

Trinket Timer notes

Bruce E. Hall, [W8BH](#)

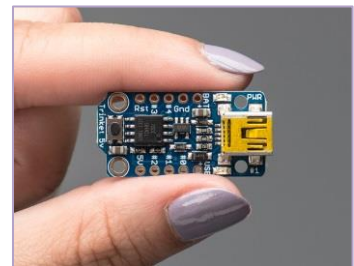
INTRODUCTION

The counter/timer is a very useful part of the AVR microcontroller! There are many tutorials on the web that explain how it works. I find most of the examples overly complicated. Here is a collection of my notes on simple counters. This isn't a full-fledged tutorial – just a demonstration of easy-peasy counters on a small micro, the ATtiny85, using the Arduino IDE. If you understand these examples you will be able to use counters on any AVR device.

HOW TO START

Get yourself a Trinket, if you don't have one! Everything you need is at Adafruit: the 5V [trinket](#), a [piezo buzzer](#), a small [breadboard](#), and a [USB-miniB cable](#). I also recommend a 200-300 ohm resistor to limit current through the piezo.

Next, you need to add the Arduino IDE to your computer. The easiest way is to use an IDE preconfigured for the Trinket. Go to the Adafruit learning site "[Introducing Trinket](#)" to learn more, and download the software.



The hardware couldn't be simpler: plug your trinket into a breadboard, and connect the (+) lead of your piezo to digital pin 1, labelled #1 on the trinket board. Connect the other piezo lead to ground (labelled GND on the trinket) through a 200 to 300 ohm resistor. That's it.

EXPERIMENT #1: HELLO WORLD, COUNTER-STYLE

What is the simplest program we can write, using a counter? How about this:

```
// Example 1: Make a LED blink with only 3 coding statements

void setup()
{
  DDRB = 0x02;           // set PB1 as an output
  TCCR1 = 0x9F;         // configure counter control register
  OCR1C = 100;          // set LED blink rate about 2 Hz
}

void loop()
{
}
```

Three coding lines are all you need to set up a counter and make the on-board LED blink. Only two lines are counter-related. Most “hello-world” programs are significantly longer. This one is short and simple, right?

The first line sets pin PB1 (digital output 1) as an output, so that we can drive an LED connected to this pin. Digital output #1 is bit 1 of data PortB , abbreviated PB1. DDRB is the register name for “Data Direction Register B”, which controls the data direction for each output on Port B. Each bit in the register corresponds to an output: bit0 is PB0, bit1 is PB1, etc. Use a ‘1’ value for outputs and ‘0’ for inputs.

DDRB = 0x02:

Bit 7	bit6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0	0	0	0	0	0	1	0
Input	Input	Input	Input	Input	Input	OUTPUT	Input

An easy-to-read way of doing the same thing is to use the pinMode function: pinMode(1,OUTPUT) will also set digital output pin 1 as an output. Either way is fine.

The second line in our program, TCCR1=0x9F, configures our timer for a handy-dandy LED blinker. Just like the first line, we are setting a microcontroller register with a specific value, and each bit of the register has a specific purpose. In this case, the register is the Timer/Counter Control Register 1, or TCCR1. If you go to the [ATtiny85 datasheet](#), you will find a description like this:

TCCR1 = 0x9F:

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
CTC1	PWM1A	COM1A1	COM1A0	CS13	CS12	CS11	CS10
1	0	0	1	1	1	1	1

The control register contains three bit groups, as shaded above. The first group specifies the counter mode. In this example we are enabling the CTC mode by setting bit7 to logic 1. CTC is an abbreviation for “Clear Timer on Compare” and works like this: the counter increments from zero to a user-specified 8-bit upper limit value. When the value is reached, the timer resets to zero and the cycle repeats.

The middle group in green, bits 4 & 5, controls the counter action. In this experiment we use these bits to make our LED flash. The following table lists the actions associated with these bits.

COM1A1	COM1A0	Description
0	0	Disconnect counter from OC1A output.
0	1	Toggle the OC1A output.
1	0	Clear the OC1A output.
1	1	Set the OC1A output.

In our example, we set the bits to '01', so that our output line (OC1A, or digital output #1) toggles. The LED is connected to this line, and flashes as the line toggles its state.

The last group in orange, bits 0 to 3, control the prescaler value. The prescaler sits between the microcontroller clock and the counter, and divides the counting frequency by a user-determined amount. Assuming an 8 MHz clock frequency, the resulting counter frequency is set according to the following table.

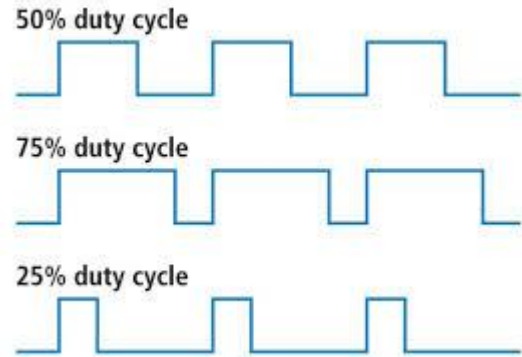
CS13	CS12	CS11	CS10	Divisor	Counter Input Frequency
0	0	0	0	Disconnect	Stopped
0	0	0	1	/1	8 MHz
0	0	1	0	/2	4 MHz
0	0	1	1	/4	2 MHz
0	1	0	0	/8	1 MHz
0	1	0	1	/16	500 kHz
0	1	1	0	/32	250 kHz
0	1	1	1	/64	125 kHz
1	0	0	0	/128	62500 Hz
1	0	0	1	/256	31250 Hz
1	0	1	0	/512	15625 Hz
1	0	1	1	/1024	7813 Hz
1	1	0	0	/2048	3906 Hz
1	1	0	1	/4096	1953 Hz
1	1	1	0	/8192	977 Hz
1	1	1	1	/16384	488 Hz

In our example, all of the prescaler bits are set to 1. The prescaler divides the 8 MHz clock by 16384, resulting in a counter frequency of 488 Hz.

Summarizing the above, we put the counter into CTC mode. The counter increments its register with each input pulse, and then resets to zero whenever the counter value matches the value of register OCR1C. We set the register value to 100, so the counter increments from 0 to 100 (in 101 clock pulses) and then resets itself to 0. The counter clock is running at 488 Hz, so the counting cycle repeats at a frequency of 488 Hz/101 or 4.8 Hz. The output line toggles at this rate, so the LED flashes half as fast, or 2.4 Hz. The output frequency for CTC mode is: $\text{freq(Hz)} = 8 \text{ MHz} / 2 / \text{prescaler} / (\text{OCR1C} + 1)$.

EXPERIMENT #2

The first experiment looked at the CTC counter mode. Experiment #2 uses a different mode, called Pulse Width Modulation. The output waveform is changed by altering the [duty cycle](#), or percentage of time spent in the 'ON' or logic 1 state. Unlike CTC, where the output voltage is always 50% on/50% off, PWM can vary in duty cycle between full on (100%) and full off (0%). PWM mode is especially useful for controlling LED brightness and servo motor direction. Here is the code:



```
// Example 2: Make a LED blink with PWM, shortening the 'ON' time

void setup()
{
  DDRB = 0x02;           // set PB1 as an output
  TCCR1 = 0x5F;         // configure counter control register
  OCR1C = 200;          // set LED blink rate about 2 Hz
  OCR1A = 20;           // set 10% duty cycle (21/201 = 0.1)
}
```

The LED still flashes about twice a second, but each flash is shorter. The ratio $(OCR1A+1)/(OCR1C+1)$ sets the duty cycle. Change the value in OCR1A to 100, and we are back to a 50% duty cycle. Increase OCR1A again to 160, and the LED is on 80% of the time.

To put the counter in PWM mode, set the PWM1A bit in the TCCR1 register:

TCCR1 = 0x5F:

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
CTC1	PWM1A	COM1A1	COM1A0	CS13	CS12	CS11	CS10
0	1	0	1	1	1	1	1

The prescaler and output action bits are the same as in Experiment #1. The frequency of the output is doubled, however, because the waveform period requires one counter cycle instead of two: $f(\text{PWM output}) = 8 \text{ MHz}/\text{prescaler}/(\text{OCR1C}+1)$.

You can use this same mode for audio output. For example, by setting the prescaler to 256, the clock input frequency will jump for 488 Hz to $8 \text{ MHz}/256 = 31250 \text{ Hz}$. If we set OCR1C to 70, the resulting output frequency will be $8 \text{ MHz}/256/71 = 440 \text{ Hz}$. See #2A in the source code listing below.

EXPERIMENT #3

Counter 1 provides PWM output in complimentary pairs: while one output line is high, the other is low, and vice versa. The inverted output is enabled when The ComA1x bits are set to '01', as shaded in green above, and disabled when the bits are set to '10'. The inverted output line is PB0. To enable the second output pin, add PB0 as an output:

DDRB = 0x03:

Bit 7	bit6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0	0	0	0	0	0	1	1
Input	Input	Input	Input	Input	Input	OUTPUT	OUTPUT

Next, adjust the prescaler to produce output in the audio range. Here is the code:

```
// Example 3: Sound! PWM on counter 1, with complementary outputs

void setup()
{
  DDRB = 0x03;           // set PB0 & PB1 as outputs
  TCCR1 = 0x59;         // configure counter control register
  OCR1C = 69;           // set audio pitch: 8 Mhz/256/(69+1) = 446 Hz
  OCR1A = 34;           // set 50% duty cycle (35/70 = 0.50)
}
```

This value of DDRB sets digital #0 and #1 as outputs. The updated value of TCCR1 reduces the prescaler to /256, setting the clock input frequency at 31250 Hz. Using the equations above, an OCR1C of 69 will set the output frequency to $8 \text{ MHz}/256/70 = 446 \text{ Hz}$, and the duty cycle will be $35/70 = 50\%$.

Connect a piezo element between PB0 and ground, and you will hear the audio note. Try PB1 instead of PB0, and you should hear the same pitch, at the same volume. Now, connect the piezo element between PB0 and PB1, and the sound will be louder. Why? With either output by itself, current flows through the element only half the time. When it is connected between complementary outputs, current is flowing 100% of the time.

EXPERIMENT #4

Although all of the Atmel AVR series of microcontrollers contain similar timers, they are not the same. Code meant for one counter may not work at all on another counter, even for counters on the same microcontroller. Here is a comparison of the ATtiny85 counters:

	Counter 0	Counter 1
Counter size	8 bits	8 bits
Prescaler choices	4 (/8,/64,/256,/1024)	14 (/2 through /16384)
Modes of operation	6 (4 PWM)	3 (1 PWM)
Configuration registers	2	1

6	1	1	0	Reserved
7	1	1	1	Fast PWM, top=OCR0A

The three bits in orange, the clock select bits, determine the value of our prescaler. Here are the choices. Our experiment uses a /64 prescaler, yielding a counter input frequency of $8 \text{ MHz}/256 = 125 \text{ kHz} = 125,000 \text{ Hz}$.

CS02	CS01	CS00	Prescaler Divisor	Counter Input Frequency
0	0	0	Disconnect	Counter stopped
0	0	1	/1	8 MHz
0	1	0	/8	1 MHz
0	1	1	/64	125 kHz
1	0	0	/256	31250 Hz
1	0	1	/1024	7813 Hz
1	1	x	/1	External clock source on PB2

In PWM mode, the output pin cycles once as the counter register goes from 0 to 255. The output frequency is therefore fixed at $125000/256 = 488 \text{ Hz}$. The duty cycle is $(\text{OCR0A}+1)/256 = 86/256 = 0.33$ or 33%. Connect a piezo element between digital output 0 and ground, and you should hear an audio tone near middle B ($B_4 = 494 \text{ Hz}$). Change the prescaler to /8, and the output frequency would jump to 3906 Hz. This note is B7, three octaves higher in pitch. In general, the output frequency is fixed at $f(\text{PWM output}) = 8 \text{ MHz}/\text{prescaler}/256$.

EXPERIMENT #5

In the last experiment, we could easily adjust the duty cycle, but had few options for the frequency. The next PWM mode adds frequency adjustability. Here is the code:

```
// Example 5: Fast PWM (mode 7), using Counter 0

void setup()
{
  DDRB = 0x02;           // set PB1 as an output
  TCCR0A = 0x23;        // configure control register A
  TCCR0B = 0x0B;        // configure control register B
  OCR0A = 200;          // set top of counter
  OCR0B = 100;          // set 50% duty cycle (101/201 = 0.50)
}
```

The output frequency is more agile, and controllable by register OCR0A. This is the equivalent mode to the counter1 PWM.

If OCR0A is used to control the output frequency, which register is used to control the duty cycle? Honestly, I don't see that information in the datasheet. The trick is to use the second output compare unit! Use the compare register from unit A to set the frequency; for the rest, use

compare unit B. The compare register from B sets the PWM, and OC0B (which is PB1) is the output line.

Here are the configuration registers for PWM mode 7 on Counter0:

TCCR0A = 0x23:

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
COM0A1	COM0A0	COM0B1	COM0B0	-----	-----	WGM01	WGM00
0	0	1	0	0	0	1	1

TCCR0B = 0x0B:

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
FOC0A	FOC0B	-----	-----	WGM02	CS02	CS01	CS00
0	0	0	0	1	0	1	1

Notice in the green boxes that we are using output compare unit B, not A. All three of the WGM bits are set in PWM mode 7.

In general, the **frequency = 8 MHz/prescaler/(OCR0A+1)** and the **duty cycle = (OCR0B+1)/(OCR0A+1)**. Here, the frequency is 8 MHz/64/201 = 622 Hz.

EXPERIMENT #6

There are two other PWM modes. They are phase-corrected versions of the two “fast-modes” shown above. Phase-correct PWM is useful for motor control.

With phase-correction, each counter cycle is doubled in length: the counter counts up to its top value, then reverses direction and counts down to zero. As a consequence, the frequency of the output is halved. The important property of phase-correct PWM is that the center point of the high and low states of the output cycle is the same for all duty cycles.

Start with the same code as the last experiment, and change the WGM01 bit in TCCR0A:

```
// Example 6: Phase-correct PWM (mode 5), using Counter 0

void setup()
{
  DDRB = 0x02;           // set PB1 as an output
  TCCR0A = 0x21;         // configure control register A
  TCCR0B = 0x0B;         // configure control register B
  OCR0A = 100;           // set top of counter for frequency
  OCR0B = 50;            // set 50% duty cycle (51/101 = 0.50)
}
```

Are you confused about the register bits, and which bits control the mode? Look back at the tables in first experiment. Frequency in this Phase-correct PWM mode:

freq = 8 MHz/prescaler/(OCR0A+1)/2. Duty cycle = (OCR0B+1)/(OCR0A+1).

SOURCE CODE

The following pages contain the full, commented code for all six experiments.

- EXP #1: Hello, World! Blinking a LED using CTC on Counter1
- EXP #2: LED blink with PWM on Counter1, shortening the 'ON' time
- EXP 2a: Same as above, but higher frequency (audio) output.
- EXP #3: Sound, using Fast PWM on Counter1, complementary outputs
- EXP #4: Sound, using Fast PWM mode 3 on Counter0
- EXP #5: Sound, using Fast PWM mode 7 on Counter0
- EXP #6: Sound, using Phase-correct PWM mode 5 on Counter0

Look at the [Trinket Tone Generator](#) to see a counter used in CTC mode.

```

//=====
// EXP #1: Hello, World!  Blinking a LED using CTC on Counter1
//
// Author:      Bruce E. Hall
// Version:    1.0
// Date:       14 Jan 2014
// Purpose:    Testing use of Timers with the Adafruit Trinket
// Hardware:   ATTINY85
// Software:   Arduino 1.0.5
//
// Here is an experiment for generating an audio tone.
//
// TCCR1 configuration for CTC mode on counter 1:
//
// -----
// CTC1  PWM1A  COM1A1  COM1A0  CS13  CS12  CS11  CS10
// 1      0      0        1        1      1      1      1
// -----
//
// Setting CTC1 will put this counter in CTC mode
// COM1A0 will activate a toggle output on OC1A (PB1 or digital pin 1)
// Setting CS13..CS11 will result in prescaler of /16384
// Counter clock = 8 MHz/16384 = 488 Hz
// Output Frequency = 488 Hz/2/(top+1) = 488/2/101 = 2.4 Hz
//
// In CTC mode, the output pin is toggled & counter is reset
// when the value of OCR1C is reached:
// 0 -----> OCR1C (top)
//                                     output pin toggled
//
// Frequency in this mode = 8 MHz/prescaler/(OCR1C+1)/2.
// The duty cycle is always 50%

void setup()
{
  DDRB = 0x02;          // PB1 is an output, connected to onboard LED
  TCCR1 = 0x9F;        // configure control register, as above
  OCR1C = 100;         // set LED blink rate to 2.4 Hz
}

void loop()
{
  // nothing!
}

```

```

//=====
// EXP #2: LED blink with PWM on Counter1, shortening the 'ON' time
//
// Author: Bruce E. Hall
// Version: 1.0
// Date: 14 Jan 2014
// Purpose: Testing use of Timers with the Adafruit Trinket
// Hardware: ATTINY85
// Software: Arduino 1.0.5
//
//
// TCCR1 configuration for (fast) PWM mode:
//
// -----
// CTC1 PWM1A COM1A1 COM1A0 CS13 CS12 CS11 CS10
// 0 1 0 1 1 1 1 1
// -----
//
// Setting PWM1A will put this counter in PWM mode.
// COM1A0 will activate output on OC1A (PB1 or digital pin 1)
// Setting CS13..CS11 will result in prescaler of /16384
// Counter clock = 8 MHz/16384 = 488 Hz.
// Output Frequency = 488 Hz/(top+1) = 488/201 = 2.4 Hz.
//
// With PWM, the output pin is set when counter is 0, and then
// cleared when the value of OCR1A is reached:
// 0 -----> OCR1A -----> OCR1C (top)
// output ON output OFF
//
// The duty cycle is the percentage of ON time:
// Duty cycle = (OCR1A+1)/(OCR1C+1) = 21/201 = 10%.
//
// For a 50% duty cycle, change OCR1A to 100.
// For an 80% duty cycle, change OCR1A to 160.
//

void setup()
{
  DDRB = 0x02; // set PB1 as an output
  TCCR1 = 0x5F; // configure control register, as above
  OCR1C = 200; // set LED blink rate about 2.4 Hz
  OCR1A = 20; // set 10% duty cycle (21/201 = 0.1)
}

void loop()
{
  // nothing!
}

//=====
// EXP 2A: PWM on Counter1, with higher frequency (audio) output.
// Modify the code above with the following values.
//
// void setup()
// {
//   DDRB = 0x02; // set PB1 as output; connect to piezo.
//   TCCR1 = 0x59; // prescaler is now /256
//   OCR1C = 70; // set output frequency to 440 Hz (A)
//   OCR1A = 34; // set 50% duty cycle
// }

```

```

//=====
// EXP #3: Sound! Fast PWM on Counter1, with complementary outputs
//
// Author: Bruce E. Hall
// Version: 1.0
// Date: 12 Dec 2013
// Purpose: Testing use of Timers with the Adafruit Trinket
// Hardware: ATTINY85
// Software: Arduino 1.0.5
//
// Here is an experiment for generating an audio tone.
//
// TCCR1 configuration for (fast) PWM mode:
//
// -----
// CTC1 PWM1A COM1A1 COM1A0 CS13 CS12 CS11 CS10
// 0 1 0 1 1 0 0 1
// -----
//
// Setting PWM1A will put this counter in PWM mode
// COM1A0 will activate output on OC1A (PB1 or digital pin 1)
// Setting CS13..CS11 will result in prescaler of /256
// Counter clock = 8 MHz/256 = 31250 Hz
// Output Frequency = 31250 Hz/(top+1) = 31250/71 = 440 Hz
//
// With PWM, the output pin is set when counter is 0, and then
// cleared when the value of OCR1A is reached:
// 0 -----> OCR1A -----> OCR1C (top)
// output ON output OFF
//
// The duty cycle is the percentage of ON time:
// Duty cycle = (OCR1A+1)/(OCR1C+1) = 35/71 = 49%
//
// An inverted output will be present on PB0, digital pin 0
// with a complementary duty cycle of 100-49 = 51%
//
// Connect a piezo buzzer in series with a 220 ohm resistor
// from either output to ground. 440 Hz is middle A.
// Now connect the buzzer+resistor BETWEEN the two outputs.
// The sound should be louder due to the larger voltage swing.

void setup()
{
  DDRB = 0x03; // PB0 & PB1 are outputs
  TCCR1 = 0x59; // configure control register, as above
  OCR1C = 70; // set output frequency to 440 Hz (A)
  OCR1A = 34; // set 50% duty cycle
}

void loop()
{
  // nothing!
}

```

```

//=====
// EXP #4: Sound, using Fast PWM mode 3 on Counter0
//
// Author:      Bruce E. Hall
// Version:    1.0
// Date:       12 Dec 2013
// Purpose:    Testing use of Timers with the Adafruit Trinket
// Hardware:   ATTINY85
// Software:   Arduino 1.0.5
//
// TCCR0A configuration:
// -----
// COM0A1  COM0A0  COM0B1  COM0B0  ---  ---  WGM01  WGM00
// 1       0       0       0       0   0   1       1
// -----
//
// TCCR0B configuration:
// -----
// FOC0A  FOC0B  ---  ---  WGM02  CS02  CS01  CS00
// 0       0       0   0   0       0   1       1
// -----
//
// Setting WGM02..WGM00 = 3 will put this counter in PWM mode 3.
// COM0A1 will activate output on OCOA (PB0 or digital pin 0)
// Setting CS02..CS00 = 3 will result in prescaler of 64.
// Counter clock = 8 MHz/64 = 125000 Hz.
// Freq output = 125000/256 = 488 Hz.
//
// With fast PWM mode 3, the output pin is set at zero & reset
// when the value of OCR1A is reached:
// 0 -----> OCR1A ----->255->0
// output HI          output LOW
//
// The frequency is fixed at 8 MHz/prescaler/256.
// The duty cycle = OCR1A/256.
//
// To hear the audio output, connect a piezo element between
// PB0/digital output #0 and ground.

void setup()
{
  DDRB = 0x01;          // set PB0 as an output
  TCCR0A = 0x83;        // toggle PB0, use fast PWM mode 3
  TCCR0B = 0x03;        // use /64 prescaler to set freq.
  OCR0A = 85;           // set duty cycle to 85/256 = 33%
}

void loop()
{
  // nothing!
}

```

```

//=====
// EXP #5: Sound, using Fast PWM mode 7 on Counter0
//
// Author:      Bruce E. Hall
// Version:    1.0
// Date:       12 Dec 2013
// Purpose:    Testing use of Timers with the Adafruit Trinket
// Hardware:   ATTINY85
// Software:   Arduino 1.0.5
//
// TCCR0A configuration:
// -----
// COM0A1  COM0A0  COM0B1  COM0B0  ---  ---  WGM01  WGM00
// 0        0        1        0      0    0     1     1
// -----
//
// TCCR0B configuration:
// -----
// FOC0A  FOC0B  ---  ---  WGM02  CS02  CS01  CS00
// 0        0      0    0     1     0     1     1
// -----
//
// Setting WGM02..WGM00 = 7 will put this counter in PWM mode 7.
// COM0B1 will activate output on OCOB (PB1 or digital pin 1)
// Setting CS02..CS00 = 3 will set the prescaler to 64.
// Counter clock = 8 MHz/64 = 125000 Hz.
// Freq output = 125000/201 = 622 Hz.
//
// With fast PWM mode 7, the output pin is set at zero & reset
// when the value of OCR0B is reached:
// 0 -----> OCR0B ----->OCR0A
// output HI      output LOW          counter reset
//
// The output frequency = 8 MHz/prescaler/(OCR0A+1).
// The duty cycle = (OCR0B+1)/(OCR0A+1).
//
// To hear the audio output, connect a piezo element between
// PB1/digital output #1 and ground.

void setup()
{
  DDRB = 0x02;          // set PB1 as an output
  TCCR0A = 0x23;        // configure control register A
  TCCR0B = 0x0B;        // configure control register B
  OCR0A = 200;          // set top of counter
  OCR0B = 100;          // set 50% duty cycle (101/201 = 0.50)
}

void loop()
{
  // nothing!
}

```

```

//=====
// EXP #6: Sound, using Phase-correct PWM (mode 5) on Counter0
//
// Author:      Bruce E. Hall
// Version:    1.0
// Date:       12 Dec 2013
// Purpose:    Testing use of Timers with the Adafruit Trinket
// Hardware:   ATTINY85
// Software:   Arduino 1.0.5
//
// TCCR0A configuration:
// -----
// COM0A1  COM0A0  COM0B1  COM0B0  ---  ---  WGM01  WGM00
// 0        0        1        0        0    0    0        1
// -----
//
// TCCR0B configuration:
// -----
// FOC0A  FOC0B  ---  ---  WGM02  CS02  CS01  CS00
// 0        0        0    0    1        0    1    1
// -----
//
// Setting WGM02..WGM00 = 5 will put this counter in PWM mode 5.
// COM0B1 will activate output on OCOB (PB1 or digital pin 1)
// Setting CS02..CS00 = 3 will result in prescaler of 64.
// Counter clock = 8 MHz/64 = 125000 Hz.
// Freq output = 125000/101/2 = 619 Hz.
//
// With phase-correct PWM mode 5, the output pin is set at zero &
// reset when the value of OCR0B is reached. The counter continues
// to top (OCR0A), then reverses direction & decrements. When it
// it hits OCR0B on the way down, the output goes HI again:
// 0 -----> OCR0B ----->OCR0A-----> OCR0B -----> 0
// output HI   output LOW   reverse!   output HI   reverse!
//
// The output frequency = 8 MHz/prescaler/(OCR0A+1)/2.
// The duty cycle = (OCR0B+1)/(OCR0A+1).
//
// To hear the audio output, connect a piezo element between
// PB1/digital output #1 and ground.

void setup()
{
  DDRB = 0x02;          // set PB1 as an output
  TCCR0A = 0x21;        // configure control register A
  TCCR0B = 0x0B;        // configure control register B
  OCR0A = 100;          // set top of counter
  OCR0B = 50;           // set 50% duty cycle (51/101 = 0.50)
}

void loop()
{
  // nothing!
}

```