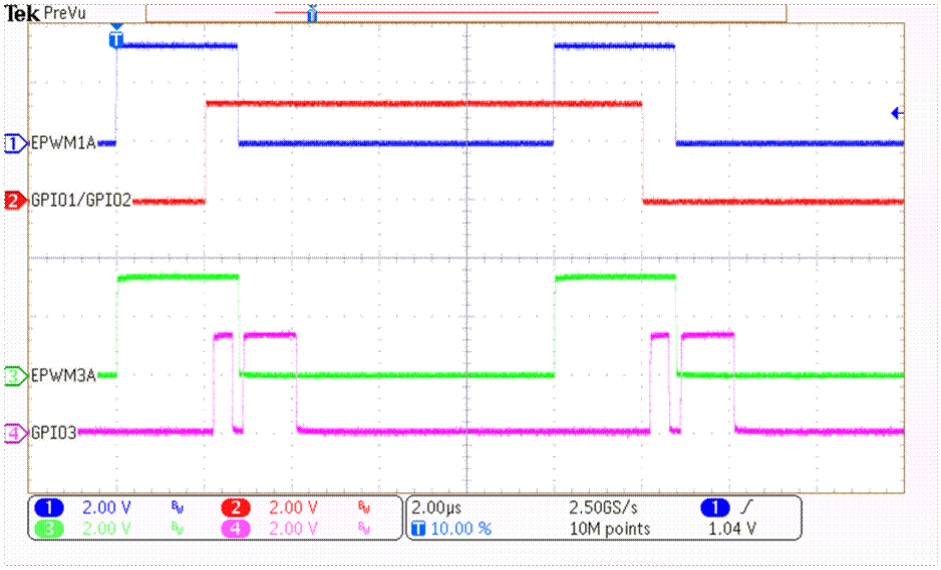

Figure 4-3. IO Trigger + DMA Transfer — Experimental Waveforms

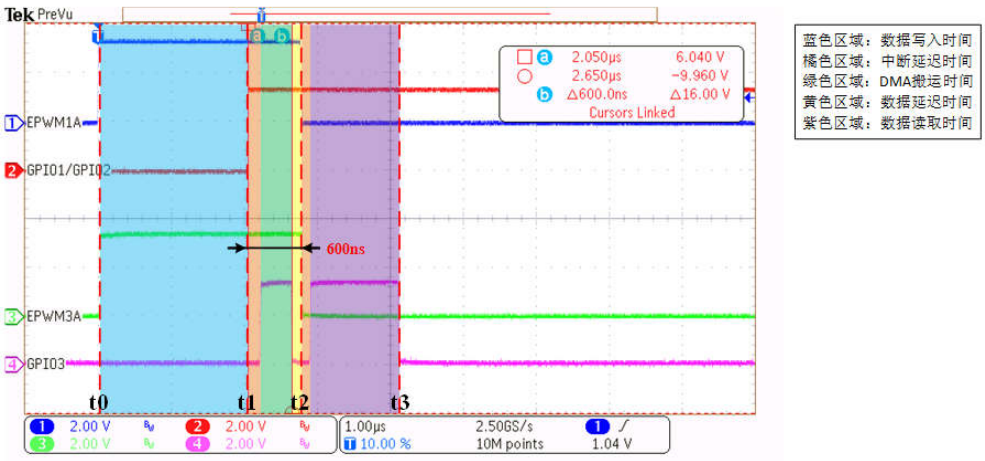

Figure 4-4. IO Trigger + DMA Transfer — Experimental Timing

Figure 4-4 illustrates the timing verification waveforms for data transfer between CPU2 and CPU1.CLA1 using the "IO trigger + DMA transfer" method. From the completion of data writing (time t1) to the end of the DMA transfer (time t2), the duration is approximately 600ns. Compared with the IPC approach, this method achieves the system's fast data transfer requirements without increasing CPU1 overhead.

Copyright © 2026 Texas Instruments Incorporated

5 Summary

To address the requirement for fast data interaction between CPU2 and CLA1 in the 28388D device, this work proposes an innovative method that leverages existing on-chip peripherals by utilizing GPIO-triggered DMA interrupts. This approach effectively resolves the difficulties and latency issues associated with data exchange between CPU2 and CPU1.CLA1. By employing available resources, the method fulfills customer requirements. The proposed concept is not only applicable to the 28388D/28385D but can also be extended to solve inter-core data interaction challenges in other multi-core MCUs with similar architectures.

6 References

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

Copyright © 2026 Texas Instruments Incorporated

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you fully indemnify TI and its representatives against any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale, TI’s General Quality Guidelines, or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products. Unless TI explicitly designates a product as custom or customer-specified, TI products are standard, catalog, general purpose devices.

TI objects to and rejects any additional or different terms you may propose.

Copyright © 2026, Texas Instruments Incorporated

Last updated 10/2025