Arduino Hands-On 2
CS5968 / ART4455

Disclaimer

- Many of these slides are mine
- But, some are stolen from various places on the web
  - todbot.com – Bionic Arduino and Spooky Arduino class notes from Tod E. Kurt
  - ladyada.net – Arduino tutorials by Limor Fried

Getting Input (Digital)

- Switches make or break a connection
- But Arduino wants to see a voltage
  - Specifically, a “HIGH” (5 volts)
  - or a “LOW” (0 volts)

*How do you go from make/break to high/low?*

Switches

- Digital inputs can “float” between 0 and 5 volts
- Resistor “pulls down” input to ground (0 volts)
- Pressing switch sets input to 5 volts “pull-down”
- Press is HIGH Release is LOW

*Why do we need the “pull down” resistor?*

Another Switch

- Resistor pulls up input to 5 volts
- Switch sets input to 0 volts
- But now the sense is inverted
  - Press is LOW
  - Release is HIGH “pull-up”

A Switch

- Pressing the button, “closes the gap”
- always connected together
Using a Switch

digitalRead(pin);

// constants won't change. They're used here to set pin numbers:
const int buttonPin = 2;     // the number of the pushbutton pin
const int ledPin = 13;      // the number of the LED pin

// variables hold values that will change:
int buttonState = 0;         // variable for reading the pushbutton status

void setup() {
  pinMode(ledPin, OUTPUT);  // initialize the LED pin as an output:
pinMode(buttonPin, INPUT); // initialize the pushbutton pin as an input:
}

void loop() {
  buttonState = digitalRead(buttonPin); // read the state of the pushbutton

  if (buttonState === HIGH) { // buttonState HIGH means pressed
    digitalWrite(ledPin, HIGH); // turn LED on:
  } else { digitalWrite(ledPin, LOW);} // turn LED off:
}

Using digitalRead()

- In setup(): use pinMode(myPin, INPUT) to make pin an input
- In loop(): use digitalRead(myPin) to get switch position
  - If doing many tests, use a variable to hold the output value of digitalRead().
  - e.g. val = digitalRead(myPin)

Moving on…

- Write a program that reads the value on an input pin
  - Use the button to change from blinking fast to blinking slow

Multiple Switches

Same sub-circuit, just duplicate
Make Your Own Switches

- Anything that makes a connection
- Wires, tin foil, tinfoil balls, ball bearings
- Pennies!
- Nails, bolts, screws

- Or repurpose these tiny switches as bump detectors or closure detectors

Analog Input

To computers, analog is chunky

Analog Read(pin); reads an analog pin returns a digital value between 0-1023 analog pins need no pinMode declaration

Analog Input on Arduino

- Our version uses ATmega328p
- six ADC inputs (Analog to Digital Converter)
- Voltage range is 0-5v
- Resolution is 10 bits (digital values between 0-1023)
- In other words, 5/1024 = 4.8mV is the smallest voltage change you can measure

Analog Input

- Many states, not just two (HIGH/LOW)
- Number of states (or “bins”) is resolution
- Common computer resolutions:
  - 8-bit = 256 states
  - 16-bit = 65,536 states
  - 32-bit = 4,294,967,296 states

Analog Input

Sure sure, but how to make a varying voltage? With a potentiometer. Or just pot.

+5V

measure

10k potentiometer

gnd

measure here

+5V—
gnd—
int sensorPin = 0; // 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() {
  pinMode(ledPin, OUTPUT); // declare the ledPin as an OUTPUT. Note that you don’t need to declare the Analog pin – it’s always input
}

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

Moving on...

- Write a program to read an analog value from a pot and use that value to control the brightness of an LED
- Fade the LED by turning the pot
- Useful function is map(value, fromlow, fromhigh, tolow, tohigh);
  \[ y = \text{map}(x, 0, 1023, 50, 150); \]
- Also remember analogWrite(pin, value);
- PWM value from 0-255

What good are pots?

- Anytime you need a ranged input:
  - (we’re used to knobs)
- Measure rotational position
  - steering wheel, etc.
- But more importantly for us, potentiometers are a good example of a resistive sensor
Sensing the Dark

- Pots are example of a voltage divider
- Voltage divider splits a voltage in two
- Same as two resistors, but you can vary them

Sensing the Dark: Photocells

- aka. photoresistor, light-dependent resistor
- A variable resistor
- Brighter light == lower resistance
- Photocells you have range approx. 0-10k

Photocell Circuit

Wave your hand over it = blink faster
Point it towards the light = blink slower

Moving on…

- Connect a photocell instead of a pot to your fading circuit
- Do you get the same range of fade as with the pot?
- Why or why not?

Resistive sensors

thermistor (temperature)
flex sensor (bend, deflection)
force sensors (pressure)
also air pressure and others
Serial from Arduino to PC

- **Serial.begin(baud-rate);**
  - Sets serial bit rate

- **Serial.print(arg);**
  - Sends arg to the serial output – can be number or string
    - **Serial.print(arg,format);** formats the arg
      - format can be BYTE, BIN, OCT, DEC, HEX

- **Serial.println(arg);**
  - Same, but also prints a new line to the output

Send data to PC

```cpp
void setup() {
  Serial.begin(9600); // init the serial port
}
void loop() {
  Serial.println("Hello World!"); // print to the screen!
  delay(500); // Wait so you don’t print too fast
}
```
Checking on Analog Inputs

```c
int sensorPin = 0;  // 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() {
  pinMode(ledPin, OUTPUT); // declare the ledPin as an OUTPUT:
  Serial.begin(9600);    // Init serial communication at 9600 baud
}

void loop() {
  sensorValue = analogRead(sensorPin); // read the value from the sensor:
  Serial.println(sensorValue, DEC);    // print the value you got
  delay(500);                          // wait so you don’t print too much!
}
```

Serial From PC to Arduino

- `Serial.available();` returns an int that tells you how many bytes remain in the input buffer
- `Serial.read();` returns the next byte waiting in the input buffer
- `Serial.flush();` clear the input buffer of any remaining bytes

Serial Read Example

```c
int incomingByte = 0; // for incoming serial data

void setup() {
  Serial.begin(9600); // opens serial port, sets data rate to 9600 bps
}

void loop() { // send data only when you receive data:
  if (Serial.available() > 0) { // read the incoming byte:
    incomingByte = Serial.read();
    // say what you got:
    Serial.print("I received: ");
    Serial.println(incomingByte, DEC);
  }
}
```

Arduino Says “Hi”

“SerialHelloWorld”

Sends “Hello world!” to your computer

Click on “Serial Monitor” button to see output

Watch TX LED compared to pin 13 LED

Telling Arduino What To Do

“SerialReadBasic”

You type “H”, LED blinks

In “Serial Monitor”, type “H”, press Send

`Serial.available()` tells you if data present to read

Arduino Communications

- Pst, Arduino doesn’t really do USB
- It really is “serial”, like old RS-232 serial
- All microcontrollers can do serial
- Not many can do USB
- Serial is easy, USB is hard
Serial Communications

- “Serial” because data is broken down into bits, each sent one after the other down a single wire.
- The single ASCII character ‘B’ is sent as:
  \[ 'B' = 01000010 \]
  \[ = L H L L L L H L \]
- Toggle a pin to send data, just like blinking an LED
- You could implement sending serial data with `digitalWrite()` and `delay()`
- A single data wire needed to send data. One other to receive.

Arduino Mini

Arduino Mini separates the two circuits

Arduino & USB

- Since Arduino is all about serial
- And not USB,
- Interfacing to things like USB flash drives, USB hard disks, USB webcams, etc. is not possible

Arduino & USB-to-serial

Arduino board is really two circuits

Arduino to Computer

USB is totally optional for Arduino
But it makes things easier

Controlling the Computer

- Can send sensor data from Arduino to computer with `Serial.print()`
- There are many different variations to suite your needs:

```cpp
int val = 123;
Serial.print(val); // sends 3 ASCII chars "123"
Serial.print(val,'B'); // same as above
Serial.print(val,'0'); // sends 2 ASCII chars "7B"
Serial.print(val,'B'); // sends 8 ASCII chars "01110011"
Serial.print(val,'h'); // sends 8 bytes, the hexadec value
```
Controlling the Computer

You write one program on Arduino, one on the computer.

In Arduino: read sensor, send data as byte

In Processing: read the byte, do something with it

Controlling Arduino, Again

“SerialReadBlink”

Type a number 1-9 and LED blinks that many times

Converts typed ASCII value into usable number

Most control issues are data conversion issues

Reading Serial Strings

- The function “Serial.available()” makes reading strings easier
- Can use it to read all available serial data from computer
- The “readSerialString()” function at right takes a character string and reads available serial data into it

Moving on... Servos

- Servo motors are small DC motors that have a range of motion of 0-180°
- Internal feedback and gearing to make it work
- Easy three-wire interface
- Position is controlled by PWM signals
**PWM**

Output voltage is averaged from on vs. off time

\[
\text{output\_voltage} = (\text{on\_time} / \text{off\_time}) \times \text{max\_voltage}
\]

- 5 volts
  - 0 volts: 20%, 80%
  - 5 volts: 50%, 50%
- 2.5 Volts
  - 0 volts: 40%, 60%
  - 5 volts: 50%, 50%
- 1.0 Volts
  - 0 volts: 60%, 40%
  - 5 volts: 50%, 50%

**Servomotors**

- Can be positioned from 0-180° (usually)
- Internal feedback circuitry & gearing takes care of the hard stuff
- Easy three-wire PWM 5V interface

**Servos are Awesome**

- DC motor
- High-torque gearing
- Potentiometer to read position
- Feedback circuitry to read pot and control motor
- All built in, you just feed it a PWM signal

**Servos, good for what?**

- Roboticists, movie effects people, and puppeteers use them extensively
- Any time you need controlled, repeatable motion
- Can turn rotation into linear movement with clever mechanical levers

**Servos**

- Come in all sizes
- from super-tiny
- to drive-your-car
- But all have the same 3-wire interface
- Servos are spec’d by:
  - weight: 9g
  - speed: 120°/0.1s @ 6V
  - torque: 20 oz/in @ 6V
  - voltage: 4.8V
  - size: 20x11x28 mm

Our servos are: weight: 9g, speed 0.12s/60°deg at 4.8V, torque (@4.8V) 17.5oz/in (1kgf/cm), voltage range: 3.0 – 7.2V
Servo Example Program

```c
#include <Servo.h>  // include the built-in servo library
Servo myservo;  // create a servo object to control the servo (one per servo)
int pos = 0;   // variable to store the servo position

void setup() {
  myservo.attach(9);  // attach servo control to pin 9
}

void loop() {
  for (pos = 0; pos < 180; pos++) {     // go from 0 to 180 degrees
    myservo.write(pos);  // move the servo
    delay(15);   // give it time to get there }
  for (pos = 180; pos>=1; pos--) {    // wave backwards
    myservo.write(pos );
    delay(15);
  }
}
```

Servo Functions

- **Servo is a class**
  - `Servo myservo`, // creates an instance of that class
- `myservo.attach(pin);`
  - attach to an output pin (doesn’t need to be PWM pin!)
  - Servo library can control up to 12 servos on our boards
  - but a side effect is that it disables the PWM on pins 9 and 10
- `myservo.write(pos);`
  - moves servo – pos ranges from 0-180
- `myservo.read();`
  - returns the current position of the servo (0-180)
Moving on…

- Write a program to control the position of the servo from a pot, or from a photocell
- remember pot `analogRead();` values are from 0-1023
- measure the range of values coming out of the photocell first?
- use `Serial.print(val);` for example
- use `map(val, in1, in2, 0, 180);` to map in1-in2 values to 0-180
- Can also use `constrain(val, 0, 180);`

Side Note - Power

- Servos can consume a bit of power
- We need to make sure that we don’t draw so much power out of the Arduino that it fizzes
- If you drive more than a couple servos, you probably should put the servo power pins on a separate power supply from the Arduino
- Use a wall-wart 5v DC supply, for example

Robo Cat Toy Idea

Tape on a pipe cleaner, and using random behavior similar to the “CandleHeight” sketch, make a randomly moving cat toy

Servo/Light Assignment

- Use a photocell on the input
- put in series with 10k ohm resistor
- use a servo on the output
- connect to a PWM pin
- make the servo do something in response to the amount of light falling on the photocell

Summary – Whew!

- LEDs – use current limiting resistors (remember color code!)
  - drive from `digitalWrite(pin, val);` for on/off
  - drive from `analogWrite(pin,val);` for PWM dimming (values from 0-255)
- buttons – current limiting resistors again
  - active-high or active low (pullup or pulldown)
  - read with `digitalRead(pin);`
- potentiometers (pots)– voltage dividers with a knob
  - use with `analogRead(pin);` for values from 0-1023

Summary – Whew!

- photocells – variable resistors
  - use with current-limiting resistors (to make voltage divider)
- Serial communications – read a byte, or write a value
  - communicate to the Arduino environment, or your own program
- Servos – use Servo library to control motion
  - might need external power supply
  - range of motion 0-180°
- Also `setup()` and `loop()` functions, and various C programming ideas
More Later...

- DC Motors
  - Use transistors as switches for larger current loads

- Stepper motors
  - Sort of like servos, but with continuous range of motion
  - Can also be more powerful

- I2C serial bus
  - Various LED driver chips
  - Other serially-controlled devices

- Piezo buzzers
  - Make some noise!
  - But you can also use them as input devices to sense movement

- IR motion sensors
  - Simple motion and also distance sensors

- Accelerometers
  - Wii nunchucks, for example

- Others?