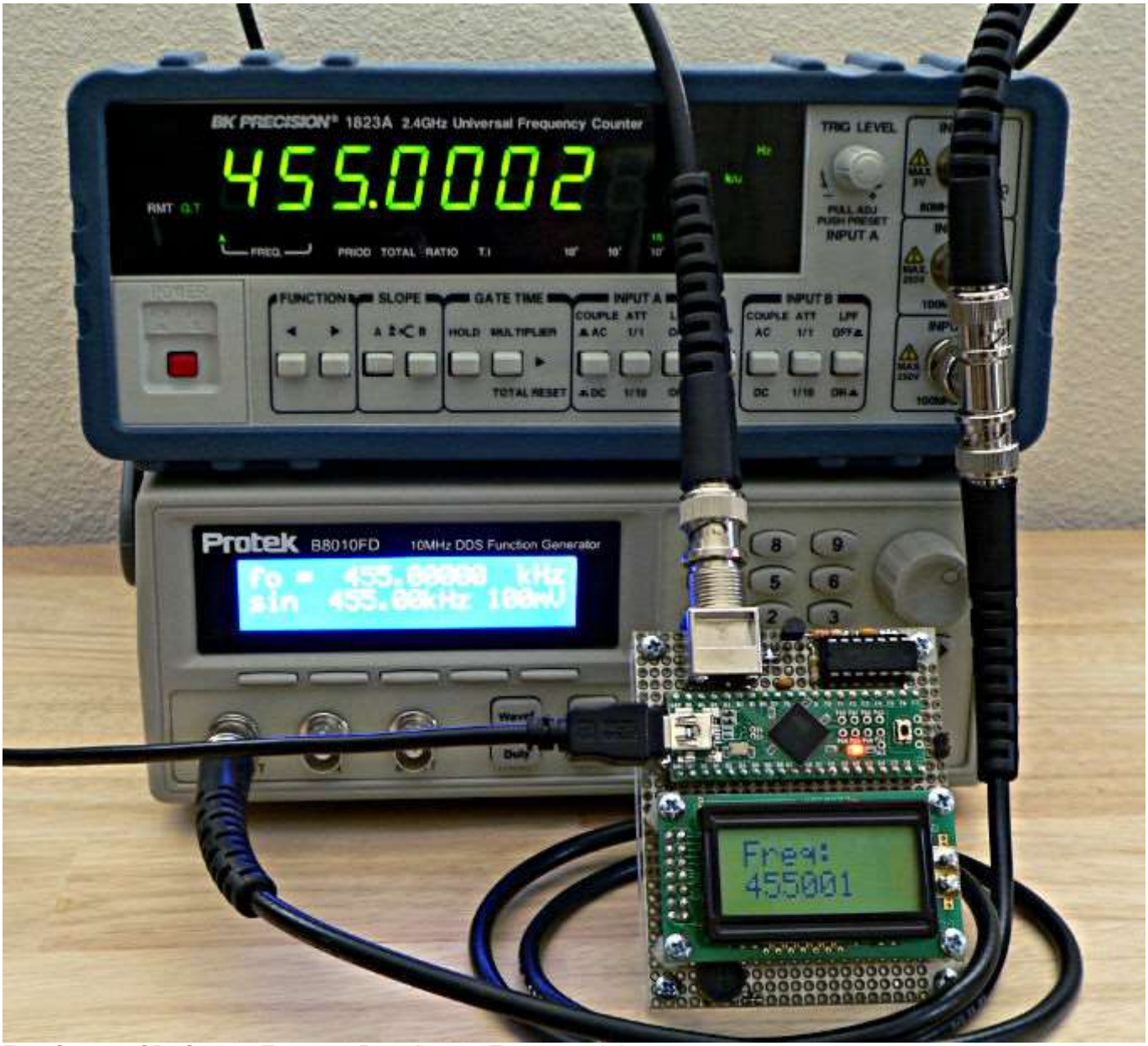
FreqCount Library

FreqCount measures the frequency of a signal by counting the number of pulses during a fixed time. See FreqCount vs FreqMeasure below to choose the best library.

Download: Included with the Teensyduino Installer
Latest Developments on Github



FreqCount LCD_Output Example Running on Teensy++ 2.0

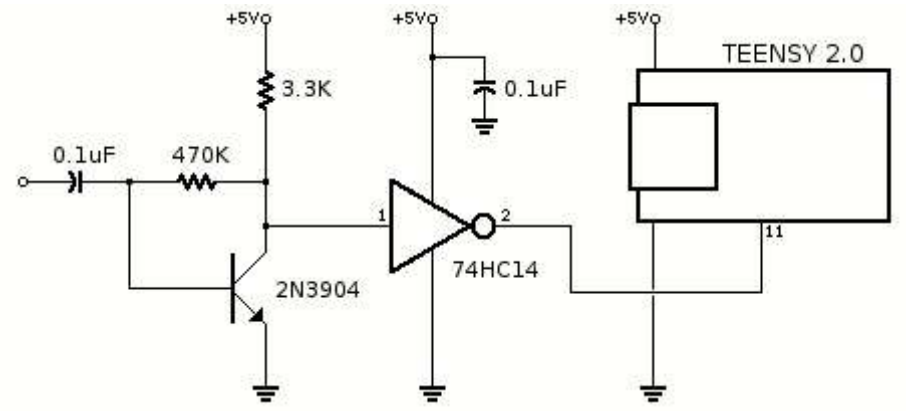
Hardware Requirements

FreqCount requires the input frequency as a digital level signal on a specific pin.

Board	Frequency Input Pin	Pins Unusable with analogWrite()
Teensy 3.5, 3.6	13	-
Teensy 3.0, 3.1, 3.2	13	-
Teensy LC	13	-

Teensy 2.0	11	4, 10, 12, 14, 15
Teensy++ 2.0	6	1, 24, 25, 26, 27
Teensy++ 1.0	6	1, 24, 25, 26, 27
Arduino Uno	5	3, 9, 10, 11
Arduino Leonardo	12	6, 9, 10, 13
Arduino Mega	47	9, 10, 44, 45, 46
Sanguino	1	12, 13, 14, 15

The Teensy and Teensy++ boards have a LED connected to the input pin. If the input signal is unable to drive the LED, a buffer must be used. The 74HC14 is a good choice.



An amplifier may be needed if the input signal is a sine wave or small AC signal which can not directly drive a TTL logic level input.

Basic Usage

FreqCount.begin(GateInterval);

Begin frequency counting. GateInterval is the time in milliseconds for each measurement. Using 1000 provides direct frequency output without mulitplying or dividing by a scale factor.

FreqCount.available();

Returns true when a new measurement is available. Only a single measurement is buffered, so it must be read before the next gating interval.

FreqCount.read();

Returns the most recent measurement, an unsigned long containing the number of rising edges seen within the gating interval. There is no delay between gating intervals, so for example a 1000.5 Hz input (and perfectly accurate crystal osciallator) will alternate between 1000 and 1001 when used with a 1 second gate interval.

FreqCount.end();

Stop frequency counting. PWM (analogWrite) functionality may be used again.

Example Program

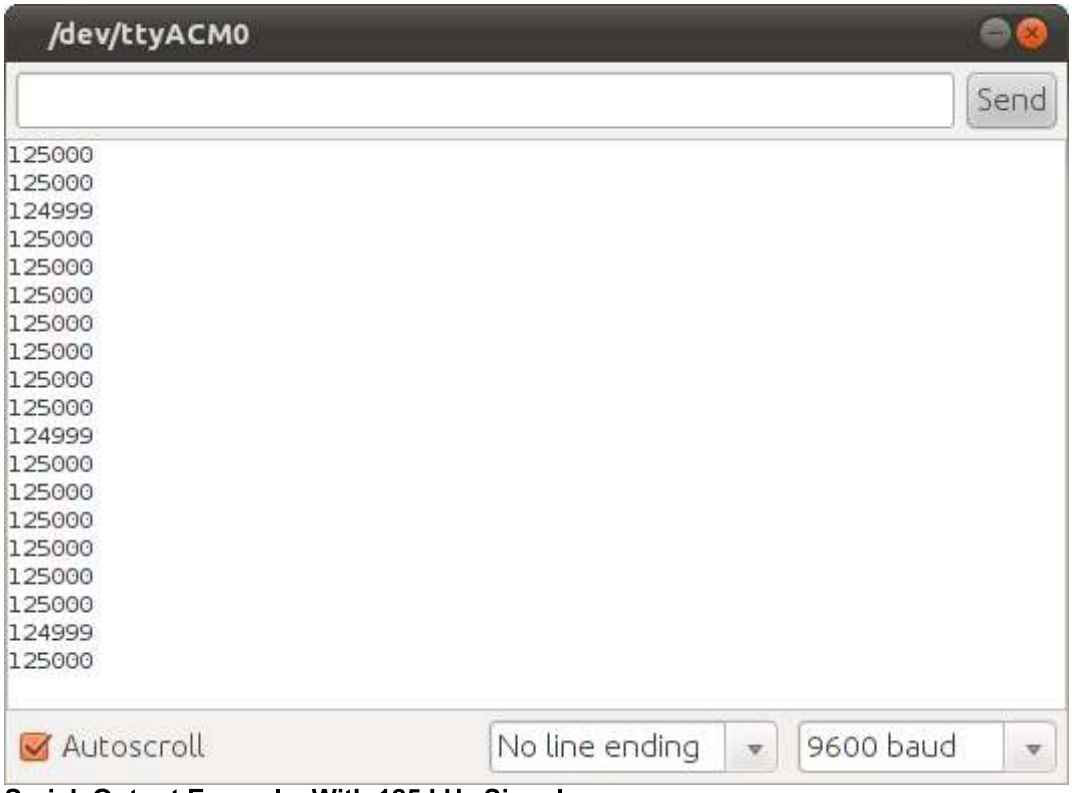
Open from the menu: **File > Examples > FreqCount > Serial_Output**

```
#include <FreqCount.h>

void setup() {
  Serial.begin(57600);
  FreqCount.begin(1000);
}

void loop() {
  if (FreqCount.available()) {
```

```
        unsigned long count = FreqCount.read();
        Serial.println(count);
    }
}
```



Serial_Output Example, With 125 kHz Signal

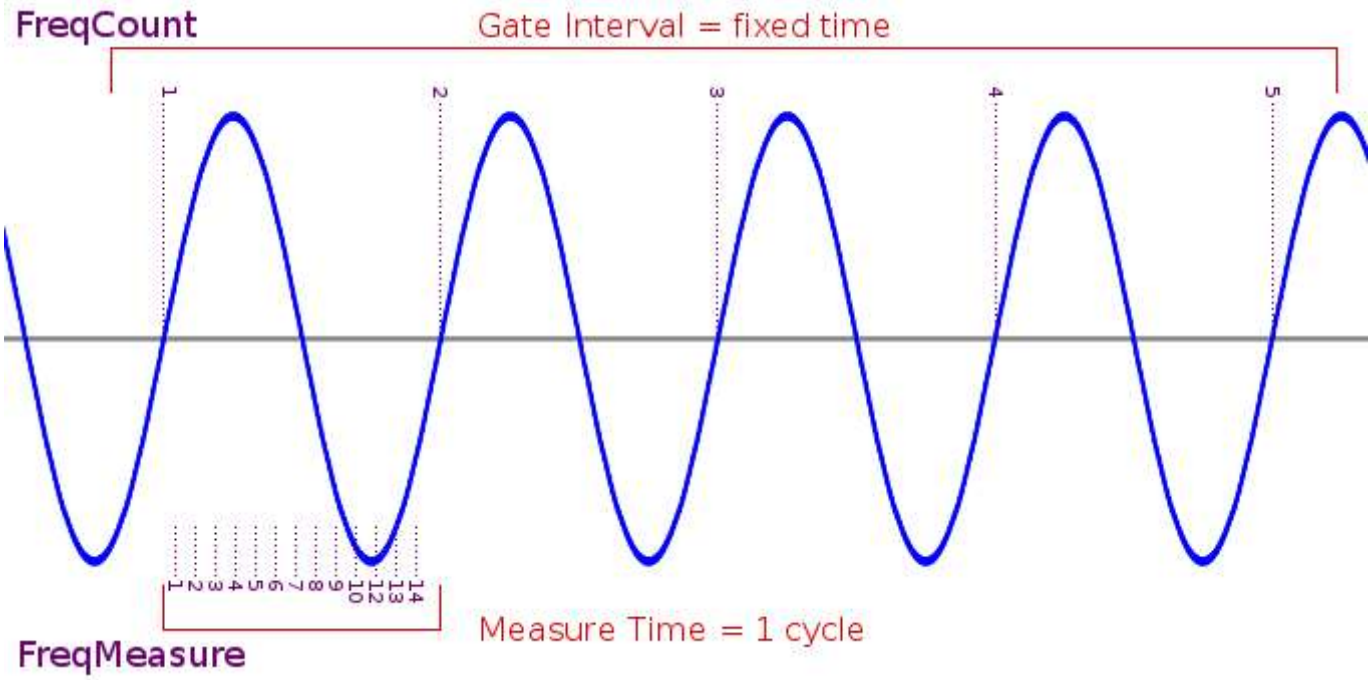
Interrupt Latency Requirements

FreqCount uses a timer interrupt for the gate interval. If another interrupt is running, or interrupts are disabled by the main program, response to the timer could be delayed. That will lengthen the gate interval, perhaps leading to counting more cycles. The next gate interval will be shorter (if normal interrupt response occurs), so a corresponding decrease will occur in the next measurement.

During USB startup on Teensy, activity from the PC can cause interrupts which interfere with FreqCount's gate interval. Other libraries which disable interrupts for long times, such as NewSoftSerial, can cause trouble.

FreqCount vs FreqMeasure

FreqCount: best for 1 kHz to 8 MHz (up to 65 MHz with Teensy 3.0 & 3.1)
FreqMeasure: best for 0.1 Hz to 1 kHz



FreqCount measures the number of cycles that occur during a fixed "gate interval" time. This works well for relatively high frequencies, because many cycles are likely to be counted during gate interval. At lower frequencies, very few cycles are counted, giving limited resolution.

FreqMeasure measures the elapsed time during a single cycle. This works well for relatively low frequencies, because a substantial time elapses. At higher frequencies, the short time can only be measured at the processor's clock speed, which results in limited resolution.

Other Frequency Measurement

FreqCount development has moved to this GitHub repository:
https://github.com/PaulStoffregen/FreqCount

Martin Nawrath's FreqCounter and FreqPeriod are similar to FreqCount and FreqMeasure. I originally ported FreqCounter to Teensy, but could not get it to work reliably. It did work, but had trouble at certain frequencies, and requires a "compensation" factor for accurate results. Ultimately I wrote a FreqCount from scratch, using a thin hardware abstraction layer for easy porting to different boards, and accurate results without a compensation factor. I also used a more Arduino-like API (begin, available, read) and designed for continuously repeating measurements. I did not try to use FreqPeriod.