99 000 RPM Contact-Less Digital Tachometer

Posted on January 23, 2008, by Ibrahim KAMAL, in Sensor & Measurement, tagged

This article describes how to build a contact -less tachometer (device used to count the revolutions per minute of a rotating shaft) using a 8051 microcontroller and a proximity sensor.

As the name implies, what makes this device special, is that it can very accurately measure the rotational speed of a shaft without even touching it. This is very interesting when making direct contact with the rotating shaft is not an option or will reduce the velocity of the shaft, giving faulty readings.

This device is built on an AT89S52 (or AT89C52) microcontroller, an alpha-numeric LCD module and and a proximity sensor to detect the rotation of the shaft whose speed is being measured.

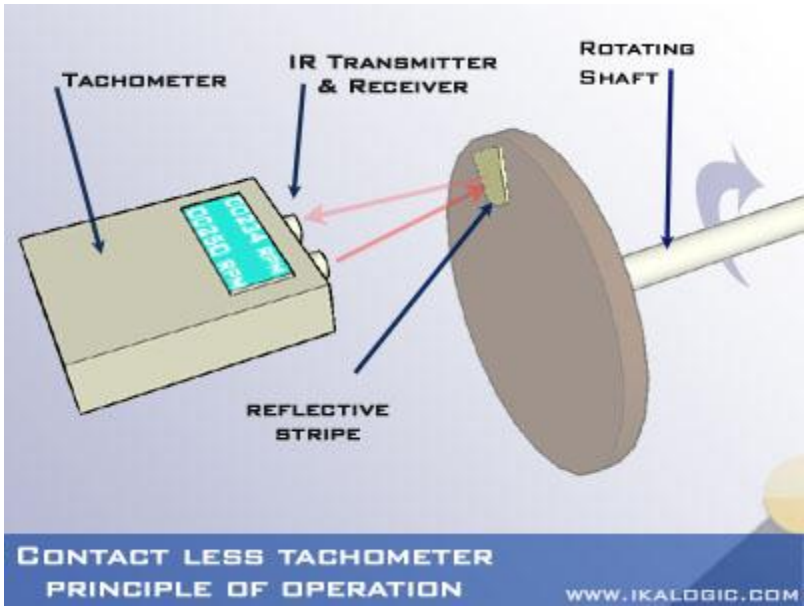
A 600 mA.h Ni-Cd battery provides months of regular use of this device before it needs to be recharged.

Key Features

- Measures up to 99 000 RPM
- Instantaneous measurement
- Automatic DATA Hold Function
- LCD display
- Ni-Cad Rechargeable battery

Important: this tachometer uses a proximity sensor. In case you don't know how to make a proximity sensors, and/or how to operate them, please refer to this article first.

Contact less tachometer principle of operation



The idea behind most digital counting device, frequency meters and tachometers, is a micro-controller, used to count the pulses coming from a sensor or any other electronic device.

In the case of this tachometer, the counted pluses will come from proximity sensor, which will detect any reflective element passing infront of it, and thus, will give an output pulse for each and every rotation of the shaft, as show in the picture. Those pulses will be fed to the microcontroller and counted.

To understand how a micro controller counts pulses, and deduce the frequency of those pulse, please refer to ***this tutorial about building a frequency meter***, that elaborates the process of frequency counting.

The main difference between this tutorial about tachometer and frequency meters, is that we need the reading in pulses per minutes (to count revolutions per minutes), but in the same time, we don't want to wait a whole minute before getting a correct reading. Thus we need some additional processing to predict the number of revolutions per minute in less than a second.

Instantaneous measurement algorithm

To be able to deduce an RPM reading in less than second, while constantly refining the reading's accuracy, a simple algorithm have been developed, where a counter and a timer are used. Counter and

timers are part of the internal features of a micro-controller, (like the AT89C52 used in this project) and they can be easily configured through programming.

The schematic below (***figure 1***), shows how the timer and the counter are used for this task; The counter is connected i such a way to count pulses coming from the proximity sensor, while the timer is used to precisely feed the counted value to the microcontroller every filth of a second, and reset the counter to 0. The microcontroller can now take an average of the last 3 readings (saved in C_1, C_2 and C_3) and calculate the average numbers of pulses per fifth second, then multiply this value by 5, to get the number of pulses per second, then multiply this value by 60 to get the number of pulses per minute, which represents the measured RPM. The only purpose of calculating an average reading is that it will allow to get more stable reading and prevent display flickering.

C_1, C_2 and C_3 are used to store the last three reading.

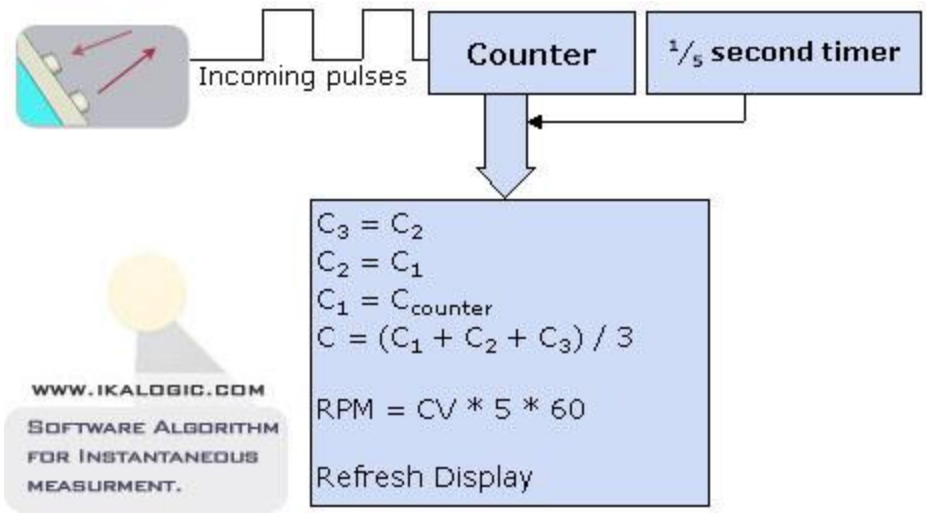
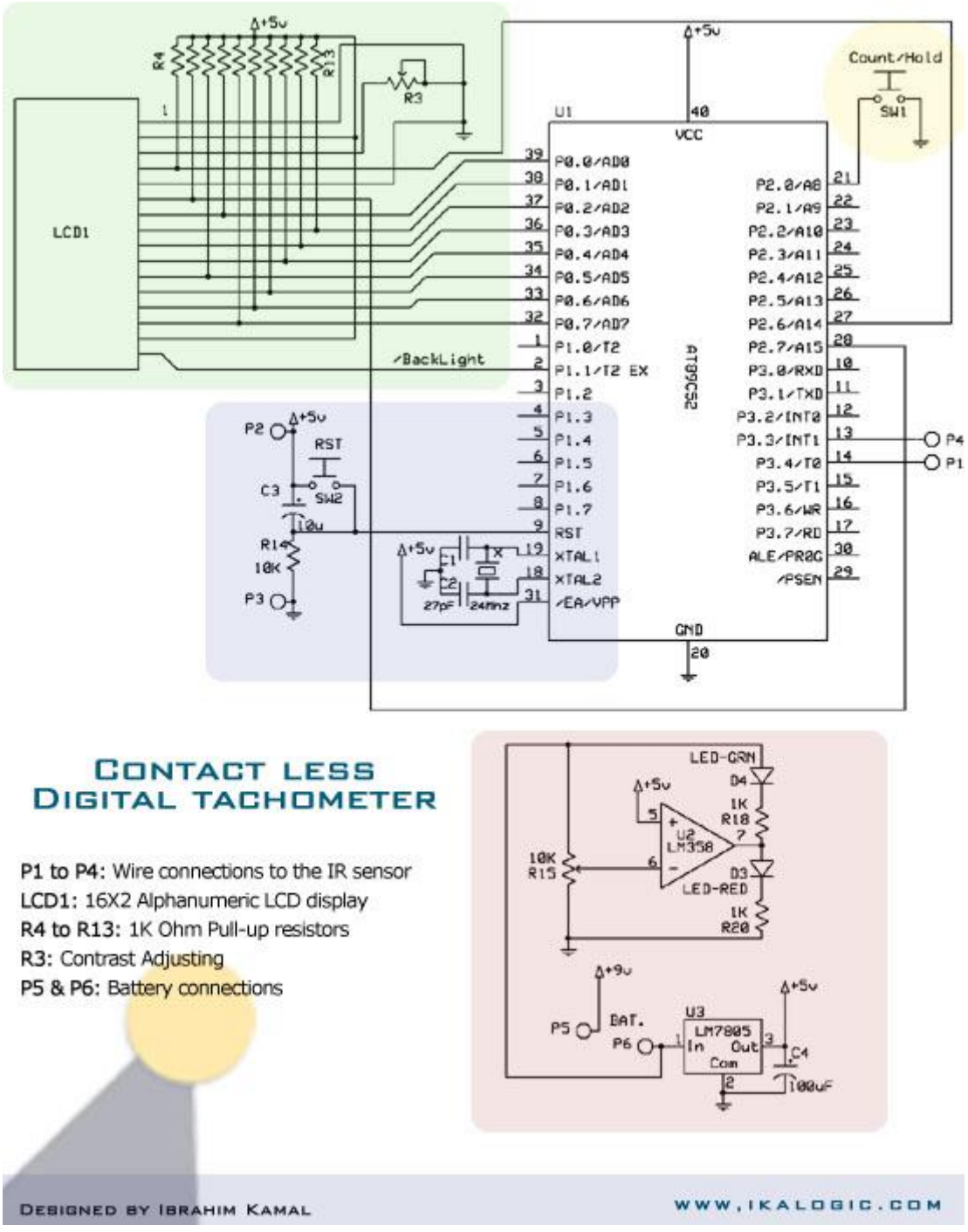


figure 1

The electronic Circuits

This device is composed of two electronic circuits: the Sensor, which is a slightly modified proximity sensor, and the microcontroller board, which analyses pulses coming from the sensor, process them and display the result on the LCD display.

The micro controller board:



Circuit explanation:

The LCD connections in the green shading is a standard for most of alpha numeric LCDs, the only feature I added is to be able to control the back light via the 80c52 micro controller. The LCD protocol can seem complicated to some of you, and an article should be released soon to explain it.

The part in the blue shading is also standard in any 8051 microcontroller circuit, which includes the reset circuitry along with the crystal resonator that generates the clock pulses required.

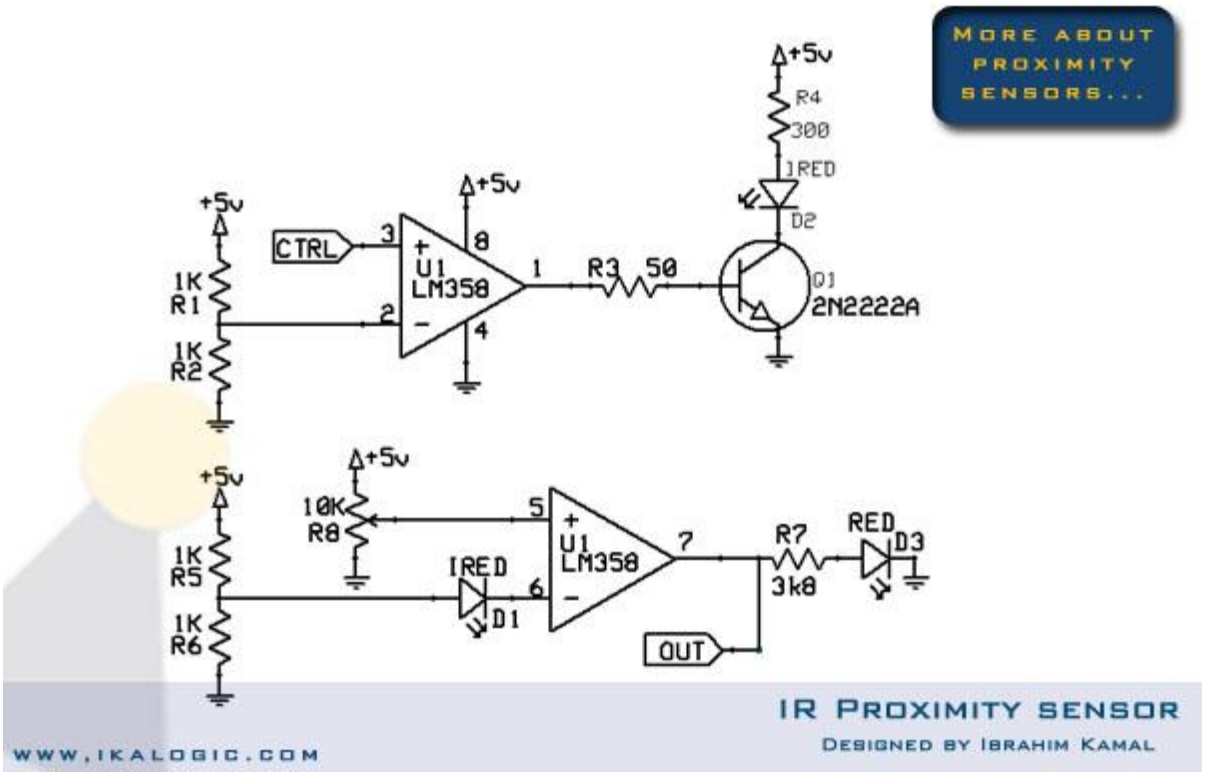
The power supply, shaded in light red, regulates a 9V rechargeable Ni-CD battery and also provides a very simple battery monitor, with a green and a red LED, showing whether the battery need to be recharged or not.

The switch SW_1, **shown in the upper yellow circle**, is used to enable/disable the measurement or the counting process. When the switch is pressed, the device measures the RPM of the shaft under test, and constantly updates the reading on the LCD, when the switch is released, the last reading is held unchanged on the display, as long as the device stays on. When the switch is pressed again the old reading is replaced by the new one.

The wire connection ***P₁***, which is connected to the output of the sensor, is connected to the pin 3.4 of the microcontroller, this pin has a dual function which is to count incoming pulses and increment a 8, 13, or 16 bit register according to the configuration of the timer T_0.

As you may have noticed, this schematics misses tow important items to be called a tachometer: The C code loaded into the microcontroller, which will be discussed later, and the proximity sensor, which will feed the pulses to be counted.

The modified IR proximity sensor



This schematic show the slight modification over the one proposed in this tutorial, which is the fact that the emitter LED uses a current limiting resistor of a higher value, to allow it to be turned on for a long period of time, because in this specific application, we need to turn the IR emissions on or off, but we don't need to inject high currents to reach high ranges… I recommend the reading of this article that fully covers all the aspects of this sensor.

The CTRL line, is an input coming from the microcontroller (*at the wire connection: P_4*), turning the IR emissions ON and OFF, and the OUT line, is the output of the sensor, which is fed to the microcontroller (*at the wire connection: P_1*).

After analyzing both the main board holding the microcontroller and the sensor, here is a simple diagram (***figure 2.A***) showing how they are connected together. You will have to refer to the above schematics to see where P_1, P_2, P_3 and P4 goes in the main board, as well as the other lines concerning the sensor.

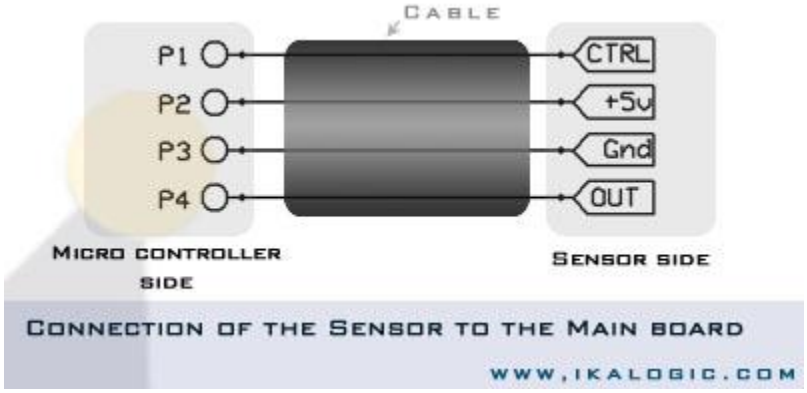


figure 2.A

This picture also shows what is meant by the connection of the sensor to the main board. The reason for separating the sensor from the main board, is to allow better performance sensors, or even other types of sensors to be connected to the device. In general, modular designs cost more, but is more useful in the prototyping phase…

The software

Here are ***only small relevant parts*** of the full C program, that was loaded into the microcontroller after being compiled to a HEX file. Those part of the code were selected as the ones that emphasize the main purpose of a microcontroller in such an application. For examples, function dealing with the LCD operation are not included in this description. Comments in green explains the program. The full code is available in the Project folder, downloadable at the bottom of this article .

```
#include <REGX51.h>
#include <math.h>


unsigned int clk_tmp,clk_tmp2,clk_sec,clk_sec2;
unsigned intex_pulses,rps,rps_tmp,temp,rps_avg,rps_max;
unsigned int rps_his[5];
char a,b,c,d,e;
unsigned char count1,count2;
unsigned char scale = 4;


delay(y){ // A function to make software delays
unsigned int i;
for(i=0;i<y;i++){;}
}
setup_interrupts(){ // This function initialises the TIMER and the COUNTER to
EA = 1;             // be used in in the trachometre
ET0 = 1;      //set the Timer/counter 0
TR0 = 1;      //Enable Timer/counter 0 to count
TMOD = 0X25;  //counter 0 in mode 1 (16 bit counter),
              //timer 1 in mode 2 (auto reload from TH1)
TH1 = 0;      //start counter from 0
ET1 = 1;      //enable timer 1
TR1 = 1;      //Enable Timer/counter 1 to count
PT0 = 1;      //Setup the priorities of timer 1 and timer 0, a 0 gives a
```

```
PT1 = 0;      //higher priority.
}

void int_to_digits(unsigned int number){ //store the 5 digits of an integer
float itd_a,itd_b;                       //number in the variable a,b,c,d,e
itd_a = number / 10.0;
e = floor((modf(itd_a,&itd_b)* 10)+0.5);
itd_a = itd_b / 10.0;
d = floor((modf(itd_a,&itd_b)* 10)+0.5);
itd_a = itd_b / 10.0;
c = floor((modf(itd_a,&itd_b)* 10)+0.5);
itd_a = itd_b / 10.0;
b = floor((modf(itd_a,&itd_b)* 10)+0.5);
itd_a = itd_b / 10.0;
a = floor((modf(itd_a,&itd_b)* 10)+0.5);
}

clk() interrupt 3        //timer 1 interrupt
{
clk_tmp++;          //Software counter for the timing of the tachometer
readings
clk_tmp2++;         //Software counter for the display refresh rate
if (clk_tmp2 > (1236)){  // update display
clk_tmp2 = 0;
rps_avg = floor(((rps_his[0] + rps_his[1] + rps_his[2] + rps_his[3] + ___
          ___rps_his[4])/5)*60);
}

if (clk_tmp > (6584/scale)){ // update the measured RPM
```

```
clk_tmp = 0;
if (P2_0 == 0){
rps = TL0;
temp = TH0;
temp = temp * 256;
rps = (rps + temp)* scale;
rps_his[4] = rps_his[3];
rps_his[3] = rps_his[2];
rps_his[2] = rps_his[1];
rps_his[1] = rps_his[0];
rps_his[0] = rps;
}
TL0 = 0;
TH0 = 0;
}
}

count_pulses() interrupt 1 //counter 0 interrupt
{
if (scale < 10)      // If the pulses are so fast that the internal counter
scale++;             // overflows, increase the variable 'scale' so that
}                    // so that readings are recorded at a higher rate

void main(){
   scale = 10 ;
   P3_3 = 0; // ini proximity sensor, OFF
   P3_4 = 1; // ini sensor input
   P1_1 = 0; //turn LCD backlight ON
   P2_0 = 1; //ini count/hold button
```

```
    ini_lcd(); // ini the LCD
    setup_interrupts();


    while(1){
        P3_3 = ~P2_0;
            if (P2_0 == 1){
            scale= 4;
            }
    }
}
```

To understand the functioning of this source code, you must have some basic microcontroller and C language skills. The variable ***scale*** is used to control the rate at which the tachometer reads and resets the counter.

The housing of the tachometer

For the housing, an old floppy disk drive case is used, where the tachometer and the battery fits perfectly. Here, those few pictures are worth a thousands words.


DIGITAL TACHOMETER SIDE VIEW


TACHOMETER BOTOM VIEW


TACHOMETER FRONT VIEW


TACHOMETER TOP VIEW

Source: http://www.ikalogic.com/99-000-rpm-contact-less-digital-tachometer/