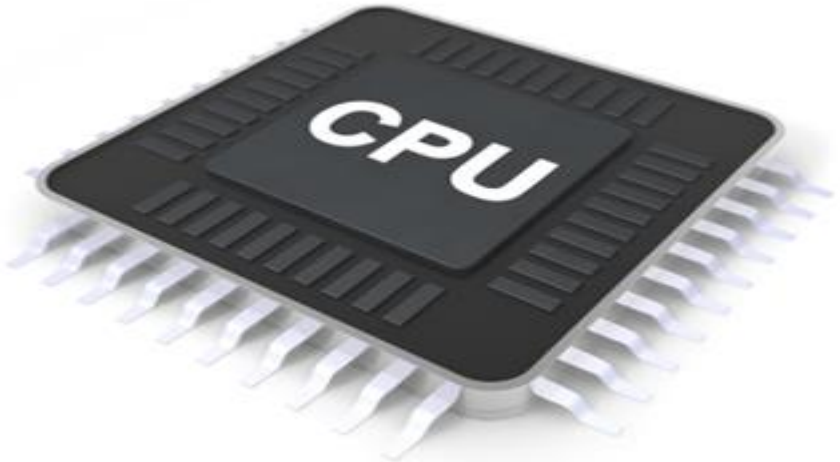


Interrupts

ENEE 3587

Microp Interfacing

Hardware Interrupt

- Asynchronous event initiated by a hardware signal
- Examples of events:
  - IO device management
  - Performance of routine tasks
  - Response to emergency events
- The interrupt causes a function call to an interrupt service routine
  - Involves considerable “overhead”
- Polling methods can be used to detect flags set by these events
  - Requires constant monitoring of flags
  - Advantage: faster response to events than function calls
  - Disadvantages: power “hungry”. Difficult to monitor multiple events concurrently

Hardware Interrupt Terminology

- Non-maskable interrupts: interrupts that don't require enabling
- Maskable interrupts: interrupts that must be enabled
- Mask bit: the enable bit
- Interrupt Service Routine:
  - a function that is executed when an interrupts occurs
- Interrupt vector table: organized by number and corresponding address
  - Each interrupt has a unique number
  - Vector address is used to store address of ISR

Interrupt Steps (Overhead)

1. Finishes executing current instruction
2. PUSHs the address of the next instruction (PC) on the stack.
3. Disable global interrupt bit
   - To prevent other bits from causing interrupts
4. Jumps to a fixed location in memory called the interrupt vector table.
   - The interrupt vector table holds the address of the interrupt service routine (ISR).
5. Jump to ISR
6. Return from interrupt: POPs (PC) address from stack
7. Set global interrupt bit

- Steps 2-5: 3 cycles min – 5 cycles min
- Steps 6-7: 5 cycles min

Microcontroller Configuration Register (MCUCR)

7	6	5	4	3	2	1	0	
JTD	—	—	PUD	—	—	IVSEL	IVCE	Reset value = 0x00
R/W	R	R	R/W	R	R	R/W	R/W	

- **JTD**: JTAG Disable
  - 0: JTAG interface is Enabled.
  - 1: JTAG interface is Disabled.
- **PUD**: Pullup Disable
  - 0: Enable pull-ups of all pins of all I/O PORTs.
  - 1: Disable pull-ups of all pins of all I/O PORTs.
- **IVSEL**: Interrupt vector select
  - 0: Interrupt vectors are placed at the start of the flash memory (i.e., 0x0002).
  - 1: Interrupt vectors are placed at the start of the boot section. The start of the boot section is set by programming the BOOTSZ fuses.
- **IVCE**: Interrupt vector change enable
  - 0: Disable interrupt vector change
  - 1: Enable interrupt vector to be changed

BOOTRST	IVSEL	Reset address	Interrupt vectors start address
1	0	0x0000	0x0002
1	1	0x0000	Boot reset address + 0x0002
0	0	Boot reset address	0x0002
0	1	Boot reset address	Boot reset address + 0x0002

Enabling Interrupts

- Aka unmasking
- Each maskable interrupts requires two mask bits:
  - Global mask bit: an enable bit to enable all maskable interrupts
  - Local mask bit: an enable bit to enable specific maskable interrupts
- Global enable: bit I (bit 7) of the status register (SREG)
  - From interrupt.h, use: sei() to set and cli() to clear
- Local enable: enable bit in the interrupt enable register
  - E.g. TIMSK

ATmega Interrupt Table

Vector no.	Vector addr.	Source	Interrupt definition
1	0x0000	RESET	Reset/Power-on,brown-out,watchdog, JTAG reset
2	0x0002	INT0	External interrupt request 0
3	0x0004	INT1	External interrupt request 1
4	0x0006	INT2	External interrupt request 2
5	0x0008	INT3	External interrupt request 3
6	0x000A	INT4	External interrupt request 4
7	0x000C	INT5	External interrupt request 5
8	0x000E	INT6	External interrupt request 6
9	0x0010	INT7	External interrupt request 7
10	0x0012	PCINT0	Pin change interrupt request 0
11	0x0014	PCINT1	Pin change interrupt request 1
12	0x0016	PCINT2	Pin change interrupt request 2
13	0x0018	WDT	Watchdog time-out interrupt
14	0x001A	Timer2 COMPA	Timer/Counter2 compare match A
15	0x001C	Timer2 COMPB	Timer/Counter2 compare match B
16	0x001E	Timer2 OVF	Timer/Counter2 overflow
17	0x0020	Timer1 CAPT	Timer/Counter1 capture event
18	0x0022	Timer1 COMPA	Timer/Counter1 compare match A
19	0x0024	Timer1 COMPB	Timer/Counter1 compare match B
20	0x0026	Timer1 COMPC	Timer/Counter1 compare match C
21	0x0028	Timer1 OVF	Timer/Counter1 overflow
22	0x002A	Timer0 COMPA	Timer/Counter0 compare match A
23	0x002C	Timer0 COMPB	Timer/Counter0 compare match B
24	0x002E	Timer0 OVF	Timer/Counter0 overflow
25	0x0030	SPI, STC	SPI serial transfer complete
26	0x0032	USATR0	RX USART0 Rx complete
27	0x0034	USATR0	UDRE USART0 data register empty
28	0x0036	USART0	TX USART0 Tx complete

Vector no.	Vector addr.	Source	Interrupt definition
29	0x0038	Analog	Comp Analog comparator
30	0x003A	ADC	ADC conversion complete
31	0x003C	EE Ready	EEPROM ready
32	0x003E	Timer3 CAPT	Timer/Counter3 capture event
33	0x0040	Timer3 COMPA	Timer/Counter3 compare match A
34	0x0042	Timer3 COMPB	Timer/Counter3 compare match B
35	0x0044	Timer3 COMPC	Timer/Counter3 compare match C
36	0x0046	Timer3 OVF	Timer/Counter3 overflow
37	0x0048	USART1 RX	USART1 Rx complete
38	0x004A	USART1 UDRE	USART1 data register empty
39	0x004C	USART1 TX	USART1 Tx complete
40	0x004E	TWI	2-wire serial interface
41	0x0050	SPM	Ready Store program memory ready
42	0x0052	Timer4 CAPT	Timer/Counter4 capture event
43	0x0054	Timer4 COMPA	Timer/Counter4 compare match A
44	0x0056	Timer4 COMPB	Timer/Counter4 compare match B
45	0x0058	Timer4 COMPC	Timer/Counter4 compare match C
46	0x005A	Timer4 OVF	Timer/Counter4 overflow
47	0x005C	Timer5 CAPT	Timer/Counter5 capture event
48	0x005E	Timer5 COMPA	Timer/Counter5 compare match A
49	0x0060	Timer5 COMPB	Timer/Counter5 compare match B
50	0x0062	Timer5 COMPC	Timer/Counter5 compare match C
51	0x0064	Timer5 OVF	Timer/Counter5 overflow
52	0x0066	USART2 RX	USART2 Rx complete
53	0x0068	USART2 UDRE	USART2 data register empty
54	0x006A	USART2 TX	USART2 Tx complete
55	0x006C	USATR3 RX	USART3 Rx complete
56	0x006E	USATR3 UDRE	USART3 data register empty
57	0x0070	USART3 TX	USART3 Tx complete

RESET

- **Power-on reset:** reset when VCC is below the power-on reset threshold (VPOT)
- **External reset:** RESET pin signal is low longer than the min pulse width
- **Brown-out reset:** VCC is lower than the brown-out reset threshold (VBOT–)
  - the brown-out detector must be enabled.
- **JTAG AVR reset.** The MCU is reset as long as there is a logic 1 in the reset register
  - Joint Test Action Group
  - interface used for debugging and programming MCU
  - Requires special connector
- **Watchdog reset.** The MCU is reset when the watchdog timer overflows and the watchdog is enabled.
  - A counter used to ensure that program doesn't hang

MCU Status Reg (MCUSR)

7	6	5	4	3	2	1	0
—	—	—	JTRF	WDRF	BORF	EXTRF	PORF
R	R	R	R/W	R/W	R/W	R/W	R/W

- Flags reset by power-on or writing 0 into flag.
- **JTRF**: JTAG reset flag
  - 0 = Reset is not caused by JTAG reset.
  - 1 = Reset is caused by a logic 1 in the JTAG reset register selected by the JTAG instruction AVR_RESET.
- **WDRF**: Watchdog reset flag
  - 0 = Reset is not caused by watchdog time out.
  - 1 = Reset is caused by watchdog time out (timer overflow).
- **BORF**: Brown-out reset flag
  - 0 = Reset is not caused by brown out.
  - 1 = Reset is caused by brown-out reset.
- **EXTRF**: External reset flag
  - 0 = Reset is not caused by RESET pin.
  - 1 = Reset is caused by RESET pin.
- **PORF**: Power-on reset flag
  - 0 = Reset is not caused by power-on reset.
  - 1 = Power-on reset occurred.

Interrupt Code Template

```
// define global variables
#include <avr/io.h>
#include <avr/interrupt.h>
int main (void)
{                   // enable local interrupt
   sei();           // enable interrupts globally
}

ISR (ISR_name)   // ISR_name is from table on next slide
{

}
```

Interrupt.h: ISR Symbolic Name

Interrupt source	Vector no.	ISR name
INT0	2	(INT0_vect)
INT1	3	(INT1_vect)
INT2	4	(INT2_vect)
INT3	5	(INT3_vect)
INT4	6	(INT4_vect)
INT5	7	(INT5_vect)
INT6	8	(INT6_vect)
INT7	9	(INT7_vect)
PCINT0	10	(PCINT0_vect)
PCINT1	11	(PCINT1_vect)
PCINT2	12	(PCINT2_vect)
WDT	13	(WDT_vect)
TIMER2 COMPA	14	(TIMER2_COMPA_vect)
TIMER2 COMPB	15	(TIMER2_COMPB_vect)
TIMER2 OVF	16	(TIMER2_OVF_vect)
TIMER1 CAPT	17	(TIMER1_CAPT_vect)
TIMER1 COMPA	18	(TIMER1_COMPA_vect)
TIMER1 COMPB	19	(TIMER1_COMPB_vect)
TIMER1 COMPC	20	(TIMER1_COMPC_vect)
TIMER1 OVF	21	(TIMER1_OVF_vect)
TIMER0 COMPA	22	(TIMER0_COMPA_vect)
TIMER0 COMPB	23	(TIMER0_COMPB_vect)
TIMER0 OVF	24	(TIMER0_OVF_vect)
SPI STC	25	(SPI_STC_vect)
USART0 RX	26	(USART0_RX_vect)
USART0 UDRE	27	(USART0_UDRE_vect)
USART0 TX	28	(USART0_TX_vect)
ANALOG COMP	29	(ANALOG_COMP_vect)

Interrupt source	Vector no.	ISR name
ADC	30	(ADC_vect)
EE READY	31	(EE_READY_vect)
TIMER3 CAPT	32	(TIMER3_CAPT_vect)
TIMER3 COMPA	33	(TIMER3_COMPA_vect)
TIMER3 COMPB	34	(TIMER3_COMPB_vect)
TIMER3 COMPC	35	(TIMER3_COMPC_vect)
TIMER3 OVF	36	(TIMER3_OVF_vect)
USART1 RX	37	(USART1_RX_vect)
USART1 UDRE	38	(USART1_UDRE_vect)
USART1 TX	39	(USART1_TX_vect)
TWI	40	(TWI_vect)
SPM READY	41	(SPM_READY_vect)
TIMER4 CAPT	42	(TIMER4_CAPT_vect)
TIMER4 COMPA	43	(TIMER4_COMPA_vect)
TIMER4 COMPB	44	(TIMER4_COMPB_vect)
TIMER4 COMPC	45	(TIMER4_COMPC_vect)
TIMER4 OVF	46	(TIMER4_OVF_vect)
TIMER5 CAPT	47	(TIMER5_CAPT_vect)
TIMER5 COMPA	48	(TIMER5_COMPA_vect)
TIMER5 COMPB	49	(TIMER5_COMPB_vect)
TIMER5 COMPC	50	(TIMER5_COMPC_vect)
TIMER5 OVF	51	(TIMER5_OVF_vect)
USART2 RX	52	(USART2_RX_vect)
USART2 UDRE	53	(USART2_UDRE_vect)
USART2 TX	54	(USART2_TX_vect)
USART3 RX	55	(USART3_RX_vect)
USART3 UDRE	56	(USART3_UDRE_vect)
USART3 TX	57	(USART3_TX_vect)

Timer OC (non-PWM) Interrupt Example 1

- 5 kHz square wave, 50% duty cycle, IO frequency of 16 MHz.

```
#include <avr/io.h>
#include <avr/interrupt.h>
#define NN 200 // delay count to be used in OC0A compare operation
int main (void)
{   DDRB |= 0x80;    // configure OC0A pin for output
    TCCR0A = 0x40;   // OC0A pin toggle on compare match
    TCCR0B = 0x02;   // normal mode, clock source set to clkI/O÷8
    TIMSK0 = 0x02;   // enable OC0A match interrupt
    TIFR0  = 0x02;   // clear OCF0A flag
    OCR0A  = TCNT0 + 10; // to avoid overflow
    sei();                    // enable interrupt globally
    while(1);                 // wait for interrupt to occur
}
ISR(TIMER1_COMPA_vect)
{   OCR0A += NN;              // start the next OC0A compare with NN as delay count
}
```

Timer OC (non-PWM) Interrupt Example 2

- 50 Hz square wave, 75% duty cycle, assume I/O freq = 16 MHz.
- Use interrupts to signal OC match

```
#include <avr\io.h>
#include <avr\interrupt.h>
#define hiCnt 30000        // delay count for high interval in a period
#define loCnt 10000        // delay count for low interval in a period
char HILO;                 // HILO=1 => ON time,  HILO=0 => OFF time
```

```
void main (void)
{   DDRB  |= 0x20;                 // configure OC1A pin for output
    TCCR1A = 0xC0;          // OC1A pin pull high on compare match
    TCCR1B = 0x02;          // normal mode, clock source set to clk_I/O/8
    TCCR1C|= 0x80;          // force OC1A pin to high
    TCCR1A = 0x40;          // toggle
    OCR1A  = TCNT1 + hiCnt; // start the first output compare operation
    HILO   = 0;
    TIMSK1|= 0x02;          // enable OC1A interrupt
    TIFR1  = 0x02;                  // clear the OCF1A flag
    sei();                  // enable interrupt globally
    while(1);               // wait
}
```

```
ISR(TIMER1_COMPA_vect)
{    if(HILO)                          //HILO = 1 => setup ON time
     {       OCR1A = OCR1A + hiCnt;
             HILO = 0;//next time it's OFF time
     }
     else                              //HILO = 0 => setup OFF time
     {       OCR1A = OCR1A + loCnt;
             HILO = 1;                 //next time it's OFF time
     }
}
```

Timer ICP Interrupt Example

- Measure the period of the signal connected to ICP1
- Solution: use polling method to detect edges, interrupts to detect overflow
  - Period = time_edge2 + OV*65536 – time_edge1
  - Make time_edge1 = 0 by clearing TCTNT

```
#include <avr\io.h>
#include <avr\interrupt.h>
unsigned char tovCnt;
unsigned long period;


ISR (TIMER4_OVF_vect)
{   tovCnt++;
}
```

```
int main (void)
{   tovCnt = 0;
    TCCR4A = 0;                     // configure to normal mode
    DDRL   = 0xFE;                  // configure ICP4 pin for input (PL0)
    TIFR4  = 0x2F;                  // clear all flags related to Timer
    TCCR4B = 0x42;                  // capture rising edge, presc = 8
    while(!(TIFR4 & 0x20));         // wait for the arrival of the 1st edge
    TCNT4  = 0;                     // set TCNT=0 clear edge1 time
    TIFR4  = 0x21;                  // clear ICF4 and TOV4 flags
    TIMSK4 = 1;                     // enable TOV interrupt
    sei();                          // enable interrupt globally
    while(!(TIFR4 & 0x20));         // wait for the arrival of the second rising edge
    TCCR4B = 0;                     // stop Timer
    period = (unsigned long)tovCnt * 65536 + (unsigned long)ICR4;
    while(1);
}
```

External Interrupts

- Interrupt caused by an event received on INT7:0
  - INT0:4 connected to Port D pins 0:4
  - INT5:7 connected to Port E pins 5:7
  - Events: rising, falling, low level
- Registers used to control EI:
  - EICRA, EICRB: EI control registers A, B
  - EIMSK: EI mask register
  - EIFR: EI flag register

EICRA, EICRB

7	6	5	4	3	2	1	0	
ISC31	ISC30	ISC21	ISC20	ISC11	ISC10	ISC01	ISC00	Reset value = 0x00
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

EICRA

7	6	5	4	3	2	1	0	
ISC71	ISC70	ISC61	ISC60	ISC51	ISC50	ISC41	ISC40	Reset value = 0x00
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

EICRB

- **ISC<u>n</u>1: ISC<u>n</u>0**: External interrupt 0-7 sense control (**<u>n</u>** = 0-7)
  - 00 = The low level of INT**<u>n</u>** generates an interrupt request.
  - 01 = Any edge of INT**<u>n</u>** generates an interrupt request.
  - 10 = The falling edge of INT**<u>n</u>** generates an interrupt request.
  - 11 = The rising edge of INT**<u>n</u>** generates an interrupt request.

EIMSK, EIFR

7	6	5	4	3	2	1	0	
INT7	INT6	INT5	INT4	INT3	INT2	INT1	INT0	Reset value = 0x00
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

EIMSK

7	6	5	4	3	2	1	0	
INTF7	INTF6	INTF5	INTF4	INTF3	INTF2	INTF1	INTF0	Reset value = 0x00
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

EIFR

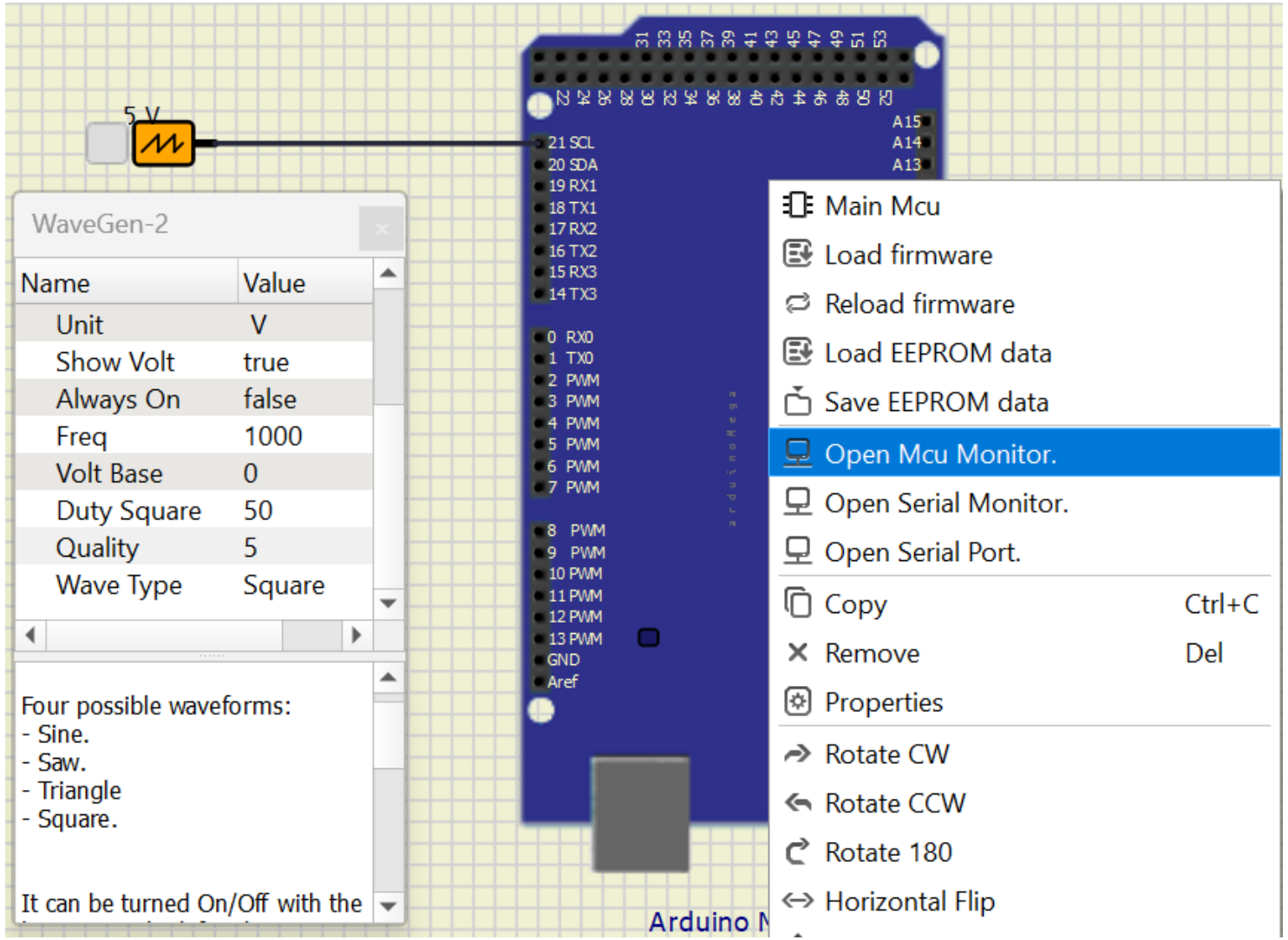
- **INT<u>n</u>**: External interrupt request n enable (**<u>n</u>** = 0, …, 7)
  - 0 = External interrupt **<u>n</u>** is disabled.
  - 1 = External interrupt **<u>n</u>** is enabled.
- **INTF<u>n</u>**: External interrupt flag (**<u>n</u>** = 0, …, 7)
  - 0 = External interrupt **<u>n</u>** condition did not occur since last time this flag was cleared.
  - 1 = External interrupt **<u>n</u>** condition has occurred.

Example: External INT Interrupt

- Measure the frequency of a square signal connected to external INT pins.
- Solution
  - Connect signal to external INT0
  - INT0 for rising edges
    - Count rising edges
  - Configure INT0
    - DDRD = 0 (input, INT0 is PD0)
    - EICRA = 0b00000011 (INT0: detect rising edges)
    - EIMSK = 1 (INT0:1 enable)

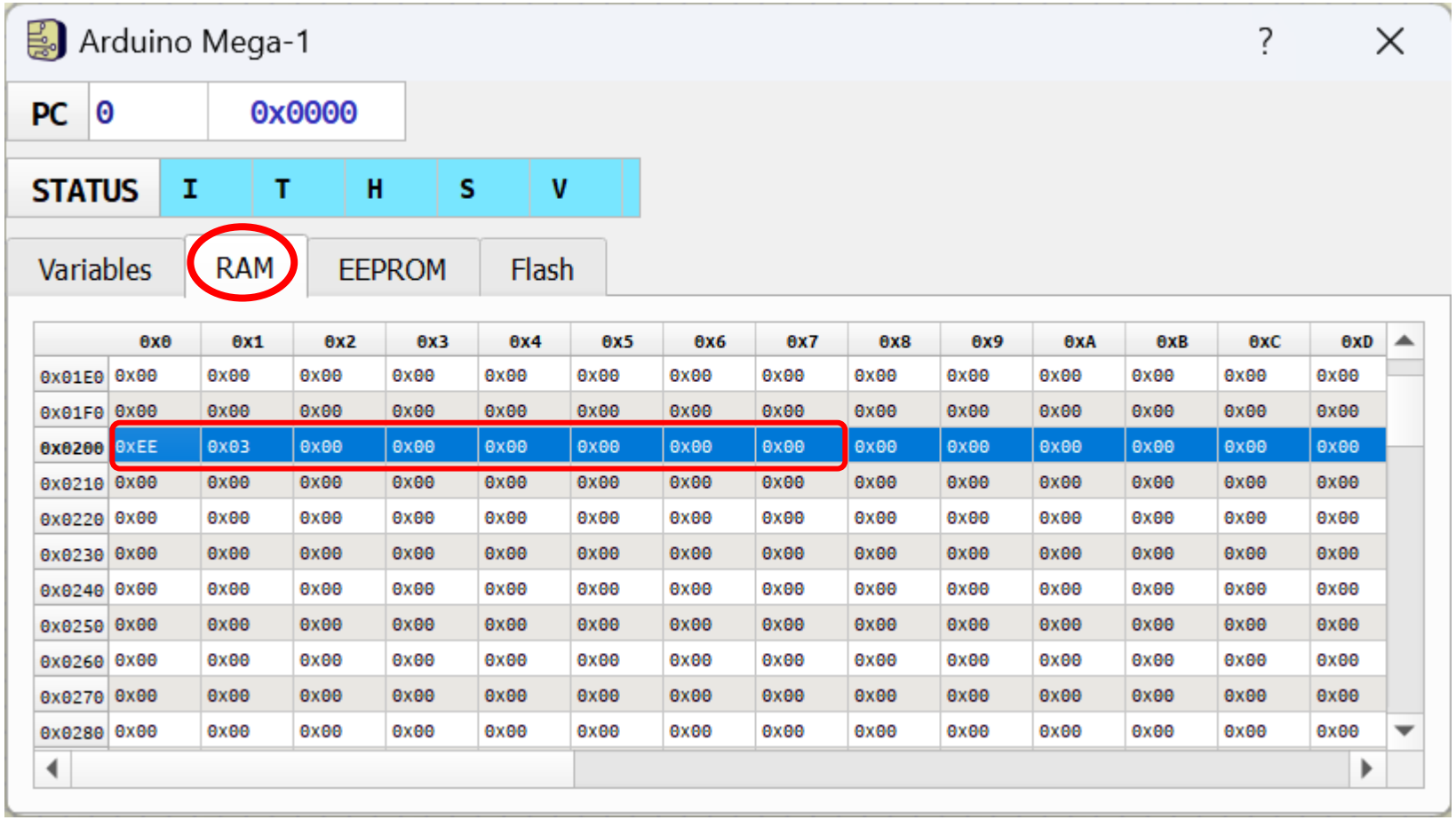
```
#include <avr\io.h>
#include <avr\interrupt.h>
unsigned long long int freq=0;        // 64 bit variable

int main (void)
{    DDRD &= ~(0x01);           // PD0 Ext INT0 input
     EICRA = 0b00000011;        // INT0 rising edges(11)
     EIMSK = 1;                 // INT0 enable only
     sei();                     // global INT enable
     _delay_1ms(1000);          // wait 1 sec
     cli();                     // disable INT globally
     while(1);
}

ISR (INT0_vect)
{    freq++; }
```

External INT Example Simulation IDE



Simulation IDE: MCU Monitor

- Click on MCU Monitor
- Click on RAM tab
- Scroll to address 0x0200
- Data in little endian format
  - LSB first
  - 0x000000000000EE03

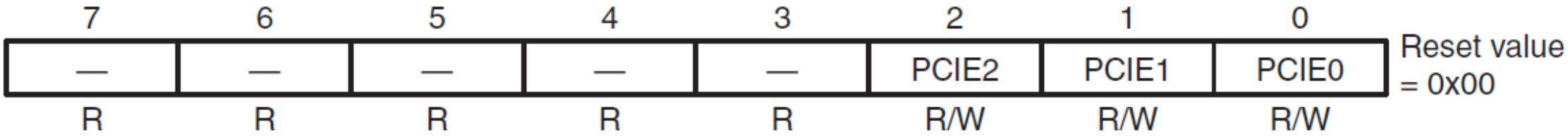


	0x0	0x1	0x2	0x3	0x4	0x5	0x6	0x7	0x8	0x9	0xA	0xB	0xC	0xD
0x01E0	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0x01F0	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0x0200	0xEE	0x03	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0x0210	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0x0220	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0x0230	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0x0240	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0x0250	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0x0260	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0x0270	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0x0280	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00

Pin Change (PC)

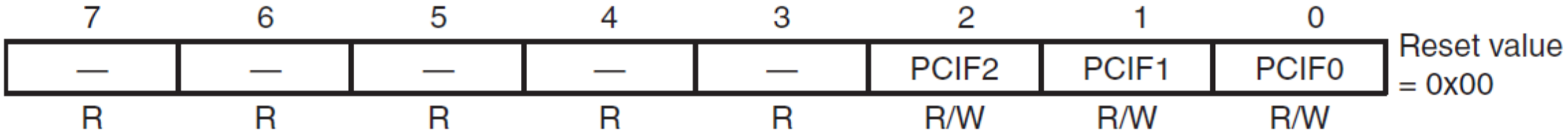
- 24 PC pins that generate at interrupt when one or more pin(s) toggle
  - Toggle aka change from 1 to 0, or vice-versa
  - PC pin 0:7 are PB0:7
  - PC pin 8:15 are PE0, PJ0:6
    - PJ2:6 not connected on the board
  - PC pin 16:23 are PK0:7
  - PCINT0 interrupt: when any PC pins 7:0 toggle
  - PCINT1 interrupt: when any PC pins 15:8 toggle
  - PCINT2 interrupt: when any PC pins 23:16 toggle
- Registers to control PC:
  - PCMSK0,1,2: PC masks register 0,1,2
  - PCICR: PC interrupt control reg
  - PCIFR: PC interrupt flags reg

PCICR



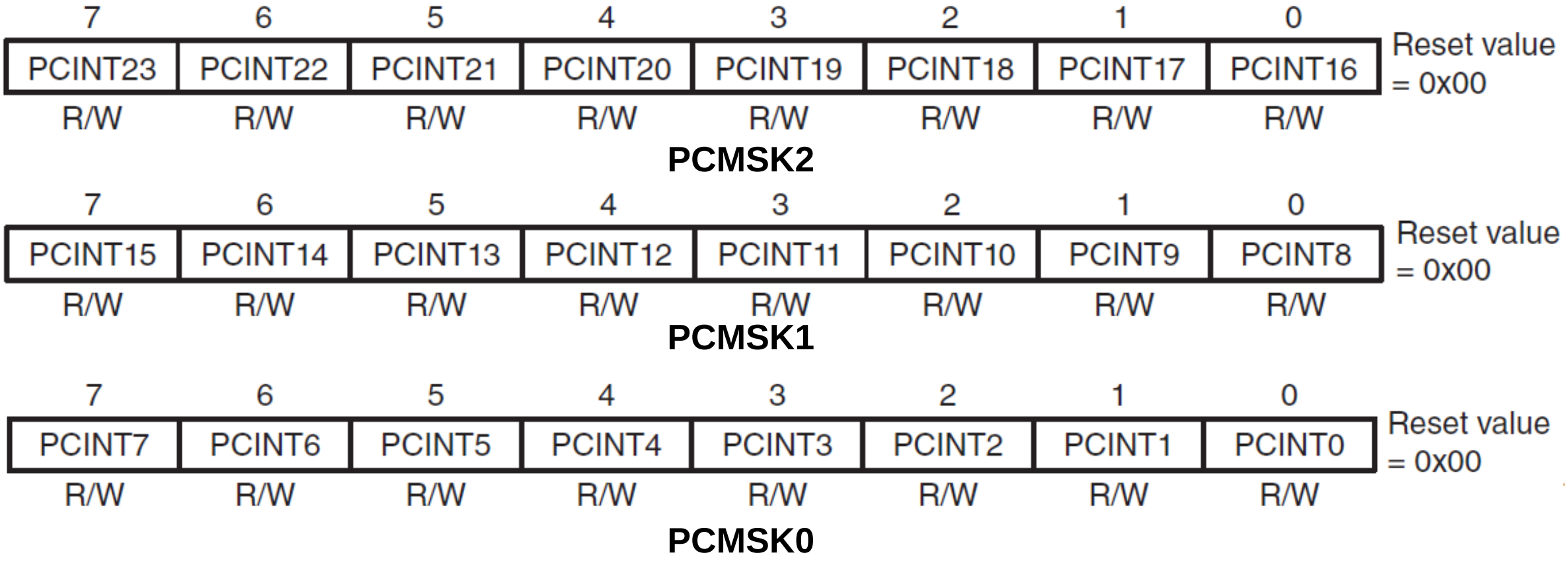
- **PCIE2**: PC interrupt enable 2
  - 0 = PC interrupt 2 disabled
  - 1 = PC interrupt 2 enabled (detects change on enabled PC pins 23:16)
- **PCIE1**: PC interrupt enable 1
  - 0 = PC interrupt 1 disabled
  - 1 = PC interrupt 1 enabled (detects change on enabled PC pins 15:8)
- **PCIE0**: PC interrupt enable 0
  - 0 = PC interrupt 1 disabled
  - 1 = PC interrupt 1 enabled (detects change on enabled PC pins 7:0)

PCIFR



- **PCIF2**: Pin change interrupt flag 2
  - 0 = No change on PC pins 23:16.
  - 1 = At least one change on PC pins 23:16.
- **PCIF1**: Pin change interrupt flag 1
  - 0 = No change on PC pins 15:8.
  - 1 = At least one change on PC pins 15:8.
- **PCIF0**: Pin change interrupt flag 0
  - 0 = No change on PC pins 23:16.
  - 1 = At least one change on PC pins 7:0.

PCMSK0,1,2



- **PCINT<u>n</u>**: Pin change enable mask (**<u>n</u>** = 23...0)
  - 0 = Disable interrupt on PC pin **<u>n</u>**
  - 1 = Enable interrupt on PC pin **<u>n</u>** (Must be set with PCICR)

PC Example

- Measure the frequency of a square signal using the PCINT.
- Solution:
  - Connect signal to PB0 (PC pin0)
  - Setup: PORTB, PCMSK0, PCINT0, PCICR
  - Create a 1 sec delay in OC3.
  - Count the number of toggles
  - Freq = number of toggles /2

```
#include <avr\io.h>
#include <util\delay.h>
#include <avr\interrupt.h>
long int edges=0, freq;
void main (void)
{   DDRB  &= !(0x01);              // configure PB0 (PC pin0) for input
    PCICR  = 1;                 // PCINT0 enable for PC pins 0:7
    PCMSK0 = 1;                 // PC pin0 interrupt enable
    sei();                      // enable interrupt globally
    _delayby_1ms(1000);         // wait for 1 second
    cli();                      // disable interrupts
    freq = cnt/2;
    while(1);
}
ISR (PCINT0_vect)
{   edges++;       }
```

Timer Example: Measure Frequency (non-ICP)

- Measure frequency of an unknown square signal connected to the T1 pin, assume 16 MHz IO clock. Don't use the ICP.
- Solution:
  - Set signal T1 as source for Timer 1
  - Set Timer 3 to count delay for 1sec, using CPU cock source
  - Count T1 clock cycles in Timer 1 to get frequency

```
#include <avr\io.h>
#include <util\delay.h>
#include <avr\interrupt.h>
unsigned char tovCnt;
unsigned long freq;

ISR (TIMER1_OVF_vect)
{   tovCnt++;
}
```

```
void main (void)
{   tovCnt = 0;
    DDRD &= ~0x40;          // configure T1 for input
    TCCR1A = 0;             // configure Timer 1 to normal mode, no output
    TCCR1B = 0;             // stop Timer 1
    TCNT1=0
    TIFR1 = 1;              // clear TOV1 flag
    TIMSK1 = ;              // enable TOV1 interrupt
    TCCR1B = 0x07;          // enable Timer 1 to count using T1 input as clock source
    sei();                  // enable interrupt globally
    _delayby_1ms(1000);     // wait for 1 second
    TCCR1B = 0;             // stop Timer 1
    freq = (long)tovCnt * 65536 + (long)TCNT1;
    while(1);
}
```

Example: External INT Interrupt

- Measure the duty cycle of a square signal connected to external INT pins.
- Solution
  - Connect signal to external INT0 and INT1
  - INT0 for rising edges (er), INT1 for falling edges (ef):
    - Detect 3 edges: 2 rising edges (period), 1 falling edge
    - Duty = (ON time)/period = (ef1 – er0)/(er2 – er0)
  - Configure INT0, INT1
    - DDRD = 3 (INT0:1: PD0:1)
    - EICRA = 0b00001011 (INT1: falling edges (10), INT0: detect rising edges (11))
    - EIMSK = 3 (INT0:1 enable)

```
#include <avr\io.h>
#include <avr\interrupt.h>
unsigned char tovCnt;
unsigned long int er0=0,er2=0,ef1=0;

int main (void)
{    DDRD &= ~(0x03);          // PD0:1 Ext INT0:1 input
     EICRA = 0b00001011;       // INT0 rising edges(11), INT1 falling (10)
     EIMSK = 1;                // INT0 enable only
     sei();                    // global INT enable
     while(1);
}

ISR((INT0_vect)
{     EIMSK = 2;               //enable INT1 only
```

PC Example 2

- Measure the frequency of a square signal using the PCINT. Use another signal to create a delay.
- Solution:
  - Connect input signal to PB0 (PC pin0)
  - Setup: PORTB, PCMSK0, PCINT0, PCICR
  - Connect a 0.5 Hz signal to PE0 to simulate a 2sec delay.
    - Period = 2s: rises in 1sec, falls in 1 sec
  - Count the number of toggles
  - Calculate frequency every edge
    - Freq = number of toggles

```
#include <avr\io.h>
#include <avr\interrupt.h>
long int edges=0,freq;

int main (void)
{   DDRB  &= 0xFE; // configure PB0 (PC pin0) for input
    PCICR  = 3;    // enable PCINT0 for PC pins 0:7, PCINT1 for pins 8:15
    PCMSK0 = 1;    // PC pin0 interrupt enable
    PCMSK1 = 1;    // PC pin8 interrupt
    sei();
    while(1);
}
ISR (PCINT0_vect)
{   edges++;       }
ISR (PCINT1_vect)
{   freq = edges;
    edges= 0;      }
```