

The background is a close-up, slightly blurred photograph of a green printed circuit board (PCB). A large, square, brown microcontroller chip is the central focus, with its gold pins visible. Other components like smaller chips, capacitors, and a circular component are also visible on the board.

**MCT 4334**

# **Embedded System Design**

**Week 07 Interrupts**

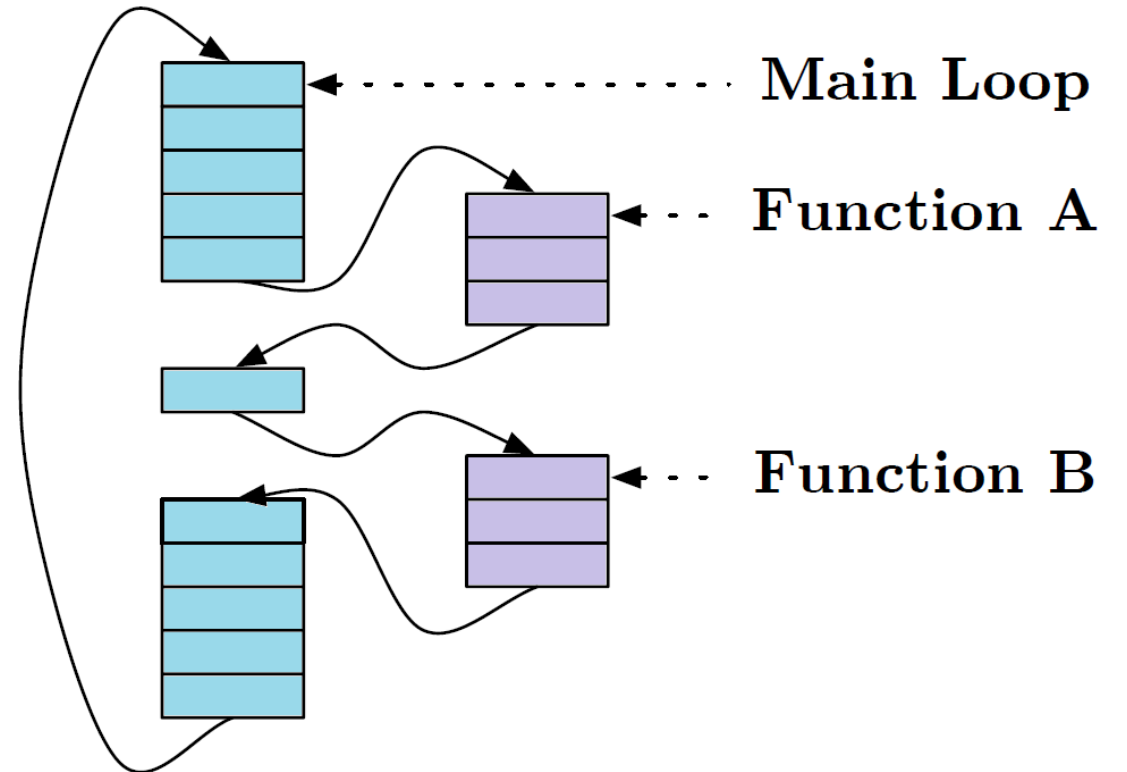
# Outline

- Polling vs interrupt
- Interrupt vector table
- Interrupt-related registers
- Programming examples

# Polling

- This program **continuously** monitors PD0 and PD1.  
When PD2 is LOW, Function A gets called  
When PD3 is LOW, Function B gets called
- Checking for input states and all the subsequent processing are done inside the main loop.

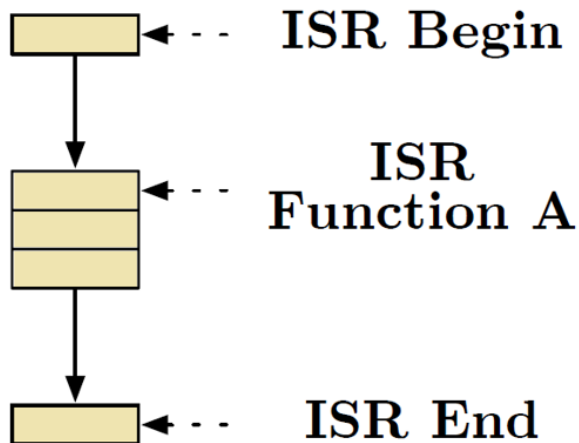
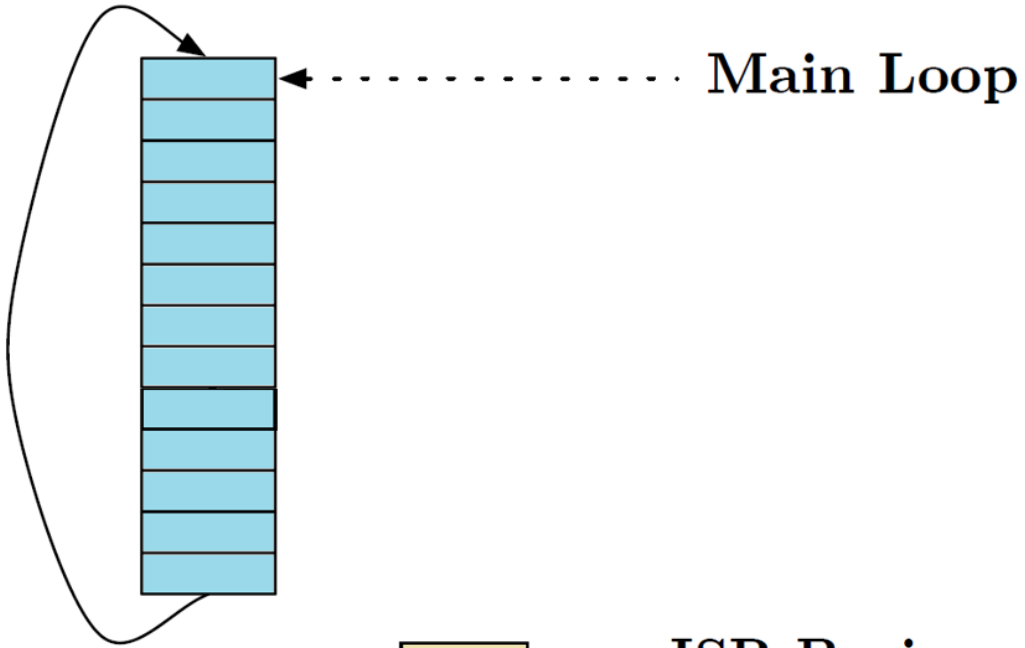
```
int main()  
{  
    while(1)  
    {  
        if (!digitalRead(2))  
        {  
            Function_A();  
        }  
        if (!digitalRead(3))  
        {  
            Function_B();  
        }  
    }  
}
```



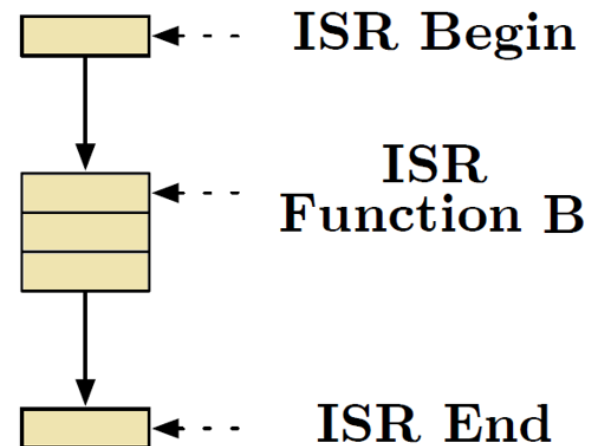
# Limitations of Polling

- Performance can suffer in complicated embedded systems which require input from many sources.
- If the main loop has many things to perform, short blips of input can be missed.
- Multi-tasking is difficult
  - e.g. driving 3 servo motors and 3 normal DC motors in response to 10 digital inputs and 5 analog inputs.

# Interrupt-based approach



- Main loop is free of polling
- The CPU can sleep (to conserve power)
- Whenever PD2 or PD3 is low, an interrupt signal gets generated.
- The CPU stops whatever it is doing and jumps to an associated Interrupt Service Routine (ISR) and the respective function gets executed.
- At the end of the ISR, CPU resumes its previous work



# Drawbacks of Interrupt-based Approach

- Added complexity to make the CPU stop its work, save the context, jump to the ISR and then restore the context and resume its previous work.
- CPU can be interrupted **any** time. Software needs to be written with this behavior in mind.

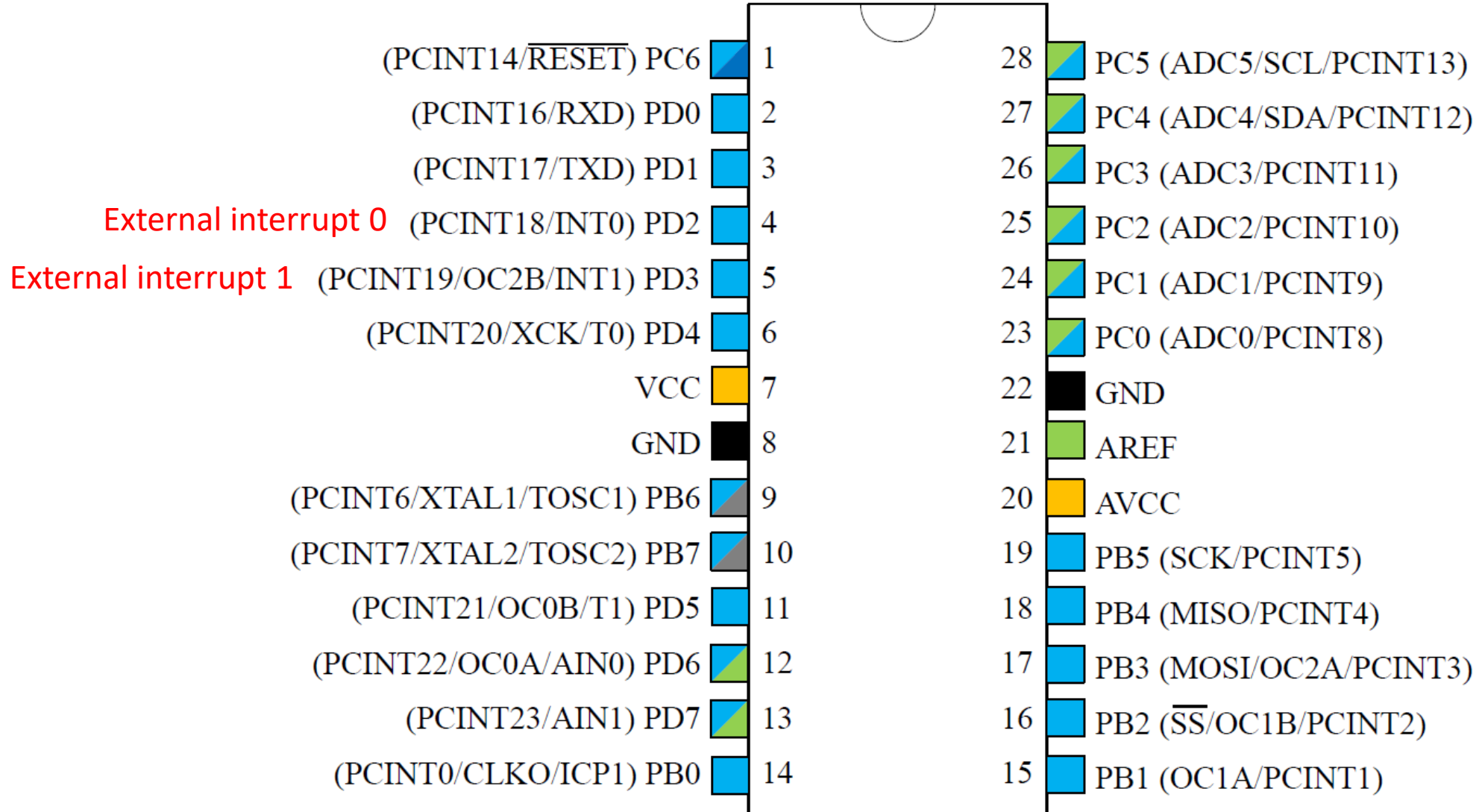
# Context Management

- **Context** can be defined as the state of the CPU including the contents of the registers (most importantly the program counter, which points to the current instruction).
- It is very important to save the context of the CPU and restore it upon completion of the ISR.
- Context management is quite tedious to perform manually.

# Interrupts on ATmega328p

- RESET interrupt
- 2 external interrupts (**INT0 and INT1**)
- 3 pin change interrupts that map to 23 GPIO pins (**PCINT0, PCINT1 and PCINT2**)
- Timer overflow interrupt (when timers reset to 0 after reaching the max value)
- Timer OCRxA match interrupt (when the value of the timer is OCRxA)
- Timer OCRxB match interrupt (when the value of the timer is OCRxB)
- SPI interrupts
- USART interrupts
- ADC interrupt
- EEPROM interrupt
- Analog comparator interrupt
- I2C interrupt
- Store Program Memory (SPM) interrupt





# Interrupt Vector Table

ATmega328p has an interrupt vector table that holds the list of interrupts along with the address of the respective interrupt handlers.

# ATmega328p

Pri.	Address	Interrupt Source	ISR C Function Name	Description
1	0x0000	RESET		System reset (power-on)
2	0x0002	INT0	INT0_vect	External Interrupt Request 0
3	0x0004	INT1	INT1_vect	External Interrupt Request 1
4	0x0006	PCINT0	PCINT0_vect	Pin Change Interrupt Request 0
5	0x0008	PCINT1	PCINT1_vect	Pin Change Interrupt Request 1
6	0x000A	PCINT2	PCINT2_vect	Pin Change Interrupt Request 2
7	0x000C	WDT	WDT_vect	Watchdog Time-out Interrupt
8	0x000E	TIMER2 COMPA	TIMER2_COMPA_vect	Timer/Counter2 Compare Match A
9	0x0010	TIMER2 COMPB	TIMER2_COMPB_vect	Timer/Counter2 Compare Match B
10	0x0012	TIMER2 OVF	TIMER2_OVF_vect	Timer/Counter2 Overflow
11	0x0014	TIMER1 CAPT	TIMER1_CAPT_vect	Timer/Counter1 Capture Event
12	0x0016	TIMER1 COMPA	TIMER1_COMPA_vect	Timer/Counter1 Compare Match A
13	0x0018	TIMER1 COMPB	TIMER1_COMPB_vect	Timer/Counter1 Compare Match B

Pri.	Address	Interrupt Source	ISR C Function Name	Description
14	0x001A	TIMER1 OVF	TIMER1_OVF_vect	Timer/Counter1 Overflow
15	0x001C	TIMER0 COMPA	TIMER0_COMPA_vect	Timer/Counter0 Compare Match A
16	0x001E	TIMER0 COMPB	TIMER0_COMPB_vect	Timer/Counter0 Compare Match B
17	0x0020	TIMER0 OVF	TIMER0_OVF_vect	Timer/Counter0 Overflow
18	0x0022	SPI, STC	SPI_STC_vect	SPI Serial Transfer Complete
19	0x0024	USART, RX	USART_RX_vect	USART Receive Complete
20	0x0026	USART, UDRE	USART_UDRE_vect	USART Data Register Empty
21	0x0028	USART, TX	USART_TX_vect	USART Transmit Complete
22	0x002A	ADC	ADC_vect	ADC Conversion Complete
23	0x002C	EE READY	EE_READY_vect	EEPROM Ready
24	0x002E	ANALOG COMP	ANALOG_COMP_vect	Analog Comparator
25	0x0030	TWI	TWI_vect	2-wire Serial Interface
26	0x0032	SPM READY	SPM_READY_vect	Store Program Memory Ready

- AVR-GCC compiler predefines functions in the interrupt vector table.
- When an algorithm needs to handle an interrupt, all that is required is to write the desired function using the appropriate function name.
- Context management is automatically handled.

### Examples

```
ISR(INT0_vect)  
{
```

```
    //Code here  
}
```

```
ISR(INT1_vect)  
{
```

```
    //Code here  
}
```

```
ISR(ADC_vect)  
{
```

```
    //Code here  
}
```

```
ISR(PCINT0_vect)  
{
```

```
    //Code here  
}
```

# Optional Parameters in ISR

- **ISR\_BLOCK** (default behavior if the parameter is unspecified). When one ISR is being executed, another interrupt will not occur. Global interrupts automatically get disabled before it begins and re-enabled after it ends.
- **ISR\_NOBLOCK** (allows nested ISRs).
- **ISR\_NAKED** (Manual context management).

## Examples

```
ISR(INT0_vect, ISR_BLOCK)
{
    //
}
```

```
ISR(ADC_vect)
{
    //Equivalent to ISR_BLOCK
}
```

```
ISR(PCINT0_NAKED)
{
    //
}
```

# SREG

Status Register (x5F)

Bit	7	6	5	4	3	2	1	0
0x5F	I	T	H	S	V	N	Z	C
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Default	0	0	0	0	0	0	0	0



- The MSB of the SREG enables or disable interrupts in general.
- It is also called global interrupt enable/disable.
- The first step to work with any interrupt is SET this bit first and then enable specific interrupt in the respective register. (*General and then specific*)

# **External Interrupts**



- There are two external interrupts (INT0 and INT1) on PD2 and PD3 respectively.
- Four modes are supported
  - Low level
  - Any transition
  - High-to-low transition
  - Low -to-high transition

# EIMSK

External Interrupt Mask Register (x3D)

Bit	7	6	5	4	3	2	1	0
0x3D	-	-	-	-	-	-	INT1	INT0
Read/Write	R	R	R	R	R	R	R/W	R/W
Default	0	0	0	0	0	0	0	0

Set INT0 to turn on external interrupt 0

Set INT1 to turn on external interrupt 1

# EICRA

## External Interrupt Control Register (x69)

Bit	7	6	5	4	3	2	1	0
0x69	-	-	-	-	ISC11	ISC10	ISC01	ISC00
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W
Default	0	0	0	0	0	0	0	0

ISCx (Interrupt Sense Control) set the mode for the respective external interrupt

ISCx	Description
00	LOW level
01	Change in logic level
10	HIGH to LOW transition
11	Low to HIGH transition

# EIFR

External Interrupt Flag Register (x3C)

Bit	7	6	5	4	3	2	1	0
0x3C	-	-	-	-	-	-	INTF1	INTF0
Read/Write	R	R	R	R	R	R	R/W	R/W
Default	0	0	0	0	0	0	0	0

A flag indicates that a specific interrupt has occurred.

For example:

As soon as INT0 gets triggered, INF0 becomes HIGH.

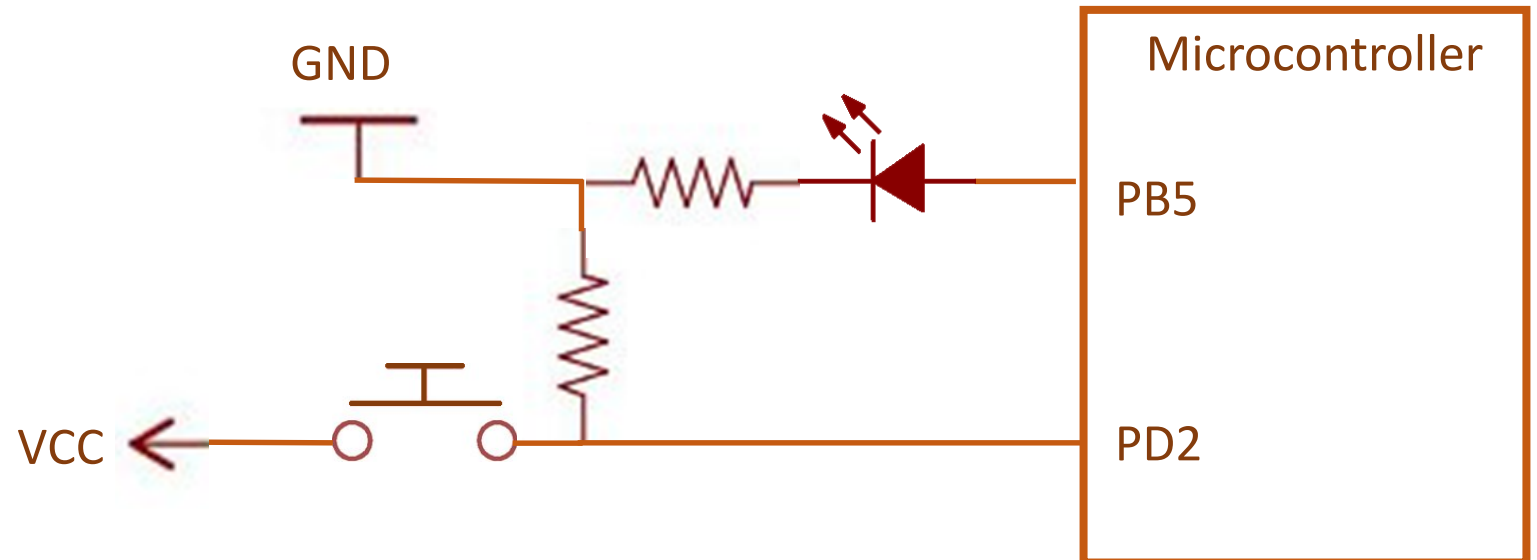
After the end of the ISR routine, INF0 becomes LOW again.

# Example – external interrupt

A tactile button is connected to PD2 of ATmega328p via an external pull-down resistor and an LED is connected to PB5. When the button is pressed, the LED should be ON and when the button is not pressed, the LED should be OFF.

Write an interrupt-based program for the microcontroller.

Tip: INT0 is on PD2



```
unsigned char* pind = (unsigned char*) 0x29;  
unsigned char* ddrb = (unsigned char*) 0x24;  
unsigned char* portb = (unsigned char*) 0x25;  
unsigned char* sreg = (unsigned char*) 0x5F;  
unsigned char* eimsk = (unsigned char*) 0x3D;  
unsigned char* eicra = (unsigned char*) 0x69;
```

```
int main() //Just change this to void setup() for Arduino  
{  
    *ddrb = 1 << 5; //Set PB5 as output  
    *sreg |= (1 << 7); //Enable interrupts in general  
    *eimsk = 1; //Enable INT0  
    *eicra = 1; //Set interrupt mode to "any logic change"  
}
```

```
ISR(INT0_vect)  
{  
    bool pin_status = ((*pind) & 4);  
    *portb = pin_status << 5; //Same as digitalWrite(13, digitalRead(2));  
}
```

# **Pin Change Interrupt**

- ATmega328p supports 3 pin change interrupts (**PCINT0, PCINT1 and PCINT2**) that map to all 23 GPIO pins
- **PCINT0** is for **PORTB** (8 pins)
- **PCINT1** is for **PORTC** (7 pins)
- **PCINT2** is for **PORTD** (8 pins)
- Only one mode is supported (any change of logic level)
- If PCINT0 gets triggered, you cannot determine which pin actually triggered it. It could be any one of the PORTB pins. (Assuming all the PORTB pins have been set up for PCINT)



# PCICR

Pin Change Interrupt Control Register (x68)

Bit	7	6	5	4	3	2	1	0
0x68	-	-	-	-	-	PCIE2	PCIE1	PCIE0
Read/Write	R	R	R	R	R	R/W	R/W	R/W
Default	0	0	0	0	0	0	0	0

Set PCIE0 to enable PCINT0

Set PCIE1 to enable PCINT1

Set PCIE1to enable PCINT1

# PCIFR

Pin Change Interrupt Flag Register (x3B)

Bit	7	6	5	4	3	2	1	0
0x3B	-	-	-	-	-	PCIF2	PCIF1	PCIF0
Read/Write	R	R	R	R	R	R/W	R/W	R/W
Default	0	0	0	0	0	0	0	0

A flag indicates that a specific interrupt has occurred.

For example:

As soon as PCINT0 gets triggered, PCIF0 becomes HIGH.  
After the end of the ISR routine, PCIF0 becomes LOW again.

# PCMSK0

Pin Change Interrupt Mask Register 0 (x6B)

Bit	7	6	5	4	3	2	1	0
0x6B	PCINT7	PCINT6	PCINT5	PCINT4	PCINT3	PCINT2	PCINT1	PCINT0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Default	0	0	0	0	0	0	0	0

Pin-level interrupt enable for PCINT0 (Port B)

For example:

Setting the 0<sup>th</sup> bit enables pin change interrupt for the PB0

Setting the 1<sup>st</sup> bit enables pin change interrupt for the PB1

Setting the 2<sup>nd</sup> bit enables pin change interrupt for the PB2

# PCMSK1

Pin Change Interrupt Mask Register 1 (x6C)

Bit	7	6	5	4	3	2	1	0
0x6C	-	PCINT14	PCINT13	PCINT12	PCINT11	PCINT10	PCINT9	PCINT8
Read/Write	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Default	0	0	0	0	0	0	0	0

Pin-level interrupt enable for PCINT1 (Port C)

For example:

Setting the 0<sup>th</sup> bit enables pin change interrupt for the PC0

Setting the 1<sup>st</sup> bit enables pin change interrupt for the PC1

Setting the 2<sup>nd</sup> bit enables pin change interrupt for the PC2

# PCMSK2

Pin Change Interrupt Mask Register2 (x6D)

Bit	7	6	5	4	3	2	1	0
0x6D	PCINT23	PCINT22	PCINT21	PCINT20	PCINT19	PCINT18	PCINT17	PCINT16
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Default	0	0	0	0	0	0	0	0

Pin-level interrupt enable for PCINT2 (Port D)

For example:

Setting the 0<sup>th</sup> bit enables pin change interrupt for the PD0

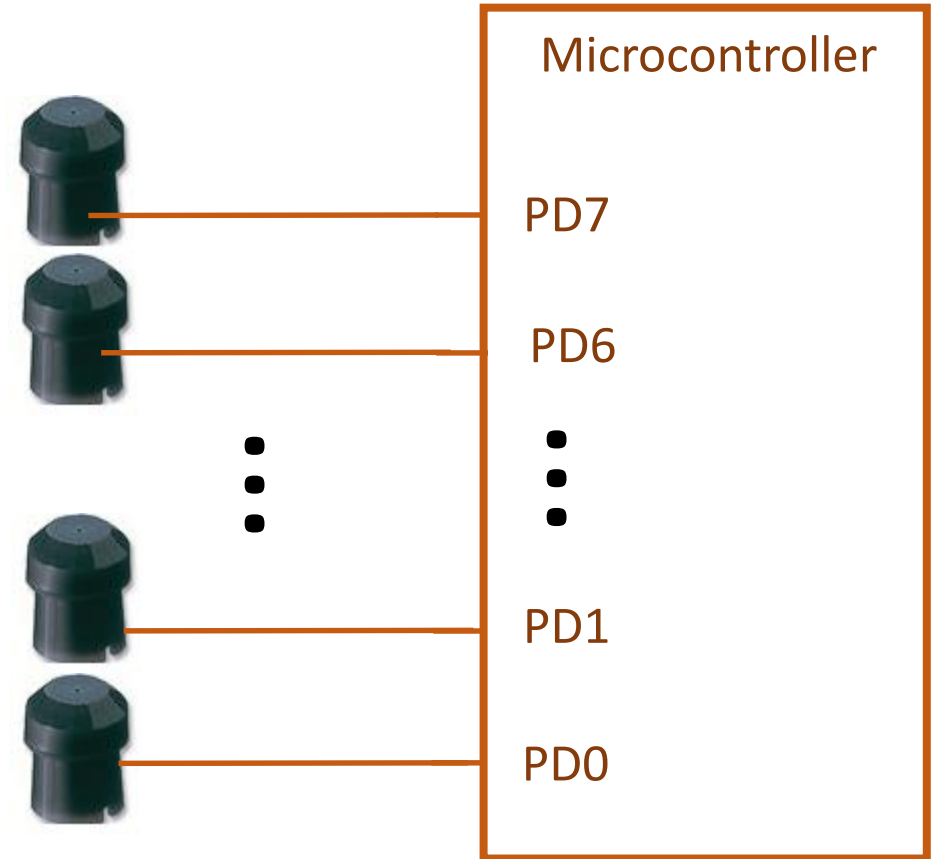
Setting the 1<sup>st</sup> bit enables pin change interrupt for the PD1

Setting the 2<sup>nd</sup> bit enables pin change interrupt for the PD2

# Example – pin change interrupt

An ATmega328p microcontroller is embedded part of a home security system.  
8 motion sensors are connected to the port D of microcontroller.

Pin change interrupt can be used to eliminate polling.



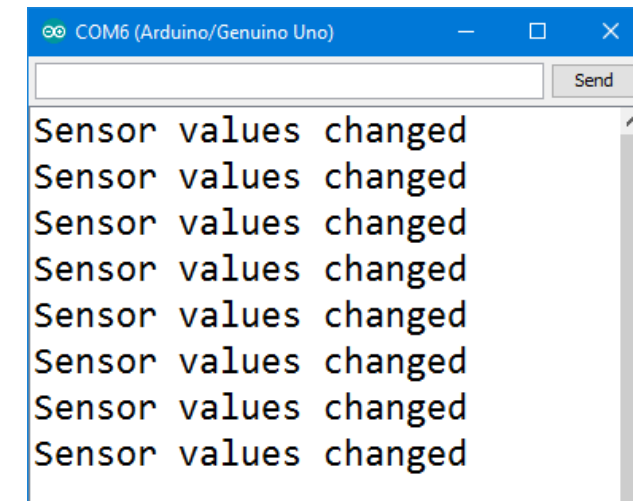
```
volatile bool changed;
unsigned char* sreg = (unsigned char*) 0x5F;
unsigned char* pcicr = (unsigned char*) 0x68;
unsigned char* pcmsk0 = (unsigned char*) 0x6B;

int main()
{
    *sreg |= (1 << 7); //Enable interrupts in general
    *pcicr = 1;        //Enable pin change interrupt 0
    *pcmsk0 = 255;      //Enable pin change interrupt on all the Port B pin
    Serial.begin(9600);

    while(1)
    {
        if (changed)
        {
            Serial.println("Sensor values changed");
            //Perform necessary investigation and subsequent operations
            changed = 0;
        }
        //Do other things or go back to sleep
        Sleep(); //There is no such function called 'Sleep'. Just for demo.
    }
}

ISR(PCINT0_vect)
{
    changed = 1;
}
```

- Most of the time, there will not be any motion and there will not be any change in sensor values.
- The microcontroller can sleep to conserve power (or do other things) as long as the sensor values are static.
- As soon as there is a change in state in any of the port D pins, an interrupt will wake up the microcontroller and necessary operations can be performed.

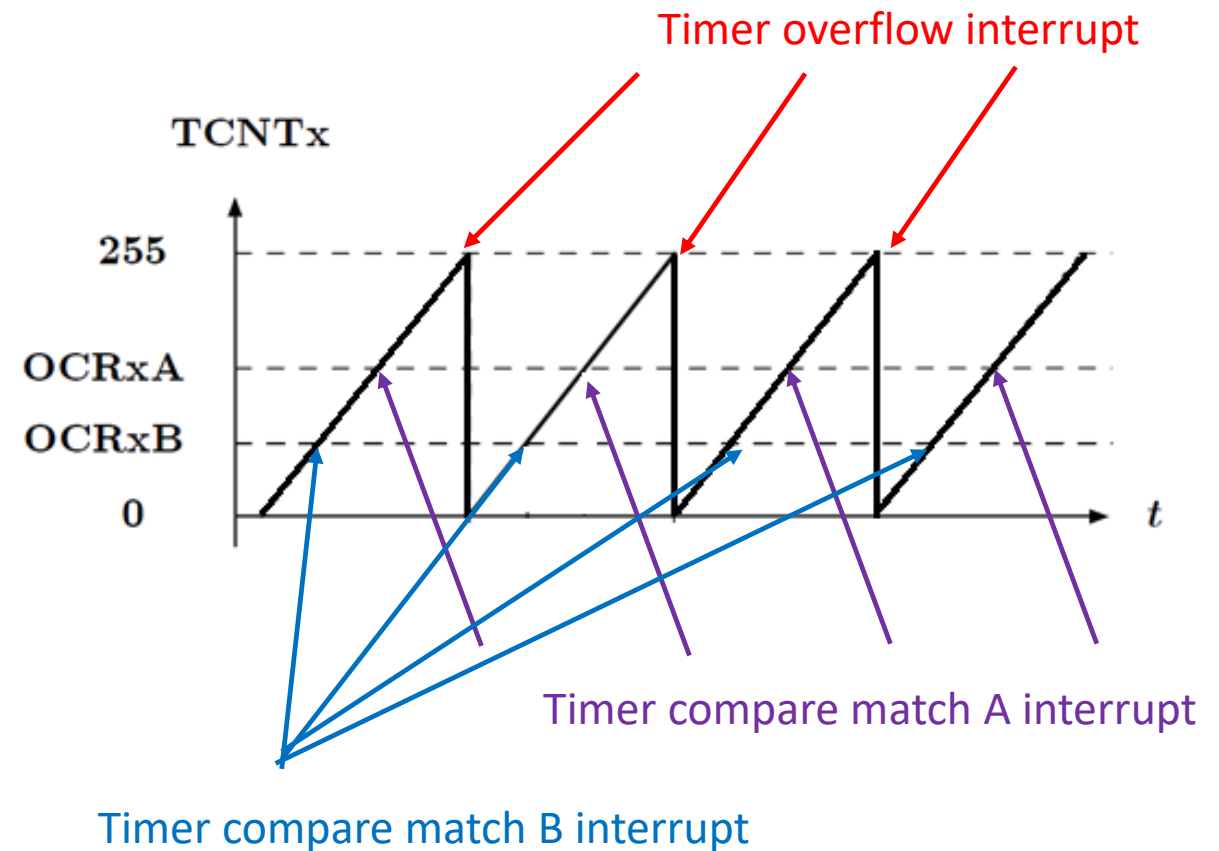


# Timer interrupts



**Each** timer is capable of generating 3 interrupts

- Overflow interrupt
- Compare match A interrupt
- Compare match B interrupt



# TIMSK0

Timer0 Interrupt Mask Register (x6E)

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
Default	0	0	0	0	0	0	0	0

Set to enable compare match  
B interrupt for timer0

Set to enable compare match  
A interrupt for timer0

Set to enable timer0 overflow  
interrupt

# TIFR0

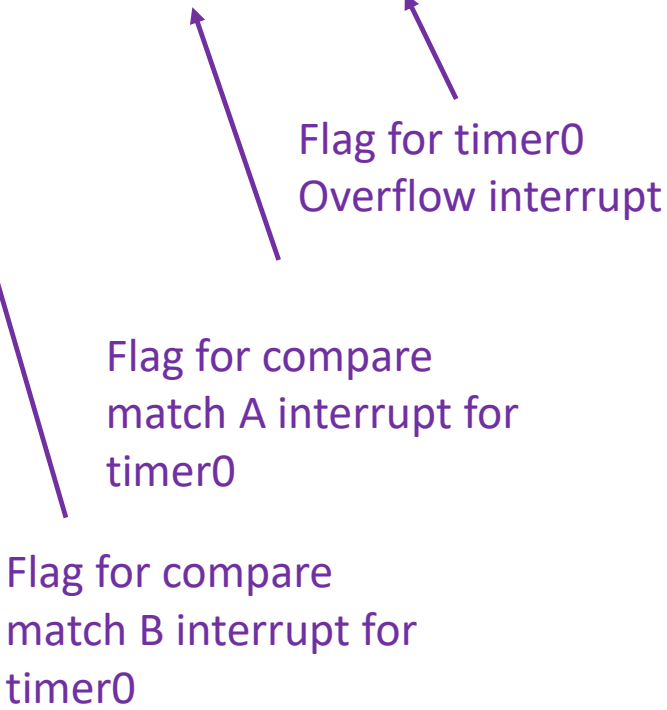
Timer0 Interrupt Flag Register (x35)

Bit	7	6	5	4	3	2	1	0
0x35	-	-	-	-	-	OCFOB	OCFOA	TOV0
Read/Write	R	R	R	R	R	R/W	R/W	R/W
Default	0	0	0	0	0	0	0	0

A flag indicates that a specific interrupt has occurred.

For example:

As soon as timer0 overflow interrupt has occurred, TOV0 is HIGH  
After the end of the ISR routine, TOV0 becomes LOW again.

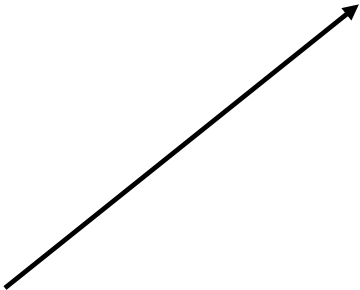


# TIMSK1

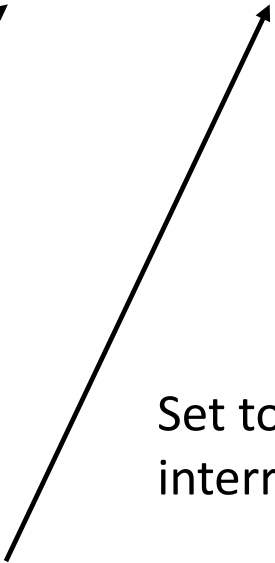
Timer1 Interrupt Mask Register (x6F)

Bit	7	6	5	4	3	2	1	0
0x6F	-	-	ICIE1	-	-	OCIE1B	OCIE1A	TOIE1
Read/Write	R	R	R/W	R	R	R/W	R/W	R/W
Default	0	0	0	0	0	0	0	0


Set to enable compare match  
B interrupt for timer1



Set to enable compare match  
A interrupt for timer1



Set to enable timer1 overflow  
interrupt



# TIFR1

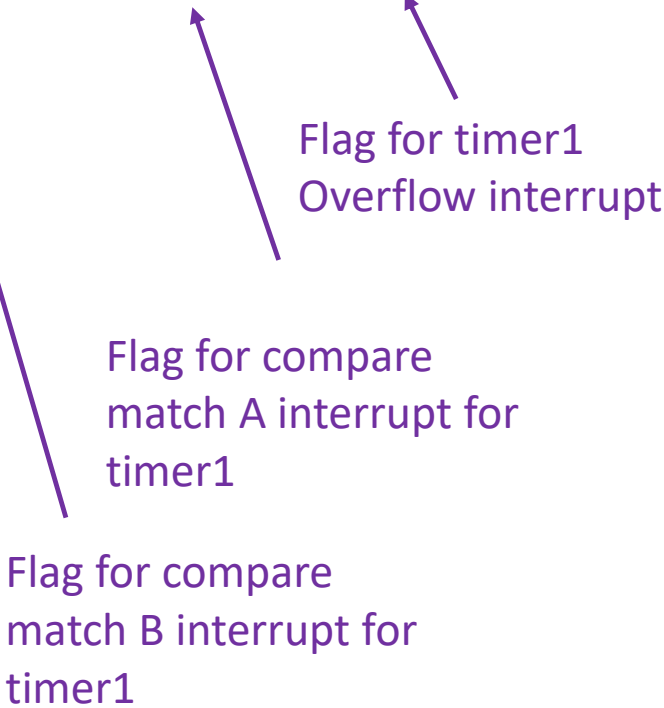
Timer1 Interrupt Flag Register (x36)

Bit	7	6	5	4	3	2	1	0
0x36	-	-	ICF1	-	-	OCF1B	OCF1A	TOV1
Read/Write	R	R	R/W	R	R	R/W	R/W	R/W
Default	0	0	0	0	0	0	0	0

A flag indicates that a specific interrupt has occurred.

For example:

As soon as timer0 overflow interrupt has occurred, TOV1 is HIGH  
After the end of the ISR routine, TOV1 becomes LOW again.



# TIMSK2

Timer2 Interrupt Mask Register (x70)

Bit	7	6	5	4	3	2	1	0
0x70	-	-	-	-	-	OCIE2B	OCIE2A	TOIE2
Read/Write	R	R	R	R	R	R/W	R/W	R/W
Default	0	0	0	0	0	0	0	0

Set to enable compare match  
B interrupt for timer2

Set to enable compare match  
A interrupt for timer2

Set to enable timer2 overflow  
interrupt

# TIFR2

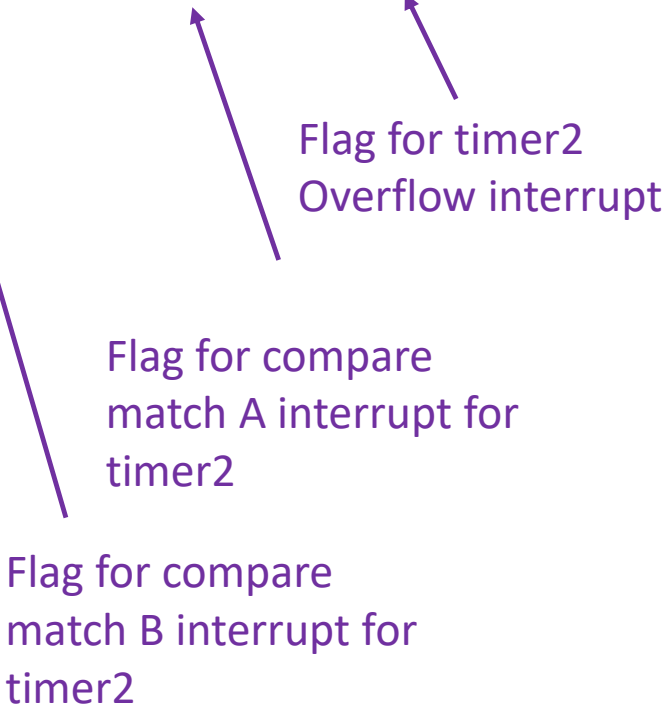
Timer2 Interrupt Flag Register (x37)

Bit	7	6	5	4	3	2	1	0
0x37	-	-	-	-	-	OCF2B	OCF2A	TOV2
Read/Write	R	R	R	R	R	R/W	R/W	R/W
Default	0	0	0	0	0	0	0	0

A flag indicates that a specific interrupt has occurred.

For example:

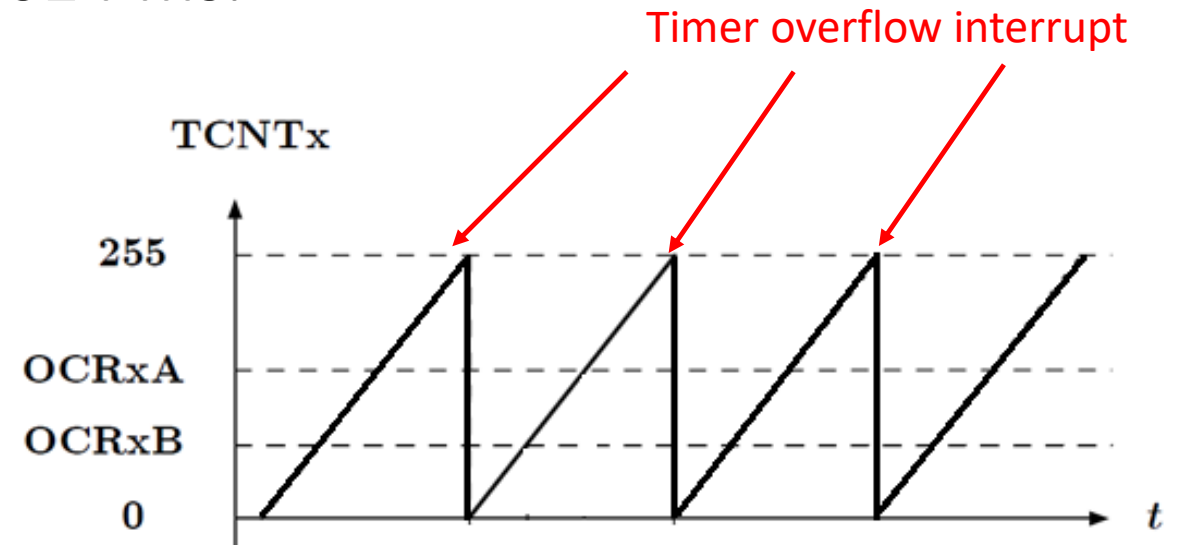
As soon as timer0 overflow interrupt has occurred, TOV2 is HIGH  
After the end of the ISR routine, TOV2 becomes LOW again.



# Example – Timer Interrupt

- `millis()` function utilizes timer0 overflow interrupt.
- On Arduino, timer0 is configured as 8-bit fast PWM with pre-scaler = 64.
- If the frequency of the system is 16MHz, the timer0 overflows 976.56 times per second.
- The period of timer flow interrupt is 1.024 ms.

Lets blink an LED (1 sec ON and 1 sec OFF) on PD5 without any library (using timer2 overflow interrupt)





```
volatile unsigned long ms;
```

```
unsigned char* tccr2a = (unsigned char*) 0xB0;  
unsigned char* tccr2b = (unsigned char*) 0xB1;  
unsigned char* timsk2 = (unsigned char*) 0x70;  
unsigned char* portb = (unsigned char*) 0x25;  
unsigned char* ddrb = (unsigned char*) 0x24;  
unsigned char* sreg = (unsigned char*) 0x5F;
```

```
unsigned long previousTime;
```

```
ISR(TIMER2_OVF_vect)
```

```
{  
    ms++;  
}
```

```
unsigned long getMilliseconds()
```

```
{  
    *timsk2 = 0;    //Disable timer2 interrupts  
    unsigned long val = ms; //Read the variable  
    *timsk2 = 1;    //Re-enable timer2 interrupts  
    return val;  
}
```

```
int main()
```

```
{  
    *ddrb = 32;        //Set PD5 as output  
    *tccr2a = 3;       //8-bit Fast PWM on timer2  
    *tccr2b = 4;       //Set pre-scaler to 64  
    *sreg |= 1 << 7;  //Enable interrupts  
    *timsk2 = 1;       //Enable timer2 overflow interrupt  
  
    while(1)  
    {  
        unsigned long currentTime = getMilliseconds();  
        if ((*portb) & 32) //If LED is currently ON  
        {  
            if (currentTime - previousTime >= 1000)  
            {  
                *portb = 0; //Turn it OFF  
                previousTime = currentTime;  
            }  
        }  
        else //If LED is currently OFF  
        {  
            if (currentTime - previousTime >= 1000)  
            {  
                *portb = 32; //Turn it ON  
                previousTime = currentTime;  
            }  
        }  
    }  
}
```

# Interrupt Etiquette

- Keep ISRs as short as possible.
- It is a good practice to set some flags inside ISRs and do actual heavy lifting inside the main loop.
- Do not perform serial communication or use delay or any other functions that utilize interrupts inside ISRs.
- Variables that are shared between ISRs and the main function should be declared as volatile.
- Large variables (double, strings, arrays, structures, classes, etc) need several CPU cycles to update. If an interrupt occurs in the middle of an update, the data can get corrupted. Disable global interrupts before accessing them.