WHY YOU NEED A CLOCK SOURCE

AN INTRODUCTION TO CHOOSING AND USING CLOCK SOURCES

COLIN O'FLYNN

MAY 2003

TABLE OF CONTENTS

Why you need a Clock Source

The clock for the AVR is its heartbeat. It basically defines when a processor will do an instruction. Normal instructions for the chip take between one to three clock cycles to complete; a clock cycle is the low-hi-low transition of the clock. So the faster the clock runs, the faster the chip works. This is why a Pentium 1 GHz chip is a LOT faster than a Pentium 100 MHz chip - many more instructions are executed per second. However be careful - the speed of the clock doesn't actually mean a lot. One processor could take 1 clock cycle to move data from one register to another, but another processor could take 2 clock cycles to do the same thing. This means that processor number 2 will have to run at double the speed as processor 1 to do the same instruction in the same amount of time! The key here is a little rating called MIPS (Million of Instructions Per Second), which is surprisingly (not) how many instructions the chip could do in a second at a specified clock frequency. Anyway the point is that you need something that tells the processor when it will be doing instructions (the clock). Even THEN that's not a perfect indicator of speed, as stuff like on-board cache, bus width and loads of other things make a huge difference!

Selecting a Clock Source

There are a few different ways to generate a clock for the AVR. You can use an external clock that feeds in the low-hi pulse stream, a crystal or a ceramic resonator. A few of the chips even have a built-in RC oscillator! We'll talk about properly interfacing them in a second, first lets see the advantages, disadvantages of each.

External Clock

This one is fairly easy - all you do is have some circuit that creates TTL compatible logic pulses, and feed those into pulses into pin XTAL1. Your external clock could be a cesium-based atomic clock, or a chipmunk spinning a wheel, with a unit counting the RPM of the wheel. The external clock should be very clean and fall and rise times should be fast.

Crystal

The crystal is a very accurate and (in theory) simple way to get a clock. As well crystals do not change their value (much) with a changing external temperature. Crystals are available in many frequencies and sizes, and needs two external 'load capacitors' to work.

Two Pin Ceramic Resonator

A ceramic resonator is similar to a crystal, except that it is much cheaper. However as a consequence it is also not as accurate and the value can change at different temperatures. An advantage is that they are less fragile than a crystal - so in high-vibration environments they can perform (fairly) well. The ceramic resonator may not be accurate enough for reliable serial data transfer (using the UART), but sometimes it is (depends on quality). If you need reliable data transfer, a crystal is a better choice. The two pin ceramic resonator needs two external 'loading capacitors'.

Three Pin Ceramic Resonator

The three pin ceramic resonator is similar to the two pin - except that it doesn't need the external loading capacitors as they are built in. One pin of the resonator goes to ground, one to XTAL1 and one to XTAL2 (it doesn't matter which pin goes to XTAL1 or XTAL2, but the pin to ground matters!).

Internal Oscillator

A few of the chips are given an internal RC oscillator. The new 'Mega' series almost all have an internal oscillator. This oscillator can vary from different chips, but one of the faults of them is that the frequency will change with changing voltage on VCC pin and temperature. However you can often calibrate them for relative accuracy (still not near a crystal and not quite as good as a resonator). To use them you have to enable the proper fuse.

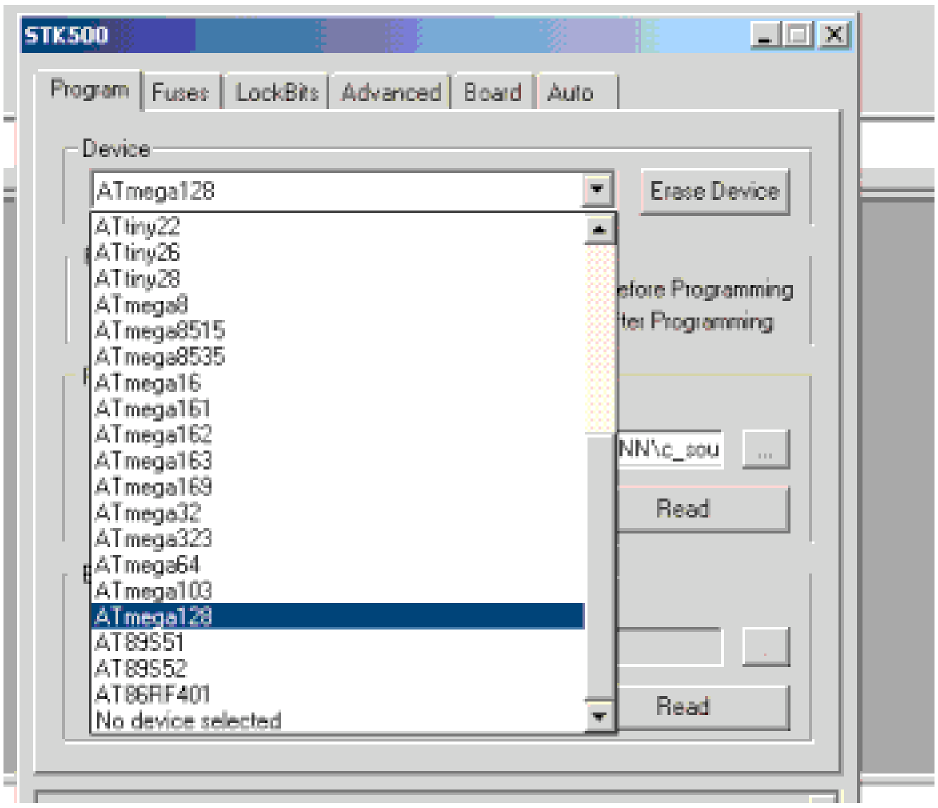
Important note: Many chips have this selected by default, so even if you connect a 16 MHz crystal to your AtMega128-16AC it may only be running at 1 MHz! Make sure to select the proper fuse bits if you are using a Mega series chip (as opposed to the AT90S series).

So how do you select the proper one? Well it depends on application. If you need high accuracy, then external clock or crystal is the way to go. If you need low cost, then internal oscillator or ceramic resonator is the way to go. If you want the easiest solution, then internal oscillator or three pin ceramic resonator is the best choice.

Fuses and Clocks

For most of the AT90S series of AVRs there was no worry about fuse bits and clock sources. You just connected the proper oscillator up, and away you went! This article was originally written when this was the case, so this fuse bit section had to be added. If you are using the Mega series of AVRs them you will need to pay close attention here.

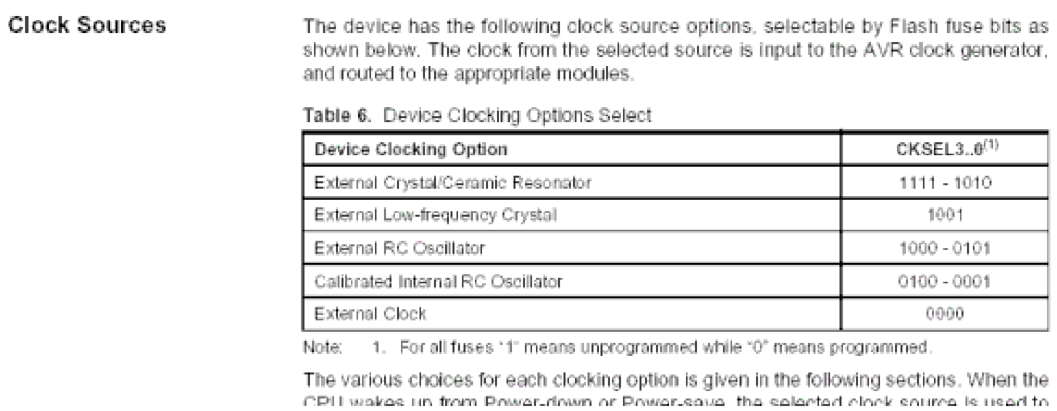
The exact fuse settings will change depending on what AVR you are using and your programming software. If you are using AVR Studio programming interface (for STK500, JTAG ICE, or ATAVRISP) it is fairly straight forward. You simply select the AVR, and then select what fuse bits you want. In this case the *Ext. Crystal/Resonator High Freq.; Start-up time: 16 CK + 64 ms* is checked, as well as the *CKOPT* Fuse is checked. The CKOPT fuse must be programmed to use the AVR at high speeds (above 8 MHz).



Device: ATmega128

Don't forget to hit the 'Program' button as well to burn the fuses in!

If you are using some other programming software such as avrdude it is still not that complex. Go to the data-sheet section of interest, the clock section. Look for a table to program the CKSEL value, it should look like this:



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 6. Device Clocking Options Select

Device Clocking Option	CKSEL3..0[1]
External Crystal/Ceramic Resonator	1111 - 1010
External Low-frequency Crystal	1001
External RC Oscillator	1000 - 0101
Calibrated Internal RC Oscillator	0100 - 0001
External Clock	0000

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

The various choices for each clocking option is given in the following sections. When the CPU wakes up from Power-down or Power-save, the selected clock source is used to

If we look at the table we will be using an external crystal, so the CKSEL fuse must be between 1111 to 1010, which is 0x0F to 0x0A in hex. Reading further on it says that for frequencies above 8 MHz the CKOPT fuse muse be programmed (note: programmed = 0, unprogrammed = 1). As well you can see that for the highest frequencies the CKSEL fuse should be set to 0x0F. You can confirm this by looking in the AVR Studio, where it shows you the value of the fuse you are setting beside each check-box.

Now to set the start-up delay; further on there is another table. To set a 16K clock cycle + 64 ms delay the SUT fuse should be set to 10 (0x02 hex). Now we have to look what fuse byte this should all be written into, as there is no actual CKSEL or CKOPR or SUT fuse, there is just one or two bytes that holds all these.

Look near the end of the datasheet in the 'Memory Programming' section, then look under the subtopic of fuse bits.

For the Mega128 the CKSEL and SUT fuse bits are in the *Fuse Low Byte*, and the CKOPT bit is in the *Fuse High Byte*. The *Fuse High Byte* table looks like this:

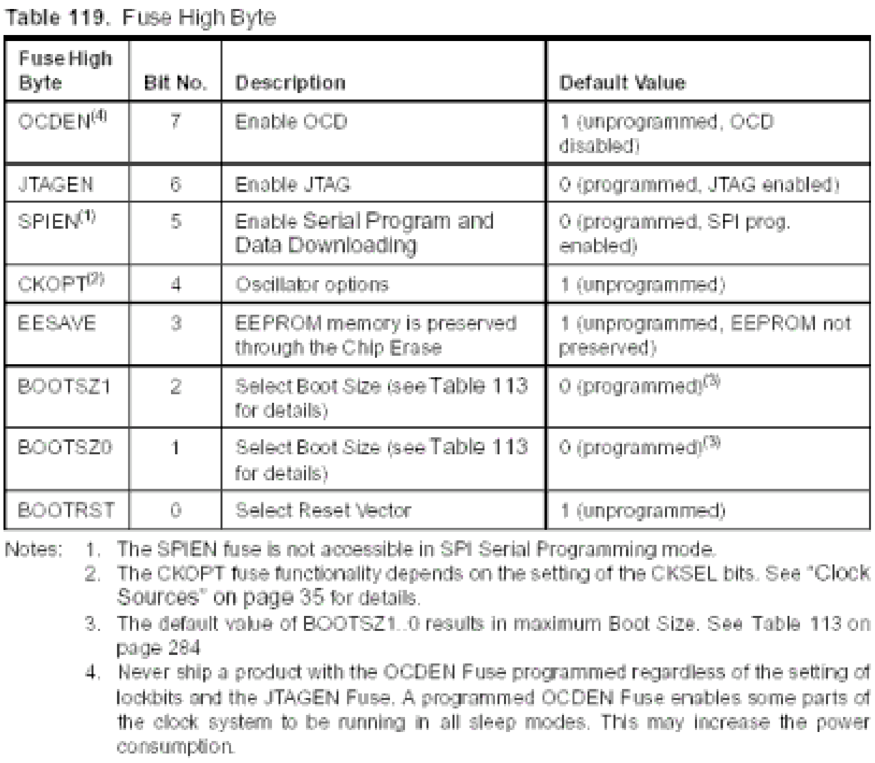


Table 119. Fuse High Byte

Fuse High Byte	Bit No.	Description	Default Value
OCDEN[4]	7	Enable OCD	1 (unprogrammed, OCD disabled)
JTAGEN	6	Enable JTAG	0 (programmed, JTAG enabled)
SPIEN[1]	5	Enable Serial Program and Data Downloading	0 (programmed, SPI prog. enabled)
CKOPT[2]	4	Oscillator options	1 (unprogrammed)
EESAVE	3	EEPROM memory is preserved through the Chip Erase	1 (unprogrammed, EEPROM not preserved)
BOOTSZ1	2	Select Boot Size (see Table 113 for details)	0 (programmed)[3]
BOOTSZ0	1	Select Boot Size (see Table 113 for details)	0 (programmed)[3]
BOOTRST	0	Select Reset Vector	1 (unprogrammed)

Notes: 1. The SPIEN fuse is not accessible in SPI Serial Programming mode.
2. The CKOPT fuse functionality depends on the setting of the CKSEL bits. See "Clock Sources" on page 35 for details.
3. The default value of BOOTSZ1..0 results in maximum Boot Size. See Table 113 on page 284
4. Never ship a product with the OCDEN Fuse programmed regardless of the setting of lockbits and the JTAGEN Fuse. A programmed OCDEN Fuse enables some parts of the clock system to be running in all sleep modes. This may increase the power consumption.

We will only change the fuses which we need to, the rest will be default for this example. The default value for this fuse byte is: 10011001. We are changing the CKOPT for a 1 (unprogrammed) to a 0 (programmed), so the resulting fuse byte value will be 10001001, which is 0x89 hex.

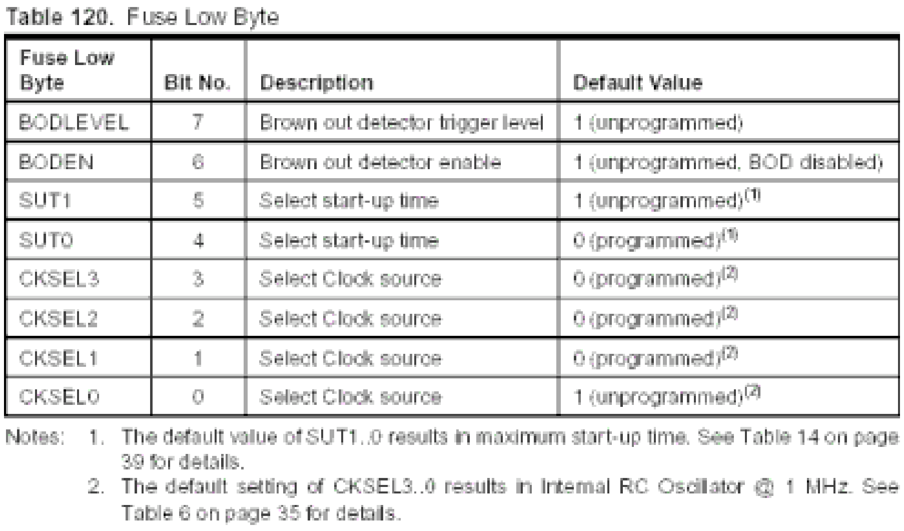


Table 120. Fuse Low Byte

Fuse Low Byte	Bit No.	Description	Default Value
BODLEVEL	7	Brown out detector trigger level	1 (unprogrammed)
BODEN	6	Brown out detector enable	1 (unprogrammed, BOD disabled)
SUT1	5	Select start-up time	1 (unprogrammed)[1]
SUT0	4	Select start-up time	0 (programmed)[1]
CKSEL3	3	Select Clock source	0 (programmed)[2]
CKSEL2	2	Select Clock source	0 (programmed)[2]
CKSEL1	1	Select Clock source	0 (programmed)[2]
CKSEL0	0	Select Clock source	1 (unprogrammed)[2]

Notes: 1. The default value of SUT1..0 results in maximum start-up time. See Table 14 on page 39 for details.
2. The default setting of CKSEL3..0 results in Internal RC Oscillator @ 1 MHz. See Table 6 on page 35 for details.

The *Fuse Low Byte* looks like this: Again only the fuses of concern will be changed. In this case it is most of them, except for the BOD (Brown Out Detector). The default is 11100001. However, we are changing that to 11101111, which is 0xCF hex.

At this point the programming interface would be used to program the fuse bytes into the AVR.

Help! I programmed the fuse byte, and my AVR stopped working!

Don't panic. There are two possibilities here:

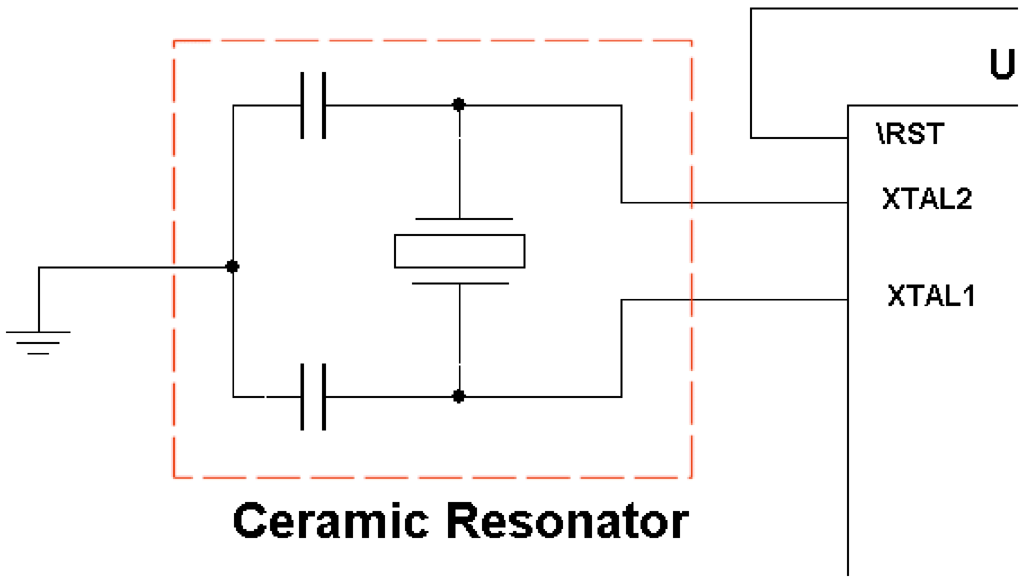
- Disabled ISP (if using ISP to program) or Disabled JTAG (if using JTAG to program)
- Changed CKSEL bits

If you disabled the programming interface your using to program the AVR, you will have to use an alternate method of programming for the AVR. Likely you may have to use parallel programming, which if you have an STK500 you can do.

A few people have accidentally selected the wrong oscillator source though on AVRFreaks, which is easy to fix. To program the chip you need an oscillator source. But for example if you were using the internal oscillator, then accidentally enabled the external oscillator input, you won't be able to re-program! To re-program you will have to get a TTL or CMOS level oscillator signal and apply it to the XTAL1. You could have also possibly change the clock source to an external crystal or external RC oscillator, so try all the different possibilities of oscillator sources to get the chip re-programmed.

Using the Ceramic Oscillator

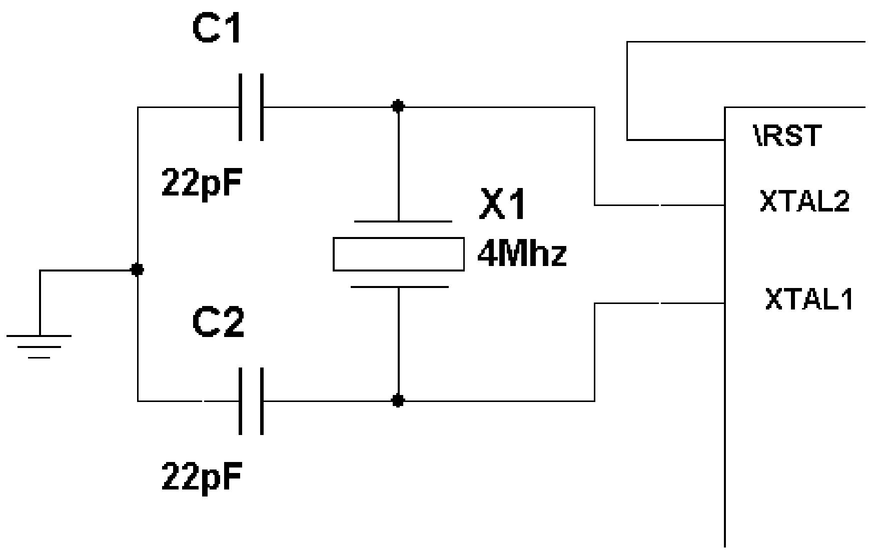
We will start with the easiest one to get working - the ceramic resonator. The two resonators are shown above - hopefully you can figure out the three pin version is the one with three pins ;) So how do you get one? While if your local electronics distributor then you'll have to buy one online. There are lots of electronics distributors, I use Digikey because they have no minimum order and they are about the only place that has fast and cheap shipping to Canada. So head over to www.digikey.com and enter 'ceramic resonator' in the search box and hit go. Then click on the link for 'ceramic resonator' and a page should coming up letting you filter the results. Pick the frequency you want, say 4Mhz. Then set Package/case to 'through hole' and hit filter. Lets first say you want the three pin variety. It doesn't actually say 'three pin', however in either the 'description' or features column it will say 'w/caps' or 'w/o caps' - w/caps means 'with internal capacitors' (three pin variety) and w/o caps means 'with out internals capacitors' (two pin variety). So click on a link for the three-pin type. Now that you are on the product page, click on the link for 'Technical/Catalog Information'. Then there should a link for the datasheet for the device - if there isn't one go back and try another part number. If it is there click on it - you will need Adobe Acrobat Reader. The datasheet might tell you some useful info - but really the important thing is the pinout. If you don't have a datasheet for the resonator you are using, most likely the middle pin is ground. So now lets see how easy it is to connect a resonator! Look at this basic schematic -as you can see it is very easy to hook up!



It doesn't matter which pin goes to XTAL1 vs. XTAL2, but the pin that goes to ground DOES matter. Try to keep the distance between the resonator and XTAL1 as short as possible, doing the same thing with the resonator and XTAL2. The two-pin variety needs external loading capacitors, and isn't used as much. These capacitors have to be of the correct value. Check the data-sheet for the resonator - it will tell you the value of the capacitors. However they are normally something between 20pF - 30pF.

Using the Crystal Oscillator

The crystal is a fairly widespread clock source. Normally your local electronics distributor will sell a crystal of some value that you can use. The crystal is depicted at the right. The crystal can be fairly easy to use, but that depends who you ask. The problem is that for proper operation a crystal needs two 'loading capacitors'. These are low-value (18-27pF) that go from each pin of the crystal to ground. See this schematic that shows you how to connect them.



So seems simple enough - get a crystal and through two capacitors on it. Well, those capacitors are the problem. Every crystal needs a proper size capacitor to insure that the crystal will oscillate properly. The datasheet for the microcontroller tells you to ask the crystal manufacture, and the crystal manufacture tells you to ask the microcontroller manufacture! And stray capacitance can have an effect on the crystal as well! So sometimes when you put your hand near it the crystal starts working, but otherwise it doesn't! So, as you can see finding the right size value is the real trick! When you buy a crystal off an online distributor (such as www.digikey.com) you can view the 'load capacitance' the crystal needs. There will be several options, ranging from something like 6-32 pF as well as one option called 'series'. You don't want series resonance crystals; they won't work with your microcontroller. You want parallel resonance crystals. If a load capacitance is specified, then that crystal is parallel resonance. So lets do an example. Go to www.digikey.com and type 'crystal' in the search box, and hit go. Next click on the link for 'crystals', again a screen should come up to let you filter results. Set the frequency to whatever you want, say 4Mhz. Then click on the 'Load Capacitance' selection box, and you should see all the choices that you can use. For now lets select 20pF and hit 'apply filter'. So the crystal needs to 'see' a capacitance of 20pF to oscillate properly. This doesn't mean that each capacitor should be 20pF in size that goes from each lead of the crystal to ground. The fact is that the traces on the circuit board, nearby wires and all sorts of things contribute to the total capacitance. This extra capacitance generated by things such as that is called 'parasitic capacitance'. This value could be anything, but normally is between 7-10pF. So anyway, enough chatter about this, lets see how we figure out the proper value of the capacitor.

rLC = 2 * (LC - PC)

rLC = Real Load Capacitor value (the value of C1 and C2) LC = Load Capacitance manufacture gave PC = Parasitic Capacitance (normally between 7-10pF)

So for our example crystal that had a load capacitor value of 20pF

rLC = 2 * (LC-PC) rLC = 2 * (20-8) rLC = 2 * (12) rLC = 24pF

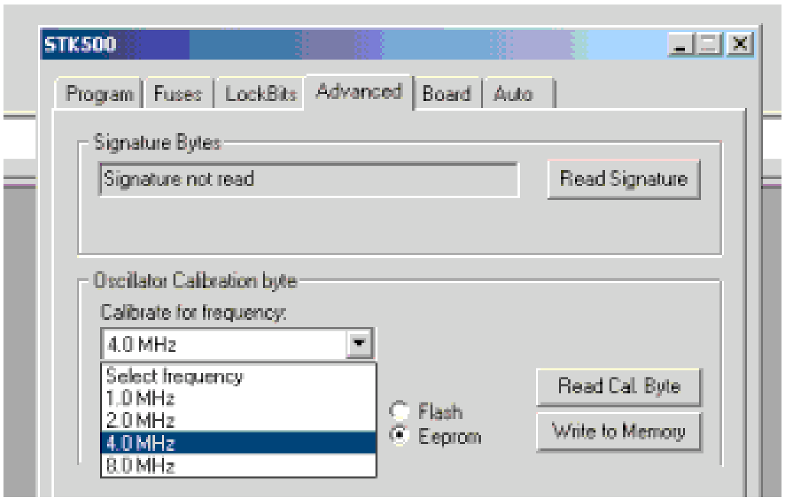
So both C1 and C2 will be 24pF, which 'sounds right'. So what if you just picked up the crystal from the bargain bin, and only know the frequency? Well then you'll have to experiment around (what fun ;). What you should do is set the circuit up, with power to the microcontroller and everything, then connect your unknown crystal up to the XTAL1 and XTAL2 pins. Then put two load capacitors on these pins, try something like 24pF. Even better is to have one fixed and one variable, or even better yet is two variable capacitors. Load a test program into the micro, one that say flashes an LED you have connected to a pin. Then check if the LED flashes. If it does, then the load capacitors are probably okay. If it doesn't then try changing the values, say to 27pF or 20pF. Note one important thing: you CANNOT just connect a scope up to pin XTAL1 or XTAL2, as the capacitance of the scope probe will change the oscillation way to much. However you CAN use a logic probe with a high impedance (don't worry, they almost all do) and you might see some really fast logic pulses if you put the tip on pin XTAL1 or XTAL2. Also when testing the crystal make sure to try all the VCC voltages that the micro will see, as well as doing things like putting your hand near the crystal.

Using the Internal Oscillator

Using an internal oscillator is really more of a software discussion, as you control the oscillator by software. This was discussed in the 'Fuses' section of this article. Also note there is a OSCCAL register on some chips (some of the Mega series) which can be used to change the value of the internal oscillator. You must read the data-sheet section for your chip for more information, but some basics will be mentioned here.

First, you should burn the CKSEL fuse to the value you intend to use. This alone doesn't change the frequency of the internal oscillator, it just readies the system to work at that frequency.

You then get the calibration value for the frequency you are using. This can be done easily in AVR Studio, first select the value you want to calibrate your AVR connected to the STK500 for and hit the *Read Cal. Byte* button:



The value it reads can then be hard-coded into your program. The only problem with that is its different for every single AVR, so that is not very practical if you might be using more than one different AVR. So AVR Studio lets you write that value to a memory address, which your program can then read from. That way if you use a different AVR you can just write the unique calibration byte to a memory location which the program will read from. This calibration byte should then be written to the OSCCAL register on program start-up.

The internal RC oscillator when it is calibrated is fairy accurate, and has an acceptable temperature constant for many non-demanding applications. You can use it for asynchronous serial communication in some cases even. If you do this, you may want to nudge the OSCCAL byte to a more 'useable' number. For example 7.3728 Mhz can be easily divided to get perfect baud rate generation. This is done by first writing OSCCAL to the closest value, in this case 8.0 MHz. Then using either a processor of trial-and error or generating a signal in the AVR (using a timer for example) which is then measured to determine the clock frequency the AVR is measured at, the OSCCAL value is changed until the AVR is running at the desired speed.

Note that the data-sheet warns you this may affect the operation of some of the AVR's systems.

Specialty Crystals

There are a few special breeds of crystals, and these are covered here. They are all crystals, but have special names. Treat them just like normal crystals, with loading capacitors and such (unless otherwise noted).

Colourburst Crystals

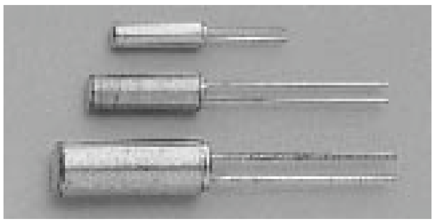
You will probably here the phrase 'colourburst crystal' (or 'colorburst' if your American :), these are normal crystals with a special frequency of 3.579 MHz. The only reason that is a special value is that that frequency is needed by colour TV's for stripping the colour information from the signal. There isn't anything special about these crystals, except for the name and the fact that they are the most commonly available frequency because TV repairmen need them.

UART Friendly Crystals

The UART needs to get a special frequency for each baud rate. It gets this frequency by dividing the main clock by a certain number. However, the problem is that if you have a 4MHz (for example) crystal, it can't get the EXACT frequency it needs. That means that a 4MHz crystal will be out by 0.16% between 1200-19200 baud rate. This isn't terrible, but at 28800 and above the error is 7.84%, pretty bad! However there is a special value of crystal - 11.0592MHz that creates a perfect timing for the UART for baud rates from 4800 - 115200. Note that 1200 and 2400 baud rates are not possible with the 11.0592MHz crystal. Also a lot of the AVRs only run up to 8MHz, so you have to use a newer type that supports higher frequencies or overclock your AVR (this is NOT recommended for any commercial products or anything that matters). The frequency of 7.3728 Mhz is also very UART friendly, and it is below the maximum speed. Check the AVR data-sheet for a few other error-free frequencies, see the tables that tell you error at different UBRR and crystal settings.

Watch Crystals

This is a special crystal, not just a special value. The exact value is 32.768 KHz, which probably doesn't seem like anything magical. However this is a value that is easily dividable to create a precise 1Hz signal. The chips that support this crystal (AT90S8535 and all of the AtMega chips) have two special pins - TOSC1 and TOSC2 that you can connect the watch crystal to. You do not need any external loading capacitors to work this crystal; everything is taken care of by the chip. Then you use the signal from the clock to generate an interrupt every one second, and you can make a software real-time clock. A few compilers support this, and there is a project in the AVRFreaks project section - just search for "RTC". The use of the crystal has minimal documentation in the datasheets, so really looking at the code from the project is the best way to figure out how to use the crystal! Just so you know what these little guys look like, there is a photo. The crystals are very small cylinders.



Placement of Crystals and Resonators

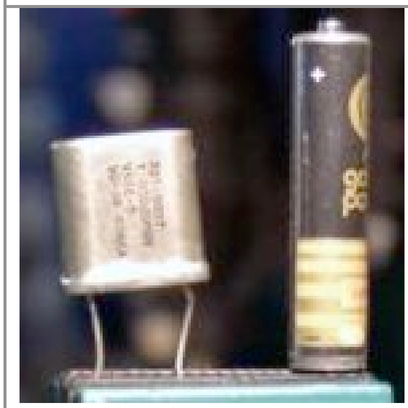
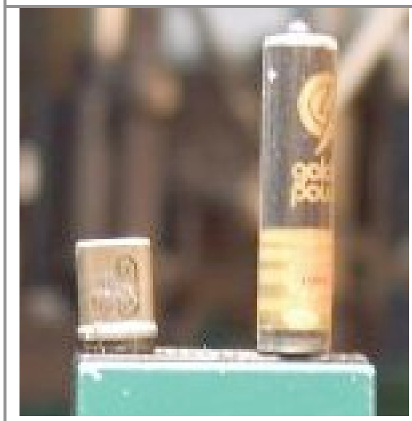
You have to be careful where you put the crystal or resonator, as the leads going from the crystal body to the microcontroller are high frequency. This means that you should have them as short as possible - no more than 1cm is a good size. If they are long leads (like crystal is on one side of board, and wires run from crystal to microcontroller) two bad things will happen. First, they will increase the capacitance that the crystal sees, and second they will create severe EMF interference.

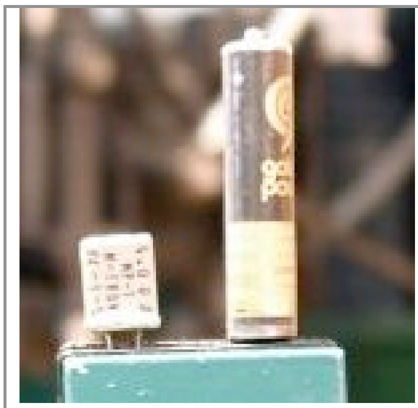
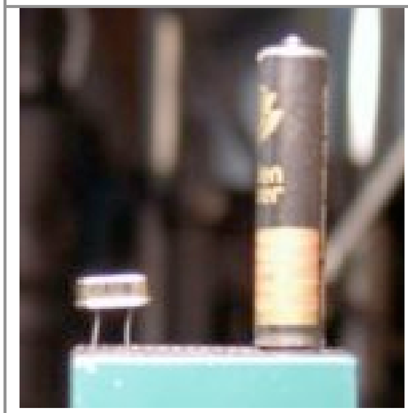
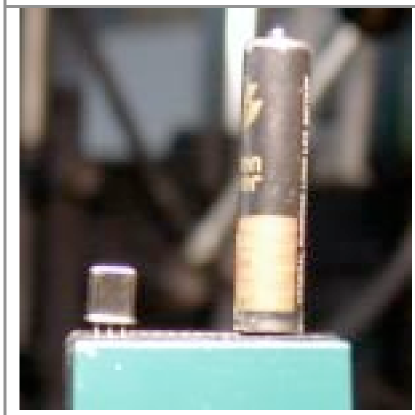
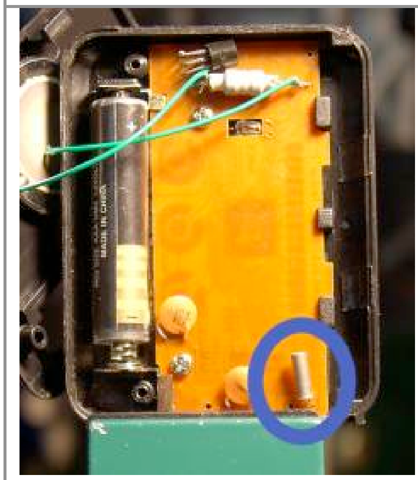
Crystals and Oscillators - The Physical Body

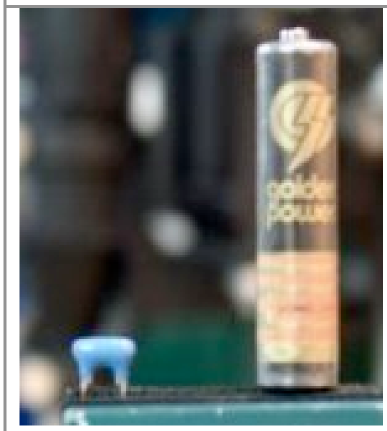
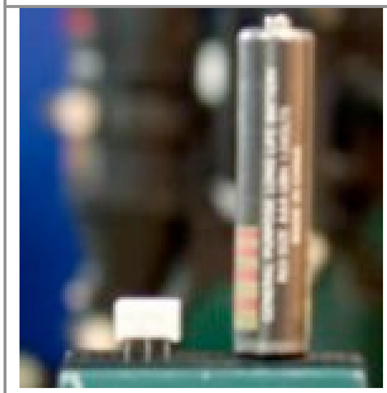
When it comes down to it, there isn't a huge variation in the sizes and shapes of crystals and ceramic resonators. Well, actually there is but you will probably only encounter a few :p

Here are a few photographs of different crystals and resonators I have. Photographing small shiny things is hard when you don't have the proper diffused light source, so I've included a description of each photograph, as well as the size of each.

The battery is an AAA cell, measuring 44.4mm from tip to tip, and 9.9mm in diameter (all measurements have been rounded to 1 decimal place and are in mm)

Picture	Frequency	Notes	Size and Package
	1.00000MHz	This is a big one; normally you probably won't see these too often.	18.4 X 19.4 X 7.9 mm Metal can packaging
	---------	This is the normal size for a crystal. This one didn't have any markings on it to indicate frequency and could be almost any value.	9.9 X 12.9 X 3.5 mm Metal can packaging

	6.000MHz	This is the same size of above, but is covered by a sort of rubber heat-shrink looking thing - this was done by the manufacture as the markings are printed right on the rubber.	10.4 X 12.3 X 3.9 mm Metal can, covered in rubber
	3.9936MHz	This is another fairly popular crystal size, as it takes up a lot less vertical space.	10.0 X 4.1 X 3.5 mm Metal can packaging
	45.0MHz	At first this looks like a crystal to the untrained eye, and in fact I thought it was. It wasn't until Rick Brown pointed out to me that in fact this device is a 45 MHz filter and not a crystal.	
	32.768 KHz	This is a picture of a watch crystal. There isn't any markings on it, but you know its probably a watch crystal as it is (1) in a clock device (2) its round	2.7mm X 8.1 mm Round metal can

Picture	Frequency	Notes	Size
	3.58MHz	Two pin resonator, but a lot of two and three pin resonators look like this.	7.8 X 3.6 X 3.1 mm Bluish blob is packing (very technical, eh :)
	4.000 MHz	This is a three-pin resonator	8.2 X 4.4 X 3.8 mm White rectangular piece of plastic-like material

A Quick Note On Crystal / Resonator Markings

When you see a crystal, there is a BIG difference between 4.00MHz and 4.000000000MHz. That is that the 4.00MHz crystal is guaranteed to be 4.00MHz, but not more accurate. So in reality the 4.00MHz crystal could be 4.0038297173MHz, as only the first two decimal places are accurate. But the 4.000000000MHz will be accurate to nine decimal places, so in reality it could be 4.00000000042830840MHz.

Also the frequency isn't always marked in MHz. Sometimes it is marked in Hz or KHz, so the crystal may only say 4000, and that's it! So you are left to wonder if that number is Hz or KHz. Most likely that is KHz, so it will be 4.000 MHz. Just think about what would make the most sense, and normally you should be right!

Where to get Crystals / Resonators and Final Remarks

Why You Need a Clock Source

There are a few places to get them. Normally your local electronic store has them in stock. However, you can also get them off on-line places such as

- http://www.digikey.com/
- http://www.arrow.com/
- http://www.future.ca/
- http://www.future-active.com/
- http://www.ied.pios.com/
- http://www.mouser.com/

A good site to look at for information on crystals is http://www.tla.co.nz/xtal1.html They have lots over there. Also a web search should reveal something as well.