Building a Basic AVR System:

1. How To Treat Your AVR Processor (and yourself) Right!

We already know, of course, that to satisfy the course requirements you must use an AVR processor in your final project. But have you considered what you need to satisfy the AVR's requirements? You may need one or more of the following items:

- ?? An IC socket
- ?? Bypass capacitors for power filtration (0.1uF and 10uF)
- ?? A suitable **REGULATED** power source
- ?? One (or maybe two) crystals with load capacitors (or the configuration to use a different oscillator source)
- ?? A reset button

That covers the basic needs of the processor, but what about some helpful hardware features for you:

- ?? Debugging LEDs
- ?? Programming interface (PonyProg or STK500)
- ?? Serial communication interface (good for debugging too!)

This page and the attached schematic will help you get the basics up and running quickly so you can concentrate on the unique parts of your project instead.

2. The Specifics

2.1. You Must Socket Your AVR Processor! Enough said!

It's recommended that you socket all other chips wherever possible too!

2.2. Bypass Capacitors

Connect a 0.1uF capacitor across the power pins of the AVR processor to filter, or "bypass", power supply noise. Although the AVR processors are extremely power efficient, often running off as little as a few milliamps, they can demand large currents during the instant when output pins switch states. These currents, sometimes as large as a few amps, can cause the processor to reset, glitch, or behave erratically if there isn't a capacitor nearby to provide current during this instantaneous power surge. A bypass capacitor is a good idea for each and every logic chip on your board, and you should place the capacitors as close as possible to the power pins of their associated chip. Finally, it's a good idea to place a 10uF capacitor at the place where power enters the board to filter any larger glitches.

2.3. Power Supplies

Although you're welcome to use the regulated power supplies in lab to power your project, you may find these to be a less-than-ideal solution. AVR processors can run from 4.0-5.5V (L-versions 2.7V-5.5V) and should be operated from a regulated power source (constant voltage). Fortunately, it's easier to get a constant voltage than you might think. An old computer power supply can usually provide +5V, +12V, -5V, -12V and maybe +24V. Or for a smaller solution, combine a wall-adapter brick and a linear

voltage regulator chip like the LM7805 or LM3940-5.0 from National Semiconductor (http://www.national.com). These regulator chips have only three leads, Vin, GND, and Vout, and can usually regulate inputs from 6-24V to a nice smooth 5V. They're cheap too: $0.50-1.00.

2.4. Crystals and Oscillator sources

Most AVR processors need a crystal from which to generate the clock that synchronizes processor operation. While you should consult the datasheet of the specific processor you plan to use, most AVRs can use crystal frequencies of 0-8MHz. Lower frequencies mean that the processor runs slower, but also uses less power. Most processors also require small ~22pF capacitors to be placed between the crystal pins and ground (see sample schematic for details). These capacitors help the oscillator start when powered up, and they are needed!

Also, don't forget that you have to configure many of the processors for the type of clock source you wish to use. The Atmega163, for example, has more than a dozen possible clock configurations (accessible from the "Fuses" section of the AVR studio programmer or PonyProg). Read the datasheet for specifics about what each configuration means.

2.5. Reset Button

Sometimes it's easy to forget that a reset button might be convenient, or even necessary. The sample schematic has an example of how to connect one.

2.6. Debugging LEDs

Using LEDs to indicate when you've reached a certain place in your program is a tried and trusted method of debugging, even if it's not exactly high-tech. Although you may quickly find that using the UART or maybe an LCD is more informative, building one or two LEDs into your system for status and debugging can still be really helpful. Blinking a single simple LED from software can help you verify that your processor has power, has a working clock, and is executing instructions.

2.7. Programming Interface

The ability to program your processor right in your project is sometimes an indispensable tool. It helps speed code development, testing, and makes future upgrades to your code, after the class is over, much easier. Fortunately the free programming software PonyProg does all the hard work for you. Adding a serial connector, a few diodes, resistors, and one transistor is all that's necessary to make your project *Firmware Upgradeable*.

2.8. Serial Communication Interface

Many of you may wish to use the AVRs serial port (UART) to provide a means of transferring data or controlling your code. A serial port is also a great way to get debugging information out to the world. Although we've discussed how to use the UART at length already, there's still the issue of converting the logic-level serial signals produced by the AVR to +/-10V signals that can be connected to a PC serial port. Dozens of manufacturers make parts which can do this conversion in a single simple chip. Perhaps the best known is the MAX232 made by Maxim Semiconductor (http://www.maxim-ic.com).