Arduino

James D Emery

edited: 11/22/2012

Contents

1 Getting Started Book 3
2 Downloading and Running Software 3
3 Craig Berscheight Arduino Servo Class 3
4 Arduino Cookbook 3
5 H-Bridge 5
6 Arduino Class, April 2012 5
7 PID (Proportional Integral Derivative Controller) 5
8 Programs 6
  8.1 The Arduino ”Hello World” Program, Blink.ino 6
  8.2 Cycling LEDs, Finding a Bug, cylon_bug.ino 6
  8.3 Stopping LEDs With a Button, cylon_button.ino 8
  8.4 Control by Reading Serial, cylon_serial_toggle.ino 10
  8.5 Debouncing the Button, cylon_button_toggle.ino 12
  8.6 Analog Input, analoginput.ino 14
  8.7 Reading a Potentiometer Voltage on an Analog Pin, analoginputmap.ino 15
  8.8 Servo Motor, Reading Analog, Writing Digital, Using Map and the Servo Library: knob2.ino 16
1 Getting Started Book

*Getting Started With Arduino*, Massimo Banzi, Make Makezine.com, O'Reilly Publications. Banzi is the creator of the Arduino.

2 Downloading and Running Software

Downloading software: arduino-0018.dmg, for mac. The image file is mounted as a disk. You can start the software, but it will vanish after use. So drag folder the to the application folder. Load a sketch (a program source file), an example in servos, change the sketch program for different angles. Continuous rotation servos, rc servos, can be run with an 555 timer, pulse width modulation. Upload a sketch to the Arduino through the USB cable, blinking light.

3 Craig Berscheight Arduino Servo Class

The class lasted 4 weeks, 1 session per week. I attended sessions 1, and 4. I was in San Francisco for the others, so the class must have been in February 2010. The classes were on Friday nights in the cave. I purchased a servo motor from a hobby store on north oak. We must have run the servo sketch example that came with the software.

4 Arduino Cookbook

http://arduino.cc/Main/Software

http://www.7-zip.org/
Figure 1: The pins of the Arduino. On the right are the digital pins, on the left the analog pins. Some of the digital pins can write PWM (Pulse Width Modulation), which is a signal of about 960 Hz, with a variable duty cycle. So the average voltage can vary between 0 and 5 volts.
5 H-Bridge

An H-Bridge is a switch used to reverse the direction of a DC motor by changing the current direction, or to change the current direction in a coil. It is called an H-bridge because the switch circuit diagram looks like the letter "H" with the horizontal center edge being the location of the motor, and each of the legs and arms of the "H", being switches. These could be mechanical switches, or say FETs, for electronic controlled switching. In driving some stepper motors, the phases are changed by sending current to different stator windings, and also by changing current direction in such windings, to change say a North pole to a south pole. This direction changing is often done by electronic drivers circuits, which is often an IC which incorporates an H-bridge.

6 Arduino Class, April 2012

There will be introductory classes on Arduino at the Hammer Space, 440 E 63rd street, in April. There is no charge, except for possible materials, and nonmembers are welcome I think:

Jestin Stoffel and Josh Bookout:
Arduino: Class lab
Starting Saturday April 7, 14, 21 and 28 10am to 2pm.

Arduino books I have:
There is a 2nd edition published September 2011.
The following book is good:
Arduino Cookbook by Michael Margolis (Dec 30, 2011)
There are probably tutorials on the internet.

7 PID (Proportional Integral Derivative Controller)

See Jim Emery Robotics.
8 Programs

8.1 The Arduino ”Hello World” Program, Blink.ino

/*
   Blink
   Turns on an LED on for one second, then off for one second, repeatedly.

   This example code is in the public domain.
 */

void setup() {
   // initialize the digital pin as an output.
   // Pin 13 has an LED connected on most Arduino boards:
   pinMode(13, OUTPUT);
}

void loop() {
   digitalWrite(13, HIGH); // set the LED on
   delay(100); // wait for a second
   digitalWrite(13, LOW); // set the LED off
   delay(1000); // wait for a second
}

8.2 Cycling LEDs, Finding a Bug, cylon_bug.ino

This program lights a set of five LEDs with the positive legs of the LEDs connected to arduino pins 2 through 6. There is a single resistor connected to the negative legs of all the LEDs. The program lights the first LED, waits for 50 milliseconds, then lights LED 2, and so on. When the last LED at pin 6 is lit, the variable dir changes sign so that pin 5 is the next led, and so on. There is a bug in the program which is illustrated by using the serial write which is viewable by clicking the button on the far right of the toolbar. The bug is in the initial value for dir which is 1. When the loop starts the curPin is 2, so pin 2 is set high and all other pins low. But in checking the direction the first time the direction is switched because the current pin
equals the start pin. So the direction becomes -1. But then the next value of
the current pin will be 1, and after that 0,-1,-2,-3 ... , so the current pin will
never be in the range 2 to 6, and the LEDs will not light. To correct the bug
set the initial value of dir not to 1, but to -1. I am told that cylons are some
science fiction characters with a scanning laser beam in their forheads. This
element sequentially lights a set of LEDs, which apparently is supposed to
recall the Cylon scanning.

// cylon_bug.ino
const int startPin = 2;
const int endPin = 6;
int curPin;
int dir;

void setup() {
  // set up serial
  Serial.begin(9600);

  // initialize the pin modes
  for(int i = startPin; i <= endPin; i++) {
    pinMode(i, OUTPUT);
  }

  // initialize the current pin and the direction
  curPin = 2;
  dir = 1;
}

void loop() {
  Serial.println(curPin);
  // write to all the pins, the current will be set HIGH
  for(int i = startPin; i <= endPin; i++) {
    digitalWrite(i, (curPin == i) ? HIGH : LOW);
  }

  // check if the direction needs to be switched
  if(curPin == endPin || curPin == startPin) {
    dir = dir * -1;
  }
}
8.3 Stopping LEDs With a Button

This program stops or starts the LED switching with a button. When the buttonpin, which is pin 12, is high, the LED cycling is on. When low the cycling is off. The button is a switch connecting the 5 volt supply voltage on the arduino to pin 12. When depressed there is 5 volts at pin 12. when not depressed, pin 12 is connected to ground through a resistor $R$. The resistor pulls down the voltage to pin 12 to ground, when the button is up.

```cpp
const int startPin = 2;
const int endPin = 6;
const int buttonPin = 12;
int curPin;
int dir;
int buttonState;

void setup() {
  // set up serial
  Serial.begin(9600);

  // initialize the pin modes
  for(int i = startPin; i <= endPin; i++) {
    pinMode(i, OUTPUT);
  }

  pinMode(buttonPin, INPUT);

  // initialize variables
```

```cpp
```
Figure 2: This program stops or starts the LED switching with a button. When the buttonpin, which is pin 12, is high, the LED cycling is on. When low the cycling is off. The button is a switch connecting the 5 volt supply voltage on the arduino to pin 12. When depressed there is 5 volts at pin 12. When not depressed, pin 12 is connected to ground through a resistor $R$. 
curPin = 2;
dir = -1;
buttonState = LOW;
}

void loop() {
    buttonState = digitalRead(buttonPin);

    if(buttonState == HIGH) {
        cylon();
    }
}

void cylon() {
    Serial.println(curPin);
    // write to all the pins, the current will be set HIGH
    for(int i = startPin; i <= endPin; i++) {
        digitalWrite(i, (curPin == i) ? HIGH : LOW);
    }

    // check if the direction needs to be switched
    if(curPin == endPin || curPin == startPin) {
        dir = dir * -1;
    }

    // advance to the next pin
    curPin += dir;

    // wait
    delay(50);
}

8.4 Control by Reading Serial, cylon_serial_toggle.ino

Opening the serial monitor, the LED cycling is controlled by typing a 't' in the monitor window.
const int startPin = 2;
const int endPin = 6;
int curPin;
int dir;
boolean active;

void setup() {
  // set up serial
  Serial.begin(9600);

  // initialize the pin modes
  for(int i = startPin; i <= endPin; i++) {
    pinMode(i, OUTPUT);
  }

  // initialize the current pin and the direction
  curPin = 2;
  dir = -1;
  active = false;
}

void loop() {
  if(Serial.available()) {
    if(Serial.read()== 't') {
      active = !active;
    }
  }
  if(active) {
    cylon();
  }
}

void cylon() {
  Serial.println(curPin);
  // write to all the pins, the current will be set HIGH
  for(int i = startPin; i <= endPin; i++) {

digitalWrite(i, (curPin == i) ? HIGH : LOW);
}

// check if the direction needs to be switched
if(curPin == endPin || curPin == startPin) {
    dir = dir * -1;
}

// advance to the next pin
curPin += dir;

// wait
delay(50);

8.5 Debouncing the Button, cylon_button_toggle.ino

const int startPin = 2;
const int endPin = 6;
const int buttonPin = 12;
int curPin;
int dir;
boolean active;
int buttonState;
int debounceButtonState;
int prevButtonState;
long lastDebounceTime = 0;
long debounceDelay = 50;

void setup() {
    // set up serial
    Serial.begin(9600);

    // initialize the pin modes
    for(int i = startPin; i <= endPin; i++) {
        pinMode(i, OUTPUT);
    }
}
pinMode(buttonPin, INPUT);

// initialize the current pin and the direction
curPin = 2;
dir = -1;
active = false;
buttonState = LOW;
debounceButtonState = LOW;
prevButtonState = LOW;
}

void loop() {
    // read the state of the switch into a local variable:
    int reading = digitalRead(buttonPin);

    // check to see if you just pressed the button
    // (i.e. the input went from LOW to HIGH), and you’ve waited
    // long enough since the last press to ignore any noise:
    // If the switch changed, due to noise or pressing:
    if (reading != debounceButtonState) {
        // reset the debouncing timer
        lastDebounceTime = millis();
    }

    if ((millis() - lastDebounceTime) > debounceDelay) {
        // whatever the reading is at, it’s been there for longer
        // than the debounce delay, so take it as the actual current state:
        buttonState = reading;
    }

    debounceButtonState = reading;

    if(buttonState == HIGH && prevButtonState == LOW) {
        active = !active;
    }

    prevButtonState = buttonState;
if (active) {
    cylon();
}
}

void cylon() {
    Serial.println(curPin);
    // write to all the pins, the current will be set HIGH
    for (int i = startPin; i <= endPin; i++) {
        digitalWrite(i, (curPin == i) ? HIGH : LOW);
    }

    // check if the direction needs to be switched
    if (curPin == endPin || curPin == startPin) {
        dir = dir * -1;
    }

    // advance to the next pin
    curPin += dir;

    // wait
    delay(50);
}

8.6 Analog Input, analoginput.ino

/*@ Analog Input
Demonstrates analog input by reading an analog sensor on analog pin 0 and
turning on and off a light emitting diode(LED) connected to digital pin 13.
The amount of time the LED will be on and off depends on
the value obtained by analogRead().

The circuit:
* Potentiometer attached to analog input 0
* center pin of the potentiometer to the analog pin
* one side pin (either one) to ground
* the other side pin to +5V
* LED anode (long leg) attached to digital output 13
* LED cathode (short leg) attached to ground
*/
* Note: because most Arduinos have a built-in LED attached to pin 13 on the board, the LED is optional.

Created by David Cuartielles
modified 30 Aug 2011
By Tom Igoe

This example code is in the public domain.

http://arduino.cc/en/Tutorial/AnalogInput

```c
int sensorPin = A0;  // select the input pin for the potentiometer
int ledPin = 13;     // select the pin for the LED
int sensorValue = 0; // variable to store the value coming from the sensor

void setup() {
  // declare the ledPin as an OUTPUT:
  pinMode(ledPin, OUTPUT);
}

void loop() {
  // read the value from the sensor:
  sensorValue = analogRead(sensorPin);
  // turn the ledPin on
  digitalWrite(ledPin, HIGH);
  // stop the program for <sensorValue> milliseconds:
  delay(sensorValue);
  // turn the ledPin off:
  digitalWrite(ledPin, LOW);
  // stop the program for for <sensorValue> milliseconds:
  delay(sensorValue);
}
```

8.7 Reading a Potentiometer Voltage on an Analog Pin, analoginputmap.ino

This is a variation of the previous program using map rather than delay.

```c
int sensorPin = A0;  // select the input pin for the potentiometer
int ledPin = 13;     // select the pin for the LED
int sensorValue = 0; // variable to store the value coming from the sensor

void setup() {
  // declare the ledPin as an OUTPUT:
  pinMode(ledPin, OUTPUT);
}

void loop() {
  // read the value from the sensor:
  sensorValue = analogRead(sensorPin);
  // turn the ledPin on
  digitalWrite(ledPin, HIGH);
  // stop the program for <sensorValue> milliseconds:
  delay(sensorValue);
  // turn the ledPin off:
  digitalWrite(ledPin, LOW);
  // stop the program for for <sensorValue> milliseconds:
  delay(sensorValue);
}
```
* the other side pin to +5V
* LED anode (long leg) attached to digital output 13
* LED cathode (short leg) attached to ground

* Note: because most Arduinos have a built-in LED attached
to pin 13 on the board, the LED is optional.

Created by David Cuartielles
modified 30 Aug 2011
By Tom Igoe

This example code is in the public domain.

http://arduino.cc/en/Tutorial/AnalogInput

*/

int sensorPin = A0;  // select the input pin for the potentiometer
int ledPin = 9;      // select the pin for the LED
int sensorValue = 0; // variable to store the value coming from the sensor

void setup() {
  Serial.begin(9600);
}

void loop() {
  // read the value from the sensor:
  sensorValue = analogRead(sensorPin);
  // turn the ledPin on
  //digitalWrite(ledPin, HIGH);
  // stop the program for <sensorValue> milliseconds:
  //delay(sensorValue);
  // turn the ledPin off:
  int outvalue = map(sensorValue, 0,1023,0,255);
  Serial.println(outvalue);
  analogWrite(ledPin, outvalue);
  // stop the program for for <sensorValue> milliseconds:
  //delay(sensorValue);
}

8.8 Servo Motor, Reading Analog, Writing Digital, Using Map and the Servo Library: knob2.ino

#include <Servo.h>

Servo myservo; // create servo object to control a servo

int potpin = 0; // analog pin used to connect the potentiometer
int val;       // variable to read the value from the analog pin

void setup()
Figure 3: Arduino servo motor example. By adjusting the potentiometer, analog pin 0 is set to a voltage between 0 and 5 volts. This set an angle on the servo motor. The motor used is a Futaba.
{  
    myservo.attach(9); // attaches the servo on pin 9 to the servo object  
}

void loop()  
{
    val = analogRead(potpin); // reads the value of the potentiometer (value between 0 and 1023)
    val = map(val, 0, 1023, 0, 179); // scale it to use it with the servo (value between 0 and 180)
    myservo.write(val); // sets the servo position according to the scaled value
    delay(15); // waits for the servo to get there
}

9  Stepper Motors

Stepper motors have a set of toothed stator poles around the circumference. The rotor also has similar teeth, but with a slightly smaller pitch. The teeth of each stator pole is offset by a small angle. When the stator winding is energised the teeth of the stator pole and the rotor try to line up to minimize the magnetic energy, which produces a force and resulting motor torque. When this winding is deenergised and a neighboring winding energized, the rotor rotates a small angle to line up the teeth with the newly energized pole.

10  Types of Stepper Motors

Unipolar, Bipolar.

10.1  Unipolar Motors

A unipolar stepper motor has one winding with center tap per phase. Each section of windings is switched on for each direction of magnetic field. Since in this arrangement a magnetic pole can be reversed without switching the direction of current, the commutation circuit can be made very simple (e.g. a single transistor) for each winding. Typically, given a phase, the center tap of each winding is made common: giving three leads per phase and six leads for a typical two phase motor. Often, these two phase commons are internally joined, so the motor has only five leads.

A microcontroller or stepper motor controller can be used to activate the drive transistors in the right order, and this ease of operation makes unipolar
motors popular with hobbyists; they are probably the cheapest way to get precise angular movements. Unipolar stepper motor coils

(For the experimenter, the windings can be identified by touching the terminal wires together in PM motors. If the terminals of a coil are connected, the shaft becomes harder to turn. One way to distinguish the center tap (common wire) from a coil-end wire is by measuring the resistance. Resistance between common wire and coil-end wire is always half of what it is between coil-end and coil-end wires. This is because there is twice the length of coil between the ends and only half from center (common wire) to the end.) A quick way to determine if the stepper motor is working is to short circuit every two pairs and try turning the shaft, whenever a higher than normal resistance is felt, it indicates that the circuit to the particular winding is closed and that the phase is working.

10.2 Bipolar Motors

Bipolar motors have a single winding per phase. The current in a winding needs to be reversed in order to reverse a magnetic pole, so the driving circuit must be more complicated, typically with an H-bridge arrangement (however there are several off the shelf driver chips available to make this a simple affair). There are two leads per phase, none are common.

Static friction effects using an H-bridge have been observed with certain drive topologies.[2]

Dithering the stepper signal at a higher frequency than the motor can respond to will reduce this "static friction" effect.

Because windings are better utilized, they are more powerful than a unipolar motor of the same weight. This is due to the physical space occupied by the windings. A unipolar motor has twice the amount of wire in the same space, but only half used at any point in time, hence is 50%

An 8-lead stepper is wound like a unipolar stepper, but the leads are not joined to common internally to the motor. This kind of motor can be wired in several configurations:

Unipolar.

Bipolar with series windings. This gives higher inductance but lower current per winding.

Bipolar with parallel windings. This requires higher current but can perform better as the winding inductance is reduced.
Bipolar with a single winding per phase. This method will run the motor on only half the available windings, which will reduce the available low speed torque but require less current.

11 Jason Babcock on Reverse Engineering the Stepper Wire Pinouts

Make a table with wire colors along the top and left edge. Check resistance of the wire pairs and put in table.


12 Stepper Motor Control, LadyAda Motor Schield

http://www.ladyada.net/make/mshield/use.html

13 Lady Ada Motor Control Schield Program

```c
#include <AFMotor.h>
AF_Stepper motor(48, 2);
void setup() {
    Serial.begin(9600);  // set up Serial library at 9600 bps
    Serial.println("Stepper test!");
    motor.setSpeed(10);   // 10 rpm
    motor.step(100, FORWARD, SINGLE);
    motor.release();
    delay(1000);
}
void loop() {
    motor.step(100, FORWARD, SINGLE);
    motor.step(100, BACKWARD, SINGLE);
    motor.step(100, FORWARD, DOUBLE);
    motor.step(100, BACKWARD, DOUBLE);
```

20
motor.step(100, FORWARD, INTERLEAVE);
motor.step(100, BACKWARD, INTERLEAVE);
motor.step(100, FORWARD, MICROSTEP);
motor.step(100, BACKWARD, MICROSTEP);
}

If you want two stepper motors to step at once you’ll need to write something like this:

void doublestep (int steps, int direction, int style) {
    while (steps--) {
        motor1.step(1, direction, style);
        motor2.step(1, direction, style);
    }
}

Arduino motor/stepper/servo control - How to use

http://www.ladyada.net/make/mshield/use.html

14 The Microcontroller Used in the Arduino is the Atmel, ATA128

The high-performance, low-power Atmel 8-bit AVR RISC-based microcontroller combines 128KB of programmable flash memory, 4KB SRAM, a 4KB EEPROM, an 8-channel 10-bit A/D converter, and a JTAG interface for on-chip debugging. The device supports throughput of 16 MIPS at 16 MHz and operates between 4.5-5.5 volts.

By executing instructions in a single clock cycle, the device achieves throughputs approaching 1 MIPS per MHz, balancing power consumption and processing speed.
Flash (Kbytes): 128 Kbytes
Pin Count: 64
Max. Operating Frequency: 16 MHz
CPU: 8-bit AVR
number of Touch Channels: 16
Hardware QTouch Acquisition: No
Max I/O Pins: 53
15 Servos, (Servomechanisms)

Hobby servos rotate to an angle defined by the pulse width of a PWM signal. So they rotate up to 180 degrees. If the pulse width is say 50 percent, then the rotation angle is 90 degrees. This is done in the servo by comparing the voltage induced by the pulse width to a voltage of a potentiometer inside of the servo, which is connected to the shaft. So it is a closed loop control, adjusting until this condition is met.

16 The Pololu Stepper Motor Driver


http://www.pololu.com/catalog/product/1201

Overview

This product is a carrier board or breakout board for Allegros A4983 DMOS Microstepping Driver with Translator; we therefore recommend careful reading of the A4983 datasheet (368k pdf) before using this product. This stepper motor driver lets you control one bipolar stepper motor at up to 2 A output current per coil (see the Power Dissipation Considerations section below for more information). Here are some of the drivers key features:

1) Simple step and direction control interface
2) Five different step resolutions: full-step, half-step, quarter-step, eighth-step, and sixteenth-step
3) Adjustable current control lets you set the maximum current output with a potentiometer, which lets you use voltages above your stepper motors rated voltage to achieve higher step rates
4) Intelligent chopping control that automatically selects the correct current decay mode (fast decay or slow decay)
5) Over-temperature thermal shutdown, under-voltage lockout, and crossover-current protection

Like nearly all our other carrier boards, this product ships with all surface-mount components including the A4983 driver IC installed as shown in the product picture.

We also sell a larger version of the A4983 carrier that has reverse power protection on the main power input and built-in 5 V and 3.3 V voltage
regulators that eliminate the need for separate logic and motor supplies.

Included hardware

The A4983 stepper motor driver carrier comes with one 116-pin break-away 0.1” male header. The headers can be soldered in for use with solderless breadboards or 0.1” female connectors. You can also solder your motor leads and other connections directly to the board.

This board has been replaced by the newer A4988 stepper motor driver carrier, which is a drop-in replacement with additional integrated protection.

17 Texas Instruments Chip for Driving Stepper Motors

Terry Fredrich

SN754410


Local file: c:\je\pdf\sn754410.pdf

18 Futaba S3003 Servo

Speed .23 sec/60 deg (at 4.8V)
Speed .19 sec/60 deg (at 6.0V)
Torque 44.4 oz-in (at 4.8V)
Torque 56.9 oz-in (at 6.0V)
Wires white - control, red +5 Volts, and black -5 Volts.

19 Hitec Servo HS-311

This servo was used in the April 2011 classes given by Josh Bookout and Jeston Stoffel and was purchased at Hobby Haven.
20 A Stepper Motor From an HP Scanner Disassembly

The stepper motor has number STH-39DH26-02, and its properties according to the label on the motor are:
- Bipolar, 1.8 deg /step, 4.1Ω, see page 63 notebook April 2012, for table.
- By making a table of wire colors and measuring the resistance between pairs we determined that the wires for the two windings are:
  Pair 1, brown and pink, pair 2 yellow and red.

21 Wire Size for Arduino Pins

The wire size that fits the Arduino pins is 22 gauge = .644 mm diameter. The 24 gauge wire with diameter .511 mm is too small.

22 Driving an FET

See page 71 of *Getting Started With Arduino* and document

arduinomosfetdriver.pdf

Some FETs:
- IRF510, IRF520, IRFP250. The IRF510 can not handle much current, the IRFP250 can handle a large current. The latter is rated as a maximum of 32 Amps.

  An n-channel MOSFET is somewhat similar to a vacuum tube triode with the identification of the gate with the vacuum tube grid, the source with the cathode, and the drain with the plate. So if the gate is made positive, the FET turns on like a switch, and current flows from the positive drain to the negative source. So the gate has high impedance and may be driven from a digital pin of the arduino, where one might select a pair of resistors to act as a potentiometer to set the positive voltage on the gate between 0 and 5 volts.

  The plot of current vs voltage for a given gate voltage is more or less linear. The higher the gate voltage the steeper the slope. The slope is $I/V = 1/R$ or the reciprocal of an effective gate to source resistance $R$ in the MOSFET. So As the drain to source voltage increases saturation is reached
where the current becomes constant, so the slope decreases to zero as the drain to source voltage goes to infinity. So the resistance $R$ goes to infinity. Since the power lost as heat is $I^2R$ heating of the FET goes to infinity and the FET will be destroyed. Operating even at moderate voltages requires a heat sink.

For a breadbord test circuit using the IRF510, see pages 64-65 of the april-may notes of 2012. Powers a 12 volt lightbulb using a 9 vold dc supply brick and also the arduino notebook for the class.

23 Driving a Relay

If a relay is driven directly, from a digital pin, the relay coil should draw less than 40 mA at 5 volts, because this is the maximum current that an Arduino can supply.

24 The Secrets of the Arduino PWM

The ATmega168P/328P chip runs at 16MHz. The chip supplies several PWM signals. The standard frequency is

$$\nu = \frac{16 \times 10^6}{(16)(256)} = 976.5625Hz.$$  

This can be changed in several ways. Reference:

C:\je\pdf\SecretsOfArduinoPWM.pdf


25 Writing Data From the Arduino Board to a File on Your Computer

This can be done by establishing serial communication to the Arduino with a terminal program, or reading serial data with a program you write. See the Arduino Forum, or my local file
26 Fritzing Design Application

Fritzing is an open source software initiative to support designers and artists ready to move from physical prototyping to actual product. It was developed at the University of Applied Sciences of Potsdam.

The software is created in the spirit of Processing and Arduino and allows a designer, artist, researcher, or hobbyist to document their Arduino-based prototype and create a PCB layout for manufacturing. The complementary website helps to share and discuss drafts and experiences as well as to reduce manufacturing costs. In other words, they make electronic items from your design.

Fritzing can be seen as an Electronic design automation (EDA) tool for non-engineers: the input metaphor is inspired by the environment of designers (the breadboard-based prototype), the output is offering nearly no options and is focused on accessible means of production.

27 Processing Programming Language

"Processing is an open source programming language and integrated development environment (IDE) built for the electronic arts and visual design communities with the purpose of teaching the basics of computer programming in a visual context, and to serve as the foundation for electronic sketchbooks. The project was initiated in 2001 by Casey Reas and Benjamin Fry, both formerly of the Aesthetics and Computation Group at the MIT Media Lab. One of the stated aims of Processing is to act as a tool to get non-programmers started with programming, through the instant gratification of visual feedback. The language builds on the Java language, but uses a simplified syntax and graphics programming model."

For more information, see the complete Wikipedia Article, and the book by Josua Noble called Programming Interactivity.

28 List of Examples from Arduino Site

Examples
See the foundations page for in-depth description of core concepts of the Arduino hardware and software; the hacking page for information on extending and modifying the Arduino hardware and software; and the links page for other documentation.

Note: these examples are written for Arduino 1.0 and later. Certain functions may not work in earlier versions. For best results, download the latest version.

Here’s a style guide that helps with writing examples for beginners.

**Core Functions**

Simple programs that demonstrate basic Arduino commands. These are included with the Arduino environment; to open them, click the Open button on the toolbar and look in the examples folder.

1. **Basics**
   - **BareMinimum**: The bare minimum of code needed to start an Arduino sketch.
   - **Blink**: Turn an LED on and off.
   - **DigitalReadSerial**: Read a switch, print the state out to the Arduino Serial Monitor.
   - **AnalogReadSerial**: Read a potentiometer, print it’s state out to the Arduino Serial Monitor.
   - **Fade**: Demonstrates the use of analog output to fade an LED.

2. **Digital**
   - **Blink Without Delay**: Blinking an LED without using the delay() function.
   - **Button**: Use a pushbutton to control an LED.
   - **Debounce**: Read a pushbutton, filtering noise.
   - **Button State Change**: Counting the number of button pushes.
   - **Tone**: Play a melody with a Piezo speaker.
   - **Pitch follower**: Play a pitch on a piezo speaker depending on an analog input.
   - **Simple Keyboard**: A three-key musical keyboard using force sensors and a piezo speaker.
   - **Tone4**: Play tones on multiple speakers sequentially using the tone() command.

3. **Analog**
AnalogInOutSerial: read an analog input pin, map the result, and then use that data to dim or brighten an LED.

Analog Input: use a potentiometer to control the blinking of an LED.

AnalogWriteMega: fade 12 LEDs on and off, one by one, using an Arduino Mega board.

Calibration: define a maximum and minimum for expected analog sensor values.

Fading: use an analog output (PWM pin) to fade an LED.

Smoothing: smooth multiple readings of an analog input.

4. Communication

These examples include code that allows the Arduino to talk to Processing sketches running on the computer. For more information or to download Processing, see processing.org. There are also Max//MSP patches that can communicate with each Arduino sketch as well. For more on Max//MSP see Cycling 74. For Pd patches that can communicate with these sketches, see Scott Fitzgerald’s examples.

ASCII Table: demonstrates Arduino’s advanced serial output functions.

Dimmer: move the mouse to change the brightness of an LED.

Graph: send data to the computer and graph it in Processing.

Physical Pixel: turn a LED on and off by sending data to your Arduino from Processing or Max/MSP.

Virtual Color Mixer: send multiple variables from Arduino to your computer and read them in Processing or Max/MSP.

Serial Call Response: send multiple variables using a call-and-response (handshaking) method.

Serial Call Response ASCII: send multiple variables using a call-and-response (handshaking) method, and ASCII-encode the values before sending.

SerialEvent: Demonstrates the use of SerialEvent().

Serial input (Switch (case) Statement): how to take different actions based on characters received by the serial port.

MIDI: send MIDI note messages serially.

MultiSerialMega: use two of the serial ports available on the Arduino Mega.

5. Control Structures

If Statement (Conditional): how to use an if statement to change output conditions based on changing input conditions.

For Loop: controlling multiple LEDs with a for loop and.
Array: a variation on the For Loop example that demonstrates how to use an array.
While Loop: how to use a while loop to calibrate a sensor while a button is being read.
Switch Case: how to choose between a discrete number of values. Equivalent to multiple If statements. This example shows how to divide a sensor’s range into a set of four bands and to take four different actions depending on which band the result is in.
Switch Case 2: a second switch-case example, showing how to take different actions based in characters received in the serial port.

6. Sensors

7. Display
Examples of basic display control
LED Bar Graph: how to make an LED bar graph.
Row Column Scanning: how to control an 8x8 matrix of LEDs.

8. Strings
StringAdditionOperator: add strings together in a variety of ways.
StringAppendOperator: append data to strings.
StringCaseChanges: change the case of a string.
StringCharacters: get/set the value of a specific character in a string.
StringComparisonOperators: compare strings alphabetically.
StringConstructors: how to initialize string objects.
StringIndexOf: look for the first/last instance of a character in a string.
StringLength and StringLengthTrim: get and trim the length of a string.
StringReplace: replace individual characters in a string.
StringStartsWithEndsWith: check which characters/substrings a given string starts or ends with.
StringSubstring: look for ”phrases” within a given string.

29 Bibliography


