Unit 8 – 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

Counter/Timer Registers

- Bad News: Lots of register bits to deal with
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>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output Compare Register A (OCR1AH &amp; OCR1AL)</th>
<th>OCR1A[15-8]</th>
<th>OCR1A[7-0]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Interrupt Mask Register (TIMSK1)</th>
<th>OCIE1A</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Computing the Desired Cycle Delay

• **Primary step:** calculate how many processor __________________ are required for your desired delay
  - Desired clock cycles = ______________ × ______________
  - Arduino UNO clock is fixed at 16 MHz

• Example: 0.25 second delay with a 16 MHz clock
  - Desired clock cycles = 16,000,000 c/s × 0.25s = ______________ cycles

• Problem: The desired value you calculate must fit in at most a ______________ (i.e. max ______________).
  - If the number is bigger than 65,535 then a prescaler must be used to reduce the clock frequency to the counter from 16MHz to something slower

Calculating the Prescaler

• 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 prescaler 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”.

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

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

// Load the MAX count
// Assuming prescaler=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 _________________________

```
// Set to CTC mode
TCCR1B |= (1 << WGM12);
// Enable Timer Interrupt
TIMSK1 |= (1 << OCIE1A);
// Load the MAX count
// Assuming prescaler=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)

```
#include <avr/io.h>
#include <avr/interrupt.h>
unsigned char qsecs = 0;
void init_timer1(unsigned short m)
{
  TCCR1B |= (1 << WGM12);
  // Enable Timer Interrupt
  TIMSK1 |= (1 << OCIE1A);
  OCR1A = m;
  TCCR1B |= (1 << CS12);
  // Enable interrupts
  sei();
}
int main()
{
  init_timer1(15625);
  sei();
  while(){
    // do something w/ 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.

```
Control Register A (TCCRA0)
Control Register B (TCCRB0)
Timer/Counter Register (TCNT0)
Output Compare Register A (OCR0A)
Output Compare Register B (OCR0B)
Interrupt Mask Register (TIMSK0)
Interrupt Flag Register (TIFR0)

<table>
<thead>
<tr>
<th>WGM01</th>
<th>WGM00</th>
<th>COMB0</th>
<th>COMA0</th>
<th>COMA1</th>
</tr>
</thead>
<tbody>
<tr>
<td>WGM00</td>
<td>WGM01</td>
<td>COMB0</td>
<td>COMA0</td>
<td>COMA1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>WGM3</th>
<th>WGM2</th>
<th>WGM1</th>
<th>WGM0</th>
<th>CS3</th>
<th>CS2</th>
<th>CS1</th>
<th>CS0</th>
</tr>
</thead>
<tbody>
<tr>
<td>WGM0</td>
<td>WGM1</td>
<td>WGM2</td>
<td>WGM3</td>
<td>CS0</td>
<td>CS1</td>
<td>CS2</td>
<td>CS3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>WGM3</th>
<th>WGM2</th>
<th>WGM1</th>
<th>WGM0</th>
<th>OCIE0B</th>
<th>OCIE0A</th>
<th>TOE0</th>
</tr>
</thead>
<tbody>
<tr>
<td>WGM0</td>
<td>WGM1</td>
<td>WGM2</td>
<td>WGM3</td>
<td>OCIE0A</td>
<td>OCIE0B</td>
<td>TOE0</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 */