Objectives

The objective of today's lab is to learn how to control an output from the Arduino. We will first start with the Arduino's inbuilt LED and later expand this example to control an externally connected LED.

This worksheet will walk you through the steps of being able to display your name via Morse Code on an LED. Morse code is a method for encoding text as a sequence of dots and dashes. A 'Dot' will be represented as turning the LED on for 1 second. A 'Dash' will be represented by turning on the LED for 3 seconds. After a 'dot' or 'dash' is displayed, the LED will need to be off for 1 second before displaying the next 'dot' or 'dash'.

The basic premise of the sketch is that we have an array (a vector) of signal instructions — a set of 'dots' and 'dashes'. The sketch keeps track of where within these set of signal instructions we currently are, so that every time the 'loop' function is executed, the LED can turn on for the correct amount of time for the current position within the sequence.

1 Turning on the LED

First start by opening a blank sketch: 'File' →'Examples' →'01.Basics' →'BareMinimum'. Your IDE should now contain minimum sketch discussed in the previous lecture.

Our first step is to set up the serial connection: this allows us to monitor what the Arduino is doing. Then we can set the internal LED pin to an output mode. The above is done within the 'setup' function. Within the main 'loop' we set the pin with the LED on to 'HIGH'.

```
void setup() {
  Serial.begin(9600);
  pinMode(LED_BUILTIN, OUTPUT);
  delay(3000);
  Serial.println("System Ready");
}

void loop() {
  digitalWrite(LED_BUILTIN, HIGH);
}
```

You can now compile your sketch ('Sketch' →'Verify\Compile') and then upload to your Arduino ('Sketch' →'Upload') – Ensure you have selected your Arduino in your board manager.

You should see your Arduino power on, wait 3 seconds, display 'System Ready' and then turn on the LED.

2 Add Morse Code variables

To make our sketch understand Morse Code, we need to add five lines of code within the global scope. The first line declares that a unit of time is 1000 milliseconds. The second line says that a 'dot' is 1 unit of time. The third line states that a 'dash' is 2 units of time.

```
#define TIME_UNIT 1000
#define DOT_UNITS 1
#define DASH_UNITS 3
char output[7] = { '-', '.', '.', '.', '-', '-', '.' };
int current_char = 0;
```

The fourth line creates an array (a vector) of 7 'dots' and 'dashes' representing the sequence of signals needed for the word 'DAN'. The first three (-..) represents D, the next two (.-) represents A, and the last two (-.) represents N. The fifth and last line stores the current position within the Morse code sequence, initially zero.

3 Process the Morse Code message

We now need to modify the main 'loop' function to correctly process the Morse code message. This is broken down into three stages: calculate the duration to turn the LED on for (see Section 3.1), turn the LED on for the desired duration (see Section 3.2), and lastly calculate the next signal to display (see Section 3.3).

Before continuing to the next sections, we first need to remove the existing 'digitalWrite' line from 'loop' function so that it is empty.

3.1 Calculate the duration

At the beginning of the 'loop' function, we need to enter the code to calculate the duration to turn the LED on for. This is based upon the `current_char` variable which tracks the position within the signal sequence (the `output` variable).

If the current position within the sequence equals a '.' then we need to set the LED (`led_on_time`) to be on for 1 time unit; however, if the current position within the sequence equals a '-', then we need to set the LED to be on for 3 time units.

```
int led_on_time = 0;
if(output[current_char] == '.'){
  led_on_time = DOT_UNITS * TIME_UNIT;
} else if (output[current_char] == '-'){
  led_on_time = DASH_UNITS * TIME_UNIT;
}
```

3.2 Turn on the LED

The next logic within the 'loop' function needs to be turning on the LED, waiting for the desired duration calculated in the previous step, then turning the LED off again.

```
digitalWrite(LED_BUILTIN, HIGH);
delay(led_on_time);
digitalWrite(LED_BUILTIN, LOW);
```

3.3 Calculate the next signal

Once you have displayed the current signal to the LED, the final piece of logic within the 'loop' function is to calculate where within the sequence the next iteration of the 'loop' should go.

The logic for this is that if your current signal being displayed is the last signal in the sequence, then you need to start back at zero, else go to the next signal in the sequence.

```
if (current_char > 6){
  current_char = 0;
} else {
  current_char++;
}
delay(TIME_UNIT);
```

4 Try it out on your Arduino

Now that you have the above sections complete in your sketch, compile and upload the sketch to your teams Arduino.

Once you have successfully run this sketch on your Arduino, change the sequence displayed to the name of one member from your team. You can find a Morse code chart on the Wikipedia page: `https://en.wikipedia.org/wiki/Morse_code`.

Help

If you are having issue compiling the project or don't understand where each section goes, you can see a complete version of the sketch with additional source code comments at `https://github.com/dsbrennan/mec217-arduino-labs/blob/main/lab-1-blink/lab-1-blink.ino`.