

Microprocessors & Interfacing

Interrupts (II)

Lecturer : Annie Guo

Lecture Overview

- Interrupts in AVR
 - External interrupt
 - Internal interrupt
 - Timer

External Interrupts

- The external interrupts are triggered through the INT7:0 pins.
 - If enabled, the interrupts can be triggered even if the INT7:0 pins are configured as outputs
 - This feature provides a way of generating a software interrupt.
 - Can be triggered by a falling or rising edge or a logic level
 - Specified in External Interrupt Control Register
 - EICRA (for INT3:0)
 - EICRB (for INT7:4)

External Interrupts (cont.)

- To enable an external interrupt, two bits must be set
 - I bit in SREG
 - INTx bit in EIMSK
- To generate an external interrupt, the following must be met:
 - The interrupt must be enabled
 - The associated external pin must have a designed signal produced.

EIMSK

- External Interrupt Mask Register
 - A bit is set to enable the related interrupt

Bit	7	6	5	4	3	2	1	0	
0x1D (0x3D)	INT7	INT6	INT5	INT4	INT3	INT2	INT1	INT0	EIMSK
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

NOTE: All tables in the notes are copied from the ATmega2560 data sheet

EICRA

- External Interrupt Control Register A
 - For INT0-3
 - Defines the type of signal that activates the external interrupt
 - on rising or falling edge or level sensed

Bit	7	6	5	4	3	2	1	0
(0x69)	ISC31	ISC30	ISC21	ISC20	ISC11	ISC10	ISC01	ISC00
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

ISCn1	ISCn0	Description
0	0	The low level of INTn generates an interrupt request
0	1	Any edge of INTn generates asynchronously an interrupt request
1	0	The falling edge of INTn generates asynchronously an interrupt request
1	1	The rising edge of INTn generates asynchronously an interrupt request

EICRB*

- External Interrupt Control Register B
 - For INT4-7
 - Defines the type of signals that activate the External Interrupt
 - on rising or falling edge or level sensed.

Bit	7	6	5	4	3	2	1	0
(0x6A)	ISC71	ISC70	ISC61	ISC60	ISC51	ISC50	ISC41	ISC40
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Table 15-3. Interrupt Sense Control⁽¹⁾

ISCn1	ISCn0	Description
0	0	The low level of INTn generates an interrupt request
0	1	Any logical change on INTn generates an interrupt request
1	0	The falling edge between two samples of INTn generates an interrupt request
1	1	The rising edge between two samples of INTn generates an interrupt request

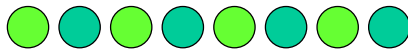
EIFR

- Interrupt flag register
 - A bit in the register is set when an edge-triggered interrupt is enabled and an event on the related INT pin happens.

Bit	7	6	5	4	3	2	1	0
0x1D (0x3D)	INT7	INT6	INT5	INT4	INT3	INT2	INT1	INT0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Example 1

- Design a system, where the state of LEDs toggles under the control of the user, and the number of toggles is counted.



Example 1 (solution)

- Use an external interrupt
 - Connect the external interrupt pin to a push button
 - When the button pressed, an interrupt is generated
- In the assembly code
 - Set up the interrupt
 - Set up the interrupt vector
 - Enable the interrupt
 - Write a service routine for this interrupt
 - Change the display pattern
 - Write the pattern to the port connected to the LEDs
 - Increase the toggle count

Code for Example 1

```
.include "m2560def.inc"

.def    temp = r16
.def    output = r17
.def    count = r18                ; count number of interrupts
.equ    PATTERN = 0b01010101

        jmp RESET                  ; set up interrupt vectors
.org    INT0addr                   ; defined in m2560def.inc
        jmp EXT_INT0

RESET:
        ser temp                   ; set Port C as output
        out DDRC, temp
        out PORTC, temp
        ldi output, PATTERN
```

Code for Example 1 (cont.)

```
ldi temp, (2 << ISC00) ; set INT0 as falling edge triggered interrupt  
sts EICRA, temp
```

```
in temp, EIMSK           ; enable INT0  
ori temp, (1 << INT0)  
out EIMSK, temp
```

```
sei                       ; enable Global Interrupt  
jmp main
```

EXT_INT0:

```
push temp                ; save register  
in temp, SREG             ; save SREG  
push temp
```

```
com output                ; flip the pattern  
out PORTC, output  
inc count
```

```
pop temp                 ; restore SREG  
out SREG, temp  
pop temp                 ; restore register  
reti
```

Code for Example 1 (cont.)

```
main:
    clr count
    clr temp
loop:
    inc temp                ; a dummy task in main

    cpi temp, 0x1F         ; the following section in red
    breq reset_temp       ; shows the need to save SREG
    rjmp loop              ; in the interrupt service routine
reset_temp:
    clr temp
    rjmp loop
```

Example 2

- Based on Example 1, implement a software interrupt
 - When there is an overflow in the counter that counts LED toggles, all LEDs are turned on.

Example 2 (solution)

- Use another external interrupt as software interrupt
 - Software generates the external interrupt request
- In the main program, test if there is an overflow
 - If there is an overflow, write a value (based on the interrupt type chosen) to the pin to invoke the interrupt.

Code for Example 2

```
.include "m2560def.inc"
.include "my_macros.inc"           ; macros for oneSecondDelay

.def    temp = r16
.def    output = r17
.def    count = r18
.equ    PATTERN = 0b01010101
.equ    OVERFLOW = 0b11111111

                                ; set up interrupt vectors
                                rjmp RESET
.org    INTOaddr
                                rjmp EXT_INT0
.org    INT1addr
                                jmp EXT_INT1

RESET:
                                ; continued
```


Code for Example 2 (cont.)

```
; continued
ser temp                                ; set Port C as output
out DDRC, temp
ldi output, PATTERN
out PORTC, temp
ldi temp, 0b00000010
out DDRD, temp                          ; set Port D bit 1 as output
out PORTD, temp

ldi temp, (2 << ISC00) | (2 << ISC10)    ; set INT0 and INT1 as
sts EICRA, temp                         ; falling edge sensed interrupts

in temp, EIMSK                          ; enable INT0 and INT1
ori temp, (1<<INT0) | (1<<INT1)
out EIMSK, temp

sei                                    ; enable Global interrupt
jmp main
; continued
```

Code for Example 2 (cont.)

; continued

EXT_INT0:

push temp **; save register**
in temp, SREG **; save SREG**
push temp

com output **; flip the pattern**
out PORTC, output
inc count

pop temp **; restore SREG**
out SREG, temp
pop temp **; restore register**
reti

; continued

Code for Example 2 (cont.)

```
; continued
EXT_INT1:
    push temp
    in temp, SREG
    push temp

    ldi output, OVERFLOW
    out PORTC, output
    oneSecondDelay           ; macro for one second delay
                                ; stored in "my_macro.inc"

    ldi output, PATTERN       ; set pattern for normal LED display
    sbi PORTD, 1              ; set bit for INT1
    pop temp
    out SREG, temp
    pop temp
    reti

                                ; continued
```

Code for Example 2 (cont.)

```
; continued

                                ; main - does nothing but increment a counter
main:
    clr count
    clr temp
loop:
    inc temp
    cpi count, 0xFF
    breq OV                    ; if overflow
    rjmp loop
OV:
    cbi PORTD, 1              ; generate an INT1 request
    clr count                 ; prepare for the next sw interrupt
    rjmp loop
```

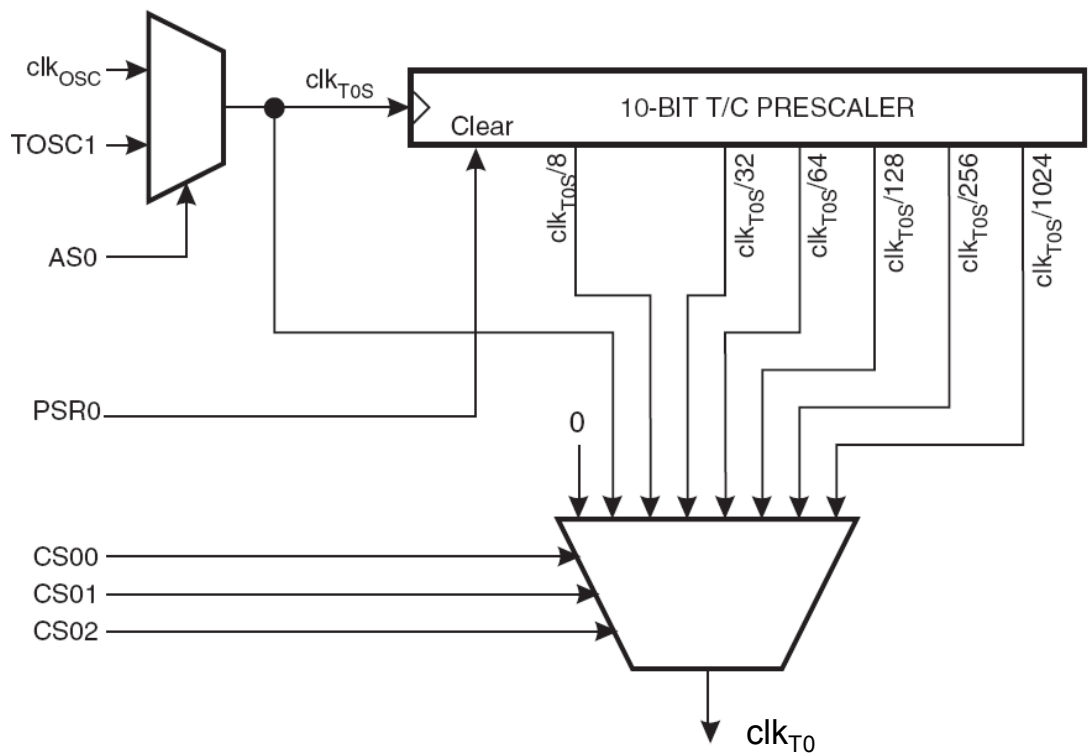
Timer

- A Timer is simply a binary counter
- Can be used to
 - Measure time duration
 - Generate PWM signals
 - Schedule real-time tasks
 - etc.

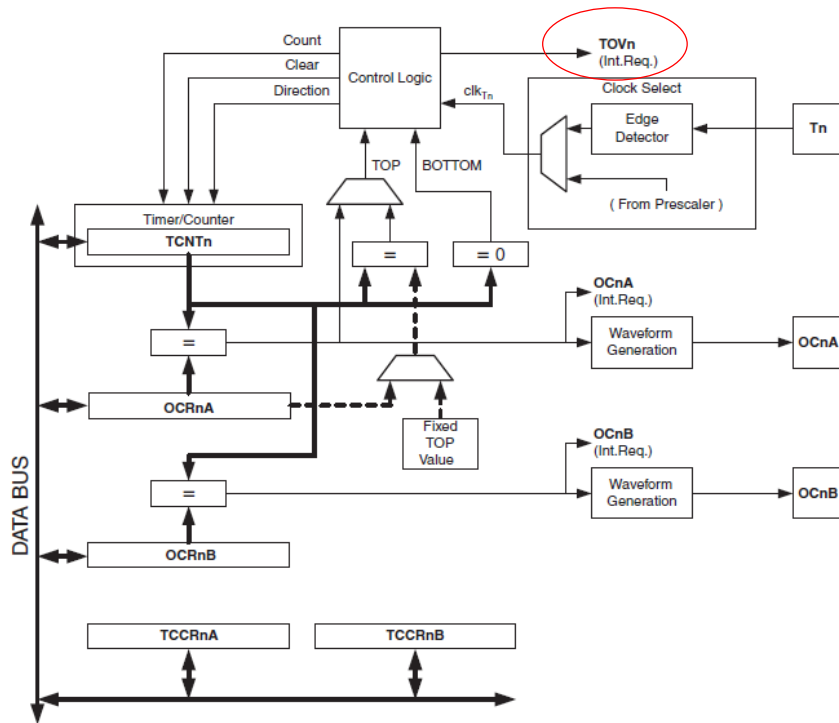
Timers in AVR

- In AVR, there are 8-bit and 16-bit timers/counters.
 - Timer 0 and Timer 2
 - 8-bit counters
 - Timer 1, 3-5
 - 16-bit counters
- Timer0 is covered in the next slides
 - Similar designs can be found for other timers
 - See the Atmega2560 data sheet

Timer0 Clock Source*



8-bit Timer Block Diagram*



8-bit Timer

- The counter can be initialized with
 - 0 (controlled by reset)
 - a number (controlled by *count signal*)
- Can count up or down
 - controlled by *direction signal*
- Those controlled signals are generated by hardware control logic
 - The control logic is further controlled by programmer by
 - Writing control bits into TCCRnA/TCCRnB
- Output
 - Overflow interrupt request bit
 - Output Compare interrupt request bit
 - OCn bit: Output Compare bit for waveform generation

TIMSK0

- Timer/Counter Interrupt Mask Register
 - Set TOIE0 (and I-bit in SREG) to enable the Overflow Interrupt
 - Set OCIE0 (and I bit in SREG) to enable Compare Match Interrupt

Bit	7	6	5	4	3	2	1	0
(0x6E)	–	–	–	–	–	OCIE0B	OCIE0A	TOIE0
Read/Write	R	R	R	R	R	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Control bits for Timer/Counter0

TIFR0*

- Timer/Counter 0 Interrupt Flag Register
 - OCF0 bit is set for a Compare Match between the counter and the data in OCR0(A/B) (Output Compare Register).
 - When $(I=1) \& \& (OCIE0(A/B)=1) \& \& (OCF0(A/B)=1)$, the related Timer/Counter Compare Match Interrupt is triggered.
 - OCF0(A/B) bit is cleared by hardware when the related interrupt is handled or can be cleared by writing a logic 0 to the flag

Bit	7	6	5	4	3	2	1	0
0x15 (0x35)	–	–	–	–	–	OCF0B	OCF0A	TOV0
Read/Write	R	R	R	R	R	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Interrupt control bits for Timer/Counters

TIFR0 (cont.)*

- Timer/Counter Interrupt Flag Register
 - TOV0 bit is set when an overflow occurs in the counter.
- When $(I=1) \& \& (TOIE0=1) \& \& (TOV0=1)$, the related Timer/Counter Overflow Interrupt is triggered.
 - In PWM mode, this bit is set when the counter changes counting direction at 0x00
- TOV0 bit is cleared by hardware when the related interrupt is handled or can be cleared by writing a logic 0 to the flag

Bit	7	6	5	4	3	2	1	0
0x15 (0x35)	–	–	–	–	–	OCF0B	OCF0A	TOV0
Read/Write	R	R	R	R	R	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Interrupt control bits for Timer/Counter0

TCCR0A/B

- Timer Counter Control Register

Bit	7	6	5	4	3	2	1	0	
0x24 (0x44)	COM0A1	COM0A0	COM0B1	COM0B0	–	–	WGM01	WGM00	TCCR0A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit	7	6	5	4	3	2	1	0	
0x25 (0x45)	FOC0A	FOC0B	–	–	WGM02	CS02	CS01	CS00	TCCR0B
Read/Write	W	W	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

TCCR0 Bit Description

- COM0xn/WGM0n/FOC0:
 - control the mode of operation
 - the behavior of the Timer/Counter and the output, is defined by the combination of the Waveform Generation mode (WGM02 WGM00) and Compare Output mode (COM0x1:0) bits.
 - The simplest mode of operation is the Normal Mode (WGM02:00 =00). In this mode the counting direction is up. The counter rolls over when it passes its maximum 8-bit value (TOP = 0xFF) and then restarts from the bottom (0x00).
- Refer to Mega2560 Data Sheet (pages 118~194) for details.

TCCR0 Bit Description (cont.)

- Bit 2:0 in TCCR0B
 - Control the clock selection

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	$\text{clk}_{\text{I/O}}$ /(No prescaling)
0	1	0	$\text{clk}_{\text{I/O}}/8$ (From prescaler)
0	1	1	$\text{clk}_{\text{I/O}}/64$ (From prescaler)
1	0	0	$\text{clk}_{\text{I/O}}/256$ (From prescaler)
1	0	1	$\text{clk}_{\text{I/O}}/1024$ (From prescaler)
1	1	0	External clock source on T0 pin. Clock on falling edge
1	1	1	External clock source on T0 pin. Clock on rising edge

T_{clk}

Bit	7	6	5	4	3	2	1	0	
0x25 (0x45)	FOC0A	FOC0B	–	–	WGM02	CS02	CS01	CS00	TCCR0B
Read/Write	W	W	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Example 3

- Implement a scheduler that can execute a task every one second.

Example 3 (solution)

- Use 8-bit Timer0 to count the time
 - Let's set Timer0 prescaler to /64 (i.e. the system frequency divided by 64)
 - The time-out for the setting should be
 - $256 * (\text{clock period}) = 256 * 64 / (16 \text{ MHz})$
 $= 1024 \text{ us}$
 - » Namely, we can set the Timer0 overflow interrupt that is to occur every 1024 us.
 - » Note, $\text{Clk}_{\text{TOS}} = 1/16 \text{ MHz}$ (obtained from the data sheet)
 - For one second, there are
 - $1000000 / 1024 \approx 1000$ interrupts
 - In code,
 - Set Timer0 interrupt to occur every 1024 microseconds
 - Use a counter to count to 1000 interrupts for counting 1 second
 - To observe the 1 second time period, use LEDs that toggles every 1000 interrupts (i.e. one second).

Example 3

```
; This program implements a timer that counts one second using
; Timer0 interrupt

.include "m2560def.inc"

.equ PATTERN=0b11110000
.def temp=r16
.def leds = r17

; The macro clears a word (2 bytes) in a memory
; the parameter @0 is the memory address for that word
.macro Clear
    ldi YL, low(@0)           ; load the memory address to Y
    ldi YH, high(@0)
    clr temp
    st Y+, temp               ; clear the two bytes at @0 in SRAM
    st Y, temp
.endmacro

; continued
```

Example 3

```
; continued
.dseg
SecondCounter:
    .byte 2                ; Two-byte counter for counting seconds.
TempCounter:
    .byte 2                ; Temporary counter. Used to determine
                           ; if one second has passed (when TempCounter=1000)

.cseg
.org 0x0000
    jmp RESET
    jmp DEFAULT            ; No handling for IRQ0.
    jmp DEFAULT            ; No handling for IRQ1.
    ...
.org OVFOaddr
    jmp Timer0OVF          ; Jump to the interrupt handler for Timer0 overflow.
    ...
    jmp DEFAULT            ; default service for all other interrupts.
DEFAULT: reti              ; no service
                           ; continued
```

Example 3

; continued

RESET:

ser temp **; set Port C as output**
out DDRC, temp

rjmp main

; continued

Example 3

; continued

```
Timer0OVF:                ; interrupt subroutine for Timer0
    in temp, SREG
    push temp              ; Prologue starts.
    push Yh                ; Save all conflict registers in the prologue.
    push YL
    push r25
    push r24               ; Prologue ends.
    ldi YL, low(TempCounter) ; Load the address of the temporary
    ldi YH, high(TempCounter) ; counter.
    ld r24, Y+              ; Load the value of the temporary counter.
    ld r25, Y
    adiw r25:r24, 1         ; Increase the temporary counter by one.
                           ; continued
```

Example 3

; continued

```
cpi r24, low(1000)           ; Check if (r25:r24)=1000
brne NotSecond
cpi r25, high(1000)          ; 1000 = 106/1024
brne NotSecond
com leds
out PORTC, leds
Clear TempCounter           ; Reset the temporary counter.

ldi ZL, low(SecondCounter)  ; Load the address of the second
ldi ZH, high(SecondCounter) ; counter.
ld r24, Z+                  ; Load the value of the second counter.
ld r25, Z
adiw r25:r24, 1              ; Increase the second counter by one.

; continued
```

Example 3

; continued

st Z, r25 ; Store the value of the second counter.

st -Z, r24

rjmp EndIF

NotSecond:

st Y, r25 ; Store the value of the temporary counter.

st -Y, r24

EndIF:

pop r24 ; Epilogue starts;

pop r25 ; Restore all conflict registers from the stack.

pop YL

pop YH

pop temp

out SREG, temp

reti ; Return from the interrupt.

; continued

Example 3

; continued

main:

```
ldi leds, 0xff          ; Init pattern displayed
out PORTC, leds
ldi leds, PATTERN
Clear TempCounter       ; Initialize the temporary counter to 0
Clear SecondCounter     ; Initialize the second counter to 0
ldi temp, 0b00000000
out TCCR0A, temp
ldi temp, 0b00000011
out TCCR0B, temp        ; Prescaling value=64
ldi temp, 1<<TOIE0      ; =1024 microseconds
sts TIMSK0, temp        ; T/C0 interrupt enable
sei                     ; Enable global interrupt
loop: rjmp loop         ; loop forever
```


Reading Material

- Chapter 10: Interrupts and Real-Time Events. Microcontrollers and Microcomputers by Fredrick M. Cady.
- Mega2560 Data Sheet.
 - External Interrupts.
 - Timer0

Homework

1. An underground oil tank monitor system has the following functions:
 1. `read()`: to read the tank oil level
 2. `display()`: to display the oil level
 3. `main()`: process a few of basic tasks: if the oil level is below the low limit, do something; if oil level is over the high limit, do something else; and other routine work.

It is required that the display should be updated every 1 minute, reading should be done at least every 10 seconds. Assume `read()` and `display()` take 1 ms and 5 ms, respectively. Design a timing schedule for those functions so that the above requirements can be met and the design leads to an easy assembly code implementation.