curilights

Dynamic Wireless Decorative Lights

John W. Peterson

March 6th, 2008
Updated August 2014

Overview

Strings of holiday lights add a nice accent to indoor and outdoor spaces. Many businesses use them to create a festive look. Setting up holiday lights used to be simple. You'd get out the lights in December, set them up for the Christmas season, and then take them down early the next year.

But the light manufacturers wised up, and started selling lights for every holiday they could think of. In addition to Christmas lights, now we have Valentine's lights (pink, red & white), St. Patrick's day lights (green, of course), red white & blue Independence day lights and Halloween lights in orange (or perhaps a ghoulish purple and green). That's a lot of lights to take up and down. For a large business (say, a big department store) changing all these lights involves a considerable amount of labor.

Curilights solves this problem. By using multi-colored LEDs instead of light bulbs, the colors are programmed on the fly. And by using the Lantronix MatchPort b/g Wireless Embedded Device server, no additional network wiring is necessary to control the lights – you just broadcast the desired pattern to change it as often as you like.

Design Overview

Figure 1 below shows an overall block diagram of the system. Each LED has an associated PIC microcontroller (MCU) driving it. All of the LED / MCU pairs are connected to a three wire bus with power (+5V / GND) and a serial data channel. The start of this serial daisy-chain is connected to one of the serial output ports of the MatchPort b/g.

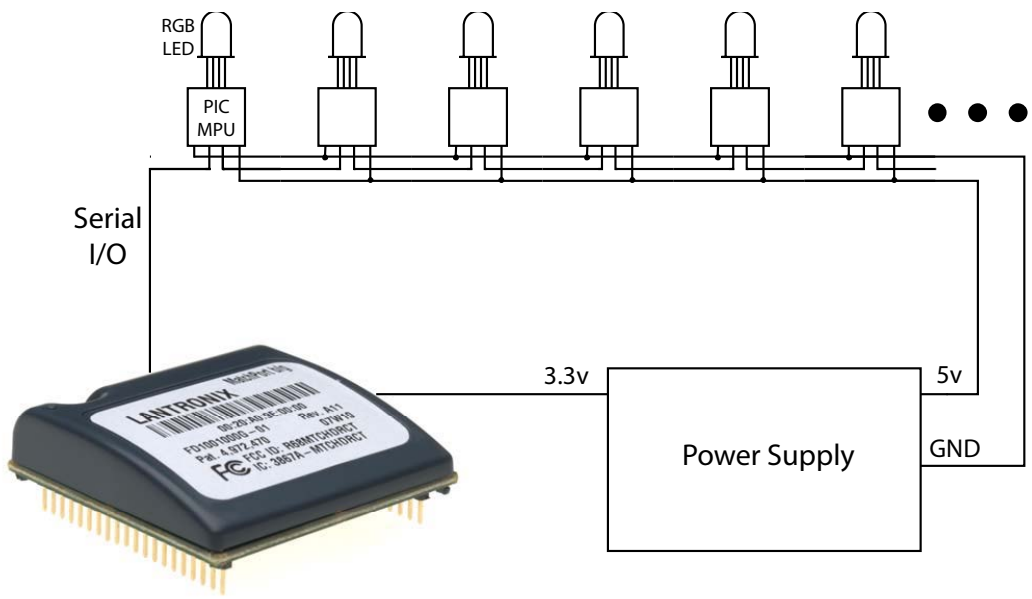

Figure 1 - System Block Diagram

At first it may seem like overkill to have a microcontroller dedicated to each LED, but this design has a lot of advantages. First, because the data is encoded serially, only three wires are needed to connect the string of lights. Trying to control that may color LEDs from a central location otherwise requires a large number of wires (each LED requires four). The local storage and programmability of the microcontroller enables advanced applications such as animating the lights. And the MCU provides excellent facilities for managing the current drive required by the LEDs.

The serial output from the MatchPort b/g is fed to the input of the first LED's MCU. The output of this goes to the input of the second, and so on in a daisy-chain fashion. This structure makes it easy to uniquely identify each MCU, without the overhead of custom programming each MCU (see below).

A power supply converts AC power into the 3.3v required by the MatchPort b/g and the 5v used by the MCU/LED string.

Hardware

For the prototype, the 14-pin Microchip PIC16F688 microcontroller was chosen. Although in theory it's possible to use an eight pin MCU such as the PIC12F series or the Atmel AVRtiny, the larger MCU has a couple of useful features. First, it has an on-board UART hardware for transmitting and receiving serial data. This enormously simplifies the software and timing for serial

communications. Second, the additional memory on board eases the protocol implementation and enables features like animation.

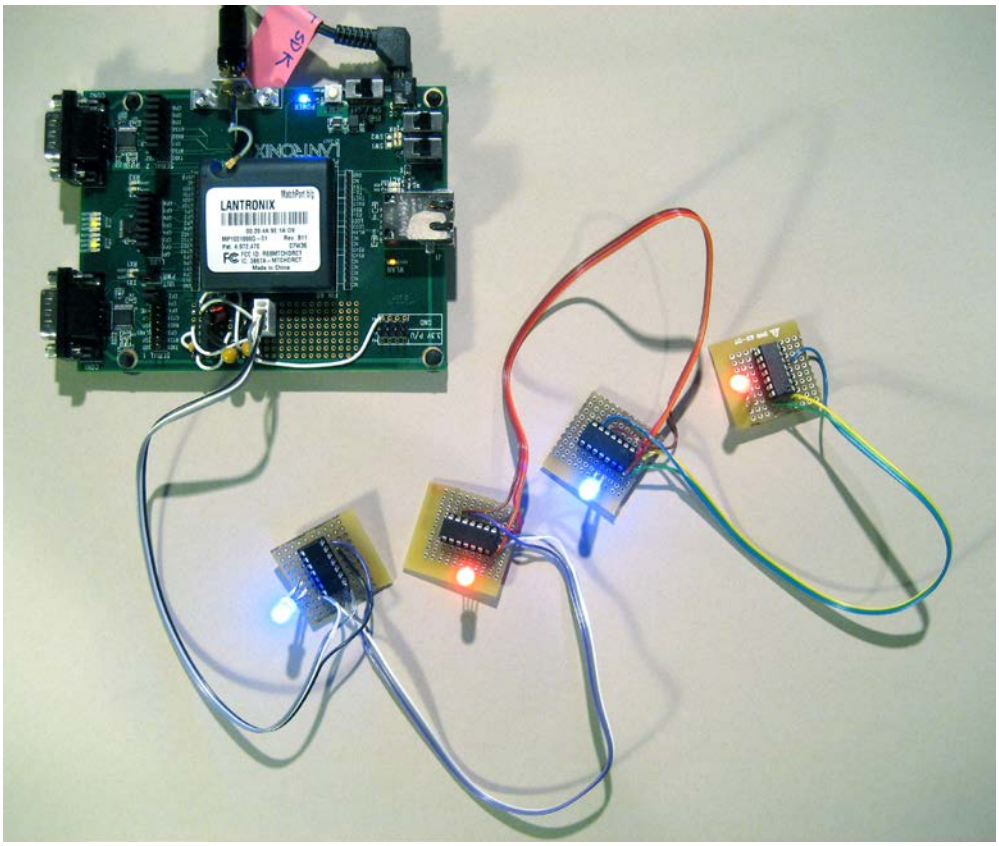

Figure 2 - Photo of the prototype Curilights system

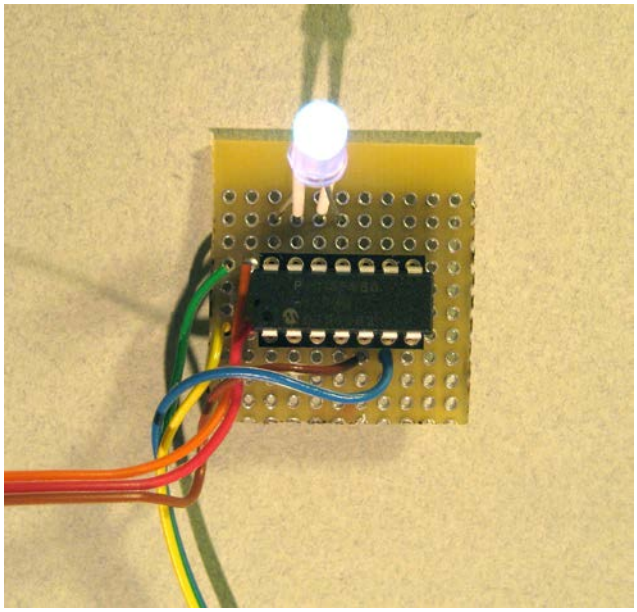

Figure 3 - Close-up of the MCU/LED light

Each light has just the MCU, LED and circuit board, no other hardware is required. The current limiting resistors typically associated with LEDs are avoided by using pulse-width modulation to keep the average current draw per LED below the 20mA rated maximum. Doing this in software adds little overhead and saves dozens of components in the finished product. The MCUs only draw 250µA, a tiny fraction of the 75-80mA used by a fully lit RGB LED.

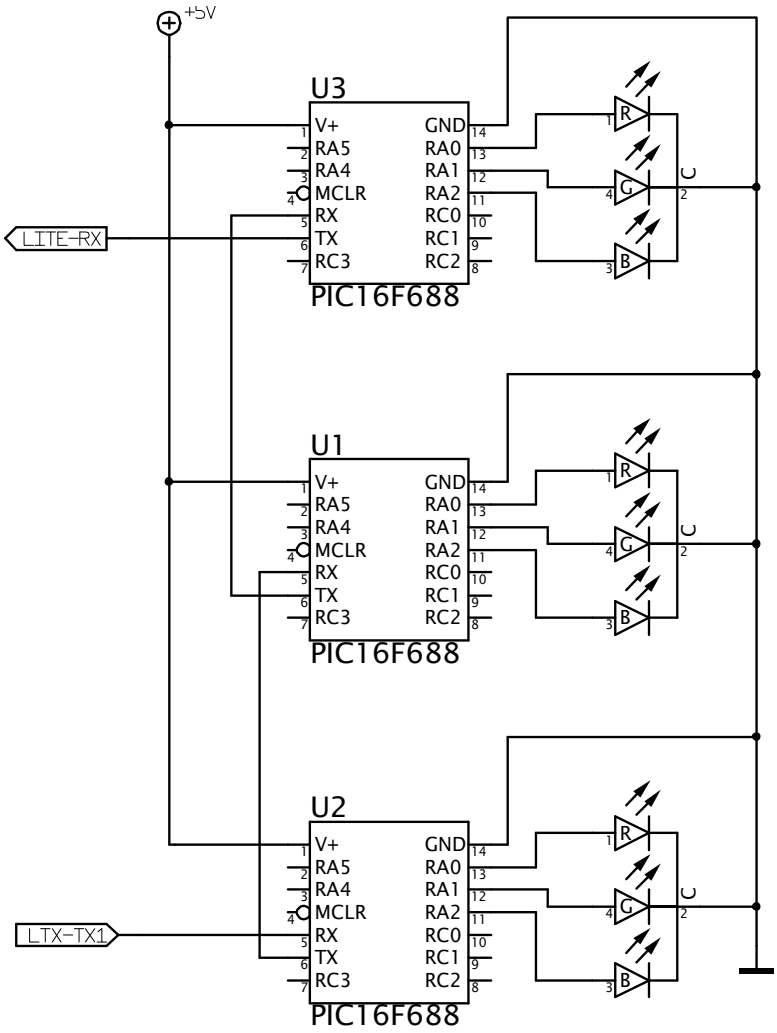

Figure 4 - Light String Schematic

Because the MCU's UART implements the standard 9600 baud serial protocol, no additional hardware is necessary to connect the MatchPort b/g to the string of lights – the MatchPort simply transmits the same protocol the lights use to communicate with each other.

Software

There are two major pieces of software for this project, the low-level microcontroller code and the high level host software. Let's look at the microcontroller first.

The MCU has two functions: driving the LEDs and interpreting the protocol on the serial line. After initialing the MCU's configuration registers, the MCU software enters the main loop controlling the LED's with pulse-width modulation (PWM). For each color LED, a variable keeps track of it's "on" duty cycle. When the PWM counter exceeds this value, the LED is shut off. When the counter reaches it's maximum, it's reset and the process starts over. The LEDs are strobed at approximately 1000Hz.

The control protocol allows four brightness levels for each of the red, green and blue LEDs, ranging from 0 (off) to 3 (maximum brightness). The maximum brightness is not 100% "on", but instead is a percentage found to keep the average current draw of the LED at a safe level (below 20ma) and to keep the output of the LEDs balanced, so 3,3,3 approximates white. Getting this right requires some trial and error to find the appropriate values.

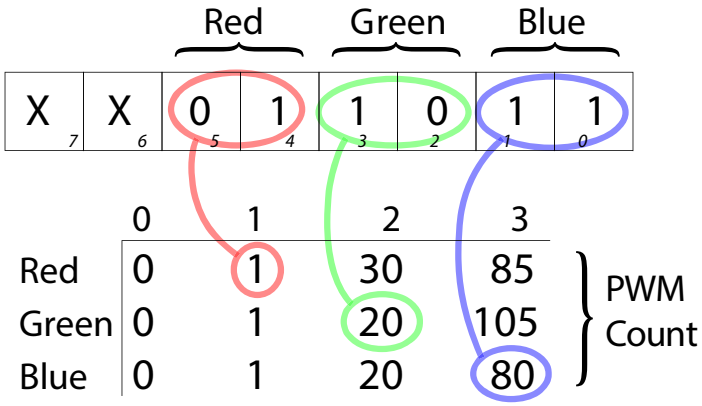

Figure 5 - Table used to convert color byte into PWM count values

The PWM counter value for each of the four levels (for each LED) is kept in a table. The protocol specifies a color as the lower six bits of a single byte, where bits 5:4 specify the red brightness, bits 3:2 specify green, and 1:0 specify blue. Software on the MCU unpacks these bit fields and uses them as an index into the table to find the proper PWM count value. When new color values are received, a flag is set to tell the PWM loop to unpack and use the new values.

The MCU fires an interrupt when a character is received on the serial port. The interrupt service routine (ISR) implements a simple state machine to interpret the protocol commands.

Serial Protocol

The protocol has a simple format. The first byte is a command, followed by up to three parameters. Each of the parameters within angle brackets is a single byte; <color> is the six bit color specification described above, <ID> is the number (starting from zero) of the MCU/LED light.

Name	Specification	Description
Init	I<ID>	Sets the IDs of this and subsequent lights
Color	C<ID><color>	Sets LED at <ID> to <color>
Frame	F<ID><frame #><color>	Sets <frame #> of LED <ID> to <color>
Step	S	Steps one forward in the animation
Time	T<ticks>	Sets the duration of each animation step
Run	R	Starts the animation at frame 0
Halt	H	Stops the animation
Number	N<count>	Broadcasts the light count

The **I**nit command is given as I<0> to the first light. This sets the ID of the first light to zero. This light then increments the ID, and sends that to the next light, giving that one an ID of one. The light after that gets an ID of 2, etc. This way the lights are all given unique IDs. When a command with an <ID> parameter is sent, each light compares the <ID> in the command with the ID assigned by the Init command. If a command doesn't apply to this particular LED, the MCU rebroadcasts the command on the serial output port, where it's fed to the receiver of the next light's MCU. Some commands, such as **T**ime, **R**un and **H**alt, are only run on the light with ID zero.

Animation[1]

Several "frames" of different color values may be stored on each light via the **F**rame command. The first light in the string (with ID = 0) takes care of sending out the **S**tep commands to advance the lights to the next frame at regular intervals to produce animated effects. This is controlled by the **T**ime, **R**un and **H**alt commands.

Since the **S**tep command must propagate from light to light, the following scheme is used to synchronize the animation. First, the **N**umber command is used to broadcast the total number of lights to all the MCUs in the string. When the first light in the string (light 0) begins a frame, it immediately sends a **S**tep command to the other lights, but delays for the number of lights times 2ms (the time to receive and send a step command) before actually changing the light color, to give the command time to propagate down the string. Each subsequent

[1] The animation feature is designed but not fully implemented in the prototype.

light delays for (N - ID) * 2ms, and the last light changes as soon as it receives the **S** command; this way all the lights change their colors at the same time.

Host Software

On the host side, the lights are easily controlled by a short program with access to TCP/Sockets. The host first initializes the lights by sending the **I**<0> command, so the lights give themselves unique IDs. Because the lights are controlled over the network, it makes a natural web application tool. A simple Python script for controlling the lights is included as an appendix.

Fabrication

The current prototype implements a string of four LEDs, these are connected directly to the MatchPort b/g Demonstration board. Each MCU/LED pair was assembled on a small piece of perfboard. The TXD1 output of the MatchPort b/g is connected directly to the serial input of the first light's MCU. A MAX756 based boost regulator was added to the Demonstration board to convert its 3.3v power to the 5v supply used by the string of lights.

For a finished product, the MCU and LED could easily be molded into a small plastic shell. MCUs like the PIC16F688 are available in very small surface mount packages. A control box at one end would house the power supply and the Lantronix MatchPort b/g. A simple configuration port may also be desirable to enable configuring the MatchPort.

The technology is well suited to strings of about 25 lights. Although the protocol supports up to 127 lights, beyond 50 or more, the animation speed becomes limited because of the required serial propagation time. This can be mitigated by using the same MatchPort b/g to control two separate strings of lights on the two separate serial ports provided.

Care must be taken to ensure adequate power is supplied; each LED uses up to 80mA, so a 50 light string uses approximately four amps at 5V (20 watts).

Conclusion

Curilights is a design capable of revolutionizing the way decorative lights are used, opening many new creative possibilities in both interior and exterior design. By using the Lantronix MatchPort wireless technology, these lights may be deployed anywhere existing lights are with no additional network wiring.

Sample Host Control Script

```
#!/usr/bin/python
#
# Simple Python script to demo controlling Curilights from a TCP
# host.  Project WDC143
#

import socket, time

# Must be changed to the IP address of your Curilights
LtxHost = '192.168.0.121'
LtxSerialPort = 10001

def openCurilights():
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((LtxHost, LtxSerialPort))
    return s

# Send a command to the light string
def sendCommand( ltxsock, cmd, id = None, arg2=None, arg3=None):
    msg = cmd;
    if (id != None):
        msg += chr(id)
    if (arg2 != None):
        msg += chr(arg2)
    if (arg3 != None):
        msg += chr(arg3)

    ltxsock.sendall(msg)

#
# Set the colors of the lights.  Each color is assumed to be
# a three digit integer, where each digit is between 0..3, and
# the digits correspond to RGB respectively.  Thus 300 would be
# full on red, and 001 would be blue, and 333 full bright white.
#
# When you pass a list like [300,  30,   3], the first number
# controls light 0, the next 1, etc.  Note: do *NOT* write
# numbers with a leading zero, or else Python will interpret them
# as being in octal!  (i.e., "010" ==> 8)
#
def setColors(ltx, colorlist, sleeptime = 0):
    def colorToChar(color):
        r = color/100
        g = (color % 100) /10
        b = color % 10
        return (r << 4) | (g << 2) | b

    i = 0
    for c in colorlist:
        sendCommand(ltx, 'C',i, colorToChar(c) )
        i += 1

    time.sleep(sleeptime)
```

```
def CurilightTest():
    ltxsocket = openCurilights()
    sendCommand( ltxsocket, 'I', 0 )
    setColors(ltxsocket, [0, 0, 0, 0])
    setColors(ltxsocket, [111, 111, 111, 111], 1)
    setColors(ltxsocket, [ 30,  30,  30,  30], 1)
    setColors(ltxsocket, [300, 300, 300, 300], 1)
    setColors(ltxsocket, [ 11,  11,  11,  11], 1)
    setColors(ltxsocket, [300,  30, 300,  30], 1)
    setColors(ltxsocket, [222,   3, 222,   3], 1)
    setColors(ltxsocket, [111, 111, 111, 111], 1)
    ltxsocket.close()

CurilightTest()

def RotationTest():
    ltxsocket = openCurilights()
    sendCommand( ltxsocket, 'I', 0 )
    speed = 0.15
    for i in range(0,10):
        setColors(ltxsocket, [300,   3,   3,   3], speed)
        setColors(ltxsocket, [  3, 300,   3,   3], speed)
        setColors(ltxsocket, [  3,   3, 300,   3], speed)
        setColors(ltxsocket, [  3,   3,   3, 300], speed)
        setColors(ltxsocket, [  3,   3, 300,   3], speed)
        setColors(ltxsocket, [  3, 300,   3,   3], speed)
    ltxsocket.close()

RotationTest()
```