Build your 40 MHz Frequency meter!

Posted on May 6, 2008, by Ibrahim KAMAL, in Sensor & Measurement, tagged

This article shows how to build a small, cheap and simple frequency meter, without any fancy, out of reach components. The simple proposed design can measure frequencies up to 40 Mhz with errors below 1%! This degree of precision will be more than enough to debug most of your analog and digital circuits, and will give you the ability to analyze many aspects that you were unable to see before.

The frequency meter is build on a veroboard, using only three components and eight resistors. It is designed to be plugged into any standard bread-board.

The hardware

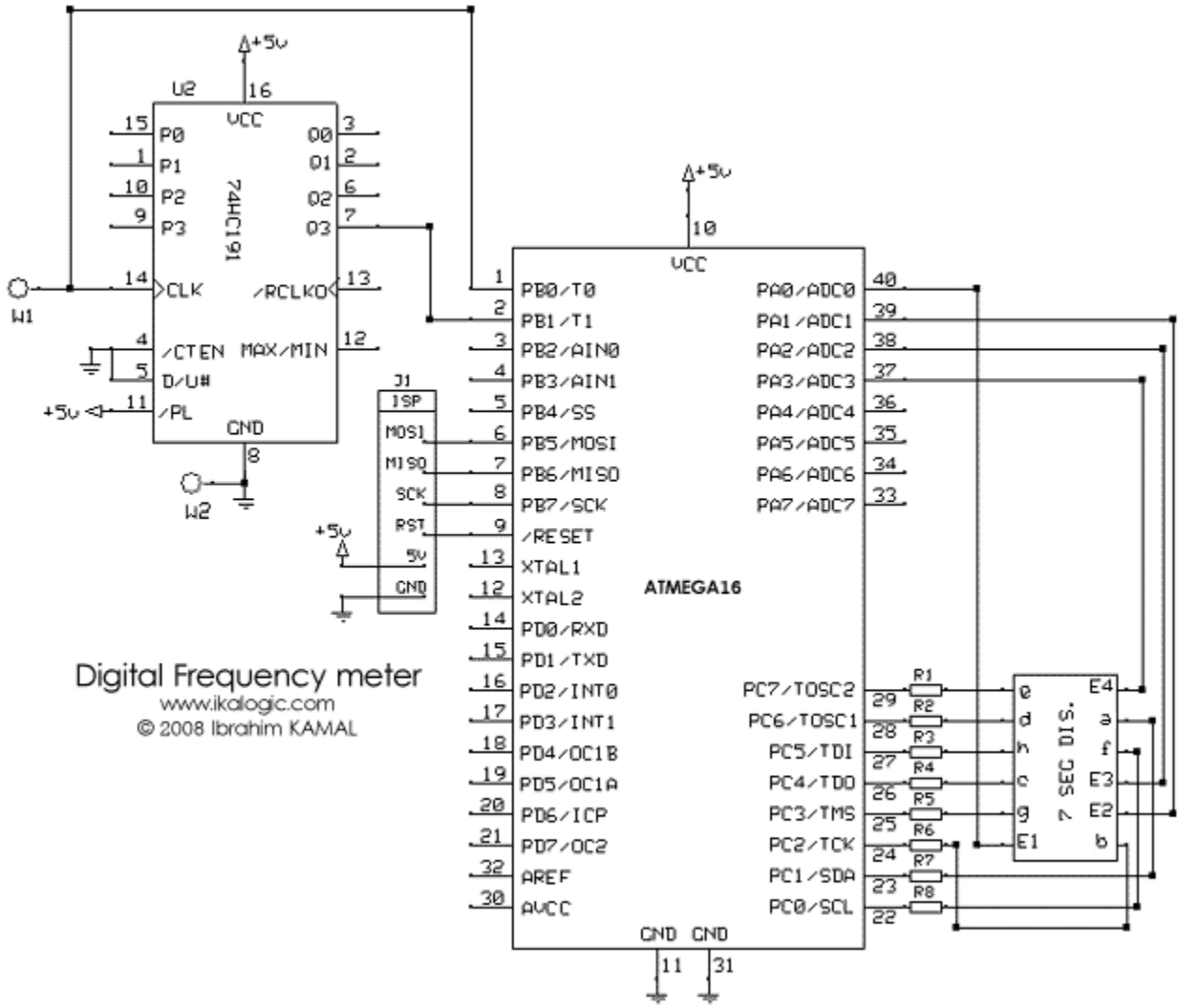

figure 1.A

As you can see in ***figure 1.A***, the design is built on a ATMEGA16 micro controller. There are no crystal oscillators, as we are using the internal RC oscillator, calibrated to 8 MHz.

The frequency measurement probe is connected to W_1 terminal, which is then fed to the pin PB0 and the clock input of the 74191 4-bit counter. The 4-bit counter will be used to divide the measured frequency by 16 before feeding it to the microcontroller. As you can see, all the features of the counter aren't used, only the Q_3 output is used, whose frequency will always be equal to the input frequency divided by 16.

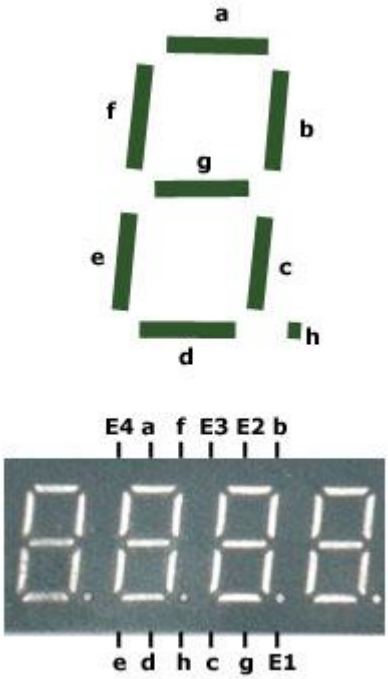

figure 1.B

The 7 segments display used is composed of four cells all integrated into one package. This reduces the number of wires to enable the different cells.

The display we used is common anode type, and had its leads arranged according to the diagram in ***figure 1.B***, and most of the 7 segment cells of this type will follow those standard connections. However, if you use different type of 7 segment display, the software can be very easily adapted to accommodate the changes. The pins E_1 to E_4 lets you enable one of the four cells, E_1 enables the right most one.

Since we are using an ATMEGA16 microcontroller that can source up to 40 mA of current per I/O pin, we don't need transistors, so the four enable signals of the 7 segment display is directly connected to the microcontroller.

The wire connection W_2 is to connect to ground, to allow measurement of frequencies on devices that do not share the same power supply as the frequency meter itself. (Remark, for one circuit to be able to

measure an signal on another circuit, they must share the same ground voltage, so we connect them together.)

Finally, J_1 is a connection for the ISP programmer (In System Programmer). In need, after you finish the project you will spend 10 minutes doing nothing by calibrating your frequency meter, and adjusting some variables to make the display clearer or to reduce the flickering of the numbers being displayed. That's why we added this ISP connector, because we will need to update the code of the microcontroller often.

The whole is assembled on a veroboard, and, as you can see in ***figure 1.C***, it is designed to be plugged directly into the power supply rails of a standard bread board. You can also notice that there is a protection diode added (though it is not present in the schematic), it prevents any damage to occur in case the frequency meter is plugged in the wrong direction.

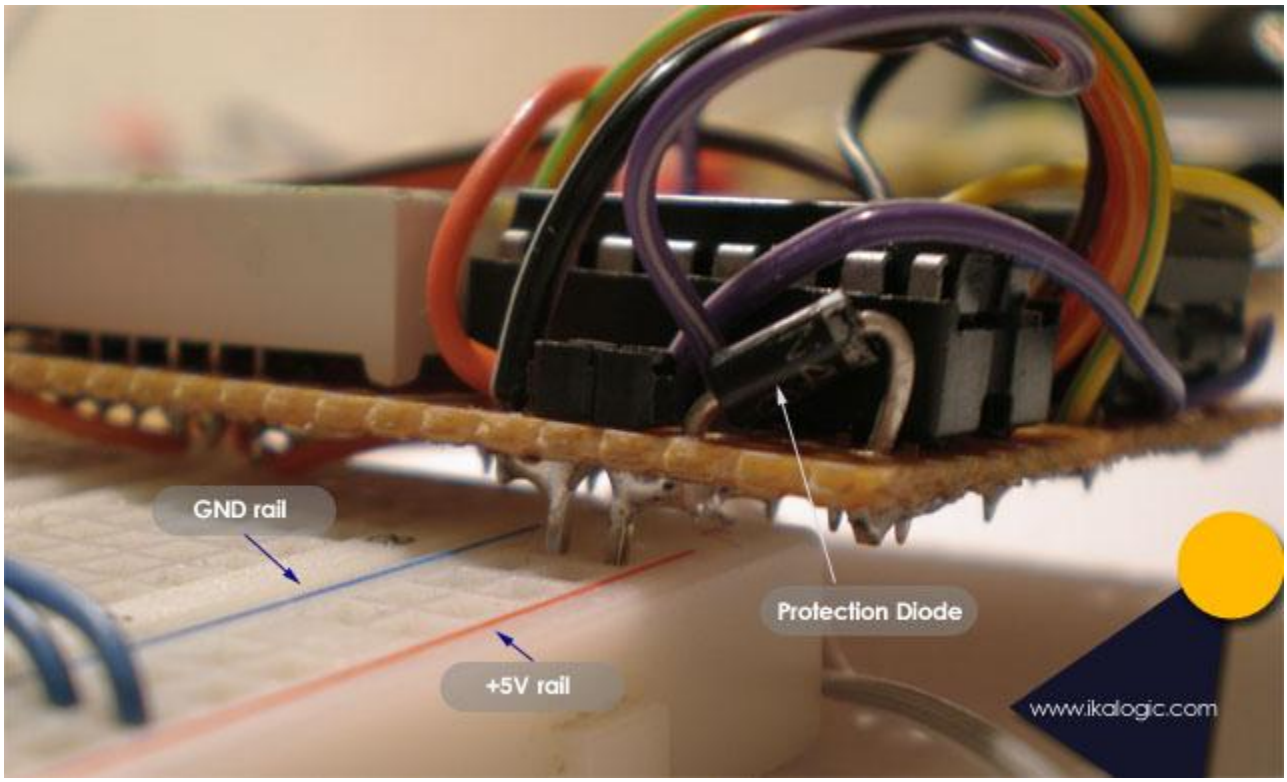

figure 1.C

Next (***figure 1.D***) shows the two probes that are connected to W_1 and W_2 (see the schematic in ***figure 1.A***). The ground probe is only used when the power supply of the frequency meter is different from the power supply of the device being tested, to connect both grounds together. the frequency measurement probe is made of a single pin header, covered with a piece of "Heat shrink tube" (or also called thermo-retractable tube).

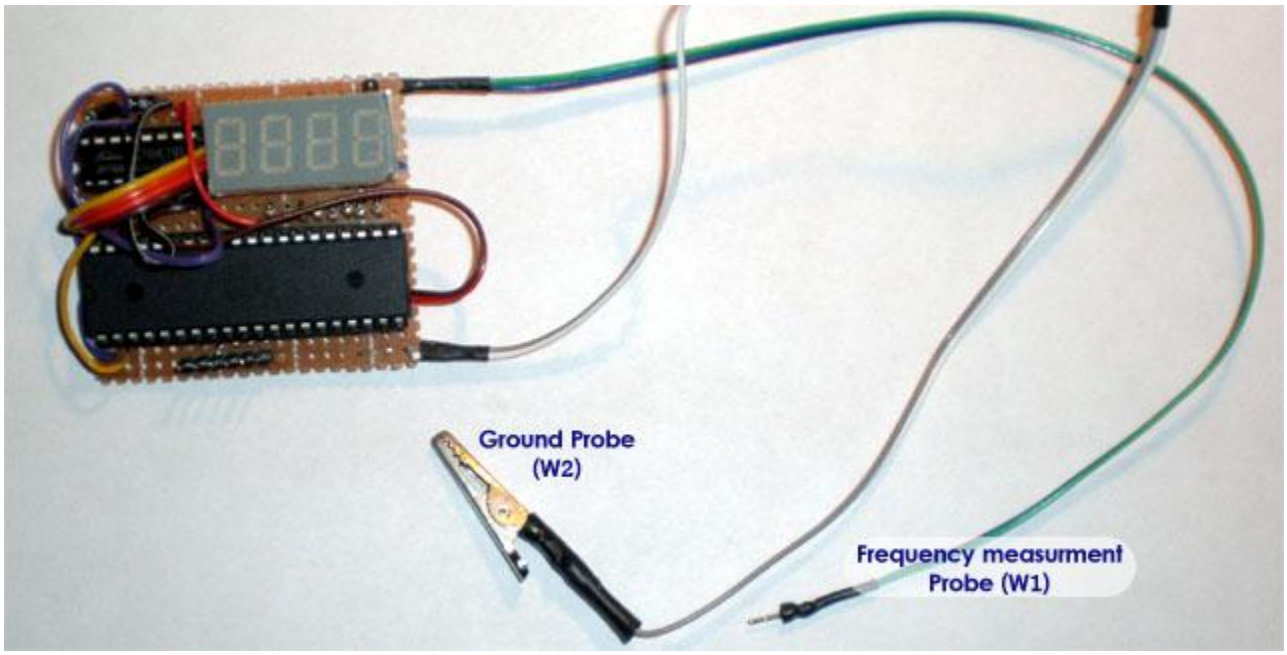

figure 1.D

Frequency measurement algorithm

We all know – at least most of the visitors that made it to that website! – that "F*requency is a measure of the number of occurrences of a repeating event per unit time*". but measuring frequencies with digital tools such as this microcontroller that have its limitations involve some further studies to achieve the required results.

The maximum frequency that can be sampled by one of the counters of the ATMEGA16 cannot exceed the CPU clock divided by 2.5. Let's call this frequency F_max. So, assuming the ATMEGA16's CPU is clocked at 8 MHz, we can directly measure frequencies up to 3.2 Mhz. Frequencies above that limit will be measured as 3.2 MHz or less, since not all the pulses will be sampled. To be able to measure frequencies above F_max, we use a 4 bit counter as a frequency divider, dividing the measured frequency by 16. This way we can also measure frequencies up to 16 times F_max, but due to the limitation of the 74191 counter, the actual maximum measurable frequency wont exceed 40 MHz.

The algorithm that we developed measures both the original frequency (let's call it F_o) and divided frequency (F_d). in normal conditions, when the frequency is below F_max, the following relation is true:

$$F_o = 16*F_d$$

But as F_o approaches to F_max, more and more pulses wont be sampled, and the the relation above will obviously become:

$$F_o < 16*F_d$$

And hence the limit of the microcontroller can be automatically detected.

The frequency meter starts by selecting the original frequency for processing and display, and as soon as it detects that it reaches F_max (using the method described above), the divided frequency is selected instead.

This algorithm can be summarized in the following flow chart (***figure 2***).

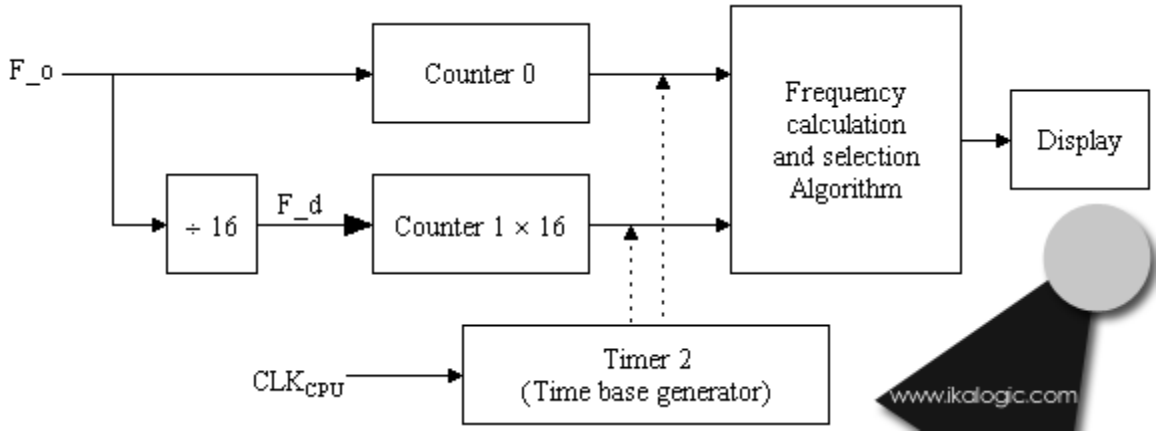

figure 2

The software

The C source code for this frequency meter can be downloaded here.

The source code is detailed with as much comments as I could, but you may need some more explanations to understand the code:

- The code is made such that the number being displayed is in KHz. For example, if you see on the 7 segments the number “325.8″ that means 325.8 KHz, “3983″ would mean 3983 KHz (or 3.983 MHz). If the number is in tenth of megahertz, it is displayed with an “m” at the end like “22.3m” meaning 22.3 MHz.
  - Timer/Counter 0 is used to count incoming pulses directly
  - Timer/Counter 1 is used to count incoming pulses divided by 16
  - Timer/Counter 2 is configured as timer with a 1024 prescaller (counting CPU frequency divided by 1024). It is used to call the “frequency calculation and selection algorithm” every timer period T. T is defined as “1024*256/(F_cpu)”.
  - The constant “factor” defined in the top of the program as “31.78581″ have to be calibrated by measuring a known frequency. This factor was initially calculated as the following:

factor = F_cpu / (1024*256) = 8.E6 / (1024*256) = 30.51757

- But due to the fact that there is some delay between the time the Timer_2 overflows and the time where the counted frequency is actually processed, this factor need some calibration to reflect more accurately the reality of the frequency being measured.

- The Anti-flickering algorithm is complicated but very effective, specially in measuring frequencies that are unstable. It will totally prevent the display from quickly switching between various values, while still showing accurate values, and quickly change if the frequency being measured “really” changes.
- One last note, the ATMEGA16 is chipped with a 1 MHz internal oscillator enabled. To set the internal oscillator to 8Mhz, you have to use some programmer (like our ISP programmer) to change the fuse bits of the micro controller to adjust the internal oscillator’s frequency. (For 8MHz, the CKSEL3..0 fuses have to be set to ’0100′).
  I hope this article was useful. Any comments and further questions are welcome in the form below.

Source: http://www.ikalogic.com/build-your-40-mhz-frequency-meter/