CprE 288 – Introduction to Embedded Systems (Timers/Input Capture)

Instructors:

Dr. Zhao Zhang (Sections A, B, C, D, E)

Dr. Phillip Jones (Sections F, G, J)

Overview of Today's Lecture

- Announcements
- Input Capture Review

Announcements

- Homework 6 is due on Thursday
- Exam 1 grades online, returned in Lab Section

INPUT CAPTURE

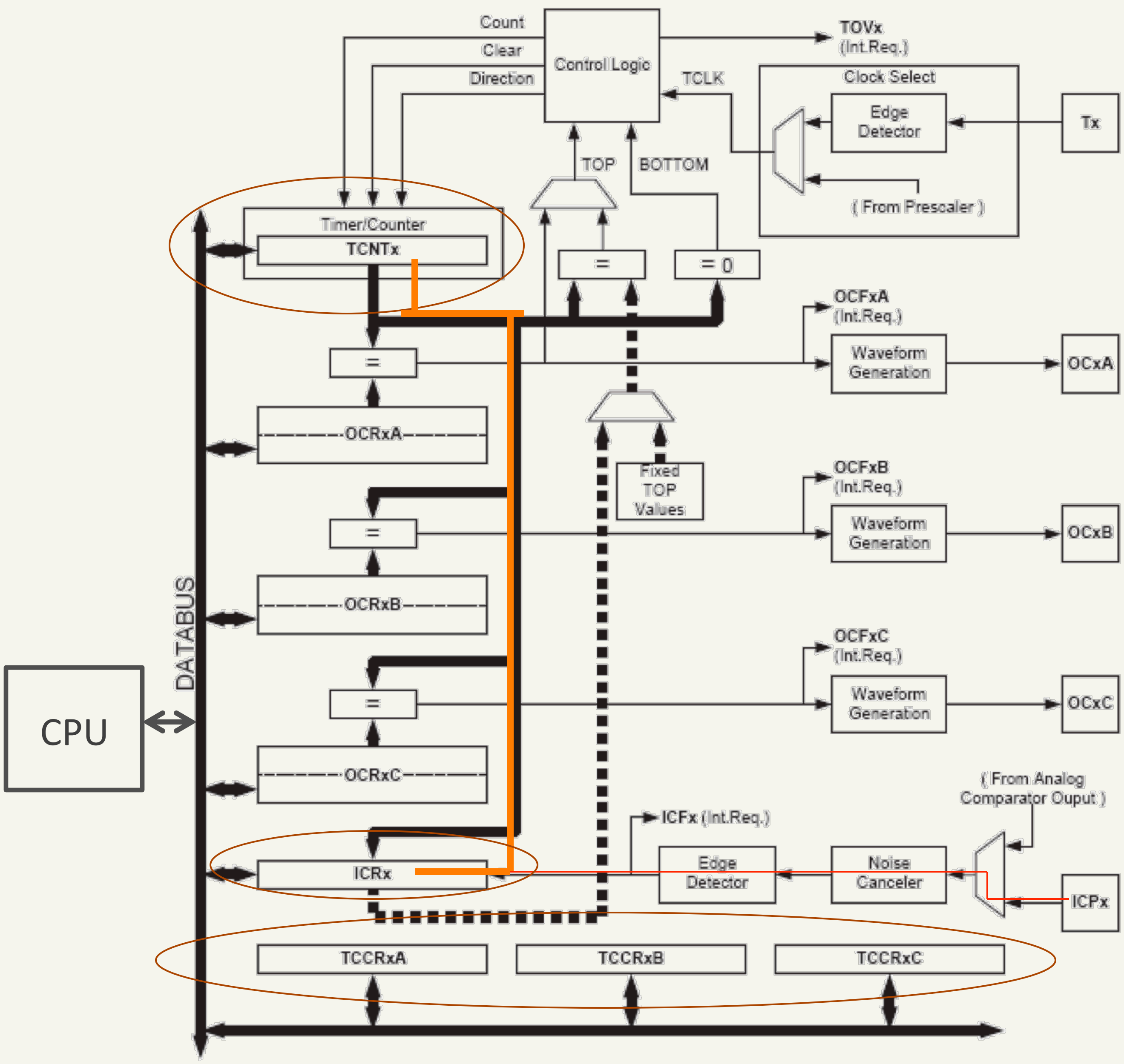
Input Capture

Capture the times of events

Many applications in microcontroller applications:

- Measure rotation rate
- Remote control
- Sonar devices
- Communications

Generally, any input that can be treated as a series of events, where the precise measure of event times is important



ATmega128 16-bit timer/counter

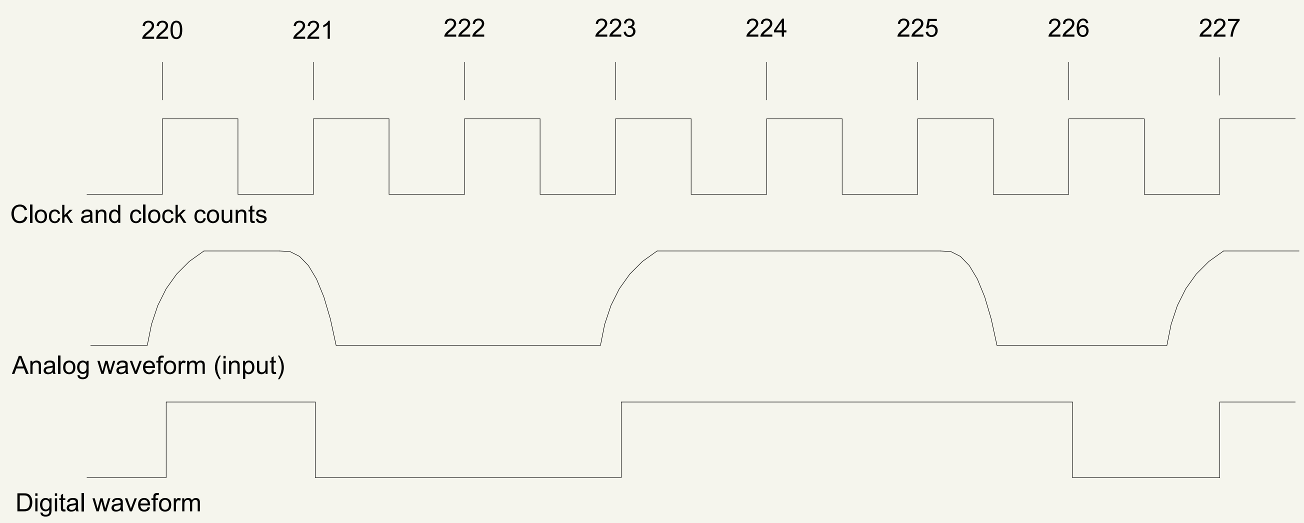
TCNTx: Timer/Counter
ICRx: Input Capture Reg
ICPx: Input Capture Pin

x is 1 or 3
for timer/counter 1
and timer/counter 3

Input Capture

An event is a transition of binary signal

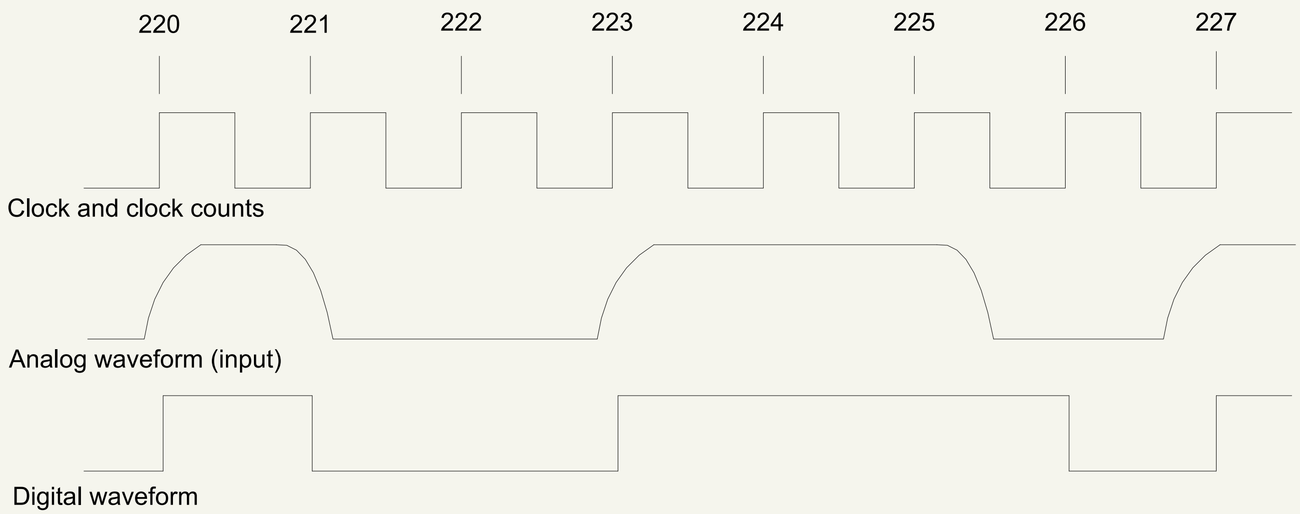
Example: How many events make up the following **waveform?**



Input Capture

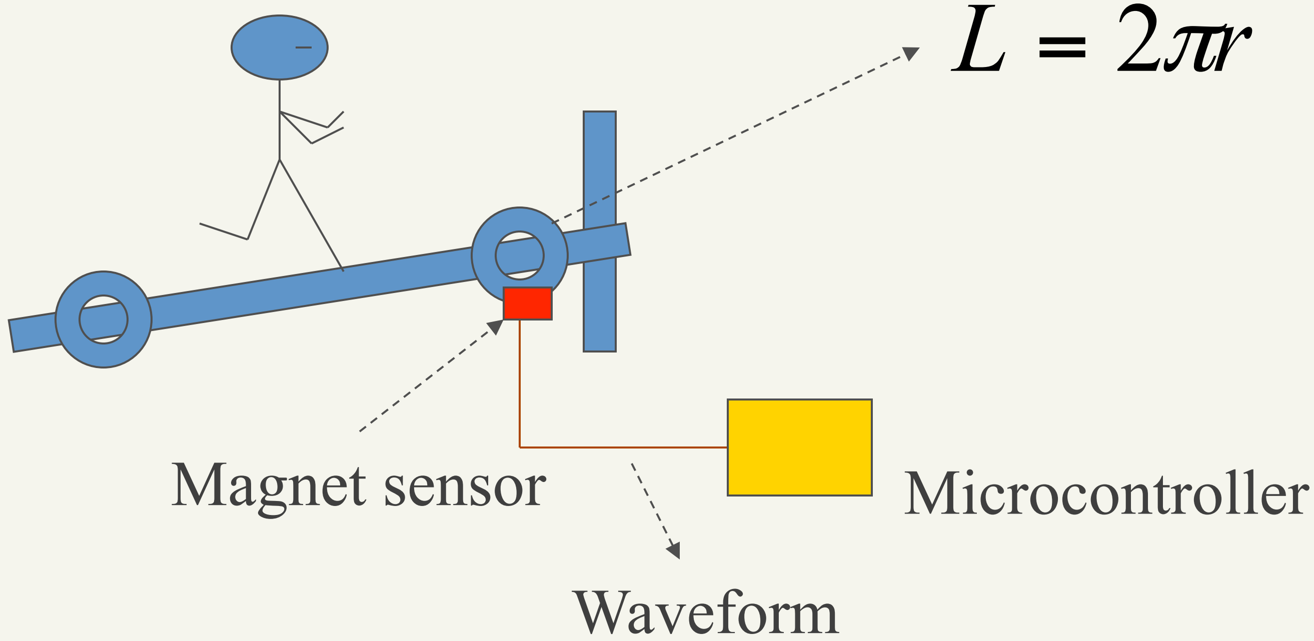
An input digitalized and then times captured

Example: The input is understood as events occurring at the following times: 220, 221, 223, 226, and 227 with initial state as low



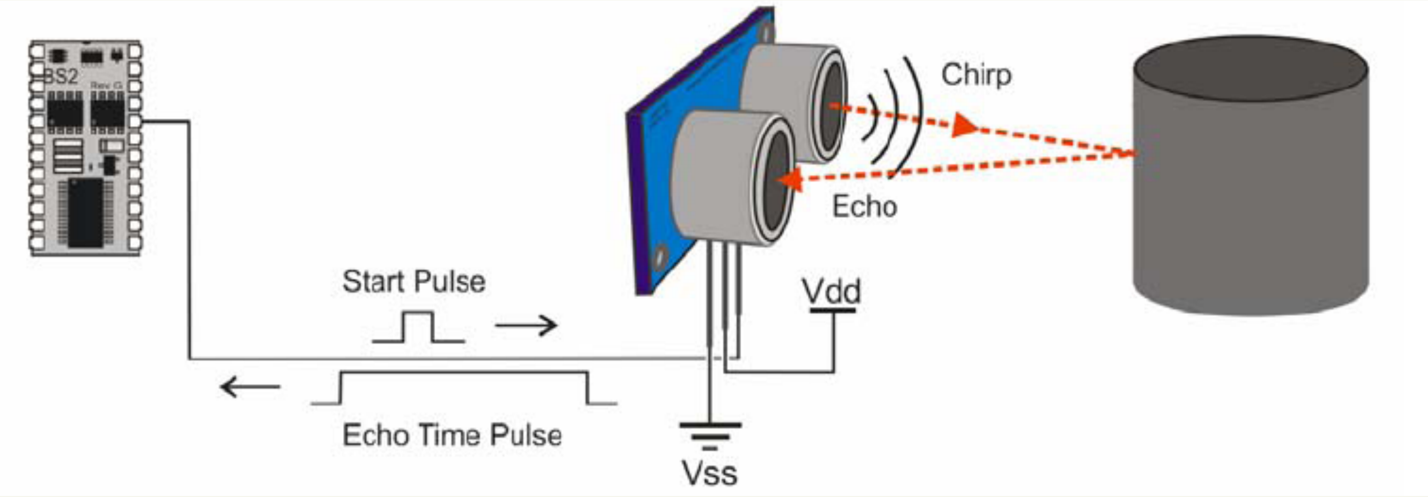
Application: Speedometer

How to detect the speed of a treadmill?



Application: Sonar Device

Ping))) sensor: ultrasound distance detection device

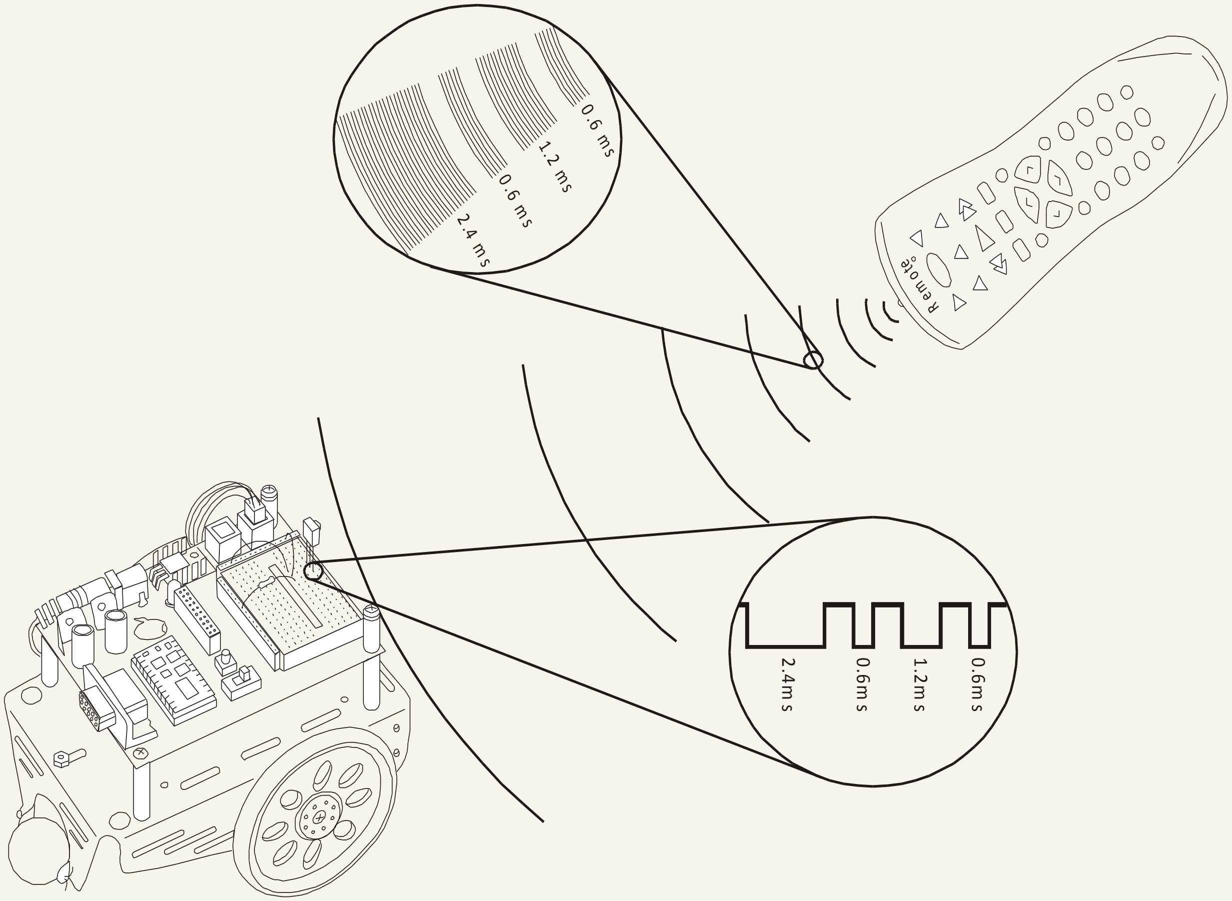


Application: Sonar Device

PING Sensor Datasheet:

- http://class.ece.iastate.edu/cpre288/resources/docs/28015-PING-v1.3.pdf

Application: Remote Control



Input Capture: Design Principle

Time is important!

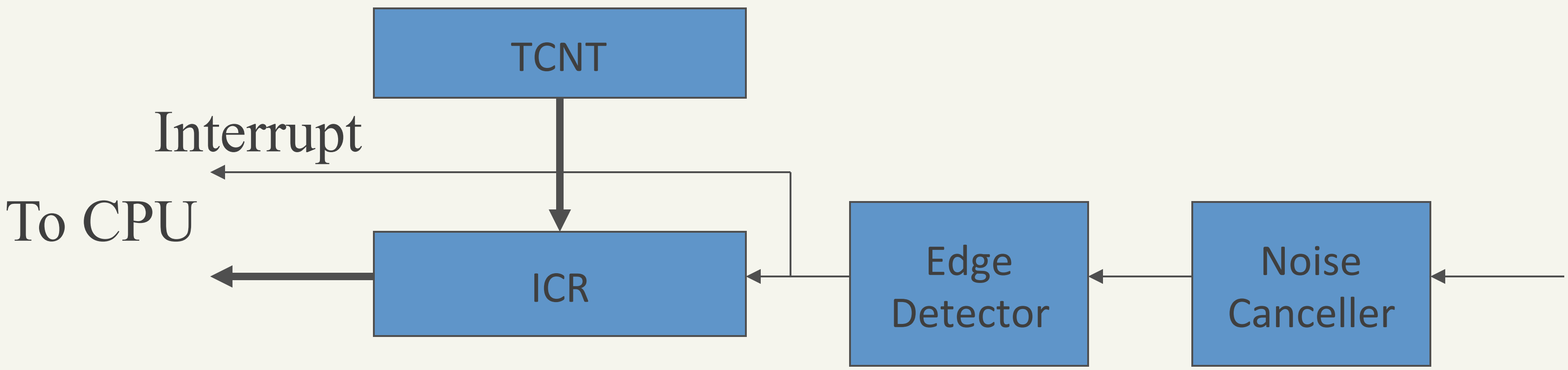
How could a microcontroller capture the time of an event, assuming a clock count can be read?

- Keep polling the input pin?
- Use an interrupt?
- ???

Precise timing is needed!

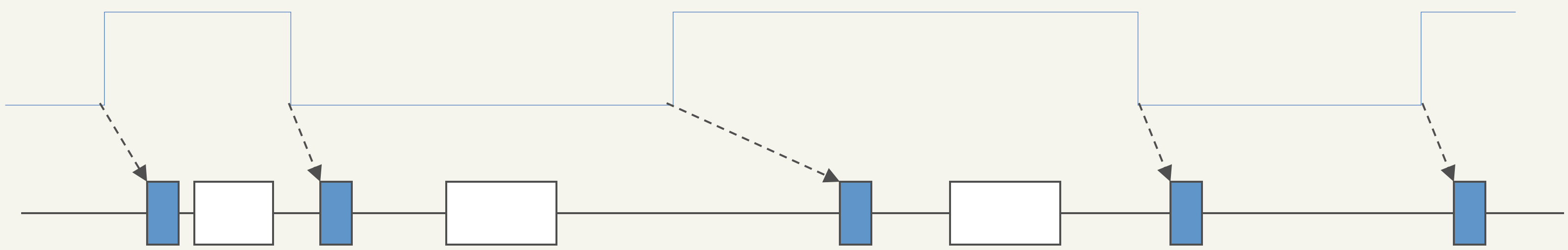
Input Capture: Design Principle

Time value (clock count) is captured first then read by the CPU



TCNT: Timer/Counter
ICR: Input Capture Register

Input Capture: Design Principle



Interrupt

CPU Interrupt processing

CPU Foreground computation

Input Capture: Design Principle

What happens in hardware and software when and after an event occurs

- The event's time is *captured* in an ICR (input capture register)
- An interrupt is raised to the CPU
- CPU executes the input capture ISR, which reads the ICR and completes the related processing

The captured time is *precise* because it's captured immediately when the event occurs

The ISR should read the ICR and complete its processing fast enough to avoid loss of events

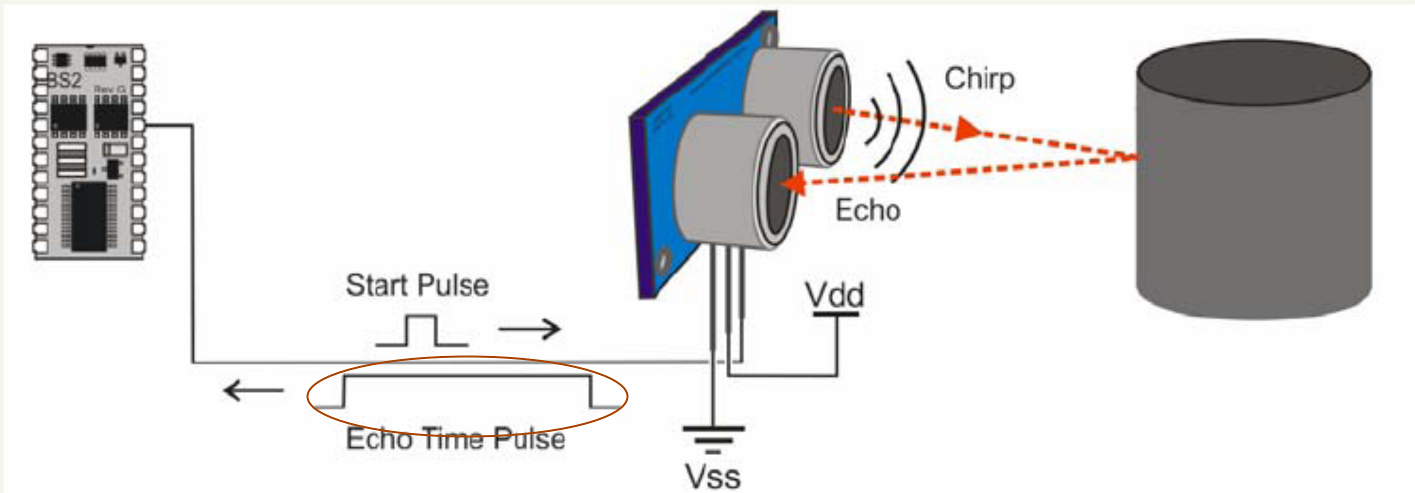
Input Capture: Design Principle

How to program the interrupt handler to do

- Count the number of pulses
- Calculate pulse width
- Decode IR signals
- And many other functions …

```
ISR (TIMER1_CAPT_vect)
{
   // YOUR PROCESSING
}
```

Sonar Principle



Sound Speed in Lab Temperature: About 340m/s
Pulse width proportional to round-trip distance

* Temperature affects sound speed

Sonar Principle

Assume 62.5KHz Input Capture clock
1ms <=> 62.5 clocks <=> 34cm

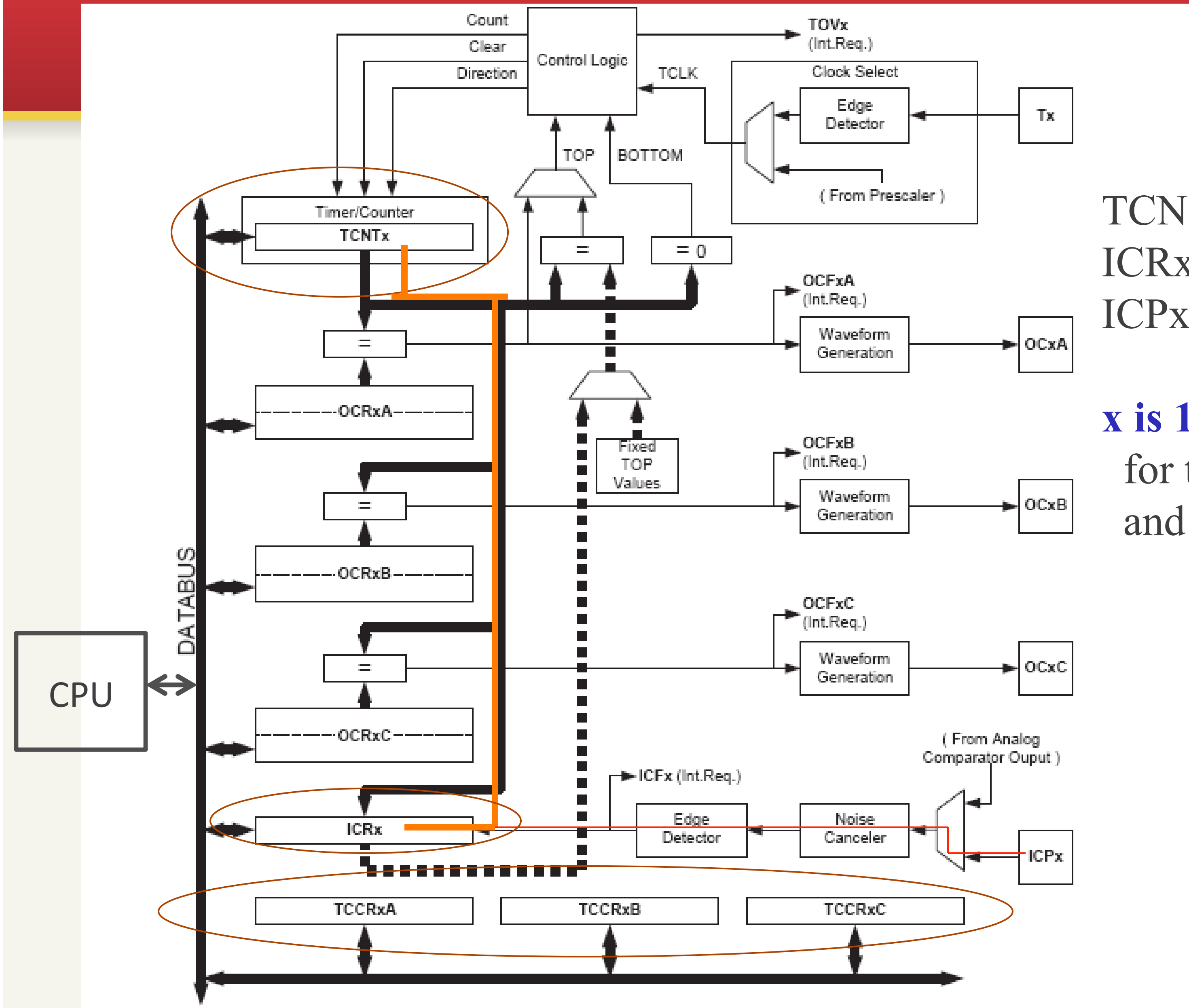
Time Diff.	Clock Count	One-way Distance
2ms	125	0.34m
4ms	250	0.68m

How to capture the times of rising edge and falling edge?

ATmega128 16-bit Timer/Counter as Input Capture Unit

ATMega128 has two, multi-purpose 16-bit timer/counter units

- One input capture unit (IC)
- Three independent output compare units (OC)
- Pulse width modulation output (PWM)
- Frequency generator
- And other features



TCNTx: Timer/Counter
ICRx: Input Capture Reg
ICPx: Input Capture Pin

x is 1 or 3
for timer/counter 1
and timer/counter 3

ATmega128 16-bit timer/counter

ATmega128 16-bit Timer/Counter as Input Capture Unit

When an edge is detected at input capture pin, current TCNTx value is captured (saved) into ICRx

Time is captured **immediately** (when an event happens) and read by the CPU later

Use Input Capture: Example

```
int last_event_time;

ISR (TIMER1_CAPT_vect)
{
   int event_time = ICR1;        // read current event time

   // YOUR PROCESSING CODE
}
```

Notes:

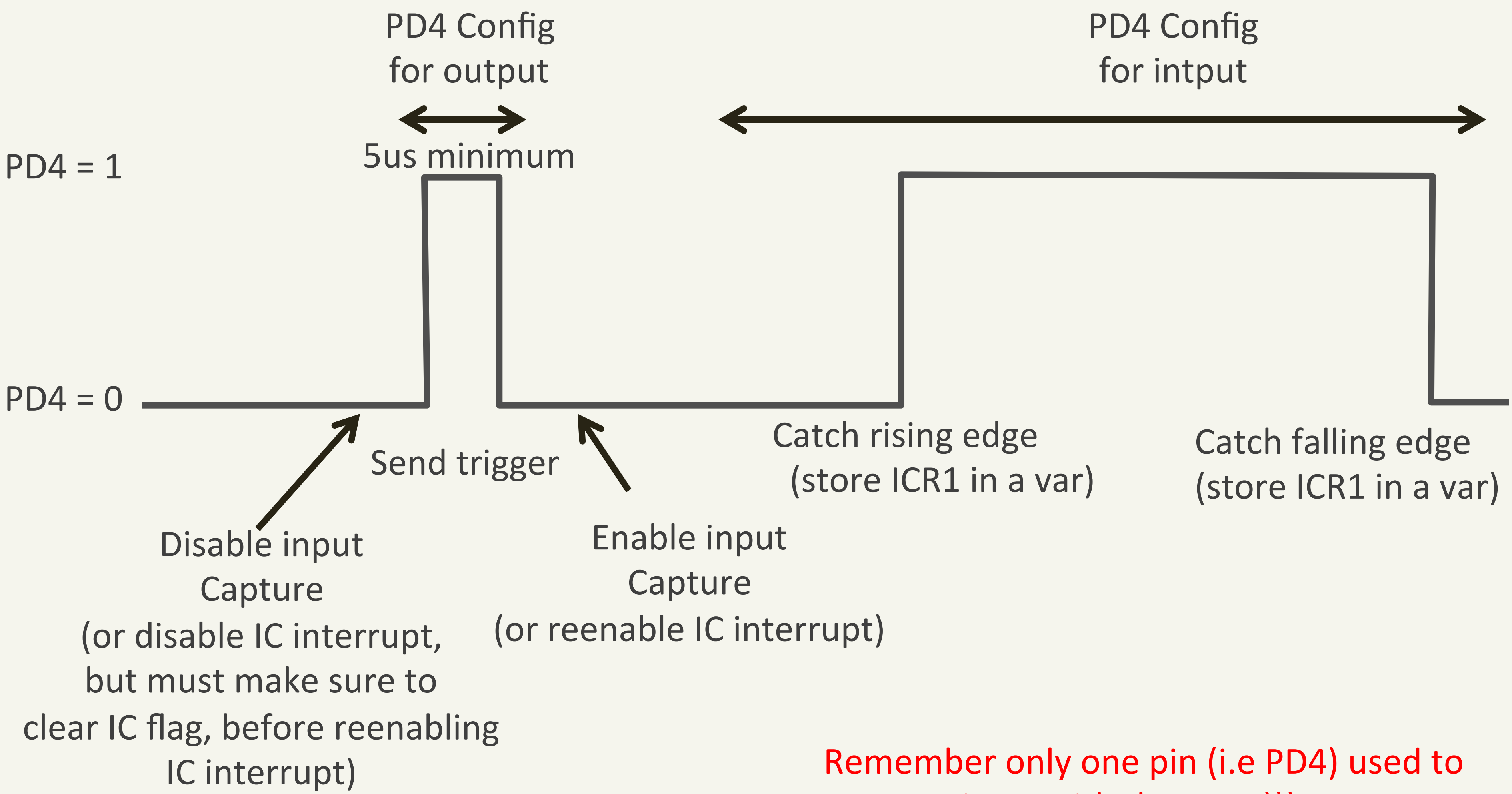
- Use Interrupt to process input capture events
- Read captured time from ICR*x* (*x* is 1 or 3)

Lab 7 General Idea of Programming

General idea:

- Configure Timer/Counter 1 for input capture
- Generate a pulse to activate the PING))) sensor
- Capture the time of rising edge event
- Capture the time of falling edge event
- Calculate time difference and then distance to any object

Lab 7 General Idea of Programming



Remember only one pin (i.e PD4) used to communicate with the PING))) sensor

16-bit Timer/Counter Programming Interface

TCCRnA: Control Register A

TCCRnB: Control Register B

TCCRnC: Control Register C

ICRn: Input Capture Register

TIMSK: Timer/Counter Interrupt Mask

ETIMSK: Extended Timer/Counter Interrupt Mask

Three channels to control: A, B, and C

Note: *Use **Timer/Counter 3** in the following discussions; Lab 7 uses **Timer/Counter 1***

16-bit Timer/Counter Programming Interface

Inside those TCCRs:

- **COM 1:0** (A): Compare Output Mode
- **WGM 3:0** (A, B): Waveform Generator Mode
- **ICNC** (B): Input Capture Noise Canceller
- **ICES** (B): Input Capture Edge Select
- **CS 2:0** (B): Clock Select
- **FOC 2:0** (B): Force Output Compare

7	6	5	4	3	2	1	0	
COM3A1	COM3A0	COM3B1	COM3B0	COM3C1	COM3C0	WGM31	WGM30	TCCR3A
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

COM: Compare Output Mode

We don't care COM bits at this moment – set them to zero in lab 7

WGM: Waveform Generator Mode

To select Timer/Counter function. Four bits in total (**WGM33** and **WGM32** in TCCR3B)

To use Input Capture:

WGM33 = 0, WGM32 = 0, WGM31 = 0, WGM30 = 0

7	6	5	4	3	2	1	0	
ICNC3	ICES3	–	WGM33	WGM32	CS32	CS31	CS30	TCCR3B
R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	

ICNC3: Input Capture **Noise Canceller**, requires four-cycle duration for an event; **use it in lab 7**

ICES3: Input Capture **Edge Select** – Which edge will trigger the capture? 0 for falling edge, 1 for rising edge

WGM32, WGM32: See previous slide

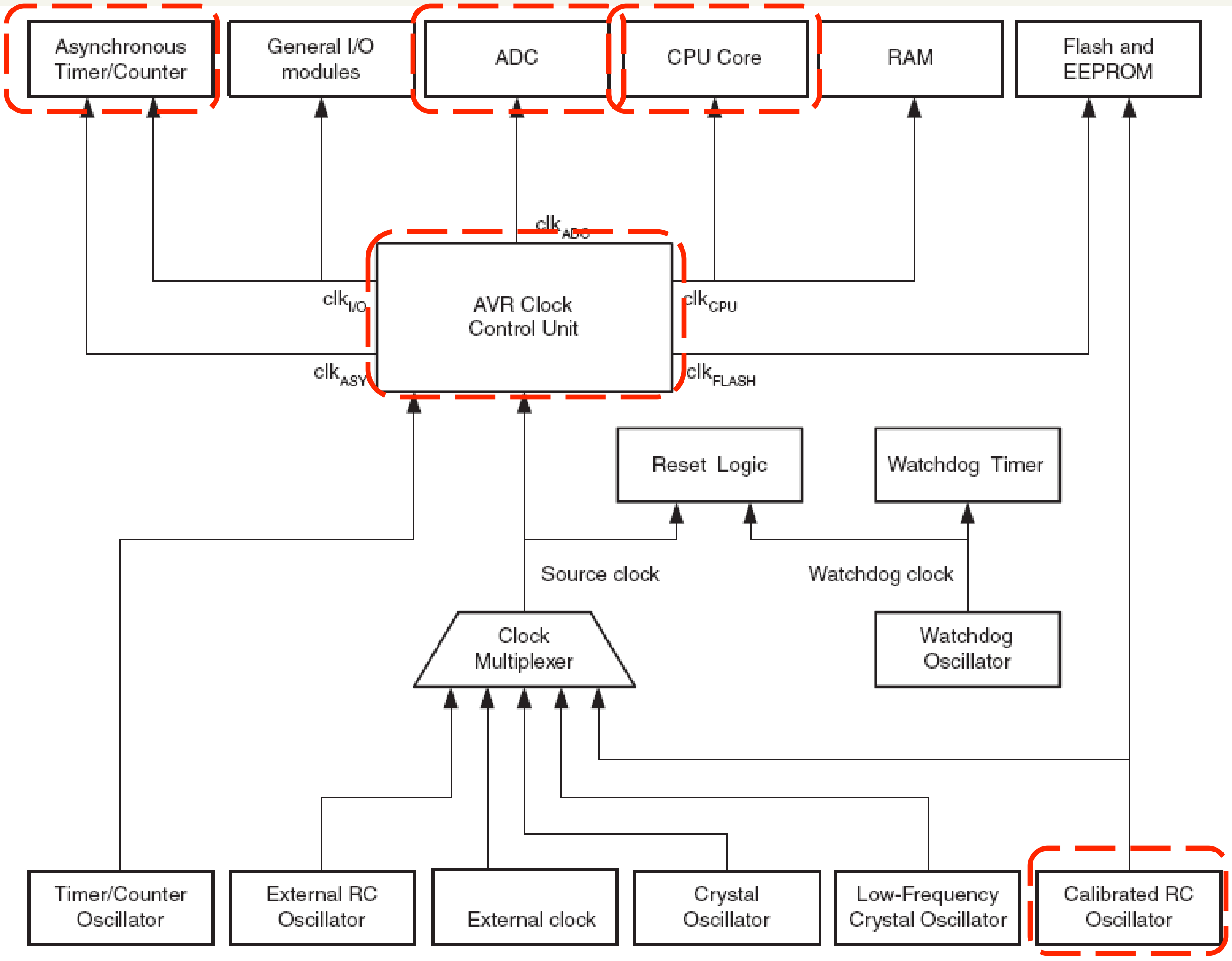
7	6	5	4	3	2	1	0	
ICNC3	ICES3	–	WGM33	WGM32	CS32	CS31	CS30	TCCR3B
R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	

CS3x: **Clock Select** bits

Table in ATmega128 User Guide, page 137

CSn2	CSn1	CSn0	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 Tn pin. Clock on falling edge
1	1	1	External clock source on Tn pin. Clock on rising edge

ATmega128 Clock Sources



ATmega128 User Guide, page 36

7	6	5	4	3	2	1	0	
FOC3A	FOC3B	FOC3C	–	–	–	–	–	TCCR3C
W	W	W	R	R	R	R	R	

FOC: Force output compare on channel A, B or C

Write 0s to those bits in lab 7 or don't write it; output compare is not used

We will see those bits later

7	6	5	4	3	2	1	0	
OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	TIMSK
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

TICIE1: Timer/Counter 1, Input Capture Interrupt Enable – Write 1 to it to use interrupt

TOIE1: Timer/Counter1, Overflow Interrupt Enable – If set to 1, interrupt is raised when Timer1/Counter 1 value (TCCN1 value) is overflowed

*Note: Use a **sufficient large prescaler value** to avoid overflow in lab 7*

The other bits are for output compare – we will see them again

7	6	5	4	3	2	1	0	
–	–	TICIE3	OCIE3A	OCIE3B	TOIE3	OCIE3C	OCIE1C	ETIMSK
R	R	R/W	R/W	R/W	R/W	R/W	R/W	

ETIMASK is for Timer/Counter 3

TICIE3: Timer/Counter 3, Input Capture Interrupt Enable – Write 1 to it to use interrupt

TOIE3: Timer/Counter 3, Overflow Interrupt Enable – If set to 1, interrupt is raised when Timer1/Counter 3 value (TCCN3 value) is overflowed

Configure Timer/Counter 1 for Lab 7

TCCR1A: WGM bits = 0

TCCR1B: Enable interrupt, Choose right Edge Select, WGM bits = 0, Choose good Clock Select

TCCR1C: Keep all bit cleared

TIMSK: Enable Timer/Counter 1 Input Capture Interrupt

Port D pin 4 (PD4) – It's Timer1/Counter1's IC pin, and connects to the input/output pin of the PING sensor

IC Programming Example

```
volatile enum {LOW, HIGH, DONE} state;
volatile unsigned rising_time;        // start time of the return pulse
volatile unsigned falling_time;       // end time of the return pulse

/* start and read the ping sensor for once, return distance in mm */
unsigned ping_read()
{
    …
}

/* ping sensor related to ISR */
ISR (TIMER1_CAPT_vect)
{
    …
}
```

Note 1: This code does not work for Lab 7 as it is.
Note 2: Does <u>not</u> follow timing example of slide 29.

```
/* send out a pulse on PD4 */
void send_pulse()
{
  DDRD |= 0x10;          // set PD4 as output
  PORTD |= 0x10;         // set PD4 to high
  wait_ms(1);            // wait
  PORTD &= 0xEF;         // set PD4 to low
  DDRD  &= 0xEF;         // set PD4 as input
}

/* convert time in clock counts to single-trip distance in mm */
unsigned time2dist(unsigned time)
{
  …
}
```

```
unsigned ping_read()
{
  send_pulse();                    // send the starting pulse to PING

  // TODO get time of the rising edge of the pulse

  // TODO get time of the falling edge of the pulse

  // Calculate the width of the pulse; convert to centimeters

}
```

ADD-ON SLIDES

IC Programming Example

- Treadmill

Assume

- The sensor input is connected to Timer/Counter 1 Input Capture Pin (ICP1)
- L is the circumference (length of circle) of the wheel

IC Programming Example

```
volatile unsigned last_time = 0;
volatile unsigned current_time = 0;
volatile int update_flag = 0;

// ISR: Record the current event time
ISR (TIMER1_CAPT_vect)
{
   last_time = current_time;
   current_time = ICR1;
   update_flag = 1;
}
```

Recall: We have to declare "volatile" for global variables changed by ISRs, otherwise a normal function may not see the changes

Critical Section

```
void print_speed() {
   if (!update_flag)  // no update? then return
      return;
   cli();                  // disable interrupt
   unsigned time_diff = current_time – last_time;
   update_flag = 0;
   sei();                  // enable interrupt
   … // calculate the speed and show it on LCD
}
```

- In this case, we want to prevent ISR execution when this function reads `current_time`, `last_time` and change `update_flag`.
- Otherwise, the function may occasionally print strange result: E.g. what happens if an IC interrupt happens after the function reads `current_time` and before it reads `last_time`?

Critical Section

```
cli();   // disable interrupt
time_diff = current_time – last_time;
update_flag = 0;
sei();   // enable interrupt
```

- This is a form of **critical section**: The execution of this code should not be interfered. ISRs should not be allowed to read/write those shared variables when this code is executing.
- We want to make critical section as short as possible, because it blocks ISR execution
  - Move computation-intensive code outside of critical section
  - No floating point calculation, printing LCD, big array access, etc. in a critical section

Critical Section

```
cli();   // disable interrupt
time_diff = current_time – last_time;
update_flag = 0;
sei();   // enable interrupt
```

- What are `cli()` and `sei()`?
  - There is a global interrupt flag in AVR microcontrollers
  - `cli()`: Clear interrupt (flag), or disable interrupt
  - `sei()`: Set interrupt (flag), or enable interrupt
- More about `cli()` and `sei()`
  - The are declared in `<avr/interrupt.h>`
  - Each is a single machine instruction, not really a function

Critical Section

```
cli();    // disable interrupt
time_diff = current_time – last_time;
update_flag = 0;
sei();    // enable interrupt
```

With this efficient code, does the MCU lose an IC event when it happens right in the critical section?

	7	6	5	4	3	2	1	0
TIFR	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
	0	0	0	0	0	0	0	0

The interrupt status is buffered in a special register, TIFR (Interrupt Flag Register for Timer/Counter 1), in its ICF1 bit.

The bit is cleared automatically but not until the ISR gets executed.

Polling- vs. Interrupt-Based Programming

Polling: Your code keeps checking I/O events

For Input Capture, your code may check ICF flag

```
while ((TIFR & _BV(ICF1)) == 0)
    {}
print_speed();
TIFR |= _BV(ICF1);     // clear ICF1
```

Note: ICF1 is cleared by writing 1 to it. (Always check the datasheet for such details.)

Polling- vs. Interrupt-Based Programming

Why polling?

- Program control flow looks simple
- Interrupts have overheads added to the processing delay
- Not every programmer likes writing ISRs

Why NOT polling?

- The CPU cannot do anything else
- The CPU cannot sleep to save power
- Using ISRs can simplify the control structure of the main program

TCNT Overflow

Are we concerned with TCNT overflow in the calculation?

```
time_diff = current_time – last_time;
```

What happens if `current_time` is *less* than `last_time`?

TCNT Overflow: Change from 0xFFFF to 0x0000

Consider having two capture events at TCNT1 = 0xFFFF and TCNT1 = 0x0005, respectively, with 6 cycles in between

`last_time` = 0xFFFF

`current_time` = 0x0005

What will be `current_time` – `last_time`?

Hardware adder for 2's complement handles this correctly

0x0005 – 0xFFFF = 0x0006

TCNT Overflow

When should we be concerned with TCNT overflow when the code calculates time difference?

- No overflow: No concern, current_time > last_time for sure
- One overflow: No concern if current_time < last_time
- Otherwise: The code should make adjustment

For lab 7, you can find a right clock prescalar value to avoid handling TCNT overflow

- Make sure the maximum time difference is less than 2^{16} clock cycles. Do not use an overly small prescalar
- Do not use an overly large prescalar, otherwise you won't get the desired resolution of measurement

TCNT Overflow

What happen if you have to deal with TCNT overflow?

	7	6	5	4	3	2	1	0
TIMSK	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

TOIE1: Timer/Counter 1 Overflow Interrupt Enable

This bit can be set to enable interrupt when TCNT1 overflows, i.e. changes from 0xFFFF to 0x0000

What to do with it? The idea: Record the number of overflows and the adjust the time difference

TCNT Overflow

```
volatile unsigned last_time = 0;
volatile unsigned current_time = 0;
volatile unsigned overflows = 0;
volatile unsigned new_overflows = 0;
volatile int update_flag = 0;

ISR (TIMER1_OVF_vect)
{
   new_overflows++;
}

ISR (TIMER1_CAPT_vect)
{
   last_time = current_time;
   overflows = new_overflows;
   current_time = ICR1;
   new_overflows = 0;
   update_flag = 1;
}
```

TCNT Overflow

```
unsigned long time_diff;
cli(); // disable interrupt
overflow -= (current_time < last_time);
time_diff = ((unsigned long)overflows<<16)
          + current_time – last_time;
update_flag = 0;
sei(); // enable interrupt
```

- The first overflow can be discounted if current_time < last_time
- For each overflow, increase time_diff by 65,536 (2^{16})
- You have to use long integer which is 32-bit (0 to 2^{32}-1)