Section 3. Exception Processing

NVIC and Its Functions

Outline

- Basic Exception Concepts
- Nested Vector Interrupt Controller
- Registers and Vector table
- Example module
  - Timer Module

Basic Concepts of Interrupt Processing

- A variety of unexpected events in a computer system
  - I/O events, error conditions, network events etc
- These events are handled by interrupt processing
  - Speed disparity of various devices in a computer
  - Allow multiple and parallel processing of tasks
- An analogous example: A reading process
  - Phone rings--→ Recognition of an event
  - Answer the phone or not? → Priority
  - Book mark the page → store context
  - Answer the phone → handler
  - Continue the reading process after phone conversation → done

Necessary Procedure of an Interrupt Process

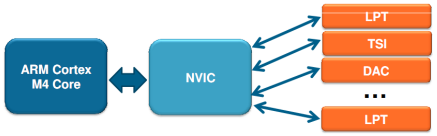
- Interrupt requester ←→ CPU
- Recognition of an interrupt request:
  - Interrupt requester makes an interrupt request
  - CPU recognizes the interrupt request
- Prioritization: Determining whether granting the request or not
  - Requester provides its priority
  - CPU compares it with the priority of its current process
- Context saving to be able to come back after interrupt
  - Program counter marks where interrupt happened
  - Status register and possible other necessary context information

NVIC (Nested Vector Interrupt Controller)

- Low latency Interrupt
  - 12 cycles to PUSH on ISR entry
  - 12 cycles to POP on ISR exit
- Up to 120 interrupt sources
  - Includes 16 ARM core specific exceptions
  - The remaining are modules specific
- Arm supports 0-255 priority levels, K-70 support 16 priority levels, all fully programmable (0 highest)
  - Reset, NMI and Hard Fault have predefined priority
- Change Interrupt Priority dynamically
- Relocable vector table

SoC interconnect diagram

- Any module capable of generating an interrupt will depend on NVIC operation
- NMI can be generated from:
  - External pin (must configure the MUX)
  - CoreSight Embedded Trace Buffer (ETB)



Exceptions and Interrupts

- Exception can be caused by the execution of an exception generating instruction or triggered as a response to a system behavior such as an interrupt, memory management protection violation, alignment or bus fault, or a debug event.

  Synchronous and asynchronous exceptions can occur within the architecture.

Table B1-4 Exception numbers

Exception number	Exception
1	Reset
2	NMI
3	HardFault
4	MemManage
5	BusFault
6	UsageFault
7-10	Reserved
11	SVCall
12	Debug Monitor
13	Reserved
14	PendSV
15	SysTick
16	External interrupt 0
.	.
.	.
.	.
16+N	External interrupt N

NVIC

- The NVIC supports up to 240 interrupts each with up to 256 levels of priority. You can change the priority of an interrupt dynamically

Table 6-1 NVIC registers

Address	Name	Type	Reset	Description
0xE000E004	ICTR	RO	-	*Interrupt Controller Type Register, ICTR*
0xE000E100 - 0xE000E11C	NVIC_ISER0 - NVIC_ISER7	RW	0x00000000	Interrupt Set-Enable Registers
0xE000E180 - 0xE000E19C	NVIC_ICER0 - NVIC_ICER7	RW	0x00000000	Interrupt Clear-Enable Registers
0xE000E200 0xE000E21C	NVIC_ISPR0 NVIC_ISPR7	RW	0x00000000	Interrupt Set Pending Registers
0xE000E280 - 0xE000E29C	NVIC_ICPR0 - NVIC_ICPR7	RW	0x00000000	Interrupt Clear-Pending Registers
0xE000E300 - 0xE000E31C	NVIC_IABR0 - NVIC_IABR7	RO	0x00000000	Interrupt Active Bit Register
0xE000E400 - 0xE000E41F	NVIC_IPR0 - NVIC_IPR59	RW	0x00000000	Interrupt Priority Register

Wake-up Sources

Wake-up source	Description
Available system resets	RESET pin and WDOG when LPO is its clock source, and JTAG
Low-voltage detect	Mode Controller
Low-voltage warning	Mode Controller
Pin interrupts	Port Control Module - Any enabled pin interrupt is capable of waking the system
ADCx	The ADC is functional when using internal clock source
CMPx	Since no system clocks are available, functionality is limited
I²C	Address match wakeup
UART	Active edge on RXD
USB	Wakeup
LPTMR	Functional in Stop/VLPS modes
RTC	Functional in Stop/VLPS modes
Ethernet	Magic Packet wakeup

Wake-up source	Description
SDHC	Wakeup
I2S	Functional when using an external bit clock or external master clock
1588 Timer	Wakeup
TSI	
CAN	
Tamper detect	Interrupt or a reset

Sequence of register setups

The steps for enabling an interrupt on NVIC:

- 1. Enable the peripheral to be used
- 2. Set the proper bit on the NVICSERx to enable the interrupt on the NVIC
- 3. Clear any pending interrupt by writing to the NVICCPRx to avoid any spurious interrupt
- 4. Configure the interrupt priority by writing to the NVICIPxx
- 5. Write the ISR
- 6. Enable global interrupts

Example Registers: NVIC_ISER0-7

Bits Name

[31:0] SETENA

Function

Interrupt set-enable bits.

Write: 0 = no effect

1 = enable interrupt.

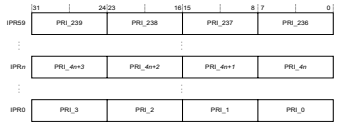
Read: 0 = interrupt disabled

1 = interrupt enabled.

Example Registers

Interrupt Priority Registers

- The NVIC_IPR0-NVIC_IPR59 registers provide an 8-bit priority field for each interrupt and each register holds four priority fields. These registers are byte-accessible.



K70 supports 16 priority levels for interrupts. Therefore, in the NVIC each source in the IPR registers contains 4 bits. For example, IPR0 is shown below:

Picking up a IPR

Find the IPR number and byte offset for interrupt *m* as follows:

- the corresponding IPR number, *n* is given by *n* = *m* DIV 4
- the byte offset of the required Priority field in this register is *m* MOD 4, where:

  byte offset 0 refers to register bits[7:0]
  byte offset 1 refers to register bits[15:8]
  byte offset 2 refers to register bits[23:16]
  byte offset 3 refers to register bits[31:24].

Table 3-4. Interrupt vector assignments

Address	Vector	IRQ[1]	NVIC non-IPR register number [2]	NVIC IPR register number [3]	Source module	Source description
ARM Core System Handler Vectors						
0x0000_0000	0	--	--	--	ARM core	Initial Stack Pointer
0x0000_0004	1	--	--	--	ARM core	Initial Program Counter
0x0000_0008	2	--	--	--	ARM core	Non-maskable Interrupt (NMI)
0x0000_000C	3	--	--	--	ARM core	Hard Fault
0x0000_0010	4	--	--	--	ARM core	MemManage Fault
0x0000_0014	5	--	--	--	ARM core	Bus Fault
0x0000_0018	6	--	--	--	ARM core	Usage Fault
0x0000_001C	7	--	--	--	---	---
0x0000_0020	8	--	--	--	---	---
0x0000_0024	9	--	--	--	---	---
0x0000_0028	10	--	--	--	---	---
0x0000_002C	11	--	--	--	ARM core	Supervisor call (SVCall)
0x0000_0030	12	--	--	--	ARM core	Debug Monitor
0x0000_0034	13	--	--	--	---	---
0x0000_0038	14	--	--	--	ARM core	Pendable request for system service (PendableSrvReq)
0x0000_003C	15	--	--	--	ARM core	System tick timer (SysTick)
Non-Core Vectors						
0x0000_0040	16	0	0	0	DMA	DMA channel 0, 16 transfer complete
0x0000_0044	17	1	0	0	DMA	DMA channel 1, 17 transfer complete

0x0000_016C	91	75	2	18	Ethernet MAC	IEEE 1588 Timer Interrupt
0x0000_0170	92	76	2	19	Ethernet MAC	Transmit interrupt
0x0000_0174	93	77	2	19	Ethernet MAC	Receive interrupt
0x0000_0178	94	78	2	19	Ethernet MAC	Error and miscellaneous interrupt
0x0000_017C	95	79	2	19	---	---
0x0000_0180	96	80	2	20	SDHC	---
0x0000_0184	97	81	2	20	DAC0	---
0x0000_0188	98	82	2	20	DAC1	---
0x0000_018C	99	83	2	20	TSI	Single interrupt vector for all sources
0x0000_0190	100	84	2	21	MCG	---
0x0000_0194	101	85	2	21	Low Power Timer	---
0x0000_0198	102	86	2	21	---	---
0x0000_019C	103	87	2	21	Port control module	Pin detect (Port A)
0x0000_01A0	104	88	2	22	Port control module	Pin detect (Port B)
0x0000_01A4	105	89	2	22	Port control module	Pin detect (Port C)

Sample code Explained

- Set up the LPT interrupt:
  - Locate the interrupt vector that you want on the Vector Table list from the Kinetis device used

Address	Vector	IRQ	Source module	Source description
0x0000_018C	99	83	TSI	Single interrupt vector for all sources
0x0000_0190	100	84	MCG	
0x0000_0194	101	85	Low Power Timer	

- Find the NVIC Interrupt Set Enable Register (NVICISERx) for your vector:
  - NVICISER2 for the LPT
- Then take the modulo value of your IRQ number by 32 to calculate which bit to set in the NVICISER2 register
  - 85%32 = 21
  - *NVICISER2 |=(1<<21); //Enable LPT interrupts*

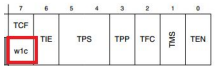
Sample code Explained cont.

- Clear any pending interrupts from the NVICICPRx
  - Use the same modulo result for knowing which bit to set

    NVICICPR2|=(1<<21); //Clear any pending interrupts on
- Set the interrupt priority writing to the NVICIPx
  - X is the IRQ number
  - Just the 4 most significant bits are used

    NVICIP85 = 0x30; //Set Priority 3 to the LPT module
- Write your ISR

```
Void lptmr_isr()
{   LPT0_CSR|=LPT_CSR_TCF_MASK; //Clear LPT Compare flag
    LPTMR0_CSR = ( LPTMR_CSR_TEN_MASK | //enable timer
                   LPTMR_CSR_TIE_MASK | //enable interrupt
                   LPTMR_CSR_TCF_MASK );//clear flag
}
```



NVIC Prog Hints

void __disable_irq(void) // Disable Interrupts
void __enable_irq(void) // Enable Interrupts

CMSIS interrupt control function	Description
`void NVIC_SetPriorityGrouping(uint32_t priority_grouping)`	Set the priority grouping
`void NVIC_EnableIRQ(IRQn_t IRQn)`	Enable IRQn
`void NVIC_DisableIRQ(IRQn_t IRQn)`	Disable IRQn
`uint32_t NVIC_GetPendingIRQ (IRQn_t IRQn)`	Return true (IRQ-Number) if IRQn is pending
`void NVIC_SetPendingIRQ (IRQn_t IRQn)`	Set IRQn pending
`void NVIC_ClearPendingIRQ (IRQn_t IRQn)`	Clear IRQn pending status
`uint32_t NVIC_GetActive (IRQn_t IRQn)`	Return the IRQ number of the active interrupt
`void NVIC_SetPriority (IRQn_t IRQn, uint32_t priority)`	Set priority for IRQn
`uint32_t NVIC_GetPriority (IRQn_t IRQn)`	Read priority of IRQn
`void NVIC_SystemReset (void)`	Reset the system