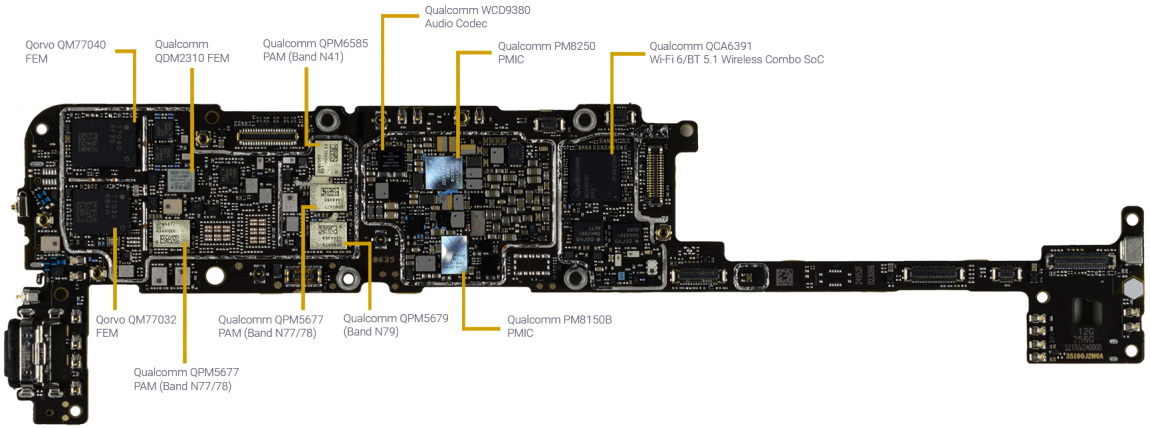


AVR Interrupts programming in C

Kizito NKURIKIYEYEZU, Ph.D.

Steps in executing an interrupt

Upon activation of an interrupt, the microcontroller goes through the following steps:

- It finishes the instruction it is currently executing and saves the address of the next instruction (program counter) on the stack.

Steps in executing an interrupt

Upon activation of an interrupt, the microcontroller goes through the following steps:

- It finishes the instruction it is currently executing and saves the address of the next instruction (program counter) on the stack.
- 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).

Steps in executing an interrupt

Upon activation of an interrupt, the microcontroller goes through the following steps:

- It finishes the instruction it is currently executing and saves the address of the next instruction (program counter) on the stack.
- 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).
- The microcontroller starts to execute the interrupt service subroutine until it reaches the last instruction of the subroutine, which is RETI (return from interrupt).

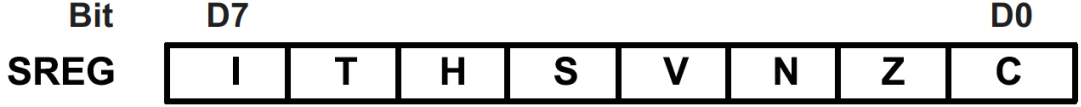
Steps in executing an interrupt

Upon activation of an interrupt, the microcontroller goes through the following steps:

- It finishes the instruction it is currently executing and saves the address of the next instruction (program counter) on the stack.
- 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).
- The microcontroller starts to execute the interrupt service subroutine until it reaches the last instruction of the subroutine, which is RETI (return from interrupt).
- Upon executing the RETI instruction, the microcontroller returns to the place where it was interrupted. First, it gets the program counter (PC) address from the stack by popping the top bytes of the stack into the PC. Then it starts to execute from that address.

Enabling and disabling an interrupt

- Upon reset, all interrupts are disabled (masked), meaning that none will be responded to by the microcontroller if they are activated.
- The interrupts must be enabled (unmasked) by software in order for the microcontroller to respond to them.
- The D7 bit of the SREG (Status Register) register is responsible for enabling and disabling the interrupts globally
- The CLI (Clear Interrupt) instruction is designed to disable (e.g., I=0) all interrupts if necessary



C – Carry flag
Z – Zero flag
N – Negative flag
V – Overflow flag
S – Sign flag
H – Half carry
T – Bit copy storage
I – Global Interrupt Enable

How to enable an interrupt

1. Step 1—Bit D7 (I) of the SREG register must be set to HIGH to allow the interrupts to happen. This is done with the SEI (Set Interrupt) instruction.
2. If I = 1, each interrupt is enabled by setting to HIGH the interrupt enable (IE) flag bit for that interrupt. This is done through the the Timer/Counter Interrupt Mask Register (TIMSK)[1] [2] [3] [4]

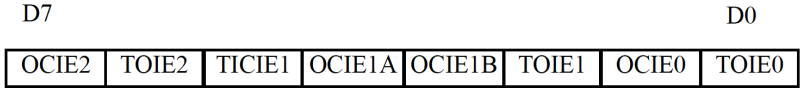


FIG 2. Timer Interrupt Mask Register (TIMSK)

[1] See summary at https://web.ics.purdue.edu/~jricha14/Timer_Stuff/TIMSK.htm

[2] These bits, along with the I bit, must be set high for an interrupt to be responded to.

[3] Upon activation of the interrupt, the I bit is cleared by the AVR itself to make sure another interrupt cannot interrupt the microcontroller while it is servicing the current one.

[4] At the end of the ISR, the RETI instruction will make I = 1 to allow another interrupt to come in.

The TIMSK register

- **Bit 0: TOIE0**—Timer/Counter0 Overflow Interrupt Enable. When the TOIE0 bit is set (one) and the I-bit in the Status Register is set (one), the Timer/Counter0 Overflow interrupt is enabled. The corresponding interrupt is executed if an overflow in Timer/Counter0 occurs, i.e. when the TOV0 bit is set in the Timer/Counter Interrupt Flag Register - TIFR.
  - 0—Disables Timer0 overflow interrupt
  - 1—Enables Timer0 overflow interrupt

The TIMSK register

- **Bit 0: TOIE0**—Timer/Counter0 Overflow Interrupt Enable. When the TOIE0 bit is set (one) and the I-bit in the Status Register is set (one), the Timer/Counter0 Overflow interrupt is enabled. The corresponding interrupt is executed if an overflow in Timer/Counter0 occurs, i.e. when the TOV0 bit is set in the Timer/Counter Interrupt Flag Register - TIFR.
  - 0—Disables Timer0 overflow interrupt
  - 1—Enables Timer0 overflow interrupt
- **Bit 1: OCIE0** —Timer0 output compare match interrupt enable
  - 0—Disables Timer0 compare match interrupt
  - 1—Enables Timer0 compare match interrupt

The TIMSK register

- **Bit 0: TOIE0**—Timer/Counter0 Overflow Interrupt Enable. When the TOIE0 bit is set (one) and the I-bit in the Status Register is set (one), the Timer/Counter0 Overflow interrupt is enabled. The corresponding interrupt is executed if an overflow in Timer/Counter0 occurs, i.e. when the TOV0 bit is set in the Timer/Counter Interrupt Flag Register - TIFR.
  - 0—Disables Timer0 overflow interrupt
  - 1—Enables Timer0 overflow interrupt
- **Bit 1: OCIE0** —Timer0 output compare match interrupt enable
  - 0—Disables Timer0 compare match interrupt
  - 1—Enables Timer0 compare match interrupt
- **Bit 2: TOIE1** —Timer1 overflow interrupt enable
  - 0—Disables Timer1 overflow interrupt
  - 1—Enables Timer1 overflow interrupt

The TIMSK register

- **Bit 3: OCIE1B** Timer1 output compare B match interrupt enable
  - 0—Disables Timer1 compare B match interrupt
  - 1—Enables Timer1 compare B match interrupt
- **Bit 4: OCIE1A** Timer1 output compare A match interrupt enable
  - 0—Disables Timer1 compare A match interrupt
  - 1—Enables Timer1 compare A match interrupt
- **Bit 5 TICIE1** Timer1 input capture interrupt enable
  - 0—Disables Timer1 input capture interrupt
  - 1—Enables Timer1 input capture interrupt
- **Bit 6 TOIE2** Timer2 overflow interrupt enable
  - 0—Disables Timer2 overflow interrupt
  - 1—Enables Timer2 overflow interrupt
- **Bit 7: OCIE2** Timer2 output compare match interrupt enable
  - 0—Disables Timer2 compare match interrupt
  - 1—Enables Timer2 compare match interrupt

Steps to program an interrupt in C

To program an interrupt, 5 steps are required:

- Include header file **avr\interrupt.h**
- Use C macro **ISR()** to define the interrupt handler and update the Interrupt Vector Table
- Enable the specific Interrupt
- Configure details of the interrupt by setting relevant registers.
- Enable the interrupt subsystem globally using **sei().**
- **Define ISR**—To write an ISR (interrupt service routine) for an interrupt we use the following structure[5]:

```
1 ISR(interrupt vector name)
2 {
3   //Here goes the code for the ISR
4 }
```

LISTING 1: Interrupt service routine structure

[5]The name of the ISR to use is found in the datasheet and the header file of the MCU

```c
#include<avr/io.h>
#include<avr/interrupt.h>
#define F_CPU (1000000UL * 16UL) //16MHz clock
ISR (TIMER1_OVF_vect){
  PORTD ^= (1 << PD0);
  TCNT1 = 63974;   // for 100ms at 16 MHz
}
int main(){
  DDRD = (1 << PD0);
  TCNT1 = 63974;   // for 100ms at 16 MHz
  TCCR1A = 0x00;
  TCCR1B = (1<<CS10)|(1<<CS12); // Timer mode with 1024 prescler
  TIMSK = (1 << TOIE1); // Enable timer1 overflow interrupt
  sei();    // Enable global interrupts
  while(1){/*Do nothing here! Everything is done via the ISR*/}
}
```

LISTING 2: Example—Blink an LED with an ISR