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

Instructors:
Phillip Jones
Akhilesh Tyagi

Overview of Today's Lecture

- Announcements
- Input Capture Review

Announcements

- This week lab will use an Ultrasonic sensor

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

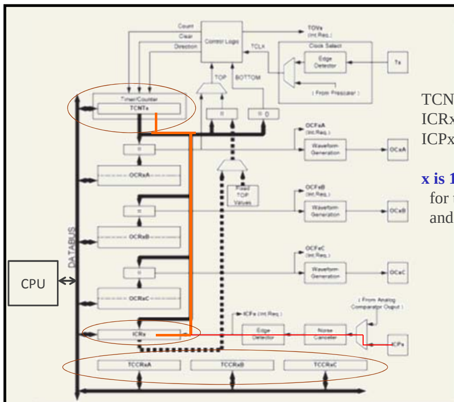
Timers/Counters

- The ATmega128's fundamental means of keeping track of time.

Timers/Counters

- The ATmega128 has 4 timers

- Timer0 – 8 bit
- Timer1 – 16 bit (used in sonar lab)
- Timer2 – 8 bit
- Timer3 – 16 bit



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

Timers/Counters

- Modes of operation
  - Normal
  - Clear Timer on Compare Match (CTC)
  - Pulse Width Modulation (PWM)

- See pages 123 to 130 of User Guide (doc2467) for detailed descriptions of each mode

Timers/Counters (Prescalers)

- Available prescalers = **1, 8, 64, 256, 1024**

- Prescalers **slow the speed** of a timer with respect to the system clock by a given factor

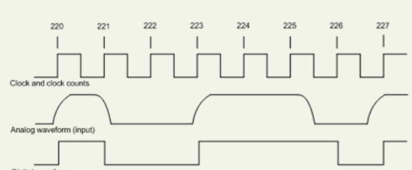
- Let ***p*** be the prescaler

- Tick rate of clock is *(system_clock_speed / p)*

Input Capture

An event is a transition of binary signal

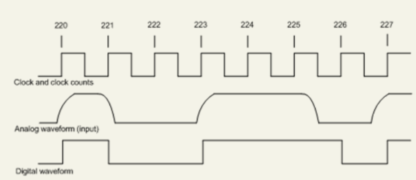
Example: How many events make up the following **waveform?**



Input Capture

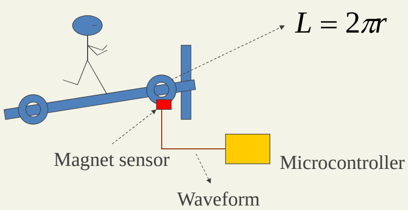
An input digitized and then times captured

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



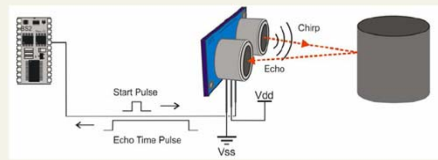
Application: Speedometer

How to detect the speed of a treadmill?

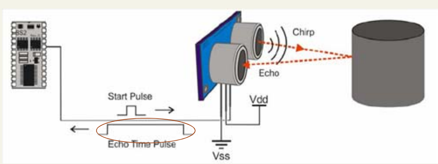


Application: Sonar Device

Ping))) sensor: ultrasound distance detection device



Sonar Principle



Sound Speed at Lab (ambient) Temperature: About 340m/s
Pulse width proportional to round-trip distance

* Temperature affects sound speed

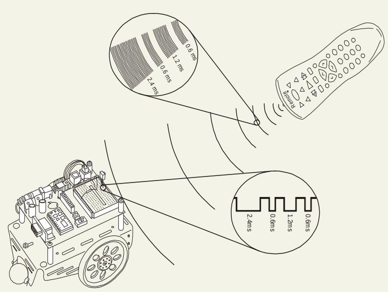
Sonar Principle

Assume 62.5KHz Input Capture clock
1ms <=> 62.5 clock ticks/periods <=> 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?

Application: Remote Control (Decoding)



Input Capture: Design Principle

Time is important!

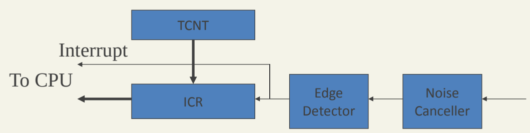
How could a microcontroller capture the time of an event, assuming a clock count can be read?

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

Precise timing is needed!

Input Capture: Design Principle

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



TCNT: Timer/Counter
ICR: Input Capture Register

Input Capture: Design Principle

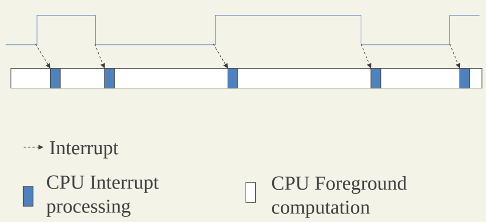
What happens in hardware and software when and after an event occurs

- 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



Input Capture: Design Principle

How to program the interrupt handler to:

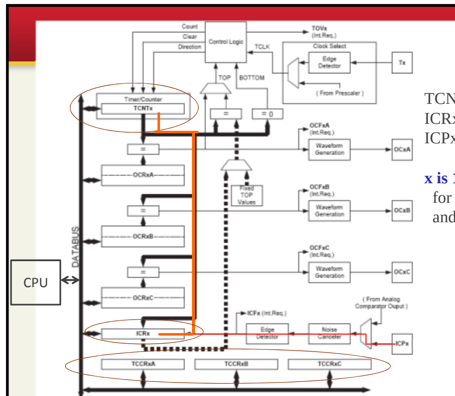
- Count the number of pulses
- Calculate pulse width
- Decode IR signals
- And many other functions …

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


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

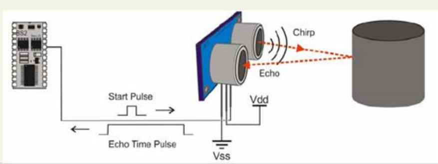
Use Input Capture: Example

```
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

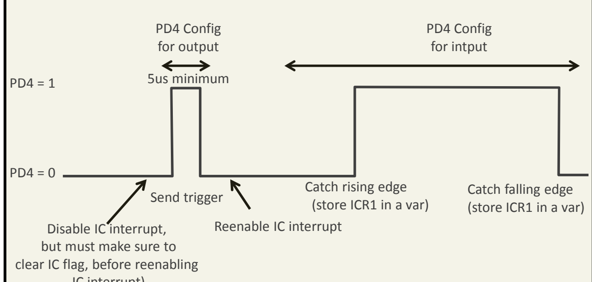
General idea:
- Configure Timer/Counter 1 for input capture
- Generate a pulse to activate the PING))) sensor
- Capture the time of rising edge event
- Capture the time of falling edge event
- Calculate time difference and then distance to any object



Application: Sonar Device

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

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
TIFR: Timer/Counter Interrupt Flag Register
ETIFR: Extended Timer/Counter Interrupt Flag Reg.

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 for 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 input capture? 0 for falling edge, 1 for rising edge

WGM32, WGM32: See previous slide

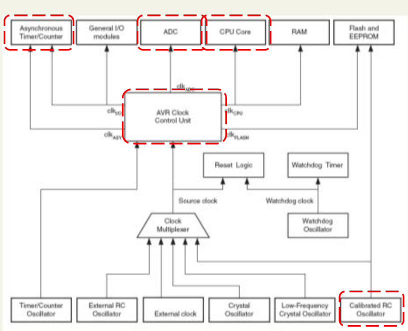
7	6	5	4	3	2	1	0	
ICNC3	ICES3	–	WGM33	WGM32	CS32	CS31	CS30	TCCR3B
R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	

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; Force output compare is not used for Lab7

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 (TCNT1 value) is overflowed

*Note: Use a **sufficiently 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 (TCNT3 value) is overflowed

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

TIFR is for Timer/Counter 1

ICF1: Timer/Counter 1, Input Capture Flag – Is set to 1 when an Input Capture even occurs

TOV1: Timer/Counter1, Overflow Flag – Is set to 1 when Timer1/Counter 1 value (TCCN1 value) is overflowed

OCF1A-C (or 3): Timer/Counter1, compare flag – Is set to 1 when Timer1/Counter 1 value (TCCN1 value) is equal to the corresponding Output Compare Register(OCR). See ETIFR as well (next slide).

See Data Sheet for **How to clear flag bits**, and under what conditions the programmer MUST clear the flag bits.

7	6	5	4	3	2	1	0	
–	–	ICF3	OCF3A	OCF3B	TOV3	OCF3C	OCF1C	ETIFR
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

ETIFR is for Timer/Counter 3

ICF3: Timer/Counter 3, Input Capture Flag – Is set to 1 when an Input Capture even occurs

TOV3: Timer/Counter3, Overflow Flag – Is set to 1 when Timer3/Counter 3 value (TCCN1 value) is overflowed

See Data Sheet for **How to clear flag bits**, and under what conditions the programmer MUST clear the flag bits.

Configure Timer/Counter 1 for Lab 7

TCCR1A: WGM bits = 0

TCCR1B: Enable interrupt, Choose correct Edge Select, WGM bits = 0, Choose appropriate Clock Select

TCCR1C: Keep all bit cleared

TIMSK: Enable Timer/Counter 1 Input Capture Interrupt

Port D pin 4 (PD4) – It is Timer1/Counter1's Input Capture (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 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 not follow timing example of slide 25.

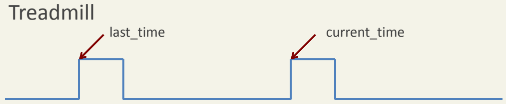
```
/* send out a pulse on PD4 */
void send_pulse()
{
  DDRD |= 0x10;          // set PD4 as output
  PORTD |= 0x10;         // set PD4 to high
  wait_ms(1);            // wait
  PORTD &= 0xEF;         // set PD4 to low
  DDRD  &= 0xEF;         // set PD4 as input
}

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

Note 1: This code does not work for Lab 7 as it is.
Note 2: Does not follow timing example of slide 25.


```
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

}
```


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

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

49

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 prescaler 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 prescaler
- Do not use an overly large prescaler, otherwise you won't get the desired resolution of measurement

50

TCNT Overflow

What happens if you have to deal with TCNT overflow?

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	

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

51

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;
}
```

52

TCNT Overflow

```
unsigned long time_diff;

overflow -= (current_time < last_time);
time_diff = ((unsigned long)overflows<<16)
          + current_time - last_time;
update_flag = 0;
```

- 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)

53