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

Instructors:
Dr. Phillip Jones

Overview of Today’s Lecture

• Announcements
• Input Capture Review

Announcements

• Homework 4 due Sunday 11:59pm (6/19)
• This week lab will use an Ultrasonic sensor
• Exam 2: Friday June 24

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

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
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

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

<table>
<thead>
<tr>
<th>Time Diff.</th>
<th>Clock Count</th>
<th>One-way Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>2ms</td>
<td>125</td>
<td>0.34m</td>
</tr>
<tr>
<td>4ms</td>
<td>250</td>
<td>0.68m</td>
</tr>
</tbody>
</table>

How to capture the times of rising edge and falling edge?
An interrupt is raised to the CPU

CPU executes the input capture ISR, which reads the ICR

Count the number of pulses

—and many other functions …

Keep polling the input pin?

Use an interrupt?

???

Precise timing is needed!

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

TCNT: Timer/Counter

Interrupt

To CPU

ICR: Input Capture Register

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!

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

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

}
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

Use Input Capture: Example

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

PD4 = 0
PD4 Config for output
Sus minimum

PD4 = 1
Send trigger

Catch rising edge (store ICR1 in a var)

Catch falling edge (store ICR1 in a var)

Disablle I C interrupt, but must make sure to clear IC flag before reenabling IC interrupt

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

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 time a second with 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

16-bit Timer/Counter Programming Interface

Inside those TCCRs:

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

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>Clk2</th>
<th>Clk1</th>
<th>Clk0</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>chyoff? (No prescaling)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>chyoff/2 (From prescaler)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>chyoff/64 (From prescaler)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>chyoff/256 (From prescaler)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>External clock source on Tc pin. Clock on rising edge</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>External clock source on Tc pin. Clock on falling 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 (TCCN3) is overflowed

TIFR is for Timer/Counter 1

ICF1: Timer/Counter 1, Input Capture Flag – Is set to 1 when an Input Capture event occurs
TOV1: Timer/Counter1, Overflow Flag – Is set to 1 when Timer1/Counter 1 value (TCCN1) is overflowed
OCF1A-C (or 3): Timer/Counter1, compare flag – Is set to 1 when Timer1/Counter1 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.
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/Counter 3, 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.

ICF3 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()
{
    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
}

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()
{
    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.

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

```
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

```c
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.*)

---

**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

\( 0x0005 - 0xFFFF = 0x0006 \)

---

**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**

*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 2¹⁶ 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 int last_time = 0;
volatile unsigned int current_time = 0;
volatile unsigned int overflows = 0;
volatile unsigned int 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;

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)