American International University- Bangladesh
Department of Electrical and Electronic Engineering
EEE4103: Microprocessor and Embedded Systems Laboratory

Title: Part-1: Taking external inputs in Arduino: Implementation of runway approach lights

Introduction:

- The objective of this experiment is to learn how to take external inputs in Arduino. Here, the external input will be given by a push switch.
- The objective of this experiment is also to get familiarized with Debouncing: Implementation and effects in Microcontroller.

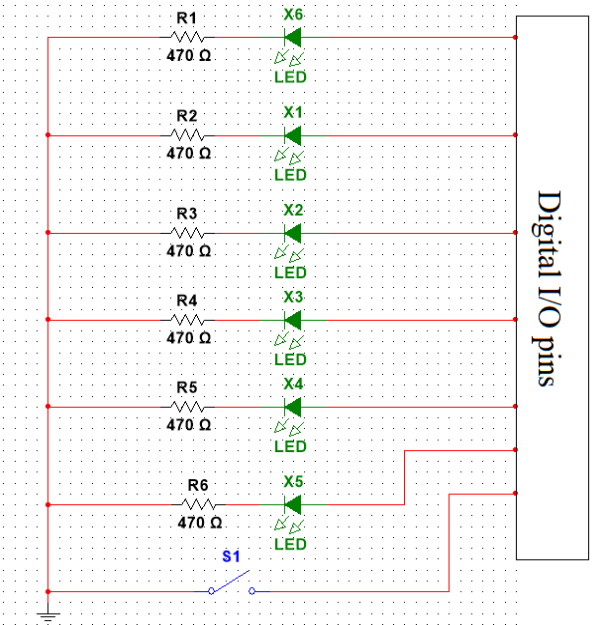
Theory and Methodology:

Runway approach lights to an airport runway are a series of lights that flash quickly in sequence to guide an aircraft to the runway. These lights are especially effective during bad weathers, such as heavy rain; fog etc. when the visibility is very poor. The aim of this experiment is to mimic this system.

A series of 6 LEDs will flash in a specific sequence and upon pressing the switch; the sequence will reverse the direction. The switch will be connected to an I/O pin and will be set as an input. Upon pressing the switch, the microcontroller will read the change of state of the I/O pin (here set as an input) and execute instructions to reverse the flash sequence.

Debouncing: Bouncing is the tendency of any two metal contacts in an electronic device to generate multiple signals as the contacts close or open; debouncing is any kind of hardware device or software that ensures that only a single signal will be acted upon for a single opening or closing of a contact.

Experimental setup:



Apparatus:

- Arduino Uno
- LED
- Resistor
- Push switch

Code implementation of a traffic system with Timer:

```
#define GREEN1 8  //define name of pins used
#define GREEN2 10
#define GREEN3 12
#define RED1 9
#define RED2 11
#define RED3 13
#define switch1 2

//define the delays for each LED
int LED_blink = 700;

//define variable for switch press
int switch_read;    //defining a variable which will read the state of the switch
int LED_sequence=1; //defining which way the LEDs will light up (left to right or right to
left)

int delay_timer (int miliseconds)
{
  int count = 0;
  while(1)
  {
    if(TCNT0 >= 16)  // Checking if 1 milisecond has passed
      {
        TCNT0=0;
        count++;
        if (count == miliseconds) //checking if required miliseconds delay has passed
        {
           count=0;
           break; // exits the loop
        }
      }
  }
  return 0;
}

void setup() {
   //define pins connected to LEDs as outputs
   pinMode(GREEN1, OUTPUT);
   pinMode(GREEN1, OUTPUT);
   pinMode(GREEN3, OUTPUT);
```

```
   pinMode(RED1, OUTPUT);
   pinMode(RED2, OUTPUT);
   pinMode(RED3, OUTPUT);
   pinMode(switch1, INPUT);

   //set up timer
   TCCR0A = 0b00000000;
   TCCR0B = 0b00000101; //setting highest prescaler for timer clock
   TCNT0=0;
}

void loop() {

  switch_read=digitalRead(switch1);
  if (switch_read==LOW){
    LED_sequence=!LED_sequence;
  }
  if (LED_sequence==1){

   //to make green1 LED blink
   digitalWrite(GREEN1, HIGH);
   delay_timer(LED_blink);
   digitalWrite(GREEN1, LOW);

   //to turn red1 LED blink
   digitalWrite(RED1, HIGH);
   delay_timer(LED_blink);
   digitalWrite(RED1, LOW);

   //green2 blink and so on
   digitalWrite(GREEN2, HIGH);
   delay_timer(LED_blink);
   digitalWrite(GREEN2, LOW);

   digitalWrite(RED2, HIGH);
   delay_timer(LED_blink);
   digitalWrite(RED2, LOW);

   digitalWrite(GREEN3, HIGH);
   delay_timer(LED_blink);
   digitalWrite(GREEN3, LOW);

   //green2 blink and so on
   digitalWrite(RED3, HIGH);
   delay_timer(LED_blink);
   digitalWrite(RED3, LOW);
```

```
//    digitalWrite(GREEN1, HIGH);
//    digitalWrite(RED1, HIGH);
//    digitalWrite(GREEN2, HIGH);
//    digitalWrite(RED2, HIGH);
//    digitalWrite(GREEN3, HIGH);
//    digitalWrite(RED3, HIGH);
//    delay_timer(LED_blink);
//    delay_timer(LED_blink);
//    digitalWrite(GREEN1, LOW);
//    digitalWrite(RED1, LOW);
//    digitalWrite(GREEN2, LOW);
//    digitalWrite(RED2, LOW);
//    digitalWrite(GREEN3, LOW);
//    digitalWrite(RED3, LOW);
//
    delay_timer(LED_blink);

  }
  else {

    digitalWrite(RED3, HIGH);
    delay_timer(LED_blink);
    digitalWrite(RED3, LOW);

    digitalWrite(GREEN3, HIGH);
    delay_timer(LED_blink);
    digitalWrite(GREEN3, LOW);

    digitalWrite(RED2, HIGH);
    delay_timer(LED_blink);
    digitalWrite(RED2, LOW);

    digitalWrite(GREEN2, HIGH);
    delay_timer(LED_blink);
    digitalWrite(GREEN2, LOW);

    digitalWrite(RED1, HIGH);
    delay_timer(LED_blink);
    digitalWrite(RED1, LOW);

    digitalWrite(GREEN1, HIGH);
    delay_timer(LED_blink);
    digitalWrite(GREEN1, LOW);

    delay_timer(LED_blink);
  }
}
```

Questions for report writing:

1) Include all codes and scripts into the lab report following the writing template mentioned in appendix A of Laboratory Sheet Experiment 4.
2) According to your opinion, explain how to prevent a switch from debouncing.
3) Design the same system using the Proteus simulation tool.

Reference(s):

1) https://www.arduino.cc/.
2) ATMega328 manual

American International University- Bangladesh
Department of Electrical and Electronic Engineering
EEE4103: Microprocessor and Embedded Systems Laboratory

Title: Part-2: Digital Timer project using millis() function to avoid the problems associated with delay().

Introduction: In this project, you'll build a digital timer that turns on an LED every one minute. Know how long you are working on your projects by using the 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 delay(), it freezes its current state for the duration of the delay. That means there can be no other input or output while it's 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.

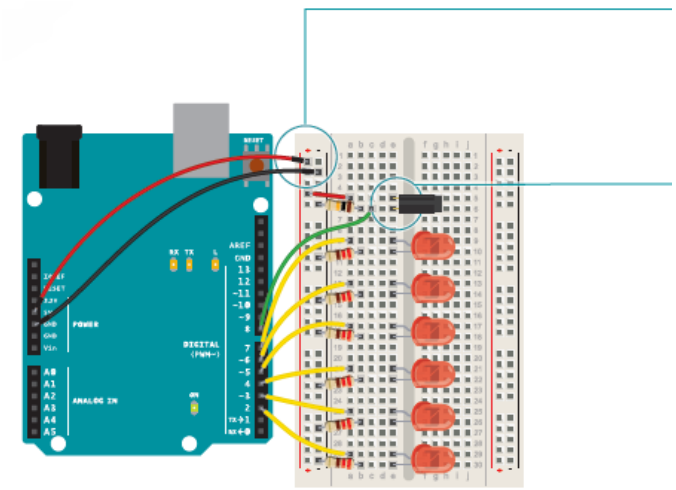
The millis() function helps to solve these problems. It keeps track of the time your Arduino has been running in milliseconds.

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. Those may be some large numbers, but if the Arduino is counting 1000 times a second with millis(), you'd run out of space in less than a minute. The long datatype holds a 32-bit number (between -2,147,483,648 and 2,147,483,647). Since you can't run time backwards 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. An unsigned long can count up to 4,294,967,295. That's enough space for milis() to store time for almost 50 days. By comparing the current millis() to a specific value, you can see if a certain amount of time has passed.

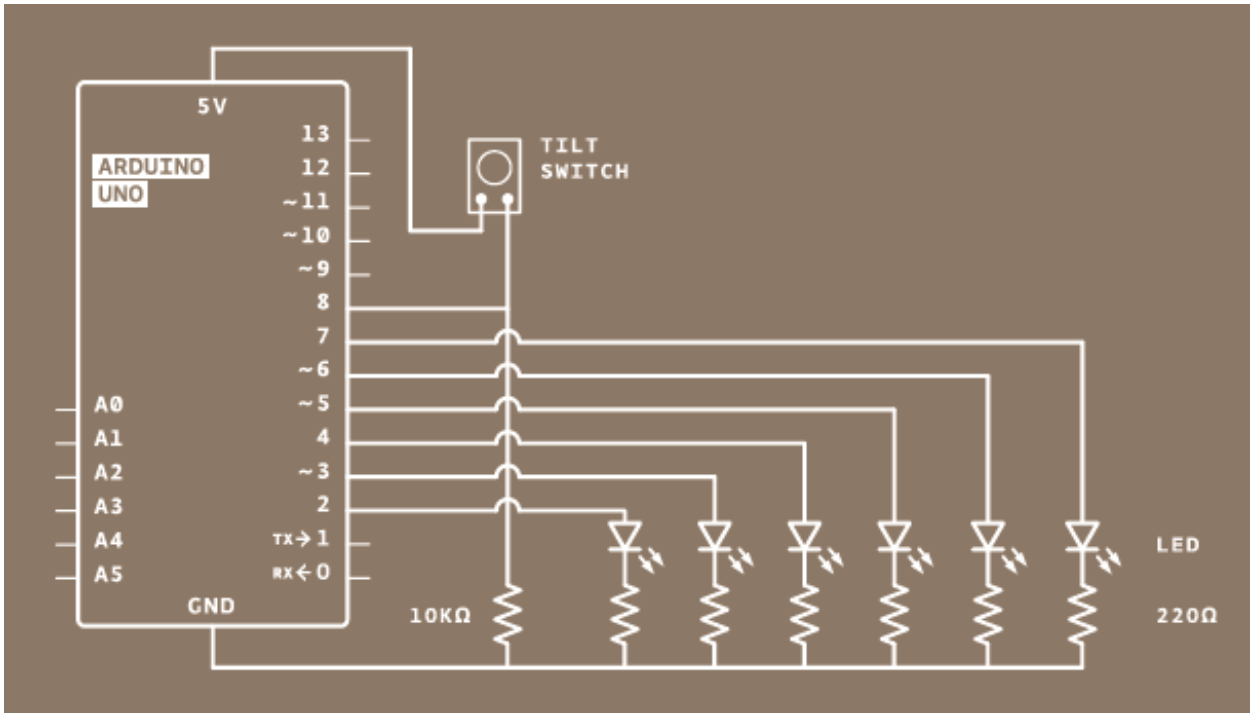
When you turn your Digital Timer over, a tilt switch will change its state, and that will set off another cycle of LEDs turning on.

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:



1. Connect power and ground to your breadboard.
2. Connect the anode (longer leg) of six LEDs to digital pins 2
   Connect the LEDs to ground through 220-ohm resistors.
3. Connect one lead of the tilt switch to 5V. Connect the other
   a 10-kilohm resistor to ground. Connect the junction where th
   meet to digital pin 8.



Step-by-Step process for Code writing:

Declare a named constant	You're going to need a number of global variables in your program to get this all working. To start, create a constant named switchPin. This will be the name of the pin your tilt switch is on.
Create a variable to hold the time	Create a variable of type `unsigned long`, This will hold the time an LED was last changed.
Name variables for the inputs and outputs	Create a variable for the switch state, and another to hold the previous switch state. You'll use these two to compare the switch's position from one loop to the next. Create a variable named led. This will be used to count which LED is the next one to be turned on. Start out with pin 2.
Declare a variable describing the interval between events	The last variable you're creating is going to be the interval between each LED turning on. This will be be a `long` datatype. In 10 minutes (the time between each LED turning on) 600,000 milliseconds pass. If you want the delay between lights to be longer or shorter, this is the number you change.
Set the direction of your digital pins	In your `setup()`, you need to declare the LED pins 2-7 as outputs. A `for()` loop declares all six as OUTPUT with just 3 lines of code. You also need to declare switchPin as an `INPUT`.
Check the time since the program started running	When the `loop()` starts, you're going to get the amount of time the Arduino has been running with `millis()` and store it in a local variable named `currentTime`.
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 previousTimeand check to see if it is greater than the interval variable. If 600,00 milliseconds have passed (one minute), you'll set the variable **previousTime** to the value of **currentTime.**

Turn on an LED, prepare for the next one	`previousTime` indicates the last time an LED was turned on. Once you've set `previousTime`, turn on the LED, and increment the led variable. The next time you pass the time interval, the next LED will light up.
Check to see if all lights are on	Add one more if statement in the program to check if the LED on pin 7 is turned on. Don't do anything with this yet. You'll decide what happens at the end of the hour later.
Read the value of the switch	Now that you've checked the time, you'll want to see if the switch has changed its state. Read the switch value into the `switchState` variable.
Reset the variables to their defaults if necessary	With an `if()` statement, check to see if the switch is in a different position than it was previously. The `!=` evaluation checks to see if `switchState` does not equal `prevSwitchState`. 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 `currentTime`.
Set the current state to the previous state	At the end of the `loop()`, save the switch state in `prevSwitchState` , so you can compare it to the value you get for `switchState` in the next `loop()`.

The Code:

```
const int SwitchPin = 8;
unsigned long PreviousTime = 0;
int SwitchState = 0;
int PrevSwitchState = 0;
int led = 2;
long interval = 1000; // 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
- Tilt sensor
- LED lights (Six)
- Resistors (One 10K and six 220 ohms)

Questions for report writing:

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

Reference(s):

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/