Timing

Blink1 §

```
// Uses pin 6 to control the blinking of an external red LED.

const int RED_LED_PIN = 6;

void setup() {

  pinMode(RED_LED_PIN, OUTPUT);
}

void loop() {

  digitalWrite(RED_LED_PIN, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);                       // wait for a second
  digitalWrite(RED_LED_PIN, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);                       // wait for a second
}
```

The simple program that blinks the red LED used the `delay()` function. This is very handy function for introducing a short time delays into programs. It helps buffer the differing time scales of humans and micro-controllers — seconds for humans and microseconds for micro-controllers. It is easy use — simply pass the number of milliseconds to delay. (1000 ms = 1 s.) The programs essentially "pauses" for that amount of time. Try changing the delay. See how fast you make the LED blink and still see it as blinking.

In many of the programs that you might develop, you might be able to use the delay function. However, it has a major flaw — the processor cannot do anything else while the delay function is operating.

The delay function is simply a while loop where nothing is happening inside the loop.

Recall that a micro-controller program is a continuous loop of checking current operating conditions responding to those when appropriate. Blink1 program, the loop takes slightly more than 2 seconds. For 99.9% of that time, the program is stuck inside the delay loop, and nothing else can happen.

A better approach might be use a timer or stopwatch approach.

1. Note the time when something happened, for example, when he LED turned on.
2. Let the program run through its loop in the usual fashion. Each time through the loop, note the current time.
3. Compare the elapsed time (difference between the current time and the start time) to the predefined time that the LED should be turned on.
4. When the elapsed time becomes equal to or greater than the “on” time, make the change — turn the LED off. Note the time at which the LED turns off.
5. Repeat the whole cycle to determine when to turn the LED back on.

`millis()`

The key to this slightly different approach is the function `millis()`. (Again, see the description in the language reference.) The function returns the number of milliseconds that the current program has been running since it started. The result is stored in a variable of type unsigned long.

```
unsigned long currentTime;
currentTime = millis();
```

You can use millis() to note the time when a process started. ("I put the turkey in the oven at noon.") Then use millis() to check the current time occasionally — maybe once per loop — and determine the elapsed time. (Look at your watch from time to time.) When the elapsed time is "long enough", do whatever needs to be done. ("My watch says that it is 3:00 p.m. Three hours is long enough to cook the turkey, so I'll take it out.")

This approach requires more variable declarations and some if–else statements, but it gives much more flexibility to do other things within the program.

A subtlety in using `millis()`

The `millis()` function returns a value in the form of an unsigned long integer, meaning that cannot have a negative value. (See the language reference.) This is fine for the actual number of milliseconds that the processor returns, since it starts at zero and counts up. However, if you are doing any math (subtraction in particular) with with unsigned longs, it might be possible to have a result that should be negative, but won't be represented properly by an unsigned long. (Some value will result, but it will be wrong)

In using unsigned longs, you must be careful. Either you need to check before doing any subtractions to make sure that negative values won't occur, or you can first convert them to regular longs, which do have a sign and won't have any math difficulties.

You can use regular C typecasting nomenclature. In addition, Arduino does provide a typecast (or conversion) functions. (See the language reference.) For example, if `timeNow` is defined as a long integer, the program line

```
timeNow = long( millis() );
```

will take the unsigned long returned by `millis()`, converted it to signed long, and assign it to the variable `timeNow`.

Blink one LED, using better timing

```
// Uses millis function to blink an LED attached to pin 6.

const int RED_LED_PIN = 6;
long currentTime, startTime;
long onTime = 1000;           //LED will be on for 1000 ms.
long offTime = 1000;          //LED will be off for 1000 ms.
int ledIsOn = 0;              //A boolean to keep of the LED state.

void setup() {

  pinMode(RED_LED_PIN, OUTPUT);
  startTime = long( millis() );     //Set the start time for the LED.
}

void loop() {

  currentTime = long( millis( ));             //Get the current time.

  //If the LED has been on long enough, turn it off.
  if( ledIsOn == 1 && (currentTime - startTime >= onTime)  ){
    digitalWrite(RED_LED_PIN, LOW );
    ledIsOn = 0;
    startTime = long( millis() );
  }

  //If the LED has been off long enough, turn it on.
  if( ledIsOn == 0 && (currentTime - startTime >= offTime)  ){
    digitalWrite(RED_LED_PIN, HIGH );
    ledIsOn = 1;
    startTime = long( millis() );
  }

  //Now there is lots of time for other things.
}
```

More code, but flexible is the program will do more than just blinking LEDs.

The LED will blink in an identical fashion to the earlier blink program.

Blink two LEDs

Now we can add a second LED, which can have its own timer and blink at its own rate. Here is the set up portion of a program that will blink a red LED at 0.5 Hz and a greed LED at 1 Hz.

Basically, everything that happened in the single-LED program is doubled here. It gets big in a hurry.

Blink4 §

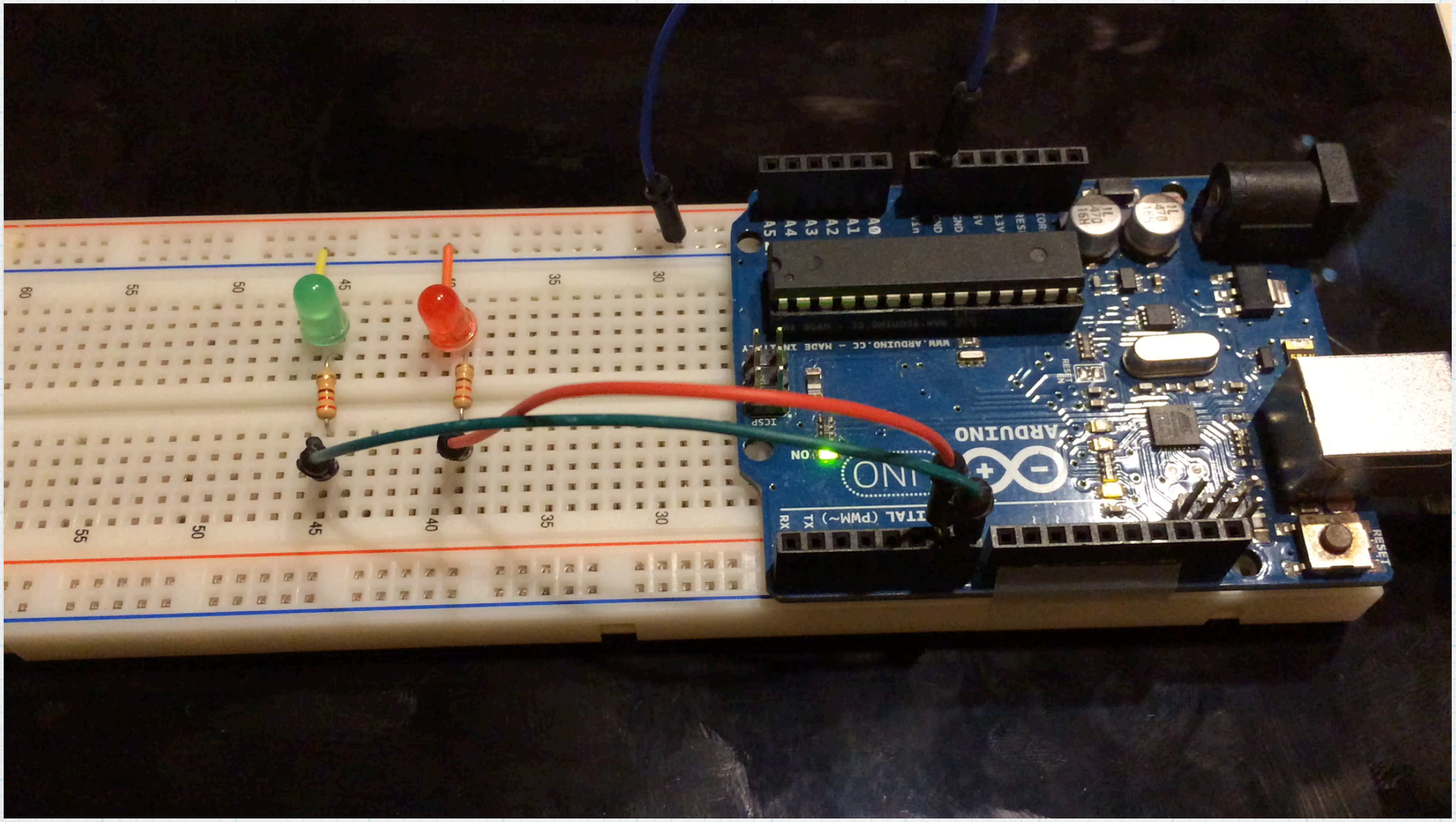
```
// Uses millis function to blink red and green LEDs.

const int RED_LED_PIN = 6;
const int GREEN_LED_PIN = 7;
long currentTime, redStartTime, greenStartTime;
long redOnTime, greenOnTime, redOffTime, greenOffTime;
int redIsOn, greenIsOn;

void setup() {
  //Initialize everything, with both LEDs turned off.
  pinMode(RED_LED_PIN, OUTPUT);
  pinMode(GREEN_LED_PIN, OUTPUT);
  redOnTime = 1000;
  redOffTime = 1000;
  greenOnTime = 500;
  greenOffTime = 500;
  redIsOn = 0;
  greenIsOn = 0;
  digitalWrite( RED_LED_PIN, LOW );
  digitalWrite( GREEN_LED_PIN, LOW );
  redStartTime = long( millis() );
  greenStartTime = long( millis() );
}
```

And here is the loop portion of that program.

Blink4 §

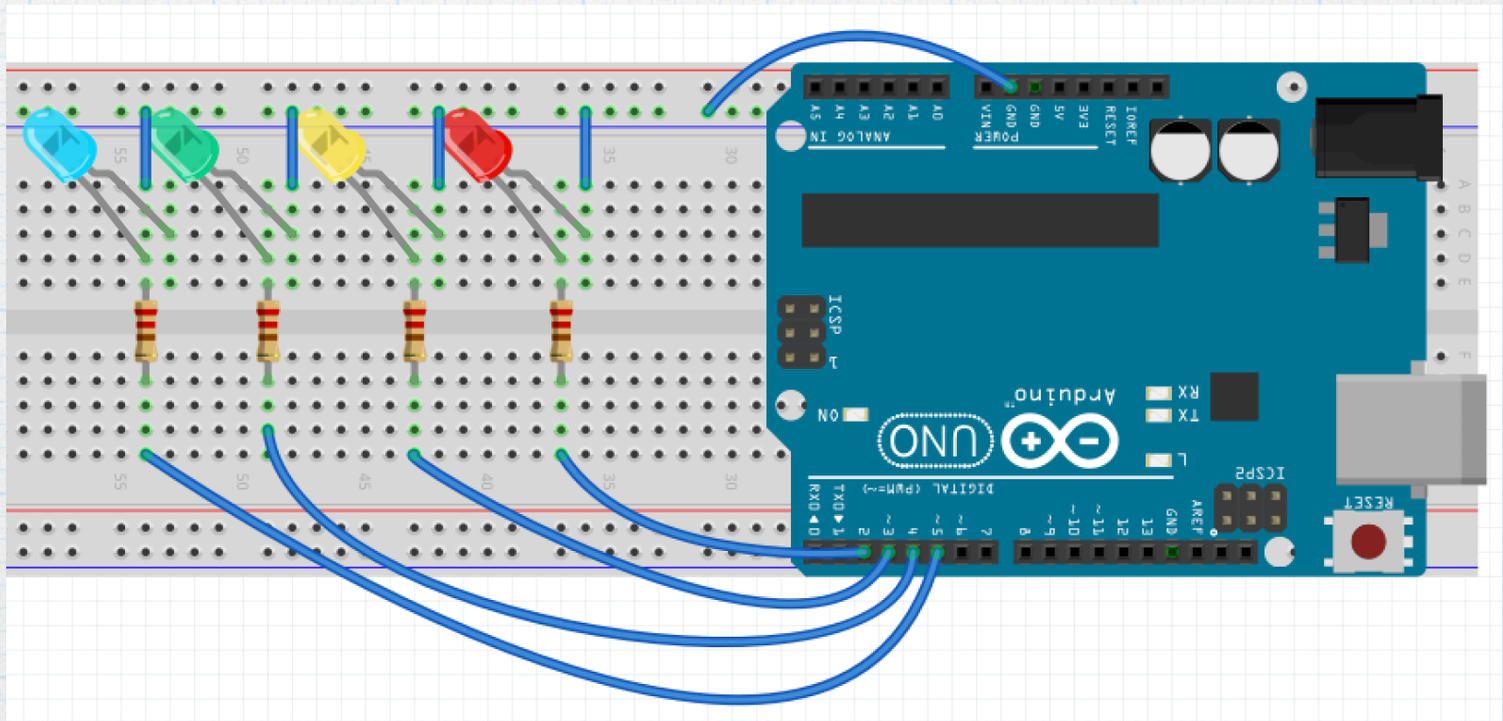
```
void loop() {

  currentTime = long( millis( ));

  //If the red LED has been on long enough, turn it off.
  if( redIsOn == 1 && (currentTime - redStartTime >= redOnTime)  ){
    digitalWrite(RED_LED_PIN, LOW );
    redIsOn = 0;
    redStartTime = long( millis() );
  }

  //If the red LED has been off long enough, turn it on.
  if( redIsOn == 0 && (currentTime - redStartTime >= redOffTime)  ){
    digitalWrite(RED_LED_PIN, HIGH );
    redIsOn = 1;
    redStartTime = long( millis() );
  }

  //If the green LED has been on long enough, turn it off.
  if( greenIsOn == 1 && (currentTime - greenStartTime >= greenOnTime)  ){
    digitalWrite(GREEN_LED_PIN, LOW );
    greenIsOn = 0;
    greenStartTime = long( millis() );
  }

  //If the green LED has been off long enough, turn it on.
  if( greenIsOn == 0 && (currentTime - greenStartTime >= greenOffTime)  ){
    digitalWrite(GREEN_LED_PIN, HIGH );
    greenIsOn = 1;
    greenStartTime = long( millis() );
  }

  //Now there is lots of time for other things.
}
```

Here is a very boring movie of the resulting program in action.



Adding more LEDs is straight-forward conceptually, but the program is getting rather long and repetitive. Let's make use of what we have learned previously and use structs, arrays, and loops.

1. Make an "LED" struct with members that contain all of the info about the LED — pin number, on/off start, start time, on and off times.
2. Make an array of the structs for all of the LEDs we might want to include.
3. Use a loop inside of the setup section to initialize all of the information relating to the various.
4. Use a for loop inside the loop section to check times and turn LEDs on and off.



Variables and definitions.

Blink5

```
struct lightEmittingDiode{
  int pin;
  long onTime;
  long offTime;
  long startTime;
  int onOrOff;
};

const int NUM_OF_LEDS = 4;
int i;
long currentTime;
struct lightEmittingDiode LED[NUM_OF_LEDS];
```

1. Define a struct that has all info pertinent to each individual LED: pin number, on and off times, start time, and on/off status. Everything relating to time is defined as a long. (Not unsigned long.)
2. For this example, we will use 4 LEDS (red, yellow, green, blue)
3. An integer for counting.
4. A long for the currentTime, used at various points in the program.
5. An array of 4 of the structs.

Set up.

Blink5

```
void setup() {

  currentTime = long( millis() );

  for( i = 0; i < NUM_OF_LEDS; i++ ){

    LED[i].pin = i + 2;
    LED[i].onTime = 500;
    LED[i].offTime = 1500;
    LED[i].startTime = currentTime + i*500;
    LED[i].onOrOff = 0;

    pinMode(LED[i].pin, OUTPUT);
    digitalWrite(LED[i].pin, LOW);
  }
}
```

1. Grab the current time.
2. Everything is initialized inside a loop.
3. We will use digital pins 2 thru 5.
4. On time for all diodes is 0.5 s. Off time is 1.5 s.
5. Initially, all diodes are off. The start time of the diodes is staggered so that they will come on in sequence. Of course, this can be changed to whatever sort of pattern you like.

The loop.

Blink5

```
void loop() {

  currentTime = long( millis() );

  for(i = 0; i < NUM_OF_LEDS; i++){

    if( LED[i].onOrOff == 1 && (currentTime - LED[i].startTime >= LED[i].onTime) ){
       digitalWrite(LED[i].pin, LOW);
       LED[i].startTime = long( millis() );
       LED[i].onOrOff = 0;
    }

    if( LED[i].onOrOff == 0 && (currentTime - LED[i].startTime >= LED[i].offTime) ){
       digitalWrite(LED[i].pin, HIGH);
       LED[i].startTime = long( millis() );
       LED[i].onOrOff = 1;
    }
  }
}
```

Inside the loop, the program is much the same as the previous examples. The current time at the beginning of each loop is saved. Each diode is checked in sequence — if it is on and been on long enough, it is turned off and its individual timer is re-started. If it is off and been off long enough, it is turned on with the timer being re-started.

Four LEDs — the movie.