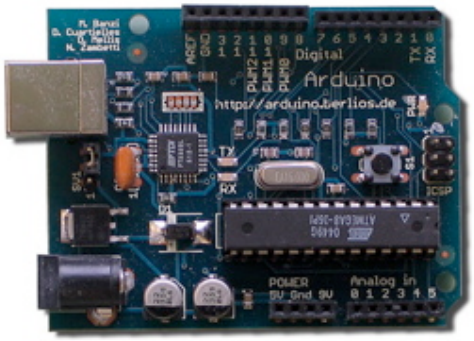
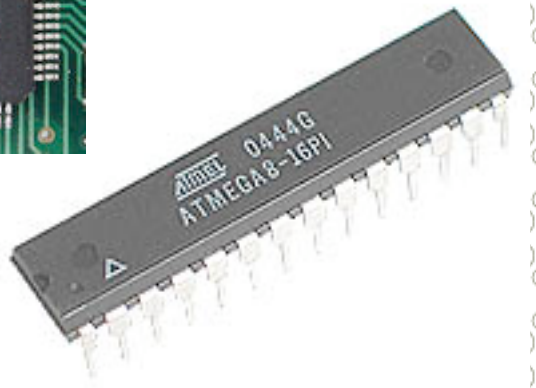
Introducing the Arduino…

- The Ardunio is a microcontroller development platform (not a microcontroller…)
- Its hardware/software system is open source
- It has a programming language very similar to Processing
- It is developed by an international group of designers and programmers
- You can find its home page at http://www.ardunio.cc

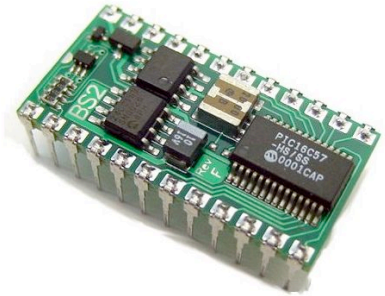


What is a microcontroller?

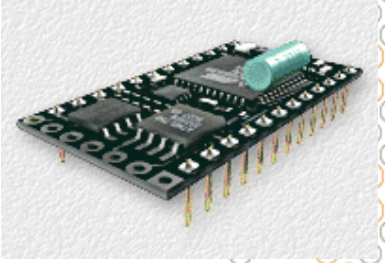


- A single Integrated Circuit (IC) with:
  - Central Processing Unit (CPU)
    - 8 bit to 64 bit
  - Input/Output Interfaces
    - e.g. Serial ports
  - Peripherals
    - e.g. Timer circuits
  - RAM
    - Data storage
  - ROM/EEPROM/Flash memory
    - Program storage
  - A clock
- A microcontroller is a type of microprocessor that incorporates all the memory and interface systems necessary to operate as a standalone computer.
- Microcontrollers are generally small and low-cost, and typically are used to control electronic devices.
- We find them inside cars, homes, electronic toys, and increasingly inside art installations…

Microcontroller Products

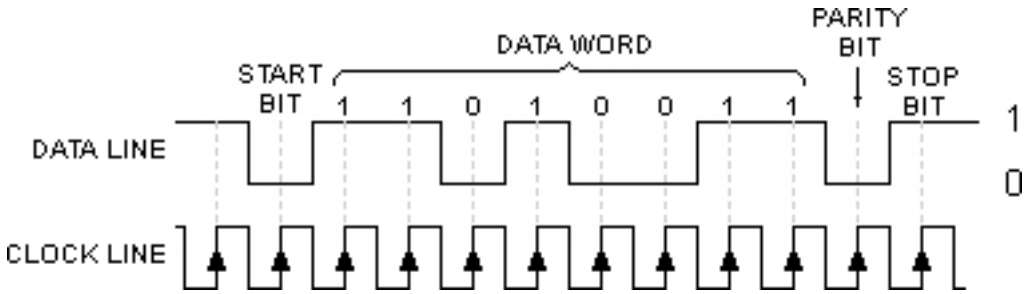


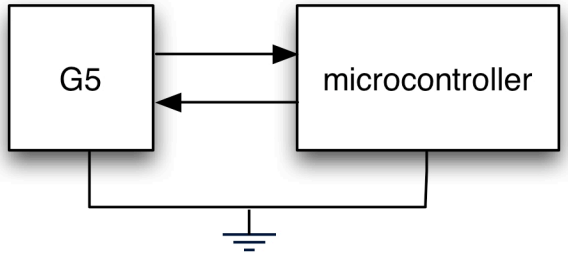
- There are a large number of manufacturers of microcontrollers, mostly building products for embedded devices:
  - ATMEL AVR
  - Dallas / Intel 8051
  - Microchip PIC
  - Motorola 6811
- Then there are an ever-increasing number of smaller companies that produce packaged platforms for using microcontrollers:
  - Arduino (AVR)
  - Basic Stamp (PIC)
  - HandyBoard (6811)
  - BX-24 (AVR)
- Typically these platforms are commercial, closed source ventures whose products are mostly used by hobbyists. They all use similar paradigms, meaning that browsing their forums can often be a good source for problem-solving circuits, code, electronics, etc.



Serial Communication

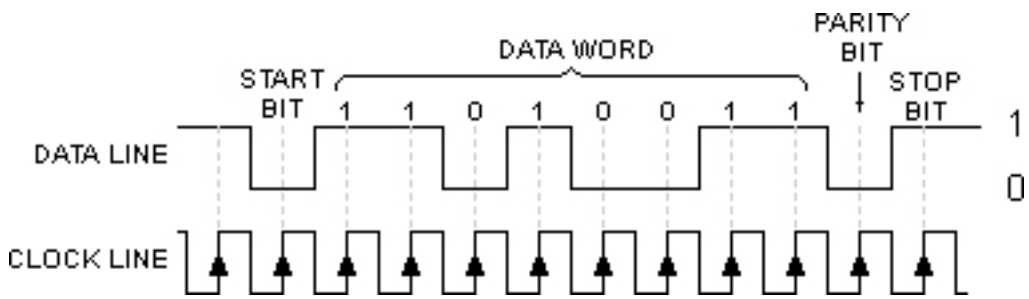
- Most microcontrollers are standalone, not needing to be tethered to another computer in order to run.
- In order to upload a control program however, we typically connect them to the serial port of a computer.
- Serial communications basically involves the sending of bytes of data one bit at a time.
- Serial cables usually have a send wire and a receive wire, known as TX and RX. This allows them to perform full duplex communications, where data can be sent and received simultaneously.





Serial Timing

- In order to set up a serial communication, we need to define:
  - Baud Rate
    - The speed at which data travels between computer and microcontroller.
    - Typically 9600 bits per second.
  - Start bit
    - Lets the receiver know a transmission is coming.
    - Usually dealt with directly by a serial library.
  - Data bits
    - The data that you want to send
    - E.g. “H” in binary is 01001000 (ASCII 72)
  - Stop bits
    - Lets the receiver know transmission is finished.
  - Parity bit
    - A simple error checking protocol that states whether the number of 1s in the byte sent are odd or even. This is usually not used.



Arduino: Hardware

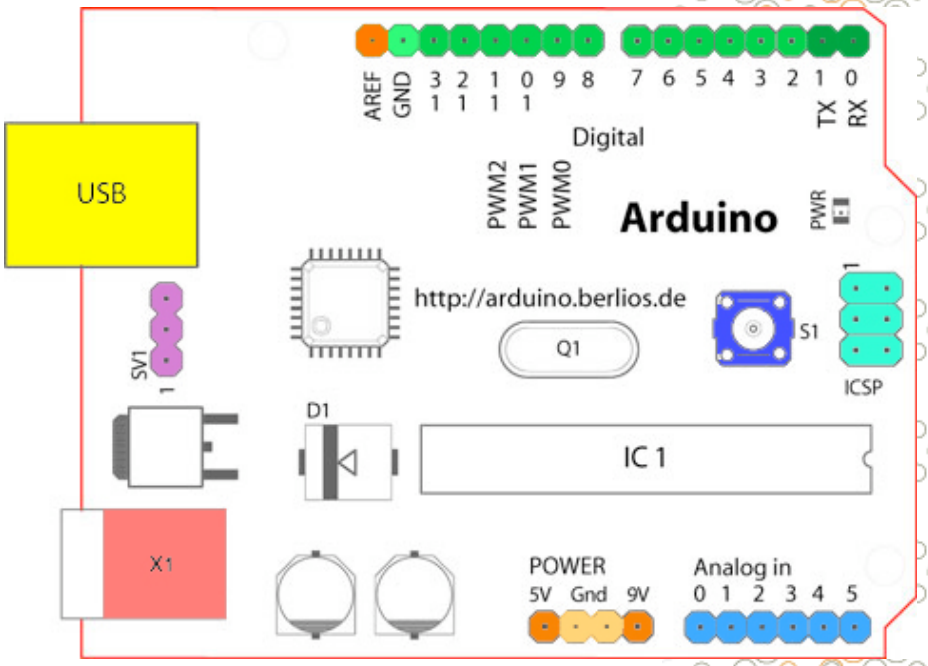
- The Arduino NG uses the Atmel ATMega8 microcontroller.
- Some features of the ATMega8:
  - 8 bit processor can handle values from 0-255 in a single operation
  - Flash Memory (8k) stores the programs you write.
  - I2C, SPI serial interfaces allow communication with specific types of serial devices.
  - USART serial interface converts program data into serial data
  - Interrupts allow us to establish background events that can interrupt the program's regular execution cycles.
  - 23 I/O lines allow data in and out of the microcontroller
  - 6 channel ADC (10 bit) converts analog signals into digital numbers

Arduino: Interface

- There are several different arduino packages, allowing both serial and USB interface cables to be used for communication between board and computer.
- The USB port on the arduino works with an FTDI chip to convert RS232 to USB.
- What the USB does:
  - By connecting the arduino to a computer, we can upload sketches to the board.
  - When a program is running on the arduino, it can send and receive serial data to a computer via the USB cable. This allows us to control computer events from the arduino (eg video, audio, web) and vice versa (eg motors, outputs, etc.)
  - The USB cable can also be used to power the arduino directly from a computer.

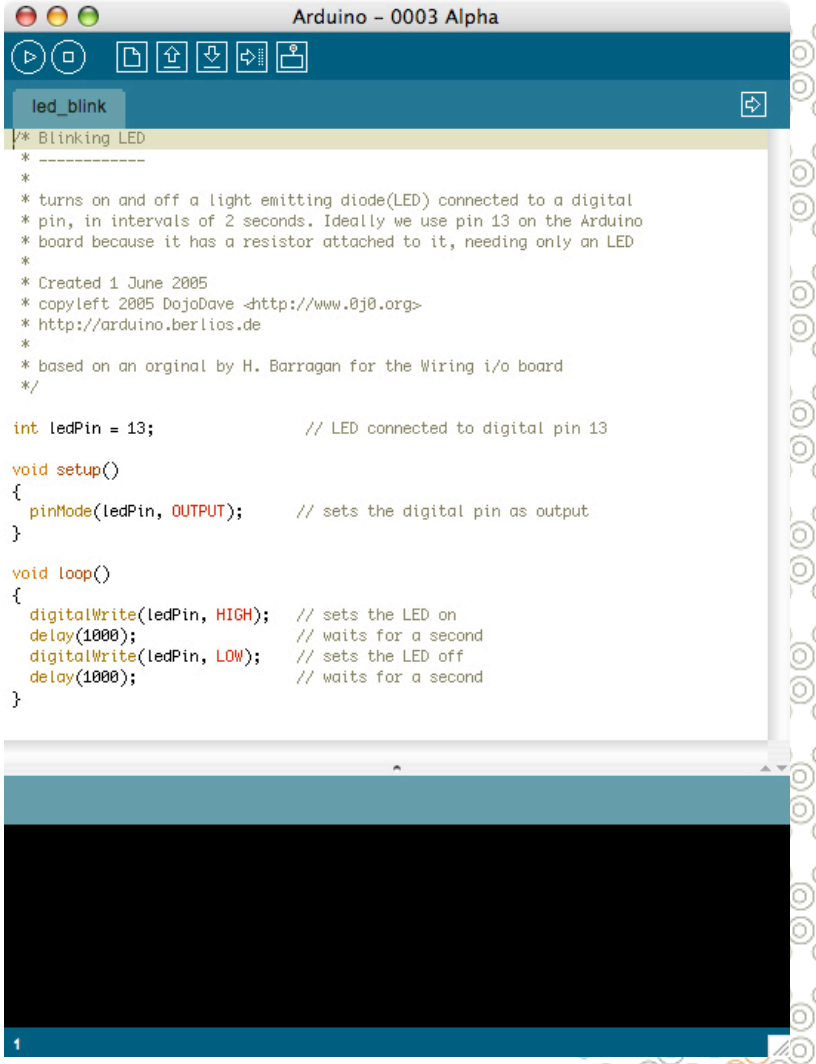
Arduino: Board Layout

- Power LED (green)
- ICSP (in circuit serial programmer)
  - For programming the microcontroller directly (eg bootloader)
- Reset button
- AVR
- Polarity protection diode (D1)
- Surge protection capacitors
- TX/RX LEDs
- 16Mhz crystal
- Power pins -- 5v/GND(3)/9V
- Analog input pins (6)
- USB socket
- Power selector jumper -- USB/ext.
- 5V regulator for ext. power
- Ext. power socket (9-15V)
- USB-serial chip
- Analog Reference pin
  - For tweaking the ADC
- Digital input/output pins (14)
  - Pin 13 has a resistor attached
  - Pins 0 and 1 are TX/RX pins



Arduino: Software

- Arduino boards can be controlled using an implementation of Wiring, which is a version of Processing developed specifically for electronic I/O.
- Arduino looks like Processing, but is actually built in C, so there are a few differences to look out for.
- Arduino can be downloaded from http://www.arduino.cc
- We are currently using version 004, although 005 is available.



Arduino: Programming

- Arduino has two reserved functions:
  - void setup()
  - void loop()
- There is no pop-up display window, hence void draw() is not special. Loop() can be considered to do the same thing as draw() for the Arduino.
- There are three types of variable in Arduino:
  - char
  - int
  - long
- Arrays are created slightly differently from Processing:
  - int[] myArray; // in Processing
  - int myArray[]; // in Arduino
- Arduino has a few reserved constants, which do not need to be defined:
  - HIGH // 5 volts
  - LOW // 0 volts
  - INPUT // pin is input
  - OUTPUT // pin is output
- Conditional statements are the same as in Processing
- Functions can be defined the same as in Processing
- LED_blink.pde

Arduino: Communication with Processing

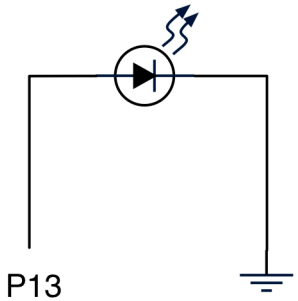
- Arduino talks to Processing (or any other programming language) by sending commands via the serial port.
- Our first step therefore is to tell Arduino which serial port we are planning to use:
  - Tools > Serial Port > /dev/cu.usbserial
- Both Arduino and Processing have serial libraries. In Arduino, it is present by default.
- Arduino has five core serial commands:
  - Serial.begin(baud);
    - Sets up the baud rate (eg. 9600) of the serial communication. Put this into void setup()
  - Serial.available()
    - Use when Arduino is receiving serial data. It tells us whether there is any data ready to receive -- i.e. use it in a conditional statement.
  - Serial.read()
    - Also for receiving serial data. Once you have established there is data to receive, use this to read it into a variable.
  - Serial.print(data, format)
    - Use this to send serial data or strings. Data can be followed by an optional format (DEC, HEX, OCT, BIN, BYTE).
  - Serial.println(data, format)
    - Same as Serial.print, but also sends an newline (10) and carriage return (13) character.
- Arduino_serial.pde

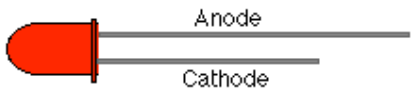
Processing: Communication with Arduino

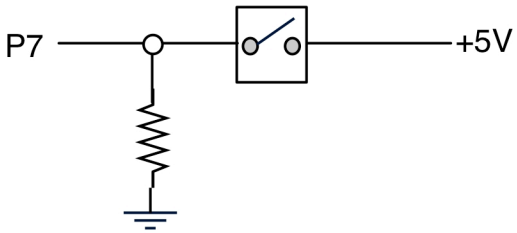
- In Processing, the serial library must be explicitly imported:
  - import processing.serial.*
- When you import the serial library, the first thing you need to do is create an instance of the class Serial (more on classes in a few sessions…)
  - Serial myPort;
  - myPort = new(Serial(this, Serial.list()[1], 9600);
    - Here, an instance of the class Serial is created. The serial class has three arguments: the parent class (Serial), the port and the baud rate.
- In order to work out which port we use, typically we choose the dev/cu.usbserial by calling
  - println(Serial.list())
- Once we have the instance set up, we are ready to interact with the serial port. In order to read the port, we will do
  - while(myport.available() > 0){
  - serialEvent(port.read());}
- serialEvent is a called automatically when data is available. We create a
  - void serialEvent(int serial)
- function, within which we can process the incoming data.
- Processing_serial.pde

A few things you need to know…

- We will be covering digital input and output with the Arduino thoroughly in the next class. Today, we are focusing upon getting up and running with the Arduino and Processing.
- To do the Lab exercises today, you need to use an LED and a pushbutton.
  - LED: put an LED into Pin 13 and GND.
  - Pushbutton: use your breadboard to connect one side of the switch to +5V and the other to Pin 7 and GND.
- Uploading a program to Arduino is straightforward:
  - In order to compile a program, use the play button 
  - In order to upload a program, first press the reset button on the board, then click the upload button 
  - In order to monitor the serial port, use the serial monitor 
- When using the Arduino pins, we need to define whether we want them to be inputs or outputs:
  - pinMode(pbutton, INPUT);
  - pinMode(ledPin, OUTPUT);
- In order to read the value of a Pin, we use
  - digitalRead(pbutton)
- In order to change the value of a Pin, we use
  - digitalWrite(ledPin, HIGH);

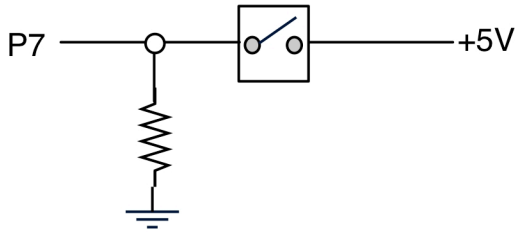






LAB exercise: serial data from Arduino to Processing

- Download Arduino_serial_send.pde and Processing_serial_rec.pde from the DXARTS 471 website.
- Set up your Arduino environment and your Processing environment for serial communications.
- Connect a pushbutton to your Arduino, verify it generates a serial message using the Arduino Serial Monitor.
- Verify that you can receive the switch message in Processing.
- Modify void serialEvent in Processing to control the display window based upon the switch input.
- Work in pairs. If you finish early, help somebody else.
- Make sure you understand the code (you'll need to write your own soon!)



And now for something completely different…

- To send data the other way, we need to identify an event in Processing.
- Mouse Events
  - void mousePressed()
  - {
  - port.write("B");
  - }
- Conditional Events
  - if (val > 123)
  - {
  - port.write("B");
  - }
- Motion Events
  - if (myLine > 100 && myLine < 120)
  - {
  - port.write("B");
  - }

ASCII

- On your mouse pads, you will see the ASCII Character Chart.
- Sending data via serial can be made more efficient by sending data using its ascii character:
  - 65 = 101(ASCII)
  - “A” = 65(ASCII)
  - “65” = 54, 53 (ASCII) // two bytes!
- Conversion at either end is simple:
  - int a = 65;
  - char b = char(a);

  - char c = ‘B’;
  - int d = int(c);
- So watch out when you are using conditional statements to look for incoming bytes -- make sure you are testing for their ascii values.

LAB exercise: Processing to Arduino (and then back again…)

- Reform your groups and download the rest of the files for this week.
- Wire up your LED as below.
- Then create an event in Processing that will generate a serial write, and read it into Arduino. Make the Arduino's LED light go on and off when it receives the message.
- Then build code to allow two-way intelligent responses between Processing and Arduino. For example, have your pushbutton make something happen in Processing and then receive a message *back* from Processing to turn on the LED. And vice versa…