# American International University- Bangladesh

## Department of Electrical and Electronic Engineering

EEE4103: Microprocessor and Embedded Systems Laboratory

**Experiment Title:** Study of a Digital Timer using millis() function of Arduino to avoid problems associated with the delay() function.

## Introduction:

In this project, you'll build a digital timer that turns on an LED every minute. Besides, you will be able to know how long you are working on your projects by using Arduino's built-in Timer.

## Theory and methodology:

Up to now, when you've wanted something to happen at a specific time interval with the Arduino, you've used delay(). This is handy, but a little confining. When the Arduino calls the delay() function, it freezes its current state for the entire duration of the delay. That means, there can be no other input or output while it is waiting. Delays are also not very helpful for keeping track of time. If you wanted to do something every 10 seconds, having a 10-second delay would be cumbersome.

The millis() function helps to solve these problems. It keeps track of the time your Arduino has been running in milliseconds.

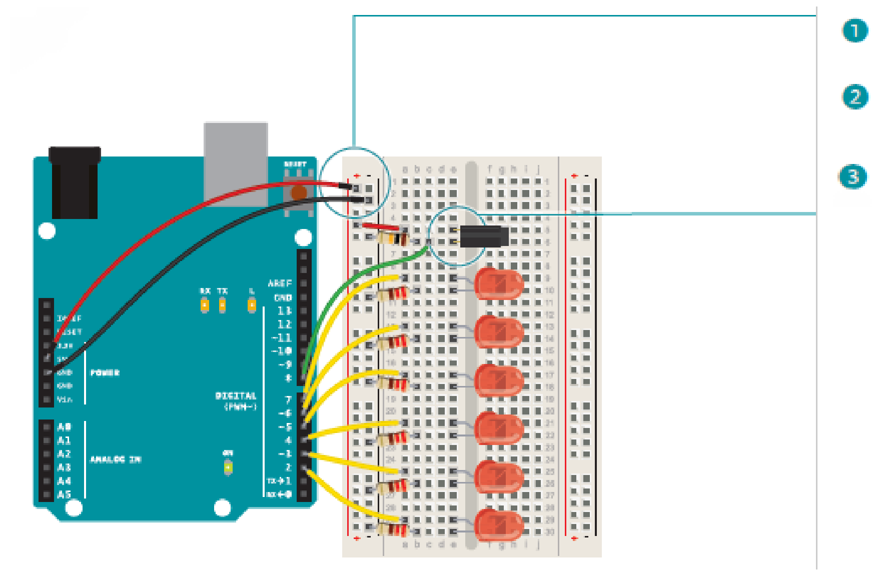
So far, you've been declaring variables as int. An int (integer) is a 16-bit number, it holds values between -32,768 and 32,767 (since the MSB is used for a signed bit, so number ranges are between $-2^{15}$ and $2^{15}-1$). Those may be some large numbers, but if the Arduino is counting 1000 times a second with millis(), you would run out of space in less than a minute. The long data type holds a 32-bit number (between -2,147,483,648 and 2,147,483,647). Since you can't run time backward, to get negative numbers, the variable to store millis() time is called an unsigned long. When a datatype is called unsigned, it is only positive. This allows you to count even higher numbers. An unsigned long number can count up to 4,294,967,295. That is enough space for milis() function to store time for almost 50 days. By comparing the current millis() function to a specific value, you can see if a certain amount of time has passed.

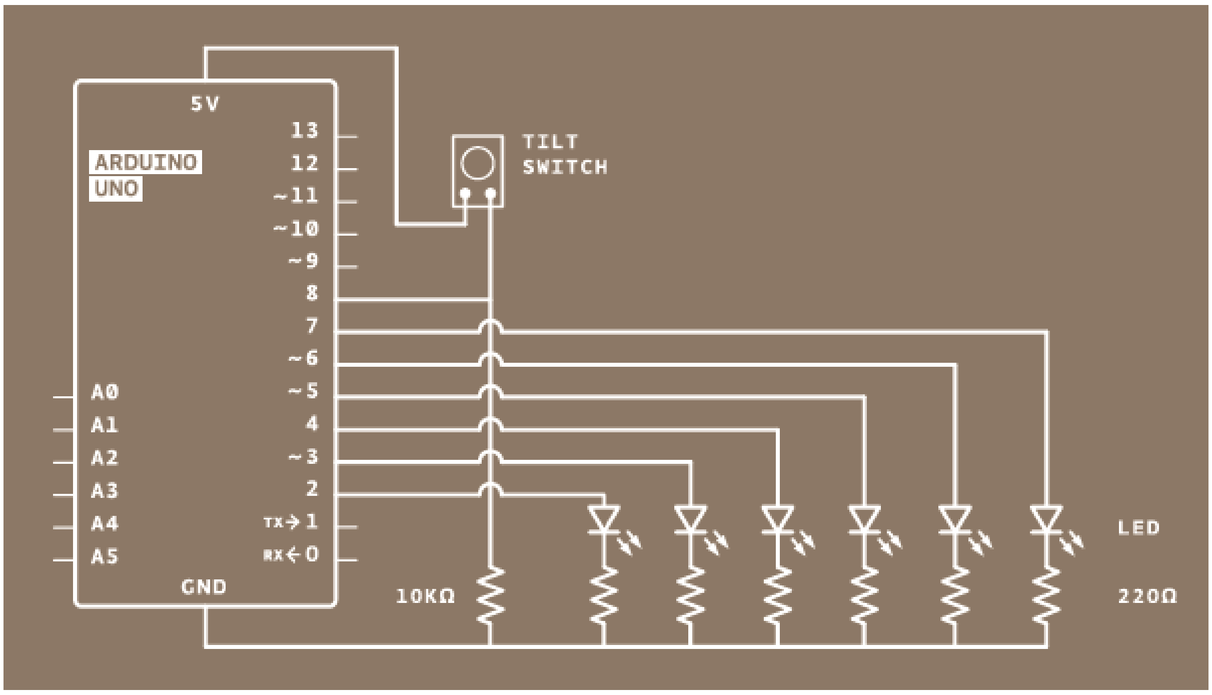
When you turn your Digital Timer over, a tilt switch will change its state, and that will set off another cycle of LEDs turning on.

The tilt switch works just like a regular switch in that it is an on/off sensor. You'll use it here as a digital input. What makes tilt switches unique is that they detect orientation. Typically, they have a small cavity inside the housing that has a metal ball. When tilted in the proper way, the ball rolls to one side of the cavity and connects the two leads that are in your breadboard, closing the switch. With six LEDs, your timer will run for six minutes.

## Experimental Setup and Procedures:

1. Connect power and ground to your breadboard.
2. Connect the anode (longer leg) of six LEDs to digital pins 2-7 and connect the LEDs to the ground through 100 Ω resistors.
3. Connect one lead of the tilt switch to +5 V. Connect the other lead to a 10 kΩ resistor to the ground and connect the junction where they meet to digital pin 8.





## The Step-by-Step Process for Code writing:

a. Declare a named constant:

You need several global variables in your program to get this all working. To start, create a variable named *SwitchPin* where the tilt switch would be connected.

b. Create a variable to hold the time

Create a variable of name **unsigned long** to hold the time of the LED that was changed last time.

c. Declare variable names for the inputs and outputs

Create two variables for the current and previous switch states to compare the switch positions from one loop to the next. Create another variable named *led* to store the pin numbers of the LEDs to be stored to determine which one to be turned on next time. Start out with pin 2.

d. Declare a variable describing the interval between events

The last variable you need to create is the interval between each LED turning on. This will be a **long** data type. In 10 minutes (i.e., the time between each LED turning on), 6,00,000 milliseconds (1 minute = 60 seconds and 1 second = 1000 milliseconds) pass. If you want the delay between lights to be longer or shorter, this is the number you change.

e. Set the direction of your digital pins

In the **void setup()** function, you need to declare the LED pins 2-7 as the output pins. A **for** loop declares all six pins as OUTPUT (so that the microcontroller can send signals to these pins) with just 3 lines of code. You also need to declare *SwitchPin* as an INPUT (so that the microcontroller can receive a signal from this pin).

f. Check the time since the program started running

When the **void loop()** starts, you are going to get the amount of time the Arduino has been running with **millis()** and store it in a local variable named *CurrentTime*.

g. Evaluate the amount of time that has passed since the previous loop()

Using an **if()** statement, you'll check to see if enough time has passed to turn on an LED. Subtract the *CurrentTime* from the *PreviousTime* (initialized as 0 for the first time during declaration) and check to see if it is greater than the interval variable. If 6,00,00 milliseconds have passed (one minute), you'll set the variable *PreviousTime* to the value of *CurrentTime*.

h. Turn on an LED, prepare for the next one

*PreviousTime* indicates the last time an LED was turned on. Once you have set *PreviousTime*, turn on the LED, and increment the *led* variable, for example from 2 to 3. The next time you pass the time interval, the next LED will light up.

i. Check to see if all lights are on

Add one more *if* statement in the program to check whether the LED connected to pin 7 (next pin) is turned on or not. was turned on. However, don't do anything with this yet, wait for the end of an hour (set time) to decide.

j. Read the value of the switch

After checking the time, you should check the switch state by reading it and storing the value in the *SwitchState* variable.

k. Reset the variables to their defaults if necessary

With an *if* statement, check the position or status of the switch. The != evaluation checks to see if *SwitchState* variable doesn't equal the *PrevSwitchState* variable. If they are different, turn the LEDs off, return the *led* variable to the first pin, and reset the timer for the LEDs by setting *PreviousTime* to the value of *CurrentTime*.

l. Set the current state to the previous state

At the end of the **void loop()** function, save the value of the *SwitchState* variable in the value of the *PrevSwitchState* variable so that you can compare it to the value you get for the *SwitchState* variable in the next **void loop()**.

## Code:

```
const int SwitchPin = 8;
unsigned long PreviousTime = 0;
int SwitchState = 0;
int PrevSwitchState = 0;
int led = 2;
long interval = 60000; // for 1 min = 60,000 ms delay


void setup() {
        for (int x = 2; x < 8; x++)
                {
                pinMode(x, OUTPUT);
                }
        pinMode(SwitchPin, INPUT);
        }
```

```
void loop() {
unsigned long CurrentTime = millis();
        if (CurrentTime - PreviousTime > interval) {
        PreviousTime = CurrentTime;

        digitalWrite(led, HIGH);
        led++;
                if (led == 7){
                }
        }

        SwitchState = digitalRead(switchPin);
        if (SwitchState != PrevSwitchState){
                for (int x = 2 ; x < 8; x++)
                {
                digitalWrite(x, LOW);
                }
        led = 2;
        PreviousTime = CurrentTime;
        }
        PrevSwitchState = SwitchState;
        }
```

## Apparatus:

- Arduino Uno/Arduino Mega Microcontroller Board
- Tilt sensor (One)
- LEDs (Six)
- Resistors (One 10 kΩ and Six 100 Ω)

## Questions for Report Writing:

1) Include all codes and scripts in the lab report.
2) Implement this system using Proteus or an online simulation platform *www.tinkercad.com*.

## References:

1) https://www.arduino.cc/.
2) ATMega328 manual
3) https://www.avrfreaks.net/forum/tut-c-newbies-guide-avr-timers
4) http://maxembedded.com/2011/06/avr-timers-timer0/