Interfacing with Microcontrollers

Now that you know what programming languages and IDEs are all about, and the fundamentals of DC electrical flow, we need to look at the device we are going to be using for the rest of the class.

We are going to use a standard computer to actually do our programming, but we are not programming it. Instead we are programming another device called a **Microcontroller**.

You may be familiar with the term Microprocessor. When it was coined, the CPU, Central Processing Unit, of a computer system was a large and complicated component. Miniaturized (and simplified at the time) versions were then called Microprocessors. The CPU performs the basic operations like addition, subtraction, moving data, etc., and is the main 'programmable' component in a computer system. However, it needs other components to operate - memory, storage, input & output (I/O) devices, etc.

Microcontrollers are small and simplified entire computer systems on a single chip, sometimes called a System on a Chip or **SoC**. A microcontroller contains the CPU,, RAM, Flash storage and various I/O mechanisms.

Also referred to as *Embedded Systems*, Microcontrollers are typically used to provide certain functionality in devices which are not primarily computing devices, such as printers, microwave ovens, security systems, televisions, and automobiles. Some devices, like a car, may have many individual microcontrollers, some of which may communicate with each other. For example, a microcontroller might monitor the rotational speed of each wheel and report it to several others which in turn might take various actions such as regulating the hydraulic pressure of the braking system to prevent individual wheel lock-ups (i.e. anti-lock brakes), or regulate the power supplied to a particular wheel to keep the vehicle on course (i.e. traction control).

Regular desktop/laptop computers have multiple micro controllers in them — Each hard disk or solid state drive (aka HDD or SDD) will have a microcontroller (or 2 or 3) in it which controls and monitors the actual physical storage and the communication with the primary system. Other peripherals will use them as well, like the display, keyboard, and even the battery in a laptop has a microcontroller which manages its charging.

In this course we are going to use a microcontroller produced by a company called Atmel, in a line called the **AVR** series. Atmel makes many microcontrollers with different sizes, speeds, storage, etc. They even make Radiation Hardened (RadHard) version for use in space. Atmel has shipped over 1 billion AVRs packed into everything from light switches to BMWs. There are many others from other companies including Intel, Motorola, Microchip, and Texas Instruments (Microchip recently acquired Atmel for $3.6B.)

The AVRs from Atmel have gained popularity, especially in the hobbyist and maker market over the last several years primarily due to the efforts of the Arduino group. They built several small carrier boards for various AVRs and, most importantly, a simple IDE and language extensions which are free and open source. In addition to using the Arduino IDE, we are using an Arduino board design as well, specifically a 'clone' of the Arduino **Nano**.

The carrier board provides us with several resources. First of all, it connects the various I/O pins of the very tiny surface mount (SMD) AVR chip to the still-small-but-usable .1" spaced (pitch) pins. It also provides a reset switch, activity LEDs, voltage regulation (to provide the chip with a constant 3.3 and 5v dc from a single 5v input), a crystal oscillator for timing, and most importantly a USB interface for programming.

The Nano we are using runs at 16Mhz (million cycles per second), has 32K of Flash storage, 2K of RAM, 14 Digital I/O pins, and 8 Analog I/O pins. A complete schematic and pinout description is provided in the appendix.

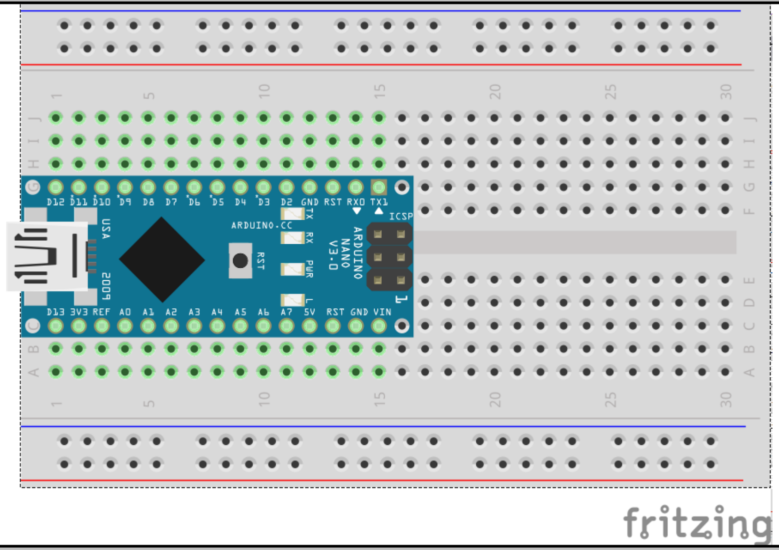
Of the 32K of flash storage, 2K is dedicated to some special software called the ***bootloader.*** When the USB port is connected and a serial link established, the bootloader is activated. The bootloader receives the incoming binary data of your compiled program from the host (your computer in this case) and stores it in the flash, then begins executing it once it has been completely received.

When using the Arduino IDE, it just takes a single click of the 'Upload' (right arrow) icon to both compile your program (if needed) and initiate the transfer. You can watch the progress both in the black activity area and the bottom of the IDE and on the TX (Transmit) & RX (Receive) LEDs on the Nano.

The Serial connection (over USB) can be used for other things too, as we will see in the next section.

In this exercise we are going to take the simple program you just corrected, compile it and push it down to the AVR and observe it running.

Experiment: Interfacing with your microcontroller.

Components	**Wiring Diagram**
✓Computer ✓Arduino IDE	

D12 D11 D10 D9 D8 D7 D6 D5 D4 D3 D2 GND RST RX0 TX1
D13 3V3 REF A0 A1 A2 A3 A4 A5 A6 A7 5V RST GND VIN
ARDUINO.CC
ARDUINO NANO V3.0
ICSP
RST
USA
2009
TX
RX
PWR
L
fritzing

Connection Instructions

open your microcontroller and breadboard. Place controller on breadboard as shown, being careful not to bend or break any pins. It may require some finesse. Ask for help if you can't get it after a while

Sketch(es)	needsWork.ino (your edited and fixed version from the previous section)

Analysis Questions

Did your program upload and run okay?
If not can you see the error?
If so, how do you know?
Is the 'output' as you expected?
What happens when you press the reset button — both the result and what do you think actually happens behind the scenes?

Programming Tasks

The Arduino app should still be open from the previous section, and should have your 'fixed' version of the program open. If you were unable to fix the program or have lost your fixed version, open uploadDemo.ino instead.

- Connect the provided USB monitor device to your computer's USB port.
- Connect the Mini-B USB cable to the Arduino Nano from your parts box.
- Finally connect the other end of the USB cable to one of the ports on the USB monitor device.
- Double check that the Board is **Arduino Nano** and the Processor is **ATmega328** under the Tools pulldown menu.
- Now select the Port, also under Tools. (The exact port will vary, on a mac it will look like: /dev/`cu.wcusbserialXXXX` , on windows it will be a `COM` port number, on linux it will be `/dev/ttyUSBX`)
- Once selected, click the Upload button and observe the output area and the LEDs on your Nano.