Interrupt Control and System Control

Overview

- *NVIC and System Control Block*
- *Interrupt Enable and Clear Enable*
- *Example to Enable and clear interrupt #2*
- *Interrupt Pending and Clear Pending*
- *Example to trigger pending and clear pending on interrupt #2*
- *Interrupt Priority Level*
- *Example to set the priority of level of interrupt #2 to 0xC0*
- *Exception Masking Register*

NVIC features

- Flexible interrupt management include enable/disable, priority configurations
- Hardware nested interrupt support
- Vectored exception entry
- Interrupt masking

Overview of the NVIC and System Control Block

- The NVIC in the Cortex-M0 processor supports up to **32 external interrupts** and one **nonmaskable interrupt (NMI).**
- The interrupt input requests can be level triggered, or they can be pulsed with a minimum of one clock cycle.
- Each external **interrupt can be enabled or disabled** independently, and its pending status can also be set or clear manually.
- The NVIC control registers are **memory mapped** and can be easily accessed in C language.
- The location of the **NVIC registers starts from address 0xE000E100**. For the Cortex-M0 processor, the accesses to the NVIC register must be in word size.
- Similar to the NVIC registers, the **SCB registers are also word accessible, and the address starts from 0xE000E010**.
- The SCB registers handle features like the SysTick timer operations, system exception management and priority control, and sleep mode control.

Interrupt Enable and Clear Enable

Table 9.1: Interrupt Enable Set and Clear Register

Address	Name	Type	Reset Value	Descriptions
0xE000E100	SETENA	R/W	0x00000000	Set enable for interrupt 0 to 31. Write 1 to set bit to 1, write 0 has no effect. Bit[0] for Interrupt #0 (exception #16) Bit[1] for Interrupt #1 (exception #17) ... Bit[31] for Interrupt #31 (exception #47) Read value indicates the current enable status
0xE000E180	CLRENA	R/W	0x00000000	Clear enable for interrupt 0 to 31. Write 1 to clear bit to 0, write 0 has no effect. Bit[0] for Interrupt #0 (exception #16) ... Bit[31] for Interrupt #31 (exception #47) Read value indicates the current enable status

Interrupt Enable and Clear Enable

To Enable interrupt #2

- **NVIC with one access:**

```
*((volatile unsigned long *)(0xE000E100))=0x4; //Enable interrupt #2
```

- **Assembly**

```
LDR R0,=0xE000E100 ; Setup address in R0
MOVS R1,#0x4 ; interrupt #2
STR R1,[R0] ; write to set interrupt enable
```

- **CMSIS**

```
void NVIC_EnableIRQ(IRQn_Type IRQn);
 // Enable Interrupt // IRQn value of 0 refer to Interrupt #0
```

Interrupt Enable and Clear Enable

To Disable interrupt #2

- **NVIC with one access:**

*((volatile unsigned long *)(0xE000E180))=0x4; //Enable interrupt #2

- **Assembly**

LDR R0,=0xE000E180 ; Setup address in R0

MOVS R1,#0x4 ; interrupt #2

STR R1,[R0] ; write to set interrupt enable

- **CMSIS**

void NVIC_DisableIRQ(IRQn_Type IRQn);

// Disable Interrupt // IRQn value of 0 refer to Interrupt #0

Interrupt Pending and Clear Pending

- If an interrupt takes place but cannot be processed immediately ,for example, if the processor is serving a higher-priority interrupt ,the interrupt request will be pended.
- The pending status is held in a register and will remain valid until the current priority of the processor is lowered so that the pending request is accepted or until the application clears the pending status manually.
- The interrupt pending status can be accessed or modified, through the Interrupt Set Pending (SETPEND) and Interrupt Clear Pending (CLRPEND) register addresses (Table 9.2).
- Similar to the Interrupt Enable control register, the Interrupt Pending status register is physically one register, but it uses two addresses to handle the set and clear the bits.

Interrupt Pending and Clear Pending

Table 9.2: Interrupt Pending Set and Clear Register

Address	Name	Type	Reset Value	Descriptions
0xE000E200	SETPEND	R/W	0x00000000	Set pending for interrupt 0 to 31. Write 1 to set bit to 1, write 0 has no effect. Bit[0] for Interrupt #0 (exception #16) Bit[1] for Interrupt #1 (exception #17) ... Bit[31] for Interrupt #31 (exception #47) Read value indicates the current pending status
0xE000E280	CLRPEND	R/W	0x00000000	Clear pending for interrupt 0 to 31. Write 1 to clear bit to 0, write 0 has no effect. Bit[0] for Interrupt #0 (exception #16) ... Bit[31] for Interrupt #31 (exception #47) Read value indicates the current pending status

Interrupt Pending and Clear Pending to trigger interrupt #2

- **NVIC**

*((volatile unsigned long *)(0xE000E100))=0x4; //Enable interrupt #2

*((volatile unsigned long *)(0xE000E200))=0x4; //Pend interrupt #2

- **Assembly**

```
LDR R0,=0xE000E100 ; Setup address in R0
MOVS R1,#0x4 ; interrupt #2
STR R1,[R0] ; write to set interrupt enable
LDR R0,=0xE000E200 ; Setup address in R0
STR R1,[R0] ; write to set pending status
```

- **CMSIS**

void NVIC_SetPendingIRQ(IRQn_Type IRQn); // Set pending status of a interrupt

Interrupt Pending and Clear Pending
to clear interrupt #2

- **NVIC**

*((volatile unsigned long *)(0xE000E280))¼0x4;// Clear interrupt #2
// pending status

- **Assembly**

```
LDR R0,=0xE000E280 ; Setup address in R0
MOVS R1,#0x4 ; interrupt #2
STR R1,[R0] ; write to clear pending status
```

- **CMSIS**

void NVIC_ClearPendingIRQ(IRQn_Type IRQn); // clear pending status of a interrupt

Interrupt Priority Level

- Each external interrupt has an associated priority-level register.
- Each of them is 2 bits wide, occupying the two MSBs of the Interrupt Priority Level Registers.
- Each Interrupt Priority Level Register occupies 1 byte (8 bits).
- NVIC registers in the Cortex-M0 processor can only be accessed using word-size transfers, so for each access, four Interrupt Priority Level Registers are accessed at the same time (Figure 9.1).
- The unimplemented bits are read as zero. Write values to those unimplemented bits are ignored, and read values of the unimplemented bits return zeros (Table 9.3).

Interrupt Priority Level

Bit	31 30 … 24	23 22 … 16	15 14 … 8	7 6 … 0
0xE000E41C	31	30	29	28
0xE000E418	27	26	25	24
0xE000E414	23	22	21	20
0xE000E410	19	18	17	16
0xE000E40C	15	14	13	12
0xE000E408	11	10	9	8
0xE000E404	7	6	5	4
0xE000E400	IRQ 3	IRQ 2	IRQ 1	IRQ 0

Figure 9.1:

Interrupt Priority Level Registers for each interrupt.

Interrupt Priority Level

Table 9.3: Interrupt Priority Level Registers (0xE000E400–0xE000E41C)

Address	Name	Type	Reset Value	Descriptions
0xE000E400	IPR0	R/W	0x00000000	Priority level for interrupt 0 to 3 [31:30] Interrupt priority 3 [23:22] Interrupt priority 2 [15:14] Interrupt priority 1 [7:6] Interrupt priority 0
0xE000E404	IPR1	R/W	0x00000000	Priority level for interrupt 4 to 7 [31:30] Interrupt priority 7 [23:22] Interrupt priority 6 [15:14] Interrupt priority 5 [7:6] Interrupt priority 4
0xE000E408	IPR2	R/W	0x00000000	Priority level for interrupt 8 to 11 [31:30] Interrupt priority 11 [23:22] Interrupt priority 10 [15:14] Interrupt priority 9 [7:6] Interrupt priority 8
0xE000E40C	IPR3	R/W	0x00000000	Priority level for interrupt 12 to 15 [31:30] Interrupt priority 15 [23:22] Interrupt priority 14 [15:14] Interrupt priority 13 [7:6] Interrupt priority 12
0xE000E410	IPR4	R/W	0x00000000	Priority level for interrupt 16 to 19
0xE000E414	IPR5	R/W	0x00000000	Priority level for interrupt 20 to 23
0xE000E418	IPR6	R/W	0x00000000	Priority level for interrupt 24 to 27
0xE000E41C	IPR7	R/W	0x00000000	Priority level for interrupt 28 to 31

Interrupt Priority Level

- For example, if we want to set the priority level of interrupt #2 to 0xC0, we can do it by using the following code:
- **NVIC**

```
unsigned long temp; // a temporary variable
temp = *((volatile unsigned long *)(0xE000E400)); // Get IPR0
temp = temp & (0xFF00FFFF) j (0xC0 << 16); // Change Priority level
*((volatile unsigned long *)(0xE000E400)) = temp; // Set IPR0
```

Interrupt Priority Level

- For example, if we want to set the priority level of interrupt #2 to 0xC0, we can do it by using the following code:
- **Assembly**

```
LDR R0,=0xE000E400 ; Setup address in R0
LDR R1,[R0] ; Get PRIORITY0
LDR R2,=0x00FF0000 ; Mask for interrupt #2's priority
BICS R1, R1, R2 ; R1 = R1 AND (NOT(0x00FF0000))
LDR R2,=0x00C00000 ; New value for interrupt #2's priority
ORRS R1, R1, R2 ; Put new priority level
STR R1,[R0] ; write back value
```

Interrupt Priority Level

- For example, if we want to set the priority level of interrupt #2 to 0xC0, we can do it by using the following code:
- **CMSIS-compliant** device driver libraries, the interrupt priority level can be accessed by two functions:

```
void NVIC_SetPriority(IRQn_Type IRQn, uint32_t priority); // Set the priority
// level of an interrupt or a system exception
uint32_t NVIC_GetPriority(IRQn_Type IRQn); // return the priority
// level of an interrupt or a system exception
```

Interrupt Priority Level

- Note that these two functions automatically shift the priority level values to the implemented bits of the priority level registers.
- Therefore, when we want to set the priority value of interrupt #2 to 0xC0,
- we should use this code:

  NVIC_SetPriority(2, 0x3); // priority value 0x3 is shifted to become 0xC0

Exception Masking Register (PRIMASK)

- In some applications, it is necessary to disable all interrupts for a short period of time for time critical processes.
- Instead of disabling all interrupts and restoring them using the interrupt enable/disable control register, the Cortex-M0 processor provides a separate feature for this usage.
- One of the special registers, called PRIMASK can be used to mask all interrupts and system exceptions, apart from the NMI and hard fault exceptions.

Exception Masking Register (PRIMASK)

- The PRIMASK is a single bit register and is set to 0 at reset.
- When set to 0, interrupts and system exceptions are allowed.
- When set to 1, only NMI and hard fault exceptions are allowed.
- Effectively, when it is set to 1, it changes the current priority level to 0.
- Set or clear the PRIMASK register using the MSR instruction.

```
MOVS R0, #1 ; New value for PRIMASK
MSR PRIMASK, R0 ; Transfer R0 value to PRIMASK
```