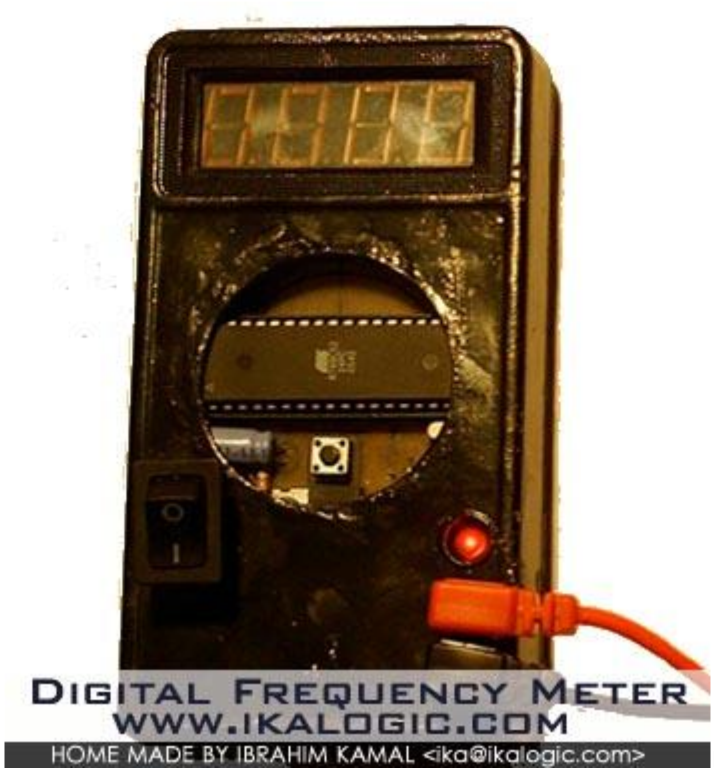
Digital (5hz to 500 Khz) Frequency-Meter

Posted on April 4, 2008, by Ibrahim KAMAL, in Sensor & Measurement, tagged

*Based on the famous AT89C52 microcontroller , this 500 Khz frequency-Meter will be enough to trace and debug most of your circuits, to adjust 555 timers frequency and perform all kind of frequency measurements in Digital circuits. And once you have this tool between your hand, you will get used to it to the point of no-return! and you will say "**remember the days we adjusted the frequency by trial and error till we get it..!**"*



Note: This tutorial assumes you already have some basic notions about microcontrollers and general electronics skills.

To understand the block diagram below, you should recall the simplest definition of frequency: "the number of occurrences within a given time period". What we are trying to do is to count the number of electric pulses during a time of one second. To do this we need a counter, to count the pulses, and a timer so that every 1000 milliseconds, the processor stop counting, calculate the frequency and display it, then start counting again from zero.

Below (***figure 1***) this is the principle of operation of a frequency meter, however, to build a reliable frequency meter, we will need to do some more mathematical operations. For example: Some operations will accurately predict the frequency before waiting for a whole second to elapse, which will also increase the refresh rate. Another minor upgrade is to to display the average of the last 5 reading rather than displaying the frequency instantaneously (which can cause lot of display flickering if the frequency being measured is not very stable).

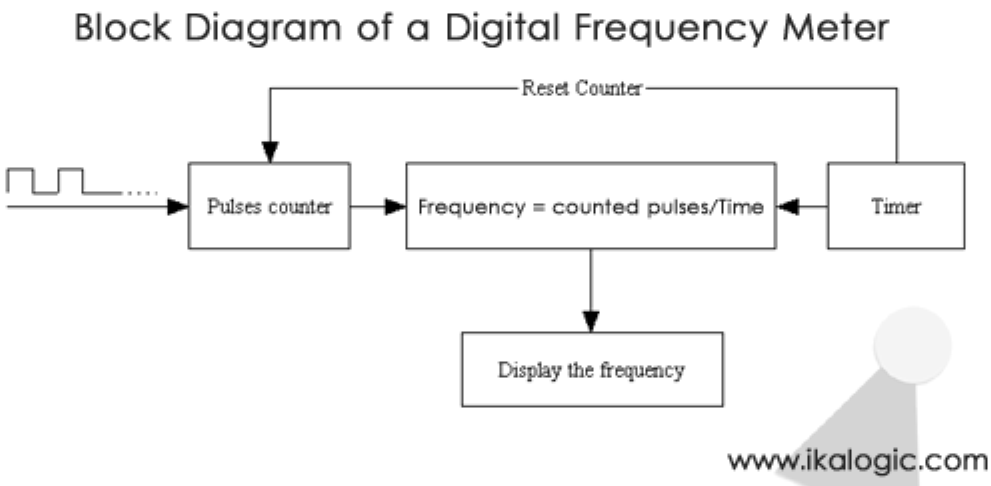


figure 1

This project will be discussed in two Parts, the Hardware and the Software.

PART 1: HARDWARE

The Hardware Parts

Lets start by this easy part,to make the project look far-or-less like a profession lab equipment, well I simply built it in an old Avo-Meter, one of those you find in stores for less than $5! (god bless china!), and then painted it all in black. (it was originally yellow!) and by the way, after I hacked that poor Avo-Meter, I also used the same leads for our frequency meter.

As you can notice in the picture, there are four seven segments display. Finally you can also notice the two connections for the leads (**Blue shaded area**). Those connectors are standard for the test leads of most AVO meters and testing devices.

The Display

Before Getting into the electronics, here is a final trick: To make the display look even neater and Protect it from dust and scratches, simply add a thin plastic film on top of it (*it will be firmly held in its place when the frequency meter is re-assembled*).

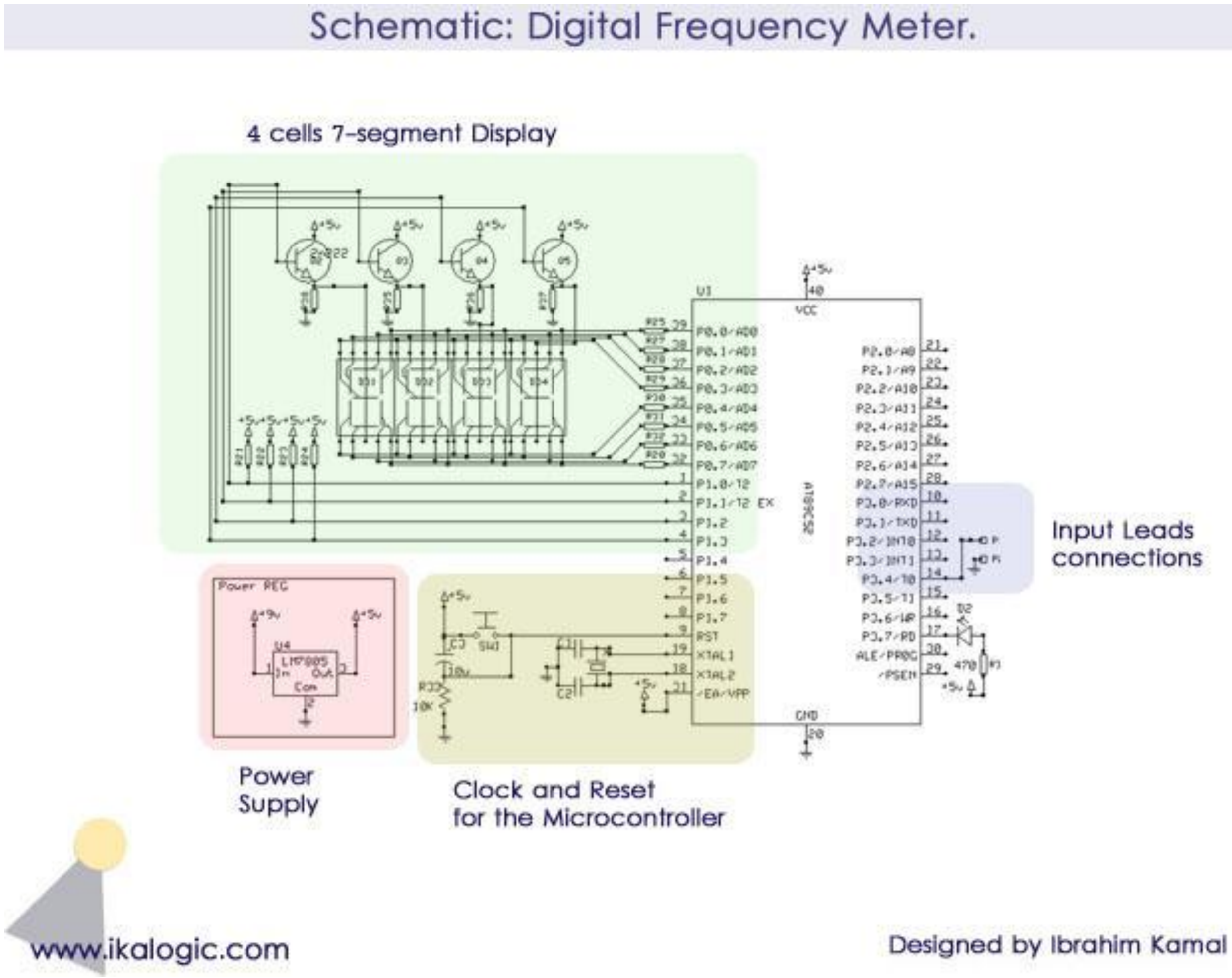
A piece of plastic film, less than 1 mm thick. preferably with a liltle shade to hide eventual defects beneath it

The 7 Segments Displays without the Protective Plastic Layer

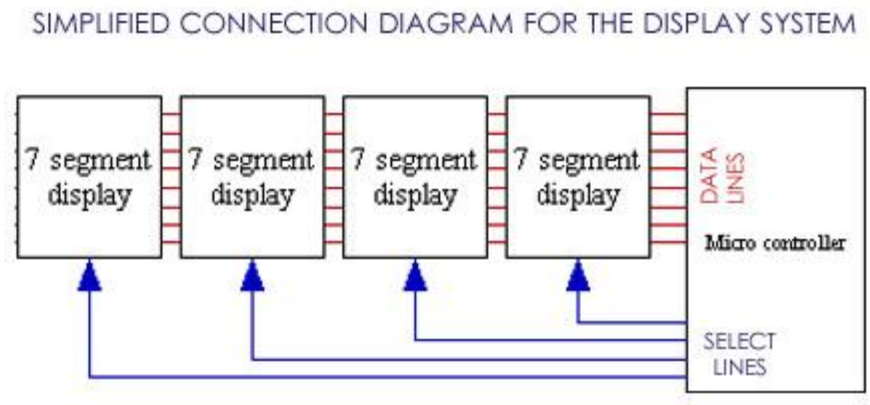
Notice How the seperations between each 7-segments cell and the other are less visible with the Shaded Plastic Layer

The electronics

I am sure that you understand that to accomplish this project, we will need to perform some mathematical operations, as mentioned before, to increase the performance of the device. Thus we will rely on an AT89C52 microcontroller to perform all the required tasks in this project.



The Display system

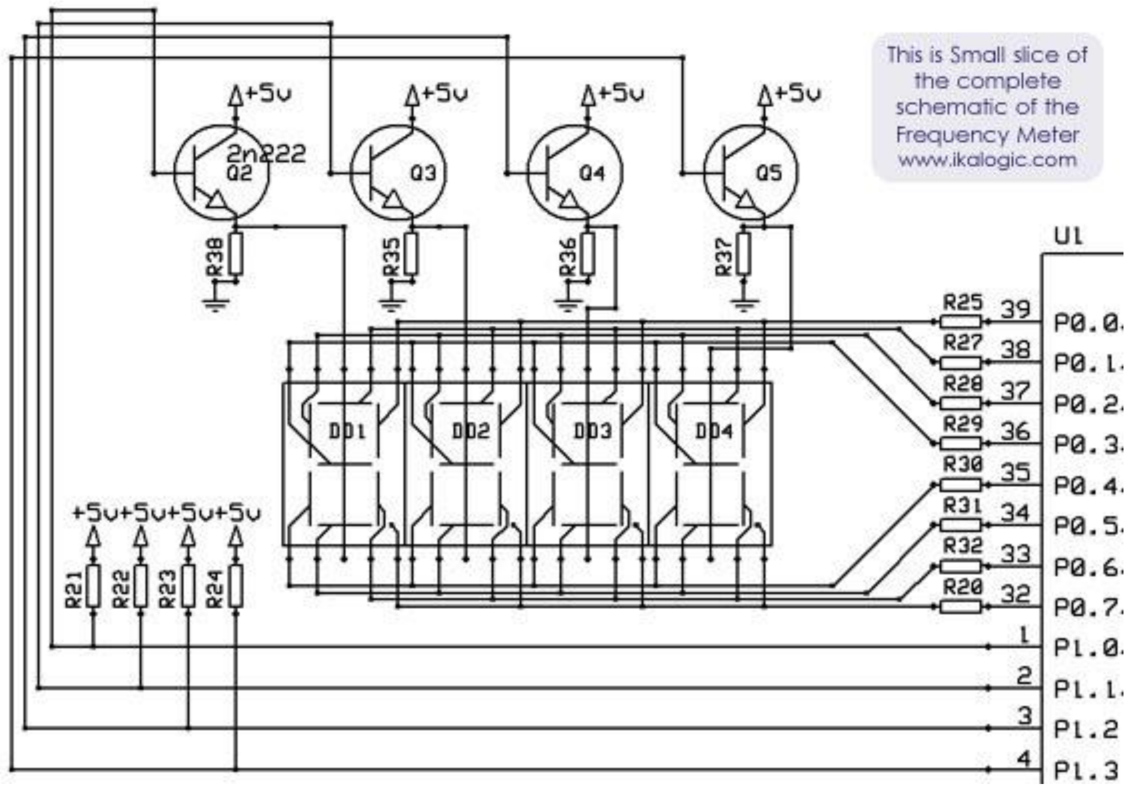


While being very simple (as you can see in this diagram), the display system can seem complicated due to the high number of connections (as you can see in the schematic below).

The main idea for this display system, is to connect all the 7-segment cells together in parallel, but only power one of them in the same time.

For example, here is the sequence to show the number “1984″:

1-Send the number “1″ though the data lines
2-Energize the first cell while all other cells are off
3-wait for a short time delay (in my code i paused for 0.6 Millisecond)
4-Send the number “9″ though the data lines
5-Energize the second cell while all other cells are off
6-wait for a short time delay..
7-Send the number “8″ though the data lines
8-Energize the third cell while all other cells are off
9-wait for a short time delay
10-Send the number “4″ though the data lines
11-Energize the fourth cell while all other cells are off
12-wait for a short time delay
13-Start over from the step No: 1.
Believe it or not, the human eye wont notice anything wrong, because, with the given delays the display will refresh at the rate of **375 cycles per second** (375 Hz!) and the human eye can hardly notice some flickering in a display refreshing at 24 Hz.

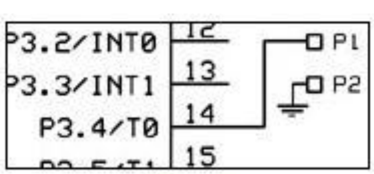


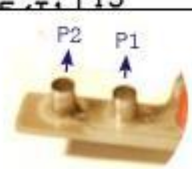
Now That you understand the display process, you should be able to easily follow the schematic above.

Here is a brief description

Q_1,Q_2,Q_3,Q_4	2N2222 switching transistors to drive the 7-segment displays.
DD1 to DD4	7-segmets display, common Anode type. *Note that it is called 7 segments, but actually each cell contain 8 leds (7 leds for the digit, and 1 led for the decimal point)*
R_{21} to R_{24}	1 Kohm pull up resistors
R_{35} to R_{38}	100Kohm pull down resistors
U1	The AT89C52 Micro-Controller (*actually you can only see the Port 0 and the 4 used pins of PORT 1*)

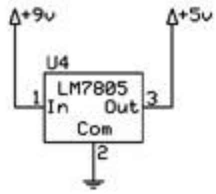
Input leads connections





Nothing really critical about this part, since we only intend to measure TTL frequencies, the input is directly fed to the T_O pin of the AT89C52, which will be used as a counter (more details about this in the software part).P_1 and P_2 are simply the two connections for the test leads. Note that in order to make any kind of measurements on another circuit, the ground of the Measurement device and of the circuit being tested **must be connected together**, this is why there are at least two leads in any testing device (one for the Ground and one for the measured signal).

Power Supply, clock generator, and reset button

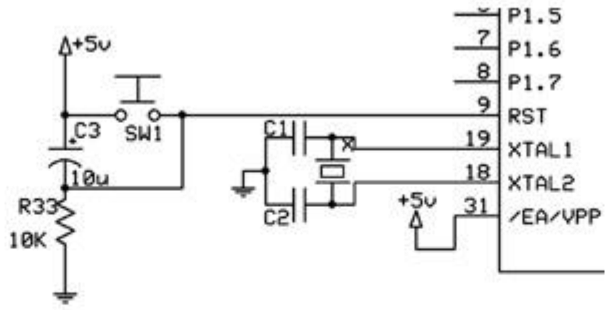


The Power supply here is not critical either, because the device will be powered from a 9V alkaline battery, so we don't need any filter capacitors, we just need to reduce 9V to 5V, and this is simply the job of the IC 7805 (Usually we add decoupling and filter capacitors when there is a lot of disturbance like in motor controllers, or AC to DC power supplies)

This part is standard in any circuit containing an AT89C52

- The switch SW_1will reset the micro-controller and start the program from the beginning.
- C_1 and C_2 are 27 pF, any nearby value between 20 and 40 pF will work just fine.
- X is a 24 Mhz crystal. it is imperative to use a crystal of this frequency, which is the maximum frequency the AT89C52 can respond to. using a 12 Mhz crystals will result in maximum measurable frequency of 250 Khz instead of 500 Khz.

Don't forget to connect the PIN 31 to the 5V, otherwise the micro controller wont work.



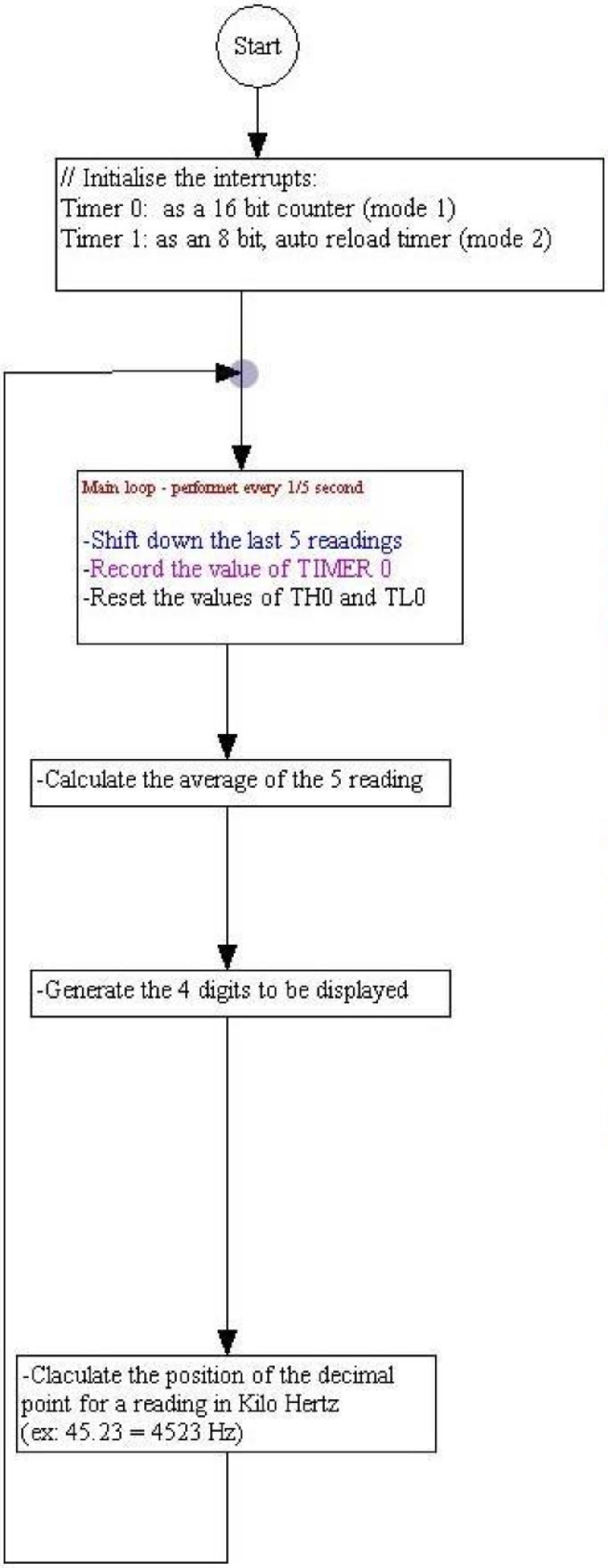
PART 2: SOFTWARE

NOTE: While I tried to make things as clear as i could, this is not a tutorial to explain to beginners what is a microcontroller and how to use it…

Now you have to bear in mind the simple block diagram of a frequency meter shown at the top of the page. You can download the whole code in one file here, but even a professional programmer would easily get lost in someone's else code, so my advice is to try to understand how it is working, and then write your own version of the code.

The Main loop

Now, here is a simplified flow chart of the main loop. Next to each box, you can see the corresponding source code, this can help you to understand how each step is programmed.




```
setup_interrupts(){
EA = 1;
ET0 = 1; //Enable the Timer/counter 0 interrupt
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
TL0 = 0; //empty the counting registers
TH0 = 0; //empty the counting registers
TH1 = 100; //start timer 1 from 0
ET1 = 1; //enable timer 1 interrupt
TR1 = 1; //Enable Timer/counter 1 to count
PT0 = 1; // Lower priority for timer 0
PT1 = 0; // higher priority for timer 1
}
calc_del++;
if (calc_del > (2248/scale)){ // update data
calc_del = 0;
f = (TL0 + (TH0* 256));
sample[4] = sample[3];
sample[3] = sample[2];
sample[2] = sample[1];
sample[1] = sample[0];
sample[0] = f;
TL0 = 0; // reset the two registers of
TH0 = 0; // the 16 bit counter
f = floor(
(sample[0] + sample[1] + sample[2]+
sample[3]+ sample[4]) /5
)*scale;
void int_to_digits(unsigned long number){
float itd_a,itd_b;
itd_a = itd_a = number / 10.0;
dig[3] = floor((modf(itd_a,&itd_b)* 10)+0.5);
itd_a = itd_b / 10.0;
dig[2] = floor((modf(itd_a,&itd_b)* 10)+0.5);
itd_a = itd_b / 10.0;
dig[1] = floor((modf(itd_a,&itd_b)* 10)+0.5);
itd_a = itd_b / 10.0;
dig[0] = floor((modf(itd_a,&itd_b)* 10)+0.5);
}
if (f < 1000){
pt[0] = 128;
pt[1] = 0;
pt[2] = 0;
pt[3] = 0;
}else if ((f > 999) & (f < 10000)){
f = f / 1;
pt[0] = 128;
pt[1] = 0;
pt[2] = 0;
pt[3] = 0;
}else if ((f > 9999) & (f < 100000)){
f = f / 10;
pt[0] = 0;
pt[1] = 128;
pt[2] = 0;
pt[3] = 0;
}else if ((f > 99999) & (f < 1000000)){
f = f / 100;
pt[0] = 0;
pt[1] = 0;
pt[2] = 128;
pt[3] = 0;
}else if ((f > 999999)){
f = f / 1000;
pt[0] = 0;
```

The Counter and the Timer

The function count_pulses() , which is linked to the "Interrupt 1″, will be executed each time the TIMER 0 overflows. Actually this should never happen, because the counting registers are emptied every 1/5 second (if scale = 5, by default), but in case the timer overflows, we should increase the variable: "scale", hence the counting variables will be emptied every 1/6 second, and if the the TIMER 0 still overflows, scale will be increased to 7, and so on, until the system stabilizes at a point where the counting registers are being emptied fast enough, then the calculated frequency is correct.

```
count_pulses()
interrupt 1
//counter 0
interrupt

{

if (scale < 200)

scale++;

}
```

The function calc_and_disp() will be executed every 0.66 millisecond (this is defined by the variable TH1 in the setup_interrrupts() function) Each time this function is executed, one of the 4 7-segments is energized, and a corresponding digit is being sent, then, next time the function is called, the next digit is energized to show the next corresponding number… And the sequence goes on as explained before in the display system. Note: **dcnt** is the variable used to select on of the 4 display cells.

```
calc_and_disp()
interrupt 3 {

P0 = (bcd[dig[3-
dcnt]] - pt[3-
dcnt]) ;

P1 = ord[3-dcnt];

dcnt++;

if(dcnt > 3){

dcnt = 0;

}

}
```

Now you should be able to build your own Frequency meter. To help you a litle more, I'm giving you a zip file containing the full schematic, the PCB design and the source code.

Source: http://www.ikalogic.com/5hz-to-500-khz-frequency-meter/