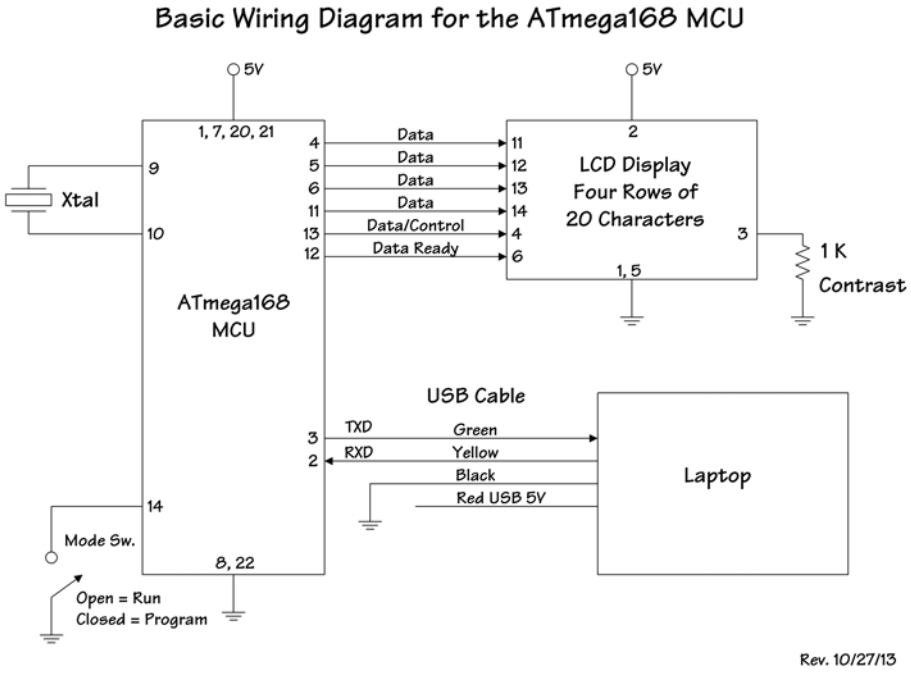
MCU Overview and Approach to Projects

1. The Approach

The approach I am taking is to get as close to the hardware as possible (I'm really just a hardware geek). This approach reflects my background in product design. When you design a product you are continually thinking about robustness and simplicity. That means the product has to work under all kinds of situations and you want to reduce the parts count to a minimum – every resistor, capacitor and connector costs money and makes the product more complex and expensive.

2. Other Approaches to MCU Design

Products like the Arduino (which uses the same MCU that I'm using) and the Raspberry Pi are wonderful tools for introducing people to hardware and software. But all the extra built in components (and operating system in the case of the Raspberry Pi) separate you from understanding what's really going on with the hardware. Here is my basic wiring diagram:



So, I'm starting with just the MCU itself (a 28 pin chip) and adding only the components necessary to complete the projects. This way I will understand the architecture of the MCU and what every pin does. Also, if I ever want to package the application as a product it will have minimum parts count.

You may be tempted to say, "Only 28 pins, how complicated can that be?" Well, the data sheet for the Atmega168 is 376 pages long. So, there are a few details to be understood. ☺

3. Tools

In the "old days" I used to do machine language programming for the Zilog Z80 on coding sheets. Now that's really ancient! Later I graduated to writing in assembler with hex (still pretty ancient). So, the routine used to be:

1. Write the program – including op codes and addresses
2. Burn an EEPROM with the program
3. Stick it in a ZIF socket and see if it would run
4. If it didn't run you try all kinds of strange and exotic things like inserting special loops and checkpoints to isolate the problem
5. Correct the program – which could involve a lot of eraser work and moving stuff on the coding sheet
6. Erase the EEPROM and burn the new program, etc.
7. You get the picture

Today we have higher level programming languages and better tools. BUT I still want to be as close to the hardware as possible. The main tools I am using are:

1. C Compiler: Today we don't write in machine language (even though you have to realize there is still such a thing). Higher level languages make it so much easier, and we have compilers that will do all the drudge work of converting it to machine language (1's and 0's that the machine understands).
2. Programmer's Notepad: A C program is just a text file, but special word processors like the Programmer's Notepad are really helpful because they check syntax and put things in different colors. Aren't we blessed! ☺
3. Make File and USB to Serial Cable: Downloading the program you just wrote on your computer to the non-volatile memory on the MCU is easy. The Make File tells the compiler what files to compile and in what order. So, running the Make File will program the MCU! The USB to Serial cable does the electrical conversion so you can talk to the Tx and Rx pins of the MCU.

4. Let's Get Started!

So, that's the approach and the tools we will be using for all of these amazing projects. So Let's get started!