

# Interrupts in Decoupled Parallel Mode for MPC5675K Configuration and Usage

by: **Tomas Kulig**  
Automotive and Industrial Solutions Group

## 1 Introduction

This application note describes how to use interrupts modules in Decoupled Parallel mode (DPM) on the MPC5675K microcontroller unit (MCU), which is targeted for the Chassis and Safety market. The MPC5675K has two independent interrupt controller modules (INTC) in DPM, one for each core. One of the sections of this application note provides an example project which shows interrupt handling from analog-to-digital converters (ADCs). There are two different ways in which the software can handle interrupts in DPM. One of the ways is using different ADCs for each core (independent interrupts) and the second one uses the same interrupt from one ADC by both the cores (shared interrupt). The application uses an interrupt software vector mode, and is developed in GreenHills compiler.

The key words for this application note are DPM, Lock Step mode (LSM), and interrupts.

## 2 Interrupts

### Contents

1	Introduction.....	1
2	Interrupts.....	1
2.1	Definition of interrupts.....	1
2.2	DPM versus LSM.....	2
2.3	Software versus Hardware Vector mode.....	3
2.4	Interrupts—Description of sequence during handling of ISR.....	3
3	Description of using files and important settings.....	5
3.1	Different interrupts.....	6
3.2	Shared interrupts.....	7
4	References.....	8

## 2.1 Definition of interrupts

The interrupts signalize an event, or refer to some changes in the peripherals such as the end of conversation from ADC, and direct memory access (DMA) transfer completed. The main advantage of the interrupt service routine (ISR) is the independence on core. For example, consider a situation where the user wants the ADC to start as soon as the DMA transfer is completed. In this case, the core does not have to continuously check the state of DMA transfer but it is enough to enable interrupt request from DMA transfer completed and if it occurs, the ISR which starts ADC, will begin. The sequence of interrupt service for external interrupts which is always executed, is shown in [Figure 1](#). Both the cores have the same sources of interrupts with some peripherals being shared by both the cores and some being used individually for each core. External interrupts are those which are not generated inside the core, but enter the core from external sources, which in this case, are peripherals. There are 35 core interrupts and each has its own interrupt handler. Only IVOR4 handler is used for external interrupts while all the other handlers are used only for internal events as described in the table below.

**Table 1. Core interrupt**

IVORn	Interrupt type
None	System reset (not an interrupt)
0	Critical input
1	Machine check
2	Data storage
3	Instruction storage
4	External input
5	Alignment
6	Program
7	Floating-point unavailable
8	System call
9	APU unavailable (not used by this core)
10	Decrementer
11	Fixed-interval timer
12	Watchdog timer
13	Data TLB error
14	Instruction TLB error
15	Debug
16-31	Reserved
32	SPE unavailable
33	SPE data exception
34	SPE round exception

## 2.2 DPM versus LSM

The MPC5675K MCU supports the following two modes:

- LSM: This mode is used for safety applications where both the cores run with the same code and their outputs are compared.
- DPM: In this mode, the cores run separately with independent code.

The performance of DPM mode is 60% more than that of the LSM mode. Some peripherals such as ADCs, Enhanced Direct Memory Access 0 (EDMA0), EDMA1, and so on, are shared by both the cores whereas some peripherals like INTC, and System Timer Module (STM) are dedicated for Core0 or Core1.

## 2.3 Software versus Hardware Vector mode

ISRs or handlers can be either software or hardware depending on their implementation in the microcontroller. The basic difference is in the method of operation. Hardware interrupts have their own dedicated prolog and epilog. These interrupts are meant for actions/functions which must be started in a very short time after the interrupt occurs. For example, consider a declaration variable named COUNTER. As soon as the interrupts occur, the value of variable COUNTER increases. For the Hardware Vector mode, the code will include following steps:

- Clear the flag,
- Increase the value of variable COUNTER,
- Return back from ISR into the application with value of priority of this application.

The main feature of the Hardware Vector mode is the faster reaction for interrupt request, because the handling code, which has only few instructions is saved and executed on the address given by following formula:

$$16 * (\text{number of interrupt request}) + \text{Interrupt Vector Prefix Register (IVPR)}$$

The constant 16 suggests that the space between two interrupt offset addresses is 16 bytes.

Software Vector mode has common prolog and epilog (see [Definition of interrupts](#)) for all interrupts. Each interrupt request has its own ISR. The list of ISR addresses is stored in the interrupt vector table. Each interrupt request can have different ISRs, but it is possible to use the same ISR for more than one interrupt request.

Each interrupt request has variable priorities set by the Priority Select Registers (PSRx). There are 16 levels of priorities from the lowest priority 0, to the highest priority 15.

## 2.4 Interrupts—Description of sequence during handling of ISR

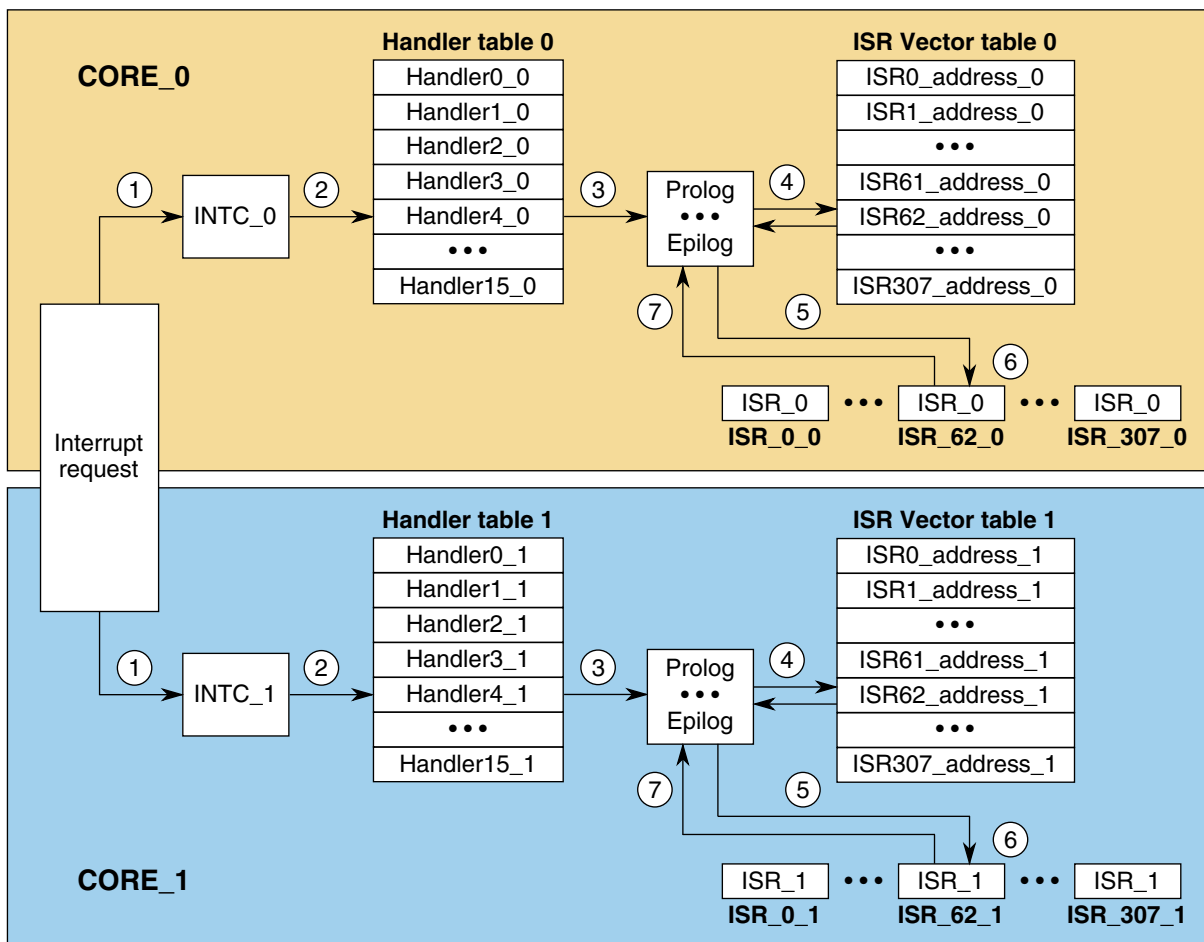
INTC records incoming interrupt events (number 1 in [Figure 1](#)) and arbitrates which incoming interrupts are serviced first and which are waiting. Those that have higher priority than the current running application or interrupt will be executed. Other interrupts are masked and will have to wait for handling until their priority is the highest. The servicing of interrupt will begin by code execution on IVOR4 handler address (number 2 in [Figure 1](#)), which serves the external interrupt for handling.

IVOR Handler 4 has two main parts. The first one is prolog (number 3 in [Figure 1](#)) and second one is epilog, but their functionality is inverted.

- Prolog creates space on stack to create backup for the work registers, loads address of interrupt vector table and the ISR, depending on interrupt request, and saves the returning address of function (number 4 in [Figure 1](#)). Interrupt vectors for all available interrupts are saved in the interrupt vector table. This table contains pointers to interrupt service functions. IVOR4 Handler goes to address of the ISR (number 5 in [Figure 1](#)). ISR for the interrupt request is executed (number 6 in [Figure 1](#)).
- Epilog restores the work registers and space on stack. The epilog is executed after interrupt service function is finished (number 7 in [Figure 1](#)). The last step is returning to the main function from which the ISR was called.

The description given above is executed for each core independently if the interrupt request is not from the same peripheral event, that is, the cores are running in parallel. The procedure of processing the shared interrupt request by both the cores is shown in [Figure 1](#). The servicing is identical as if different interrupts are used for each of the cores. It is possible because each core has its own INTC module which is working independently. The interrupt request enters the INTC modules of both

the cores at the same time. If the service routines have the same length, their handling will run for the same time because both the cores have the same system clock. Clearing of interrupt flag is very important at the start of service routine, otherwise, the interrupt handler would run over and over again. It is necessary to clear the interrupt flag by the core which has shorter interrupt handling, otherwise, the shorter ISR would be executed few times during the execution of the longer ISR. Therefore, it is recommended to clear the flags at the start of interrupt handling in the both the core routines.



**Figure 1. Interrupt service procedure in Software Vector mode**

The numbered labels in the above figure can be described as follows:

- 1: Interrupt request
- 2: Jump to IVOR4 external input interrupt.
- 3: Start prolog of IVOR4 external input interrupt.
- 4: Find ISR which is given by interrupt request in the interrupt vector table.
- 5: Go to ISR.
- 6: Run ISR.
- 7: Return back to IVOR4 Epilog and continue with interrupted code.

**CAUTION**

The important part of interrupt function is to clear the interrupt flag which caused interrupt. If the interrupt flag is not cleared, the interrupt function will run over and over again.

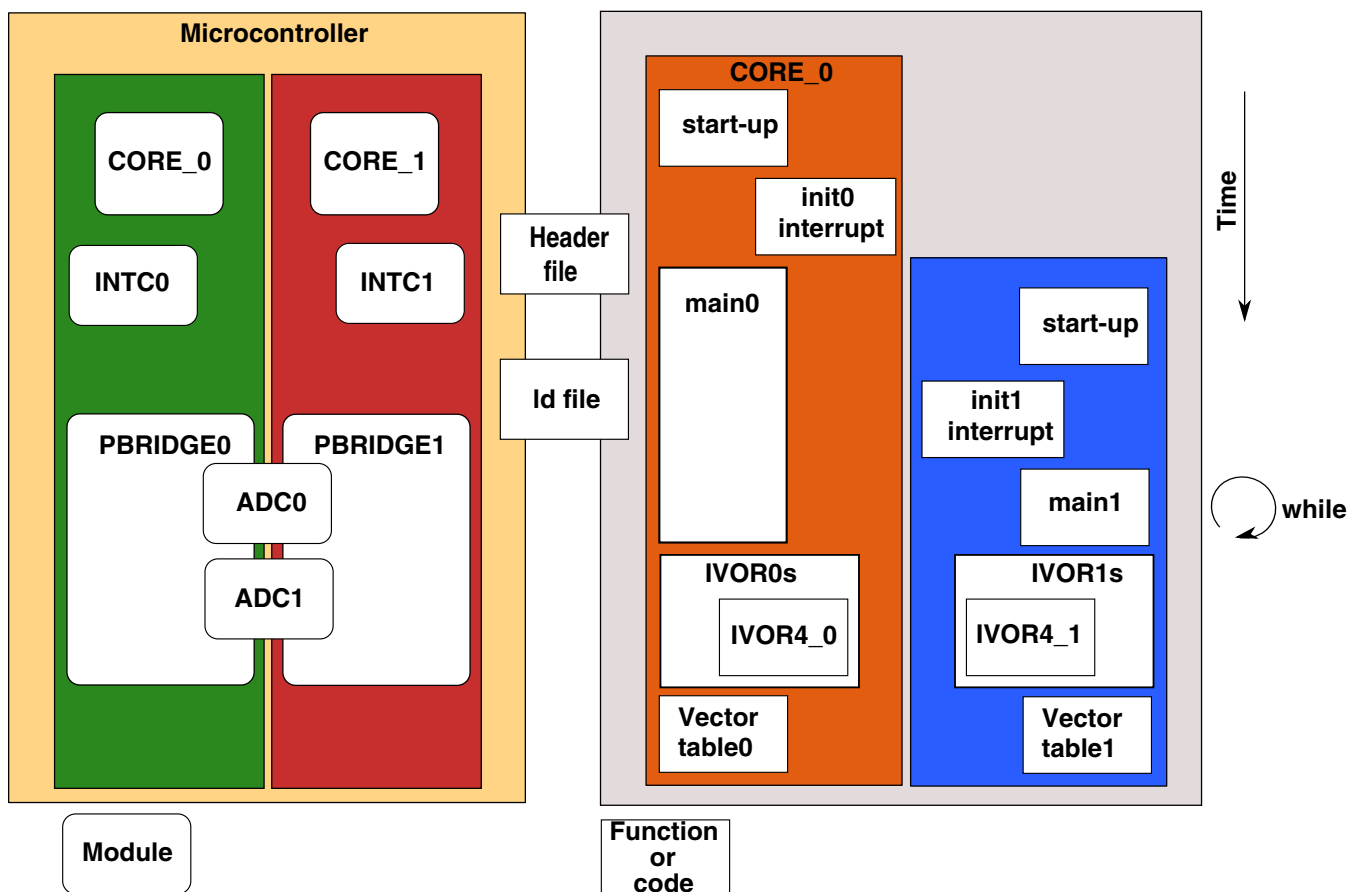
### 3 Description of using files and important settings

One project is chosen for both the cores in Multi 5.0 which creates only one Executable and Linkable Format file (ELF). The application independence of both the cores is given by the main function for each core—main0 for Core0, and main1 for Core1. This kind of project construction allows easy sharing of functions used on both the cores such as controlling ADCs, and driving light-emitting diodes. It must be noted that the application uses Software Vector mode so that each core can have its own interrupt vector table which is significant for the connection interrupt request and function for its handling. These functions can be shared between more interrupts of both the cores, such as solving peripheral errors which can lead to Safe mode. The next necessary part of work with interrupts is setting their priorities which default to 0 (the lowest priority) after reset. The values of priorities are required to be higher than the priority of current running process. At the end, the global interrupt must be allowed.

#### CAUTION

The default priority of the running application (main or other function) for both the cores is 15 after reset (the highest priority). This must be decreased so that the priority of used interrupt will be higher than the priority of running application.

The graphic interpretation of relationship among the used files is shown in [Figure 2](#). Header file and Linker files are the basic files for using a microcontroller and are common for both of its cores. The Header file (Header in [Figure 2](#)) allows entrance to the peripherals of a given microcontroller (mapping of peripheral). The Linker file defines sections in memory where the code is stored. These sections can have different length and start addresses, and use different type of memory. For example, the code variables are located in the SRAM memory and code is in flash memory. The interrupt handling routines for internal request of cores must be on the different location in memory for each core which is aligned to 64 KB boundary. Their addresses are saved in the Interrupt Vector Offset Registers (IVORs). The simplest way is to create different memory sections for each core and align these to 64 KB boundary. Similarly, IVOR4 handlers must also be aligned to 64 KB boundary. There is one software vector table for each core which contains indexes to handle routines for external interrupt requests. These tables must be aligned to 2 KB boundary. The ISR is selected on the basis of address which consists of two parts. The first part is given by the address of software vector table, which is saved in IVPR, and the second part is given by interrupt request which serves as offset from the start of table. For example, the start address of software vector table will be 0x1 0000 and ADC0 will generate the End of Conversation (EOC) interrupt. The number of this interrupt is 62 with offset address 0x3E0 = 62\*16, where space for each of the interrupts is 16 bytes. The IVOR4 starts to execute code from address 0x1 03E0 where the address of ISR is located. The collection of all the used files is in shown in [Table 3](#).



**Figure 2. Using files and modules in example project**

The example project includes demonstration of using ISRs in DPM. If the microcontroller is used in DPM, interrupts can be handled by two basic ways—one which uses the same interrupt for both the cores (sharing interrupt), and the other which uses different interrupts for each of the cores. The demonstration application includes both.

### 3.1 Different interrupts

The example project uses different interrupts for each core which looks like working with one core (interrupts and cores are independent). The project uses interrupt request End Of Conversation (EOC) from ADC0 for Core0 and EOC from ADC1 for Core1. Each core uses 2 different LEDs. Core0 turns on LED2 and disconnects LED1 while Core1 turns on LED4 and disconnects LED3 in the ISR. See the table below.

**Table 2. LED pins**

LEDs	Package LBGA 473	Package LBGA 257
1	P2	L16
2	P1	K17
3	P4	P15
4	P3	N15

## 3.2 Shared interrupts

In this method of interrupt handling, the same interrupt request is used by both the cores. This is possible because each core has its own INTC module. The application uses interrupt request EOC from ADC0 for both the cores. Therefore, it is very important to ensure that both the cores are running correctly at the time when the interrupt occurs because Core1 is started by Core0. If the sharing interrupt request occurs before initialization and start up Core1, this core will not know about it because Core0 clears the flag of this interrupt request. The flags of interrupts are saved in the peripherals and not in the INTC module, therefore Core1 does not know about this. The global flag ensures that Core1 is ready when flag=1. Each core uses 2 LEDs. Core0 turns on LED1 and disconnects LED2 and Core1 turns on LED3 and disconnects LED4 in the ISR.

Global variable INTC\_SEL which can contain two values, INTC\_SHARED for shared interrupts or INTC\_INDEPE for independent interrupts, provides separation of these two modes in application.

**Table 3. List of project files**

Name of file	Type of file	Mode	Breath definition	Cores
Mpc5675k-0200	header	LS/DP	Description of used device (consists of description registers)	Both
IVOR4_SW_Handler_VLE	asm	LS	Code for external interrupt IVOR4 handler (it means code for external interrupt).	Core0
IVOR4_SW_Handler_VLE_1	asm	DP	Code for external interrupt IVOR4 handler (it means code for external interrupt).	Core1
GHS_MPC56xx_55xx_FLASH	ld	LS/DP	Linker file for definition memory spaces	Both
crt0_VLE_Flash	asm	LS/DP	Startup code: Both the cores use the same MMU table besides definition of RAM memory, shadow, and test memory.	Both
Vector_SW_VLE - Z3_Z4_Z6_Z7_0	asm	LS	Setting IVPR and IVOR registers of Core0. It is for definition of interrupt subroutine of core interrupts besides IVOR4 which has own file.	Core0
Vector_SW_VLE - Z3_Z4_Z6_Z7_1	asm	DP	Setting IVPR and IVOR registers of Core1. It is for definition of interrupt subroutine of core interrupts besides IVOR4 which has its own file.	Core1
main	c	LS/DP	main code	Both

*Table continues on the next page...*

**Table 3. List of project files (continued)**

Name of file	Type of file	Mode	Breath definition	Cores
IntclsrVectors	c	LS/DP	Separate vector tables for each core	Each core has own vector table.
MPC56xx_55xx_Dual_PRC0_Interrupt_Init	c	LS/DP	The interrupt controller configuration file	Setting priority of the main function, start address vector table

## 4 References

- MPC5675KRM available at <http://www.freescale.com>
- E200Z760RM Power Architecture® Core reference manual available at <http://www.freescale.com>



## **How to Reach Us:**

### **Home Page:**

[www.freescale.com](http://www.freescale.com)

### **Web Support:**

<http://www.freescale.com/support>

### **USA/Europe or Locations Not Listed:**

Freescale Semiconductor  
Technical Information Center, EL516  
2100 East Elliot Road  
Tempe, Arizona 85284  
+1-800-521-6274 or +1-480-768-2130  
[www.freescale.com/support](http://www.freescale.com/support)

### **Europe, Middle East, and Africa:**

Freescale Halbleiter Deutschland GmbH  
Technical Information Center  
Schatzbogen 7  
81829 Muenchen, Germany  
+44 1296 380 456 (English)  
+46 8 52200080 (English)  
+49 89 92103 559 (German)  
+33 1 69 35 48 48 (French)  
[www.freescale.com/support](http://www.freescale.com/support)

### **Japan:**

Freescale Semiconductor Japan Ltd.  
Headquarters  
ARCO Tower 15F  
1-8-1, Shimo-Meguro, Meguro-ku,  
Tokyo 153-0064  
Japan  
0120 191014 or +81 3 5437 9125  
[support.japan@freescale.com](mailto:support.japan@freescale.com)

### **Asia/Pacific:**

Freescale Semiconductor China Ltd.  
Exchange Building 23F  
No. 118 Jianguo Road  
Chaoyang District  
Beijing 100022  
China  
+86 10 5879 8000  
[support.asia@freescale.com](mailto:support.asia@freescale.com)

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductors products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.

Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any liability, including without limitation consequential or incidental damages. "Typical" parameters that may be provided in Freescale Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including "Typicals", must be validated for each customer application by customer's technical experts. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claims alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.

RoHS-compliant and/or Pb-free versions of Freescale products have the functionality and electrical characteristics as their non-RoHS-complaint and/or non-Pb-free counterparts. For further information, see <http://www.freescale.com> or contact your Freescale sales representative.

For information on Freescale's Environmental Products program, go to <http://www.freescale.com/epp>.

Freescale™ and the Freescale logo are trademarks of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners.

© 2012 Freescale Semiconductor, Inc.