Pro Trinket Tachometer

Created by Bill Earl
Table of Contents

Overview 5

Circuit Design 6
  • Components:

Code Design 10
  • Libraries and Variables:
  • Setup():
  • The Interrupt Handler:
  • The Loop:
  • The Complete Code:

Install and Use 13
  • Lathe Mounting:
  • Drill Press Mounting:
  • Band Saw Mounting:
  • Enclosure and Display Mounting:
Overview

Cutting tools and drill bits work best when run at the correct speed for the material being cut. Too fast or too slow can result in inefficient cutting, rough surfaces, heat buildup, premature dulling and/or tool breakage. References such as Machinery's Handbook have extensive tables of "Feeds and Speeds" for various types of cutting tools and materials.

But how do you know how fast your machine is running? Some machines have a variable speed control, but no precise feedback on the actual speed. The Pro Trinket Tachometer attaches to your machine to sense and display the actual rotation rate.
Materials Required:

- 5v Pro Trinket
- 7-Segment Display with i2c backpack
- Reflective IR sensor with 470 ohm resistor
- Wire
- Soldering Supplies

Recommended or Optional Materials

(as needed for installation on your machine)

- Power Supply
- Barrel Jack
- Project box or other enclosure
- Black Tape
- White Paint

Circuit Design

Components:

The circuit consists of 3 major components:

- Reflective sensor
- Pro Trinket microcontroller
- 7-Segment Display w/backpack

You will also need a power supply, enclosure, connectors, wire and soldering supplies.
The Sensor:

An optical reflective sensor is easily adapted to work with most machines. It only requires a light/dark target pattern on the rotating part to produce a signal. The target pattern can usually be added safely and easily with tape and/or paint.

You will also need a 470 ohm current limiting resistor for the emitter. This resistor is included in the package when you purchase the sensor.

Since the Pro Trinket has internal 20K pullup resistors, we won't be needing the 10K pullup resistor packed with the sensor. You can save this in your parts bin for future projects.
It is also possible to use a Hall Effect sensor and a magnet. But this can be dangerous for machines operating at high speeds. A magnet can turn into a projectile if it detaches from the shaft at high speed.

The Processor:

Most any small microcontroller has the horsepower necessary to implement a tachometer. We’re using a Pro Trinket here because it is inexpensive, small and easy to work with. We'll use the i2c interface to drive a 7-segment led display, and the external interrupt on pin 3 as a trigger for timing the revolutions.
The Display:

Other than routers and rotary tools like a Dremel™, most shop machinery runs at 3,000 RPM or less. So a 4-digit display is sufficient. The 4-digit 7-segment display backpack makes these simple to wire and control via the i2c bus.
Code Design

The Tachometer code is relatively simple. We'll explain each section in detail. But if you just want to get up and running, you can skip down to "The Complete Code" and continue from there.

Libraries and Variables:

We'll start by including the necessary libraries, and defining some global variables. Note that some of these are defined as 'volatile'. These variables are accessed both in the main loop and in the interrupt handler. The 'volatile' modifier instructs the compiler that these variables can change at any time and it should not attempt to optimize the code it generates for them.

```cpp
#include <Wire.h>
#include "Adafruit_LEDBackpack.h"

// External interrpt pin for sensor
#define interruptPin 3

Adafruit_7segment matrix = Adafruit_7segment();
long lastUpdate = 0; // for timing display updates
volatile long accumulator = 0; // sum of last 8 revolution times
volatile unsigned long startTime = 0; // start of revolution in microseconds
volatile unsigned int revCount = 0; // number of revolutions since last display update
```

Setup():

In the setup code, we initialize the display, attach the interrupt handler and set the input mode of the pin we are using for the sensor.

The pinMode() function sets the mode to Input, but also enables the internal pullup resistor. This assures that the sensor reading will be a logic HIGH until it detects a reflection from the target on the drive spindle.

The call to attachInterrupt() specifies the interrupt pin we are using, the address of the interrupt handler and whether we want to trigger on the rising or falling edge of the signal. Since we are just counting one pulse per revolution, we don't really care if it is triggered on the rising edge or falling edge.

```cpp
// Setup - runs once at startup
// void setup()
{
```
The Interrupt Handler:

The interrupt handler is invoked once per revolution as triggered by the sensor. It calculates the number of microseconds since the last interrupt and adds that to an ‘accumulator’. But before adding, we subtract 1/8 of the accumulator value. So when the speed stabilizes, the accumulator will converge on the sum of the times for the last 8 revolutions.

```cpp
// Interrupt Handler
// IR reflective sensor - target passed
// Calculate revolution time
void tach_interrupt()
{
    // calculate the microseconds since the last interrupt
    long usNow = micros();
    long elapsed = usNow - startTime;
    startTime = usNow;  // reset the clock

    // Accumulate the last 8 interrupt intervals
    accumulator -= (accumulator >> 3);
    accumulator += elapsed;
    revCount++;
}
```

The Loop:

Once per second, the loop code calculates the rpm based on the accumulator value and updates the display.

```cpp
// Main Loop - runs repeatedly.
// Calculate RPM and Update LCD Display
void loop()
{
    if (millis() - lastUpdate &gt; 1000) // update every second
    {
        unsigned int rpm = 0;
        // divide number of microseconds in a minute, by the average interval.
        if (revCount &gt; 0)
        {
            rpm = 60000000 / (accumulator &gt;= 3);  // divide
        }
        Serial.println(rpm);
    }
```
matrix.println(rpm);
matrix.writeDisplay();

lastUpdate = millis();
revCount = 0;
}

The Complete Code:

Copy and paste this into your Arduino IDE and upload to the Pro Trinket.

#include <Wire.h>
#include "Adafruit_LEDBackpack.h"

#define interruptPin 3

Adafruit_7segment matrix = Adafruit_7segment();

long lastUpdate = 0;  // for timing display updates
volatile long accumulator = 0;  // sum of last 8 revolution times
volatile unsigned long startTime = 0; // start of revolution in microseconds
volatile unsigned int revCount = 0; // number of revolutions since last display update

void setup()
{
    Serial.begin(115200);
    matrix.begin(0x70);

    pinMode(interruptPin, INPUT_PULLUP);
    attachInterrupt(digitalPinToInterrupt(interruptPin), tach_interrupt, FALLING);
}

void loop()
{
    if (millis() - lastUpdate &gt; 1000) // update every second
    {
        unsigned int rpm = 0;
        // divide number of microseconds in a minute, by the average interval.
        if (revCount &gt; 0)
        {
            rpm = 60000000 / (accumulator&gt;&gt;3);
        }
        Serial.println(rpm);
        matrix.println(rpm);
        matrix.writeDisplay();
        lastUpdate = millis();
        revCount = 0;
    }
// Interrupt Handler
// IR reflective sensor - target passed
// Calculate revolution time
void tach_interrupt()
{
    // calculate the microseconds since the last interrupt
    long usNow = micros();
    long elapsed = usNow - startTime;
    startTime = usNow;  // reset the clock

    // Accumulate the last 8 interrupt intervals
    accumulator -= (accumulator &gt;&gt; 3);
    accumulator += elapsed;
    revCount++;
}

Install and Use

This is where you may need to get a little creative. Every machine is different, so you will need to examine your machine to find a good place to mount the sensor and display.

If the controls on your machine run on low-voltage DC, you may be able to tap into that to power your tach. Otherwise, you will need a separate DC power supply. The Pro Trinket can run on anything in the 5v-12v range.

You will also need an enclosure of some sort for the processor and display. There are lots of small project boxes that could work well here. Or, if you are inspired, you could design a custom 3D printed one to fit your machine.

Most machinery has one or more belts and pulleys between the motor and the final drive spindle. These may increase or decrease the speed relative to the motor. Ideally, the sensor should be attached so that it can monitor the rotation of the final drive spindle. Below are a few examples:

- Make sure that all components are securely mounted and will not interfere with normal operation.
- Route all wiring away from any moving parts and tie down securely to avoid entanglements.
Lathe Mounting:
For this small lathe, we created a target pattern on the headstock handwheel using black photographer's tape.

We left one small gap in the tape so that the chromed surface would reflect IR back to the sensor once per revolution.

The sensor is mounted to a strip of brass spring-stock and positioned just a couple millimeters from the target on the handwheel.
Drill Press Mounting:
For this 17" drill press, the final drive pulley was already painted black. So we just added a white patch* on the edge.

The sensor is screwed to the base of the pulley shroud - using a small wooden spacer block to align the sensor with the edge of the pulley.

The display is mounted to the base of the pulley shroud - just above eye level.

*Tip: Wite-Out ™ or similar error correcting fluid is quick drying and opaque - which makes it perfect for this application.

Band Saw Mounting:
The main drive wheel on this 14" band-saw was also conveniently pre-painted black. So all we needed was a spot of white error-correcting fluid on the edge.

The sensor is screwed to a wooden block which is sized to position it a couple millimeters from the rim.

Note: For a band-saw, you will be interested in the linear blade speed rather than the RPM. To calculate the blade speed in surface feet per minute (SFPM) multiply the RPM by the circumference of the drive wheel as in the code below:
if (revCount &gt; 0)
{
    rpm = 6000000 / (accumulator&amp;gt;&amp;gt;3);
}
// calculate SFPM assuming diameter is in inches
int sfpm = (rpm * diameter * Pi) / 12;
Serial.println(sfpm);
matrix.println(sfpm);
matrix.writeDisplay();

Enclosure and Display Mounting:
The best location for the display will vary from machine to machine. Choose a location with good visibility - but out of the way so that it does not create a safety hazard or interfere with normal machining operations.

For the prototype, I've mounted all the electronics (except for the sensor) in a small translucent project box (Hammond #1591). This is affixed to the base of the lathe near the power switch. Wiring is routed under the chassis and away from any moving parts.