Unit 14 – Timers and Counters
A BRIEF SUMMARY

Review of some key concepts from the first half of the semester
A Few Big Ideas 1

• Setting and clearing bits in a register tells the hardware what do and when (this is SW interacting with HW)

• Speed matters
  – Your software is executing quickly compared to how fast a human can do something
  – You can use that to your advantage: blinking an LED at a fast rate can give the illusion it's always on but just more dim
  – Or it can work to your disadvantage: One button press may look like many because a loop may see one press on multiple iterations.
  – **We must write our software with this in mind**
A Few Big Ideas 2

- Clocking or enables are necessary to say "when"
  - Digital signals are always 1's and 0's so just looking at the bits doesn't tell us how many we have
  - We usually need clocks (pulses) to tell the hardware when we want it to grab the data

Just looking at this set of digital values, are we sending 0101 once, twice, three times, how many? Once because we use the clock/enable to indicate that. But without the clock we'd have no clue how many times we are trying to write 0101
A Few Big Ideas 3

• External events happen asynchronously with your software (don't know "when" something has happened)
  – Your software program is the brains for how to process information but it doesn't magically know "when" something has happened?
  – We have to keep checking it (polling) or
  – Hardware designers built "interrupt" mechanisms to help

• Many tasks can be done in either SW or HW; SW may be easier to code/use but HW provides parallelism
  – A 0.1 second timer can be done in SW using delays but then software can't do much else
  – Or in HW using timers allowing SW to do other tasks
TIMERS
Counter/Timers Overview

- ATmega328P has two 8-bit and one 16-bit counters.
  - Can configure to count at some frequency up to some value (a.k.a. counter modulus), generate an interrupt and start over 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 other useful work while we are waiting for time to elapse!
General Overview of Timer HW

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>Register Type</th>
<th>Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control Register A (TCCR1A)</td>
<td>COM1A1</td>
</tr>
<tr>
<td>Control Register B (TCCR1B)</td>
<td>ICNC1</td>
</tr>
<tr>
<td>Control Register C (TCCR1C)</td>
<td>FOC1A</td>
</tr>
<tr>
<td>Timer/Counter Register (TCNT1H &amp; TCNT1L)</td>
<td></td>
</tr>
<tr>
<td>Output Compare Register A (OCR1AH &amp; OCR1AL)</td>
<td></td>
</tr>
<tr>
<td>Output Compare Register B (OCR1BH &amp; OCR1BL)</td>
<td></td>
</tr>
<tr>
<td>Input Capture Register (ICR1H &amp; ICR1L)</td>
<td></td>
</tr>
<tr>
<td>Interrupt Mask Register (TIMSK1)</td>
<td></td>
</tr>
<tr>
<td>Interrupt Flag Register (TIFR1)</td>
<td></td>
</tr>
</tbody>
</table>
Counter/Timer Registers

• Good News: Can ignore most for simple timing

Control Register B (TCCR1B)  WGM13  WGM12  CS12  CS11  CS10

Output Compare Register A (OCR1AH & OCR1AL)  OCR1A[15:8]

Interrupt Mask Register (TIMSK1)  OCIE1A
Computing the Desired Cycle Delay

- **Primary step**: calculate how many processor clock cycles are required for your desired delay
  - Desired clock cycles = clock frequency × delay time
  - 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 = 4,000,000 cycles

- **Problem**: The desired value you calculate must fit in at most a 16-bit register (i.e. max 65,535)
  - 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 slower.

• Can divide the processor clock by four different powers of two: 8, 64, 256, or 1024.

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

• In this example, either of the last three 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 = 0, WGM12 = 1
  - This tells the hardware to start over 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 OCR1A
  - 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 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 \( \iff \) Timer starts when prescaler set to non-zero
  - 001 = clock/1
  - 010 = clock/8
  - 011 = clock/64
  - 100 = clock/256
  - 101 = clock/1024

- Enable global interrupts

```c
// 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 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.

<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</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CS01</td>
</tr>
<tr>
<td>TCNT0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CS00</td>
</tr>
<tr>
<td>Output Compare Register A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OCR0A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Output Compare Register B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OCR0B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interrupt Mask Register</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIMSK0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interrupt Flag Register</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIFR0</td>
<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>OCIE0B</td>
<td>OCIE0A</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>OCF0B</td>
<td>OCF0A</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 */