بسم الله الرحمن الرحیم

طراحی سیستم‌های ریزپردازنده

وقفه‌های ارتباط سریال و خارجی

دانشکده مهندسی برق

دانشگاه صنعتی شاهرود

حسین خسروی

۱۴۰۰

Interrupt

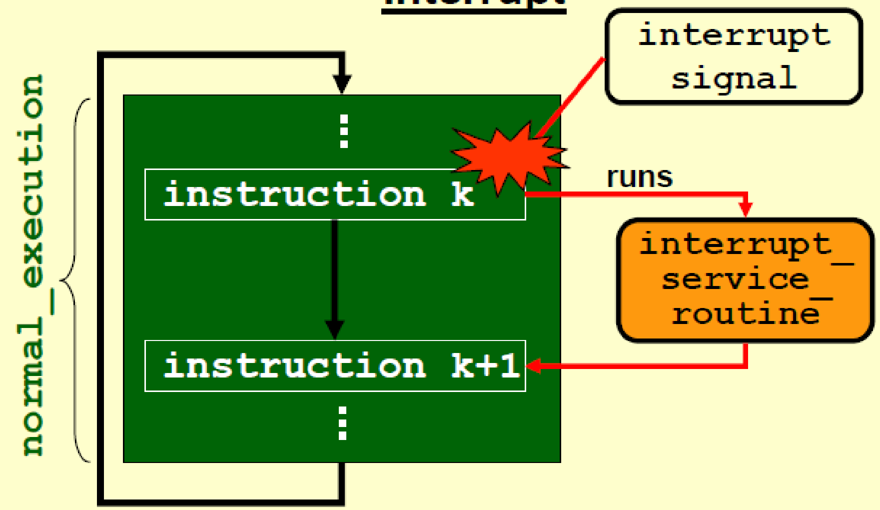
- Two ways for handling peripheral devices
  - Polling
  - Interrupt
- Compared to polling, interrupt is a more efficient approach for the CPU to handle peripheral devices, e.g.
  - serial port, external switches, timers, PWM and ADC.

Polling versus Interrupt

Polling

```
while (1){
    get_device_status;
    if (service_required){
        service_routine;
    }
    normal_execution;
}
```

Interrupt



- Using polling, the CPU must continually check the device's status.
- Using interrupt:
  - A device will send an interrupt signal when needed.
  - In response, the CPU will perform an interrupt service routine, and then resume its normal execution.

Interrupt execution sequence

1. A device issues an interrupt
2. CPU finishes the current instruction
3. CPU acknowledges the interrupt
4. CPU saves its states and PC onto stack
5. CPU loads the address of ISR onto PC
6. CPU executes the ISR
7. CPU retrieves its states and PC from stack
8. CPU continue to the next instruction

ATmega32 interrupt subsystem

- The ATmega32 has 21 interrupts:
  - 1 reset interrupt
  - 3 external interrupts
  - 8 timer interrupts
  - 3 serial port interrupts
  - 1 ADC interrupt
  - 1 analogue comparator interrupt
  - 1 SPI interrupt
  - 1 TWI interrupt
  - 2 memory interrupts

ATmega32 interrupt subsystem: Complete list

Vector No.	Program Address	Interrupt vector name	Description
1	$000	RESET_vect	Reset
2	$002	INT0_vect	External Interrupt Request 0
3	$004	INT1_vect	External Interrupt Request 1
4	$006	TIMER2_COMP_vect	Timer/Counter2 Compare Match
5	$008	TIMER2_OVF_vect	Timer/Counter2 Overflow
6	$00A	TIMER1_CAPT_vect	Timer/Counter1 Capture Event
7	$00C	TIMER1_COMPA_vect	Timer/Counter1 Compare Match A
8	$00E	TIMER1_COMPB_vect	Timer/Counter1 Compare Match B
9	$010	TIMER1_OVF_vect	Timer/Counter1 Overflow
10	$012	TIMER0_OVF_vect	Timer/Counter0 Overflow
11	$014	SPI_STC_vect	Serial Transfer Complete
12	$016	USART_RXC_vect	USART, Rx Complete
13	$018	USART_UDRE_vect	USART Data Register Empty
14	$01A	USART_TXC_vect	USART, Tx Complete
15	$01C	ADC_vect	ADC Conversion Complete
16	$01E	EE_RDY_vect	EEPROM Ready
17	$020	ANA_COMP_vect	Analog Comparator
18	$022	TWI_vect	2-wire Serial Interface
19	$024	INT2_vect	External Interrupt Request 2
20	$026	TIMER0_COMP_vect	Timer/Counter0 Compare Match
21	$028	SPM_RDY_vect	Store Program Memory Ready

Remarks on previous Table

- **Vector No**
  - An interrupt with a lower 'Vector No' will have a higher priority.
  - E.g., INT0 has a higher priority then INT1 and INT2.
- **Program Address**
  - The fixed memory location for a given interrupt handler.
  - E.g., in response to interrupt INT0, CPU runs instruction at $002.
- **Interrupt Vector Name**
  - This is the interrupt name, to be used with C macro ISR().

Remarks

- When an interrupt occurs, the Global Interrupt Enable I-bit is cleared and all interrupts are dis-abled!
- The user software can write logic one to the I-bit to enable nested interrupts.
- All enabled interrupts can then interrupt the current interrupt routine.
- The I-bit is automatically set when a Return from Interrupt instruction – RETI – is executed.

Bit	7	6	5	4	3	2	1	0	
	I	T	H	S	V	N	Z	C	SREG
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	

- **Bit 7 – I: Global Interrupt Enable**

Program setup for the Reset and Interrupt Vector Addresses

Address	Labels	Code		Comments
$000		jmp	RESET	; Reset Handler
$002		jmp	EXT_INT0	; IRQ0 Handler
$004		jmp	EXT_INT1	; IRQ1 Handler
$006		jmp	EXT_INT2	; IRQ2 Handler
$008		jmp	TIM2_COMP	; Timer2 Compare Handler
$00A		jmp	TIM2_OVF	; Timer2 Overflow Handler
…				
$028		jmp	SPM_RDY	; SPM Ready Handler
$02A	RESET:	ldi	r16,high(RAMEND)	; Main program start
$02B		out	SPH,r16	; Set Stack Pointer to top of RAM
$02C		ldi	r16,low(RAMEND)	
$02D		out	SPL,r16	
$02E		sei		; Enable interrupts
$02F		<instr>	xxx	
… … …				

Steps to program an interrupt in C

- To program an interrupt in C, five steps are required.
  - 1. Include header file <avr\interrupt.h>.
  - 2. Use C macro ISR() to declare the interrupt handler and update IVT.
  - 3. Enable the specific interrupt.
  - 4. Configure details about the interrupt by setting relevant registers.
  - 5. Enable the interrupt subsystem globally using sei().

Using C macro ISR()

- The C macro ISR() is used to declare the handler for a given interrupt.
- Its basic syntax is given as

```
ISR(interrupt_vector_name){
   // … code for interrupt handler here
}
```

- where `interrupt_vector_name` is given in Previous Table.

- Example: To process interrupt 'RXD Complete' and put the received character in Port B, we write:

```
ISR(USART_RXC_vect){
   PORTB = UDR; // put the received char in Port B
}
```

Serial RXC interrupt

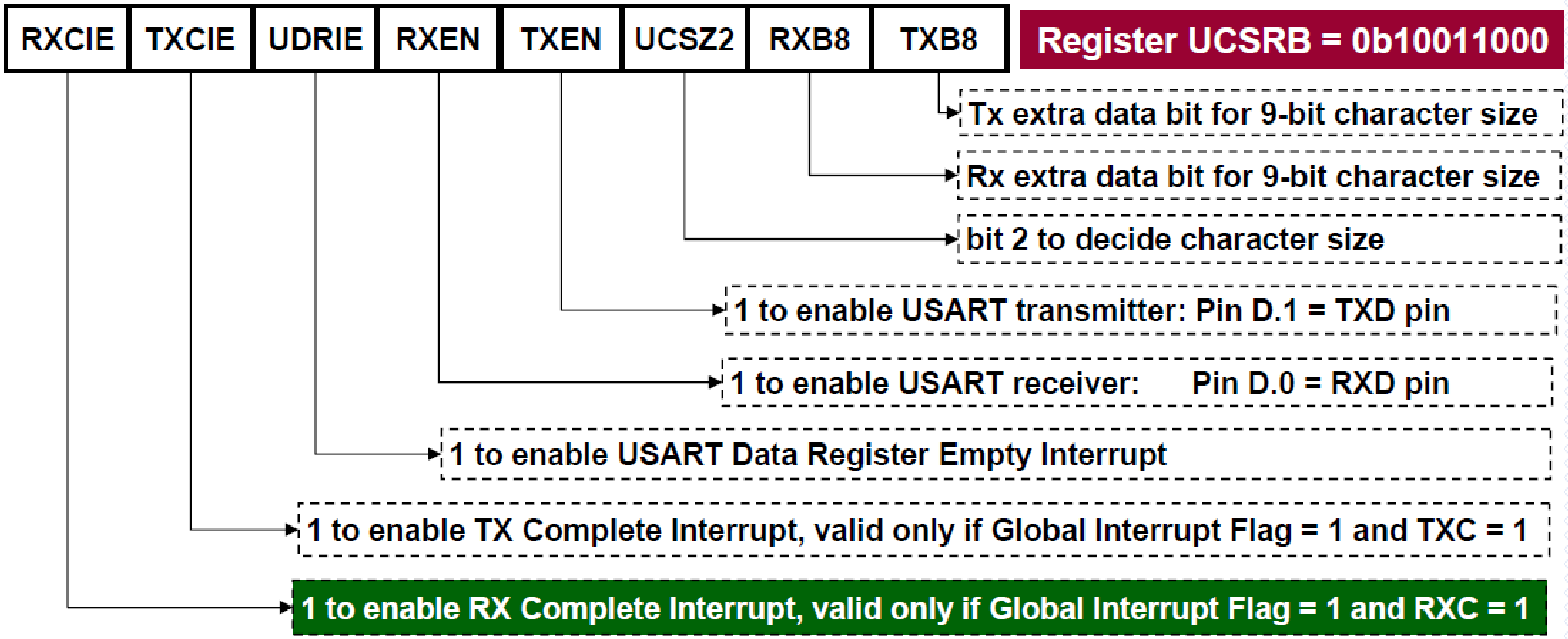
- **Write a C interrupt-driven program to use the serial port of ATmega16 at baud rate 1200, no parity, 1 stop bit, 8 data bits, clock speed 1MHz. Whenever a character is received, it should be sent to Port B.**

- **The serial port on ATmega32 can trigger an RXC interrupt whenever a character is received.**

- **We enable this interrupt by setting a flag in a serial port register.**

- **We then need to write the interrupt handler, to be run whenever the interrupt is triggered.**

Serial RXC interrupt: Enabling



- **For any interrupt, the ATmega32 datasheet can be searched to learn how to enable the interrupt.**
- **E.g., for serial RXC interrupt, we look at 'USART' section.**

Serial RXC interrupt: serial_int.c

```c
#include <avr/io.h>
#include <avr/interrupt.h>

void USART_init(void){
    // Normal speed, disable multi-proc
    UCSRA = 0b00000000;

    // Enable Tx and Rx pins, enable RX interrupt
    UCSRB = 0b10011000;

    // Asynchronous mode, no parity, 1 stop bit, 8 data bits
    UCSRC = 0b10000110;

    // Baud rate 1200bps, assuming 1MHz clock
    UBRRL = 0x33; UBRRH = 0x00;
}

ISR(USART_RXC_vect){ // Handler for RXD interrupt
    PORTB = UDR;       // Received character is displayed on port B
}

int main(void) {
    USART_init(); // initialise USART
    sei();        // enable interrupt subsystem globally
    DDRB = 0xFF;  // set port B for output
    while (1) {;} // infinite loop
    return 0;
}
```

1 3 2 4 5

Serial RXC interrupt: Testing

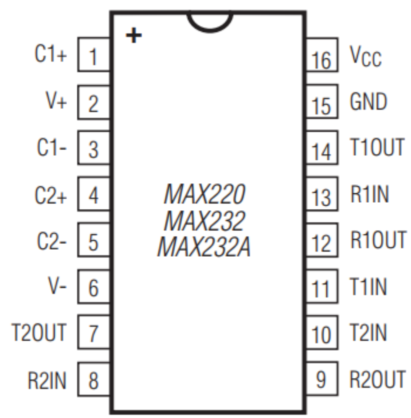
- **To test the serial RXD interrupt example:**
  - **Connect RXD pin (pin D.0) to RXD pin of RS232 Spare (Max232).**
  - **Connect TXD pin (pin D.1) to TXD pin of RS232 Spare.**
  - **Connect Port B to LED connector.**
  - **Compile, download program.**
  - **Connect RS232 Spare Connector to Serial Port of PC.**
  - **Configure and run HyperTerminal and use it to send characters.**

- **Video demo link: [avr]/ecte333/serial_int.mp4**
  - **avr = http://www.elec.uow.edu.au/avr**

MAX232 – TTL to RS232

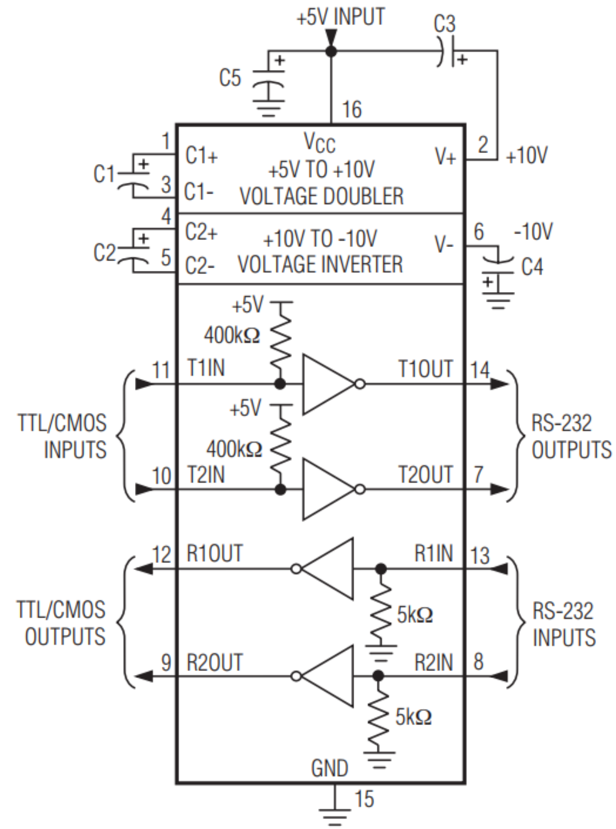
MAX220/MAX232/MAX232A Pin Configuration and Typical Operating Circuit

TOP VIEW



DIP/SO

CAPACITANCE (μF)					
DEVICE	C1	C2	C3	C4	C5
MAX220	0.047	0.33	0.33	0.33	0.33
MAX232	1.0	1.0	1.0	1.0	1.0
MAX232A	0.1	0.1	0.1	0.1	0.1



MAX233 – No Capacitor Required

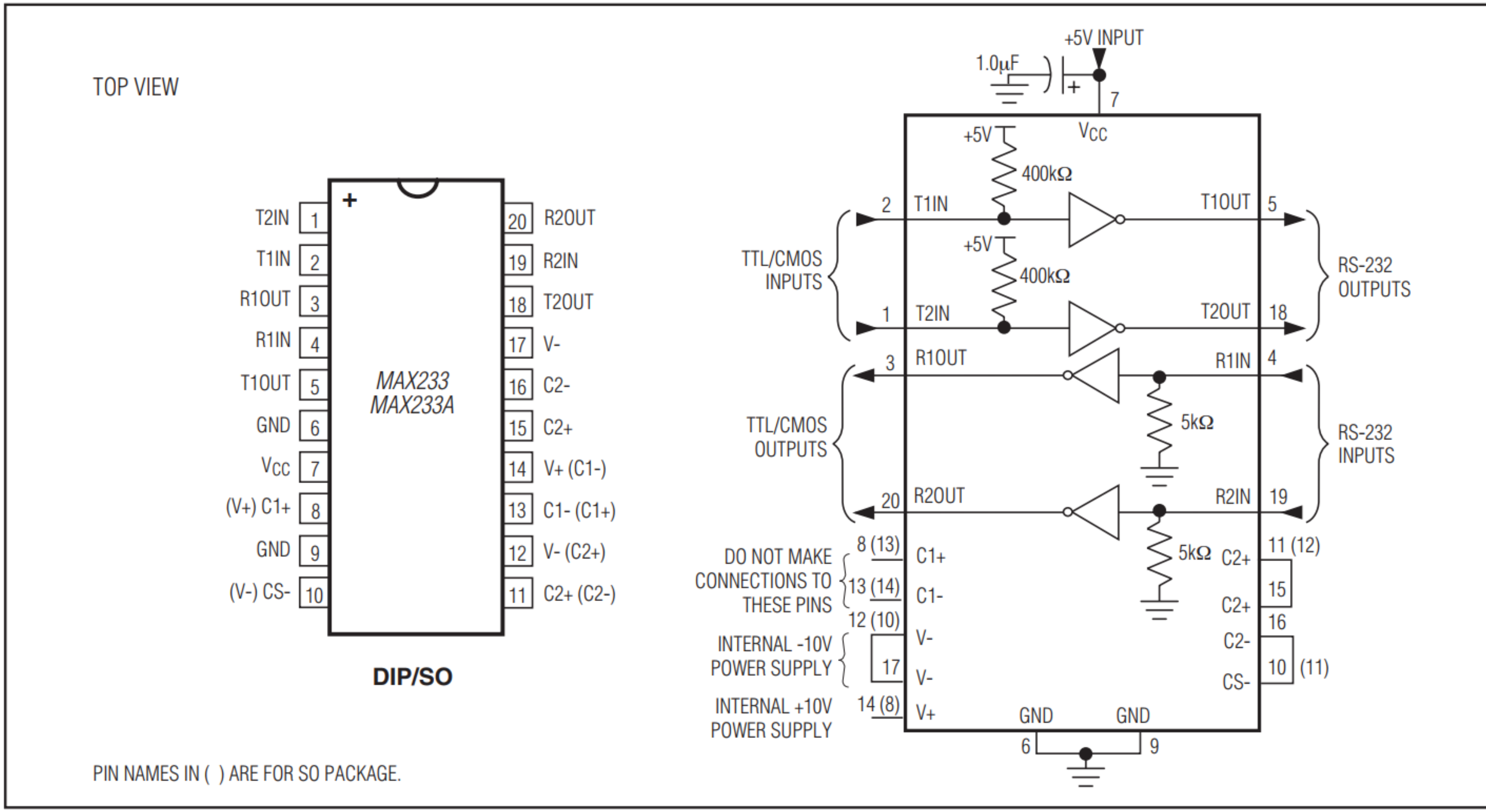


Figure 11. MAX233/MAX233A Pin Configuration and Typical Operating Circuit

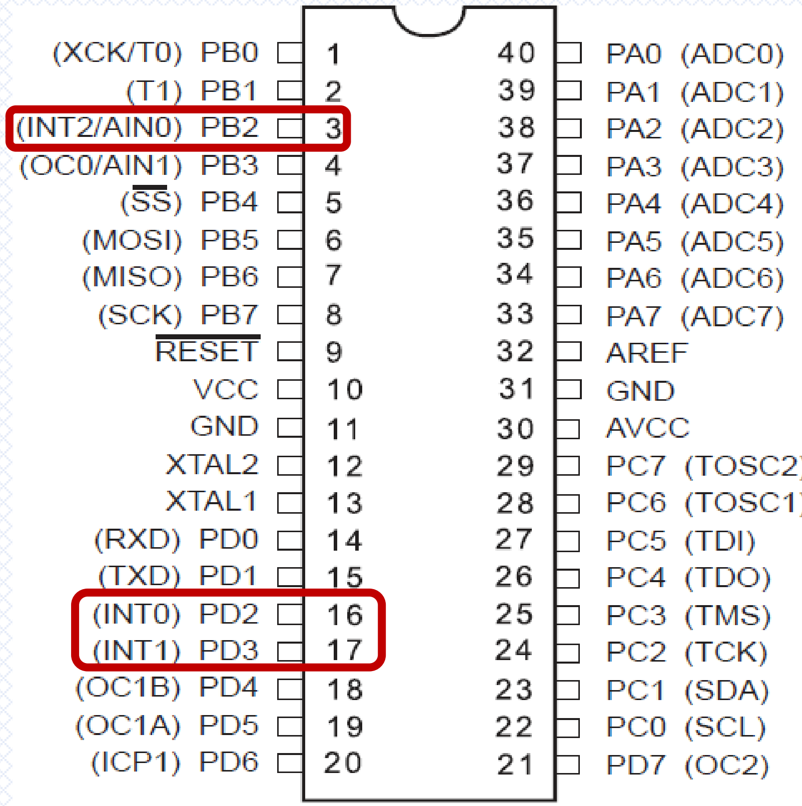
Serial RXC – Polling approach

- For comparison, the program below uses polling for the same effect.

```
#include <avr/io.h>

void USART_init(void){
   // Normal speed, disable multi-proc
   UCSRA = 0b00000000;

   // Enable Tx and Rx, disable interrupts
   UCSRB = 0b00011000;

   // Asynchronous mode, no parity, 1 stop bit, 8 data bits
   UCSRC = 0b10000110;

   // Baud rate 1200bps, assuming 1MHz clock
   UBRRL = 0x33; UBRRH = 0x00;
}

int main(void) {
   USART_init(); // initialise USART
   DDRB = 0xFF;  // set port B for output
   while (1) {   // infinite loop
        // Poll until RXC flag = 1
       while ((UCSRA & (1<<RXC)) == 0x00){;}
       PORTB = UDR;  // received character is displayed on port B
   }
   return 0;
}
```

External interrupts

- Three external interrupts can be triggered.
  - INT0 on pin D.2,
  - INT1 on pin D.3,
  - INT2 on pin B.2.



- Key steps in using external interrupts.
  - Enable the interrupt,
  - Specify what types of event will trigger the interrupt.

External interrupts: Enabling

- **To enable an external interrupt, set a flag in General Interrupt Control Register (GICR).**

	INT1	INT0	INT2	-	-	-	IVSEL	IVCE
Read/Write	R/W	R/W	R/W	R	R	R	R/W	R/W
Initial value	0	0	0	0	0	0	0	0

- **Example: to enable INT1 on pin D.3, we can write**
  - `GICR = (1 << INT1);`

- **Note that `INT1` and `GICR` names are already defined in <avr/io.h>.**

External interrupts: Specifying events

- **To specify the type of events that triggers an external interrupt, set MCU Control Register or MCU Control and Status Register.**

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	
	SM2	SE	SM1	SM0	ISC11	ISC10	ISC01	ISC00	MCUCR

ISC11	ISC10	Description
0	0	The low level of INT1 generates an interrupt request.
0	1	Any logical change on INT1 generates an interrupt request.
1	0	The falling edge of INT1 generates an interrupt request.
1	1	The rising edge of INT1 generates an interrupt request.

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.

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	
	JTD	ISC2	-	JTRF	WDRF	BORF	EXTRF	PORF	MCUCSR

0: falling edge generates an interrupt INT2
1: rising edge generates an interrupt INT2

External interrupts: Example

- Write a C interrupt-driven program to toggle port B whenever a switch on the STK500 board is pressed. The program should use an external interrupt.

- **Let us use interrupt INT1. This interrupt is triggered on pin D.3.**
- **To enable interrupt INT1**
  - `GICR = (1 << INT1);`
- **To specify that INT1 is triggered on any change in pin D.3**
  - `MCUCR = (1<<ISC10);`
- **We then write interrupt handler and enable interrupt subsystem globally as usual.**

External interrupts: ext_int.c

```
#include <avr/io.h>
#include <avr/interrupt.h>                                      1

ISR(INT1_vect){          // handler for INT1 interrupt
   PORTB = (~PORTB);     // toggle port B                       2
}

int main(void) {
   GICR = (1<< INT1);   // enable interrupt INT1                3
   MCUCR = (1<<ISC10); // triggered on any change to INT1 pin (D.3)  4
   sei();               // enable interrupt subsystem globally  5
   DDRB = 0xFF;         // set port B for output
   PORTB = 0b10101010; // initial value
   while (1) {;}        // infinite loop
   return 0;
}
```

Example: Pulse Counter

Example: Counter 0-9999