PJRC Store

- Teensy 4.1, $26.85
- Teensy 4.0, $19.95
- Teensy 3.6, $29.25
- Teensy 3.5, $24.25
- Teensy 3.2, $19.80
- Teensy LC, $11.65
- Teensy 2.0, $16.00
- Teensy++ 2.0, $24.00

Teensy

- Main Page
- **Hardware**
- **Getting Started**
- **Tutorial**
- **How-To Tips**
- **Code Library**
- Projects
- **Teensyduino**
  - Main
  - Download+Install
  - Basic Usage
  - Digital I/O
  - PWM & Tone
  - **Timing**
  - USB Serial
  - USB Keyboard
  - USB Mouse
  - USB Joystick
  - USB MIDI
  - USB Flight Sim
  - Serial
  - **Libraries**
    - Main List
    - GLCD
    - LiquidCrystal
    - OctoWS2811
    - FastSPI_LED
    - Matrix/Sprite
    - LedDisplay
    - LedControl
    - DogLcd
    - ST7565
    - AltSoftSerial
    - NewSoftSerial
    - SoftwareSerial
    - MIDI
    - PS2Keyboard
    - DmxSimple
    - Firmata
    - Wire
    - SPI
    - OneWire
    - XBee
    - VirtualWire
    - X10
    - IRremote
    - TinyGPS
    - USBHostShield
    - Ethernet
    - Bounce
    - Keypad
    - **Audio**
    - Encoder
    - Ping
    - CapacitiveSensor
    - FreqCount
    - FreqMeasure
    - Servo
    - PulsePosition
    - Stepper

FreqMeasure Library

FreqMeasure measures the elapsed time during each cycle of an input frequency. See FreqCount vs FreqMeasure below to choose the best library. FreqMeasure works well for RPM (rotations per minute) tachometer applications.

Download: Included with the Teensyduino Installer
Latest Developments on Github

Update: FreqMeasureMulti can measure up to 8 frequencies simultaneously on Teensy 3.x, or up to 6 signals on Teensy LC.

FreqMeasure LCD_Output Example Running on Teensy++ 2.0

Hardware Requirements

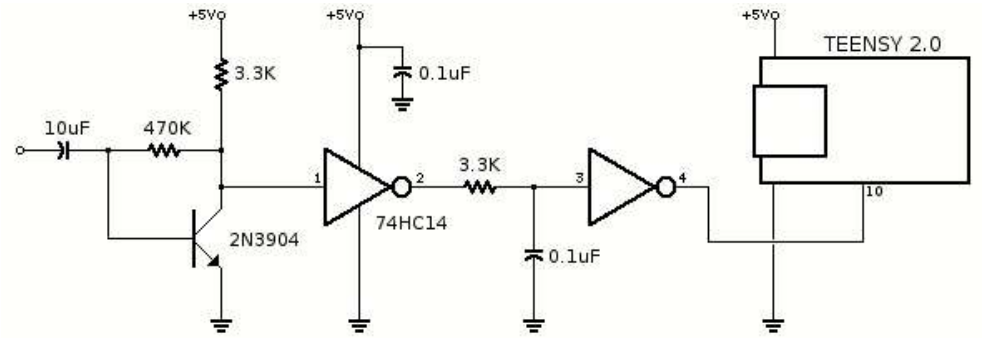
FreqMeasure requires the input frequency as a digital level signal on a specific pin.

Board	Frequency Input Pin	Pins Unusable with analogWrite()
Teensy 4.0, 4.1	22	23
Teensy 3.5, 3.6	3	4
Teensy 3.0, 3.1, 3.2	3	4
Teensy LC	16	17

Teensy 2.0	10	9
Teensy 1.0	16	15, 17, 18
Teensy++ 2.0	17	14, 15, 16
Teensy++ 1.0	17	14, 15, 16
Arduino Uno	8	9, 10
Arduino Leonardo	13	5
Arduino Micro	13	5
Arduino Mega	49	6, 7, 8
Sanguino	14	12, 13

See below for other boards

If the input signal may have noise or could be a high frequency, adding a low pass filter is a good idea.



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

FreqMeasure.begin();
: Begin frequency measurement.

FreqMeasure.available();
: Returns the number of measurements available to read, or 0 (false) if none are unread. If your program spends time on other tasks and a relatively fast waveform is being measured, several readings may be available.

FreqMeasure.read();
: Read a measurement. An unsigned long (32 bits) containing the number of CPU clock cycles that elapsed during one cycle of the waveform. Each measurement begins immediately after the prior one without any delay, so several measurements may be averaged together for better resolution.

FreqMeasure.countToFrequency(count);
: Convert the 32 bit unsigned long numbers from read() to actual frequency.

FreqMeasure.end();
: Stop frequency measurement. PWM (analogWrite) functionality may be used again.

Example Program

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

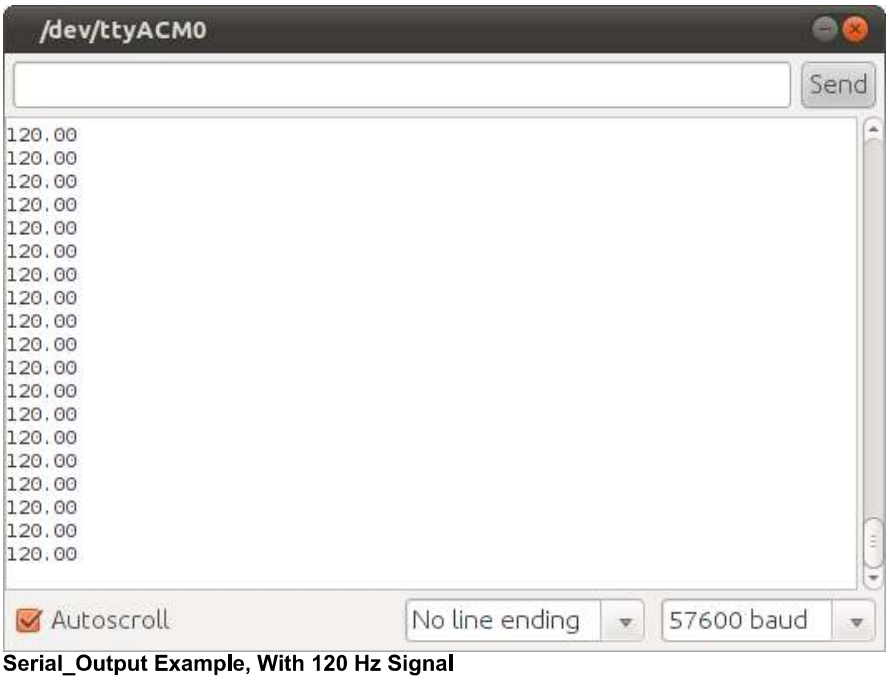
```
#include <FreqMeasure.h>

void setup() {
  Serial.begin(57600);
  FreqMeasure.begin();
}

double sum=0;
int count=0;

void loop() {
  if (FreqMeasure.available()) {
    // average several reading together
    sum = sum + FreqMeasure.read();
    count = count + 1;
    if (count > 30) {
      float frequency = FreqMeasure.countToFrequency(sum / count);
```

```
        Serial.println(frequency);
        sum = 0;
        count = 0;
      }
    }
  }
```



Serial_Output Example, With 120 Hz Signal

Zero Handling

Because FreqMeasure works on a per-cycle time frame, it is impossible to directly measure zero frequency. When displaying frequency, such as the LCD_Output example, if the input frequency stops, the most recent measurement will remain on the display.

To detect zero, a timeout must be implemented. On simple approach would be to record the millis() time when FreqMeasure.available() returns true. When it returns false, check if too much time as elapsed and update the output to show zero.

CPU Requirements

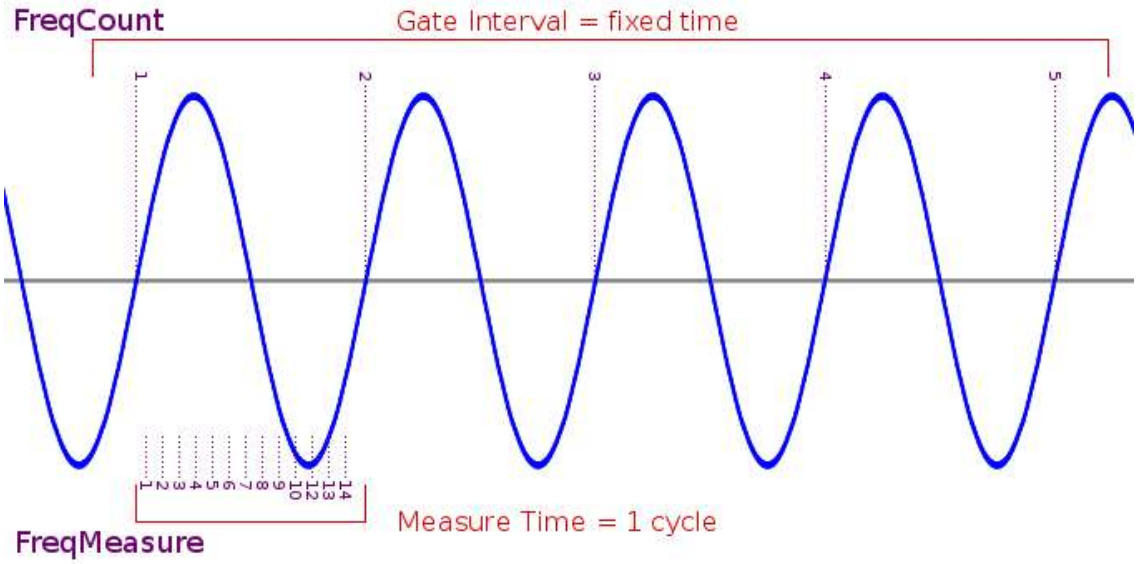
At the end of each cycle, an interrupt routine runs. The actual capture of elapsed time is done by hardware, so some interrupt latency is acceptable.

At relatively low frequencies, under 1 kHz, only minimal CPU time is used. However, as the frequency increases, the interrupt demands more CPU time. A hardware low-pass filter is recommended if the input frequency could possibly be much higher than several kHz.

If interrupts are disabled for more than 1 cycle of the waveform, the measurement can span 2 or more cycles. Libraries which disable interrupts for long times (eg, NewSoftSerial) can be problematic.

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 Arduino Compatible Boards

FreqMeasure can be used on Arduino, Sanguino and probably other boards. See the file util/capture.h for details to port to other AVR-based boards.

Arduino Mega's pin 49 may be used by the SPI library, or SPI-based libraries like Ethernet or SD. You may need to edit FreqMeasure's util/capture.h to use ICP5 (pin 48) instead of ICP4.

Seeeduino GPRS requires editing FreqMeasureCapture.h. This this issue report for details.

Other Frequency Measurement

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.