Unit C – Timers and Counters

Counter/Timers Overview

ATmega328P has two 8-bit and one 16-bit counter/timers.
- Can count at some rate up to a value, generate an interrupt and start over counting from 0.
- Useful for performing operations at specific time intervals.
- Can be used for other tasks such as pulse-width modulation or counting external events.

But we already have delay() functions ... why do we need timers?
- Delay functions tie up the processor while executing.
- Better to let a timer measure the delay, and generate an interrupt when complete.
- We can do other useful work while we are waiting for time to elapse!

while(1) {
    lcd_moveto(0,0);
    lcd_stringout("hello");
    _delay_ms(500);
} // is "hello" printed // every 500 ms?

Use of Timers

- Use 1: Generate an interrupt at a regular interval
  - Great for fixed time periods
- Use 2: Use HW timer to measure time duration
  - Great for measuring unknown timer intervals
General Overview of Timer HW

- **16-bit Counter (TCNTx)**: Increments every prescaled "clock".
- **Modulus A (OCRxA)**: 0000 0010 0000 0000
- **Modulus B (OCRxB)**: 0000 1010 0110 1100

We'll just use the modulus A register so you can ignore B for our class.

Duration Timer

- Timer can be configured to count at a certain rate ($\Delta t$)
- Start/stop the timer when the microcontroller senses the start/stop of an event
- The count value, n, can determine the duration of the event: $T = n \cdot \Delta t$

Periodic Interrupt Timer

- Timer can be configured to count at a certain rate ($\Delta t$) and an upper bound (aka modulus count / OCRxA register)
- Start the timer and whenever its value reaches the upper bound an interrupt will be generated
  - Can be configured to immediately restart the count at 0 and repeat

Periodic Interrupt Timer Steps

To use the counter to generate interrupts at a fixed interval:

- Decide how long an interval is required between interrupts (1 sec, 50 ms, etc.) for your application
- Determine a counter frequency (time period), and a counter modulus (max period) that will make the counter take that long to count from 0 to the modulus value.
- Configure registers.
- Write an ISR.
- Start the timer.
### Counter/Timer Registers

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

<table>
<thead>
<tr>
<th>Control Register A (TCRA1A)</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 (TCRA1B)</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 (TCRA1C)</td>
<td>FOCA1</td>
<td>FOCA2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Timer/Counter Register (TCNT1H &amp; TCNT1L)</th>
<th>TCNT1[15:8]</th>
<th>TCNT1[7:0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output Compare Register A (OCR1AH &amp; OCR1AL)</td>
<td>OCR1A[15:8]</td>
<td>OCR1A[7:0]</td>
</tr>
<tr>
<td>Output Compare Register B (OCR1BH &amp; OCR1BL)</td>
<td>OCR1B[15:8]</td>
<td>OCR1B[7:0]</td>
</tr>
<tr>
<td>Input Capture Register (ICR1H &amp; ICR1L)</td>
<td>ICR1[15:8]</td>
<td>ICR1[7:0]</td>
</tr>
<tr>
<td>Interrupt Mask Register (TIMSK1)</td>
<td>ICIE1</td>
<td>OCIE1B</td>
</tr>
<tr>
<td>Interrupt Flag Register (TIFR1)</td>
<td>ICF1</td>
<td>OCF1B</td>
</tr>
</tbody>
</table>

**Good News:** Can ignore most for simple timing

<table>
<thead>
<tr>
<th>Control Register B (TCRA1B)</th>
<th>WGM13</th>
<th>WGM12</th>
<th>CS12</th>
<th>CS11</th>
<th>CS10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output Compare Register A (OCR1AH &amp; OCR1AL)</td>
<td>OCR1A[15:8]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Output Compare Register B (OCR1BH &amp; OCR1BL)</td>
<td>OCR1B[15:8]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Input Capture Register (ICR1H &amp; ICR1L)</td>
<td>ICR1[15:8]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interrupt Mask Register (TIMSK1)</td>
<td>OCIE1A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Computing the Desired Cycle Delay

**Primary step:** calculate how many processor clock cycles are required for your desired delay

- Desired clock cycles (aka "modulus") = clock frequency × desired 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
- The Arduino timer starts at 0, so we will set the max count to 3,999,999
  - 4,000,000 - 1 = 3,999,999

**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 prescalar 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 / 8 = 500,000 ← too big
  - 4,000,000 / 64 = 62,500 ← OK
  - 4,000,000 / 256 = 15,625 ← OK
  - 4,000,000 / 1024 = 3906.25 ← OK, but not an integer
- 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 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”.

```
// 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
  - 001 = clock/1
  - 010 = clock/8
  - 011 = clock/64
  - 100 = clock/256
  - 101 = clock/1024
- Enable global interrupts

```
// 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 3

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

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

volatile unsigned cnt = 0;

void init_timer1(unsigned short m) {
  TCCR1B |= (1 << WGM12); // Set to CTC mode
  TIMSK1 |= (1 << OCIE1A); // Enable Timer Interrupt
  OCR1A = m; // Load the MAX count
  TCCR1B |= (1 << CS12); // Set prescalar = 256
  sei(); // Enable interrupts
}

ISR(TIMER1_COMPA_vect) {
  // increments every 0.25s
  cnt++;
}
```

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.
8-bit Timers (Timer 0 & Timer 2)

- Timer0 (Timer2) of the Arduino only have an 8-bit timer and max count value (thus we can only count up to 255)
- Set WGM01 (WGM21) bit to CTC
- Enable interrupt via OCIE0A (OCIE2A) bit in TIMSK0 (TIMSK2) register
- Load the OCR0A (OCR2A) Register
- Start timer when desired by setting appropriate prescalar

<table>
<thead>
<tr>
<th>Prescaler</th>
<th>WGM7:1, WGM0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>Timer off</td>
<td>00 Normal (Counter)</td>
</tr>
<tr>
<td>010</td>
<td>Ck / 8</td>
<td>01 Phase Correct PWM</td>
</tr>
<tr>
<td>011</td>
<td>Ck / 64</td>
<td>10 CTC (Timer)</td>
</tr>
<tr>
<td>100</td>
<td>Ck / 256</td>
<td>11 Fast PWM (Top=255, Thresh=OCR)</td>
</tr>
<tr>
<td>101</td>
<td>Ck / 1024</td>
<td></td>
</tr>
</tbody>
</table>

8-bit Timers can only count up to 255. Be sure to select a prescalar such that your OCR value will fit in 8-bits.

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 */