Name/NetID:	Teammate:

Module 4B: Arduino Input/Output (I/O) Pins in OUTPUT mode

Module Outline

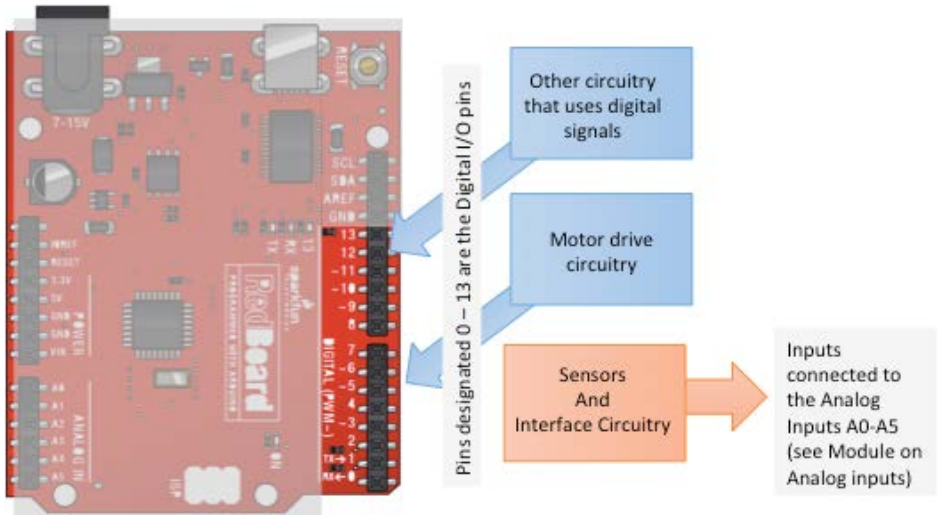
In this module you will be learning how to use the Digital I/O pins on the Arduino. Digital? What does digital mean in this context? So far voltages and current are related to physical processes that you have learned to model – what does it mean to say that a signal is digital? The Arduino board has analog input pins that you will learn to use in a subsequent module – how is digital different from analog?

The history of the Digital Age is a fascinating one and the personnel in the ECE department played no small role in this history (and TI too). The achievement of being able to miniaturize devices that are to first order electronically controlled switches truly changed how humans live their lives. This ability allowed devices based on the concept of ON and OFF to be put together to build computers that are now an integral part of our lives. Using the digital I/O pins will help you see how this abstract Boolean 2-state notion of ON or OFF maps onto real voltage signals.

After constructing the chassis of your car and mounting the motors, the only way to get the vehicle to move is to physically connect the motor terminals to either the power supply or battery. So far you have done this by hand or with hand-operated switches. The *autonomous* in autonomous vehicle means that your car must navigate on its own responding only to the input of sensors and other devices that monitor some parameter of the environment.

Multi-Valued Logic – our current computers are based on a 2 – valued logic – a concept dating back to at least the ancient Greeks. Everything is either TRUE or FALSE, ON or OFF, but there are peopling looking into the value of a construct that has more than just 2 states. I wonder what advantage there would be to having a 3-state logic system with TRUE, FALSE, and MAYBE. Or 4 or 5 or… isn't a signal we call an *analog* signal just a system with an infinite number of states?

.

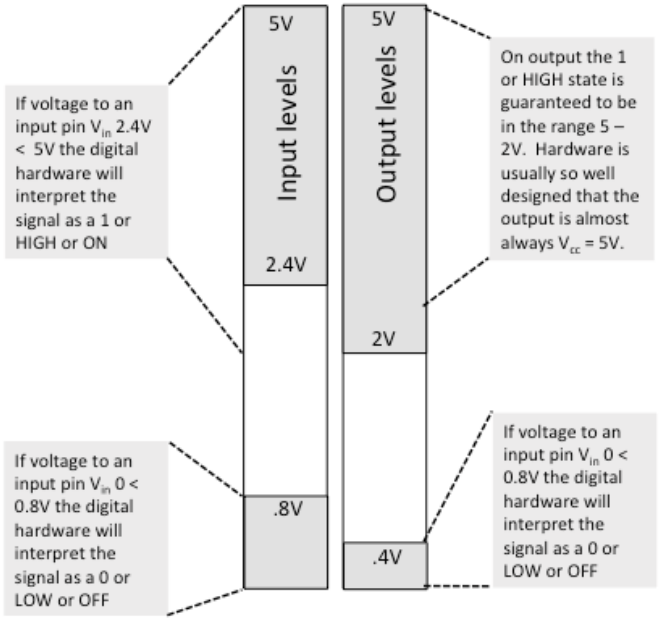


Notes:

At the hardware level a digital signal is just a voltage that either has a “high” value or a “low” value. These values are different for different families of devices – the terminology *family* refers to the fabrication process that made the digital hardware. The Arduino board and clones are made to interface with two logic types or families – TTL transistor-transistor-logic and CMOS complementary-metal-oxide-semiconductor. Each has a different definition for what constitutes a HIGH and LOW voltage. The SparkFun website https://learn.sparkfun.com/tutorials/logic-levels has a marvelous and more detailed description.

Any of the logic families works with a range of voltages – the TTL family used a voltage range from 0-5V and the CMOS most typically uses a range from 0-3.3V. This is where things get a little confusing because the hardware dealing with the digital signals must define ranges that are considered “high” and ranges that are “low”. In most technologies there is a middle range of “I don’t know” where the hardware behaves unpredictably. If a digital device is connected to another digital device this is not a problem since the output levels are always in the appropriate ranges. It is more problematic when interfacing to analog hardware.

The figure below summarizes how the Low-Power Schottky TTL circuits behave on both input and output. The term Schottky refers to a fabrication method used to make the transistors on the integrated circuit switch more quickly without compromising low-power consumption. The digital inputs of most circuitry can be considered perfect measuring devices meaning that the behavior of the circuitry connected to the input pin is not modified and the internal hardware interprets the voltage range 0-.8V as a LOW or 0 state, and 2.4-5V as a HIGH or 1 state. The outputs can be considered ideal voltage sources. Even though there is a range for the voltages associated with HIGH and LOW most Integrated circuits provide voltages that are nearly the highest (in this case 5V) as a HIGH and 0V as a LOW. In fact a voltage that deviates much from 5V indicates that there is a problem with the circuit.



TTL LS family – this is the logic family that is used in our lab almost exclusively. The naming convention includes the designation of the type of TTL logic used. LS stands for Low power Schottky. It’s advantage is lower power at the cost of speed. There are many other types of digital circuitry mostly optimized around speed, power, or device size.

Notes:

The electrical interface of the digital I/O pins on the Arduino are actually more flexible than most common logic families. They were constructed so that the input pins recognize as many different types of devices as possible. On output the Arduino consistently provides a HIGH voltage very close to 5V and a LOW close to 0V. The output pin interface on the board are well-designed so that the voltages remain the same for a wide range of loads acting like an ideal voltage source that is either ON or OFF as the pin is connected to hardware that draws 40mA or less.

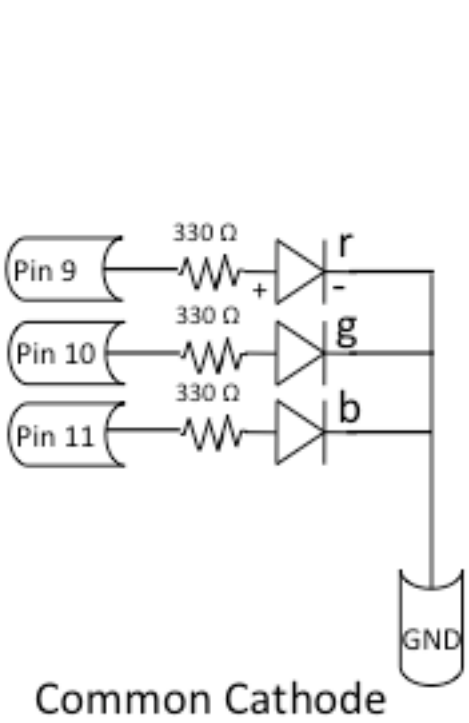
Notes:

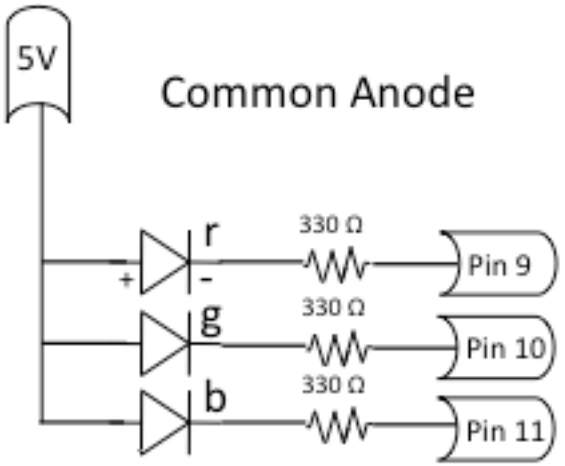
Procedures

OUTPUTS

Most of your designs will involve using the digital I/O pins in output mode so the questions are primarily directed at using them correctly. For the robot cars you will need to turn the motors on and off in response to other inputs to steer the vehicle. As you will see by turning these digital output pins on and off really quickly you can also control the speed of the motors.

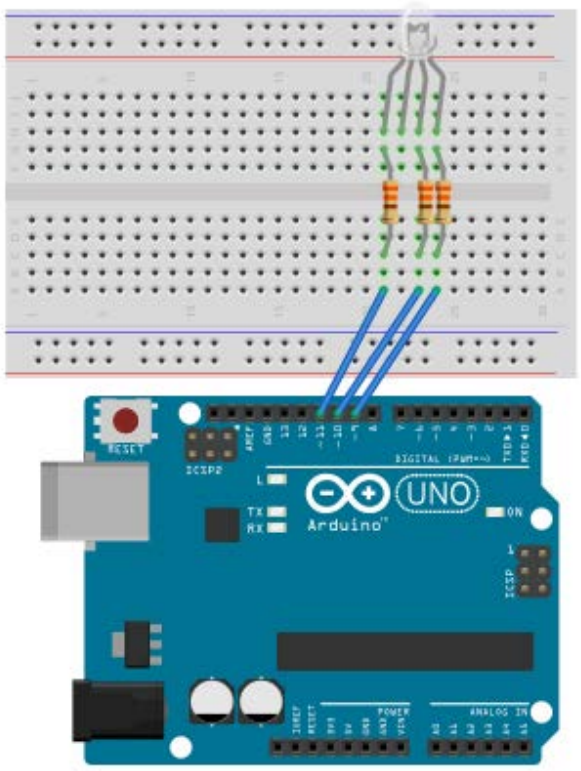
Instead of beginning with controlling your motors let's begin by turning ON and OFF an RGB LED. The RGB LEDs come in two flavors the Common Cathode and the Common Anode. Cathode and Anode – interesting jargon used to denote the positive and negative terminals of devices NOT necessarily respectively. In general the current enters the *anode* exits through the *cathode* as looking from OUTSIDE the device. To begin you must determine which type you have in your kit.





Notes:

Question 1: Determine which configuration you have by determining the common pin (either anode or cathode) and inserting one of the diodes (red, green, or blue) across 3.3V in both orientations remembering that diodes only conduct current when the voltage at the + terminal is sufficiently greater then that at the – terminal. Once you know draw on the physical diagram below how the power and/or ground should be connected. Use the schematics on the previous page as a guide. NOTE: The common pin is not connected in the diagram below and I believe it is the shortest pin.



Notes:

To make the LED turn on you need to apply enough voltage across the terminals. To turn the LED on and off the voltage across its terminals must go high and low. This is where the digital I/O pins on the Arduino become useful. A program running on the Arduino board that raises and lowers the voltage on one of the I/O pins is simple to write.

- ✓ Run the Arduino program on the computer that will be used to download the program and power the board.
- ✓ By default a window will open with an empty Sketch.

```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

- o Setup code to designate Pin 11 as a digital output pin. Since these pins can be either input or output a special statement is needed. Amend the code (see below). NOTE: syntax and semicolon at the end of the line.

```
void setup() {
  // put your setup code here, to run once:

  pinMode(11,OUTPUT); // set pin 11 in OUTPUT mode
}

void loop() {
  // put your main code here, to run repeatedly:

}
```

- o Turn on one of the LEDs – the one connected to pin 11 whichever that is by telling the microprocessor to connect Pin 11 to 5V. There are two versions of the code, one for the common anode RGB LED and one for the common cathode. Refer to the schematic to determine which you. Or trial and error works too.

```
void setup() {
  // put your setup code here, to run once:

  pinMode(11,OUTPUT); // set pin 11 in OUTPUT mode

}

void loop() {
  // put your main code here, to run repeatedly:

  digitalWrite(11,HIGH);

}
```

```
void setup() {
  // put your setup code here, to run once:

  pinMode(11,OUTPUT); // set pin 11 in OUTPUT mode

}

void loop() {
  // put your main code here, to run repeatedly:

  digitalWrite(11, LOW );

}
```

Notes:

- ✓ Upload code to your Arduino/RedBoard

Electric Characteristics of the Output Pins

Question 2: Measure the voltage delivered by Pin 11. All of the voltages on the Arduino are referenced to the pins labeled GND. What is the voltage? Is LED on? Is it red, green, or blue?

Question 3: Modify the code and upload it to the board so that the statement in the loop portion of the code sets pin 11 LOW instead of HIGH. Measure the voltage delivered by Pin 11? Is the LED on? Is it red, green, or blue?

Question 4: Include statements that allow you to control the other 2 diodes that are connected to pins 10 and 9. Remember to set them into output mode in the setup section. Experiment with the different combinations. What are all of the colors that you can get by simply turning the Red, Green, and Blue diodes in eight different configurations?

Notes:

Controlling the digital outputs

With only two additional statements you can control the LED by turning each on and off at different intervals. In this way you can obtain over 16 thousand different colors. The first statement that can be used is the universally useful ***delay*** function.

- ✓ Modify your code:

```
void setup() {
  // put your setup code here, to run once:

  pinMode(11,OUTPUT); // set pin 11 in OUTPUT mode

}

void loop() {
  // put your main code here, to run repeatedly:

  digitalWrite(11,HIGH);
  digitalWrite(10,LOW);  //these are ment to be the statements
  digitalWrite(9,HIGH);  //that you added in Question 4.  Your
                         //choices for the logic levels pins
                         //9, 10, 11 might be different.

  delay(1000);           //This is a delay statements that
                         //inserts a 1000 millisecond pause
                         //in the execution of the statements.

  digitalWrite(11,LOW);  //How does this statement change the behavior
                         //of the signal on pin 11?

  delay(1000);           //Another 1000 millisecond delay is added.

}
```

Question 5: Predict the behavior of the circuit.

Question 6: Through the parameter passed to the ***delay*** functions you have control over the time that the LED is ON and OFF. Experiment with values between 1-1000 keeping the parameter passed to the ***delay*** both functions the same for now. At what value are your eyes no longer able to see the LED turn on and off. This may be different for each of you.

Notes:

With the ability to delay the execution of statements you have gained control of the timing of the circuits attached to the digital outputs. A common method for controlling attributes of circuits that respond to the average flow of power is called *Pulse Width Modulation* (PWM). The LEDs are one such device and your motors are another.

Pulse Width Modulation uses a square wave with a period chosen that works best with the device/circuit it is connected to. By varying the duty cycle of the signal you can control the brightness of the LED or the speed of the motors. The brightness of the LED is not related to the behavior of the device, at least at the speeds that you are using to turn the device on and off, it is turning all the way off and all the way back on. It is ***your eye*** that does the averaging. When we use this method with the car motors, it is the motors that coast during the off time slowing down that over-all average speed of the motor.

Constructing a Pulse Width Modulated Signal

Question 7: Since you found the value where your eyes are unable to see the LED turn on and off, what is the largest period, or lowest frequency for the PWM signal so that your eye sees a steady brightness from the LED.

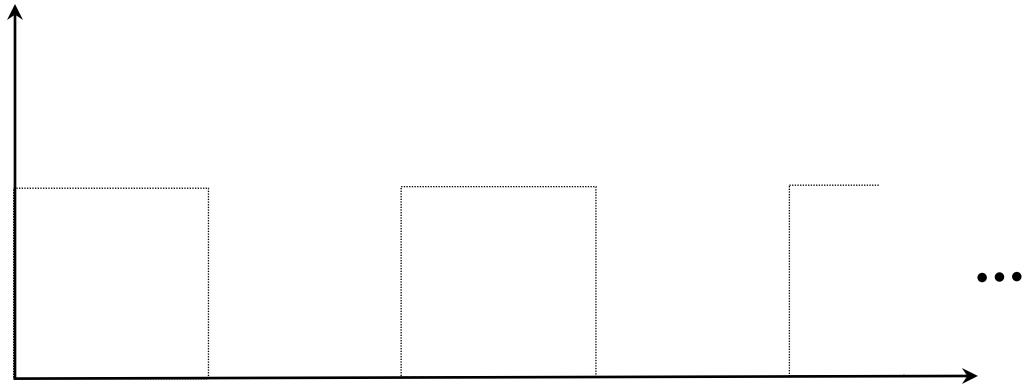
The signal generated on pin 11 is now a square-wave with a 50% duty cycle. The next questions have you construct square wave signals with the SAME period – 10 microseconds will be used – but a varying duty-cycle. The period chosen is an arbitrary number that is fast enough so that your eye can not see the flicker and – well its 100 not 117. NOTE: In the parlance of the signal generator this is a 100 Hz square-wave with peak-to-peak voltage of 5V and a 2.5V offset.

Question 8: Set the value in the ***delay*** functions both to 5, a value that should be above the one you found in Question 7. The LED should not be blinking but should be steadily on to your eyes. Vary the value of the value passed to the delay following the ***digitalWrite*** function that sets turns on the LED connected to Pin 11 from 5 to 1 – a couple of values should be enough to get the trend in behavior of the LED. Since you want to keep the total to remain 10 adjust the value of the other ***delay*** function to compensate. Describe what happens.

Notes:

Question 9: Reset the value in the ***delay*** functions both to 5. Now vary the value of the value passed to the delay following the ***digitalWrite*** function that sets output on Pin 11 LOW from 5 to 1. Again, since you want to keep the total to remain 10 adjust the value of the other ***delay*** function to compensate. Describe what happens.

Question 10: Make an educated guess about how the voltage varies with time using a sketch for 3 different pairs of delays. The dashed line is a square wave with a 10 microsecond period as a reference. Put as much detail as you can. Voltage levels. Time dependence. Make your best guess as you will be checking it in the next section of these procedures with the oscilloscope.

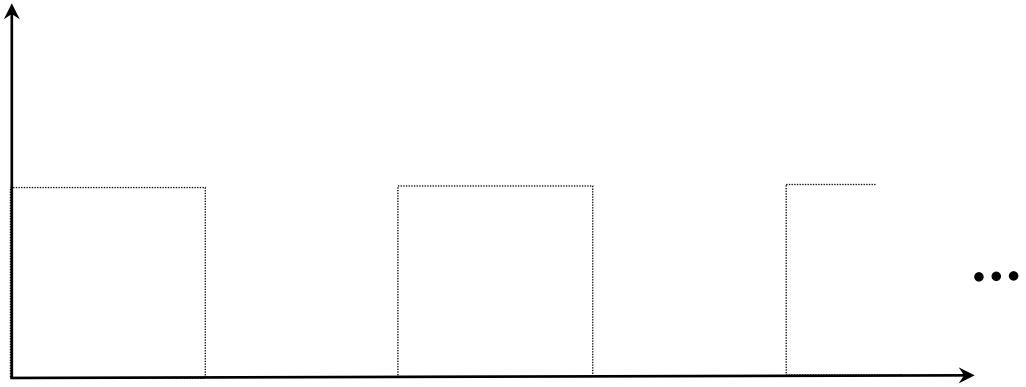


From the experimenting you have done so far it should be clear that the voltage signal that is driving the LED varies with time. It is always a good habit to check that the signals you generate.

- ✓ Connect the oscilloscope to measure the voltage across the LED setting the values of the delay functions to match the values used for one of your sketches you made for Questions 10.

Notes:

Question 11: Make corrections to you plots you drew if needed based on what the oscilloscope shows. Notate on the sketch the behavior of the LED.



Notes:

There is a shortcut to creating square wave pulses without resorting to manually setting the time the output signal is high and low if all that you want is a square wave with a FIXED period but a variable duty cycle. The Arduino instruction set includes a special function *analogWrite* to generate a very clean square wave with a variable duty cycle.

- ✓ Make a new Sketch – to be on the safe side save the new sketch to a filename of your choice. Now all of the edits will be to this new file.
- ✓ Enter this code:

```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:
  analogWrite(11,128);
}
```

- ✓ Upload the new program to the board.
- ✓ Look on the board at the pins labeled DIGITAL (PWM~). The pins with the ~ next to their number are the only ones that can be used with the *analogWrite* function. Luckily pin 11 is one of those pins. Your LED should be on.
- ✓ Monitor the voltage across the LED with the oscilloscope.

NOTE:

i. The *analogWrite* function DOES NOT need the digital I/O pin to be set up in OUTPUT mode.
ii. Two parameters are needed. The first is the pin number to control and the second is related to a parameter associated with the resulting square wave output. You will determine its purpose empirically but your probably already know its purpose.

Question 12: On a new graph draw/plot the waveform available on Pin 11 from the oscilloscope– notate the amplitude, period, and duty cycle.

Notes:

Question 13: Now vary the second number from 128 to 64. What changed about the resulting signal and what did not.

Question 14: Without experimenting guess purpose of the second parameter and the range of values. Try both setting it to 0 (which is the minimum) to the maximum value you just guess to see if you are correct.

- ✓ Add two *analogWrite* statements to the program to control the brightness of the other two LEDs. Set the second parameter to different values for all three – the red, green, and blue to see all the colors you can make with different amounts of red, green, and blue.

Question 15: (**Optional**) Google html colors, and locate the hexadecimal number associated with your favorite. The first two digits correspond to the amount of red in the color. The second two – the amount of green. The third – blue. RGB. Each pair of hexadecimal digits can be turned into a decimal number between 0 and 254 – how convenient. Formula: 16*(most significant digit) + (Least significant digit). See if you can recreate some of the colors.

Notes:

What if the motors on your car were connected in the same manner to the same digital outputs. The physics of the process is different. The brightness of the LED changes in response to a signal that is turned ON and OFF quickly because of the response time of your eye. The motors cannot start and stop instantaneously so a similar method can be used to control their speed. Instead of the averaging happening in your eye it happens in the energy exchange in the motor itself.

Question 16: Describe using the LED as an example how you would use this method to control the motor of the cars. You will experiment with this method – commonly referred to as Pulse Width Modulation - in a future lab.