

Remote Control LED Display

Overview:

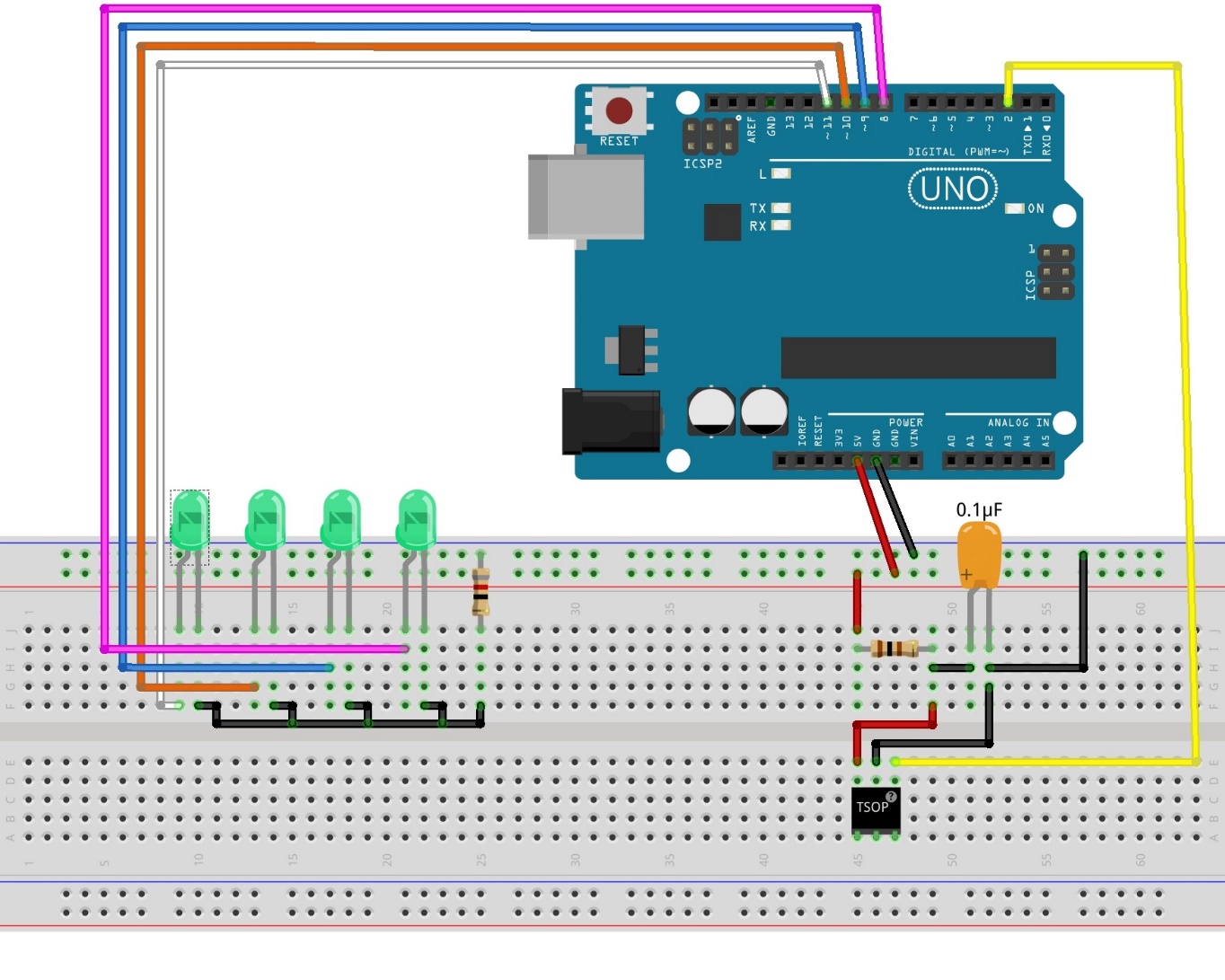
This tutorial demonstrates how to use an IR (infrared) receiver to read signals coming from a TV remote using a UNO and a LED array.

Materials:

This project requires an Arduino UNO (or compatible board), 13 male/male jumper wires, a TSOP3848 (or other compatible) IR receiver, a 4.7μF tantalum capacitor, two 100Ω resistors, and four green LEDs.

Procedure:

1: Wire together the UNO and electrical components as shown in figure 1. Caution: the TSOP receiver should be facing “down” in relation to the diagram with the black dome facing away from the rest of the circuit.

fig. 1

2: Open the Arduino root library folder (“C:\Program Files (x86)\Arduino\libraries” or “C:\Program Files\Arduino or C:\Program Files\Arduino\libraries” for Windows machines) and delete the RobotIRRemote directory.

3: Open the Arduino IDE

4: Go to Sketch>Include Library>Manage Libraries to open the Arduino library manager.

5: Type “IRRemote” into the search field.

6: Install the IRRemote library from the list of options. Do not install the “Robot IR Remote” library from the list as it conflicts with the IRRemote library.

7: Go to [www.rabbitrobots.com](http://www.rabbitrobots.com) and download the IR demonstration program from the “software” tab.

8: Open the IR\_LED\_Sequencer.ino file with the Arduino IDE and upload it to the UNO.

9: Press the power button on the remote to turn the LEDs off and on.

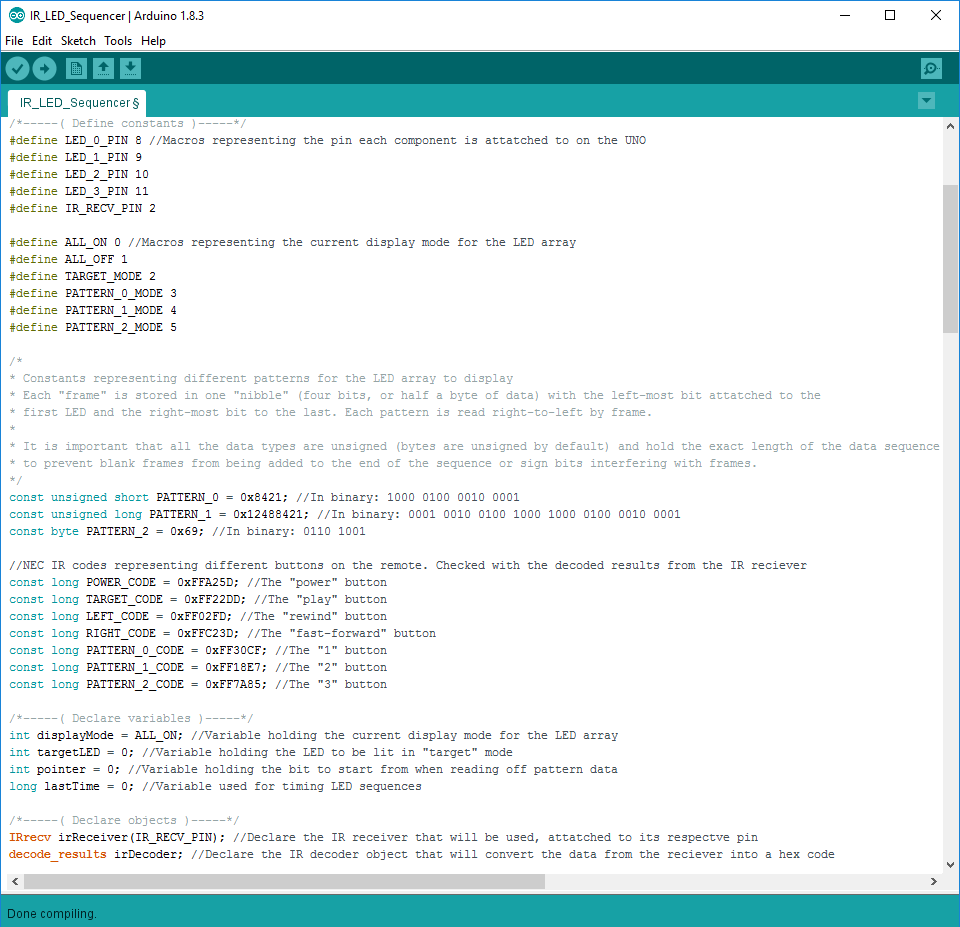
10: Press the play, fast-forward, and reverse buttons on the remote to light a single LED at a time in the array.

11: Press the “1,” “2,” and “3” buttons on the remote to run pre-set LED sequences on the array.

Code explanation

The first section of the code (fig.2) gets and declares the libraries, constants, variables, and objects needed to use the program. The first line “#include <IRRemote.h>” imports the IRRemote library containing the code needed to get and decode signals from the TSOP IR receiver for the program to use. The following lines starting with “#define” create the constants for the program. The first set of constants represent the pins that each component is attached to on the UNO while the second set defines the different display modes for the LED array. These constants values are not variables, but “macros” due to how they are defined in the program and interpreted by the compiler. However, for the purpose of this project, they can just be thought of numerical constants.

Fig. 2



The next part of this first section defines the pattern sequences for the LEDs. How these are stored and used is covered in the “Creating an LED Sequencer” project. The section of constants after the LED patterns holds signal constants for the IR Remote. The IR Remote that we sell uses the NEC protocol for transmitting signals, which a very common protocol for IR devices. Each button is assigned its own special code the remote emits when pressed along with a special “repeat” code of 0xffffff that is sent whenever a button is held down. Each code is encoded in the IR signal by variable spaces in the IR signal sent. A long space represents “1” while a short space represents “0.” See figure 3 for an example.

Fig. 3

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | 0 |  | 1 |  |  | 0 |  | 1 |  |  | 1 |  |  |  |  |  |  |  |  |  |  |  |

The next section of code needed to use the IR receiver is the declaration of the IRRecv irReceiver and decode\_results irDecode objects. The irReceiver object represents the IR receiver and takes the digital pin on the UNO the receiver’s data pin is connected to. irDecode handles the decoding of the modulated signal from the receiver and can be declared without any arguments.

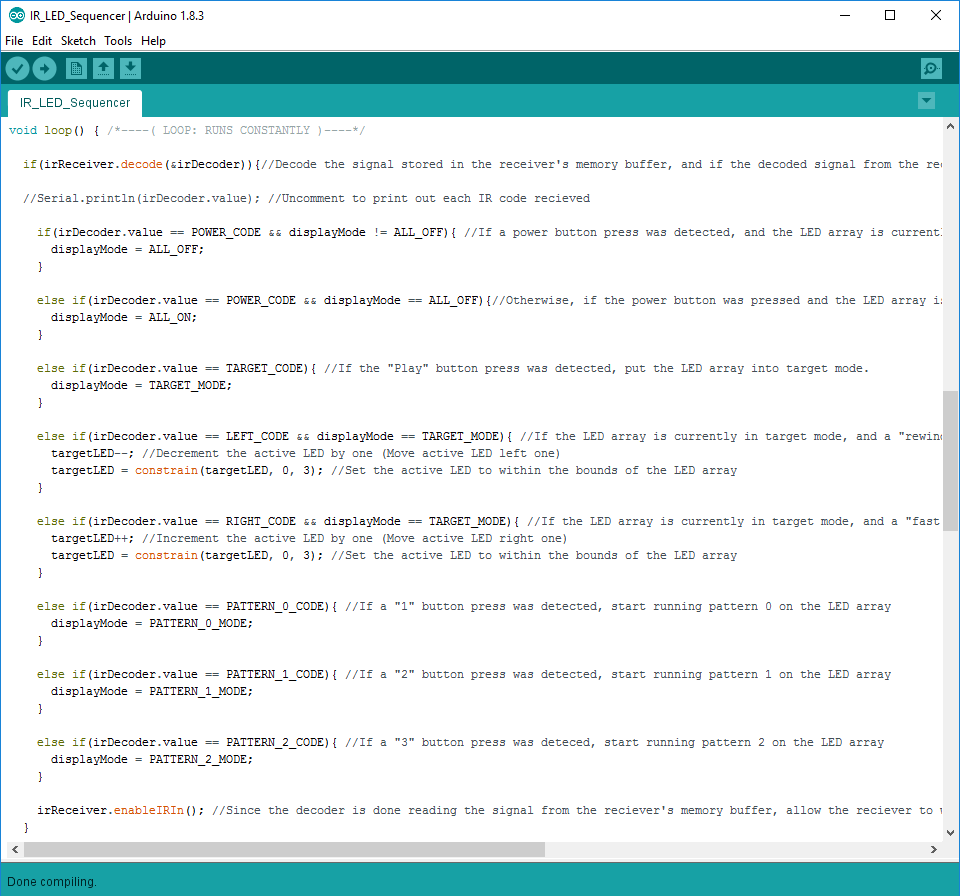
The next section of the code is the setup() function (fig. 4) that runs one time when the UNO starts up. There are only five lines of code in this section, four of which just initialize the LED array. The remaining line, “irReceiver.enableIRIn();” allows the IR receiver to store IR data to memory. The receiver will still be able to read IR data without this line, but it will not record the data anywhere and it will be lost. Therefore, to make the receiver useful, this line has to be called each time a signal needs to be read.

Fig. 4

Once the setup() function ends, the loop() function (fig. 5) begins. The loop() function runs continuously on the UNO and should include most of the instructions needed for the program to run. The first segment of the loop() function is an if statement with the condition irReceiver.decode(&irDecoder)

The IRRecv.decode() function takes in the location of a decode\_results object in the UNO’s memory and returns whether the IR receiver has a valid IR code. The “&” symbol before irDecoder is the reference operator in C, much like “+” is the addition operator. The reference operator “&” next to the name of a variable or object represent its memory address. Entering &irDecoder effectively enters irDecoder’s location in memory so the decode() function can run.

Fig. 5



If a valid code was detected by the receiver, the decoded signal stored in irDecoder.value is then compared to the pre-determined codes with if-statements to see if it matches any of the pre-determined IR codes. If it does, the LED display is updated in some form. Once this checking is complete, irReceiver.enableIRIn() has to be called again to allow the receiver to collect data again because IRRecv.decode() makes the receiver stop storing data in order to decode the signal already written to memory. The remainder of the loop() method deals with manipulating the LED array and is outside of what this guide covers. An explanation of how LED sequences are handled by the program can be read in “Creating an LED Sequencer” project guide.