# Embedded Systems
Homework Week 9

Karl Ramberg

**Problem 6.1** — Modify the program in Figure 6.5 so that it polls the EIFR register. This should allow you to eliminate the interrupt handler of Figure 6.4 and the global variable led_flag.

*Answer:*

```
.global main
main:
    sbi DDRB, PB5           ; set PB5 as output pin
    sbi EIMSK, INT0         ; enable external interrupt INT0
    lds r24, EICRA
    ori r24, 1 << ISC01
    sts EICRA, r24          ; trigger INT0 on falling edge
    sei                     ; set interrupts enabled
loop0:
    sbi PORTB, PB5          ; turn on PB5 (led)
loop1:
    andi EIFR, INTF0        ; check that interrupt flag not set
    brne loop1              ; wait until INTF0 is zero
    sbi EIFR, INTF0         ; clear INTF0 by writing 1
    cbi PORTB, PB5          ; turn off PB5 (led)
    call delay1ms           ; delay 1000ms
    rjmp loop0
```

**Problem 6.2** — Explain why (when compiled without optimizations) the instruction

```
PCIFR |= (1 << PCIF2);
```

might clear all of the pin change interrupt flags and not just the flag for PCINT2_vect.

*Answer:* Because, in a stunning move of consideration for programmers everywhere, AVR decided to make 1 clear a flag register, *or* is not a safe operation here because a flag other than PCIF2 could be set and not yet handled. If you compare a set flag bit (1) with a 0 you would result in and write a 1 - which would clear the register before we had a chance to handle the interrupt. Therefore it is possible to clear all eight flags at once depending on how many unserviced interrupts are waiting to be handled.

**Problem 6.3** — In Figure 6.12, explain exactly what happens if a button press occurs between the execution of line 20 and line 22. Make sure to explain how and when EIFR changes, and when the interrupt is executed.

*Answer:* After line 20 the microcontroller is asleep and can only be woken by an interrupt. A button press after line 20 but before line 21 would result in a jump to the ISR. Interrupts are disabled in handler so this (empty here) code will execute all the way through. If a second interrupt occurs before line 21 the EIFR flag would be set. Interrupts are restored when exiting the handler and the handler would be called again to handle the second button press. Line 21 would execute after interrupts are handled and no more interrupts would be handled until sei() is called again.

**Problem 6.4** — In Figure 6.12, explain how the behavior of the program changes if line 19 is omitted from the code.

*Answer:* sei() is essential because it tells the microcontroller to service interrupts, something it doesn't do by default even though we've configured pins to work for interrupts. The program would go to sleep on the next line and never be woken up by an interrupt.

**Problem 6.5** — In the ATmega328P datasheet it says, "To avoid the MCU entering the sleep mode unless it is the programmer's purpose, it is recommended to write the Sleep Enable (SE) bit to one just before the execution of the SLEEP instruction and to clear it immediately after waking up."

In light of this, which instruction should come immediately before the sleep instruction, a command to set the SE bit or the sei instruction?

*Answer:* We would still want to execute sei() directly before sleeping.

```
1 sei();
2 sleep_enable();
3 sleep_cpu();
```

Here an interrupt could still happen during execution of 2 or between 2 and 3. The microcontroller would then go to sleep without handling the interrupt correctly.