Geraldez-LK-2019-11336 184-WFU-FX-2 ~ (combinational-logic)

COMBINATIONAL LOGIC

combinational-logic:~# digital systems where the outputs are determined solely by the current states of the inputs to the circuit.

GitHub repo: https://github.com/schwarzschlyle/electronics-and-instrumentation/tree/master/logic-gates

binary-states-convention:~# Throughout the exploration, black-filled LEDs correspond to a HIGH digital state while white-filled LEDs correspond to a LOW digital state

Geraldez-LK-2019-11336 184-WFU-FX-2 ~ (combinational-logic)

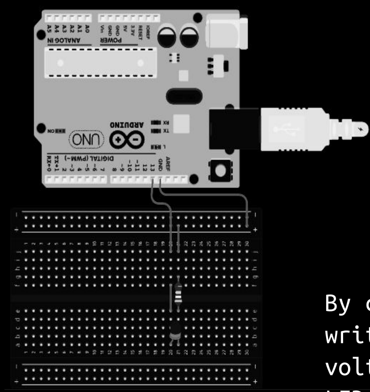
blink-exercise:~# To start the microcontroller learning, we are tasked to construct our first "Hello World" code. That is, we can start by doing the following tasks:

$ construct a blinking LED sketch,

$ adjust the frequency of the blink

$ find out the critical frequency where rapid blinking approaches a steady state

$ construct two alternately blinking LED system



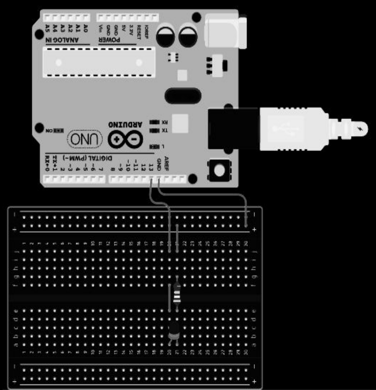
Basic blink

```
-----------------------
int led = 13;
void setup() {
pinMode(led, OUTPUT);
}
void loop() {
digitalWrite(led, HIGH);
delay(1000);
digitalWrite(led, LOW);
delay(1000);
}
-----------------------
```

GitHub repo:

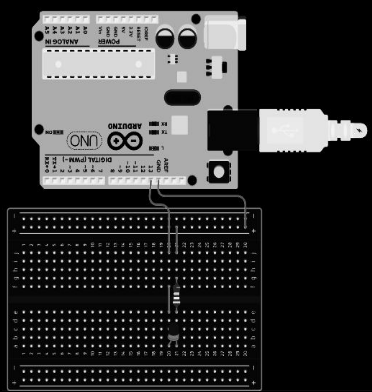
Raw data: https://bit.ly/3ST7GTQ
Data processing: https://bit.ly/3TaGLCP
Plots: https://bit.ly/3q2bVyc
Code repository: https://bit.ly/3ysqFDq

By constructing a simple LED-resistor circuit and writing a code with a delay from HIGH to LOW voltage, we have constructed our first blinking LED system.



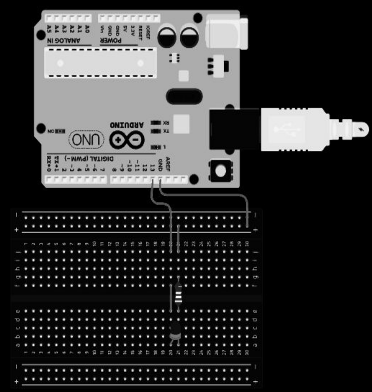
Slower blink

```
-----------------------
…
delay(2000);
…
-----------------------
```



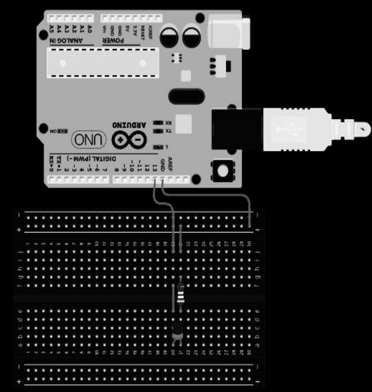
Slower blink (2)

```
-----------------------
…
delay(3000);
…
-----------------------
```



Faster blink

```
-----------------------
…
delay(500);
…
-----------------------
```

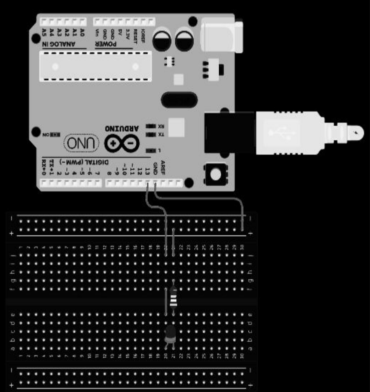


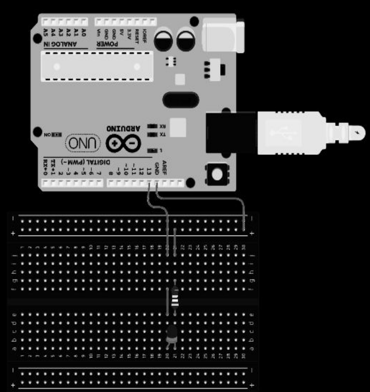
Faster blink (2)

```
-----------------------
…
delay(250);
…
-----------------------
```

One can make the blinking go faster or go slower by adjusting the delay() from HIGH to LOW voltage.

With these, an alternating blinking circuit can be constructed by simultaneously turning an LED to HIGH and another to LOW

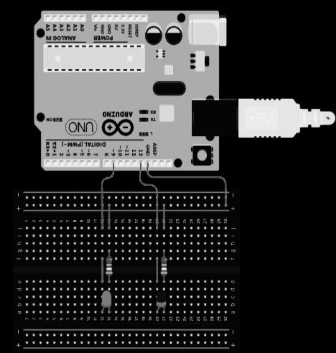




Steady blink

```
-----------------------
…
delay(20); // left
delay(10); // right
…
-----------------------
```

Observe that at delay(20), we can still see the flicker; such flicker converges to a steady state at delay(10)



Alternating Blink

```
-----------------------
int led1 = 13;
int led2 = 9;
void setup() {
pinMode(led1, OUTPUT);
pinMode(led2, OUTPUT);
}
void loop() {
digitalWrite(led1, HIGH);
delay(10);
digitalWrite(led1, LOW);
digitalWrite(led2, HIGH);
delay(10);
digitalWrite(led2, LOW);
}
-----------------------
```

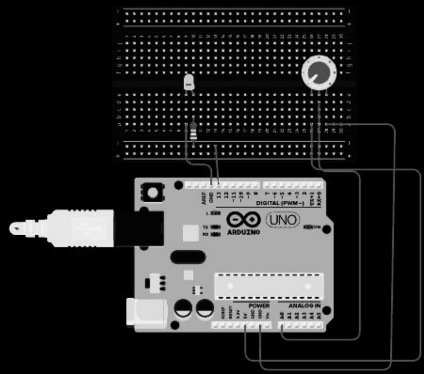
Geraldez-LK-2019-11336 184-WFU-FX-2 ~ (combinational-logic)

blinking-through-analog-input:~# Potentiometers can be used to act like an adjustable voltage dividers. Here, we can explore one of its application to control sensor values by doing the following tasks:

$ adjust LED blinking rate via a potentiometer,

$ display sensor value from the analog input,

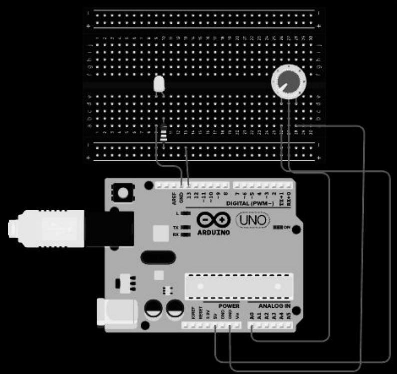
$ display actual voltages value



Blinking Rate via Potentiometer

```
---------------------------------------
...
sensorValue = analogRead(sensorPin);
digitalWrite(ledPin, HIGH);
delay(sensorValue);
digitalWrite(ledPin, LOW);
delay(sensorValue);
...
---------------------------------------
```

Blinking rate can be adjusted manually by incorporating a potentiometer to adjust the analog voltage read from AnalogRead() function. By setting the value to correspond with the delay between HIGH and LOW LED states, blinking rate can be directly controlled by rotating the potentiometer



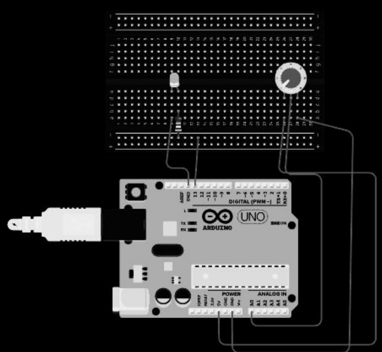
1023

Blinking Rate via potentiometer

```
---------------------------------------
...
void setup()
{
Serial.begin(9600);
…
}

void loop()
{
sensorValue = analogRead(sensorPin);
voltage = sensorValue
Serial.println(voltage);
...
---------------------------------------
```

The serial monitor allows us to visualize the range of analog values being controlled by the potentiometer.

Observe that the values displayed at the serial monitor ranges from 1023 to 0. Programming these values to correspond to the delay() between both digitalWrite() functions, the delays adjusted by this circuit ranges from 1023 milliseconds to 0 millisecond (no delay, static LED)



5

True voltage from analogRead()

```
---------------------------------------
...
sensorValue = analogRead(sensorPin);
voltage = sensorValue * 5.0/1023;
Serial.println(voltage);
...
---------------------------------------
```

Note that the operating voltage of Arduino UNO is 5V. We can scale the displayed sensor values to portray the actual voltages from analogRead().

This scaling gives us an idea that the analogRead() function maps out the voltage range 0V to 5V into corresponding integer values 0 to 1023. As we have seen, one of such applications is to control delay times between switching digital states.

Geraldez-LK-2019-11336 184-WFU-FX-2 ~ (combinational-logic)

blinking-through-analog-input:~# code it up!

codes-used

Blinking Rate via potentiometer

```
int sensorPin = A0;
int ledPin = 13;
int sensorValue = 0;

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

void loop()
{
sensorValue = analogRead(sensorPin); //read the value from the sensor
digitalWrite(ledPin, HIGH);
delay(sensorValue);
digitalWrite(ledPin, LOW);
delay(sensorValue);
}
```

Serial monitor

```
int sensorPin = A0;
int ledPin = 13;
int sensorValue = 0;
int voltage = 0;

void setup()
{
Serial.begin(9600);
pinMode(ledPin, OUTPUT);
}

void loop()
{
sensorValue = analogRead(sensorPin); // read the value from the sensor
voltage = sensorValue * 5.0/1023; // scaling to display true voltage value
Serial.println(voltage);
digitalWrite(ledPin, HIGH);
delay(sensorValue);
digitalWrite(ledPin, LOW);
delay(sensorValue);
}
```

Geraldez-LK-2019-11336 184-WFU-FX-2 ~ (combinational-logic)

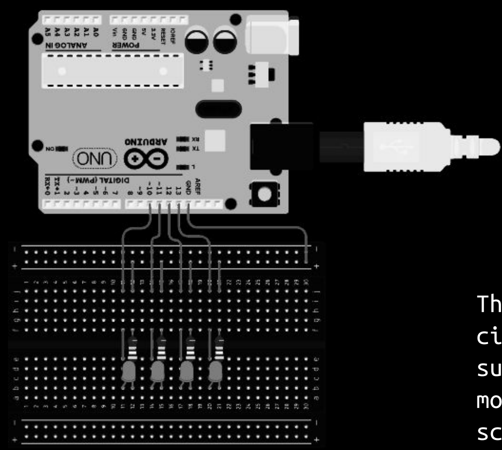
scrolling-led:~# By programming conditional statements on Arduino microcontroller one can create visual effects such as the illusion of a scrolling LED; these can be sometimes be seen on premium keyboards with RGB features. To get our hands dirty with conditional statements, we aim to do the following tasks:

$ construct a scrolling LED circuit,

$ demonstrate the utility of the reset button,

$ increase the rate of scrolling

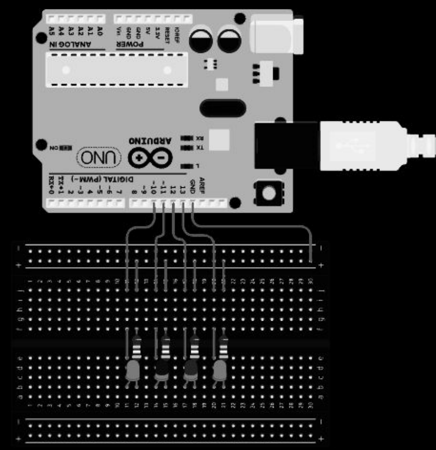
$ control the rate of scrolling via a potentiometer

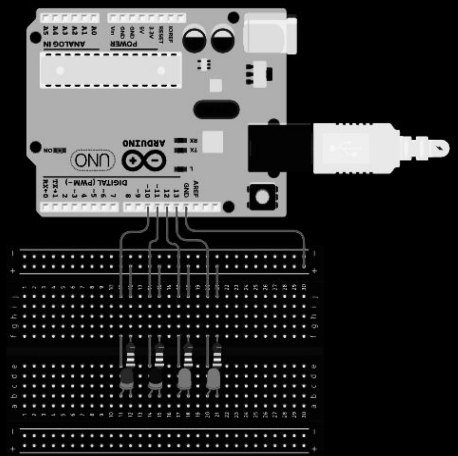


Scrolling LED

```
…
digitalWrite(led1, HIGH);
delay(500);
digitalWrite(led1, LOW);
digitalWrite(led2, HIGH);
delay(500);
digitalWrite(led2, LOW);
digitalWrite(led3, HIGH);
…
```

The alternating blink circuit can be modified such that blinking is modified into a scrolling behavior.

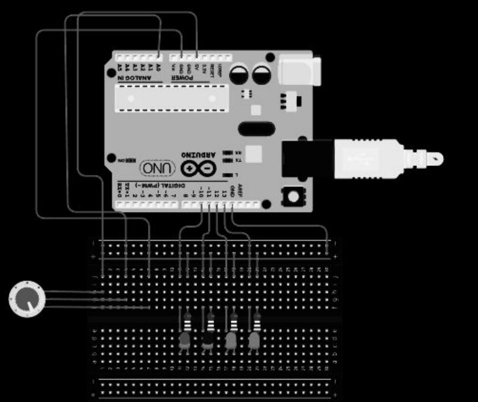




Incrementally increasing scrolling rate

```
…
digitalWrite(led1, HIGH);
delay(init_del - inc);
digitalWrite(led1, LOW);
digitalWrite(led2, HIGH);
…
inc = inc + inc_change;
…
```

One can write up a code where, at each looping instance, the delay is being increased (or decreased) by a constant value. This gives the scrolling-LED circuit to have an increasing rate after each full scrolling cycle.



Potentiometer-adjusted scrolling rate

```
…
digitalWrite(led1, HIGH);
delay(1050-sensorValue);
digitalWrite(led1, LOW);
digitalWrite(led2, HIGH);
…
```

The rate of scrolling can be further modified such the the constant change after each scrolling cycle is replaced by a variable rate dictated by the a potentiometer. This way, one can control the rate of LED scrolling as necessary.

Geraldez-LK-2019-11336 184-WFU-FX-2 ~ (combinational-logic)

scrolling-led:~# code it up!

codes-used

Incrementally increasing scrolling rate

```
int led1 = 13;
int led2 = 12;
int led3 = 11;
int led4 = 10;
int inc = 0;
int inc_change = 100;
int init_del = 500;

void setup() {
pinMode(led1, OUTPUT);
pinMode(led2, OUTPUT);
pinMode(led3, OUTPUT);
pinMode(led4, OUTPUT);
}

void loop() {
// left to right
digitalWrite(led1, HIGH);
delay(init_del - inc);
digitalWrite(led1, LOW);
digitalWrite(led2, HIGH);
delay(init_del - inc);
digitalWrite(led2, LOW);
digitalWrite(led3, HIGH);
delay(init_del - inc);
digitalWrite(led3, LOW);
digitalWrite(led4, HIGH);
delay(init_del - inc);
digitalWrite(led4, LOW);
// right to left
digitalWrite(led3, HIGH);
delay(init_del - inc);
digitalWrite(led3, LOW);
digitalWrite(led2, HIGH);
delay(init_del - inc);
digitalWrite(led2, LOW);
digitalWrite(led1, HIGH);
inc = inc + inc_change;
}
```

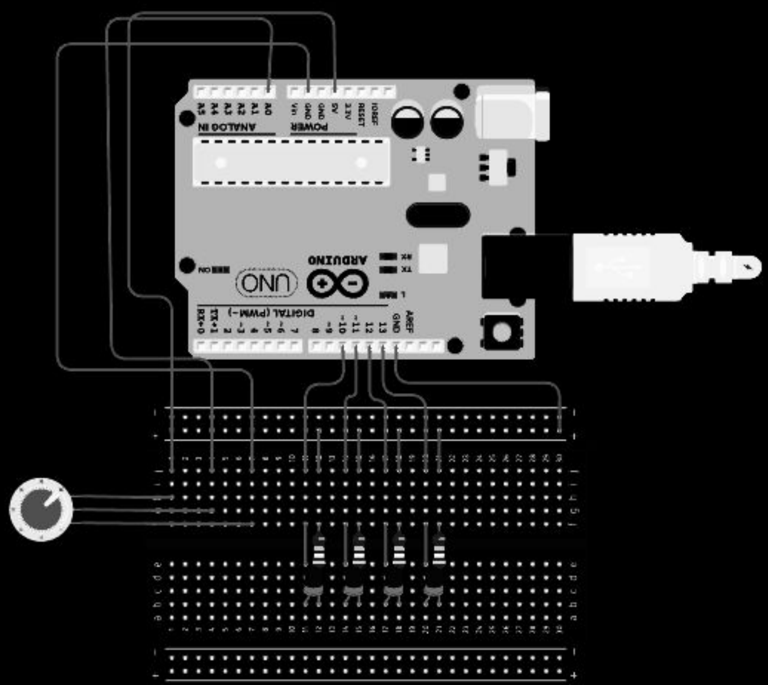
Potentiometer-adjusted scrolling rate

```
int sensorPin = A0;
int led1 = 13;
int led2 = 12;
int led3 = 11;
int led4 = 10;
int sensorValue = 0;

void setup() {
Serial.begin(9600);
pinMode(led1, OUTPUT);
pinMode(led2, OUTPUT);
pinMode(led3, OUTPUT);
pinMode(led4, OUTPUT);
}

void loop() {
sensorValue = analogRead(sensorPin);
Serial.println(sensorValue);
// left to right
digitalWrite(led1, HIGH);
delay(1050-sensorValue);
digitalWrite(led1, LOW);
digitalWrite(led2, HIGH);
delay(1050-sensorValue);
digitalWrite(led2, LOW);
digitalWrite(led3, HIGH);
delay(1050-sensorValue);
digitalWrite(led3, LOW);
digitalWrite(led4, HIGH);
delay(1050-sensorValue);
digitalWrite(led4, LOW);
// right to left
digitalWrite(led3, HIGH);
delay(1050-sensorValue);
digitalWrite(led3, LOW);
digitalWrite(led2, HIGH);
delay(1050-sensorValue);
digitalWrite(led2, LOW);
digitalWrite(led1, HIGH);
}
```

Geraldez-LK-2019-11336 184-WFU-FX-2 ~ (combinational-logic)

pot-multiple-leds:~# As a natural extension, it would be interesting to write some conditional statements the depend on some adjustable input parameters (such as values given by a potentiometer). Here, we aim to do the following:

$ construct an LED-based progress bar graph following the levels of the potentiometer as input



LED Progress Bar

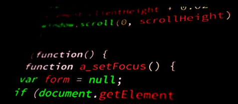
```
----------------------
…
if (sensorValue < 256)
{...}
if (sensorValue > 256)
{...}
if (sensorValue < 512)
{...}
if (sensorValue < 768)
{...}
if (sensorValue < 1020)
{...]
…
----------------------
```

Conditional statements can be written to the code referring to different circuit states. Here, five such states were programmed depending on the input values of adjusted by the potentiometer. Dividing the analog range of 0 to 1023 into five different states, we have the ranges 0 to 256, 257 to 512, 513 to 768, and 769 to 1020.

The respective five different states correspond the progress bar level of the LED array. Respectively, the states are: (i) 0 LED activated, (ii) 1 LED activated ,..., (v) ALL LEDs activated.

Geraldez-LK-2019-11336 184-WFU-FX-2 ~ (combinational-logic)

pot-multiple-leds:~#



codes-used

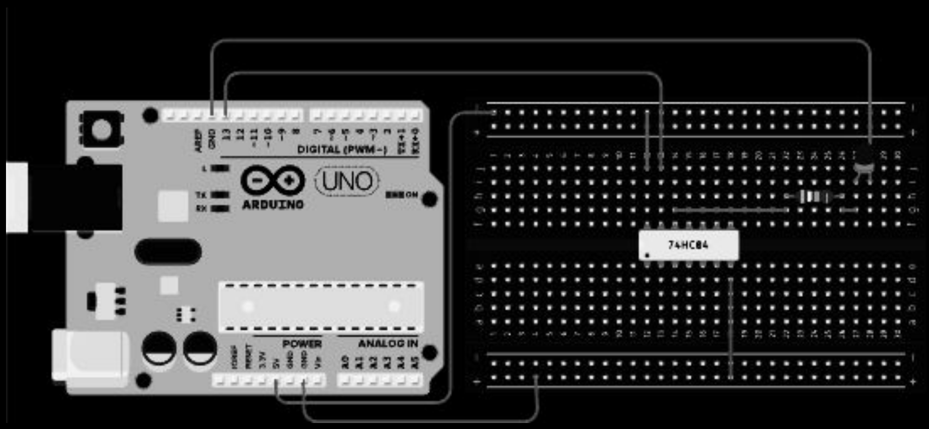
LED Progress Bar

```
int sensorPin = A0;
int led1 = 13;
int led2 = 12;
int led3 = 11;
int led4 = 10;
int inc = 0;
int inc_change = 100;
int init_del = 500;
int sensorValue = 0;

void setup() {
Serial.begin(9600);
pinMode(led1, OUTPUT);
pinMode(led2, OUTPUT);
pinMode(led3, OUTPUT);
pinMode(led4, OUTPUT);
}
void loop() {
sensorValue = analogRead(sensorPin);
Serial.println(sensorValue);

if (sensorValue < 256) {
  digitalWrite(led1, LOW);
  digitalWrite(led2, LOW);
  digitalWrite(led3, LOW);
  digitalWrite(led4, LOW);
}

if (sensorValue > 256) {
  digitalWrite(led1, LOW);
  digitalWrite(led2, LOW);
  digitalWrite(led3, LOW);
  digitalWrite(led4, LOW);
  digitalWrite(led1, HIGH);
}

if (sensorValue > 512) {
  digitalWrite(led1, LOW);
  digitalWrite(led2, LOW);
  digitalWrite(led3, LOW);
  digitalWrite(led4, LOW);
  digitalWrite(led1, HIGH);
  digitalWrite(led2, HIGH);
}


if (sensorValue > 768) {
  digitalWrite(led1, LOW);
  digitalWrite(led2, LOW);
  digitalWrite(led3, LOW);
  digitalWrite(led4, LOW);
  digitalWrite(led1, HIGH);
  digitalWrite(led2, HIGH);
  digitalWrite(led3, HIGH);
}


if (sensorValue > 1020) {
  digitalWrite(led1, LOW);
  digitalWrite(led2, LOW);
  digitalWrite(led3, LOW);
  digitalWrite(led4, LOW);
  digitalWrite(led1, HIGH);
  digitalWrite(led2, HIGH);
  digitalWrite(led3, HIGH);
  digitalWrite(led4, HIGH);
}


}
```

Geraldez-LK-2019-11336 184-WFU-FX-2 ~ (combinational-logic)

logic-gate-1:~# Now, we enter the area of logic gates - the main element to design a circuit based on some carefully-designed logic conditions. One such gate is the NOT gate. Given a binary state (1) or (0), the output must be the inverse of the input (0) or (1), respectively. To see this, aim to do the following:

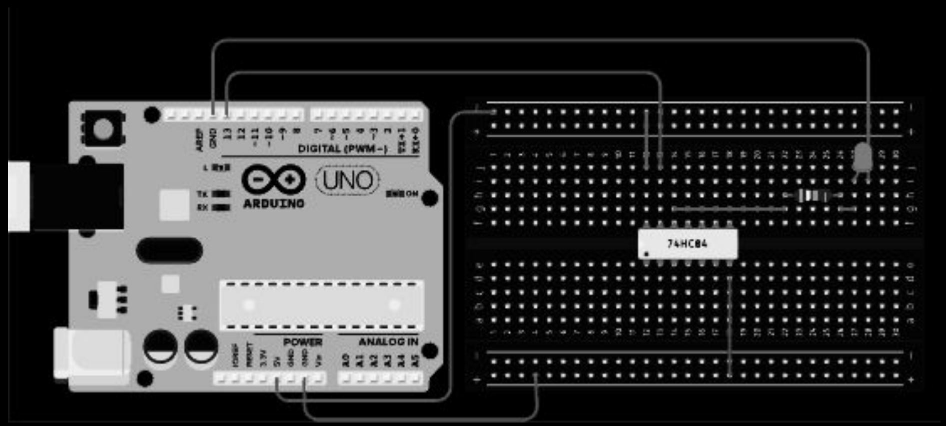
$ verify and demonstrate the validity of the NOT gate truth values



NOT gate LED states test I

```
--------------------------
…
digitalWrite(ledPin, LOW);
delay(2000);
digitalWrite(ledPin, HIGH);
delay(300);
…
--------------------------
```

To test the validity of the truth values of the constructed NOT gates, we employ to equivalent tests. In the first test, we set the input to LOW for 2000 milliseconds and to HIGH for 300 milliseconds. The resulting LED must have an inverted output: 2000 milliseconds HIGH state and 300 milliseconds LOW state.



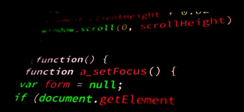
NOT gate LED states test II

```
--------------------------
…
digitalWrite(ledPin, HIGH);
delay(2000);
digitalWrite(ledPin, LOW);
delay(300);
…
--------------------------
```

Another test shows that inverting the input also inverts the output. In both cases, truth values that of a NOT gate.

Geraldez-LK-2019-11336 184-WFU-FX-2 ~ (combinational-logic)

logic-gate-1:~# code it up!



codes-used

NOT gate LED states test

```
--------------------------------------------------------------------------------
int ledPin = 13;
int sensorValue = 0;

void setup(){
pinMode(ledPin, OUTPUT);
}
void loop(){
digitalWrite(ledPin, LOW);
delay(2000);
digitalWrite(ledPin, HIGH);
delay(300);
}
--------------------------------------------------------------------------------
```

Geraldez-LK-2019-11336 184-WFU-FX-2 ~ (combinational-logic)

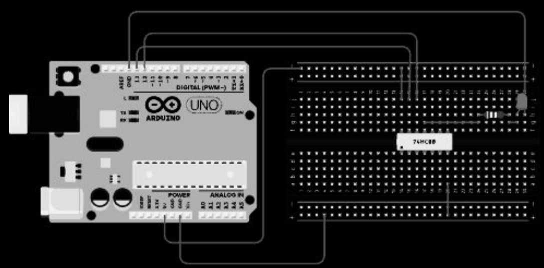
logic-gate-2:~# Several ICs contain different logic gates. We have already discussed the NOT gate using 7404). Here, we explore other kinds of logic gates using 7400, 7408, and 7432 by doing the following tasks:

$ design, verify, and demonstrate the validity of the AND gate truth values

$ design, verify, and demonstrate the validity of the NAND gate truth values

$ design, verify, and demonstrate the validity of the OR gate truth values

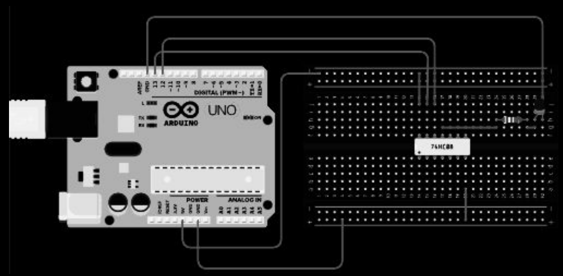
The testing codes were programmed such that it cycles through all binary states in the following order (11) > (00) > (01) > (10). It was set up such that the first state (11) runs for 2000 milliseconds and while all other states run for 667 milliseconds



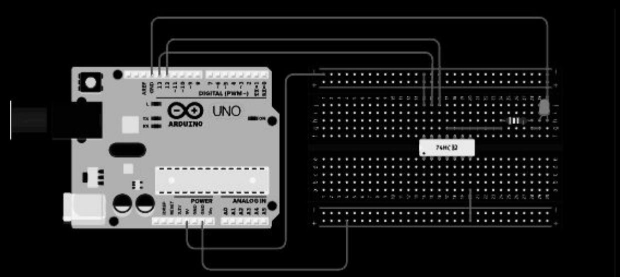
The first circuit (7400) shows an initial 2 millisecond HIGH state followed by another 2 millisecond LOW state. This corresponds to an AND gate.

AND, NAND, and OR LED states test

```
------------------------------
…
digitalWrite(ledPin1, HIGH);
digitalWrite(ledPin2, HIGH);
delay(2000);
digitalWrite(ledPin1, LOW);
digitalWrite(ledPin2, LOW);
delay(667);
digitalWrite(ledPin1, LOW);
digitalWrite(ledPin2, HIGH);
delay(667);
digitalWrite(ledPin1, HIGH);
digitalWrite(ledPin2, LOW);
delay(667);

…
------------------------------
```

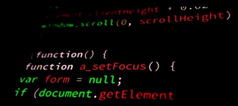


The second circuit (7408) corresponds to a 2 millisecond LOW state followed by another 2 millisecond HIGH state. This is simply an inverted AND gate, or a NAND gate.



The third circuit shows (7432) is active 3/4 of the times during the entire testing cycle. Upon careful observation, it goes into a LOW state at the second state. Hence, this is an OR gate.

Geraldez-LK-2019-11336 184-WFU-FX-2 ~ (combinational-logic)



codes-used

logic-gate-2:~# code it up!

AND, NAND, and OR LED states test

```
int ledPin1 = 13;
int ledPin2 = 12;
int sensorValue = 0;

void setup(){
pinMode(ledPin1, OUTPUT);
pinMode(ledPin2, OUTPUT);
}
void loop(){
digitalWrite(ledPin1, HIGH);
digitalWrite(ledPin2, HIGH);
delay(2000);
digitalWrite(ledPin1, LOW);
digitalWrite(ledPin2, LOW);
delay(667);
digitalWrite(ledPin1, LOW);
digitalWrite(ledPin2, HIGH);
delay(667);
digitalWrite(ledPin1, HIGH);
digitalWrite(ledPin2, LOW);
delay(667);
}
```

Geraldez-LK-2019-11336 184-WFU-FX-2 ~ (combinational-logic)

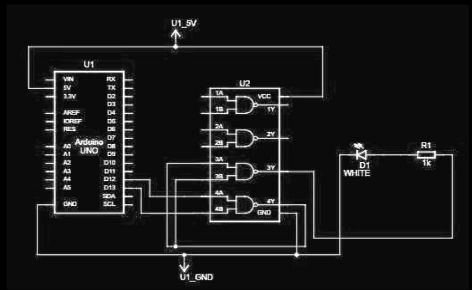
logic-gate-3:~# Some logic gates are considered to be universal. That is, one can design an appropriate circuit to repurpose an universal gate into any kind of logic gate (OR, AND, NOT, etc). A NAND gate is one example of a universal logic gate. To see this, we aim to do the following:

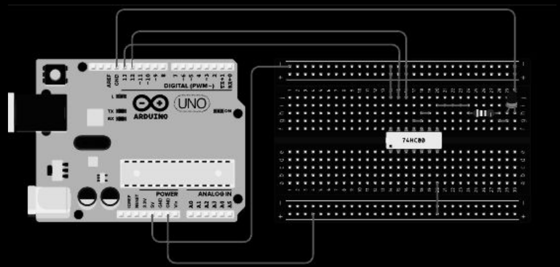
$ construct a circuit to repurpose NAND universal gate as an OR gate

$ construct a circuit to repurpose NAND universal gate as an AND gate

$ construct a circuit to repurpose NAND universal gate as a NOT gate

To test the validity of the constructed NAND-based logic gates, we employ different tests by cycling through all states. The tests were designed with different delay times such that the respective states can be uniquely identified depending on how long it was activated.

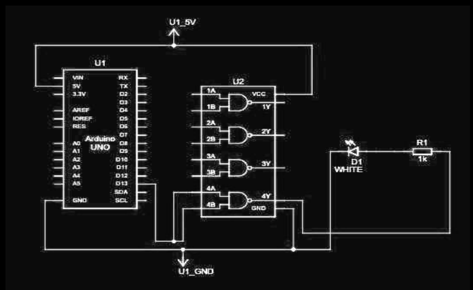


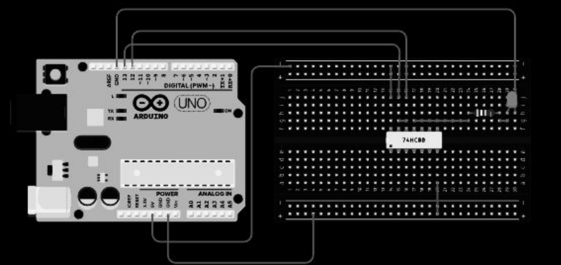


NAND as AND test

```
…
digitalWrite(ledPin1, HIGH);
digitalWrite(ledPin2, HIGH);
delay(2000);
digitalWrite(ledPin1, LOW);
digitalWrite(ledPin2, LOW);
delay(667);
digitalWrite(ledPin1, LOW);
digitalWrite(ledPin2, HIGH);
delay(667);
digitalWrite(ledPin1, HIGH);
digitalWrite(ledPin2, LOW);
delay(667);
…
```

Circuit shows that, half of the time, LED enters a HIGH state. From the written code, 2000 millisecond is spend on a (11) state input. This is characteristic of an AND gate logic.

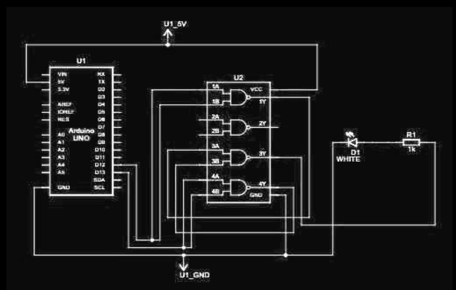


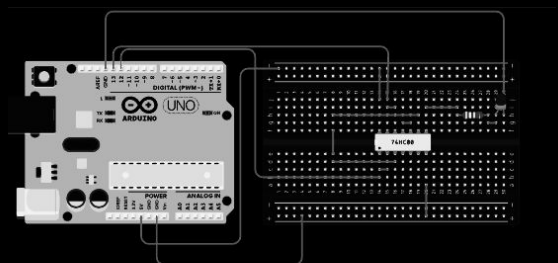


NAND as NOT test

```
…
digitalWrite(ledPin, LOW);
delay(2000);
digitalWrite(ledPin, HIGH);
delay(300);
…
```

Spending 2000 milliseconds on LOW input state and 300 milliseconds on HIGH input state gives us an inverted output state. This is characteristic of a NOT gate logic.





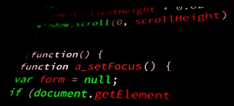
NAND as OR test

```
…
digitalWrite(ledPin1, HIGH);
digitalWrite(ledPin2, HIGH);
delay(2000);
digitalWrite(ledPin1, LOW);
digitalWrite(ledPin2, LOW);
delay(667);
digitalWrite(ledPin1, LOW);
digitalWrite(ledPin2, HIGH);
delay(667);
digitalWrite(ledPin1, HIGH);
digitalWrite(ledPin2, LOW);
delay(667);
…
```

Circuit enters only the LOW state for 667 milliseconds or 1/4 of the cycle entering. Looking at the test code, this corresponds to a (00) state input - characteristic of an OR gate logic.

Geraldez-LK-2019-11336 184-WFU-FX-2 ~ (combinational-logic)

logic-gate-3:~# code it up!



codes-used

NAND as NOT test

```
--------------------------------
int ledPin = 13;
int sensorValue = 0;

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

void loop(){
digitalWrite(ledPin, LOW);
delay(2000);
digitalWrite(ledPin, HIGH);
delay(300);
}
--------------------------------
```

NAND as AND test

```
--------------------------------
int ledPin1 = 13;
int ledPin2 = 12;
int sensorValue = 0;

void setup(){
pinMode(ledPin1, OUTPUT);
pinMode(ledPin2, OUTPUT);
}

void loop(){
digitalWrite(ledPin1, HIGH);
digitalWrite(ledPin2, HIGH);
delay(2000);
digitalWrite(ledPin1, LOW);
digitalWrite(ledPin2, LOW);
delay(667);
digitalWrite(ledPin1, LOW);
digitalWrite(ledPin2, HIGH);
delay(667);
digitalWrite(ledPin1, HIGH);
digitalWrite(ledPin2, LOW);
delay(667);

--------------------------------
```

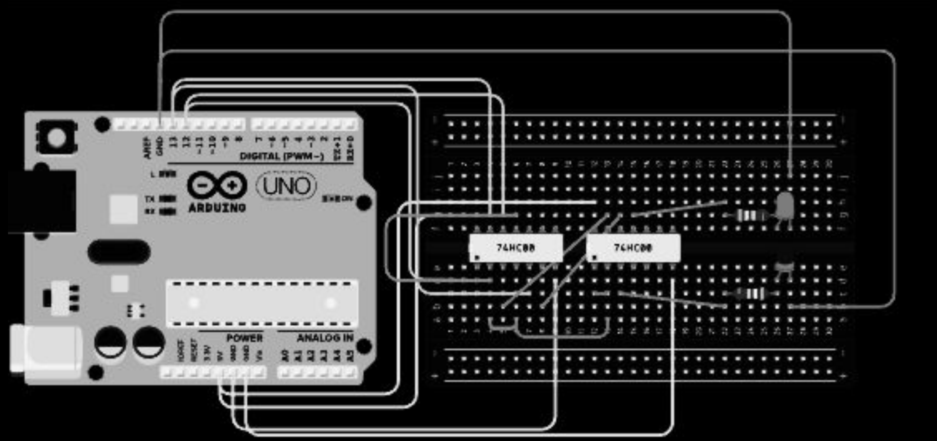
NAND as OR test

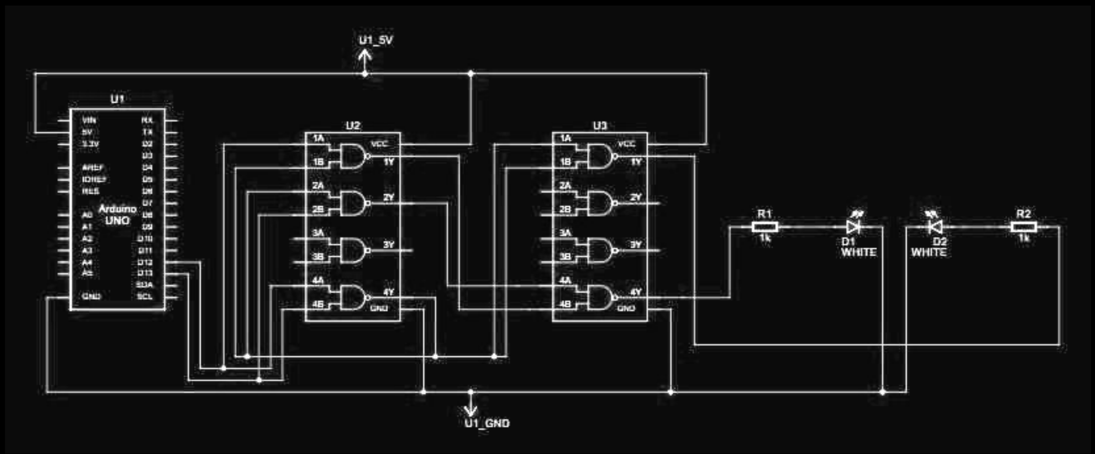
```
--------------------------------
int ledPin1 = 13;
int ledPin2 = 12;
int sensorValue = 0;

void setup(){
pinMode(ledPin1, OUTPUT);
pinMode(ledPin2, OUTPUT);
}

void loop(){
digitalWrite(ledPin1, HIGH);
digitalWrite(ledPin2, HIGH);
delay(2000);
digitalWrite(ledPin1, LOW);
digitalWrite(ledPin2, LOW);
delay(667);
digitalWrite(ledPin1, LOW);
digitalWrite(ledPin2, HIGH);
delay(667);
digitalWrite(ledPin1, HIGH);
digitalWrite(ledPin2, LOW);
delay(667);
}

--------------------------------
```

Geraldez-LK-2019-11336 184-WFU-FX-2 ~ (combinational-logic)

half-adder:~# We an appropriate circuit design, one can perform various arithmetic operations as desired. One of the simplets operations is to add two 1-bit numerical binary values. These can be easily carried out by a half-adder circuit. To see this, we aim to do the following:

$ design a half-adder circuit using NAND gate and confirm its truth statements validity.





Half-adder circuit using NAND gate

```
…
digitalWrite(ledPin1, HIGH);
digitalWrite(ledPin2, HIGH);
delay(2000);
digitalWrite(ledPin1, HIGH);
digitalWrite(ledPin2, LOW);
delay(2000);
digitalWrite(ledPin1, LOW);
digitalWrite(ledPin2, LOW);
delay(2000);
digitalWrite(ledPin1, LOW);
digitalWrite(ledPin2, HIGH);
delay(2000);
…
```

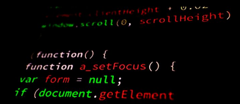
The half-adder circuit is constructed based on the following binary arithmetic logic displayed the following truth table.

Input 1	Input 2	Sum	Carry
1	1	0	1
1	0	1	0
0	0	0	0
0	1	1	0

Observe the results of the written testing code, the input cycles through (11) > (10) > (00) > (01). This results to the following (Sum, Carry) states, respectively, as (01), (10), (00), (10), which is consistent with a half-adder truth table.

Geraldez-LK-2019-11336 184-WFU-FX-2 ~ (combinational-logic)

half-adder:~# code it up!



codes-used

Half-adder circuit using NAND gate

```
int ledPin1 = 13;
int ledPin2 = 12;
int sensorValue = 0;

void setup(){
pinMode(ledPin1, OUTPUT);
pinMode(ledPin2, OUTPUT);
}
void loop(){
digitalWrite(ledPin1, HIGH);
digitalWrite(ledPin2, HIGH);
delay(2000);
digitalWrite(ledPin1, HIGH);
digitalWrite(ledPin2, LOW);
delay(2000);
digitalWrite(ledPin1, LOW);
digitalWrite(ledPin2, LOW);
delay(2000);
digitalWrite(ledPin1, LOW);
digitalWrite(ledPin2, HIGH);
delay(2000);
}
```
