Tutorial 6

Interrupts for AVR Microcontroller –Part 2

# Nested External Interrupt Usage

## Programming and Analysis

In this task, we will get to know the concept of Nested Interrupt by exploring Nested *External* Interrupts.

Once the CPU enters an ISR, the global interrupt enable bit (I-bit) in SREG will be cleared so that all other interrupts are disabled during the current ISR. *This prevents a potential second interrupt from being captured and served. In some applications, we do need this to be included therefore we need Nested Interrupts.*

To use nested interrupts, the I-bit **will be set by software** so that the 2nd interrupt can be sensed and executed while the CPU is inside the first ISR. Once the 2nd ISR is completely serviced, the CPU will resume to first ISR. We will examine the effect by looking at the nested with both external interrupts INT0 and INT1.

1. Setup the hardware with:
   * Two buttons are connected to INT0 and INT1
   * Two LEDs are connected to PORTB0 and PORTB1.

Timeline

Description automatically generated

A close-up of a circuit board

Description automatically generated with low confidence

1. We will work from the Tutorial 4 Exercise 3’s code, and then modify from there



1. We need to enable the INT0

|  |
| --- |
| DDRD &= ~(1 << DDD2) ; // Clear PD2 pins for interrupt 0  EICRA |= (1 << ISC11) | (1 << ISC01); // set INT0, INT1 to trigger on falling edge  EIMSK |= (1 << INT1) | (1<<INT0); // Turns on INT0 and INT |

1. We will use 2 external LEDs hence we need to configure

|  |
| --- |
| DDRB |= (1 << DDB0) | (1<<DDB1); // Set pins for LEDs |

1. In the two ISRs
   * ISR of the INT0
     1. Turn on the Global Enable Interrupt bit
     2. Turn on LED0
     3. Delay 2 seconds
     4. Turn off LED0

|  |
| --- |
| ISR(INT0\_vect)  {  sei(); // turn on global enable interrupt bit  PORTB |= (1<<PORTB0);  *\_delay\_ms*(2000);  PORTB &= ~(1<<PORTB0);  } |

* + ISR of the INT1
    1. Turn on LED1
    2. Delay 5 seconds
    3. Turn off LED1

|  |
| --- |
| ISR(INT1\_vect)  {  PORTB |= (1<<PORTB1);  *\_delay\_ms*(5000);  PORTB &= ~(1<<PORTB1);  } |

1. The final code look like below



1. Build and download the code to the board.
2. Test the program by do the following tests

|  |  |  |
| --- | --- | --- |
|  | Test | Effect |
| 1 | Press the button connecting to INT0 | Observe if the **LED0 is on for 2 seconds and** off |
| 2 | Wait for the LED0 is off completely  Then Press the button connecting to INT1 | Observe if the LED1 is on for 5 seconds and off |
| 3 | Press the button connecting to INT0, and within **less than 2 seconds**, press the button connecting to INT1  *The goal here is to see the behaviour of nested interrupts* | * Observe if the LED0 is on first. * While LED0 is still ON, LED1 is ON, stay on for 5 seconds, then OFF. * LED0 remains ON for a short while and turn OFF. |
| 4 | Your test – Just test random and observe whether LEDs behave as you expect | |
| 5 | Optional – you can change the time for delay to 200ms and 500ms, then use the OSC or virtual bench to observe what happens | |

# Measuring execution time of ISR

This exercise, please just go through the note to get the idea. We can run the code but we wont be able to test it with the Virtual Bench. The reason I leave it here, so you can understand the significance of the execution time for ISR

The time to process the ISR is crucial to interrupt applications. Usually, ISR is recommended to be as short as possible. One of the methods to determine this is to:

* *At the start of ISR, turn on a GPIO bit*
* *At the end of ISR before it ends the last instruction, turn off the GPIO bit above*

You then can use Virtual Bench or OSC to measure the duration that the GPIO stays on. This is the execution time of the ISR under measurement.

1. We will start from the code from **Task 3 of Tutorial 4** (if one external interrupt is detected on the falling edge).



1. Add the control to PORTB0



1. Measure the duration that PORTB0 stays on using a Virtual Bench. For example, the execution time is measured at **381.39ns**

**A screenshot of a computer

Description automatically generated with medium confidence**

1. Practice, apply the same principle to measure the ISR for Task 4 – Tutorial 4
   1. Add two GPIO bits (these two have to be different from the ones under the tasks)
   2. Change the delay time to say 20ms and 50ms (so we can measure with VB)



1. Run and measure with different scenarios
   1. Press the button connecting to INT1 Interrupt, the execution time is roughly 50ms

A screenshot of a computer

Description automatically generated with medium confidence

* 1. Press the button connecting to INT0 Interrupt, the execution time is roughly 20ms

A screenshot of a computer

Description automatically generated with medium confidence

* 1. Try to press one button and then press another button shortly afterwards. For example below show the waveforms
     1. D0 (top) connect to INT0
     2. D1 (middle) connect to INT1

Graphical user interface

Description automatically generated

Closer look you can see that

* We receive 2 INT1 requests while the previous INT0 is being served

*It does not matter if you don’t get the same Waveforms, the point is we demonstrate that we can measure ISR for different interrupts with different scenarios.*

A screenshot of a computer

Description automatically generated with medium confidence