Unit 14 – Timers and Counters
Counter/Timers Overview

• ATmega328P has two _____ and one ______ counters.
  – Can configure to count at some frequency up to some value (a.k.a. ________________), generate an ______________ and _______________ counting again, if desired
  – Useful for performing operations at specific time intervals. Every time an interrupt occurs, do something.
  – Can be used for other tasks such as pulse-width modulation (covered in future lectures)

• But don't we already have delay()...why do we need timers?
  – So that we can do __________________________ while we are waiting for time to elapse!
General Overview of Timer HW

System Clock (16MHz Arduino)

+ Prescalar (1, 8, 256, 1024)

16-bit Counter (TCNTx)
Increments every prescaled "clock"

Start Over @ 0?

0000 0001 1001 1100
Modulus A (OCRxA)
0000 0010 0000 0000
Interrupt if equal

0000 1010 0110 1100
Modulus B (OCRxB)
Interrupt if equal

We'll just use the modulus A register so you can ignore B for our class
## Counter/Timer Registers

### Bad News: Lots of register bits to deal with

<table>
<thead>
<tr>
<th>Control Register A (TCCR1A)</th>
<th>COM1A1</th>
<th>COM1A0</th>
<th>COM1B1</th>
<th>COM1B0</th>
<th>WGM11</th>
<th>WGM10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control Register B (TCCR1B)</td>
<td>ICNC1</td>
<td>ICES1</td>
<td>WGM13</td>
<td>WGM12</td>
<td>CS12</td>
<td>CS11</td>
</tr>
<tr>
<td>Control Register C (TCCR1C)</td>
<td>FOC1A</td>
<td>FOC1B</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Timer/Counter Register (TCNT1H &amp; TCNT1L)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>TCNT1[15:8]</td>
<td></td>
</tr>
<tr>
<td>Output Compare Register A (OCR1AH &amp; OCR1AL)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>OCR1A[15:8]</td>
<td></td>
</tr>
<tr>
<td>Output Compare Register B (OCR1BH &amp; OCR1BL)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>OCR1B[15:8]</td>
<td></td>
</tr>
<tr>
<td>Input Capture Register (ICR1H &amp; ICR1L)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ICR1[15:8]</td>
<td></td>
</tr>
<tr>
<td>Interrupt Mask Register (TIMSK1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ICIE1</td>
<td>OCIE1B</td>
</tr>
<tr>
<td>Interrupt Flag Register (TIFR1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ICF1</td>
<td>OCF1B</td>
</tr>
</tbody>
</table>
Counter/Timer Registers

- **Good News:** Can _______________ for simple timing

<table>
<thead>
<tr>
<th>Control Register B (TCCR1B)</th>
<th>WGM13</th>
<th>WGM12</th>
<th>CS12</th>
<th>CS11</th>
<th>CS10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Output Compare Register A (OCR1AH &amp; OCR1AL)</td>
<td></td>
<td>OCR1A[15:8]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interrupt Mask Register (TIMSK1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>OCIE1A</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Computing the Desired Cycle Delay

• **Primary step:** calculate how many processor \[ \text{__________} \] are required for your desired delay
  – Desired clock cycles = \[ \text{__________} \times \text{__________} \]
  – Arduino UNO clock is fixed at 16 MHz

• Example: **0.25 second delay** with a **16 MHz** clock
  – Desired clock cycles = \[ 16,000,000 \text{ c/s} \times 0.25\text{s} = \text{__________} \text{ cycles} \]

• Problem: The **desired value** you calculate must fit in at most a \[ \text{__________} \] (i.e. max \[ \text{__________} \])
  – If the number is bigger than 65,535 then a prescalar must be used to reduce the clock frequency to the counter from 16MHz to something slower
Calculating the Prescalar

• The counter prescaler divides the processor clock down to a lower frequency so the counter is counting ________________.

• Can divide the processor clock by four different powers of two: _____, ______, ________, or ________.

• Try prescalar options until the cycle count fits in 16-bits
  – 4,000,000 / 8 = 500,000
  – 4,000,000 / 64 = 62,500
  – 4,000,000 / 256 = 15,625
  – 4,000,000 / 1024 = 3906.25

• In this example, either of the ________________ could work but since we can only store integers in our timer count registers the last one would not yield exactly 0.25s (more like 0.249984s)
Counter/Timer Initialization 1

- **Set the mode for “Clear Timer on Compare” (CTC)**
  - WGM13 = _________, WGM12 = _________
  - This tells the hardware to _______________ at 0 once the counter is reaches your desired value

- **Enable “Output Compare A Match Interrupt”**
  - OCIE1A = 1

- **Load the 16-bit counter modulus into _____________**
  - This is the value the counter will count up to and then generate an interrupt.
  - The counter then clears to zero and starts counting up again.

  - In C, the register can be accessed as...
    - A 16-bit value "OCR1A"
    - Or as two eight bit values "OCR1AH" and OCR1AL”.

```
// Set to CTC mode
TCCR1B |= (1 << WGM12);
// Enable Timer Interrupt
TIMSK1 |= (1 << OCIE1A);
// Load the MAX count
// Assuming prescalar=256
// counting to 15625 =
// 0.25s w/ 16 MHz clock
OCR1A = 15625;
```
Counter/Timer Initialization 2

- Select the prescalar value with bits: CS12, CS11, CS10 in TCCR1B reg.
  - 000 = stop ⇐ Timer starts when prescaler set to non-zero
  - 001 = clock/1
  - 010 = clock/8
  - 011 = clock/64
  - 100 = clock/256
  - 101 = clock/1024

- Enable

```cpp
// Set to CTC mode
TCCR1B |= (1 << WGM12);

// Enable Timer Interrupt
TIMSK1 |= (1 << OCIE1A);

// Load the MAX count
// Assuming prescalar=256
// counting to 15625 =
// 0.25s w/ 16 MHz clock
OCR1A = 15625;

// Set prescalar = 256
// and start counter
TCCR1B |= (1 << CS12);

// Enable interrupts
sei();
```
Counter/Timer Initialization 3

• Make sure you have an appropriate ISR function defined
  – Using name ISR(TIMER1_COMPA_vect)

```c
#include <avr/io.h>
#include <avr/interrupt.h>

unsigned char qsecs = 0;

void init_timer1(unsigned short m)
{
    TCCR1B |= (1 << WGM12);
    TIMSK1 |= (1 << OCIE1A);
    OCR1A = m;
    TCCR1B |= (1 << CS12);
}

int main()
{
    init_timer1(15625);
    sei();

    while()
    {
        // do something with qsecs
    }
    return 0;
}

ISR(TIMER1_COMPA_vect){
    // increments every 0.25s
    qsecs++;
}
```
8-bit Counter/Timers

- The other two counters are similar but only 8-bits.
- Same principle: find the count modulus that fits in an 8-bit value.

<table>
<thead>
<tr>
<th>Control Register A (TCCR0A)</th>
<th>COM0A1</th>
<th>COM0A0</th>
<th>COM0B1</th>
<th>COM0B0</th>
<th>WGM01</th>
<th>WGM00</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control Register B (TCCR0B)</td>
<td>FOC0A</td>
<td>FOC0B</td>
<td></td>
<td></td>
<td>WGM02</td>
<td>CS02</td>
</tr>
<tr>
<td>Timer/Counter Register (TCNT0)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CS01</td>
<td>CS00</td>
</tr>
<tr>
<td>Output Compare Register A (OCR0A)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Output Compare Register B (OCR0B)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>OCR0A[7:0]</td>
<td></td>
</tr>
<tr>
<td>Interrupt Mask Register (TIMSK0)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>OCR0B[7:0]</td>
<td>OCIE0B OCIE0A TOIE0</td>
</tr>
<tr>
<td>Interrupt Flag Register (TIFR0)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>OCF0B OCF0A TOV0</td>
<td></td>
</tr>
</tbody>
</table>
ISR Names

• In CTC mode, an "Output Compare A Match Interrupt" will vector to an ISR with these names:

  – ISR(TIMER0_COMPA_vect) { } /* 8-bit Timer 0 */

  – ISR(TIMER1_COMPA_vect) { } /* 16-bit Timer 1 */

  – ISR(TIMER2_COMPA_vect) { } /* 8-bit Timer 2 */