The following document contains information on Cypress products. The document has the series name, product name, and ordering part numbering with the prefix “MB”. However, Cypress will offer these products to new and existing customers with the series name, product name, and ordering part number with the prefix “CY”.

**How to Check the Ordering Part Number**

1. Go to www.cypress.com/pcn.
2. Enter the keyword (for example, ordering part number) in the **SEARCH PCNS** field and click **Apply**.
3. Click the corresponding title from the search results.
4. Download the Affected Parts List file, which has details of all changes

**For More Information**

Please contact your local sales office for additional information about Cypress products and solutions.

**About Cypress**

Cypress is the leader in advanced embedded system solutions for the world's most innovative automotive, industrial, smart home appliances, consumer electronics and medical products. Cypress' microcontrollers, analog ICs, wireless and USB-based connectivity solutions and reliable, high-performance memories help engineers design differentiated products and get them to market first. Cypress is committed to providing customers with the best support and development resources on the planet enabling them to disrupt markets by creating new product categories in record time. To learn more, go to www.cypress.com.

AN204446

# Considerations for Using the Interrupt Controller of Traveo™ Family

**Author: Kenichi Sunada**

**Associated Part Family: Traveo Family S6J3110/S6J3120/S6J3200/S6J3300/S6J3350/S6J3400/MB9D560 Series**

**Related Documents: For a complete list, click here.**

This application note describes the considerations for using the interrupt controller of Cypress's Traveo™ family microcontrollers.

## Contents



## 1 Introduction

This application note describes the considerations for using the interrupt controller of the Traveo family MCUs in an environment summarized in Table 1.

Table 1. Development Environment

| | |
|---|---|
| Microcomputer | Traveo family |
| Integrated development environments | MULTI v6.1.4 or later |
| Optimization | Optimize for speed |

## 2 Interrupt Controller

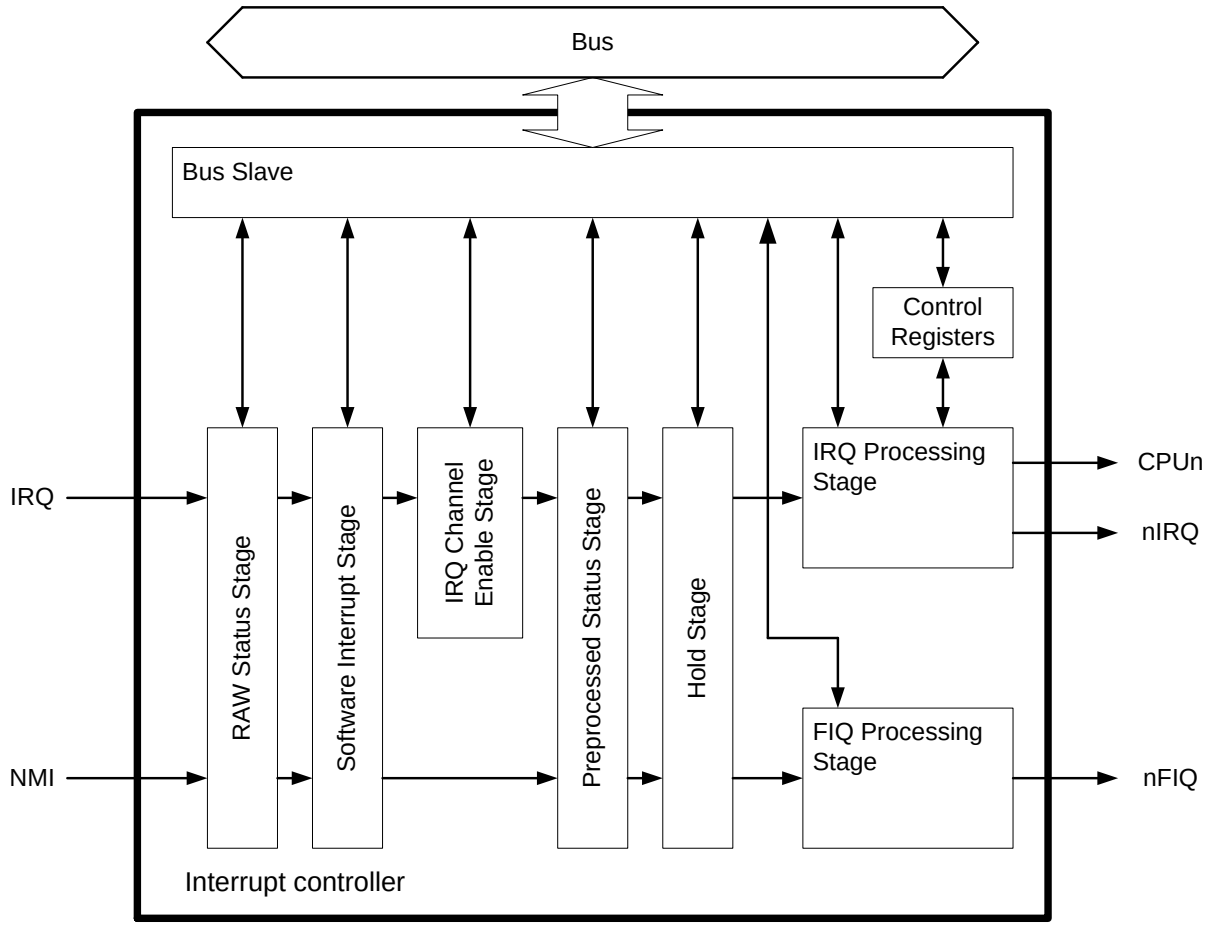
The Interrupt controller does the following when it detects an interrupt:

- Informs the issue of interrupts (nIRQ) to the CPU
- Sets the interrupt priority level/interrupt priority level mask
- Notifies the interrupt vector address

## 2.1 Interrupt Controller Configuration

The interrupt controller consists of the following blocks, as shown in Figure 1.

Figure 1. Interrupt Controller Configuration




For details on each block, refer to the Interrupt Controller chapter of the Hardware Manual of the relevant MCU. These are listed in the Related Documents section.

## 2.2 Considerations for Using Interrupt Controller

The interrupt controller controls the interrupt priority at different times during CPU processing. Therefore, it may cause an error if the register of the interrupt controller is written to while the interrupt controller is accepting interrupts. The interrupt controller remains in the interrupt pending state when the register of the interrupt controller is written. The following sections (2.2.1 and 2.2.5) describe the interrupt pending state and the writable registers during this state.

### 2.2.1 Interrupt Pending State

There are two ways to stop the interrupt controller from accepting an interrupt:

- Disable the IRQ processing block: Set the IRQ processing block enable/disable setting bit to ‘0’ (IRCn_CSR [bit0 - IRQEN]).
- Set the interrupt controller to the interrupt suppress state by reading the IRQ interrupt status bit (IRCn_IRQST [bit24 - nIRQ]).

### 2.2.2 Interrupt Suppress State

The interrupt controller does not notify the CPU of the occurrence of an interrupt when the interrupt controller status is changed to the interrupt suppress state. If the interrupt controller meets the following conditions, the status changes to the interrupt suppress state:

- Interrupt controller has notified the CPU of an interrupt.
- Interrupt controller has read the IRQ interrupt status bit of the IRC IRQ status register (IRCn_IRQST [bit24 - nIRQ]).

The IRQ interrupt status bit of the IRC IRQ status register (IRCn_IRQST [bit24 - nIRQ]) indicates the interrupt state of the interrupt controller and sets the interrupt suppression state. The interrupt controller can change to the interrupt suppress state (except when the I-FLAG of the CPU is set to ’1’ [interrupt disabled]) by reading the IRQ interrupt status bit.

The state of the interrupt controller is indicated by the value of the IRQ interrupt status bit, as described in Table 2.

Table 2. State of Interrupt Controller Indicated by Value of IRQ Interrupt Status Bit

| Value of IRQ Interrupt Status Bit | State of Interrupt Controller |
|---|---|
| 0 | One of the following:<br>▪ Interrupt controller is generating the interrupt for the CPU.<br>▪ Interrupt controller is in the interrupt suppress state. |
| 1 | Interrupt controller does not accept interrupts. It has transitioned to the interrupt suppress state by reading the IRQ interrupt status bit (IRCn_IRQST [bit24 - nIRQ]). |

### 2.2.3 Interrupt Status Bit

Consider the following on the IRQ interrupt status bit (IRCn_IRQST[bit24 - nIRQ]) read.

- Read values

  The interrupt controller enters the interrupt suppress state by reading the IRQ interrupt status bit, which sets it to ’1’. Therefore, the read value of the IRQ interrupt status bit is always ‘0’ for all subsequent reads. However, after the interrupt controller exits the interrupt suppress state, the value may change.

- Read disable timing

  Reading the IRQ interrupt status bit is not allowed when the I-FLAG of the CPU is set to ’1’ (interrupt disabled). The interrupt controller may not change the interrupt suppress state, even by reading the IRQ interrupt status bit, when the CPU (I-FLAG =1) has received the next interrupt.

### 2.2.4 Release of the Interrupt Suppress State

The interrupt suppress state is released by writing to the interrupt suppression release registers described in Table 3.

Table 3. Interrupt Suppression Release Registers

| Register Abbreviation | Register Name |
|---|---|
| IRCn_IRQVAr | IRC IRQ Vector Address Register |
| IRCn_IRQPL0-127 | IRC IRQ Priority Level Register |
| IRCn_IRQS0-15 | IRC IRQ Software Interrupt Set Register |
| IRCn_IRQR0-15 | IRC IRQ Software Interrupt Reset Register |
| IRCn_IRQCES0-15 | IRC IRQ Channel Enable Set Register |
| IRCn_IRQCEC0-15 | IRC IRQ Channel Enable Clear Register |
| IRCn_IRQCE0-15 | IRC IRQ Channel Enable Setting Register |
| IRCn_IRQHC | IRC IRQ Hold Clear Register |
| IRCn_IRQPLM | IRC IRQ Priority Level Mask Register |
| IRCn_CSR | IRC Control/Status Register |

Once the interrupt controller status is changed to the interrupt suppress state, this state persists until the interrupt suppression release register is written to. ***If the interrupt controller needs to accept interrupts again, it is necessary to write the interrupt suppression release register as a dummy write even if its value does not need to be changed.***

### 2.2.5 Writable Registers During Interrupt Pending State

Writing to the registers listed in Table 4 is allowed only while interrupts are disabled. An error may occur if these registers are written to when the interrupt controller is not in the interrupt pending state.

Table 4. Writable Registers During Interrupt Pending State

| Register Abbreviation | Register Name |
|---|---|
| IRCn_IRQPL0-127 | IRC IRQ Priority Level Register |
| IRCn_IRQHC | IRC IRQ Hold Clear Register |
| IRCn_IRQPLM | IRC IRQ Priority Level Mask Register |
| IRCn_CSR | IRC Control/Status Register |

The registers in Table 4 are included in Table 3. Therefore, the interrupt suppress state is released by writing to the registers in Table 4. Access them according to the following methods if you need to write to the Table 4 registers several times:

- Read from the IRQ interrupt status bit (IRCn_IRQST [bit24 - nIRQ]) whenever writing to the Table 4 registers.
- Disable the IRQ processing block before writing to the Table 4 registers. (Set the IRQ processing block enable/disable setting bit to ‘0’ (IRCn_CSR [bit0 - IRQEN]).)

# 3 Considerations for Interrupt Handler Corresponding to Multiple Interrupts

Because the ARM® architecture does not support multiple interrupts, it is necessary to use a software-implemented multiple interrupt handler. Sections 3.1, 3.2, and 3.3 describe the considerations for implementing the interrupt handler corresponding to multiple interrupts listed in this chapter.

## 3.1 Releasing the Interrupt Suppress State

The interrupt controller changes to the interrupt suppress state after notifying an interrupt to the CPU. Therefore, the interrupt controller is in an interrupt suppress state at the start of interrupt handler processing. If a new interrupt needs to be accepted, it is necessary to write to a register listed in Table 3 at the beginning of the processing.

In addition, at the start of the interrupt handler processing, the I-FLAG of the CPU is set to '1' by the hardware. Reset the I-FLAG of the CPU to '0' (interrupt enabled) at the same time as the release of the interrupt suppress state.

## 3.2 Clearing the Hold Bit

It is necessary to clear the hold bit when interrupt processing ends and the state of the CPU I-FLAG is set to '1' (interrupt disabled).

If the hold bit is cleared when the I-FLAG of the CPU is '0', the interrupt suppress state is released at the same time. Therefore, it may cause a stack overflow if the succeeding interrupt processing is executed before the completion of interrupt handler processing.

IRCn_IRQHC is a writable register during the interrupt pending state as listed in Table 3. It is necessary to clear the hold bit using the following procedure when the state of the I-FLAG of the CPU is set to '1' (interrupt disabled).

1. Read the IRQ interrupt status bit (IRCn_IRQST [bit24 - nIRQ]).
2. Set the I-FLAG of the CPU to '1'.
3. Write to IRCn_IRQHC the IRQ channel number that is to be cleared.

## 3.3 Disabling an Interrupt

It is possible to disable an interrupt in the interrupt handler corresponding to multiple interrupts by using the methods listed in Table 5. However, the processing described is prohibited while an interrupt is disabled. The methods are illustrated in Figure 2 to Figure 5.

Table 5. How to Disable Interrupt

| Method | How to Disable Interrupt | What is Not Allowed |
|---|---|---|
| 1 | Set I-FLAG of CPU to '1' (interrupt disabled). | ▪ Reading from the IRQ interrupt status bit<br>▪ Writing to a register listed in Table 4 |
| 2 | Read from IRQ interrupt status bit.[1] | Writing more than once to a register listed in Table 4 |
| 3 | Immediately after IRQ interrupt status bit read, set I-FLAG of CPU to '1' (interrupt disabled).[1] | ▪ After setting I-FLAG of CPU to '1' (interrupt disabled), reading from the IRQ interrupt status bit<br>▪ Writing more than once to a register listed in Table 4 |
| 4 | Set IRQ processing block enable/disable setting bit to '0' (IRQ processing block is disabled).[2] | Reading from the IRQ interrupt status bit |

1: IRQ interrupt status bit: IRCn_IRQST[bit24 - nIRQ]

2: IRQ processing block enable/disable setting bit: IRCn_CSR[bit0 - IRQEN]

Figure 2. Example of Method 1

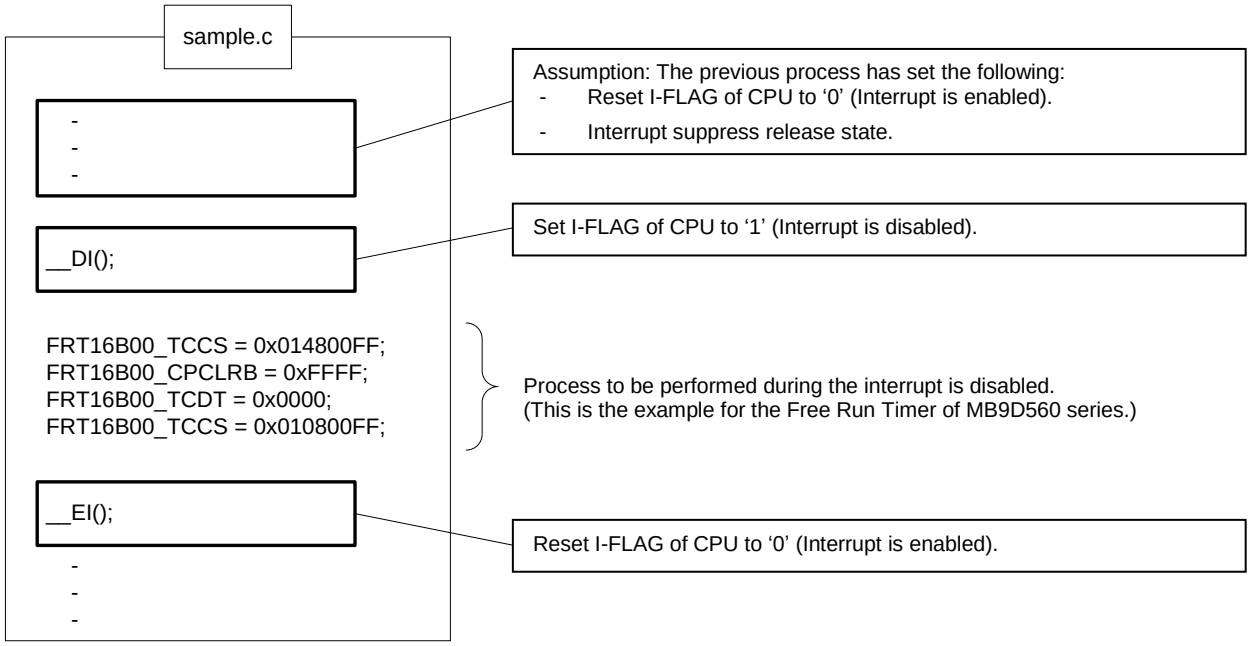



Figure 3. Example of Method 2

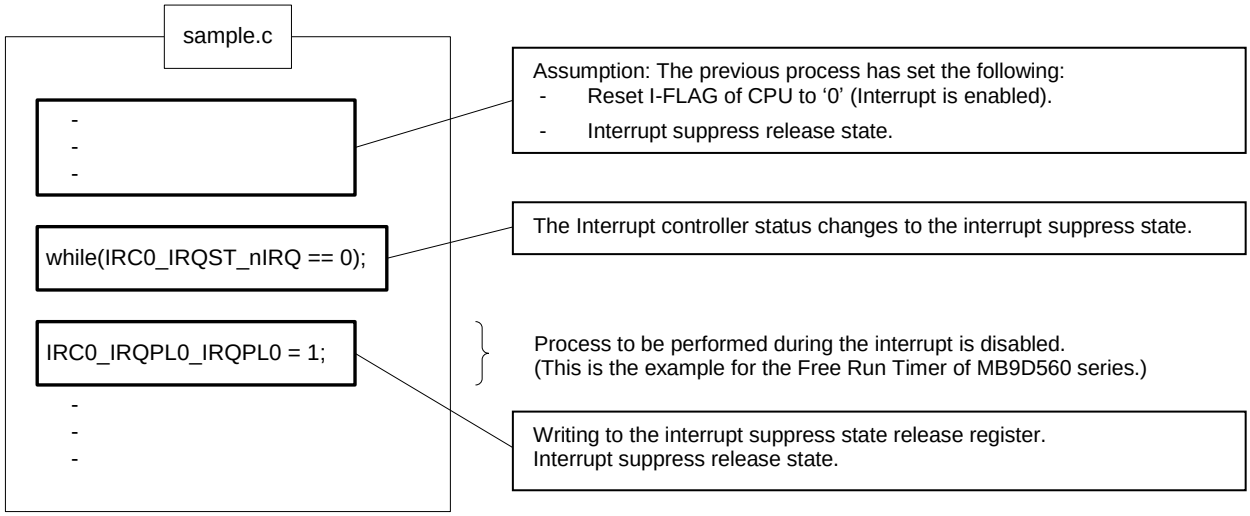


Figure 4. Example of Method 3

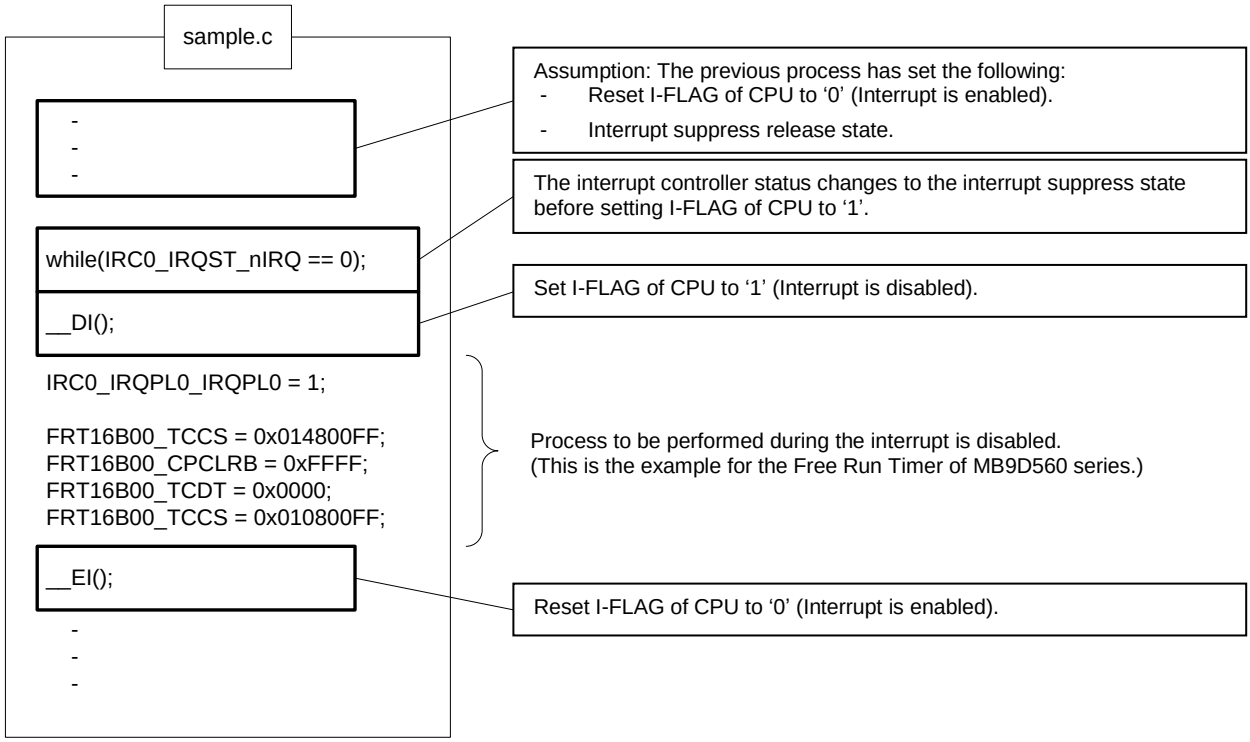
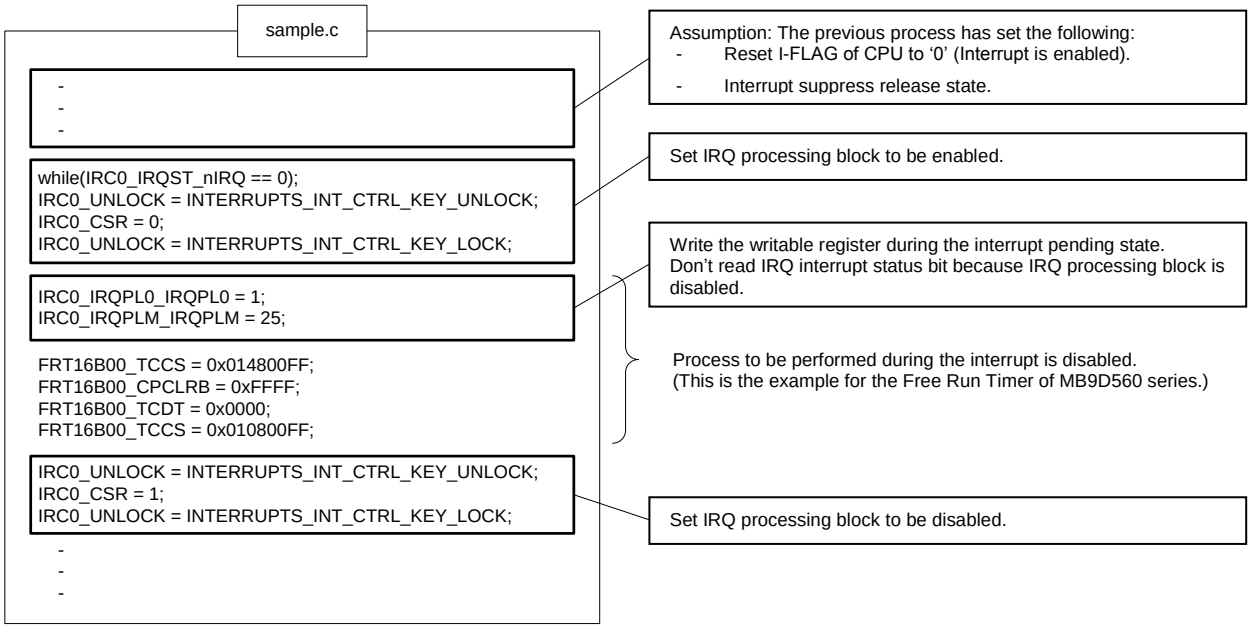



Figure 5. Example of Method 4



## 3.4 Implementation Example

The implementation example is depicted in Figure 6. The MULTI compiler does not have a pseudo-instruction corresponding to the multiple interrupts of ARM Cortex®-R. Therefore, it is necessary to implement the processing at the assembler level.

Figure 6. Implementation of Interrupt Handler Corresponding to Multiple Interrupts

sample.asm

```
ADDRESS_OF_IRC0_CSR         .equ   0xb0400c98
ADDRESS_OF_IRQST_nIRQ       .equ   0xb040000f
ADDRESS_OF_IRC0_IRQHC       .equ   0xb0400c48
ADDRESS_OF_IRC0_UNLOCK      .equ   0xb0400d30
VALUE_OF_IRC0_UNLOCK        .equ   0x17ACC911
VALUE_OF_IRC0_LOCK          .equ   0x17B10C11

nested_handler:
   SUB          lr, lr, #4
   SRSFD        sp!, #0x1f
   CPS          #0x1f
   PUSH         {r0-r3, r12}
   FSTMDBD      sp!, {d0-d7}
   FMRX         r1, fpscr
   PUSH         r1
   AND          r1, sp, #4
   SUB          sp, sp, r1
   PUSH         {r1, lr}

   CPSIE        i

   MOVW         r1, #(ADDRESS_OF_IRC0_UNLOCK & 0xFFFF)
   MOVT         r1, #(ADDRESS_OF_IRC0_UNLOCK >> 16)
   MOVW         r0, #(VALUE_OF_IRC0_UNLOCK & 0xFFFF)
   MOVT         r0, #(VALUE_OF_IRC0_UNLOCK >> 16)
   STR          r0, [r1]

   MOVW         r1, #(ADDRESS_OF_IRC0_CSR & 0xFFFF)
   MOVT         r1, #(ADDRESS_OF_IRC0_CSR >> 16)
   MOV          r0, #1
   STR          r0, [r1]

   MOVW         r1, #(ADDRESS_OF_IRC0_UNLOCK & 0xFFFF)
   MOVT         r1, #(ADDRESS_OF_IRC0_UNLOCK >> 16)
   MOVW         r0, #(VALUE_OF_IRC0_LOCK & 0xFFFF)
   MOVT         r0, #(VALUE_OF_IRC0_LOCK >> 16)
   STR          r0, [r1]

   LDR          r1, =isr_name
   BLX          r1

   MOVW         r1, #(ADDRESS_OF_IRQST_nIRQ & 0xFFFF)
   MOVT         r1, #(ADDRESS_OF_IRQST_nIRQ >> 16)
IRQST_check_loop:
   LDRB         r0, [r1]
   CMP          r0, #0
   BEQ          IRQST_check_loop

   CPSID        i

   MOVW         r0, #(irq_num & 0x1FF)
   MOVW         r1, #(ADDRESS_OF_IRC0_IRQHC & 0xFFFF)
   MOVT         r1, #(ADDRESS_OF_IRC0_IRQHC >> 16)
   STR          r0, [r1]

   POP          {r1, lr}
   ADD          sp, sp, r1
   POP          r1
   FMXR         fpscr, r1
   FLDMIAD      sp!, {d0-d7}
   POP          {r0-r3, r12}
   RFEFD        sp!
```

- Interrupt is enabled.
- Unlock setting for writing the dummy data.
- Write the dummy data for the release of the interrupt suppress state.
- Set for the writing lock state.
- The processing moves to the interrupt process implementation function.
  *isr_name = The start address of the interrupt process implementation function.
- The Interrupt controller status changes to the interrupt suppress state.
- Interrupt is disabled.
- Reset the Hold bit.
  *irq_num = Channel number that corresponds to the ISR of IRQ.

# 4 Considerations for Interrupt Handler Not Corresponding to Multiple Interrupts

To avoid accepting the next interrupt until the processing of the interrupt handler is complete, it is necessary to meet all of the following conditions until the interrupt handler processing is complete.

- Interrupt controller is in the interrupt pending state.
- I-FLAG of the CPU is set to '1' (interrupt disabled).

Sections 4.1 and 4.2 describe the considerations for implementing the interrupt handler that does not correspond to multiple interrupts discussed in this section.

## 4.1 Setting the Interrupt Pending State

It is necessary to set the interrupt pending state by writing a '0' to the IRQ processing block enable/disable setting bit (IRCn_CSR [bit0 - IRQEN]) at the beginning of the interrupt handler.

The interrupt controller enters an interrupt suppress state when the processing transitions to the interrupt handler because the interrupt controller enters the interrupt pending state at that time. However, in the situation where the interrupt pending state results from the interrupt suppress state, the interrupt pending state is released at the time of writing to the interrupt suppress release register.

To accept the next interrupt, it is necessary to enable the IRQ processing block after the interrupt handler processing is complete.

IRCn_IRQHC is a register in listed Table 4. Therefore, it is necessary to enable the IRQ processing block after the hold bit has cleared.

## 4.2 Reading from the IRQ Interrupt Status Bit

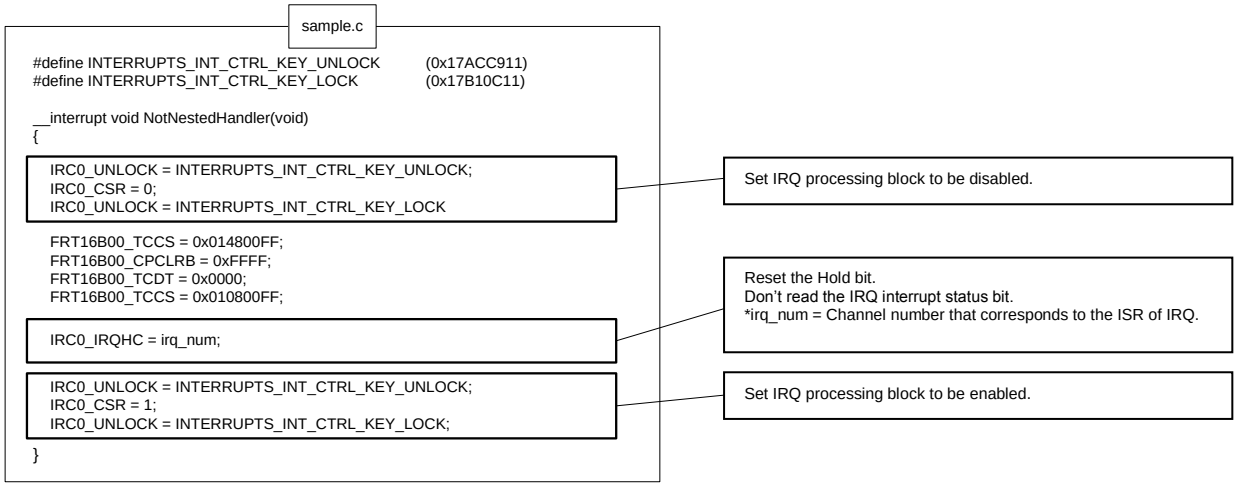
The I-FLAG of the CPU must be set to '1' during the processing of the interrupt handler. Therefore, reading the IRQ interrupt status bit (IRCn_IRQST [bit24 - nIRQ]) in the interrupt handler processing is forbidden.

The interrupt controller has entered the interrupt pending state because the IRQ processing block has been disabled during the processing by the interrupt handler. It is not necessary to stop processing the interrupt acceptance when writing to a register listed in Table 4.

## 4.3 Implementation Example

The implementation example is depicted in Figure 7. If it is not necessary to use multiple interrupts, it is possible to implement the pseudo-instruction of the MULTI compiler (__interrupt).

Figure 7. Example of Interrupt Handler Implementation That Does Not Correspond to Multiple Interrupts



# 5 Related Documents

Traveo family series datasheets and hardware manuals:

- S6J3110
- S6J3120
- S6J3200
- S6J3300
- S6J3350
- MB9D560

## Document History

Document Title: AN204446 - Considerations for Using the Interrupt Controller of Traveo™ Family

Document Number: 002-04446

| Revision | ECN | Orig. of Change | Submission Date | Description of Change |
|---|---|---|---|---|
| ** | - | KHAS | 07/16/2015 | Initial Release |
| *A | 5040207 | KHAS | 12/07/2015 | Migrated Spansion Application Note from MB9D560_AN708-00006-1v0-E to Cypress format |
| *B | 5309324 | KSUN | 06/22/2016 | Added target products S6J3110/S6J3120/S6J3200/S6J3300/S6J3350/S6J3400 series<br>Updated template |
| *C | 5874817 | AESATMP9 | 09/06/2017 | Updated logo and copyright. |

## Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

## Products

| | |
|---|---|
| ARM® Cortex® Microcontrollers | cypress.com/arm |
| Automotive | cypress.com/automotive |
| Clocks & Buffers | cypress.com/clocks |
| Interface | cypress.com/interface |
| Internet of Things | cypress.com/iot |
| Memory | cypress.com/memory |
| Microcontrollers | cypress.com/mcu |
| PSoC | cypress.com/psoc |
| Power Management ICs | cypress.com/pmic |
| Touch Sensing | cypress.com/touch |
| USB Controllers | cypress.com/usb |
| Wireless Connectivity | cypress.com/wireless |

## PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6

## Cypress Developer Community

Forums | WICED IOT Forums | Projects | Videos | Blogs | Training | Components

## Technical Support

cypress.com/support



Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709