AN99231

Using Interrupts in FM0+ Family S6E1C3 Series

Author: Mily Wang

Associated Part Family: FM0+

Related Application Notes: AN54460, AN90799

To get the latest version of this application note, or the associated project file, please visit http://www.cypress.com/go/AN99231.

This application note introduces the basic concepts of the nested vectored interrupt controller (NVIC) function of the Arm Cortex-M0+ MCU in the FM0+ family of MCUs. The operation and advantages of the newly designed S6E1C3 Series NVIC are also described.

Contents

1 Introduction....1
2 Overview....1
2.1 NVIC Architecture....1
2.2 NVIC Features....2
3 NVIC Design of S6E1C3 Series....2
3.1 General Design....4
3.2 New Design....4
3.3 Interrupt Vector Table and Interrupt List....4
4 Example Code....6
4.1 Interrupt Vector Table Example Code....7
4.2 Main Routine Example Code....9
4.3 ISR Example Code....10
5 Performance Comparison....12
5.1 Code Space....12
5.2 Execution Speed....12
5.3 Coding Complexity....13
6 Summary....13
7 Precaution for Use....13
Appendix A. Exceptions and Interrupts....14
Document History....18
Worldwide Sales and Design Support....19

1 Introduction

The S6E1C3 Series devices are highly integrated 32-bit microcontrollers with an Arm Cortex-M0+ processor. They include a new NVIC design that enables greater system efficiency. The Cortex-M0+ processor supports several interrupts and system exceptions, which are aggregated and processed by the NVIC.

This application note introduces the basic concepts of the NVIC function of the FM0+ family, the new design of the NVIC in the S6E1C3 Series, and the operation with the newly designed NVIC. An application note project is provided with the document.

2 Overview

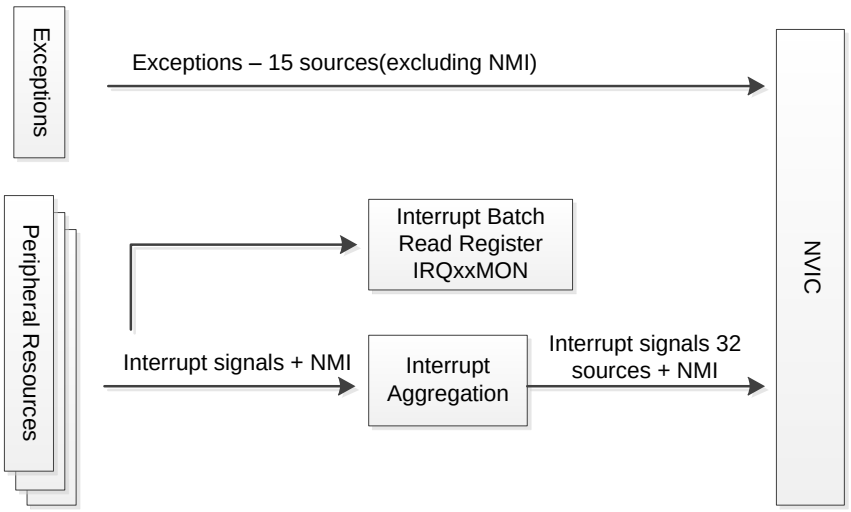
The Cortex-M0+ CPU core is equipped with the NVIC internally, within the core. The NVIC determines the priority of interrupt requests and sends the requests to the CPU.

2.1 NVIC Architecture

Figure 2-1 shows the architecture of the FM0+ interrupt function. Exceptions are system requests such as Reset, SysTick, and HardFault. These requests are input to the NVIC module and correspond to the exception numbers. For more information, refer to the Cortex-M0+ Technical Reference Manual.

The interrupt signals from several peripherals and exceptions are aggregated and input to the corresponding interrupt vector. The total number of the interrupt vector is 32, except the nonmaskable interrupt (NMI) source. A series of Interrupt Batch Registers named “IRQxxMON” indicate which interrupt request occurs in the system.

Figure 2-1. FM0+ Interrupt Architecture



2.2 NVIC Features

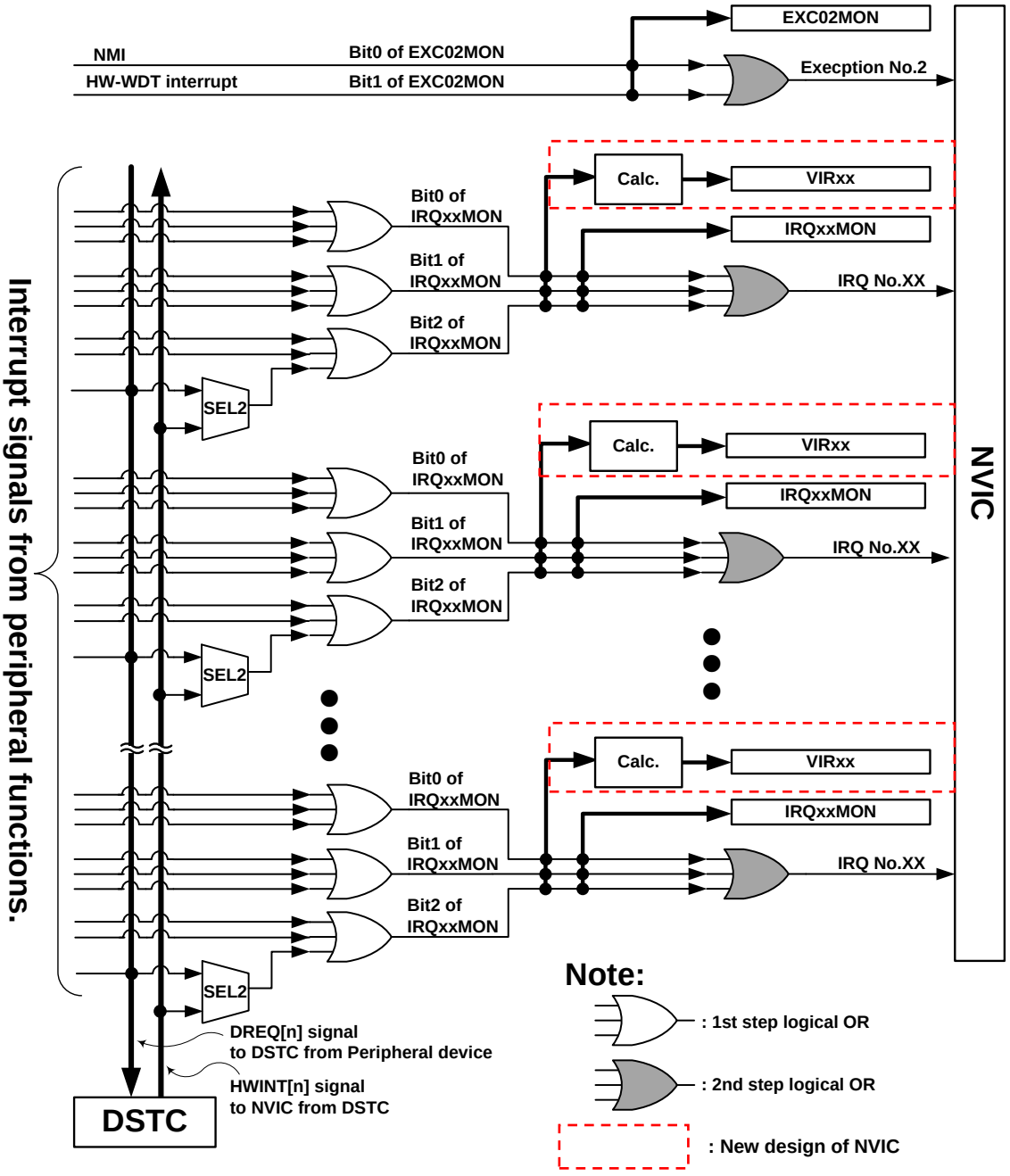
The NVIC of the FM0+ family offers the following features:

- 32 maskable peripheral interrupt channels (not including the 16 exception interrupts of Cortex-M0+)
- 4 programmable interrupt priority levels (using 2-bit prioritized interrupts)
- Support for NMI input
- Support for low-latency exception and interrupt handling
- Implementation of System Control Registers

3 NVIC Design of S6E1C3 Series

Figure 3-1 illustrates the connection between the peripheral interrupt signals and the NVIC module. This section introduces the general design and new design of the NVIC in the S6E1C3 Series.

Figure 3-1. Connection Between Interrupt Signals and NVIC



When the DMA transfer with DSTC is selected, transfer completion interrupts from the DSTC are generated instead of interrupts from the peripheral. For more information about DSTC transfer requests from each peripheral function, refer to the appropriate chapter for the peripheral function in the peripheral manual.

The new design of the NVIC is illustrated in the red dashed box. The rest of Figure 3-1 shows the general design of the NVIC.

3.1 General Design

3.1.1 Interrupt Vector Table

The interrupt vector table is shown in Table 3-1. The address from 0x00000000 to 0x000000BC is for the general design.

3.1.2 NMI and Hardware Watchdog Interrupt

The NMI signals from the external interrupt and NMI controllers, and the hardware (HW) watchdog interrupt from the HW watchdog timer, are aggregated by a second-step logical OR and then connected to the input of NVIC exception no. 2. When an interrupt of exception no. 2 is generated, the source of the interrupt, which is either an NMI or HW watchdog interrupt, can be identified by reading the EXC02 Batch Read Register (EXC02MON in Figure 3-1). For a description of EXC02MON, refer to the peripheral manual.

3.1.3 Other Peripheral Interrupts

Interrupt signals from peripheral functions are aggregated by the two steps of the OR circuit (Figure 3-1). The aggregated interrupt signals are then connected to one of the 32 peripheral interrupts of the NVIC. See Table A-1 to see which peripheral function interrupt signal is assigned to which IRQ input of the NVIC.

When an interrupt is generated, reading the Interrupt Batch Read Registers (IRQxxMON in Figure 3-1) enables the source to be identified. The IRQxxMON registers cover all interrupt inputs of the NVIC. Thirty-two registers (IRQ00MON to IRQ31MON) are supported. For a description of IRQxxMON, refer to the peripheral manual.

3.1.4 Interrupt Batch Read Registers

The IRQxxMON registers indicate which interrupt occurs. To see the relationship between the IRQxxMON registers and the peripheral interrupt, refer to Table A-1.

3.2 New Design

The new design is based on the general design. It adds some special designs for convenience and efficiency.

3.2.1 Interrupt Vector Table

The interrupt vector table is shown in Table 3-1. The address from 0x000000C0 to 0x000001FC is added for the new design, as indicated by the red box.

3.2.2 Vector Indicate Register

When an interrupt occurs, the CPU can use the Vector Indicate Register (VIRxx) to quickly start an interrupt branch operation. The VIRxx registers are 32 registers (VIR00–VIR31) corresponding to the IRQ00–IRQ31 inputs of the NVIC. When an interrupt occurs, the CPU can read out a routine address value from VIRxx. It can branch the interrupt operation quickly by locating the top address of the interrupt handler in this address area. Following is an explanation of the behavior and use of VIR00 when IRQ00 occurs.

As shown in Table 3-1 the program prepares the top address value of the IRQ interrupt handler in address area 0x0000 0040 to 0x0000 00BC. It also prepares the top address value of the IRQ interrupt handler corresponding to each bit0,1,2 in address area 0x0000 00C0 to 0x0000 01FC.

When IRQ00 occurs, the IRQ00 interrupt handler starts, and the CPU reads VIR00. The value read from VIR00 is shown in Table 3-2. If a Bit0 interrupt occurs, the read value of VIR00 is 0x0000 00C0. If a Bit1 interrupt occurs, the read value of VIR00 is 0x0000 0140. And if a Bit2 interrupt occurs, the read value of VIR00 is 0x0000 01C0. The IRQ00 interrupt handler branch operation to address the interrupt is located in the address read from VIR00. In this way, the interrupt handler branch operation can be performed quickly.

3.2.3 VIR Offset Register

The VIR Offset Register (VIR_OFFSET) is used to define a common offset value for VIRxx.

3.3 Interrupt Vector Table and Interrupt List

The S6E1C3 interrupt vector table is shown in Table 3-1. If the general design is adopted, the interrupt vector table address is in the range 0x00000000 to 0x000000BC. If the new design is adopted, the interrupt vector table address is in the range 0x00000000 to 0x000001FC. The vector address from 0x000000C0 to 0x000001FC is added for the new design, as illustrated by the red box in Table 3-1.

Table 3-1. Interrupt Vector Table

Address	Data
0x0000 0000	Stack pointer initial value
0x0000 0004	Exception 1: Reset vector
0x0000 0008	Exception 2: NMI/HW-WDT Handler top address
0x0000 000C	Exception 3: Hard fault Handler top address
0x0000 0010 - 0x0000 0028	Reserved
0x0000 002C	Exception 12: SoCal Handler top address
0x0000 0030 - 0x0000 0034	Reserved
0x0000 0038	Exception 14: PendSV Handler top address
0x0000 003C	Exception 15: SysTick Handler top address
0x0000 0040	IRQ00 Handler top address
0x0000 0044	IRQ01 Handler top address
....	....
0x0000 00B8	IRQ30 Handler top address
0x0000 00BC	IRQ31 Handler top address
0x0000 00C0	IRQ00 – bit0 Handler top address
0x0000 00C4	IRQ01 – bit0 Handler top address
....	....
0x0000 0138	IRQ30 – bit0 Handler top address
0x0000 013C	IRQ31 – bit0 Handler top address
0x0000 0140	IRQ00 – bit1 Handler top address
0x0000 0144	IRQ01 – bit1 Handler top address
....	....
0x0000 01B8	IRQ30 – bit1 Handler top address
0x0000 01BC	IRQ31 – bit1 Handler top address
0x0000 01C0	IRQ00 – bit2 Handler top address
0x0000 01C4	IRQ01 – bit2 Handler top address
....	....
0x0000 01F8	IRQ14 – bit2 Handler top address
0x0000 01FC	IRQ15 – bit2 Handler top address

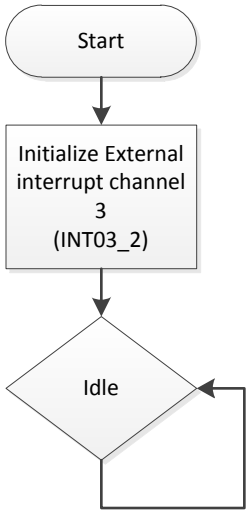
Table 3-2. Status of IRQ00 and Read Value of VIR00

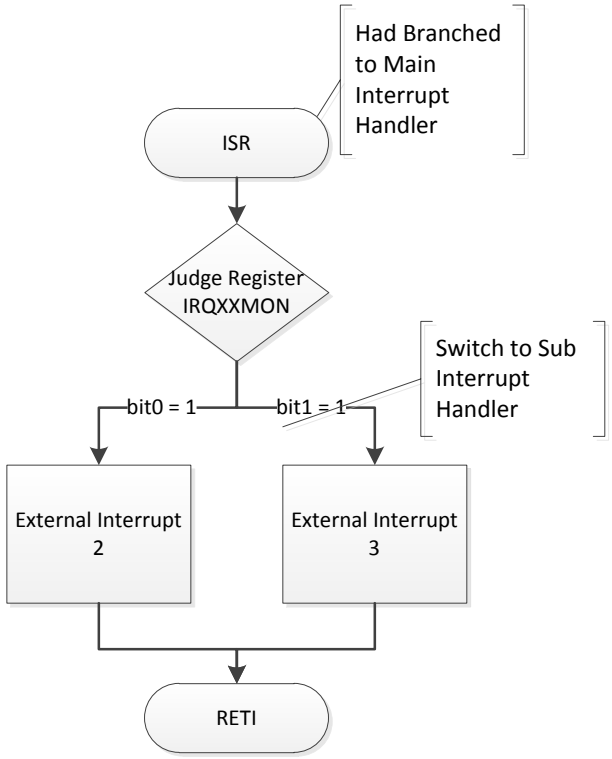
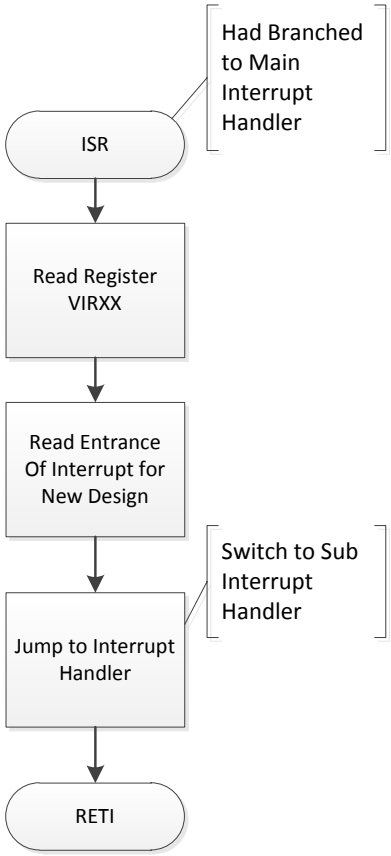
IRQ00 Input Signal Status			VIR00 Read Value
Bit0 Interrupt Status	Bit1 Interrupt Status	Bit2 Interrupt Status	
1	Ignored	Ignored	0x0000 00C0 + VIR_OFFSET
0	1	Ignored	0x0000 0140 + VIR_OFFSET
0	0	1	0x0000 01C0 + VIR_OFFSET
0	0	0	Undefined value

4 Example Code

The application code includes the interrupt vector table code, main routine code, and interrupt service routine (ISR) code. This section describes the application method for external interrupt channel 3. Pin INT03_2 was chosen for the external interrupt function. The source code for the new design and the general design includes the same main routine code. Both have approximately the same interrupt vector table, but the flow for the ISR code is different. See Table 4-1 for details.

Table 4-1. Application Methods in General Design and New Design

Code	General Design	New Design
Interrupt Vector Table	The source code is shown in Code 1 and Code 2.	The interrupt entrance follows the interrupt vector table. The source code is shown in Code 1 and Code 2.
Main Routine	 The source code is shown in Code 4.	

Code	General Design	New Design
ISR	 Source code is shown in Code 5.	 The source code is shown in Code 6.

The flow for the ISR code needs to branch to the interrupt handler twice. The first interrupt handler, called “main interrupt handler” (only in this document), is entered by the interrupt controller of the core. The second interrupt handler, called “sub interrupt handler” (only in this document), is entered by software. The difference in the ISR flow is the method of the branch process for the sub interrupt handler. Refer to “ISR” in Table 4-1.

4.1 Interrupt Vector Table Example Code

The source code for the interrupt vector table is different for the Keil compiler and the IAR compiler. Part of the source code of the interrupt vector table for the Keil compiler is shown in Code 1. It includes 16 vectors for exceptions and 32 vectors for peripheral interrupt entrance. The interrupt handler entrance for the new design follows the interrupt vector table.

Code 1. Interrupt Vector Table Source Code for Keil Compiler (applicable to applications for general design and new design)

```
                ;System Exceptions
__Vectors       DCD     __initial_sp              ; Top of Stack
                DCD     Reset_Handler             ; Reset Handler
                DCD     NMI_Handler               ; NMI Handler
                DCD     HardFault_Handler         ; Hard Fault Handler
                DCD     0                         ; Reserved
                DCD     0                         ; Reserved
                DCD     0                         ; Reserved
```

```
                DCD     0                           ; Reserved
                DCD     0                           ; Reserved
                DCD     0                           ; Reserved
                DCD     0                           ; Reserved
                DCD     SVC_Handler                 ; SVCall Handler
                DCD     0                           ; Reserved
                DCD     0                           ; Reserved
                DCD     PendSV_Handler              ; PendSV Handler
                DCD     SysTick_Handler             ; SysTick Handler
                ;Interrupt Handler
                DCD     CSV_SWDT_LVD_IRQHandler     ; 0:
                DCD     MFS0_IRQHandler             ; 1:
                DCD     MFS1_IRQHandler             ; 2:
                DCD     0                           ; 3:
                … …
                DCD     DT_RTC_WC_IRQHandler        ; 15:
                DCD     INT0_1_IRQHandler           ; 16:
                DCD     INT2_3_IRQHandler           ; 17:
                DCD     INT4_5_IRQHandler           ; 18:
                DCD     INT6_7_IRQHandler           ; 19:
                ……
                DCD     ICC1_FLASH_IRQHandler       ; 29:
                DCD     DSTC_IRQHandler             ; 30:
                DCD     0                           ; 31:
                ;Followed the General Design Vector Table
                ;Interrupt Handler of New Design
                SPACE   68
                DCD     EXINT2_IRQHandler           ; At 0x104
                SPACE   124
                DCD     EXINT3_IRQHandler           ; At 0x184
```

Part of the source code of the interrupt vector table for the IAR compiler is shown in Code 2. Two new segments are created for positioning the external interrupt handler. At the same time, the positions of the additional segments need to be defined in the ICF file that is used in the IAR project. Refer to Code 2 and Code 3.

Code 2. Interrupt Vector Table Source Code for IAR Compiler (applicable to applications for general design and new design)

```
                SECTION .exint2vec:CODE:ROOT(2)
                PUBLIC  __myvector_exitint2_table
                DATA
__myvector_exitint2_table DCD     EXINT2_IRQHandler;

                SECTION .exint3vec:CODE:ROOT(2)
                PUBLIC  __myvector_exitint3_table
                DATA
__myvector_exitint3_table DCD     EXINT3_IRQHandler;

            THUMB

                PUBWEAK EXINT2_IRQHandler
                SECTION .text:CODE:NOROOT:REORDER(1)
EXINT2_IRQHandler
                B       EXINT2_IRQHandler

                PUBWEAK EXINT3_IRQHandler
                SECTION .text:CODE:NOROOT:REORDER(1)
EXINT3_IRQHandler
                B       EXINT3_IRQHandler
```

Code 3. Position Definition in ICF File

```
define symbol __NewDesign_extint2_start__ = 0x104;
define symbol __NewDesign_extint3_start__ = 0x184;

place at address mem:__NewDesign_extint2_start__ { readonly section .exint2vec };
place at address mem:__NewDesign_extint3_start__ { readonly section .exint3vec };
```

4.2 Main Routine Example Code

Code 4 shows the source code of the main routine.

Code 4. Source Code of Main Routine (applicable to applications for general design and new design)

```
int32_t main(void)
{
    /* Initialize INT03_2 interrupt*/
    FM0P_GPIO->PFR3_f.P0 = 1;          // Use a pin as peripheral function
    FM0P_GPIO->EPFR06_f.EINT03S = 3u; // Chose peripheral pin INT03_2
    FM0P_EXTI->ELVR_f.LA3 = 1u;
```

```
    FM0P_EXTI->ELVR_f.LB3 = 1u;        // Falling edge
    FM0P_EXTI->EICL_f.ECL3 = 0;        // clear interrupt factor
    FM0P_EXTI->ENIR_f.EN3 = 1u;        // Enable INT03
    /* Enable NVIC */
    NVIC_ClearPendingIRQ(EXINT2_3_IRQn);
    NVIC_SetPriority(EXINT2_3_IRQn, 0);
    NVIC_EnableIRQ(EXINT2_3_IRQn);
    /* Initialize GPIO P0C for test*/
    FM0P_GPIO->PFR0_f.PC = 0;           // Enable P30 for GPIO function
    FM0P_GPIO->PDOR0_f.PC = 1;          // Set output high level
    FM0P_GPIO->DDR0_f.PC = 1;           // Set GPIO output

    while(1)
    {
        // write code here
    }
}
```

4.3 ISR Example Code

4.3.1 General Design

The ISR source code for the general design is shown in Code 5. In the source code, function INT2_3_IRQHandler is the main interrupt handler. Functions EXINT2_IRQHandler and EXINT3_IRQHandler are sub interrupt handlers. The system branches to the main interrupt handler via the MCU core and then branches to the sub interrupt handler by reading the value from the IRQ17MON register.

Code 5. ISR Source Code for General Design

```
void EXINT3_IRQHandler(void)
{
    FM0P_GPIO->PDOR0_f.PC = 0;    //Test IO output low
    FM0P_EXTI->EICL_f.ECL3 = 0;   //Clear interrupt factor
}

void EXINT2_IRQHandler(void)
{
    FM0P_EXTI->EICL_f.ECL2 = 0;    //Clear interrupt factor
}

void INT2_3_IRQHandler(void)
{
    uint32_t u32IrqMon = FM0P_INTREQ->PDL_IRQMON_INT2_3; //Get register IRQ17MON
```

```
    /*External interrupt 2*/
    if (0x00000001u == (u32IrqMon & 0x00000001u))
    {
        EXINT2_IRQHandler();
    }
    /*External interrupt 3*/
    if (0x00000002u == (u32IrqMon & 0x00000002u))
    {
        EXINT3_IRQHandler();
    }
}
```

4.3.2 New Design

The ISR source code for the new design is shown in Code 6. As in the general design, function INT2_3_IRQHandler is the main interrupt handler, and functions EXINT2_IRQHandler and EXINT3_IRQHandler are sub interrupt handlers. The system branches to the main interrupt handler via the MCU core and then branches to the sub interrupt handler by reading the value from the VIR17 register.

Code 6. ISR Source Code for New Design

```
void EXINT3_IRQHandler(void)
{
    FM0P_GPIO->PDOR0_f.PC = 0;    //Test IO output low
    FM0P_EXTI->EICL_f.ECL3 = 0;   //Clear interrupt factor
}

void EXINT2_IRQHandler(void)
{
    FM0P_EXTI->EICL_f.ECL2 = 0;   //Clear interrupt factor
}

void INT2_3_IRQHandler(void)
{
    typedef void (*func_ptr_parg32_t)(void);
    func_ptr_parg32_t Irq_Entry;
    uint32_t u32IrqEntrance;
    u32IrqEntrance = *((uint32_t *)FM0P_IRQ_VIR->VIR17); //Get IRQ entrance in New
Design
    Irq_Entry = (func_ptr_parg32_t)u32IrqEntrance;
    Irq_Entry();              //Run to IRQ handler
}
```

5 Performance Comparison

This section compares the performance of an application for the general design and the new design. The Keil compiler is used for this comparison.

5.1 Code Space

As described in Table 4-1, the difference between the source code for the general design and new design is in the ISR code. Code 7 and Code 8 are selected from the map file that is generated by the Keil compiler. In the map file, function INT2_3_IRQHandle takes 42 bytes for the general design application and 8 bytes for the new design application. This difference appears to indicate that the ISR code of the new design occupies a smaller code space in the current example.

However, because more code space is required for the new design application's interrupt vector table and the actual code space depends on the particular situation, it cannot be confirmed which application possesses the absolute advantage in code space.

Code 7. Code Space of General Design Source Code

```
Symbol Name              Value         Ov Type       Size     Object(Section)
EXINT3_IRQHand           0x00000375    Thumb Code    22       interrupts_fm0p.o(.text)
EXINT2_IRQHand           0x0000038b    Thumb Code    12       interrupts_fm0p.o(.text)
INT2_3_IRQHandle         0x00000397    Thumb Code    42       interrupts_fm0p.o(.text)
```

Code 8. Code Space of New Design Source Code

```
Symbol Name              Value         Ov Type       Size     Object(Section)
EXINT3_IRQHandle         0x00000375    Thumb Code     22       interrupts_fm0p.o(.text)
EXINT2_IRQHandle         0x0000038b    Thumb Code     12       interrupts_fm0p.o(.text)
INT2_3_IRQHandler        0x00000397    Thumb Code     8        interrupts_fm0p.o(.text)
```

5.2 Execution Speed

To determine which application has a higher execution speed, a GPIO is configured for the test flow. In the test flow, the GPIO outputs HIGH in system initialization. A falling-edge input at the INT03_2 pin triggers the external interrupt. The GPIO outputs LOW after entering the external interrupt 3 handler (sub interrupt handler). Then the time between the falling edge at INT03_2 and falling edge at GPIO is the record time.

As Figure 5-1 and Figure 5-2 show, when the system clock is 40 MHz, the record time for the general design application is 1.750 µs. The execution time for outputting LOW at the GPIO is about 0.325 µs. Removing the execution time of outputting LOW at the GPIO, the final interrupt branch time for the general design application is 1.425 µs. Similarly, the final interrupt branch time for the new design application is 1.259 µs.

Refer to the "Summary" in Table 5-1. It confirms that the new design application possesses an absolute advantage in execution speed.

Figure 5-1. Branch Time for General Design Application

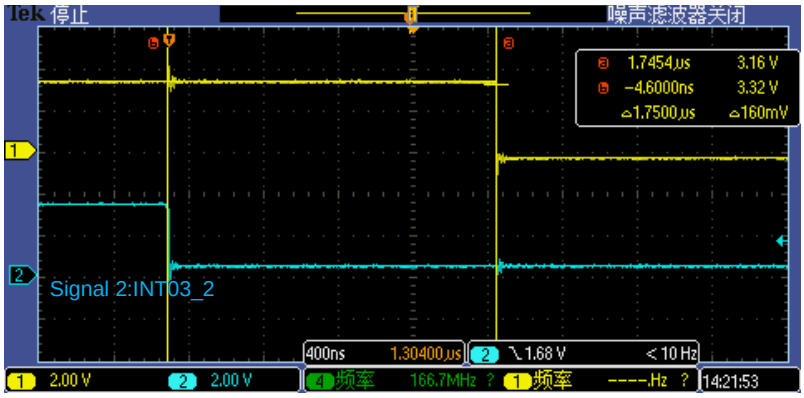


Figure 5-2. Branch Time for New Design Application

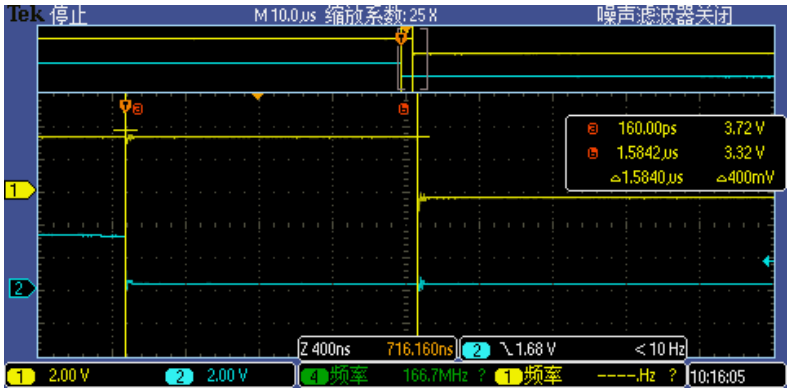


Table 5-1. Execution Speed Summary

Application	Record Time	Time for Output LOW	Interrupt Branch Time	Summary
General design	1.750 µs	0.325 µs	1.425 µs	Time saved in new design application is 0.166 µs, about 12 percent.
New design	1.584 µs	0.325 µs	1.259 µs	

5.3 Coding Complexity

As Code 5 for the general design shows, the source code of the interrupt handler will read IRQ17MON and switch to the corresponding subfunction. Thus, you must code a different function for each interrupt handler.

As Code 6 for the new design shows, the source code for each interrupt handler has the same flow chart.

With the interrupt vector table accomplished, the new design application possesses the absolute advantage in coding complexity.

6 Summary

This application note compared the performance of applications based on the general design and new design of the NVIC. It concluded that if code space is not sensitive for an application, the interrupt module based on the new NVIC design is a superior choice.

7 Precaution for Use

For the new design application, make sure that the interrupt vector table is correct. Otherwise, it will cause a serious execution error in the system.

Appendix A. Exceptions and Interrupts

Table A-1 lists exceptions and interrupts to be input to the NVIC. Following are the details of the columns in the table.

Exception No.

This is the exception number of the NVIC.

IRQ No.

This is the peripheral interrupt number of the NVIC (IRQ No. = Exception No. 16).

Vector Offset

This shows the storage address of the interrupt vector that is referred to when an interrupt occurs. The described value + Vector Table Offset Register (VTOR) in the NVIC is referred to.

Batch read register – Name

This is the name of the Batch Read Register. A "–" in this column indicates that there is no Batch Read Register for that exception or interrupt.

Batch read register – bit

This shows which bit of the Batch Read Register is assigned to each exception and interrupt from the peripheral function. A "–" in this column indicates that there is no Batch Read Register for that exception or interrupt.

VIR value

This shows the read value from VIRxx when an interrupt occurs. The described value + VIR_OFFSET is read. A "–" in this column indicates that there is no VIRxx.

Exception or Interrupt name

This shows the name of the exception or interrupt from the peripheral functions.

DSTC

This shows the support of DMA transfer by DSTC. It is skipped in this document.

Table A-1. Exception and Interrupt Sources (1 of 3)

Exception No.	IRQ No.	Vector Offset	Batch read register		VIR value	Exception or Interrupt name	DSTC
			Name	bit			
0	-	0x000	-	-	-	(Stack pointer initial value)	-
1	-	0x004	-	-	-	Reset	-
2	-	0x008	EXC02MON	0	-	Non-maskable interrupt (NMI)	-
				1	-	Hardware watchdog timer interrupt	-
3	-	0x00C	-	-	-	Reserved	-
4	-	0x010	-	-	-	Reserved	-
5	-	0x014	-	-	-	Reserved	-
6	-	0x018	-	-	-	Reserved	-
7	-	0x01C	-	-	-	Reserved	-
8	-	0x020	-	-	-	Reserved	-
9	-	0x024	-	-	-	Reserved	-
10	-	0x028	-	-	-	Reserved	-
11	-	0x02C	-	-	-	SVCall(supervisor call)	-
12	-	0x030	-	-	-	Reserved	-
13	-	0x034	-	-	-	Reserved	-
14	-	0x038	-	-	-	PendSV	-
15	-	0x03C	-	-	-	SysTick	-
16	0	0x040	IRQ00MON	0	0x0C0	Anomalous frequency detection interrupt by CSV	-
				1	0x140	Software watchdog timer interrupt	-
				2	0x1C0	Low-voltage detection (LVD) interrupt	-
17	1	0x044	IRQ01MON	0	0x0C4	MFS ch.0 reception interrupt	0
				1	0x144	MFS ch.0 transmission interrupt	1
				2	0x1C4	MFS ch.0 status interrupt	-
18	2	0x048	IRQ02MON	0	0x0C8	MFS ch.1 reception interrupt	2
				1	0x148	MFS ch.1 transmission interrupt	3
				2	0x1C8	MFS ch.1 status interrupt	-
19	3	0x04C	IRQ03MON	0	0x0CC	Reserved	-
				1	0x14C	Reserved	-
				2	0x1CC	Reserved	-
20	4	0x050	IRQ04MON	0	0x0D0	MFS ch.3 reception interrupt	6
				1	0x150	MFS ch.3 transmission interrupt	7
				2	0x1D0	MFS ch.3 status interrupt	-
21	5	0x054	IRQ05MON	0	0x0D4	MFS ch.4 reception interrupt	8
				1	0x154	MFS ch.4 transmission interrupt	9
				2	0x1D4	MFS ch.4 status interrupt	-
22	6	0x058	IRQ06MON	0	0x0D8	Reserved	-
				1	0x158	Reserved	-
				2	0x1D8	Reserved	-
23	7	0x05C	IRQ07MON	0	0x0DC	MFS ch.6 reception interrupt	12
						I2CSLAVE reception interrupt	48
				1	0x15C	MFS ch.6 transmission interrupt	13
						I2CSLAVE transmission interrupt	49
				2	0x1DC	MFS ch.6 status interrupt	-
						I2CSLAVE status interrupt	-

Table A-1. Exception and Interrupt Sources (2 of 3)

Exception No.	IRQ No.	Vector Offset	Batch read register		VIR value	Exception or Interrupt name	DSTC
			Name	bit			
24	8	0x060	IRQ08MON	0	0x0E0	MFS ch.7 reception interrupt	14
				1	0x160	MFS ch.7 transmission interrupt	15
				2	0x1E0	MFS ch.7 status interrupt	-
25	9	0x064	IRQ09MON	0	0x0E4	A/D converter unit0 priority conversion interrupt	50
				1	0x164	A/D converter unit0 scan conversion interrupt	51
				2	0x1E4	A/D converter unit0 FIFO overrun interrupt	-
						A/D converter unit0 conversion result comparison int.	-
						A/D converter unit0 range comparison result int.	-
26	10	0x068	IRQ10MON	0	0x0E8	USB ch.0 device endpoint1 DRQ interrupt	52
				1	0x168	USB ch.0 device endpoint2 DRQ interrupt	53
				2	0x1E8	USB ch.0 device endpoint3 DRQ interrupt	54
27	11	0x06C	IRQ11MON	0	0x0EC	USB ch.0 device endpoint4 DRQ interrupt	55
				1	0x16C	USB ch.0 device endpoint5 DRQ interrupt	56
				2	0x1EC	USB ch.0 device endpoint0 DRQI interrupt	-
28	12	0x070	IRQ12MON	0	0x0F0	USB ch.0 device endpoint0 DRQO interrupt	-
				1	0x170	USB ch.0 device SUSP interrupt	-
						USB ch.0 device SOF interrupt	-
						USB ch.0 device BRST interrupt	-
						USB ch.0 device CONF interrupt	-
						USB ch.0 device WKUP interrupt	-
				2	0x1F0	USB ch.0 device SPK interrupt	-
29	13	0x074	IRQ13MON	0	0x0F4	USB ch.0 host DIRQ interrupt	-
						USB ch.0 host URIRQ interrupt	-
						USB ch.0 host RWKIRQ interrupt	-
						USB ch.0 host CNNIRQ interrupt	-
				1	0x174	USB ch.0 host SOFIRQ interrupt	-
						USB ch.0 host CMPIRQ interrupt	-
				2	0x1F4	Reserved	-
30	14	0x078	IRQ14MON	0	0x0F8	Main PLL oscillation stabilization wait completion int.	-
						Main clock oscillation stabilization wait completion int.	-
						Sub clock oscillation stabilization wait completion int.	-
				1	0x178	Reserved	-
				2	0x1F8	Reserved	-
31	15	0x07C	IRQ15MON	0	0x0FC	Watch counter interrupt	57
				1	0x17C	Real timer counter (RTC) interrupt	-
						Dual timer ch.1 interrupt	-
						Dual timer ch.2 interrupt	-
				2	0x1FC	Reserved	-

Table A-1. Exception and Interrupt Sources (3 of 3)

Exception No.	IRQ No.	Vector Offset	Batch read register		VIR value	Exception or Interrupt name	DSTC
			Name	bit			
32	16	0x080	IRQ16MON	0	0x100	External pin interrupt ch.0	16
				1	0x180	External pin interrupt ch.1	17
33	17	0x084	IRQ17MON	0	0x104	External pin interrupt ch.2	18
				1	0x184	External pin interrupt ch.3	19
34	18	0x088	IRQ18MON	0	0x108	External pin interrupt ch.4	20
				1	0x188	External pin interrupt ch.5	21
35	19	0x08C	IRQ19MON	0	0x10C	External pin interrupt ch.6	22
				1	0x18C	External pin interrupt ch.7	23
36	20	0x090	IRQ20MON	0	0x110	External pin interrupt ch.8	24
				1	0x190	Reserved	-
37	21	0x094	IRQ21MON	0	0x114	Reserved	-
				1	0x194	Reserved	-
38	22	0x098	IRQ22MON	0	0x118	External pin interrupt ch.12	28
				1	0x198	External pin interrupt ch.13	29
39	23	0x09C	IRQ23MON	0	0x11C	Reserved	-
				1	0x19C	External pin interrupt ch.15	31
40	24	0x0A0	IRQ24MON	0	0x120	Base timer ch.0 source0 (IRQ0) interrupt	32
						Base timer ch.0 source1 (IRQ1) interrupt	33
				1	0x1A0	Base timer ch.4 source0 (IRQ0) interrupt	34
						Base timer ch.4 source1 (IRQ1) interrupt	35
41	25	0x0A4	IRQ25MON	0	0x124	Base timer ch.1 source0 (IRQ0) interrupt	36
						Base timer ch.1 source1 (IRQ1) interrupt	37
				1	0x1A4	Base timer ch.5 source0 (IRQ0) interrupt	38
						Base timer ch.5 source1 (IRQ1) interrupt	39
42	26	0x0A8	IRQ26MON	0	0x128	Base timer ch.2 source0 (IRQ0) interrupt	40
						Base timer ch.2 source1 (IRQ1) interrupt	41
				1	0x1A8	Base timer ch.6 source0 (IRQ0) interrupt	42
						Base timer ch.6 source1 (IRQ1) interrupt	43
43	27	0x0AC	IRQ27MON	0	0x12C	Base timer ch.3 source0 (IRQ0) interrupt	44
						Base timer ch.3 source1 (IRQ1) interrupt	45
				1	0x1AC	Base timer ch.7 source0 (IRQ0) interrupt	46
						Base timer ch.7 source1 (IRQ1) interrupt	47
44	28	0x0B0	IRQ28MON	0	0x130	CEC Reception/Remote reception ch.0 interrupt	-
						CEC Transmission ch.0 interrupt	-
				1	0x1B0	CEC Reception/Remote reception ch.1 interrupt	-
						CEC Transmission ch.1 interrupt	-
45	29	0x0B4	IRQ29MON	0	0x134	Smart Card ch.1 interrupt	-
				1	0x1B4	FLASH memory RDY/HANG interrupt	-
46	30	0x0B8	IRQ30MON	0	0x138	DSTC SW transfer complete interrupt	-
				1	0x1B8	DSTC error interrupt	-
47	31	0x0BC	IRQ31MON	0	0x13C	Reserved	-
				1	0x1BC	Reserved	-

Document History

Document Title: AN99231 – Using Interrupts in FM0+ Family S6E1C3 Series

Document Number: 001-99231

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	4925847	MIWA	09/28/2015	New application note
*A	5874856	AESATMP9	09/06/2017	Updated logo and copyright.
*B	6346291	YOST	10/12/2018	No change; sunset review.

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 MCU

Cypress Developer Community

Community | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

All other trademarks or registered trademarks referenced herein are the property of their respective owners.

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

© Cypress Semiconductor Corporation, 2015-2018. This document is the property of Cypress Semiconductor Corporation and its subsidiaries, including Spansion LLC ("Cypress"). This document, including any software or firmware included or referenced in this document ("Software"), is owned by Cypress under the intellectual property laws and treaties of the United States and other countries worldwide. Cypress reserves all rights under such laws and treaties and does not, except as specifically stated in this paragraph, grant any license under its patents, copyrights, trademarks, or other intellectual property rights. If the Software is not accompanied by a license agreement and you do not otherwise have a written agreement with Cypress governing the use of the Software, then Cypress hereby grants you a personal, non-exclusive, nontransferable license (without the right to sublicense) (1) under its copyright rights in the Software (a) for Software provided in source code form, to modify and reproduce the Software solely for use with Cypress hardware products, only internally within your organization, and (b) to distribute the Software in binary code form externally to end users (either directly or indirectly through resellers and distributors), solely for use on Cypress hardware product units, and (2) under those claims of Cypress's patents that are infringed by the Software (as provided by Cypress, unmodified) to make, use, distribute, and import the Software solely for use with Cypress hardware products. Any other use, reproduction, modification, translation, or compilation of the Software is prohibited.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS DOCUMENT OR ANY SOFTWARE OR ACCOMPANYING HARDWARE, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. No computing device can be absolutely secure. Therefore, despite security measures implemented in Cypress hardware or software products, Cypress does not assume any liability arising out of any security breach, such as unauthorized access to or use of a Cypress product. In addition, the products described in these materials may contain design defects or errors known as errata which may cause the product to deviate from published specifications. To the extent permitted by applicable law, Cypress reserves the right to make changes to this document without further notice. Cypress does not assume any liability arising out of the application or use of any product or circuit described in this document. Any information provided in this document, including any sample design information or programming code, is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Cypress products are not designed, intended, or authorized for use as critical components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or system could cause personal injury, death, or property damage ("Unintended Uses"). A critical component is any component of a device or system whose failure to perform can be reasonably expected to cause the failure of the device or system, or to affect its safety or effectiveness. Cypress is not liable, in whole or in part, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from or related to all Unintended Uses of Cypress products. You shall indemnify and hold Cypress harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of Cypress products.

Cypress, the Cypress logo, Spansion, the Spansion logo, and combinations thereof, WICED, PSoC, CapSense, EZ-USB, F-RAM, and Traveo are trademarks or registered trademarks of Cypress in the United States and other countries. For a more complete list of Cypress trademarks, visit cypress.com. Other names and brands may be claimed as property of their respective owners.