Interrupts (II)

Lecturer: Sri Parameswaran
Notes by: Annie Guo
External Interrupts

- The external interrupts are triggered by the INT7:0 pins.
  - If enabled, the interrupts will trigger even if the INT7:0 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 interrupt, two bits must be set:
  - I bit in SREG
  - INTx bit in EIMSK

- To activate an interrupt, the following must be met:
  - The interrupt must be enabled
  - The associated external pin must have a designed signal asserted.
## EIMSK

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

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>INT7</td>
<td>INT6</td>
<td>INT5</td>
<td>INT4</td>
<td>INT3</td>
<td>INT2</td>
<td>INT1</td>
<td>INT0</td>
</tr>
</tbody>
</table>
EICRA

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

<table>
<thead>
<tr>
<th>ISCn1</th>
<th>ISCn0</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>The low level of INTn generates an interrupt request.</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>The falling edge of INTn generates asynchronously an interrupt request.</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>The rising edge of INTn generates asynchronously an interrupt request.</td>
</tr>
</tbody>
</table>
EICRB

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

<table>
<thead>
<tr>
<th>ISCn1</th>
<th>ISCn0</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>The low level of INTn generates an interrupt request.</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Any logical change on INTn generates an interrupt request.</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>The falling edge between two samples of INTn generates an interrupt request.</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>The rising edge between two samples of INTn generates an interrupt request.</td>
</tr>
</tbody>
</table>
EIFR

- Interrupt flag register
  - A bit is set when an event-triggered interrupt is enabled and the related event on the related INT pin happens.
    - Event-triggered interrupt: signal edge activated.
Example 1

- Design a system, where the state of LEDs toggles under the control of the user.
Example 1 (solution)

- Use an external interrupt
  - Connect the external interrupt pin to a push button
  - When the button pressed, the 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
Code for Example 1

.include "m2560def.inc"

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

; set up interrupt vectors

jmp RESET

.org INT0addr
jmp EXT_INT0

RESET:

ldi temp, low(RAMEND) ; initialize stack
out SPL, temp
ldi temp, high(RAMEND)
out SPH, temp

ser temp ; set Port C as output
out DDRC, temp
out PORTC, temp
ldi output, PATTERN ; continued
Code for Example 1

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

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
; continued

; main - does nothing but increment a counter
main:
    clr count
    clr temp
loop:
    inc temp        ; a dummy task in main
    rjmp loop
Timer/Counters

- Simply binary counters
- Used in two different modes:
  - Timer
    - Counting time periods
  - Counter
    - Counting the events or pulse or something of this nature
- Can be used to
  - Measure time periods, speed, frequency
  - Generate PWM signals
  - Schedule real-time tasks
  - etc.
Timer/Counters in AVR

- In AVR, there are 8-bit and 16-bit timer/counters.
  - Timer 0 and Timer 2: 8-bit
  - Timer 1,3-5 16-bit
8-bit Timer/Counter Block Diagram
8-bit Timer/Counter

● 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(A/B) (and I bit in SREG) to enable Compare Match Interrupt

Control bits for timer/counter0
**TIFR0**

- **Timer/Counter Interrupt Flag Register**
  - OCF0(A/B) bit is set when a Compare Match between the counter and the data in OCR0(A/B) (Output Compare Registers).
    - When (I=1)&&(OCIE0(A/B)=1)&&(OCF0(A/B)=1), the related Timer/Counter Compare Match Interrupt is executed.
  - 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.

**Interrupt control bits for timer/counter0**
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 executed.
    - In PWM mode, this bit is set when the counter changes counting direction at 0x00
  - 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

Interrupt control bits for timer/counter0
## TCCR0A/B

- **Timer Counter Control Register**
  - For Timer/Counter0
  - Similar registers for other timers

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x24 (0x44)</td>
<td>COM0A1</td>
<td>COM0A0</td>
<td>COM0B1</td>
<td>COM0B0</td>
<td>–</td>
<td>–</td>
<td>WGM01</td>
<td>WGM00</td>
</tr>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R</td>
<td>R</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x25 (0x45)</td>
<td>FOC0A</td>
<td>FOC0B</td>
<td>–</td>
<td>–</td>
<td>WGM02</td>
<td>CS02</td>
<td>CS01</td>
<td>CS00</td>
</tr>
<tr>
<td>Read/Write</td>
<td>W</td>
<td>W</td>
<td>R</td>
<td>R</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
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 (WGM02:00) 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 always 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.
**TCCRO Bit Description (cont.)**

- Bit 2:0 in TCCRB

### Table 16-9. Clock Select Bit Description

<table>
<thead>
<tr>
<th>CS02</th>
<th>CS01</th>
<th>CS00</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>No clock source (Timer/Counter stopped)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>(\text{clk}_{\text{i/O}}\text{/}(\text{No prescaling}))</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>(\text{clk}_{\text{i/O}}\text{/8} \text{(From prescaler)})</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>(\text{clk}_{\text{i/O}}\text{/64} \text{(From prescaler)})</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>(\text{clk}_{\text{i/O}}\text{/256} \text{(From prescaler)})</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>(\text{clk}_{\text{i/O}}\text{/1024} \text{(From prescaler)})</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>External clock source on T0 pin. Clock on falling edge</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>External clock source on T0 pin. Clock on rising edge</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x25 (0x45)</td>
<td>FOC0A</td>
<td>FOC0B</td>
<td>–</td>
<td>–</td>
<td>WGM02</td>
<td>CS02</td>
<td>CS01</td>
<td>CS00</td>
</tr>
<tr>
<td>Read/Write</td>
<td>W</td>
<td>W</td>
<td>R</td>
<td>R</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Example 2

- Implement a scheduler that can execute a task every one second.
Example 2 (solution)

- Use Timer0 to count the time
  - Let’s set Timer0 prescaler to 8
    - The time-out for the setting should be
      - $256 \times (\text{clock period}) = 256 \times 8 / (16 \text{ MHz})$
      - $= 128 \text{ us}$
      - Namely, we can set the Timer0 overflow interrupt that is to occur every 128 us.
      - Note, $\text{Clk}_{\text{tos}} = 1/16 \text{ MHz}$ (obtained from the data sheet)
  - For one second, there are
    - $1000000 / 128 = \sim 7812$ interrupts

- In code,
  - Set Timer0 interrupt to occur every 128 microseconds
  - Use a counter to count to 7812 interrupts for counting 1 second
  - To observe the 1 second time period, toggle an LED every 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
.cseg
.org 0x0000
    jmp RESET
    jmp DEFAULT
    ; No handling for IRQ0.
    jmp DEFAULT
    ; No handling for IRQ1.
.org OVF0addr
    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:  ldi temp, high(RAMEND) ; Initialize stack pointer
    out SPH, temp
    ldi temp, low(RAMEND)
    out SPL, temp
    ser temp ; set Port C as output
    out DDRC, temp
    rjmp main

; continued
Example 3

; continued

Timer0OVF: ; interrupt subroutine to 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.
    ; Load the value of the temporary counter.
    lds r24, TempCounter
    lds r25, TempCounter+1
    adiw r25:r24, 1 ; Increase the temporary counter by one.
    ; continued
Example 3

cpi r24, low(7812) ; Check if (r25:r24) = 7812
ldi temp, high(7812) ; 7812 = 10^6/128
cpc r25, temp
brne NotSecond
com leds
out PORTC, leds
clear TempCounter ; Reset the temporary counter.
                    ; Load the value of the second counter.
lds r24, SecondCounter
lds r25, SecondCounter+1
adiw r25:r24, 1 ; Increase the second counter by one.
                    ; continued
Example 3

```
sts SecondCounter, r24
sts SecondCounter+1, r25
rjmp EndIF

NotSecond:
   ; Store the new value of the temporary counter.
   sts TempCounter, r24
   sts TempCounter+1, r25

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

```assembly
main:
  ldi leds, 0xFF
  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, 0b00000010
  out TCCR0B, temp     ; Prescaling value=8
  ldi temp, 1<<TOIE0   ; = 128 microseconds
  sts TIMSK0, temp     ; T/C0 interrupt enable
  sei                  ; Enable global interrupt

loop:  rjmp loop       ; loop forever
```
Reading Material

- Chapter 8: Interrupts and Real-Time Events. Microcontrollers and Microcomputers by Fredrick M. Cady.
- Mega2560 Data Sheet.
  - External Interrupts.
  - Timer0
Homework

1. What do you need to do to set up an Timer0 Output Compare Match Interrupt?
Homework

2. Based on the Example 1 in this week lecture slides, implement a software interrupt such that when there is an overflow in the counter that counts the number of LED toggles, all LEDs are turned on.