Module 003: Introduction to the Arduino/RedBoard

Module Outline

In this module you will be introduced to the *microcontroller*. To do this lab, you will need to purchase either an Arduino Uno (or other Arduino board) or a clone like that manufactured by Sparkfun called the RedBoard PLUS the appropriate cable. A "clone" behaves identically to the popular Arduino Uno. All of the inputs and outputs have the same function, the inputs and outputs behave electrically the same and it is programmed in the same way. You may also use another model (like the Arduino Mega, but the figures will differ from that of the Uno).

Arduino Hardware

The figure below shows a visual comparison of the Arduino Uno to the Sparkfun RedBoard. Price is often the reason for choosing a clone over the original, but the Arduinos are typically made of better parts and the microprocessor is in a socket so it can be removed for embedded applications. An in-depth comparison is found at the Sparkfun webpage https://learn.sparkfun.com/tutorials/redboard-vs-uno?_ga=1.150963894.1068174368.1405368730 .

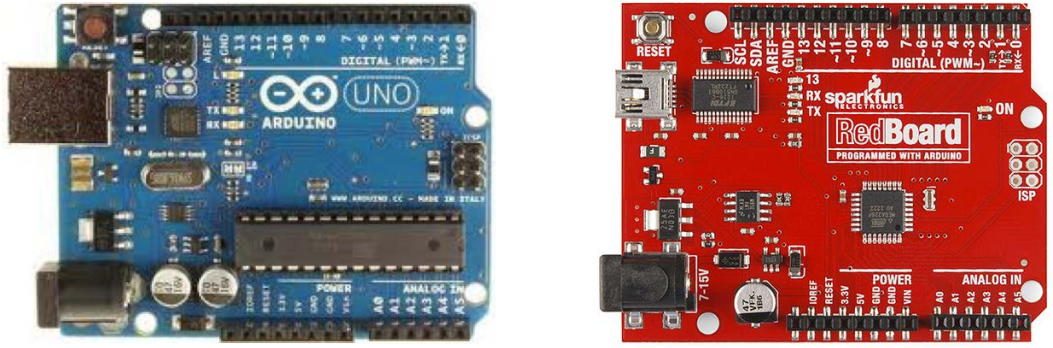
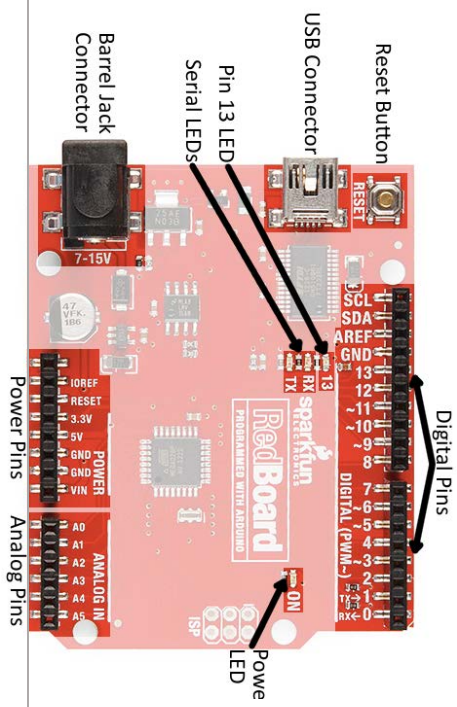


Figure 1: *Arduino Uno compared to Sparkfun's RedBoard*

Notes:

On the right is a figure provided by SparkFun to highlight critical parts of the board.



Hardware Interface Pins to External Circuitry

Digital Pins	Pins that can be configured as inputs or outputs. Digital means that the voltage measured can only have two values 0V or 5V. Some of the pins have PWM functionality described later.
Analog Pins	Pins that input a continuous voltage that the onboard processor digitizes which means that the voltage range 0-5V is mapped onto integer values of 0-1023.
Power Pins	Pins that can be used to power the Arduino board and can source a regulated 3.3V and 5V. Several GND pins are provided.

Useful Board Features

Reset Button	This button interrupts the program running on the board and restarts it from the beginning.
USB Connector	This is a mini USB port that is used to connect the board to a USB port on a computer. This is the interface by which the programs loaded into the microcontroller are downloaded. Power is also provided through this port.
Pin 13 LED	An LED that is always connected to pin 13. If this LED is on then the voltage at pin 13 is 5V if it is off the voltage is 0V. This a useful debugging feature.
Serial LEDs	The serial LEDs indicate the traffic on the receive and transmit pins. The computer and many other devices communicate to the board serially. As you upload programs to the board or communicate to the serial port using coded statements you can see the traffic as these pins flicker when data is being transmitted and received.
Barrel or Jack Connector	This jack is used to connect a battery to provide power to the board so that the board can be disconnected from the computer.
Power LED	Indicates that the board has power.

Let's now look at the board as an *engineer* would while assessing the proper hardware to use in a design project.

Notes:

The Embedded Processor

Most engineering products, even the humble light bulb, now comes with a computer *embedded* in the hardware. From the comfort of the bed, a person can remotely turn on and off your lights. Household appliances like the oven and refrigerator have not required hardware switches on them for decades. Engineers designed and built these products to include ubiquitous digital devices. How?

The design procedure of the analog part of products has a long history and much shared knowledge. Each sub-circuit in the design usually serves one purpose, for example, a voltage regulator, a filter, a mixer. If the design engineer decides that a digital interface or other digital circuitry would benefit the design, then a microprocessor will be embedded in the product. The presence of the microprocessor introduces a new level of complexity and possible enhancements to almost any project. It has many inputs and outputs that can provide *control* based on environmental inputs from sensors. It can make complex "decisions".

As an engineer, how would you begin to assess the role of a microprocessor in a design? How would you address the hardware and software issues associated with your design? It may be intimidating to even begin! For this lab we have made one decision for you by pushing the Arduino-style microprocessor. The Arduino product line (see https://www.arduino.cc/en/Main/Products for a clickable list) has been embraced by the hobby community because of the simple-yet-robust design and availability of support products for programming. This large user base makes it attractive to a first-year course in electronics, but it is not necessarily the best choice for designs you may approach later in your studies and career.

The Arduino Uno board is powered by the ATMega328P microprocessor. The figure below shows how this processor fits into the taxonomy of commonly-used microprocessors. As you enter into your own design projects in the future you would become more familiar with the different types of processors and will learn to appreciate the different designations. Eventually, you would be able to choose the most-appropriate style for yourself. The ATMega328 is produced Atmel Semiconductors. The processor uses an architecture based on a word length of 8-bits, it runs at a clock speed of 16 MHz, the memory is embedded onboard the processor itself, and it follows the more elegant RISC approach. The Reduced Instruction Set (RiSC pronounced "risk") processor uses a simple architecture based on a few well-chosen assembly level instructions. Thereby, RiSC relies on speed and flexibility to gain functionality. Other processors have a very complex set of assembly level instructions, often associated with a complicated architectural structure.

Answer Question 1.

Notes:

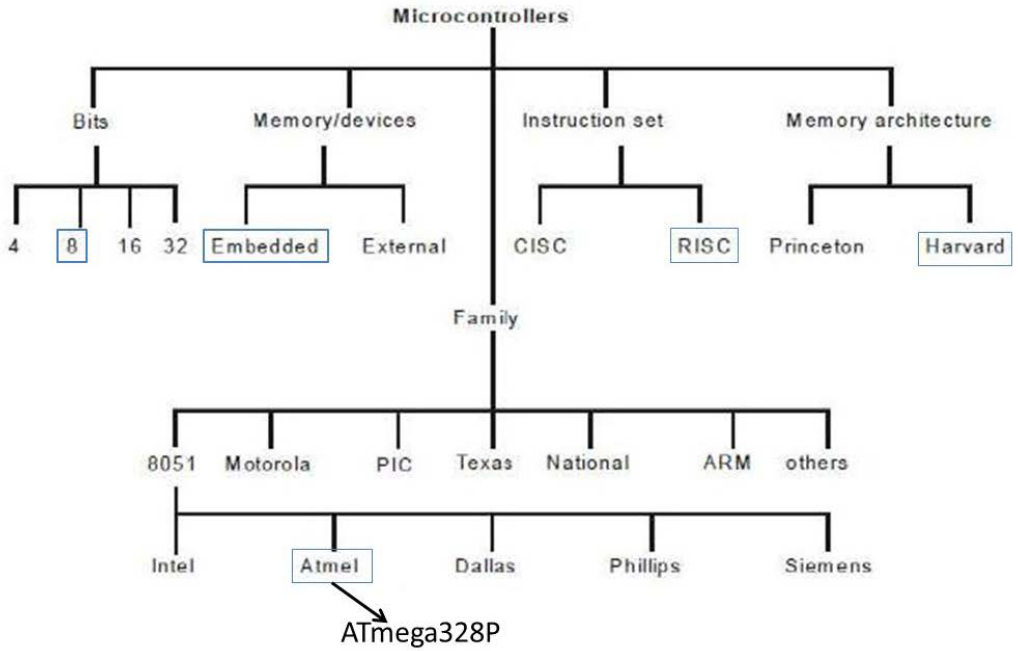


Figure 2: *The many families of Microcontrollers.*

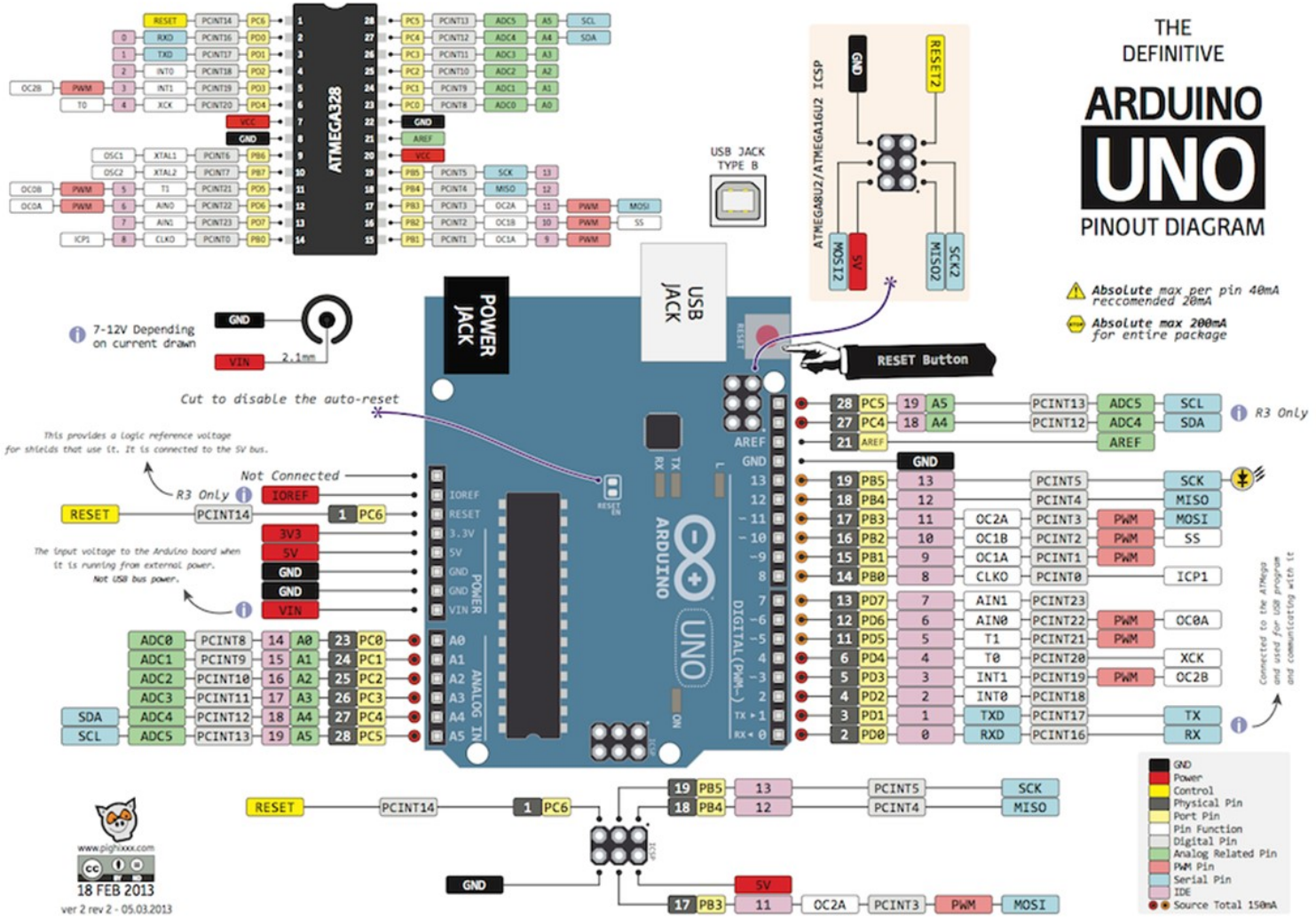
Evaluation vs. Development

In the parlance of electrical and computer engineers, the Arduino Uno is a *development board* and the Sparkfun RedBoard is an *evaluation board*. The *evaluation boards* are usually cheaper and used to test the suitability of a particular microprocessor for a given application. The board provides an interface so that the microprocessor can be programmed easily, and interfaced quickly to external hardware for fast proto-typing. The *development board* can also be used to evaluate the microprocessor, but has the additional feature that once the hardware and software design are stabilized, the processor chip can be removed and embedded in a more permanent proto-type. The board can then be used as a way to program changes to the processor chip that can be removed and re-inserted. The Arduino fulfills the role of a *development board*. If you look at the header pins (the black strips that give access to the signals and power available on the board. All of the input/output pins are connected DIRECTLY to the processor chip. With very few pieces of external circuitry – notably the voltage regulator and the clock – you can embed a fully programmed ATMega328P microprocessor into your hardware project with less expense (but more time) than just inserting the full Arduino board.

Notes:

Hardware Reference

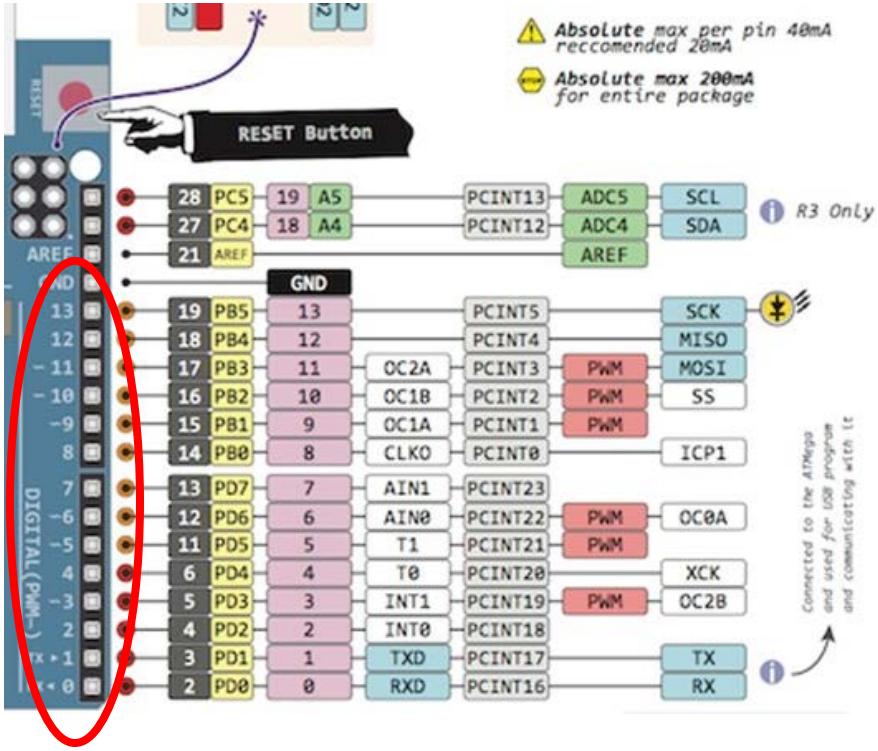
The figure below summarizes the functionality of the Arduino Uno's interface. As stated in the title, it is the definitive summary of the pinouts of the board and how they are connected to the microprocessor chip itself (sub-figure in the upper left corner). Once you learn to interpret the labeling properly, it will become the only hardware reference you need.



Introduction to the Arduino/Reboard

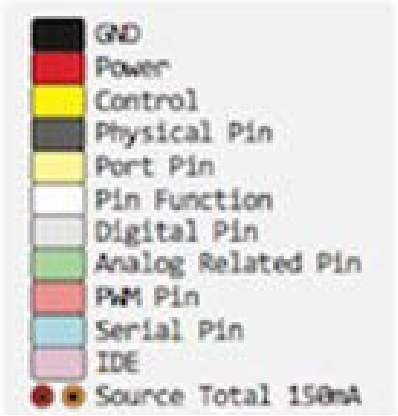
Notes:

The initial reaction to this figure is horror until it becomes clear that each pin on the board is connected directly to the processor chip and that some of the processor pins (and hence the pins on the board) can have several uses. For example, pins 0-13, as labeled on the board itself, and echoed in the pink boxes (the IDE reference values) are *digital input/output* (I/O) pins. This means that you the programmer must specify in your code whether these pins are to be used as input pins or output pins. You will come to understand this better as you work with the board.

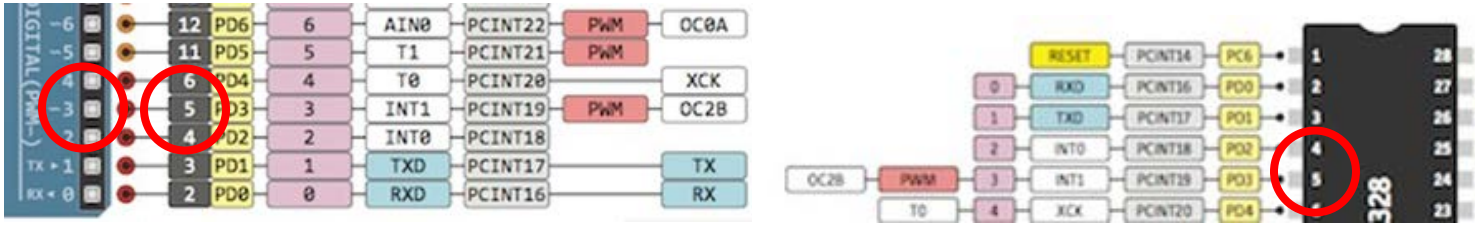


Notes:

The secret to understanding this diagram resides in the legend. The color of each pin label indicates its functionality and how the pin may be configured in different applications.



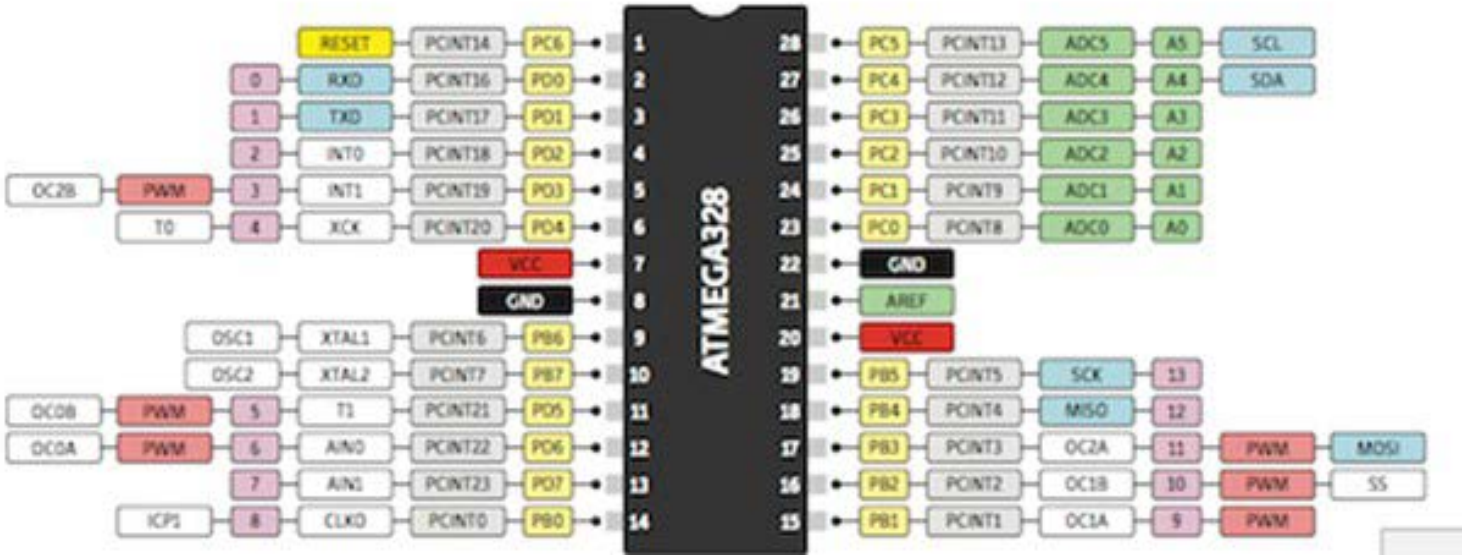
For example, note how Arduino's pin 3 (as it is silkscreened on the blue board) is directly connected to pin 5 (black) of the ATMEGA328 chip. Both diagrams explain that this is a digital pin (only takes on values of 0 or 5 volts) and that this particular digital output offers the PWM configuration option (pulse-width modulation) and that the IDE (code you write) should reference this pin as "3" (pink). The small tilde (~) preceding the 3 on the board (~3) reminds the user of the PWM capability without having to reference this reference guide while building.



Answer Question 2.

Notes:

If you were to build a stand-alone hardware solution, the direct pinout of the ATMEGA328 chip would be needed.



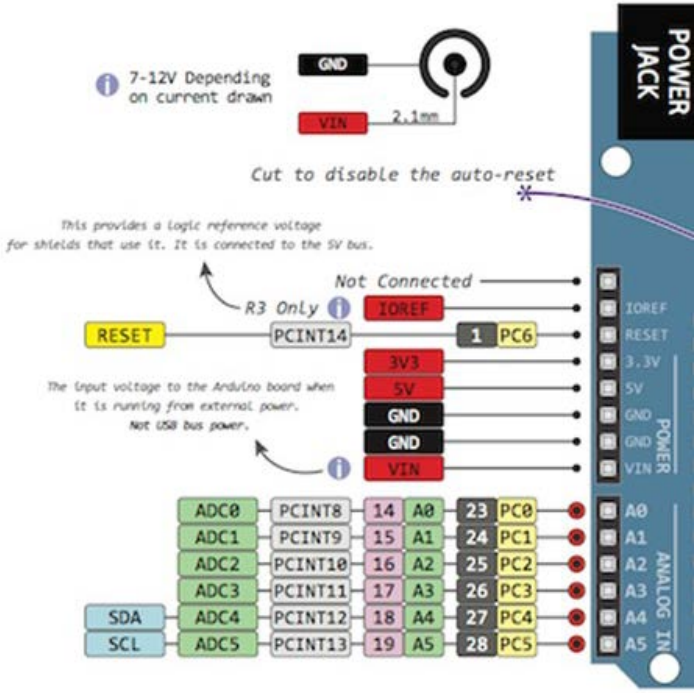
Everywhere you see a pink box, we know that particular ATMEGA328 pin has a direct connection to one of the connection points on the Arduino board for convenient user interfacing.

Notes:

On the left-hand side of the Arduino board, you will find, for example, pins to provide “ground” (GND) to your electronics circuit as well as options for 3.3 volts, 5 volts, or even the full USB or battery voltage (VIN). A reset pin is available for allowing your circuit to choose to reset the board (maybe you can think of a reason to need this?). Then, there are also six ANALOG IN pins labeled A0 through A5. These pins use an analog-to-digital converter to read a voltage signal and translate that voltage into a numerical value. There are two catches with the ANALOG IN:

1. The voltage you are measuring should be constrained to be between 0 and 5 volts (compatible with the design of the Arduino board) and
2. The numerical value will be displayed as a decimal number, but actually stored as an 10-bit binary number, thus limiting its values to live between 0 and 1023 (the decimal representation of the largest 10-bit binary number you can have).

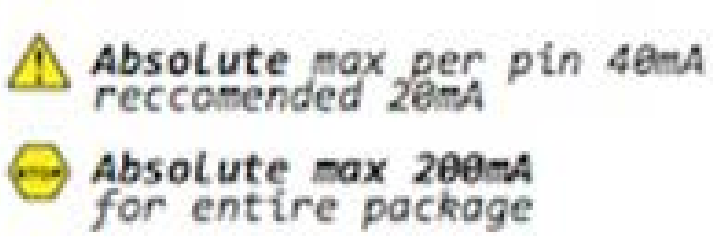
These two facts result in the necessity of recognizing/mapping numbers between 0 and 1023 to voltages between 0 and 5 volts during the design phase.



Answer Question 3.

Notes:

One final, yet important, item to mention is that the Arduino cannot be expected to supply large currents. In fact, a circuit that attempts to draw currents in excess of the rating from the board is likely to result in damage (like placing a low-resistance wire between the 5V output and GND!). For now, you are likely safe as long as you follow instructions for circuits that have been designed for the Arduino. Later, when you do your own designs, you should keep these limits in mind.



The other pin labels are associated with functionality for the more-advanced user. For example, the yellow labels show the pin names used when identifying the pins as ports. In this case, the processor is programmed at a lower level (assembly) where the inputs and outputs can be read and manipulated more quickly using the port designations rather than the designations used in the Arduino IDE.

The light blue labels are associated with the pins that can be configured to serve as a *serial communication* interface to and from a connected computer or other hardware with a serial interface. For example, the XBee, a tiny board that transmits and receives signals wirelessly, uses the serial lines to communicate to the processors on both ends. You may want to do some research to find out what this means. One type of serial connection is the protocol and hardware interface used by the USB ports on most modern computers.

Answer Question 4.

Notes:

Programming the Arduino Board

The Arduino Development Environment is a C-like programming environment

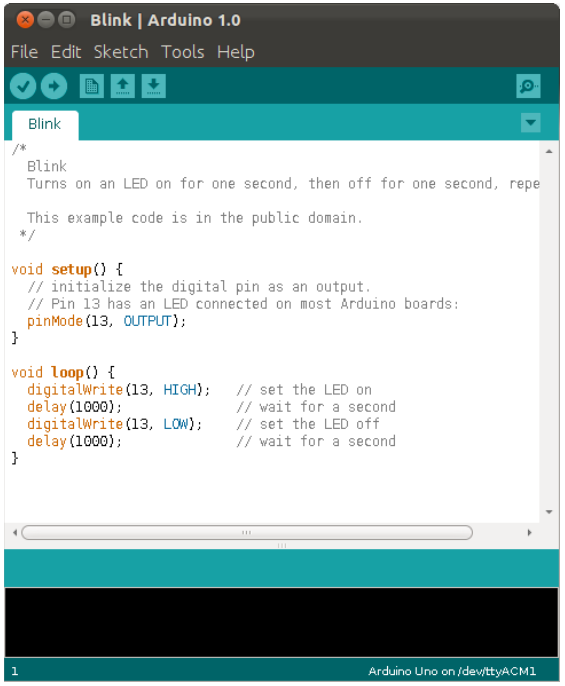
The Arduino development environment is the interface that you will use to tell the board what to do. A free program named ***Arduino*** along with some driver software is available to you and has been embraced by a large community. Most projects that you find on the web come with accompanying software written for this interface. This type of interface is commonly designated by the acronym IDE (Integrated Development Environment). MATLAB is also an IDE. There are several resources available to guide you through downloading the software onto your own computer.

Sparkfun Tutorial: https://learn.sparkfun.com/tutorials/redboard-hookup-guide

Arduino's own Tutorial: http://arduino.cc/en/Main/Software

Judy Culkin's **Graphic Novel** Tutorial: http://www.jodyculkin.com/wp-content/uploads/2014/03/arduino-comic-2014.pdf

Washington DC's Hackerspace Tutorial: http://www.hacdc.org/summer-school-2013/



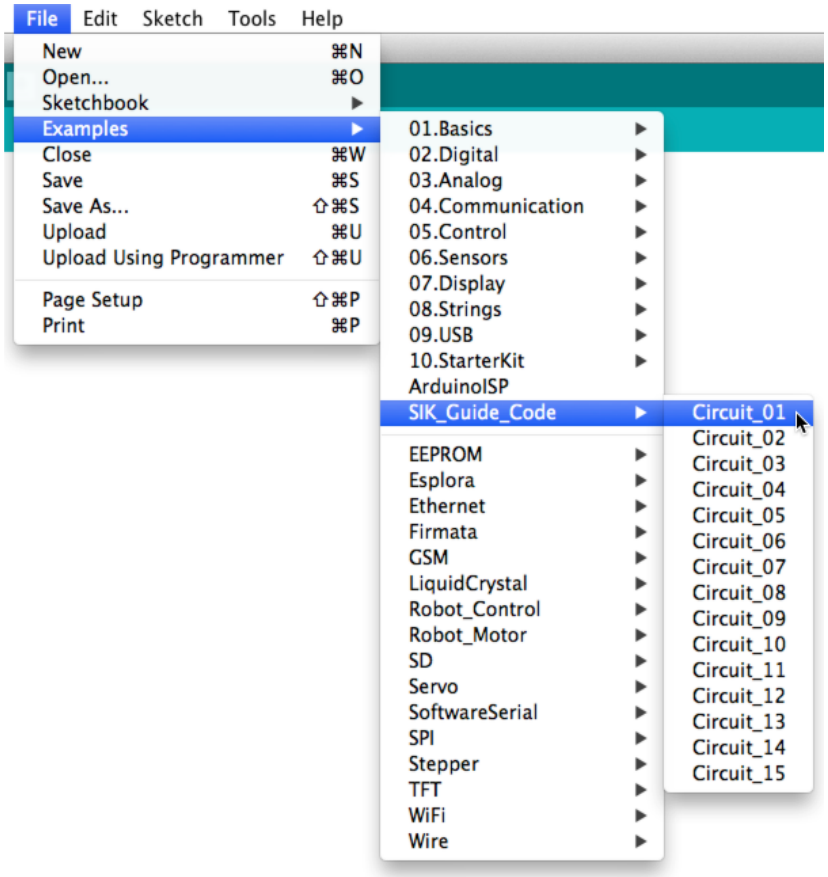

```
/*
  Blink
  Turns on an LED on for one second, then off for one second, repe

  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(1000);              // wait for a second
  digitalWrite(13, LOW);    // set the LED off
  delay(1000);              // wait for a second
}
```

Notes:

In the lab, all of the computers have the Arduino software installed but it is recommended that you also have a copy on your own computer so you can work from home if you own or purchase an Arduino or its clone.

Arduino Libraries

Many users have written libraries for the Arduino boards. A Library in this context is a group of function written and stored in a zipped file that is downloadable. To have the library contents available inside the Development environment, simply run the Arduino program. Find the menu item *Sketch* and inside the drop-down menu at the very top should be an item *Add Library…* Clicking on this item will bring up a window asking for the library file you wish to include. There is an enormous Arduino community and many free libraries.

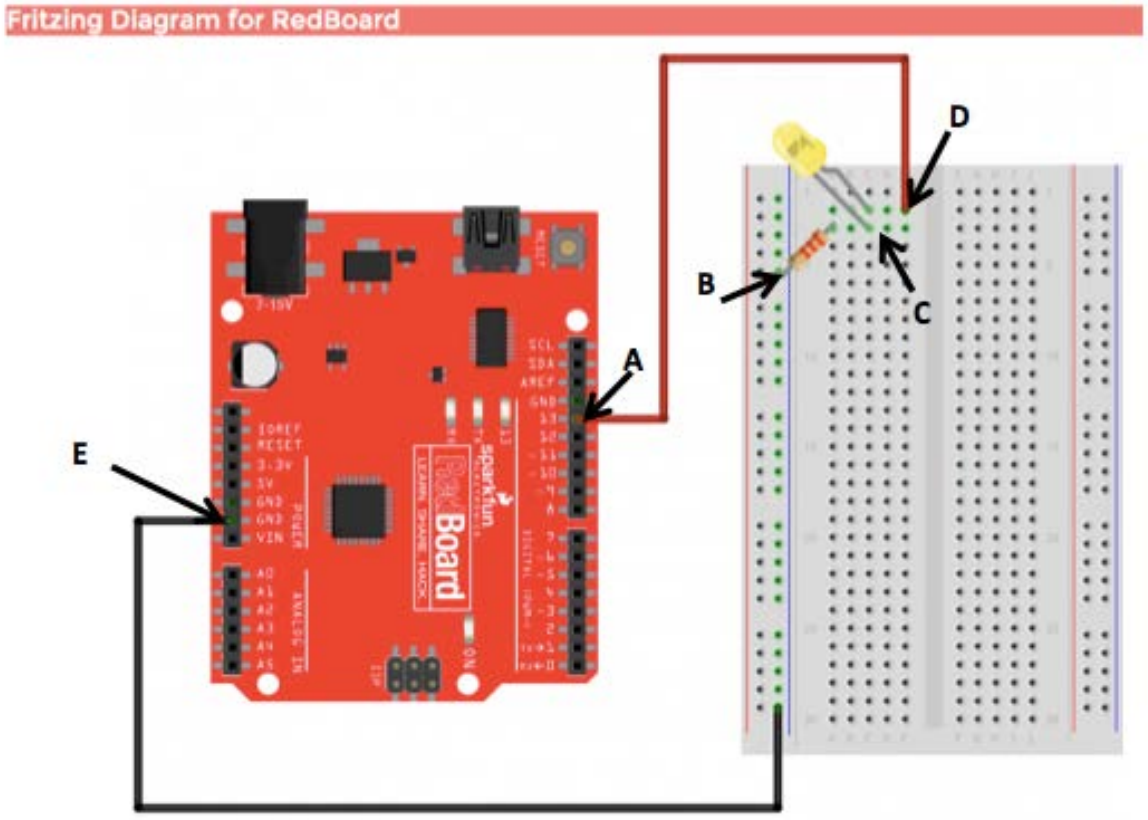


Notes:

Other Options: Assembly Language Programming

Code can be written in C or C++ and using the appropriate assembler convert the code to something that will run on the Arduino. Or you can write assembly code directly using an IDE like AVRDude that can be downloaded from the web.

Uploading and Running code through the Arduino IDE



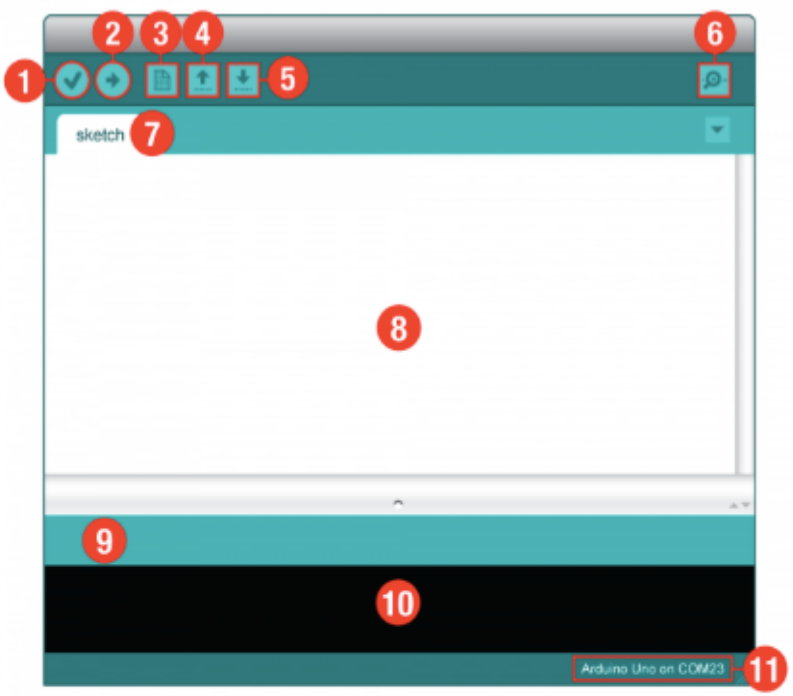
Build the simple circuit above.

Notes:

To make the LED blink you need to have a program running on the Arduino board that raises and lowers the voltage on Pin 13.

- ✓ Run the Arduino program on your computer.
- ✓ Open up a pre-written file containing code that will blink the LED. To find the program navigate through the menus File > Examples > 01.Basics > Blink and a window will open up with the software needed to blink the LED by using the Digital Pin 13 as an output.
- ✓ Insert the cable in your kit between the computer and the board. The mini USB side is connected to the adapter on the board while the other end goes into any available USB port on the computer.
- ✓ Upload the program to the board by clicking the upload button (labeled 2 in the diagram below). *NOTE: Common reasons for this step to fail are: i) the Arduino does not know which of the many boards you are using or ii) the COM port (USB port on the computer side that you plugged the board into) is incorrect. Both problems can be addressed by looking in the Tools option in the menu bar.*

Notes:



1. **Verify:** Compiles and approves your code. It will catch errors in syntax (like missing semi-colons or parenthesis).
2. **Upload:** Sends your code to the RedBoard. When you click it, you should see the lights on your board blink rapidly.
3. **New:** This buttons opens up a new code window tab.
4. **Open:** This button will let you open up an existing sketch.
5. **Save:** This saves the currently active sketch.
6. **Serial Monitor:** This will open a window that displays any serial information your RedBoard is transmitting. It is very useful for debugging.
7. **Sketch Name:** This shows the name of the sketch you are currently working on.
8. **Code Area:** This is the area where you compose the code for your sketch.
9. **Message Area:** This is where the IDE tells you if there were any errors in your code.
10. **Text Console:** The text console shows complete error messages. When debugging, the text console is very useful.
11. **Board and Serial Port:** Shows you what board and the serial port selections

Figure: The IDE interface.

Answer Question 5.

Notes:

Explore More!

Name: _____

Points awarded: _____

Net ID: _____

Notes:

Module 003: Introduction to the Arduino/RedBoard

Question 1: Research online or in the library the distinction between the Princeton and Harvard architectures.

Question 2: Which physical pin numbers (as silkscreened on the Arduino) are capable of providing a PWM signal? What is a quick way to identify them?

Question 3: How are the pins labeled **ANALOG IN** to be referenced when programming with the Arduino IDE?

Question 4: What are the suggested limits for the amount of current that can be drawn from each pin and the entire processor chip?

Notes:

Question 5: Look at the Arduino Code and all of the lovingly written comments that explain how to build the circuit, connect it to the processor board, run the 'sketch', and a short explanation of the structure of the code. Please read them – below is a condensed version without the comments so you can see more clearly the code structure. Using the information in the comments or on the Arduino website explain the function of the ***setup*** and ***loop*** portions of the code.

```
/*
SparkFun Inventor's Kit
Example sketch 01 - just the facts
*/

void setup()
{
  pinMode(13, OUTPUT);
}

void loop()
{
  digitalWrite(13, HIGH);   // Turn on the LED
  delay(1000);              // Wait for one second
  digitalWrite(13, LOW);    // Turn off the LED
  delay(1000);              // Wait for one second
}
```