Lecture 15 CS3514 31/10/17

Setting the Sensing mode : rising, falling, changing, Level

EICRA |= (1 << ISC01);

//EICRA : external Interrupt Control Register A

13.2.1 EICRA – External Interrupt Control Register A

The External Interrupt Control Register A contains control bits for interrupt sense control.

Bit	7	6	5	4	3	2	1	0	
(0x69)	–	–	–	–	ISC11	ISC10	ISC01	ISC00	EICRA
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7:4 – Reserved**

These bits are unused bits in the ATmega48A/PA/88A/PA/168A/PA/328/P, and will always read as zero.

- **Bit 3, 2 – ISC11, ISC10: Interrupt Sense Control 1 Bit 1 and Bit 0**

The External Interrupt 1 is activated by the external pin INT1 if the SREG I-flag and the corresponding interrupt mask are set. The level and edges on the external INT1 pin that activate the interrupt are defined in Table 13-1. The value on the INT1 pin is sampled before detecting edges. If edge or toggle interrupt is selected, pulses that last longer than one clock period will generate an interrupt. Shorter pulses are not guaranteed to generate an interrupt. If low level interrupt is selected, the low level must be held until the completion of the currently executing instruction to generate an interrupt.

Table 13-2. Interrupt 0 Sense Control

ISC01	ISC00	Description
0	0	The low level of INT0 generates an interrupt request.
0	1	Any logical change on INT0 generates an interrupt request.
1	0	The falling edge of INT0 generates an interrupt request.
1	1	The rising edge of INT0 generates an interrupt request.

Writing the ISR.

Example :

```
ISR ( EXT_INT0_VECT ) {
    digitalWrite (13, !digitalRead (10));
}
```

In general,

```
ISR ( {vector}_VECT ) { .... }
```

ISR is a predefined macro in interrupt.h

```
#ifdef __cplusplus
#  define ISR(vector, ...)            \
    extern "C" void vector (void) __attribute__ ((signal,__INTR_ATTRS)) __VA_ARGS__; \
    void vector (void)
#else
#  define ISR(vector, ...)            \
    void vector (void) __attribute__ ((signal,__INTR_ATTRS)) __VA_ARGS__; \
    void vector (void)
#endif
```

Arduino Interrupt functionality

Simplified way of using Interrupts

External Int. 0 (pin 2)

```
Void setup {
    attachInterrupt ( 0 , foo , FALLING);
}
```

function to call

condition on pin 2

```
Void foo () { // Called when pin 2 input falls
    :
}
```

NB: Any variable used in an ISR (foo() in this case) **must** be declared as **volatile**, since they can change in ways unknown to the compiler.

Eg: volatile int my_int;

The volatile keyword tells the compiler that the variable can be modified in ways unknown to the compiler.

This usually applies to variables that are mapped to a fixed particular memory address (e.g. device register).

Statements containing these variables should not be reordered, for optimization reasons, by the compiler

For instance, suppose KEYBOARD is a device register that accepts characters from the keyboard

This is the desired behaviour →

```
void get_two_kbd_chars(){
  extern char KEYBOARD;
  char c0, c1;
  c0 = KEYBOARD;
  c1 = KEYBOARD;
}
```

```
void get_two_kbd_chars(){
  extern char KEYBOARD;
  char c0, c1;
  register char tmp;
  tmp = KEYBOARD;
  c0 = tmp;
  c1 = tmp
}
```

The compiler could produce this code ←

extern volatile char KEYBOARD;
will prevent the compiler from making the optimization

Timers & Counters

Counter registers can count from $0 \rightarrow \text{maxvalue} \rightarrow 0$

Max value depends on the size of Register n bits $= 2^n - 1$

On overflow, an overflow flag is set

- This can be checked manually or can be used to generate an Interrupt.

To use a counter as a timer it needs to be connected to a clock source

Smallest measurable unit of time = 1 clock period

If t = clock period (s), f = clock frequency (Hz)

$$t = \frac{1}{f}$$

for a 1 MHz clock (for example)

$$t = 1/f = \frac{1}{10^6} = 10^{-6} \text{ sec} = 1 \mu\text{sec}$$

Note: 1) it is possible to clock the AVR timer via one external pin

2) Max resolution on an 8-bit AVR is 16 MHz

3) Frequency can vary with supply voltage.

Timer Types

Timer 0	Timer 1	Time 2	Watchdog
8-bit	16-bit	8-bit	–
0 → 255	0 → 65,535	0 → 255	–
delay()	Arduino Servo Lib	tone()	–
millis()			

The Control registers for the Timers include TCCRxA & TCCRxB

16.11.1 TCCR1A – Timer/Counter1 Control Register A

Bit	7	6	5	4	3	2	1	0	
(0x80)	COM1A1	COM1A0	COM1B1	COM1B0	–	–	WGM11	WGM10	TCCR1A
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	

We'll only consider the default values for TCCRxA. The other values are closely linked with waveform generation.

For our purposes, the most Important settings are the last 3 bits in the TCCRxB : CS12, CS11, CS10

16.11.2 TCCR1B – Timer/Counter1 Control Register B

Bit	7	6	5	4	3	2	1	0	
(0x81)	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 16-5. Clock Select Bit Description

CS12	CS11	CS10	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}/1$ (No prescaling)
0	1	0	$clk_{I/O}/8$ (From prescaler)
0	1	1	$clk_{I/O}/64$ (From prescaler)
1	0	0	$clk_{I/O}/256$ (From prescaler)
1	0	1	$clk_{I/O}/1024$ (From prescaler)
1	1	0	External clock source on T1 pin. Clock on falling edge.
1	1	1	External clock source on T1 pin. Clock on rising edge.

If external pin modes are used for the Timer/Counter1, transitions on the T1 pin will clock the counter even if the pin is configured as an output. This feature allows software control of the counting.

By default, all bits = $\emptyset$

Example 1 : Set Timer1 to run at clock speed, with on Count per clock cycle. On overflow, run an ISR that toggles a led on pin 2.

```
#define LedPin 2

Void setup () {
  pinMode (Ledpin, output);
  cli(); // disable global Interrupts
  TCCR1A = ø; // Clears all bits in TCCR1A register
  TCCR1B = ø; //  "   "   "   " TCCR1B  "
  TIMSK1 |= (1 << TOIE1); // enable Timer1 overflow Interrupt
  TCCR1B |= (1 << CS10); // run @ clock speed
  sei(); // enable global interrupts
}
```

TIMSK1 is the timer/Counter 1 Interrupt mask register

16.11.8 TIMSK1 – Timer/Counter1 Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
(0x6F)	–	–	ICIE1	–	–	OCIE1B	OCIE1A	TOIE1	TIMSK1
Read/Write	R	R	R/W	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 5 – ICIE1: Timer/Counter1, Input Capture Interrupt Enable**

When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Counter1 Input Capture interrupt is enabled. The corresponding Interrupt Vector (see "Interrupts" on page 57) is executed when the ICF1 Flag, located in TIFR1, is set.

- **Bit 2 – OCIE1B: Timer/Counter1, Output Compare B Match Interrupt Enable**

When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Counter1 Output Compare B Match interrupt is enabled. The corresponding Interrupt Vector (see "Interrupts" on page 57) is executed when the OCF1B Flag, located in TIFR1, is set.

- **Bit 1 – OCIE1A: Timer/Counter1, Output Compare A Match Interrupt Enable**

When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Counter1 Output Compare A Match interrupt is enabled. The corresponding Interrupt Vector (see "Interrupts" on page 57) is executed when the OCF1A Flag, located in TIFR1, is set.

- **Bit 0 – TOIE1: Timer/Counter1, Overflow Interrupt Enable**

16.11.9 TIFR1 – Timer/Counter1 Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0	
0x16 (0x36)	–	–	ICF1	–	–	OCF1B	OCF1A	TOV1	TIFR1
Read/Write	R	R	R/W	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 0 – TOV1: Timer/Counter1, Overflow Flag**

The setting of this flag is dependent of the WGM13:0 bits setting. In Normal and CTC modes, the TOV1 Flag is set when the timer overflows. Refer to Table 16-4 on page 132 for the TOV1 Flag behavior when using another WGM13:0 bit setting.

TOV1 is automatically cleared when the Timer/Counter1 Overflow Interrupt Vector is executed. Alternatively, TOV1 can be cleared by writing a logic one to its bit location.

The timer starts running as soon as CS10 is set.
Since we enabled an overflow Interrupt ISR(TIMER1_OVF_vect) will be called when the timer overflows.

```
ISR(Timer1_OVF_vect) {
        digital(LedPin, !digitalRead(Led pin));
}
```

To turn the timer off, we set TCCR1B = 0

How fast will our LED blink?

Clock Speed = 16 MHz (Atmega 328)

Timer1 = 16 bits (value 0 → 65,535)

1 clock cycle every $1/16\times10^6$ seconds $= 6.25 \times 10^{-8}$ sec

$\therefore$ 65,535 timer counts will take $6.25\times10^{-8} \times 65{,}535$
$= 0.0041$ seconds