

# Lab 1: An Introduction to Arduino: Lights, Pots, an Hbridge and Wireless!

During the relatively short and time consuming first few weeks of the class, we will be working to get you familiar with a number of different platforms and tools. Those include using and developing for the Arduino environment (a easy-to-use and low-end platform), Linux device driver writing, PCB design, and working with multiple embedded platforms at once.

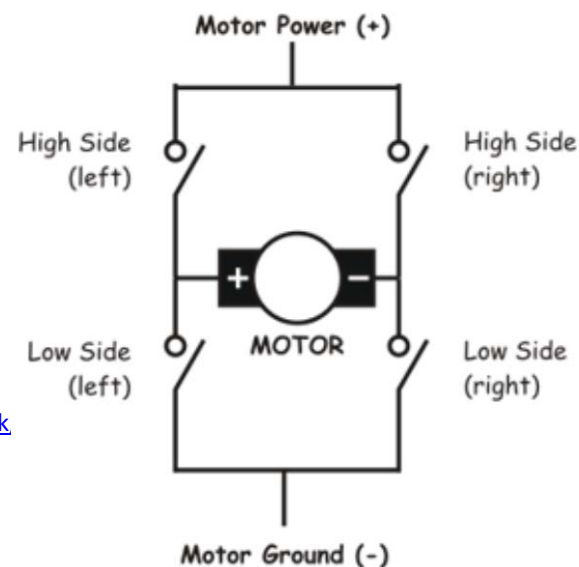
This lab will focus on making a rapid prototype and learning the Arduino environment (which has plenty of good and bad but it is good for rapid prototyping!)

## 1. Prelab

1. Read the lab from start to finish.
2. Read <http://arduino.cc/en/Guide/Environment>
3. Read <http://www.arduino.cc/en/Main/ArduinoBoardDuemilanove>

Answer the following questions:

- Q1.** Who generally uses the Arduino environment? List three projects that use it.
- Q2.** In Arduino-speak, what is a shield? Identify at least two shields, where you can get them from and how much they cost.
- Q3.** What processor family is used by Arduino?
- a. Name another programming environment that is commonly used to program that family.
  - b. What are the pros and cons of the Arduino environment compared to the environment listed above? Be sure to discuss debugging options...
- Q4.** In Arduino-speak, what is a “sketch”?
- Q5.** Look over <http://arduino.cc/en/Tutorial/Foundations>. Much of it is pretty basic.
- a. What does `pinMode()` do and what are the options associated with it?
  - b. How is pin 13 different than most?
- Q6.** Look at <http://arduino.cc/en/Reference/HomePage>
- a. What does `delay()` do?
  - b. What does `analogRead()` do?
- Q7.** What is ZigBee and where is it generally used? What is an XBee?
- Q8.** Consider the figure on the right<sup>1</sup>. Assume if you have the + side of the



<sup>1</sup> Figure taken from <http://www.mcmanis.com/chuck>

motor connected to “Motor Power” and the negative side to “Motor Ground” the motor will move in the clockwise direction. Which switches should be open and which should be closed to get the motor to move clockwise? To move counter-clockwise?

- Q9.** Look at the H-bridge model # SN754410 pin out (find it on-line).
- Explain how you would use the SN754410 to control one motor.
  - Explain how the figure above corresponds to the SN754410.

## 2. Inlab

---

The purpose of this lab is to create a basic familiarity with the Arduino environment. You will start by writing the “hello world” of embedded systems: getting a light blinking. We’ll then move quickly to reading analog input, getting wireless communication working, and finally controlling an H-bridge.

### Part 1—Das Blinkenlights<sup>2</sup>

---

Let’s get familiar with the Arduino programming environment. Open Arduino by double clicking the “infinity symbol.” This should bring you to a blank “sketch”. We are going to make a simple little sketch for blinking the LED connected to pin 13. Note that we won’t need any includes or defines for this part; you only need to define a setup and loop function:

```
void setup() {
    // Called once to set up stuff before entering the loop
}

void loop() {
    // Called in a while(1) loop indefinitely
}
```

This is because the Arduino IDE transforms your sketch into a valid C++ program by adding the following lines of code:


```
#include "WProgram.h" // Definitions relevant to Arduino

void main() {
    setup();
    while(1) {
        loop();
    }
}
```

In `setup`, use `pinMode` (which you read about in the prelab) to make pin 13 an output. In `loop()`, use `digitalWrite()` and `delay()` to cause the LED to blink at 1 Hz with a 50% duty cycle. Note that both `setup()` and `loop()` should be declared as “void”.

---

<sup>2</sup> See <http://en.wikipedia.org/wiki/Blinkenlights> if you are curious why this section is so named.

Now that the code is written, check to see if it compiles. Press the play icon named “Verify.” This will check for errors and compile your code. After the program compiles successfully, we still need to upload the program to the board. Connect the Arduino via the USB port. Select the correct board (should be fairly obvious) and port (might require you to go to the control panel and look at the device drivers. Then click the second icon on the upload icon.  If you get an error, check to be sure you’ve selected the correct device and port. You’ve written your first Arduino program!

**G1.** Have your GSI check that your code is flashing the LED at 1Hz.

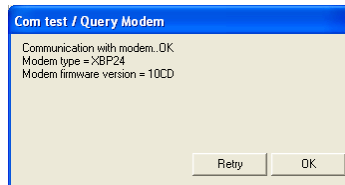
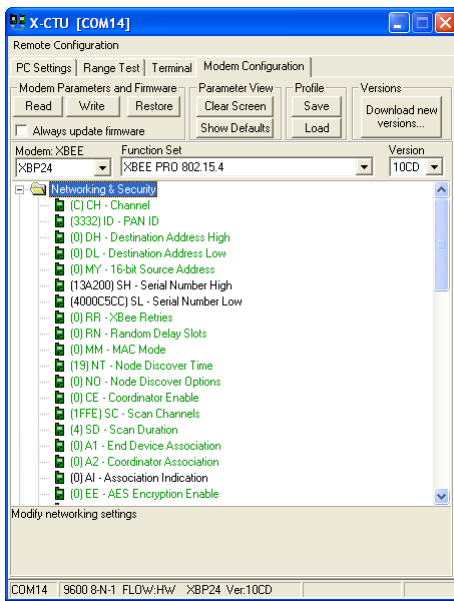
Using a breadboard, connect four digital outputs to four LEDs so you can light each of them individually. Include a resistor in series with each LED (about 1000 Ohm) so we don’t fry the board or the LED. Also make sure to share the Arduino’s ground with the LED’s. To recap, from a digital pin you will connect a resistor, the resistor will connect to an LED, and the LED will be connected to ground on the Arduino board. Write a quick test sketch to be sure you can individually control each LED. One interesting thing about the Arduino environment is that you *do* have access to the processor and its low-level interfaces just as if you were coding in C (because you are). Check out <http://www.arduino.cc/en/Reference/PortManipulation> and use the DDRx and PORTx to change the LEDs all at once.

For the last step of this section, hook up a potentiometer (pot) which we’ve supplied such that the middle pin of the pot is connected to one of the analog inputs found on the board while the other two pins on the pot are connected to 0V and 5V. Look through the Arduino documentation about how to work with analog inputs and make it so the number of LEDs lit is about proportional to how far the pot is turned (or, if you prefer, the MSBs of the value are displayed on the LEDs).

**G2.** Have your GSI check that your LEDs are properly controlled by the pot. Also show your sketch that uses DDRx and PORTx to control the LEDs.

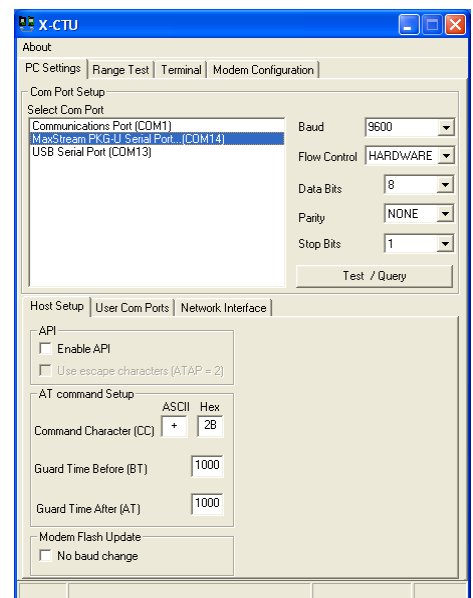
## Part 2—Xbee Wireless

Xbee is the component we will be using to communicate wirelessly to the Arduino board/microprocessor. Connect the serial modem and start the X-CTU program, which will let you speak to the chip and reprogram it appropriately. The exact COM port will vary a bit based on which modem is used and a few other things. But you



should be able to press the “Test/Query” button and get the modem type and firmware number.

Click the “modem configuration” tab. Click on the read button and you should get



the modem's current configuration, which should be pretty similar to what you see in the window on the left. Press the restore button and then the write button. This will set the XBee back to its default settings. Look and the channel ID and set it to 12+your station number in hex (so station 1 would be 13 or "D" in hex, station 10 would be 22 or "16" in hex. Skipping this step might result in your XBee communicating with another group's Xbee. (We are leaving "C" open because it's the default channel and the most likely for people to be accidentally on). Remember to change the channel of your other XBee too.

Wire up a separate XBee chip to the serial pins (0 and 1) of your Arduino board. Take a look at the XBee datasheet (<http://www.sparkfun.com/datasheets/Wireless/Zigbee/XBee-Datasheet.pdf>) and decide which pins should be connected to the Arduino board. ***With no power*** supplied, connect your XBee to the Arduino board. Once you think you have things wired correctly, ***but before you supply power***, have the GSI check your wiring. *Notice that the XBee doesn't plug in nicely to the breadboard so you're going to need to find a way to make the needed connections (just directly with wires will work).* Pay attention to the Vcc value expected by the XBee.

**G3.** Have your GSI check your connections.

One fun issue is that you can't program the board while the XBee is plugged in. You will need to unplug the XBee's serial connections (but not power and ground) when you program the board. Be sure you put them back in the right way each time.

**Q1.** Why do you suppose the XBee needs to be unplugged each time?

Now you have to make a sketch that will use the serial functions in order to use the Xbee. Have your first sketch simply echo anything it is sent back but with an "X" and a newline after each character. So if you send "AB" you should get back "AX" and then "BX". Send data via the X-CTU program's terminal from the XBee modem to your XBee/Arduino setup wirelessly. The XBee/Arduino should generate the response and sent it back over the wireless system.

## Part 3—Wireless motor controller

---

Now you are to cut-and-paste the following motor-control program code into a new sketch.

```
#define FORWARD 0
#define BACK 1
#define LEFT 2
#define RIGHT 3
#define STOP 4
const char SoP = 'C'; // \ Ideally these would be high ASCII characters and MUST
const char EoP = 'E'; // / be something that doesn't occur inside of a legal packet!
const int IDLE = 0;
const int SPEED = 150;
const int RW1 = 11;
const int RW2 = 10;
const int LW1 = 9;
const int LW2 = 6;
const int test = 8; // test point.

unsigned char inByte;
int c;
char mode;

void move(int d){

  if(d == FORWARD){
    Serial.println(" *Driving fwd*");
    digitalWrite(test, HIGH);
    analogWrite(LW1, IDLE);
    analogWrite(LW2, SPEED);
    analogWrite(RW1, IDLE);
    analogWrite(RW2, SPEED);
  }
  else if(d == BACK){
    Serial.println(" *Driving back*");
    digitalWrite(test, HIGH);
    analogWrite(LW1, SPEED);
    analogWrite(LW2, IDLE);
    analogWrite(RW1, SPEED);
    analogWrite(RW2, IDLE);
  }
  else if(d == LEFT){
    Serial.println(" *Turning left* ");
    digitalWrite(test, HIGH);
    analogWrite(LW1, IDLE);
    analogWrite(LW2, SPEED);
    analogWrite(RW1, SPEED);
    analogWrite(RW2, IDLE);
  }
  else if(d == RIGHT){
    Serial.println(" *Turning right* ");
    digitalWrite(test, HIGH);
    analogWrite(LW1, SPEED);
    analogWrite(LW2, IDLE);
    analogWrite(RW1, IDLE);
    analogWrite(RW2, SPEED);
  }
  else{
    Serial.println(" *Not driving* ");
    digitalWrite(test, LOW);
    analogWrite(LW1, IDLE);
    analogWrite(LW2, IDLE);
    analogWrite(RW1, IDLE);
    analogWrite(RW2, IDLE);
  }
}

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

```

Serial.println("START");
pinMode(RW1, OUTPUT);
pinMode(RW2, OUTPUT);
pinMode(LW1, OUTPUT);
pinMode(LW2, OUTPUT);
pinMode(test, OUTPUT);

//Initialize outputs immediately
digitalWrite(test, LOW);
analogWrite(LW1, IDLE);
analogWrite(LW2, IDLE);
analogWrite(RW1, IDLE);
analogWrite(RW2, IDLE);
delay(10);
Serial.println("Ready, Steady, Go");
delay(10);
}

void loop(){
  //Waiting for Start of Packet
  delay(10); // We add these to give the Xbee time to catch up, this
             // covers all that are followed by a return.
  while(Serial.available()<1); // wait for character
  inByte = Serial.read();
  if(inByte != SoP){
    //Not expected Start of Packet, bag
    Serial.print("Expected SOP, got: ");
    Serial.write((byte)inByte);
    Serial.print("\n");
    return;
  }
  //We've got a legal SoP, move forward.
  while(Serial.available()<1){}; //wait for next char.
  inByte = Serial.read();
  if(inByte == EoP || inByte == SoP){
    Serial.println("SoP/EoP in length field");
    return;
  }
  //Parse length
  if(inByte>'5' || inByte<'0') // We'll want this larger later!
  {
    Serial.println("Packet Length out of range");
    return;
  }

  int packet_size = inByte - '0';
  char packet[255]; // Way too big for what we are doing right now...

  for(int i=0;(i<packet_size) && mode != EoP;i++){
    while(Serial.available()<1){}; //wait
    inByte = Serial.read();
    if((inByte == EoP || inByte == SoP)){
      Serial.println("SoP/EoP in command field");
      return;
    }
    packet[i] = (char)inByte;
    //Serial.println(inByte,BYTE);
  }
  packet[packet_size]='\0'; // Null terminate String for later.

  //Check that packet ends when expected
  while(Serial.available()<1){};
  inByte = Serial.read();
  if(inByte != EoP){
    Serial.println("EoP not found");
    return;
  }
  // We have a legal packet!
  //PARSE PACKET
  if(packet[0] == '1'){
    //Move command

```

```

        mode=packet[1];
    }
    else if(packet[0] == '2'){
        //Display Read
    }
    else if(packet[0] == '3'){
        //Distance Read
    }
    else if(packet[0] == '4'){
        //Display Write
        Serial.println(packet);
        return;
    }
    else{
        Serial.println("unknown mode");
        return;
    }

    //Serial.println("Ready");
    switch(mode){
        case 'q':
            break;
        case 'B':
            move(BACK);
            break;
        case 'F':
            move(FORWARD);
            break;
        case 'L':
            move(LEFT);
            break;
        case 'R':
            move(RIGHT);
            break;
        default:
            Serial.println("Unknown command in legal packet");
            break;
    }
}

```

Above is a motor control program. Disconnect the XBee serial connections, compile and upload this program, then reconnect the XBee. Go back to the X-CTU program and go to the Terminal Tab. Click the “Assemble Packet” button to bring up a window which allows one to send packets (rather than one character at a time). Look over the code to find a sequence of characters which results in the response “moving forward”.

**Q2.** Examine the code.

- What sequence of characters results in the response “moving forward”? (There is an explanation below that will help, but try to understand the code first.)
- Consider the packet xC21RE, where “x” is any arbitrary character. What will be the possible outputs? Under what circumstance (what value of x) if any will it not be recognized as a legal reverse packet?
- What pins are used to control which motor?

Now go back to the X-CTU program and go to the Terminal Tab. Click the Assemble packet button which will bring up another window. Confirm the operation of the motor control program with the following commands: F->Forward, B->Backward, R->Right, L->Left, and S->Stop. But these commands are structured with the follow syntax. C21xE. C->Command Packet, 2->Two part command, 1->Motor Command, x->One of the above commands, E->End of Packet. Normally the Beginning and End of Packet signifiers are non-ASCII bytes, that mean bytes that don't translate into an ASCII letter, so that you can send arbitrary ASCII characters (think about trying to send a string for example).

- Q4.** Demonstrate the working motor-control code to your GSI. Show what the motor outputs look like on an O-scope.
- Q3.** What does the output of pin 11 look like on the scope when it is at “SPEED”? It’s labeled as an analog output in the code, PWM on the board, explain both labels.

## Part 4—Self-guided H-bridge

In this class there we are often expecting you to be able to figure things out largely on your own. That said, we also want to be sure we don’t destroy our equipment. So we are asking you to wire everything up but not to power your system until the GSI has looked over your setup.

We want you to use an H-bridge model # SN754410 to drive a motor so that your wireless control system determines what direction the motor turns (or if it idles). Hook up the 5V output of the Arduino board as Vcc1 and use the power supply as Vcc2 with 6V output current limited to 200mA. Get a DC motor to turn in different directions depending on the signal sent. Modify the code so that the idle output state can be reached.

- Q5.** Demonstrate your working motor (complete with an idle option) to your GSI.

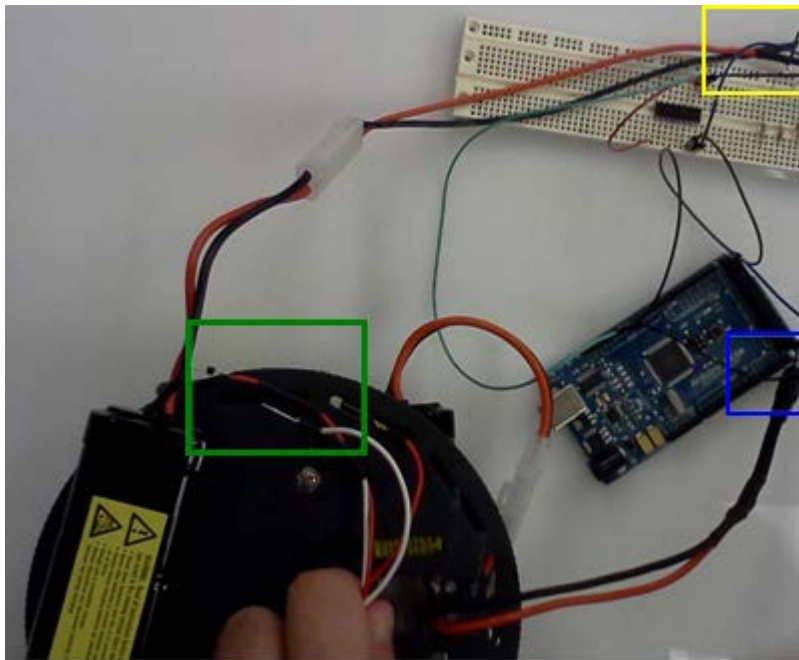


Figure 1: Battery connections. Recall that black is ground and red is power.

Now get the robot working so that you can (poorly) drive it around the lab using a terminal window. *Request:* When getting things tested, please be sure you don’t drive the robot off a table...

- Q6.** Show your GSI you can control the robot using the same motor controller you used in Part 4. You might want to slow down the motors so you have time to type the fairly long commands into the terminal.

## Part 5—Robot building time

Previously, you controlled a motor over a wireless link using an H-bridge with power coming from the lab station’s power supply. We are now going to move the Arduino and its XBee onto a robot. You should have two 7.2v batteries, one of which has a barrel connector for the Arduino, the other of which has two wires. We’ve (rather pedantically) supplied an image. The connector in the blue box powers the Arduino, the connector from the motors in the green box gets connected to outputs of the H-bridge circuit, and the connector in the yellow box is the Vcc2 of the H-



**One request:** when you leave the lab, please start recharging your batteries, especially the ones used to power the robot's motors. In general we'd all like it if the batteries were generally in a charged state for the next folks in the lab. If you don't know how to recharge the batteries, just ask.

### 3. Post-lab

---

This post-lab has a few additional questions that generally require more thought and longer answers than the in-lab questions. Please submit the in-lab and post-lab questions as a single document. These should be done by both group members working together.

- Q4.** Say the sender sends a legal motor control packet for the scheme implemented above. Could a single bit-flip (that is one bit of the message changing from a 0 to a 1 or a 1 to a 0) cause a different legal command (other than idle) to occur? If so, provide an example. If not, explain why not.
- Q5.** Rewrite the code so that all "xC21FE" packets will parse the code correctly, no matter the value of "x". Submit a diff of your code and the original code (using the Linux/Unix "diff" ideally).
- Q6.** What about packets of the form "xyzC21FE"? In general terms, for what values of x, y and z won't this end up with a Forward command being received? *Explain* how you'd fix that issue (or feel free to provide a code diff if you think that's easier).
- Q7.** Let's say you have a packet format where you have a SoP and EoP character, but otherwise all messages are legal (so no internal format, no lengths, just arbitrary bytes in the middle which can't be the SoP or EoP character). Explain an easy-to-implement way to check for errors in transmission. Assume sending an extra byte in the message is acceptable. Your scheme must protect against all single-bit errors, most double-bit errors and even most arbitrary-bit errors.
  - a. What is the probability an arbitrary message sent by someone else with the same SoP and EoP characters will be accepted by your scheme (accepted meaning a bogus random packet was treated as a command).
  - b. Describe the coding overhead associated with your scheme. About how many additional assembly commands do you think will need to be executed?
- Q8.** Do a bit of research and explain the interrupt scheme used by the processor you are using. You may freely use web resources, but please cite your sources clearly. Your answer should include:
  - a. How are different interrupts distinguished?
  - b. How do you disable interrupts?
  - c. How do you clear interrupts?
  - d. Consider the sketch found at <http://gonium.net/md/2006/12/20/handling-external-interrupts-with-arduino/>. Explain what is going on in your own words.