Department of Computer Engineering Faculty of Engineering, University of Peradeniya

CO321 Embedded Systems - 2022

Lab 2 - Interrupts

AVR Interrupt Programming in C

# Introduction

## Interrupts vs. Polling

A Single microcontroller can serve several peripherals. There are two methods by which devices receive service from the microcontroller: *interrupts and polling*.



In the interrupt method, whenever any peripheral needs the microcontroller’s service, the device notifies it by sending an interrupt signal. Upon receiving an interrupt signal, the microcontroller stops whatever it is doing and serves the device. The program associated with the interrupt is called the *Interrupt Service Routine (ISR).* Does not continuously monitor the status of devices/ peripherals.



In polling, the microcontroller continuously monitors the status of several devices and serve each of them as certain conditions are met. However, it is not an efficient use of the microcontroller.

**Interrupts:** Interrupts are signals sent by hardware devices to the CPU to request immediate attention. When an interrupt occurs, the CPU suspends its current activities, saves its state, and executes a handler function.

**Polling:** Polling involves the CPU continuously checking the status of a hardware device or condition at regular intervals. It's a straightforward method where the CPU dedicates processing time to repeatedly check if a device is ready or if a condition is met.

The most important reason why the interrupt method is preferable is that; the polling method wastes much of the microcontroller’s time by *polling devices that do not need service*.

## Interrupt Service Routine

For every interrupt, there must be an interrupt service routine. When an interrupt is invoked, the microcontroller runs the interrupt service routine.

## Steps in executing an ISR upon activation of an interrupt

The microcontroller goes through the following steps:

1. It finishes the instruction that is currently being executed and saves the address of the next instruction on the stack.



1. It jumps to a fixed location in memory called the interrupt vector table. The interrupt vector table directs the Microcontroller to the address of the interrupt service routine (ISR).



1. The Microcontroller starts to execute the interrupt service subroutine until it reaches the last instruction of the subroutine, which is RETI (return from interrupt).



1. Upon executing the RETI instruction, the Microcontroller returns to the place where it was interrupted. First, it gets the program counter address from the stack by popping the top bytes of the stack into the PC. Then it starts to execute from that address.



A table of a computer program

AI-generated content may be incorrect.

Figure 1: Example Interrupt Vector Table

## Sources of interrupts in the AVR

There are many sources of interrupts in the AVR, depending on which peripheral is incorporated into the chip. The following are some of the most widely used sources of interrupts in the AVR.

1. There are at least two interrupts set aside for each of the timers, one for overflow and another for comparing match.
2. Three interrupts are set aside for external hardware interrupts. Pins PORTD.2, PORTD.3 and PORTB.2 are for the external hardware interrupts INT0,INT1, andINT2, respectively.
3. Serial communication’s USART has three interrupts, one for receiving and two interrupts for transmitting.
4. The SPI interrupts.
5. The ADC (analog-to-digital converter). Normally, the service routine for an interrupt is too long to fit into the memory space allocated. For that reason, a JMP instruction is placed on the vector table to point to the address of the ISR.

**Note:** *In this lab, we will only focus on external interrupts*

## Enabling and disabling an interrupt

Upon reset, all interrupts are disabled, meaning that none will be responded to by the microcontroller unless they are activated. The 7th bit for the SREG (Status Register) register is responsible for enabling and disabling the interrupts globally.





Figure 2: Status Register

## Steps in enabling an interrupt

1. Bit 7(I) of the SREG register must be set to HIGH to allow the interrupts to happen.
2. If I=1, the corresponding interrupt enable (IE) flag bit for the required interrupt each interrupt must be enabled by setting to HIGH. This flag bit is usually located in a register belonging to the corresponding peripheral.



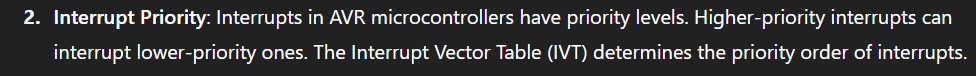
## Interrupt Priority

If two interrupts are activated at the same time, the interrupt with the higher priority is served first (typically positioned earlier in the vector table). The priority of each interrupt is related to the address of that interrupt in the interrupt vector. The interrupt that has a lower address, has a higher priority.

## Interrupt inside an interrupt

When the AVR begins to execute an ISR, it disables the I bit of the SREG register, causing all the interrupts to be disabled, and no other interrupt occurs while serving the interrupt. After serving the interrupt, the AVR enables the I bit, causing the other interrupts to be served. (i.e. the other interrupt that is happening during an execution of a interrupt has to wait until the current interrupt is being executed.)





* If the interrupt nesting is enabled in the AVR, higher priority interrupt will be executed first. (i.e. the if microcontroller is currently executing the lower priority interrupt, it will be paused and will execute the higher priority interrupt.)

## Programming External Hardware Interrupts

The ATmega has two external hardware interrupts: pins PD2 and PD3 designated as INT0, and INT1 respectively. Upon activation of these pins, the AVR is interrupted in whatever it is doing and jumps to the vector table to perform the interrupt service routine.

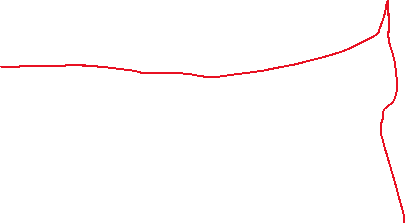
The hardware interrupts must be enabled before they can take effect. This is done using the INTx bit located in the EIMSK (External Interrupt Mask Register) register.

ISC11, ISC10, ISC01 and ISC00 bits in the EICRA (External Interrupt Control Register A) register defines which activity on the corresponding pin will generate an interrupt as shown in figure below.

A screenshot of a computer

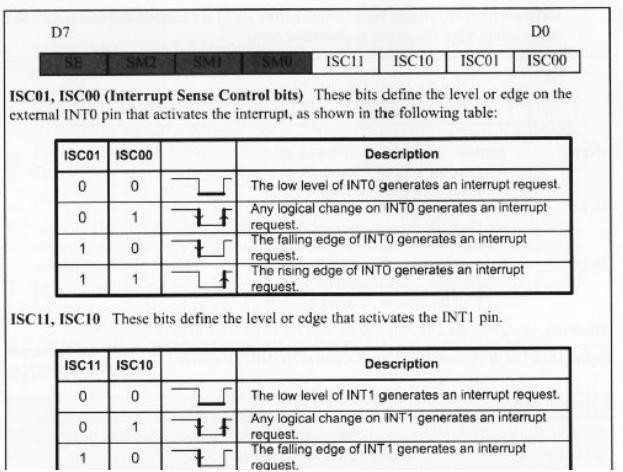
AI-generated content may be incorrect.A computer screen shot of a circuit board

AI-generated content may be incorrect.



A screenshot of a computer program

AI-generated content may be incorrect.





# Interrupt Programming in C

## Interrupt include file:

We should include the interrupt header file if we want to use interrupts in our program. Use the following statement:

#include <avr/interrupt.h>



## cli() and sei():

cli() macro clears (0) the I bit of the SREG register while sei() sets (1) it.



## Defining an ISR:

To write an ISR for an interrupt we use the following structure:

ISR (interrupt vector name) {

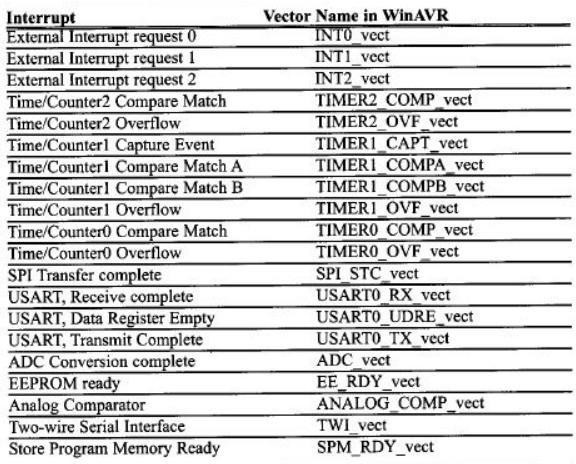
//your program



}

For the interrupt vector name we must use the ISR names in below table. For example, the following ISR serves the EEPROM ready interrupt:

ISR (EE\_RDY\_vect) {...}

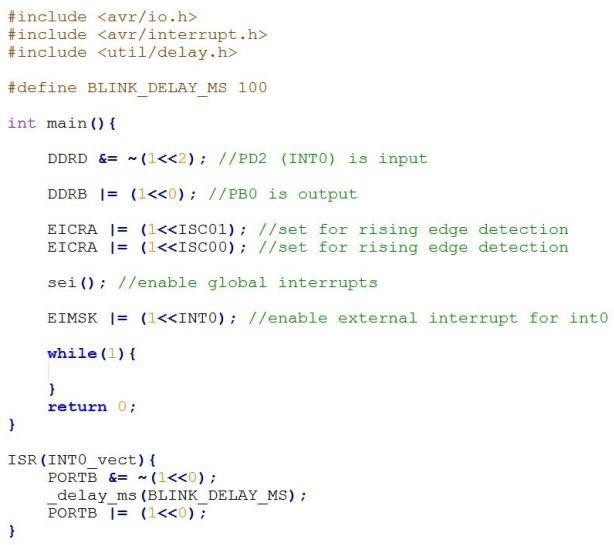


## Example:

Using INT0 generate a program to switch on a LED connected to PB6 when a button is pressed. (The LED is switched on the rising edge, i.e. when the button is pressed)



A black background with white text

AI-generated content may be incorrect.



A screenshot of a computer

AI-generated content may be incorrect.

A black background with white text

AI-generated content may be incorrect.

## Exercises:

1. PD7 pin is connected to a push button. Write a program that uses 6 LEDs connected to PORTB (6 LSBs) to display the number of times the push button is pressed, as a binary number. Implement **without using external interrupts**. (i.e. Use polling)
2. Write a program that toggles pin PB0 (show using a connected LED) whenever the push button is released, **using external interrupts**.
3. Extend the program in part 2, so that the number of times the push button was released is displayed as a binary number on a set of LEDs connected to PORTB.
4. Connect two push buttons (A and B) to two external interrupt pins. Connect 6 LEDs to a GPIO port. Write a program where a user can input a binary number using the two push buttons (press A to input a '0', press B to input a '1') through external interrupts, and display that number using the LEDs. The number must be updated and displayed every time one of the buttons are pressed