10
Interrupt Structure of Atmega328P MCU

10.1 Introduction

(1) Assume that the MCU has been continuously blinking L of Fig-10.1 at 1-sec interval. Let us designate this blinking job as the Main Line Program (MLP). **Interrupting** the MCU means telling it to suspend the MLP, turn to the side job (called Interrupt Sub Routine, ISR) to blink LED1 only for 5 times at 2-sec interval, finish the ISR and then resume the MLP.

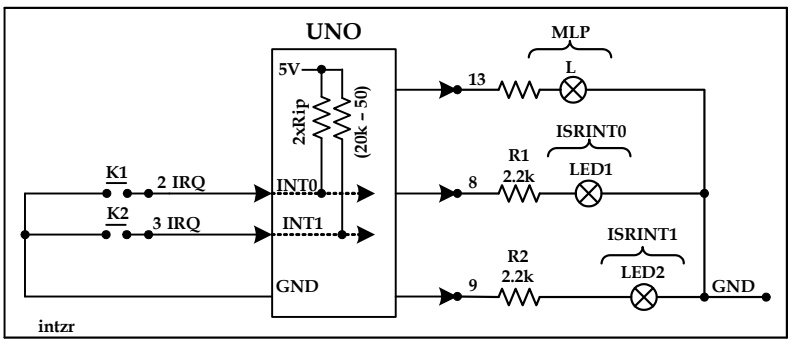

Figure-10.1: *Interrupt pins of ATmega328P MCU*

(2) To interrupt the MCU, we inject **LOW** signal (0V) at DPin-2/INT0-pin of the MCU using K1 button. The DPin-2/INT0 is known as **Interrupt Pin**; the button is known as **Interrupting Device**; the LOW signal is known as **Interrupting Signal**/Interrupt Request Signal (IRQ). This interrupt is also known as **interrupt type 0** (INT0). The internal pull-up resistor keeps the interrupt pin always at HIGH state. Because DPin-2 receives IRQ signals from external hardware devices like K1 button, we call hardware interrupt pin.

(3) The MCU has another interrupt pin (DPin-3/INT1) to receive IRQ signal from external source.

(4) It is possible to program the MCU to deny —

- **(a)** Accepting IRQ signal coming at DPin-2 pin and get interrupted (disable INT0 interrupt);
- **(b)** Accepting IRQ signal coming at DPin-3 pin and get interrupted (disable INT1 interrupt);
- **(c)** **Accepting** IRQ signal generated either from internal or external source.

(5) **Sources of IRQ signals for ATmega328P MCU**

There are about 26 sources that can place IRQ signals to the MCU. We are familiar only with the following sources of interrupts:

(a) External IRQ signal at DPin-2.

When IRQ signal arrives at DPin-2, the MCU suspends the MLP and then goes to the following ISR routine.

```
ISR (INT0_vect)
{
     //place codes here
}
```

(b) External IRQ signal at DPin-3.

When IRQ signal arrives at DPin-3, the MCU suspends the MLP and then goes to the following ISR routine.

```
ISR (INT1_vect)
{
```

```
    //place codes here
}
```

(c) Internal IRQ signal when TOV1 flag of TC1 assumes HIGH sate

When IRQ signal arrives due to HIGH state of TOV1 flag, the MCU suspends the MLP and then goes to the following ISR routine.

```
ISR (TIMER1_OVF_vect)
{
    //place codes here
}
```

(d) Internal IRQ signal when I2C/TWI bus data transfer is complete

When IRQ signal arrives due to I2C bus data transfer complete, the MCU suspends the MLP and then goes to the following ISR routine.

```
ISR (TWI_vect)
{
    //place codes here
}
```

(e) Internal IRQ signal when SPI Port data transfer is complete

When IRQ signal arrives due to SPI Port data transfer complete, the MCU suspends the MLP and then goes to the following ISR routine.

```
ISR (SPI_STC_vect)
{
    //place codes here
}
```

(f) Internal IRQ signal when UART Port data receive is complete

When IRQ signal arrives due to UART Port data receive complete, the MCU suspends the MLP and then goes to the following ISR routine.

```
ISR (UART_RX_vect)
{
    //place codes here
}
```

(g) Internal IRQ signal when UART Port data transmit is complete

When IRQ signal arrives due to UART Port data transmit complete, the MCU suspends the MLP and then goes to the following ISR routine.

```
ISR (UART_TX_vect)
{
    //place codes here
}
```

(6) Interrupt Related Arduino Instructions

(a) *(i)* `cli();  //clear (disable) global interrupt flag`

The execution of the above code tells the MCU to deny the acceptance of any IRQ signal which is generated either from internal or external source. The MCU will not suspend the MLP and will not execute any ISR routine.

(ii) `noInterrupts();  //clear (disable) global interrupt flag`

(b) *(i)* `sei();  //set (enable) global interrupt flag`

(ii) `interrupts();  //set (enable) global interrupt flag`

(c) `attachInterrupt(digitalPinToInterrupt(DPin), userDefinedISRName, triggerLevel);`

(i) The first argument configures DPin-2/DPin-3 to receive IRQ signal from external source. The DPin-2/DPin-3 is no more available as general purpose digital IO lines. The instruction automatically enables INT0 interrupt and global interrupt flag by executing **interrupts()** instruction.

(ii) The 2nd argument allows the user to declare an ISR routine whose name is automatically linked with INT0 interrupt.

(iii) The 3rd argument allows the user to accept one of the following modes for the trigger level of the external interrupting signal: **LOW, RISING, and FALLING.**

(d) `detacjInterrupt(digitalPinToInterrupt(DPin));`
The DPin2/DPin-3 is disconnected from internal interrupt logic; it is no more capable to work as an interrupt line and takes back the function of digital IO line.

10.2 Functional Check of INT0 Interrupt

(1) Upload a MLP to blink L of Fig-10.1 continuously at 1-sec interval.
(2) Add code (s) with MLP of Step-1 so that DPin-2 works as an interrupt line for INT0 interrupt.
(3) Connect K1 with DPin-2 with internal pull-up enabled.
(4) Connect R1-LED1 circuit of Fig-10.1.
(5) Add the following ISR for INT0 with MLP of Step-1 to blink LED1 only 5 times at 2-sec interval. You need to create the 2-sec time delay interval using TC1 Module.
(6) Upload the sketch that conations both MLP and ISR. Check that L is banking at 1-sec interval.
(7) Gently press the K1 button.
(8) Check that L has stopped blinking and it is OFF.
(9) Check that LED1 blinks at 2-sec interval.
(10) Check that after 5 blinks, the LED1 is OFF.
(11) Check that L has started blinking again.
(12) Repeat the interrupt process for DPin-3 for INT1 with ISR to blink LED2 for 3 times.

10.2 Registers involved in the INT0 Process

Name: EICRA **External Interrupt Control Register A**

Bit	7	6	5	4	3	2	1	0
					ISC11	ISC10	ISC01	ISC00
Access					R/W	R/W	R/W	R/W
Reset					0	0	0	0

Figure-10.2: *Bit layout of EICRA Register*

[ISC01:ISC00] : Interrupt Sense Control Bits for INT0 Interrupt to define the trigger level of IRQ signal
00 : LOW Level 10 : Falling Edge 11 : Rising Edge

Name: EIMSK **External Interrupt Mask Register**

Bit	7	6	5	4	3	2	1	0
							INT1	INT0
Access							R/W	R/W
Reset							0	0

Figure-10.3: *Bit layout of EIMSK Register*

INT0 : External Interrupt Request 0 Enable
HIGH at INT0 bit will allow the MCU to enter into ISR if **interrupts()** instruction has been executed. The MCU will not jump to the ISRINT0 (ISR due to INT0 interrupt) if LOW is stored at INT0 bit affter the execution of the **attachInterrupt()** instruction. The INT0 bit is known as local interrupt flag.

SREG **Status Register**

Bit	7	6	5	4	3	2	1	0
	I	T	H	S	V	N	Z	C
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Figure-10.4: *Bit layout of SREG Register*

When **interrupts()** instruction is executed, the I-bit of SREG Register assumes HIGH; as a result, an IRQ signal id funneled forward to interrupt the MCU. This I-bit is known as global interrupt flag.

EIFR **External Interrupt Flag Register**

Bit	7	6	5	4	3	2	1	0
							INTF1	INTF0
Access							R/W	R/W
Reset							0	0

Figure-10.5: *Bit layout of EIFR Register*

When an IRQ signal arrives at DPin-2, the INTF0 (INT0 Flag, Fig-10.6) becomes HIGH. If INT0 bit (local interrupt flag) and I bit (global interrupt flag) are at active states, the INTF0 flag will immediately interrupt the MCU. The MCU will store LOW at I-bit (global interrupt flag is cleared) to prevent the occurrence of another interrupt until the current ISRINT0 is complete. The MCU will suspend the MLP and will jump at the ISRINT0; the INTF0 bit will be automatically cleared so that the INTF0 flag can record any incoming IRQ signal.

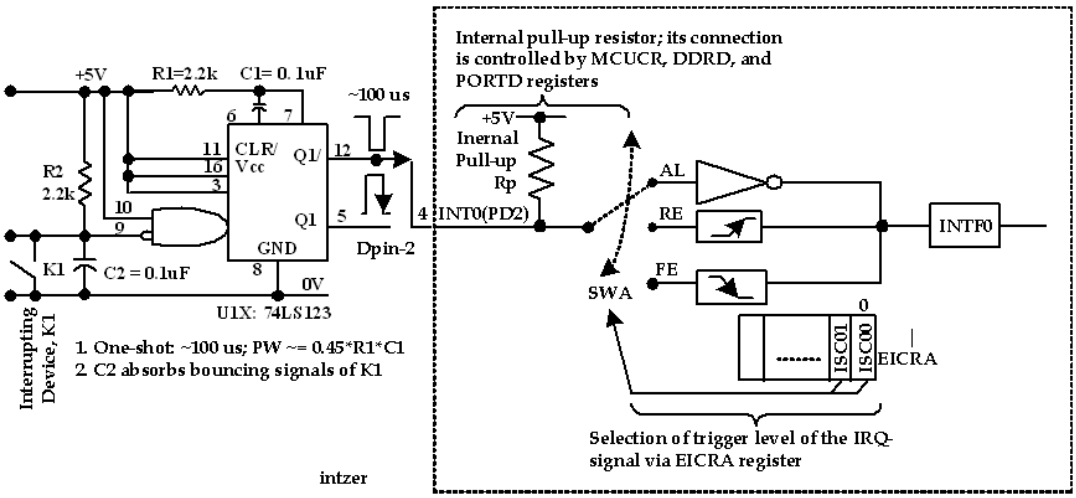

Figure-10.6: *Partial internal structure of INT0 interrupt*

10.3 Exercises

1 What happens if another interrupt fires when an ISR is in progress?

There are 26 sources of various interrupt signals in the ATmega328P architecture. Each interrupt has its own vector address. An interrupt with lower vector address has the higher priority than the one with higher vector address. Thus, INT0 with vector address 0x0002 has the higher priority than INT1 interrupt with vector address 0x0004. Interrupt with higher priority is processed first. There is no option for 'priority rotation' in the ATmega328P architecture, which we can find in the 8259 Interrupt Priority Controller of 80x86 Microprocessor.

2 Assume that ISRINT0 is in progress due to INT0 interrupt. What will happen if INT1 occurs before ISRINT0 is finished?

INT1 has the lower priority than INT0; so, ISRINT0 will not be interrupted (suspended). The MCU will remember the occurrence of INT1 interrupt. The MCU will finish ISRINT0, and it will go back to MLP. The MCU will execute only one instruction in the MLP, and then it will jump to ISRINT1.

Note: When INT0 had occurred, the MCU disabled the global interrupt flag. After arriving at the ISRINT0, we may enable it; but, the situation will not be different than what has been described above; it is due to the reason that INT1 has the lower priority than INT0.

3 Assume that ISRINT1 is in progress due to INT1 interrupt. What will happen if INT0 occurs before ISRINT1 is finished? Assume that interrupt logic has been enabled after arriving at the ISRINT1 by executing the interrupts() instruction.

INT0 has the higher priority than INT1; so, ISRINT1 will be interrupted (suspended). The MCU will suspend the current ISRINT1 and will jump to the ISRINT0. After the completion of ISRINT0, the MCU will return back to ISRINT1; it will finish the remaining codes of ISRINT1, and then the MCU will return to MLP.

Note: When INT1 had occurred, the MCU disabled the global interrupt flag. If interrupt is not enabled after arriving at the ISRINT1, the MCU will not divert to INSRINT0 routine due to INT0 interrupt though it has higher priority than INT1. The MCU will remember the occurrence of INT0 interrupt. The MCU will finish ISRINT1, and it will go back to MLP. The MCU will execute only one instruction in the MLP, and then it will jump to ISRINT0.