Engineering Challenge: Creating a Traffic Light Sequence with the Arduino Uno Microcontroller

Learning outcomes

1. Learn how to set-up and load programs (“sketches”) onto the Arduino UNO
2. Learn how to code simple Arduino programs in C
3. Learn how to prototype simple electrical/electronic circuits on breadboards
4. Learn how to troubleshoot circuits and code
5. Experience the full Process-Sense-Compute-Decide-Act cycle in action.

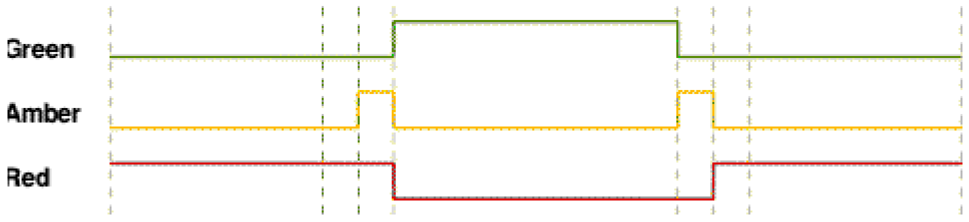
Equipment

1 x Arduino Uno & USB cable
3 x LED
3 x 470 Ohm resistors
1 x 1 kOhm resistor
Switch
PC with Arduino Integrated Development Environment (IDE)

Background

See Introduction to Arduino lecture notes on KEATS module page

A “traffic light sequence” is a series of lights that turn on and off in sequence either automatically or in response to a request. The general scheme can be represented as:



State 1	State 2	State 3	State 4
First 6 Seconds	**Next 2 Seconds**	**Next 8 Seconds**	**Next 2 Seconds**
Red Light ON Amber Light OFF Green Light OFF	Red Light ON Amber Light ON Green Light OFF	Red Light OFF Amber Light OFF Green Light ON	Red Light OF Amber Light ON Green Light OFF

Setting up a basic (automatic) traffic light sequence on the Arduino introduces the student to using the OUTPUT function of the DIGITAL microcontroller pins, where sending the pin HIGH represents turning the LED on, and sending it LOW, represents turning the LED off. Adding to this a switch introduces the idea of reacting to an input. The full traffic light sequence with switch or button control represents the full **Process-Sense-Compute-Decide-Act** cycle in action.

Method

1. Setting up the Arduino and confirming it is working
2. Build the first LED onto the breadboard and confirm this behaves as expected
3. Complete the basic traffic light sequence circuit
4. Modify the "blink" program to produce the basic traffic light sequence program, upload and run this

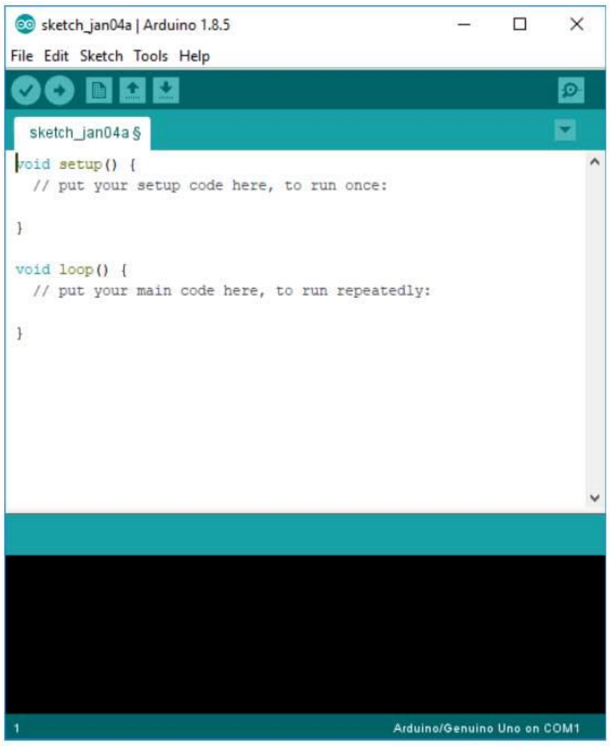
Additional

5. Add a switch for a request-controlled traffic light sequence

1. Setting up the Arduino and confirming it is working

The first step involves finding and starting the Arduino IDE program on the PC. You will probably have to navigate to the Arduino folder in the program files (x86) folder on the C disk to do this. The image on the left shows the Arduino IDE icon.

.



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

}

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

}
```

```
1                                    Arduino/Genuino Uno on COM1
```

When you first launch the IDE, you should see this window.

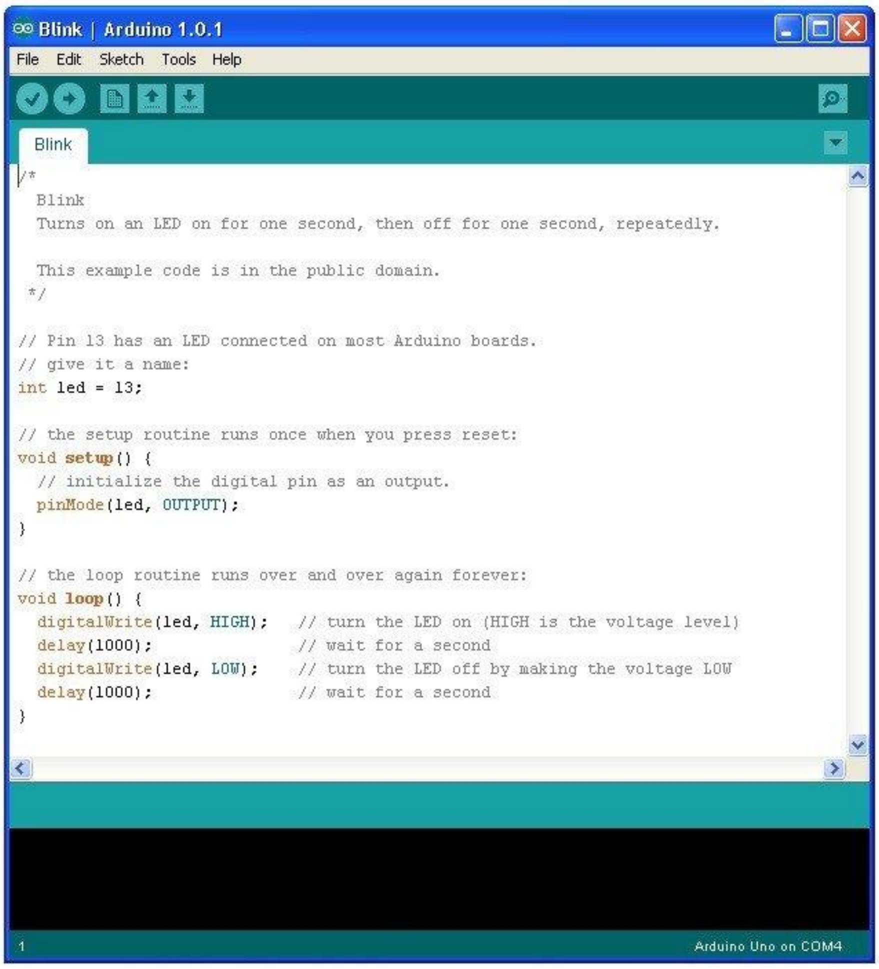
The next step is to connect the Arduino UNO to the PC via the USB cable and then confirm that the program and the board are communicating with each other.

After you have connected the board, navigate to **Tools > Port >** and click on the entry that shows the "Arduino/Genuine Uno" entry (e.g. COM4: (Arduino/Genuine Uno)).

The next step is to upload a program to the board and confirm the board is working.

Navigate to **File > Examples > 01.Basics >** and click on "Blink". This should open a new window showing the Blink sketch (program).

The Blink sketch



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

  This example code is in the public domain.
 */

// Pin 13 has an LED connected on most Arduino boards.
// give it a name:
int led = 13;

// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  pinMode(led, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
  digitalWrite(led, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);               // wait for a second
  digitalWrite(led, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);               // wait for a second
}
```

Different versions of the software have different versions of this sketch, but, fundamentally it can be seen that the sketch sets up a digital pin (13) to be an output pin then sends this HIGH for 1s and then LOW for 1s, turning on and off an on-board LED. The delay command is in milliseconds (ms), 1000 ms = 1 second.

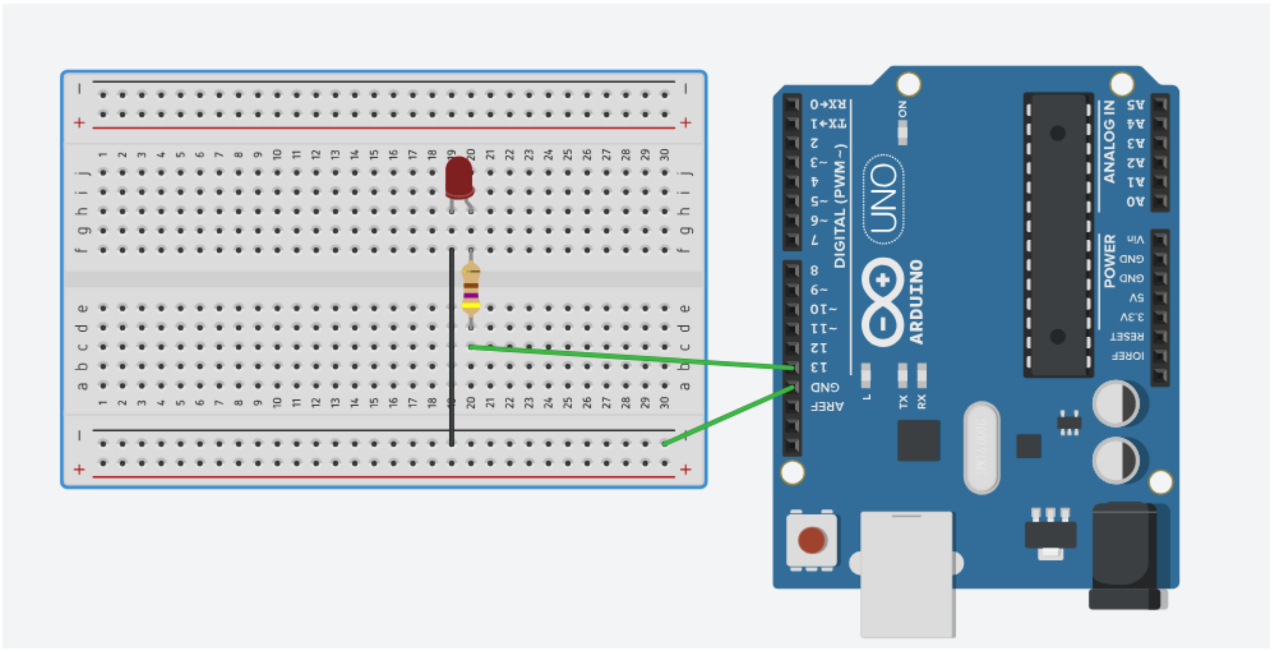
To upload this sketch to the Arduino you can either 1) navigate to **Sketch > Upload**; or 2) click on the right arrow in the blue panel above the program window.

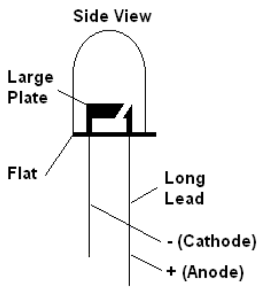
<u>If you have been successful you should now see a light on the Arduino board blinking on and off in one second intervals.</u>

2. *Build the first LED onto the breadboard and confirm this behaves as expected*

NOTE THE COLOUR OF THE WIRES YOU USE IS NOT CRITICAL, BUT YOU WILL FIND IT USEFUL WHEN BUILDING MORE COMPLICATED CIRCUITS TO FOLLOW CONVENTIONS – BLACK FOR WIRES TO GROUND, RED FOR WIRES TO SOURCES OF POWER/VOLTAGE FOR EXAMPLE

The diagram below shows the set-up for building an LED and current-limiting resistor onto the supplied breadboard and then using wires to connect this LED to the Arduino





The LEDs have one long leg and one short leg. The short leg is always connected to the ground (“GND”). In our breadboard this connection is made with a wire via the black-lined horizontal track on the breadboard – the reason for this will become obvious later. The long leg is connected to the signal that will come from, in this instance, the socket on the Arduino board marked “13”. However, this is not a direct connection but goes via a resistor that is placed across the “gutter” in the breadboard.

This resistor is there to limit the **current** that is supplied to the LED – too much current and the LED can be damaged.

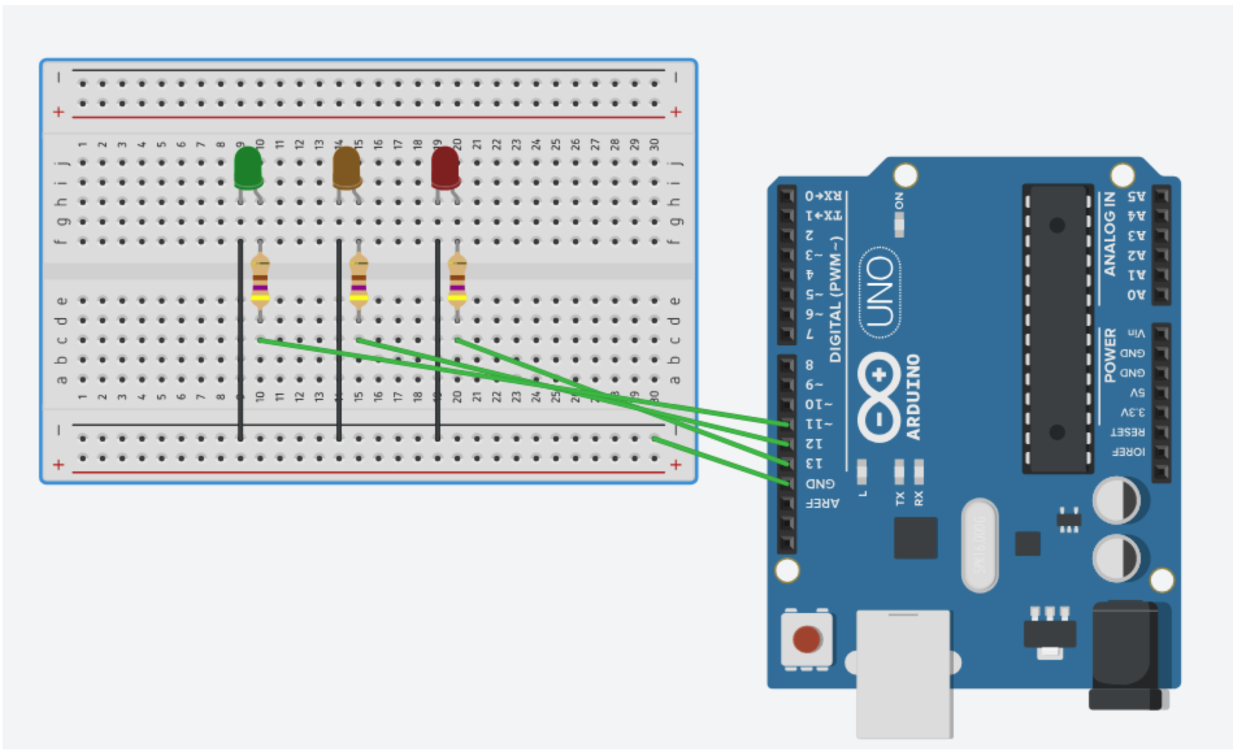
*Can you, using Ohm's Law (**Voltage = Current x Resistance**) confirm that by using a 470 Ohm resistor, we are keeping the maximum current the LED is receiving below the limit of 20 mA?*

For this calculation, you will need to know that the voltage across the LED is 2.2 V and therefore the voltage across the resistor is (5 – 2.2) = 2.8 V (i.e. the voltage supplied by the Arduino pin (5 V) minus the voltage across the LED)

<u>If you have been successful you should now see the LED blinking on and off in one second intervals.</u>

3. *Complete the basic traffic light sequence circuit*

The rest of the basic circuit is simply a "copy-paste" of the first LED placement – now you can see why we are using the black-lined track as our GND; this allows us to connect multiple components to the same GND pin/socket.



The new LEDs are connected to pin 12 (amber) and pin 11 (green)

4. *Modify the "blink" program to produce the basic traffic light sequence program, upload and run this*

Modify the blink program, save as "traffic light 001", and then upload (Sketch > Upload or the right arrow)

Tip: remember at each given moment, some lights are on, some lights are off, but <u>all lights exist in each of the "traffic lights" four states</u>

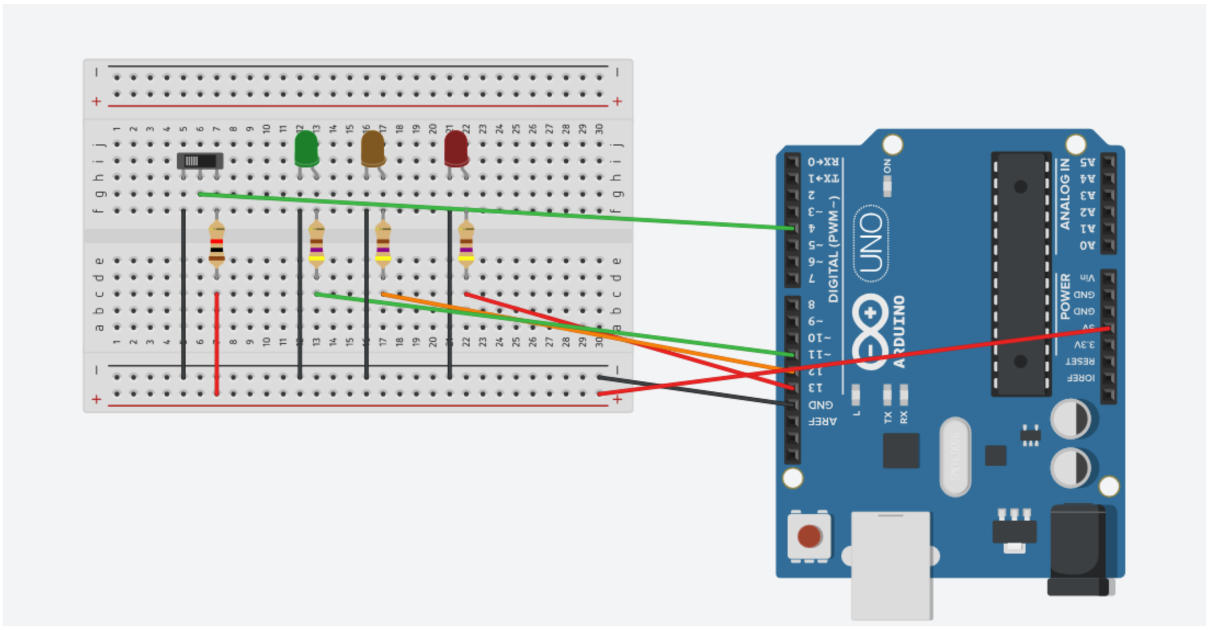
State 1	State 2	State 3	State 4
First 6 Seconds	**Next 2 Seconds**	**Next 8 Seconds**	**Next 2 Seconds**
Red Light ON Amber Light OFF Green Light OFF	Red Light ON Amber Light ON Green Light OFF	Red Light OFF Amber Light OFF Green Light ON	Red Light OF Amber Light ON Green Light OFF

<u>Tip: this is a basic copy-paste of the Blink programme with new pinMode assignments, HIGH LOW digitalWrite, timing etc.</u>

<u>If you have been successful you should now see the three LED blinking on and off in sequence in the correct intervals; this should run continuously</u>

Additional *Add a switch for a request-controlled traffic light sequence*

Finally, we can create a true **Process-Sense-Compute-Decide-Act** cycle by adding an INPUT (“the “sense” step) in the form of a voltage from a switch.



This involves setting up pin/socket 4 as an INPUT and having a conditional step that depends on the reading on Pin 4. We will need to define a variable will be assign to the reading of the state of the switch – this will go above the setup() block

```
int pb0; // we have a new variable for the reading from the switch
void setup()
```

The void setup() block will need to include a new assignment as we are INPUTing a signal INTO the microcontroller:

```
  pinMode(4, INPUT); // now using pin 4 also but as an INPUT
```

and in the main loop we will need to read the state of pin 4 (HIGH or LOW) and make a determination of an action to take based on that state; so our void loop() block will need to include:

```
pb0 = digitalRead(4);
```

and we need to make a conditional decision based on the reading:

```
if (pb0 == HIGH){ // if the switch is in the ON position
  // run the traffic light sequence
 delay(2000);
}else{
  // if the switch is off keep the traffic lights in State 1
}
```