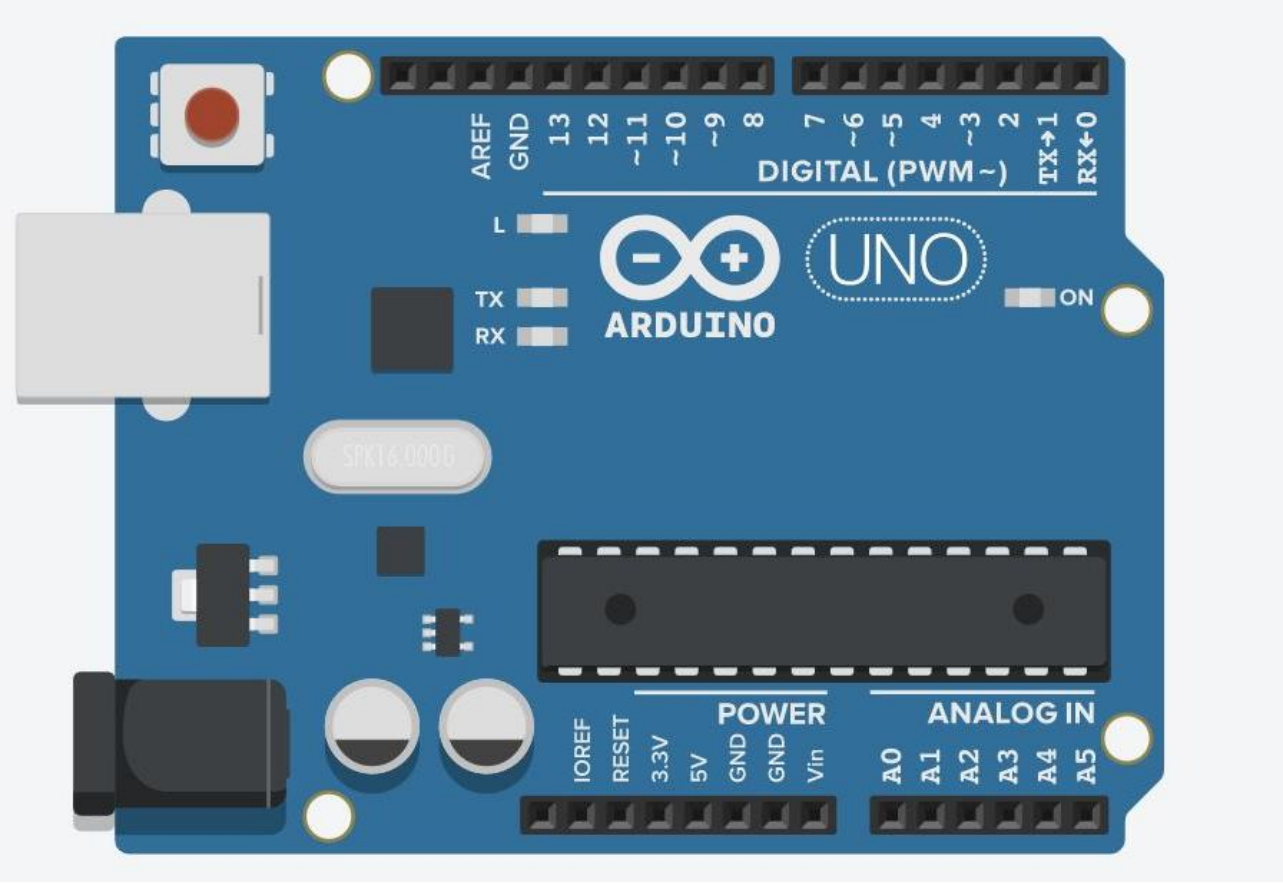
EVERGREEN VALLEY COLLEGE ENGINEERING

Arduino Features and Circuits



Experiment conducted by Harjot Gill

Engineering Processes & Tools (ENGR-10)

Dr. AH Tabrizi

TABLE OF CONTENTS

EXECUTIVE SUMMARY

In this project, the goal was to create a variety of circuits using the Arduino uno R3 board. TinkerCAD was used to create these circuits and develop the code necessary to run them. The Arduino uno R3 board comes with a variety of attractive features that make it easy to use and safe for beginners. At first, a physical circuit must be made which connects components where they need to be with the help of a breadboard. In short terms, the components must connect to power, and ground at the minimum. Resistors are added if necessary, using the help of Ohms Law. If it is a sensor, it must be connected to the one of the Arduino's pins. These pins range from Analog input to digital input/output which adds versatility to what can be read into the Arduino.

After a circuit is built and inspected, A code must be developed to program the Arduino to do whatever the user would like. This ranges from, blinking an LED light, running a DC motor, Writing onto an LCD monitor, and more. At the minimum, an Arduino code consists of two functions. One to setup, declare what pins on the Arduino are used for, and more. Inside the second function, the "action" takes place. This is where the commands are given to blink an LED light for example. This function will continue to run if the Arduino is connected to power.

In summary, six total circuits were created. One circuit was made to blink a red and yellow LED light. This was later modified to flash depending on the readings of an ultrasonic distance sensor. The third circuit created was made to turn on a DC motor. A potentiometer could be used to control how much the motor could spin. Another circuit that was made was to display a message onto an LCD board. This setup required a variety of wires and connections. A breadboard was used to organize the wiring. Breadboards were used for many of these circuits because the connections could be made without the need of soldering. The code for this program required the use of a library or header file. After completing this circuit, it was modified to output the temperature reading from a temperature sensor.

After developing these circuits, a natural order was determined on how to successfully create a circuit. First, all the necessary components were sourced, and their specs were studied. This was to prevent burning out the component and to provide proper resistance using Ohms Law. Then, using a breadboard, all the components were wired to the Arduino. After, a code would be developed, and the proper pins would be given a input/output role. Variables would be determined, and the loop function will be filled out based on the users needs.

INTRODUCTION

In this experiment, a variety of circuits will be created utilizing Arduino Uno R3 boards. More specifically, six circuits will be developed ranging from blinking an LED light, to operating a DC motor, and writing a message onto an LCD display. A code will also be developed to program the Arduino boards. More details will be given later about why it is needed and how the circuit board works. Since a physical board is not available at the moment, TinkerCAD will be used instead. This is a website created by Autodesk, which is a company known for creating and developing 3-D modeling software.

Speaking of circuits, it is important to understand the basics of electric components and physical laws related to electricity. Ohms law for example is important when dealing with LED circuits. If the proper amount of resistance is applied to the system, the LED will not burn out. If the specifications are known about the LED about how much amps it needs to run, Ohms law can be used to calculate the amount of resistance needed to run the system safely and reliably. Since a software will used, it will be easy to adjust these variables if necessary.

As mentioned before, a code is used to control the system and acts as the brain. The Arduino board can download codes from a computer via USB input. The programming language used is C++ and it can be used to perform a variety of tasks, loops, and calculations. This is what controls what the Arduino reads, inputs, and reacts to. After the code is finished it will be transferred to the Arduino board and will compile and run the program continuously if the board is hooked up to power.

THEORY AND FEATURES

As stated before, the Arduino Uno R3 board comes with a variety of features that makes reading sensors and programs easy. Before that is discussed, it is important to have some clarification on how exactly the board works.

An Arduino board is simply, put a programable "microcontroller-based open-source electronic prototyping board". It can plug and play a variety of sensors and inputs to its 6 analog and 14 digital pins. The analog pins read voltage from analog sensors and convert them to digital readings for "system understanding" (https://www.youtube.com/watch?v=_ItSHuIJAJ8).

They can also be used for digital output and input. This is like the Digital pins found on the opposite side of the Arduino. These pins however can act as a power source for output sources like LEDs and DC motors. The role of the pin is determined by the user in the code produced.

The code is an important part of the system. The Arduino can perform tasks depending on what program is coded onto it. This program can be created by the user on a free software provided by the Arduino company. The program can be uploaded to the board from the Arduino IDE onto the USB connector which will then make it way into the microcontroller. A USB interface chip is integrated to the board to convert USB signals into signals the Arduino can understand. The microcontroller acts as the "brain" of the board which consists mainly of a CPU, flash memory, RAM space and program reader. This essentially runs the program.

Other pins exist on the board which allows the user to ground electrical circuits, power external components, and more. All these components however must be powered by an external power supply. A DC/AC power pin is attached to the board to do just that. If there is power to supply the board, the program will run in a loop. It is important to limit the amount of power the Arduino receives to a maximum of 20 volts. If this is exceeded, a voltage regulator is activated as a safety measure to protect the board from burning out. Note that external equipment and sensor may not have these precautionary regulators and the use of resistors is necessary.

As stated previously, it is important to understand the basics of an electric circuit before building one. A fundamental rule to follow is to provide the proper resistance to the circuit if a component is added to it, like an LED. The movement or flow of electrons is called **current**. The **voltage** is the pressure which forces the electrons to move. **Resistance** is the opposing force which resists the flow of the current. Ohms law can be used to calculate the resistance needed to provide a safe amount of current and voltage to the components. After reworking the equation to solve for resistance, Ohms law goes as followed.

$$\frac{\boldsymbol{Voltage\ (V)}}{\boldsymbol{Current\ (amps)}} = \boldsymbol{Resistance}\ (\Omega) \qquad \text{(Equation 1)}$$

Since the Arduino outputs 5/3.3 Volts, the max current rating on a component must be known. Once it is found, the proper amount of resistance can be applied.

EQUIPMENT

1. **Arduino Uno R3 Circuit Board**
2. **2 LED lights (red and yellow)**
3. **Two resistors (1k Ω) and one 220 Ω resistor**
4. **Multicolored Male to Male pin Jumper wires**
5. **Breadboard Mini & Small**
6. **DC Motor**
7. **TMP 36 (temperature sensor)**
8. **LCD Board**
9. **Potentiometer**
10. **Ultrasonic distance sensor**
11. **DC/AC Power supply**
12. **USB A to USB B (male)**

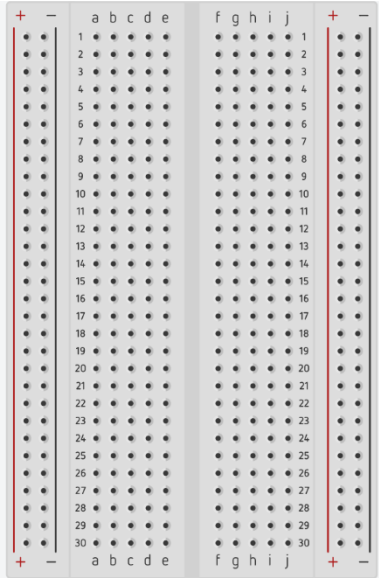


Figure 1: Breadboard used to organize the circuit and wiring.

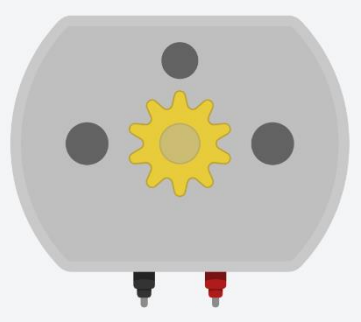


Figure 2: DC motor with two terminals

Figure 3: TMP36 sensor with a Voltage Out, Ground, and Power pin

Figure 4: Resistor used to limit the current in a circuit

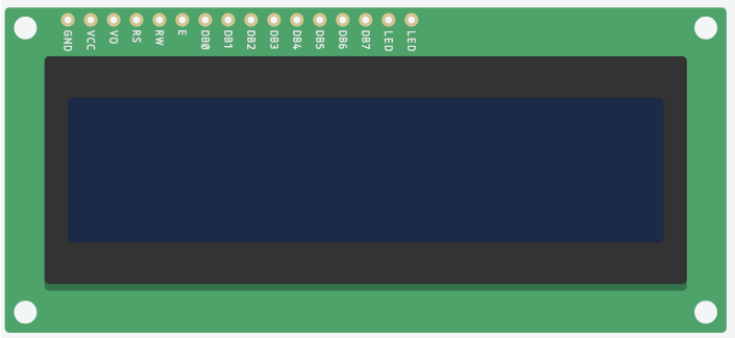


Figure 4: LCD board with multiple pins

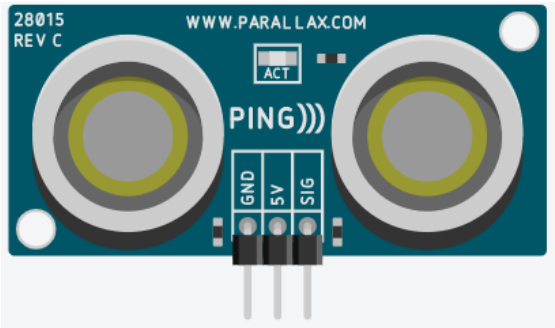


Figure 5: Ultrasonic distance sensor

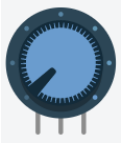


Figure 6: Potentiometer

PROCEDURE, DETAILS AND CIRCUIT RESULTS

1. LED blinking circuit

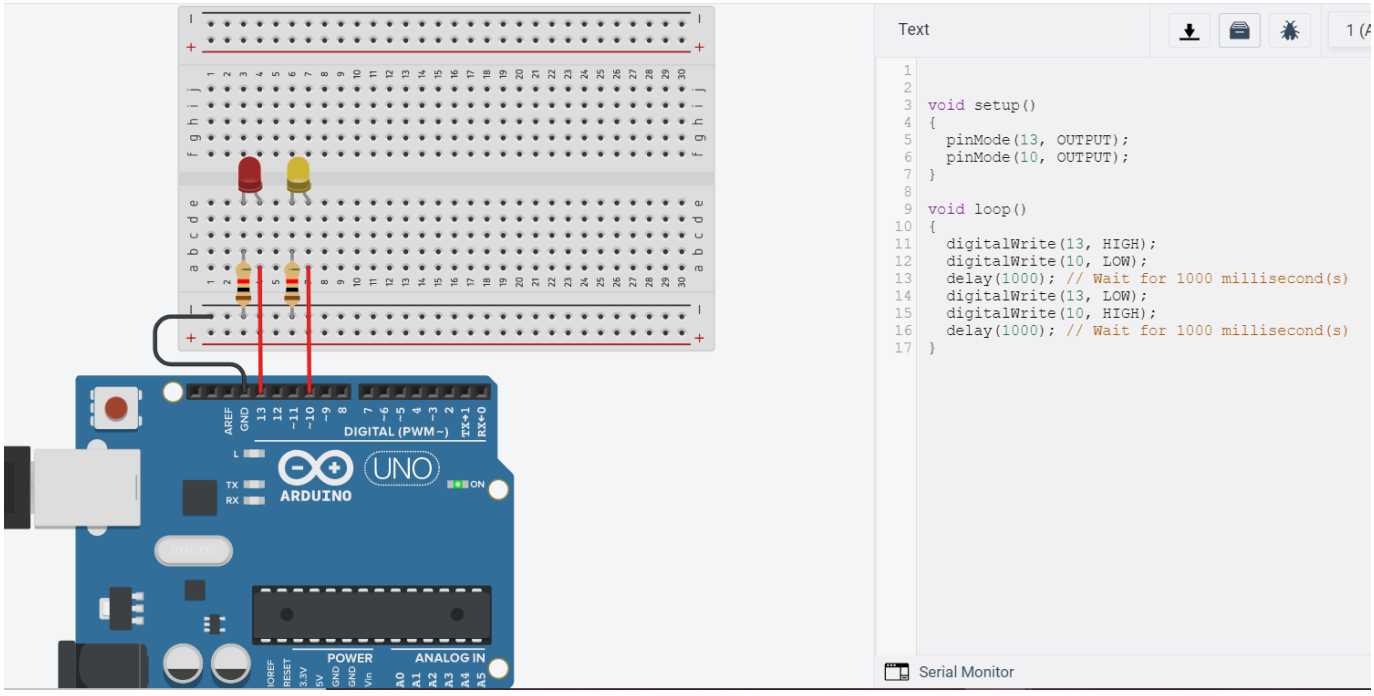


Figure 7: Displays the circuit as well as the program required to blink the LEDs.

1.1. Connect your LEDs to the breadboard so that it is in a single period and parallel to it. Both pins should be in the same row/period.

1.1.1. Note: (Figure 1) A breadboard is the construction base for the circuit, the advantage of using a breadboard is the fact that wires do not have to be soldered in order to be connected and transmit electricity.

1.2. Connect a black wire to the negative terminal of the breadboard to the ground pin of the Arduino.

1.2.1. Note: Black wires usually signify a negative terminal connection and ground.

1.3. Connect the 1k Ohm resistors (LED needs around 10 mA to run) to the cathode pin of both the LEDs as shown.

1.4. Connect the anode pin of the red LED to digital pin 13 and the yellow LED to digital pin 10

2. Developing a code

2.1. To the right of Figure 7 a code should be present.

2.1.1. Note: This code is written in a simplified C++ language which consists of two functions. One function to setup how the pins on the Arduino will be used, the Serial monitor baud rate for starters. The second function, the loop function, will run in a loop if the Arduino is connected to power.

2.2. Pins 10 and 13 are connected to the LEDs and should be stated as output pins that will send voltage to the LEDs. This is determined with the pinMode function.

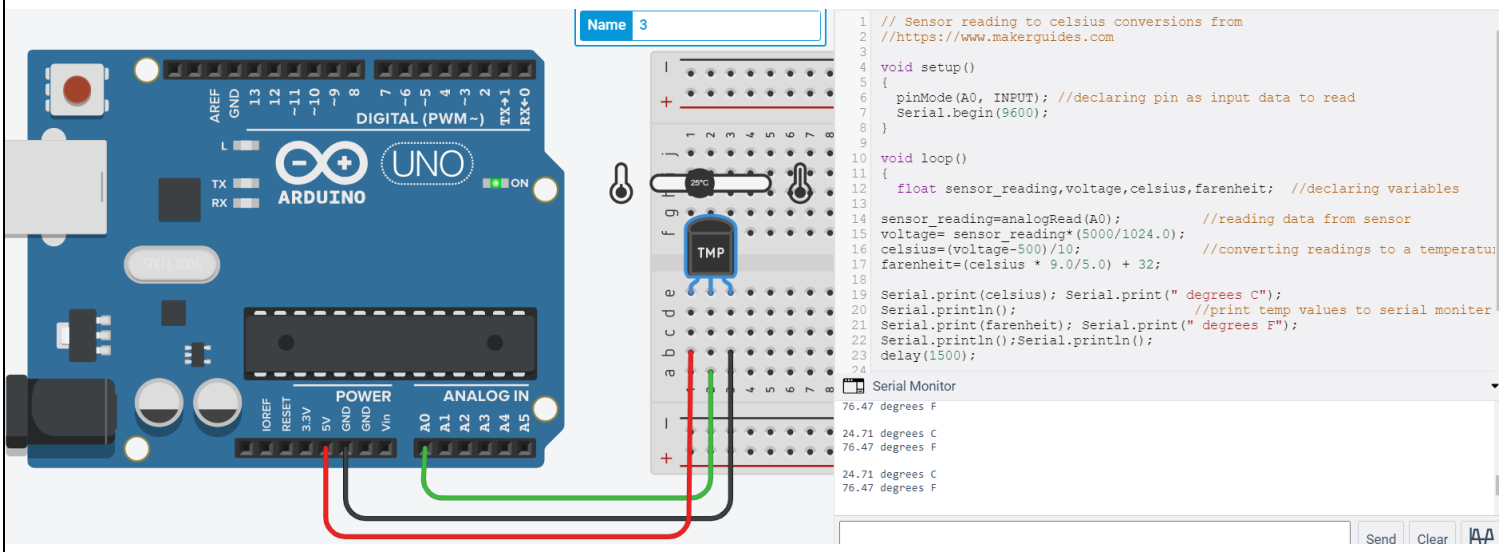
2.3. In the loop function, turn the LEDs to "HIGH" or on using the digitalWrite function. This command is used to control the components connected to the digital pins of the Arduino.

2.4. A delay is given of 1000 milliseconds (1 second)

2.5. Turn the LEDs off using the "LOW" command again using digitalWrite.

2.6. Another delay of a second is given and this is how the light blinks as the loop starts over.

3. Temperature sensor circuit




```
// Sensor reading to celsius conversions from
//https://www.makerguides.com

void setup()
{
  pinMode(A0, INPUT); //declaring pin as input data to read
  Serial.begin(9600);
}

void loop()
{
  float sensor_reading,voltage,celsius,farenheit;  //declaring variables

sensor_reading=analogRead(A0);           //reading data from sensor
voltage= sensor_reading*(5000/1024.0);
celsius=(voltage-500)/10;                //converting readings to a temperatu
farenheit=(celsius * 9.0/5.0) + 32;

Serial.print(celsius); Serial.print(" degrees C");
Serial.println();                         //print temp values to serial moniter
Serial.print(farenheit); Serial.print(" degrees F");
Serial.println();Serial.println();
delay(1500);
```

Figure 8: Displays the circuit as well as the program required to read the temp sensor.

3.1. Connect your power pins, negative to ground and positive to 5 Volts. Note that a resistor is not needed in this circuit.

3.2. The Middle pin for the sensor is for input to the Arduino. In this case it is connected to the A0 pin.

4. Coding the program to read the sensor and display the appropriate temperature value.

4.1. In the setup function, the pin A0 is declared as an input source and since a serial monitor will be used to print the temp values, the baud rate is also given.

4.2. In the loop, four variables of type “float” are declared: sensor reading, voltage, Celsius, Fahrenheit.

4.2.1. The sensor reading variable will be determined by the reading of sensor as it sends voltage to the Arduino. The voltage variable is an additional conversion. Finally, the voltage value is translated into a temperature value in Celsius and Fahrenheit.

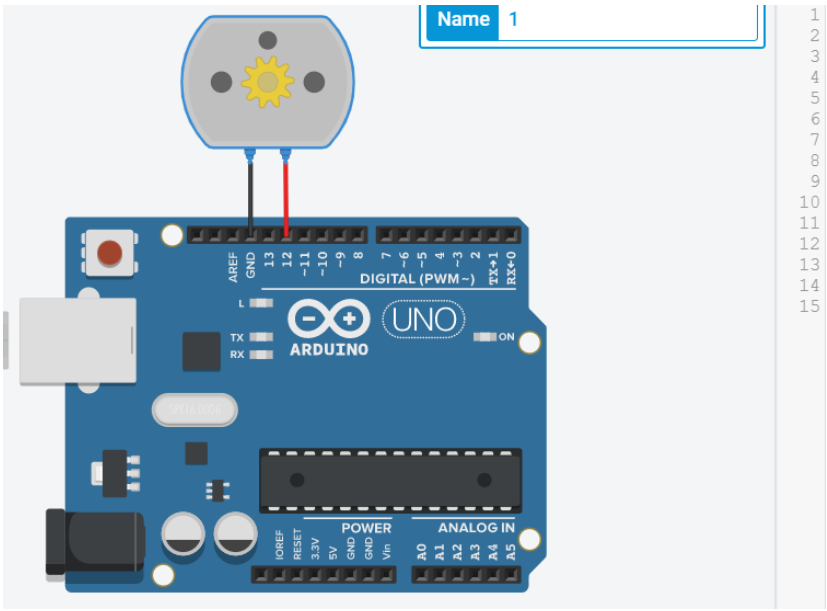
4.3. To print the values onto the screen, Serial.print is used.

4.3.1. Note: Variables and user text must be entered separately, not together in the paranthesis.

4.3.2. Note: Serial.println(); starts a new line of output.

4.4. A delay of 1.5 seconds is given.

5. DC Motor circuit




```
1
2  void setup()
3  {
4    pinMode(12, OUTPUT);
5    Serial.begin(9600);
6
7  }
8
9  void loop()
10 {
11   digitalWrite(12, HIGH);  //run at max speed allowed
12   delay(2000); // Wait for 2 second(s)
13   digitalWrite(12, LOW);
14   delay(2000);
15 }
```

Figure 9: Displays the circuit as well as the program required to read the DC motor.

5.1. Compared to other circuits, this one is quite simple. First, the DC motor must be connected to ground.

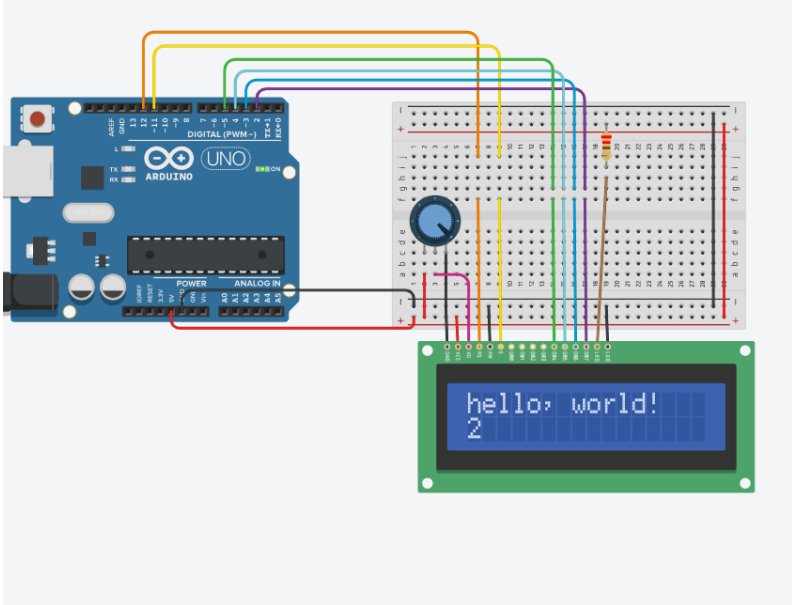
5.2. To control the DC motor, it is connected to digital pin 12.

6. Coding the program:

6.1. Much like the LED, the motor will have power sent to it. In that case, for the code, pin 12 is declared as an output pin.

6.2. digital.Write is used to turn the motor on and off every two seconds (delay)

7. LCD message printer circuit




```
/*
  LiquidCrystal Library - Hello World

 Demonstrates the use a 16x2 LCD display.  The LiquidCrystal
 library works with all LCD displays that are compatible with the
 Hitachi HD44780 driver. There are many of them out there, and you
 can usually tell them by the 16-pin interface.

 This sketch prints "Hello World!" to the LCD
 and shows the time.

  The circuit:
 * LCD RS pin to digital pin 12
 * LCD Enable pin to digital pin 11
 * LCD D4 pin to digital pin 5
 * LCD D5 pin to digital pin 4
 * LCD D6 pin to digital pin 3
 * LCD D7 pin to digital pin 2
 * LCD R/W pin to ground
 * LCD VSS pin to ground
 * LCD VCC pin to 5V
 * 10K resistor:
 * ends to +5V and ground
 * wiper to LCD VO pin (pin 3)

 Library originally added 18 Apr 2008
 by David A. Mellis
 library modified 5 Jul 2009
 by Limor Fried (http://www.ladyada.net)
 example added 9 Jul 2009
 by Tom Igoe
 modified 22 Nov 2010
 by Tom Igoe

 This example code is in the public domain.

 http://www.arduino.cc/en/Tutorial/LiquidCrystal
 */

// include the library code:
#include <LiquidCrystal.h>

// initialize the library with the numbers of the interface pins
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup() {
  // set up the LCD's number of columns and rows:
  lcd.begin(16, 2);
  // Print a message to the LCD.
  lcd.print("hello, world!");
}

void loop() {
  // set the cursor to column 0, line 1
  // (note: line 1 is the second row, since counting begins with 0):
  lcd.setCursor(0, 1);
  // print the number of seconds since reset:
  lcd.print(millis() / 1000);
}
```

Figure 10: Displays the circuit as well as the program required to display a message onto the LCD screen.

7.1. To start off, a bread board should be used to organize the circuit.

7.2. Connect the LCD ground pin to the far-left negative pin. Another wire should be used to connect the breadboard to the Arduino ground pin.

7.3. The positive terminal of the breadboard should be hooked up to 5 Volts from the Arduino. Connect the power pin to the positive terminal of the breadboard.

7.4. A Potentiometer will be used to control the contrast of the board. Connect the device to the breadboard and hook it up to power and ground. The middle pin will be connected contrast pin of the LCD.

7.4.1. Potentiometers work by varying the position of a conductor inside the dial. The voltage sent out the potentiometer can be increased if turned clockwise and vice versa (much like a volume knob)

7.5. The Read/Write pin of the LCD will also be connected to ground as well as the LED cathode.

7.6. The LED anode must be connected to a 220-ohm resistor when connected to 5 Volts.

7.7. The rest pins are interface pins and are connected to the Digital pins of the Arduino as shown above in Figure 10.

8. Coding the program

8.1. A couple of annotations are given by the codes creator to help out with understanding.

8.1.1. Note:Since coding the LCD to behave like it is supposed to would be way too difficult for a beginner, a premade library header is summoned (#include<LiquidCrystal.h>) which contains all the info needed to initialize the LCD.

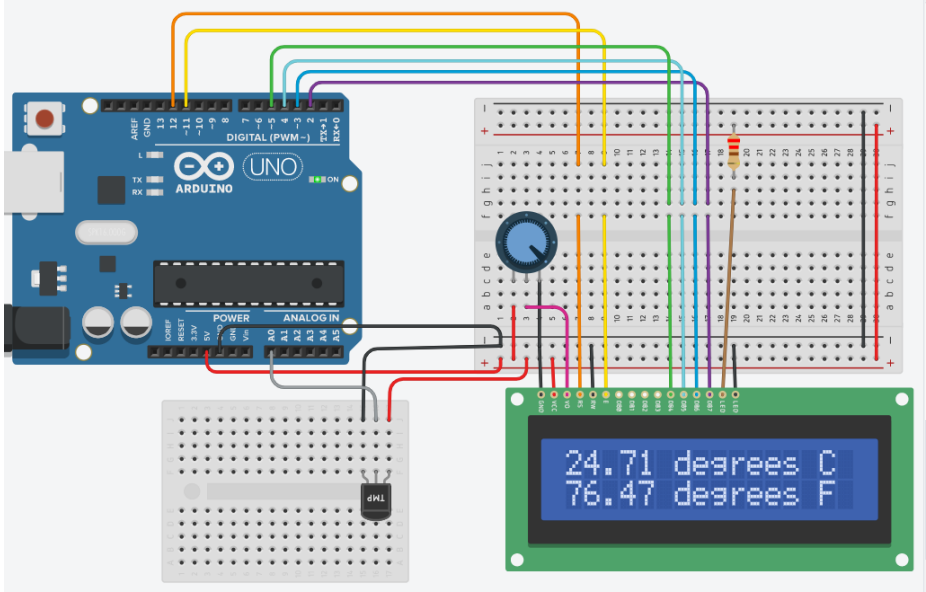
8.2. Before the code begins, the LCD pins are initialized according to what is connected to the digital pins.

8.3. A setup function to setup the LCDs number of columns and rows is given (16 columns and 2 rows)

8.4. A command is given in the setup function to print a message to the screen much like how a message is sent to the serial monitor.

8.5. In the loop function, a runtime counter will be printed to the LCD as well. This will be written on the row of the LCD. The lcd.setCursor(0, 1); command initializes the line that will print that value, as well as the command to print the runtime.

9. LCD Temperature display




```
/*
 Demonstrates the use a 16x2 LCD display.  The LiquidCrystal
 library works with all LCD displays that are compatible with the
 Hitachi HD44780 driver. There are many of them out there, and you
 can usually tell them by the 16-pin interface.

 This sketch prints Temperature to the LCD

 Library originally added 18 Apr 2008
 by David A. Mellis
 library modified 5 Jul 2009
 by Limor Fried (http://www.ladyada.net)
 example added 9 Jul 2009
 by Tom Igoe
 modified 22 Nov 2010
 by Tom Igoe

 Modified by Harjot Gill to print temperature

 This example code is in the public domain. */

// include the library code:
#include <LiquidCrystal.h>

// initialize the library with the numbers of the interface pins
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup() {
  // set up the LCD's number of columns and rows:
  lcd.begin(16, 2);
  pinMode(A0, INPUT); //declaring pin as input data to read
  Serial.begin(9600);
}

void loop() {

  float sensor_reading,voltage,celsius,farenheit;  //declaring variables

  sensor_reading=analogRead(A0);          //reading data from sensor
  voltage= sensor_reading*(5000/1024.0);
  celsius=(voltage-500)/10;             //converting readings to a temp
  farenheit=(celsius * 9.0/5.0) + 32;

 // set the cursor to column 0, line 0
  lcd.setCursor(0, 0);

  lcd.print(celsius); lcd.print(" degrees C");
  // set the cursor to column 0, line 0
  lcd.setCursor(0, 1);                    //print temp values to lcd
 lcd.print(farenheit); lcd.print(" degrees F");
  delay(100);

}
```

Figure 11: Displays the circuit as well as the program required to read the DC motor.

9.1. This circuit is identical to the previous circuit in terms of how the LCD is integrated. Repeat steps 7.1-8.3

9.2. Also, the TMP36 sensor is wired the same way as circuit 3. In that case, 3.1-4.2 should be repeated as well but the TMP sensor should be integrated into a mini breadboard for aesthetic purposes. In fact, a breadboard is optional for the sensor.

9.3. The Code:

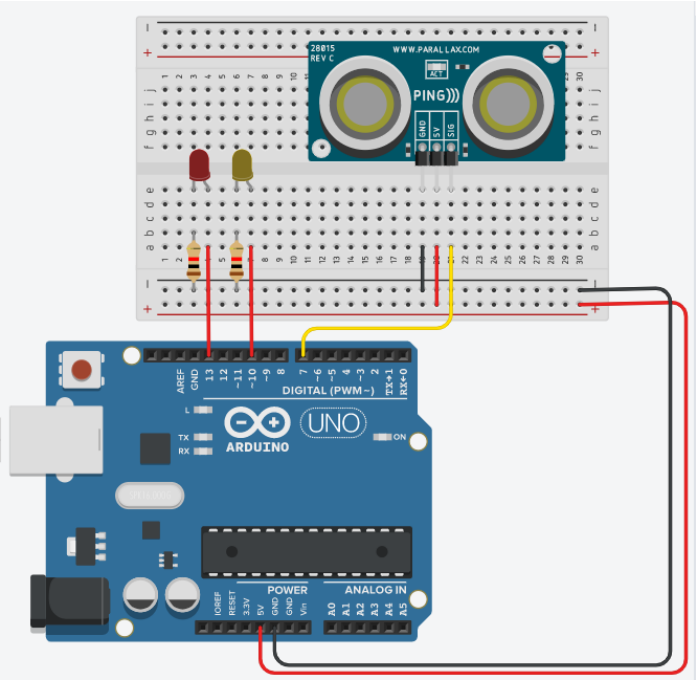
10. To print the temperature values to the cursor must be set to the first line. lcd.setCursor(0,0) sets the cursor to that line. Note that the initial value starts at 0, not 1.

10.1.1. On that line, The temperature in Celsius will be printed using the lcd.print() command

10.2. To move on to the next cursor, lcd.setCursor(0,1) command is given. On that line, the temp in Fahrenheit is printed.

10.3. A tenth of a second delay is given to end the loop

11. Ultrasonic distance sensor LED blink circuit




```
const int max=335; //max sensor distance iin cm
float distance = 0;

long readUltrasonicDistance(int triggerPin, int echoPin)
{
  pinMode(triggerPin, OUTPUT);  // Clear the trigger
  digitalWrite(triggerPin, LOW);
  delayMicroseconds(2);
  // Sets the trigger pin to HIGH state for 10 microseconds
  digitalWrite(triggerPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(triggerPin, LOW);
  pinMode(echoPin, INPUT);
  // Reads the echo pin, and returns the sound wave travel time
  return pulseIn(echoPin, HIGH);
}

void setup()
{
  pinMode(13, OUTPUT);
  pinMode(10, OUTPUT);
}

void loop()
{
  distance = 0.01723 * readUltrasonicDistance(7, 7); //distance

  if (distance >= max*0.75)
 {
  digitalWrite(10, HIGH);
  delay(10); // Wait for 10 millisecond(s)

 }

  else if (distance <= max*0.25)
 {
  digitalWrite(13, HIGH);
  delay(10); // Wait for 10 millisecond(s)

 }

  else
 {
  digitalWrite(13, HIGH); digitalWrite(10, HIGH);
  delay(500); // Wait for 1000 millisecond(s)
  digitalWrite(13, LOW); digitalWrite(10, LOW);
  delay(500); // Wait for 1000 millisecond(s)
 }
}
```

Figure 12: Displays the circuit as well as the program required to read the distance sensor and blink the lights accordingly.

11.1. In terms of wiring the LEDs the procedure 1.1-1.4 can be followed.

11.2. To start with the sensor, connect it to the breadboard connect the power pin to the positive pin of the breadboard and the ground pin to negative terminal.

11.3. Run a wire from the ends of the negative terminal of the breadboard to ground and positive to 5 Volts. A resistor is not needed.

11.4. Finally, the sensor data pin should be hooked up to digital pin 7 (It doesn't matter exactly what number pin you choose).

11.4.1. Note: Digital pins with the tilde symbol (~) on the Arduino signify pins that allow "pulse with modulation".

12. The code:

12.1. the goal is to blink two LED lights depending on the sensor reading of the ultrasonic distance sensor.

12.1.1. If the object is farther than 75% of the sensors range, the yellow LED light will turn on. If the object is 25% of the sensors range (close to the sensor), the red LED will turn on, anything in between and both LEDs will blink.

12.2. A setup function is used to teach the program how to read and integrate the distance sensor into the code. This is in a sense, a prewritten code which contains an instruction manual for the program.

12.3. The two LED pins are declared as output pins.

12.4. In the loop function, the distance in centimeters is calculated based on the sensor reading from the pin it is located at.

12.5. To control which LED flashes and such and when, an if function is used to test a logical condition first. Firstly, if the distance is greater than 75% of the sensors range, the yellow LED will be lit.

12.5.1. Note: Everything that is desired to be done in an "if" situation is enclosed in brackets

12.6. "Else, if" that condition is not met, another condition will be checked. This is whether the distance is less than 25% of the sensors range, the red LED will be lit.

12.7. "Else" if all those conditions are not met, then the system will default to blinking both LEDs at the same time.

DISCUSSION & CONCLUSION

To recap, in this experiment, the goal was to create a variety of circuits utilizing Arduino Uno R3 boards. More specifically, six circuits ranging from blinking an LED light, to operating a DC motor, and writing a message onto an LCD display. Wiring the circuit would require an attention to detail and common knowledge about electricity.

For all the circuits, it is important to protect the components from overload. Therefore, resistors are used in two of the circuits developed. To determine how much resistance should be applied, Ohms Law must be used. Ohms law states that $\frac{\boldsymbol{Voltage\ (V)}}{\boldsymbol{Current\ (amps)}} = \boldsymbol{Resistance}\ (\Omega)$. If the maximum current allowed in a component is known, the proper resistor can be used as well.

To make the circuit work, a code had to be developed using the Arduino IDE software. The code is used to control the system and acts as the brain. The Arduino board can download codes from a computer via USB input. The programming language used is C++ and it can be used to perform a variety of tasks, loops, and calculations. After the code is finished it will be transferred to the Arduino board via a desktop computer into a USB B cable. The Arduino will compile and run the program continuously if the board is hooked up to power.

The circuits created in this session are basic and help build the user's knowledge and experience to build more complex designs in the future. For example, a circuit can be produced to control a motor attached to a bike. This motor can be controlled using a potentiometer of some sort. If the motor requires more power, a 12 Volt external battery can be integrated as well if needed. The possibilities are endless.

One thing to note is the Arduino's limited RAM space. The about 2kb of data used for storing variables and the 32kb of program storage space may present a problem when dealing with multiple components. Personally, a problem that I dealt with in the past revolved around the fact that the Arduino did not have space to accommodate a project code that was developed to control and moving robot. As time goes on, these issues will be solved with evolving chip technology and advancements. As of right now, the Arduino uno is perfect for beginners and should not struggle with memory issues with the programs they will be making.

Another thing to note is the potential of the C++ language. Given that it is a straightforward and intuitive language, many things can be done with it. Loops can be created, functions can be made, and interacting with components using C++ is relatively easy. The only thing to be observant of is the conversions that must be done to convert sensor readings to usable data, like temperature, and distance for example. This is something the manufacturer should have available. If not, it can be found on the internet. Online support and coverage of Arduino projects are plentiful which makes it appealing to start learning.

Sources Cited

Mellis, David, et al. (2010) [Source code] https://www.tinkercad.com

"What Is Arduino UNO." *YouTube*, HackerEarth, 9 Mar. 2017, www.youtube.com/watch?v=_ItSHuIJAJ8.