Application Report
SPRA818 - June 2002

TMS320VC5471/TMS320VC5470 Inter-Processor Communication

Denise Ombres and Jack Rosenzweig — *Software Development Systems*

ABSTRACT

The TMS320VC547x (VC547x) generation of devices, including the TMS320VC5470 and the TMS320VC5471, has a master TMS470R1x (ARM7TDMIE) microcontroller unit (MCU) CPU with a slave TMS320C54x™ (C54x™) digital signal processor (DSP) CPU. Inter-processor communication (IPC) between the two processors can be performed using various methods. This document describes an interrupt-driven method of IPC using the ARM™ Port Interface (API) shared memory. Accompanying this application note is a code example consisting of two simple programs: one for the MCU, and one for the DSP.

Contents

List of Figures

List of Tables

1 Inter-Processor Communication on a VC547x Device

This document describes the process by which the TMS470R1x (ARM7TDMIE) MCU processor and the C54x DSP processor use interrupts to synchronize access to shared data in API memory.

Accompanying this application note is a working example of a system consisting of two simple programs: one for the MCU, and one for the DSP. The MCU program writes a 32-bit integer to shared memory, and triggers an interrupt on the DSP. After the DSP fields the interrupt, it performs a simple calculation using the value passed, and writes the computed result back to shared memory. The DSP then triggers an interrupt on the MCU to signal that the result is ready. The MCU reads and verifies the result.

1.1 Supported Configurations

Supported configurations include the following:

- Spectrum Digital TMS320VC5470/5471 evaluation module (EVM) board
- Both big- and little-endian ordering of memory on the MCU is supported
- Code Composer Studio™ Integrated Development Environment (IDE) v2.1 for the OMAP™ platform
- Microsoft™ Visual C++™ Studio v6.0 (optional)

1.2 Structure of the IPC Example

The IPC example is based on the bootloader example associated with the *TMS320VC5470/TMS320VC5471 Bootloader* application report (literature number SPRA376). Refer to this document for details on how to build and execute this example. The IPC example consists of three Code Composer Studio projects: **dsp_boot, dsp_proj** and **arm_boot**. The **dsp_proj** and the **arm_boot** projects in the IPC example are modified to support triggering and fielding interrupts. The **dsp_boot** project is unchanged.

The purpose of each project is as follows:

- **dsp_proj** – a simple application that computes the factorial of a number; communicates with the ARM via shared memory and interrupts.
- **dsp_boot** – the DSP portion of the bootloader that copies the **dsp_proj** program into DSP memory

Code Composer Studio and OMAP are trademarks of Texas Instruments.
Microsoft and Visual C++ are registered trademarks of Microsoft Corporation.

- **arm_boot** – an ARM application that boot loads the DSP with the program built by **dsp_proj,** passes data to it via shared memory and triggers its execution with interrupts.

This example also includes the Microsoft Visual C++ Studio project to build the converter program *OUT2BOOT*. Since the MCU and the DSP are different types of processors, the DSP code needs to be converted into a format usable by the MCU. To do this, the *OUT2BOOT* converter is used to convert the DSP Common Object File Format (COFF) file output into MCU C language header files, which declare arrays initialized with the contents of the COFF file.

1.3 MCU: Triggering an Interrupt on the DSP

To interrupt the DSP, the ARM sets APIC[2] (DSPINT). See function *interrupt_dsp()* in file "*arm_boot\main.c.*"

1.4 MCU: Receiving an Interrupt From the DSP

For a description of interrupt handling on the MCU see Chapter 4, Interrupt Handling, of the *TMS320VC547x CPU and Peripherals Reference Guide* (literature number SPRU038).

The ARM sets up a vector table of interrupt service routines. In the IPC example, the vector table is defined in file *arm_boot\intvectors_le_flash.c* . Function *c_irq()* in file *arm_boot\introutines.c* handles all incoming IRQ interrupts, including the API interrupt. Function *api_handler()* in the same file is the dedicated interrupt handler for the API interrupt.

The ARM enables the API interrupt by doing the following:

- Sets CPSR[7] (I) to enable IRQ interrupts
- Configures Interrupt Level Register 15 (ILR_IRQ_15) for the API interrupt. The example uses the following values: SENSE_EDGE = 1, PRIORITY = 0, FIQ = 0
- Clears IT_REG[15] (API) to clear all pending API interrupts
- Clears MASK_IT_REG[15] (API) to enable API interrupts

1.5 DSP: Receiving an Interrupt From the MCU

File *dsp_proj\intvect.s54* defines a vector table of interrupt service routines. Routine *api_isr()* is set up as the interrupt handler for the API interrupt.

The DSP sets up to receive the API interrupt from the ARM by doing the following:

- Globally disables all maskable interrupts by setting ST1[11] (INTM)
- Clears all pending API interrupts by setting IFR[9] (AINT)
- Enables the API interrupt by setting IMR[9] (AINT)
- Globally enables all maskable interrupts by clearing ST1[11] (INTM)

1.6 DSP: Triggering an Interrupt on the MCU

To interrupt the ARM, the DSP pulses BSCR[3] (HINT). See function *mcu_interrupt()* in file "dsp_proj\main.c."

1.7 IPC Example Execution Flowchart

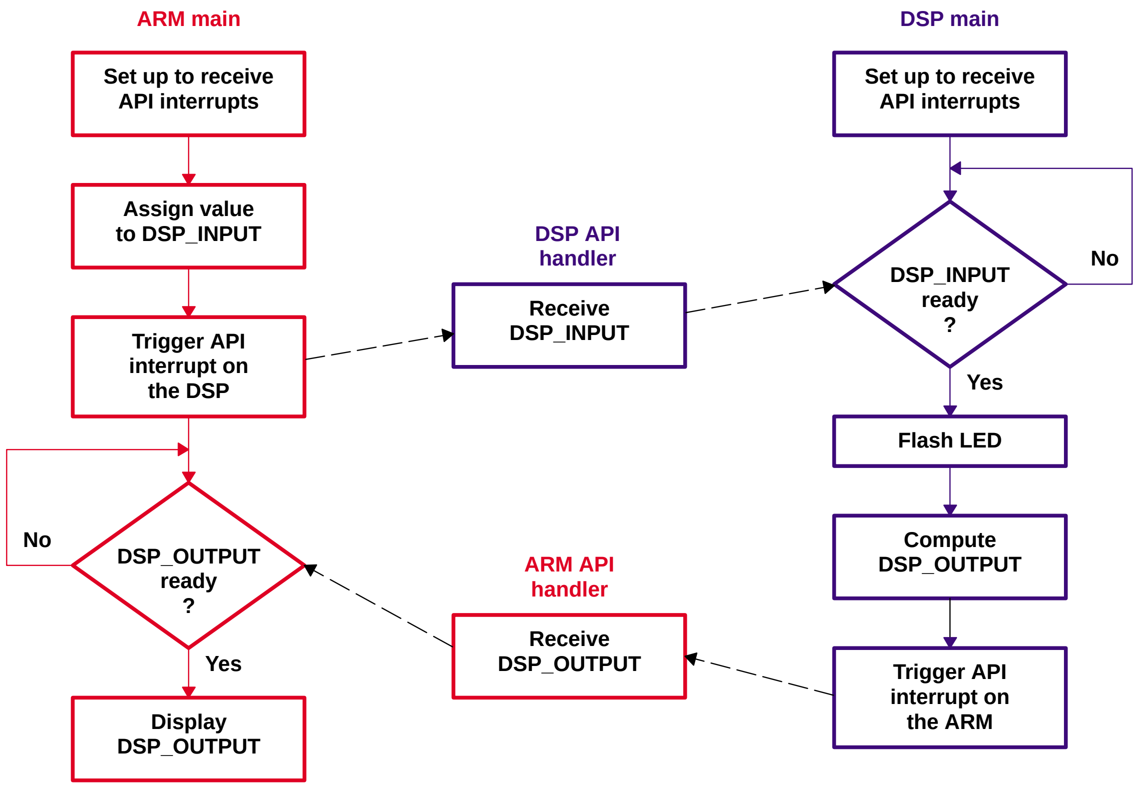


Figure 1. IPC Example Execution Flowchart

1.8 Building and Executing the IPC Example

Listed below are the Microsoft Visual C++ Studio and Code Composer Studio projects which build the individual components of this example.

NOTE: The compressed file which accompanies this application report, *spra818.zip*, should be copied to the Code Composer Studio *<drive>:\ti\myprojects* directory and extracted there into *<drive>:\ti\myprojects\c547x_ipc.* All directories referenced below and in the following build example are relative to the c547x_ipc directory.

- .**\out2boot:** Microsoft Visual C++ Studio project which builds the *out2boot.exe* conversion utility. This tool converts a DSP COFF file into an ARM C-language Header. The source code is provided, as well as the executable*, .\out2boot\out2boot.exe*.
- .**\dsp_proj:** Code Composer Studio project which builds a simple application that computes the factorial of a number, and communicates with the ARM via shared memory and interrupts.
- .**\dsp_boot:** Code Composer Studio project which builds the C54x-side bootloader program, *dsp_boot.out.* This program copies the DSP application from API memory to its run-time location in memory. This project should not be modified.
- .**\arm_boot:** Code Composer Studio project which builds the ARM-side bootloader program, *arm_boot_le_flash.out* for little-endian or *arm_boot_be_flash.out* for big-endian*.* This

program's job is to initialize the device and to copy the DSP bootloader and the DSP application into API memory. Once this is completed, the ARM main program passes data to the DSP application via shared memory and triggers its execution with interrupts.

1.8.1 Building the IPC Example

To build the IPC example, perform the following steps:

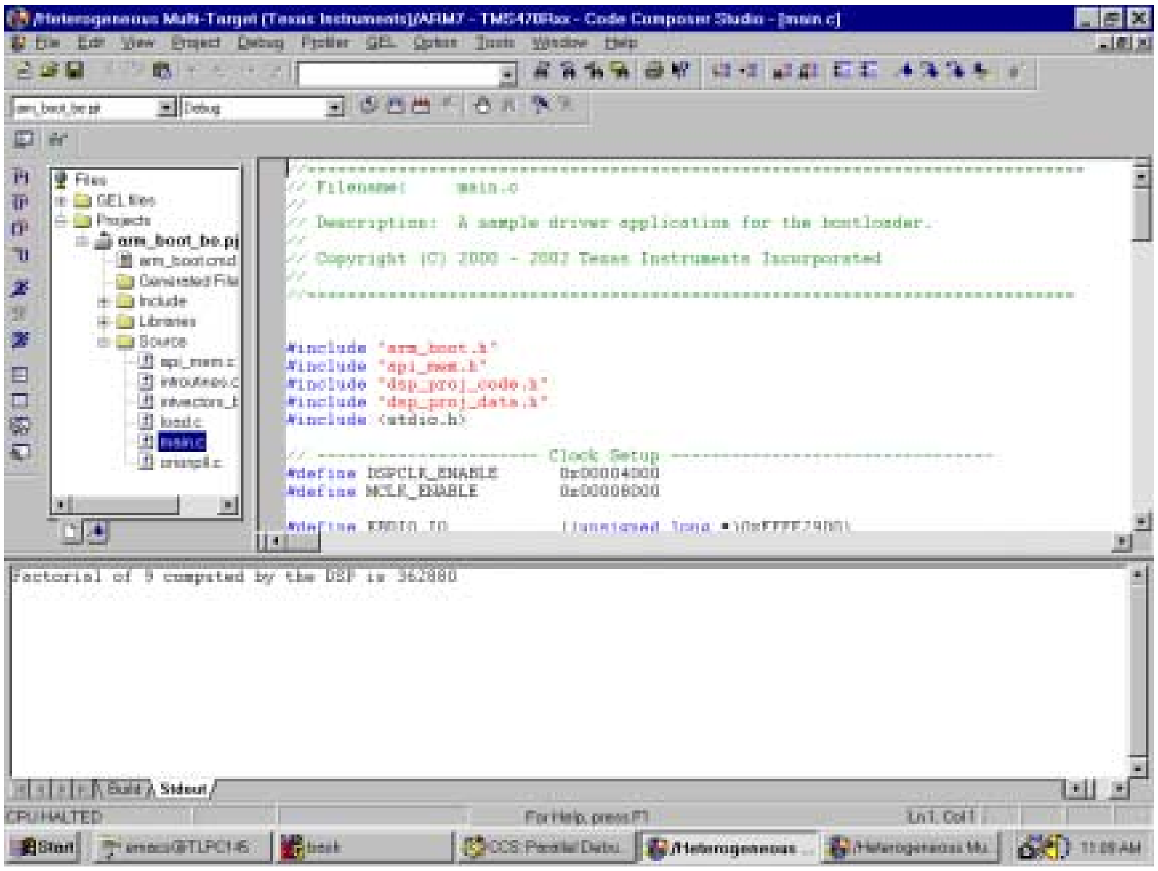
1. If needed, use the Microsoft Visual C++ Studio workspace file, *.\out2boot\out2boot.dsw,* to build the conversion utility **.\out2boot\out2boot.exe.** Otherwise, just use the existing executable and skip this step.
2. Configure Code Composer Studio using provided Code Composer Studio configuration files:
   - Run Code Composer Studio Setup program
   - Choose File –> Import menu option.
   - Click *Advanced* button.
   - Browse to the *.\config* directory, and select a .Code Composer Studio file that is appropriate for your memory setup. Different configuration files are provided to support various memory layouts. For example, choose *c5471_le_ramlow.ccs* for little-endian, RAM-low memory.
3. Using the C54x Code Composer Studio project file, *.\dsp_proj\dsp_proj.prj,* build the sample DSP application **.\dsp_proj\dsp_proj.out**. The post-link step of this project will invoke the out2boot utility on the **dsp_proj.out** image, and copy the resulting C-header files, *dsp_proj_code.h* and *dsp_proj_data.h,* into the. *\arm_boot* directory.
4. Using the C54x Code Composer Studio project file, *.\dsp_boot\dsp_boot.prj,* build the DSP bootloader program **.\dsp_boot\dsp_boot.out**. The post-link step of this project will invoke the out2boot utility on the **dsp_boot.out** image, and copy the resulting C-header file, *dsp_boot_code.h*, into the. *\arm_boot* directory.
5. Choose the ARM Code Composer Studio project file appropriate for your memory byte ordering, *.\arm_boot\arm_boot_le.prj* or *.\arm_boot\arm_boot_be.prj,* and use it to build the ARM bootloader program, **.\arm_boot\arm_boot_le_flash.out** for little-endian or **.\arm_boot\arm_boot_be_flash.out** for big-endian. When switching between big- and little-endian modes, do a *Project –> Rebuild All* within Code Composer Studio because most of the files within each project are the same and will not be recompiled with a *Project –> Build*.

1.8.2 Executing the IPC Example

To execute the bootloader example, perform the steps below. (NOTE: All directories referenced below are relative to the Code Composer Studio root installation directory and project subdirectory.)

1. On the C5471 EVM board, configure JP19 for big-endian (1–2) or little-endian (2–3) mode. Also, ensure that JP20 is set to the 32-bit ROM (Flash) Size (1–2). The arm_boot code examples included with the bootloader use ARM 32-BIS. The chosen byte ordering and ARM instruction set must match those followed during the build of the bootloader example.

2. Start Code Composer Studio. This should run the Parallel Debug Manager.
3. From the Parallel Debug Manager, open ARM Code Composer Studio.
4. From the Parallel Debug Manager, open C54x Code Composer Studio.
5. In the C54x Code Composer Studio debug window, make sure that the DSP is in the running state. This will be indicated at the bottom of the window. If it is not, then execute Debug –> Reset CPU, followed by Debug –> Run Free.
6. In the ARM Code Composer Studio debug window, load the executable: **.\arm_boot\arm_boot_le_flash.out** for little-endian or **.\arm_boot\arm_boot_be_flash.out** for big-endian.
7. In the ARM Code Composer Studio debug window, execute Debug –> Run. The DS1 LED should blink nine times and the value of factorial of nine should be displayed in the ARM Code Composer Studio output window (see Figure 2).



```
//==========================================================================
// Filename:     main.c
//
// Description:  A sample driver application for the bootloader.
//
// Copyright (C) 2000 - 2002 Texas Instruments Incorporated
//
//==========================================================================


#include "arm_boot.h"
#include "api_mem.h"
#include "dsp_proj_code.h"
#include "dsp_proj_data.h"
#include <stdio.h>

// ------------------------ Clock Setup --------------------------------
#define DSPCLK_ENABLE         0x00004000
#define MCLK_ENABLE           0x00008000

#define ENDIO_IO              (unsigned long *)(0xFFFF7900)
```

```
Factorial of 9 computed by the DSP is 362880
```

Figure 2. ARM Code Composer Studio Session Showing the Output of the IPC Example

2 References

1. *TMS320C54x DSP Reference Set, Volume 1: CPU and Peripherals* (SPRU131)
2. *TMS320VC547x CPU and Peripherals Reference Guide* (SPRU038)
3. *TMS320VC5471 Fixed-Point Digital Signal Processor Data Manual (SPRS180)*
4. *TMS320VC5470 Fixed-Point Digital Signal Processor Data Manual (SPRS017)*
5. *TMS320C54x Assembly Language Tools User’s Guide* (SPRU102)
6. *TMS470R1x User’s Guide (SPNU134)*
7. *TMS470R1x Assembly Language Tools User’s Guide* (SPNU118)
8. *TMS320C54x Evaluation Module Technical Reference* (SPRU135)
9. *TMS320VC5470/TMS320VC5471 Bootloader* (SPRA376)

Appendix A MCU and DSP Registers

31–24
Reserved

23–16
Reserved

15	14	13	12	11	10	9	8
IRQ15	IRQ14	IRQ13	IRQ12	IRQ11	IRQ10	IRQ9	IRQ8
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

7	6	5	4	3	2	1	0
IRQ7	IRQ6	IRQ5	IRQ4	IRQ3	IRQ2	IRQ1	IRQ0
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

NOTE: R – Read access; W – Write access; Value following dash (–) = value after reset

Figure A–1. MCU Interrupt Register (IT_REG)

31–24
Reserved

23–16
Reserved

15	14	13	12	11	10	9	8
IRQ_15_MSK	IRQ_14_MSK	IRQ_13_MSK	IRQ_12_MSK	IRQ_11_MSK	IRQ_10_MSK	IRQ_9_MSK	IRQ_8_MSK
R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1

7	6	5	4	3	2	1	0
IRQ_7_MSK	IRQ_6_MSK	IRQ_5_MSK	IRQ_4_MSK	IRQ_3_MSK	IRQ_2_MSK	IRQ_1_MSK	IRQ_0_MSK
R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1

NOTE: R – Read access; W – Write access; Value following dash (–) = value after reset

Figure A–2. MCU Mask Interrupt Register (MASK_IT_REG)

31 … 24
Reserved

23 … 16
Reserved

15 … 8
Reserved

7 … 6	5	4 … 1	0
Reserved	**SENSE_ EDGE**	**PRIORITY**	**FIQ**
	R/W-0	R/W-0	R/W-0

NOTE: R – Read access; W – Write access; Value following dash (–) = value after reset

Figure A–3. MCU Interrupt Level Register 15 (ILR_IRQ_15)

15 … 8
Reserved
R-0

7 … 4	3	2	1	0
Reserved	**HINT**	**DSPINT**	APIMODE	Reserved
R-0	R-0	W-0	R-0	

NOTE: R – Read access; W – Write access; Value following dash (–) = value after reset

Figure A–4. MCU API Control Register (APIC)

31	30	29	28	27	24
N	Z	C	V	Reserved	

23	16
Reserved	

15	8
Reserved	

7	6	5	4	3	2	1	0
I	F	T	M4	M3	M2	M1	M0

Figure A–5. MCU Current Program Status Register (CPSR)

15	12	11	10	8
BNKCMP		PS–DS	Reserved	
R/W-1		R/W-1	R/W-0	

7	5	4	3	2	1	0
Reserved		ABMDIS	**HINT**	APIMODE	Reserved	EXIO
R/W-0		R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

NOTE: R – Read access; W – Write access; Value following dash (–) = value after reset

Figure A–6. DSP Bank-Switching Control Register (BSCR)

15	14	13	12	11	10	9	8
Reserved		DMAC5	DMAC4	BXINT1 DMAC3	BRINT1 DMAC2	**AINT**	Reserved
R/W-0		R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

7	6	5	4	3	2	1	0
Reserved		BXINT0	BRINT0	TINT	Reserved		INT0
R/W-0		R/W-0	R/W-0	R/W-0	R/W-0		R/W-0

NOTE: R – Read access; W – Write access; Value following dash (–) = value after reset

Figure A–7. DSP Interrupt Flag Register (IFR)

15	14	13	12	11	10	9	8
Reserved		DMAC5	DMAC4	BXINT1 DMAC3	BRINT1 DMAC2	**AINT**	Reserved
R/W-0		R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

7	6	5	4	3	2	1	0
Reserved		BXINT0	BRINT0	TINT	Reserved		INT0
R/W-0		R/W-0	R/W-0	R/W-0	R/W-0		R/W-0

NOTE: R – Read access; W – Write access; Value following dash (–) = value after reset

Figure A–8. DSP Interrupt Mask Register (IMR)

Table A–1. IFR and IMR Register Bit Fields

BIT		Function
No.	**Name**	
15–14	Reserved	Reserved
13	DMAC5	DMA channel 5 interrupt flag/mask bit.
12	DMAC4	DMA channel 4 interrupt flag/mask bit.
11	BXINT1/DMAC3	McBSP1 transmit or DMA Channel 3 interrupt flag/mask bit.
10	BRINT1/DMAC2	McBSP1 receive or DMA Channel 2 interrupt flag/mask bit.
9	AINT	ARM-to-C54x (MCU-to-DSP) interrupt flag/mask bit.
8	Reserved	Reserved for future expansion.
7	DMAC1	DMA Channel 1 interrupt flag/mask bit.
6	DMAC0	DMA Channel 0 interrupt flag/mask bit.
5	BXINT0	McBSP0 transmit interrupt flag/mask bit.
4	BRINT0	McBSP0 receive interrupt flag/mask bit.
3	TINT	Timer interrupt flag/mask bit.
2	Reserved	Reserved
1	Reserved	Reserved
0	INT	External interrupt flag/mask bit.