Traffic lights with pedestrian lights and stop button

In this worksheet we will further enrich the operation of the traffic lights, as implemented in the previous worksheet, by inserting a button, which will define the operation of the traffic lights as follows: normally the green car traffic light will work permanently turn on , unless the button is pressed, which will mean that a pedestrian is asking to cross the road. In this case, car traffic should be stopped (orange and red) and then the green light for pedestrians should be lit for a specified time interval. Finally, the pedestrian traffic light will turn red again and car traffic will start again (green). Then the process will be repeated from the beginning.

Circuit description

For this task we will need, in addition to the materials in previous worksheet, a button to simulate the operation of the pedestrian traffic light button. We will also need a resistor for the button. You can see the circuit in figure 1 below.

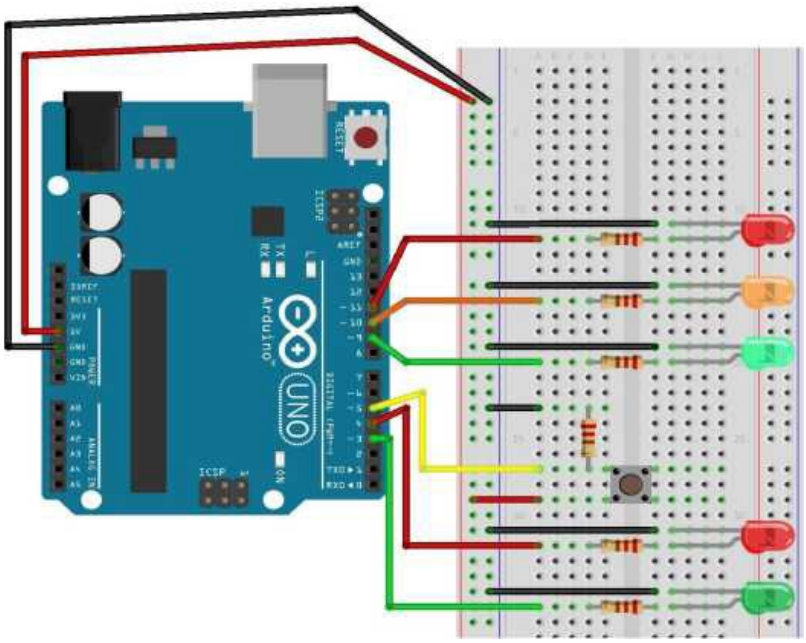

Figure 1.

Circuit programming

All that remain is to program our circuit. For convenience, you are given instructions in the form of steps:

1. Do the necessary initializations (variables, values, input/output).

2. Determine the times for the combinations of pedestrian and car traffic lights.

3. You should read the input from the corresponding input pin (button pin) and when you detect signal, take the corresponding actions described above. For this step, it's recommended to use time logging to know when the button was last pressed, using the millis() function.

Also set a small time interval during which vehicles traffic lights will not turn green although the red pedestrian light has turned on. Note which steps should be done only once and which will be repeated over and over again.

IF statement

In programming, many times we will need to check some condition to decide whether to execute a piece of code or to execute something else instead it. We achieve this by using the “If” statement, which is drawn up:

if (condition) { commands }

where, in the (condition) we have the checking we want to be done, usually using the comparison operators:

(>, <, =, <>, >=, <=) e.g. potVal > 500.

The condition can be more complex, using the logical operators (|| for OR, && for AND),

e.g. (potVal > 500) && (timePass >= 1000).

In the { commands } clause, the commands we want to be executed.

If <condition> is TRUE, <commands > will be executed, if <condition> is FALSE, the arduino will execute the first command after the close curly bracket of the “If” statement.

Time logging function - millis()

The Arduino has a built-in clock, which counts the time from the moment it is turned on (or it is reset). This information is available to us at any point by calling the function millis(), which returns the time in milliseconds (1/1000 sec) it has gone from the activation of our unit. This helps us measure time in our programs, especially in cases where we want to "remember" things.

For example, if we want to check if a certain amount of time has passed since something happened (e.g. a key was last pressed), we can log the event to a time variable and subtract this time from the next one, etc.

For example:

lastPress = millis();

if (lastPress – millis() > 1000) {...}

digitalRead()

Description: Reads the value from a specified digital pin, either HIGH or LOW.

Syntax: **digitalRead(pin)**

Parameters: (pin) the Arduino pin number you want to read

Returns: **HIGH** or **LOW**

Arithmetic Operators

- = (assignment operator)
- + (addition)
- - (subtraction)
- * (multiplication)
- / (division)
- % (modulo)

Logical Operators

- && (AND)
- || (OR)
- ! (NOT)