Overview of Today’s Lecture

- Announcements
- Input Capture Review

## Announcements

- Homework 6 due Tuesday 11:59pm (10/20)
- This week lab will use an Ultrasonic sensor

## 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
Input Capture

An event is a transition of binary signal
Example: How many events make up the following waveform?

Application: Speedometer

How to detect the speed of a treadmill?

Application: Sonar Device

Ping))) sensor: ultrasound distance detection device

Sonar Principle

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

* Temperature affects sound speed
The event’s time is captured first then read by the CPU.

**Input Capture: Design Principle**

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**

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
- The 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.

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
}

**Input Capture: Design Principle**
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

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

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](http://class.ece.iastate.edu/cpre288/resources/docs/28015-PING-v1.3.pdf)
Lab 7 General Idea of Programming

16-bit Timer/Counter Programming Interface

Lab 7 General Idea of Programming

10/13/2015

Lab 7 uses Timer/Counter 1

http://class.ece.iastate.edu/cpre288

Lab 7 uses Timer/Counter 3

http://class.ece.iastate.edu/cpre288

Data Sheet: Read it, and ask questions

• Exam 2 will predominantly consist of questions of the form
  – Program Configure Registers to meet these specs
    • UART, ADC, Input Capture, Output Compare, Timers, Interrupts
    • There is a section for each device mentioned above in the data sheet
  – Based on a given configuration, answer questions about how the
    program will behave
    • E.g. How long will something take to occur?
    • E.g. How many times a second will something occur?
  – Explain why a given configuration is incorrect for implementing a
    specified behavior
  – Assuming a given configuration, write a short program to
    implement a specific behavior
  – ADC calculation problem

Data Sheet: Read it, and ask questions

10/13/2015

http://class.ece.iastate.edu/cpre288

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

http://class.ece.iastate.edu/cpre288

16-bit Timer/Counter Programming Interface

http://class.ece.iastate.edu/cpre288

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

http://class.ece.iastate.edu/cpre288

http://class.ece.iastate.edu/cpre288
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

CS3x: Clock Select bits
Table in ATmega128 User Guide, page 137

<table>
<thead>
<tr>
<th>C3x2</th>
<th>C3x1</th>
<th>C3x0</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>No clock source, Timer/Counter stopped</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>C3x2/18 (fr from prescaler)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>C3x2/27 (fr from prescaler)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>C3x2/46 (fr from prescaler)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>C3x2/234 (fr from prescaler)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>C3x2/117 (fr from prescaler)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>External clock source on Tc pin. Clock on falling edge</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>External clock source on Tc pin. Clock on rising edge</td>
</tr>
</tbody>
</table>

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

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 (TCC3N) 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
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

---

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 (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.

---

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

---

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.

---

IC Programming Example

```c
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()
{
    ...
}

ISR (TIMER1_CAPT_vect)
{
    ...
}

/* 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.
```

---

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

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

```
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
```
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

What happen if you have to deal with TCNT overflow?

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

```c
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;
}
```

unsigned long time_diff;

```c
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)