University of Alabama
Department of Physics and Astronomy

PH 126 LeClair Fall 2011

Fun with buttons and conditional statements

1 You will need

- a prototyping board
- an Arduino microcontroller
- 10−20 kΩ resistor
- LED (1, loose)
- various lengths of wire
- switches ('logic' or 'SPDT' on protoboard)

2 Introduction

In today's lab, we will explore three short tutorials which have a common theme: using an **input** to poll the state of an external device and alter an **output** appropriately. For these simple examples, the input will be the status of a switch (open=1, closed=0), while the output will either be text printed to the serial monitor or the status of an LED (lit=1, unlit=0).

You will also be introduced to the idea of a *conditional statement*, a programming structure that basically says "**if** a given condition is true, perform this action; otherwise, do something **else**." Combined with the ability to check the status of an external device, this construct already allows for very powerful feedback and control, once we learn how to properly wield these new weapons.

3 Reading the status of a switch

Let's look a bit at the first tutorial, DigitalReadSerial, just so we may properly introduce a bit of terminology that will show up from now on. On the next page is the circuit you will construct. The "5V" terminal on the Arduino is the power supply, supplying the electrical analogue of pressure known as *voltage* measured in units of *volts*. This constant voltage power source, providing 5 Volts, (5V) is our *signal*, and we wish to control and determine the propagation of this signal, or electrical power.

The "D2" terminal on the Arduino is a digital input, a fancy way of saying that it can detect whether the pressure, or voltage, at that point is high (1) or low (0). Though that is somewhat vague, if the D2 terminal sees the full 5V power signal, it would be considered high, and D2 would report its status as "1" for high. If the D2 terminal sees no power, it reports its status as "0"

for low. To a good approximation, *input terminals draw no power from the signal*, they probe the signal and nothing more.

The GND terminal is an abbreviation for "ground," and this is the point of zero pressure (voltage) for the circuit, a sort of drain for all signals to fall into. In nearly all the circuits we construct, the signals and power sources we use will propagate through some circuit, but ultimately end up at this ground terminal. It is where signals go to die when their function is complete.

The other elements in this circuit are the switch (S2), whose function is relatively clear, and the resistor (R1). A resistor is an element which can reduce the pressure (voltage) in a circuit by dissipating some of the signal's power as heat. In this case, it is present to make sure that the 5V source does not try to dump too much power into the drain (GND), which could cause an overload and release of magic smoke. We will learn more about resistors shortly.

The basic operation of the circuit can be deduced by following the signal from the 5V terminal to its demise at the GND terminal. We assume that the wires are perfect, and no signal is lost. After leaving the 5V terminal, the signal encounters switch S2. If the switch is *closed*, the signal propagates through the resistor R1, losing a bit of power, and finds its way to GND. In this case, since terminal D2 probes the signal after the switch, seeing its full 5V and registering high (1).

If the switch is *open*, the signal has no path to GND or D2. D2 is still connected through GND through the resistor R1. Since neither GND nor R1 provides any signal of its own, D2 measures no signal, or a low status (0). Another way to think of the situation is that D2 is connected to GND, defined to be zero voltage, through the resistor, which can only attenuate (reduce) signals. If there is no signal at GND by definition, there is nothing to attenuate, and D2 must be at zero volts and register low.

The Arduino code looks like this:

```
void setup() {
  Serial.begin(9600);
  pinMode(2, INPUT);
}

void loop() {
  int sensorValue = digitalRead(2);
  Serial.println(sensorValue, DEC);
}
```

The "setup()" construction is a small function in any Arduino program that initializes the hardware in the correct way. The first thing it does initiates serial communications at 9600 baud[i] for talking

[i]Baud = symbols or pulses per second, basically this line sets the desired speed of the serial communication. We'll come back to this.

back to the terminal to display the state of the D2 input. Now comes a more technical detail: the digital pins (such as D2) can be either inputs (probes) or outputs (signal sources). The "pinMode(2,INPUT)" command defines pin 2, the D2 terminal, to be an INPUT or probe connection rather than an output.

The main body of the program, inside the "loop()" construction, does all the heavy lifting. The first line performs a digitalRead operation on pin 2, and stores that value as an integer in the variable sensorValue. Look at the syntax carefully: the **int** prefix defines the variable sensorValue to be an integer, and sets its value equal to the result of the function digitalRead performed on pin 2 (the D2 terminal). The function digitalRead() takes on a single argument within parentheses () which is the desired pin (terminal) to be polled, and the return value or result of this function is the status of desired pin.

In the next line, we see another function, Serial.println. This function uses a hierarchical notation: "Serial" is actually a whole *library* of functions, of which "println" is one of several. The Serial library is for interacting with the serial output to the terminal window, how you see data coming back from the Ardunio. The println function just prints a number. This function takes two arguments: the first is the number to be printed, the second is the format of the number to be printed. In this case, we print the value of the variable sensorValue, the reading from terminal D2, and print it as a DECimal number (as opposed to HEXadecimal, BINary, etc.). Basically: we take our reading from pin D2, and print it to the serial monitor.

The "loop" block of code is just what it sounds like: it runs, and then starts over. It keeps running so long as the Arduino has power, if nothing goes wrong. Thus, the program will measure the status of terminal D2, print the result, and start over again. Once you've wired the circuit and uploaded the code to the Arduino, open up the Serial Monitor. Every time you close the switch, D2 gets the full 5V signal, and registers high, and the monitor prints a 1. Every time you open the switch, D2 loses the signal, registers low, and the monitor prints a zero.

Maybe that doesn't sound like much fun. But you already know how to make the Arduino light up an LED. Instead of just printing a 1 or a 0, the last line of code could just as easily be a command to light an LED. Or, later on, make a motor move, or really anything you like. The main point is that you have just figured out one way to map inputs states to output states, the basis for feedback and control.

4 The tutorials:

`http://arduino.cc/en/Tutorial/DigitalReadSerial`
`http://arduino.cc/en/Tutorial/ButtonStateChange`
`http://arduino.cc/en/Tutorial/IfStatement`

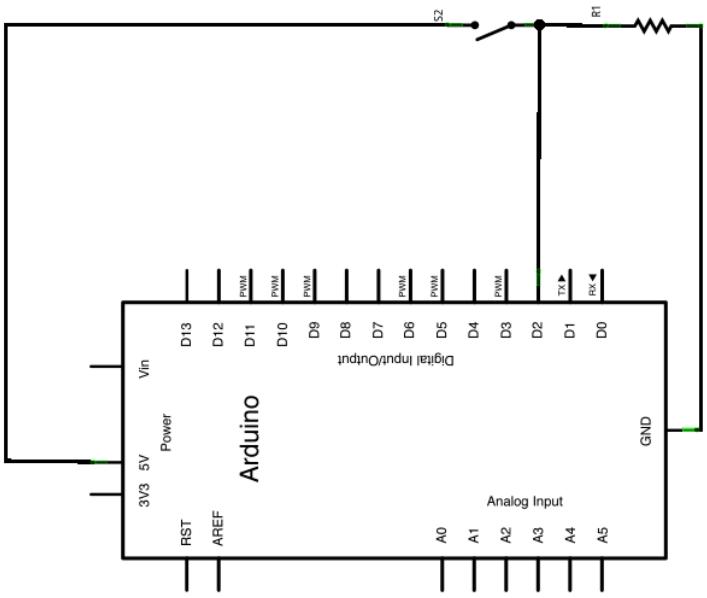

Figure 1: *Polling the state of a switch (S2).*