AVR Timing Problems

Probably the most common problem people post about on avrfeaks.net is a question like this…

"I've connected my AVR to my PC/Bluetooth Modem/GPS Module/etc via a serial port and it doesn't work. It's running at 9600 baud but nothing seems to work."

…to which the first reply is usually…

"How do you know your AVR serial port is running at 9600 baud."

In the Beginning

The first thing to know is that there are **two** steps required to get ANY application which relies on definite timings running correctly on an AVR…

1. Getting the chip to run at a known speed.
2. Telling your software what that speed is.

The second thing to know is that simply putting…

```
#define F_CPU 8000000UL
```

…near the top of your code DOES NOT make your AVR run at 8MHz.

Getting the Chip to Run at a Known Speed

So let's look at step 1 and get our AVR to run at a known speed.

There are a number of different ways to clock our AVR. If we look at a datasheet, in this case for a mega328P, we can see the options we have available.

9.2 Clock Sources

The device has the following clock source options, selectable by Flash Fuse bits as shown below. The clock from the selected source is input to the AVR clock generator, and routed to the appropriate modules.

Table 9-1. Device Clocking Options Select(1)

Device Clocking Option	CKSEL3...0
Low Power Crystal Oscillator	1111 - 1000
Full Swing Crystal Oscillator	0111 - 0110
Low Frequency Crystal Oscillator	0101 - 0100
Internal 128kHz RC Oscillator	0011
Calibrated Internal RC Oscillator	0010
External Clock	0000
Reserved	0001

Note: 1. For all fuses "1" means unprogrammed while "0" means programmed.

The most common choices are the Full Swing Crystal Oscillator and the Calibrated Internal RC Oscillator.

The Calibrated Internal RC Oscillator

As the name suggests this oscillator is inside the AVR and requires no external components. The word 'calibrated' is misleading though as for many applications, especially UART communications, the frequency it runs at is not accurate enough for reliable operation. Which means you really need to be using…

The Full Swing Crystal Oscillator

This oscillator requires that you connect a crystal and two capacitors to your AVR. Crystals looks like this…

They are used along with two low-value ceramic capacitors connected as per the AVR datasheet. Typical capacitor values are in the 15-22pF range.

The advantages of the Crystal Oscillator are…

- It's very accurate.
- You can use a frequency which is easy to use with normal UART baudrates.

It's important to note that you should not select 'External Clock' or similar descriptions in your programming software when you want to use a crystal. If you do you will 'brick' your AVR and make it un-programmable.

AVRs and Default Clock Sources

Nearly all AVRs ship from the factory set to run from the RC oscillator. You can check this for your type of AVR in the datasheet. Here's what the datasheet for the mega328P says…

> **9.2.1 Default Clock Source**
>
> The device is shipped with internal RC oscillator at 8.0MHz and with the fuse CKDIV8 programmed, resulting in 1.0MHz system clock. The startup time is set to maximum and time-out period enabled. (CKSEL = "0010", SUT = "10", CKDIV8 = "0"). The default setting ensures that all users can make their desired clock source setting using any available programming interface.

So what does this tell us? It tells us that, unless we change anything, a factory fresh mega328 will be running from an 8MHz internal oscillator but that the output from the oscillator will be divided by a

factor of 8 before being fed to the chips circuitry. So this means our chip is running at 1MHz and any peripherals, like a UART, will use that 1MHz clock as the timing source.

IMPORTANT NOTE:- not all AVRs, especially some of the older chips, have that factor of 8 divider. Again, check the datasheet for your type of chip.

Selecting Your Chosen Clock

STUFF ABOUT FUSES

Running Faster

A 1MHz clock is usually too slow and is a waste when your chip can run faster. It's quite easy to switch the divider out of circuit and you have two ways to do it…

- By programming a fuse in the chip.
- In software, but not all chips can do this.

MORE FUSES

STUFF ABOUT CLKPR

Telling your software what the speed is.

Once you have your AVR running at the right speed it is quite easy to tell the software what that speed is.

In GCC-AVR, as used in Atmel Studio, you simply add…

```
#define F_CPU 8000000UL
```

…at the top of your source code. From then on, if you use the functions in delay.h or calculate a baudrate using the usual method, any timings will be correct. You do, of course, need to replace "8000000" with the clock rate your AVR is running at.

In Codevision you use the project configuration options…