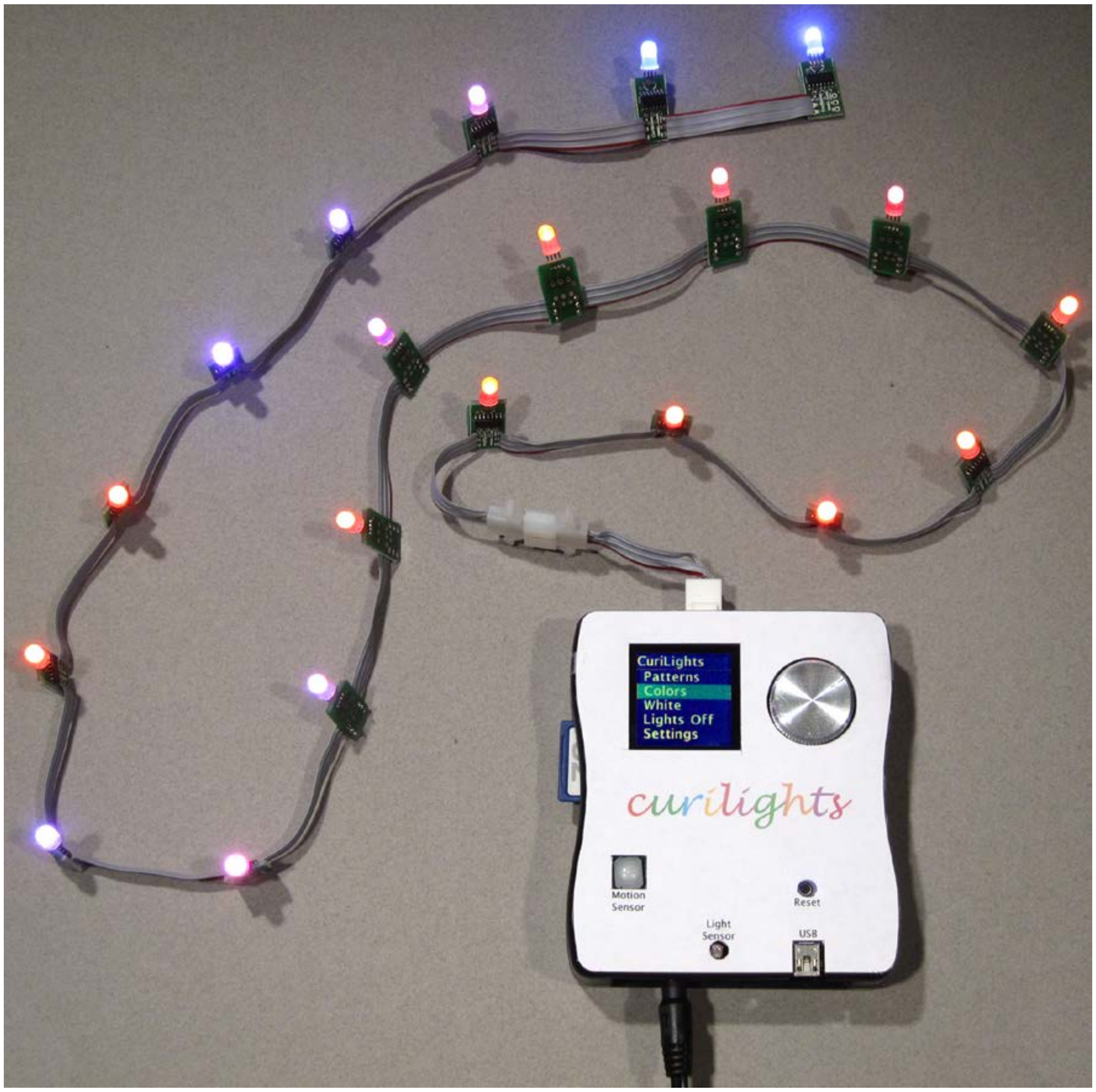
The CuriLights Controller

Project Report

John Peterson
February 2011



Introduction

Color LEDS are opening up many new opportunities for decorative and creative lighting. Using high-brightness RGB (Red-Green-Blue) LEDs each individually driven by a microprocessor, it's possible to create a string of lights with an infinite variety of patterns and designs.

However, with this tremendous flexibility comes a problem: How do you make it easy and convenient to control the lights? You want control of colors, brightness, patterns, even animations, with a simple interface anybody can use.

The system presented here solves this problem. The mbed microcontroller is used at the heart of a very user-friendly system for controlling a string of color LEDs. The controller provides direct feed back with a color LCD display, and uses a simple control knob to provide a sophisticated yet intuitive user interface. It also allows easy access to a catalog of patterns stored on an SD Card.

The system is responsive to the environment. The light string is turned off when it's dark, but a motion sensor turns them back on if it senses movement in the room, allowing their use as night lights.

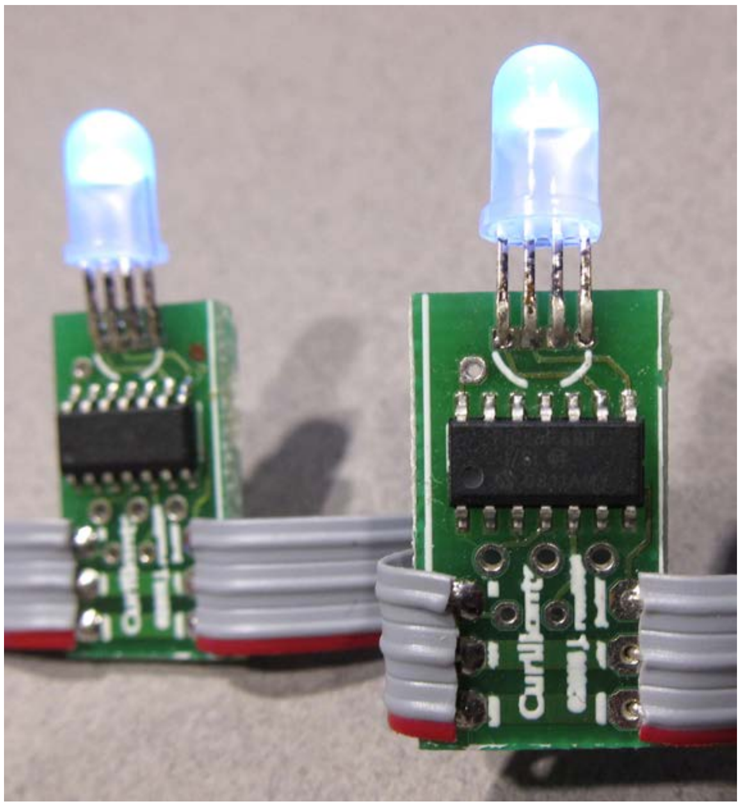


Figure 1 - CuriLights

CuriLights

Before discussing the controller, it's useful to understand the design of the LED light string. The lights in the string are based on the CuriLights design. Each CuriLight has an RGB LED driven by a small MCU (currently a PIC 16F688). A serial line transmits information to the lights in a daisy-chain fashion, where data presented to one light is re-broadcast to the next. When initialized, each light is assigned a unique ID corresponding to its position in the chain of lights. A high level serial protocol is used to transmit unique color information to each light, allowing complete control of every light in the string.

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 design uses the 14-pin Microchip PIC16F688 microcontroller. 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.

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.

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
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.

The <color> parameter is nine bits, three bits each for red, green and blue. To keep the format compact, the ninth bit (the MSB of red) is passed as the top-most bit of the ID. This means a given string of lights is restricted to 127 lights.

Animation

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.

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 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.

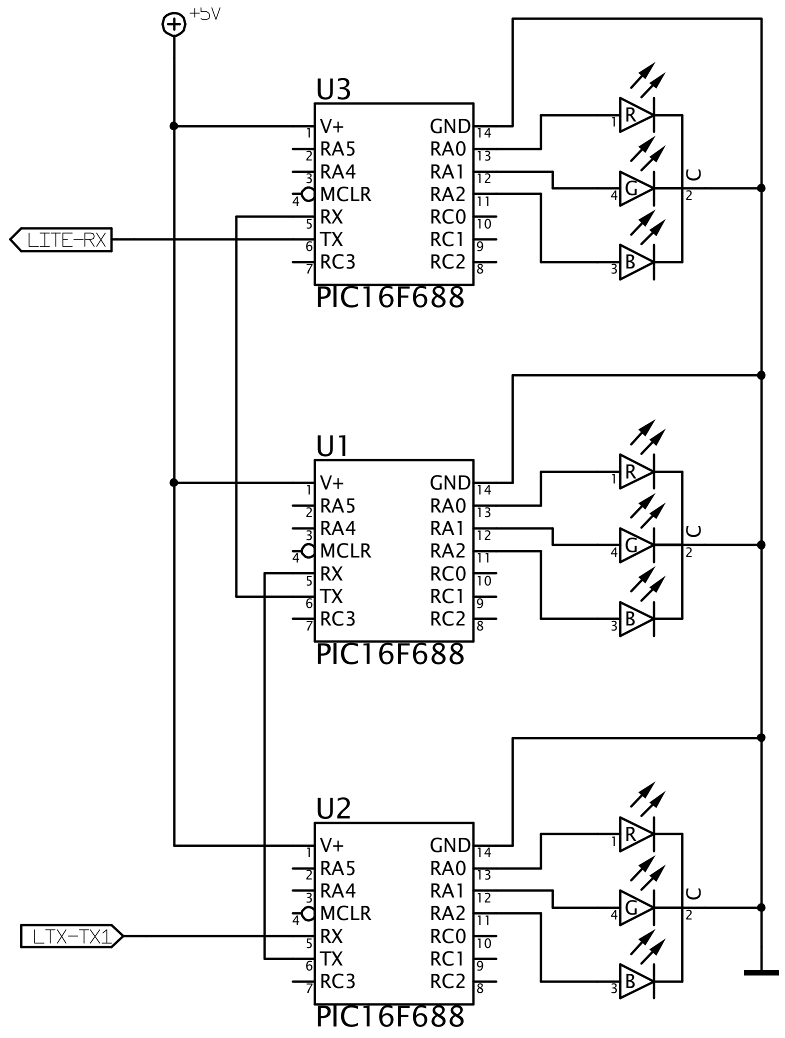


Figure 2 - Schematic of a String of CuriLights

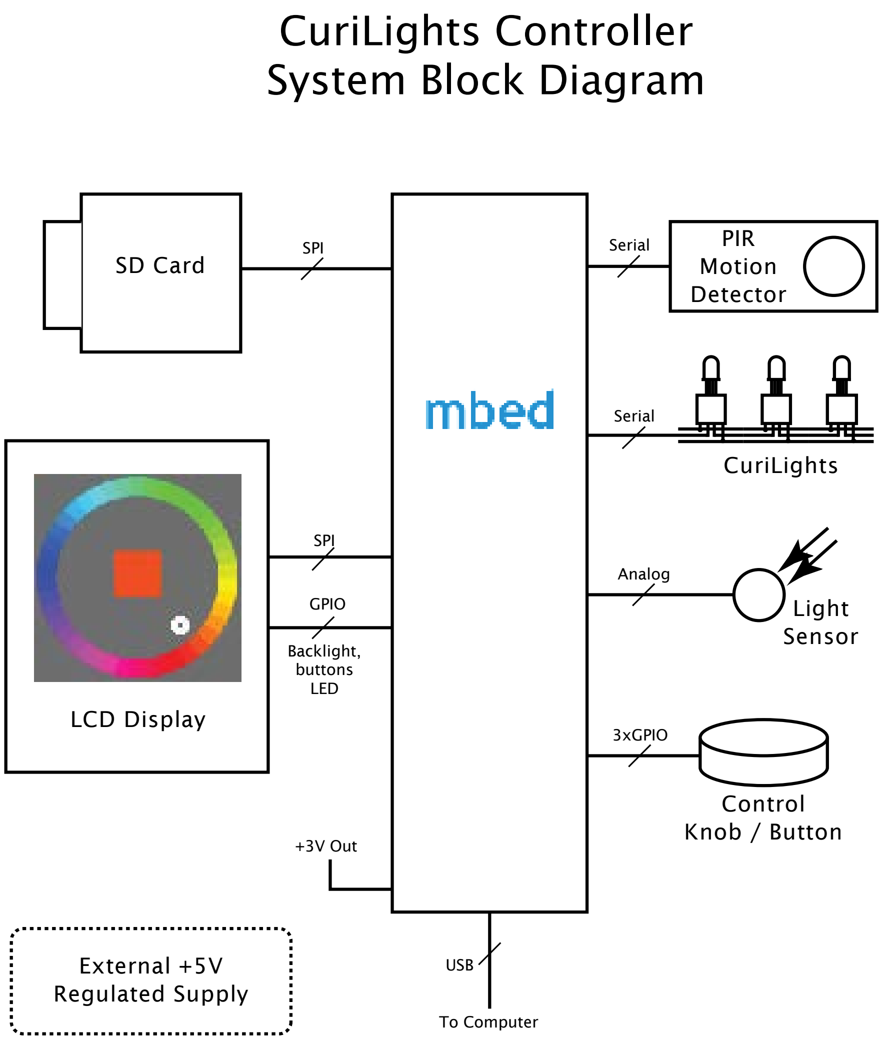


Figure 3 - System Block Diagram

The CuriLights Controller

Overview

One of the benefits to using a MCU like the LPC 1768 is its ability to drive a full color raster display. This makes for a much cleaner and more intuitive user interface. For this system, a 1.5″ Epson color LCD display is used for displaying menus and graphics. The primary interaction with the controller is a knob attached to a rotary encoder. Knobs make ideal controls, because it's easy to quickly turn past unwanted selections, yet easy to get fine control for desired ones. The knob also features a push-click momentary switch useful for confirming selections.

An SD Card is included for storing a catalog of patterns. This provides a straightforward way to update or add to the collection of patterns in cases where the controller is permanently mounted away from easy computer access. The USB port on the mbed module is also used for a couple purposes. First, the serial input connection is routed directly to the serial input on the string of lights, allowing the light string to be controlled directly by an attached PC. Second, new patterns may be downloaded via USB to the disk drive on the mbed. When the mbed is reset, any patterns downloaded on it's internal USB drive are moved over to the SD card for access by the controller.

Finally, two sensors on the controller monitor the lighting environment so the controller responds appropriately. A photocell connected to one of the mbed's analog inputs measures the ambient room light, so if the room is dark, the lights are shut off. A PIR (Pyroelectric InfraRed) motion sensor, however, "wakes up" the lights if motion is detected, making the system useful a night light.

Hardware Details

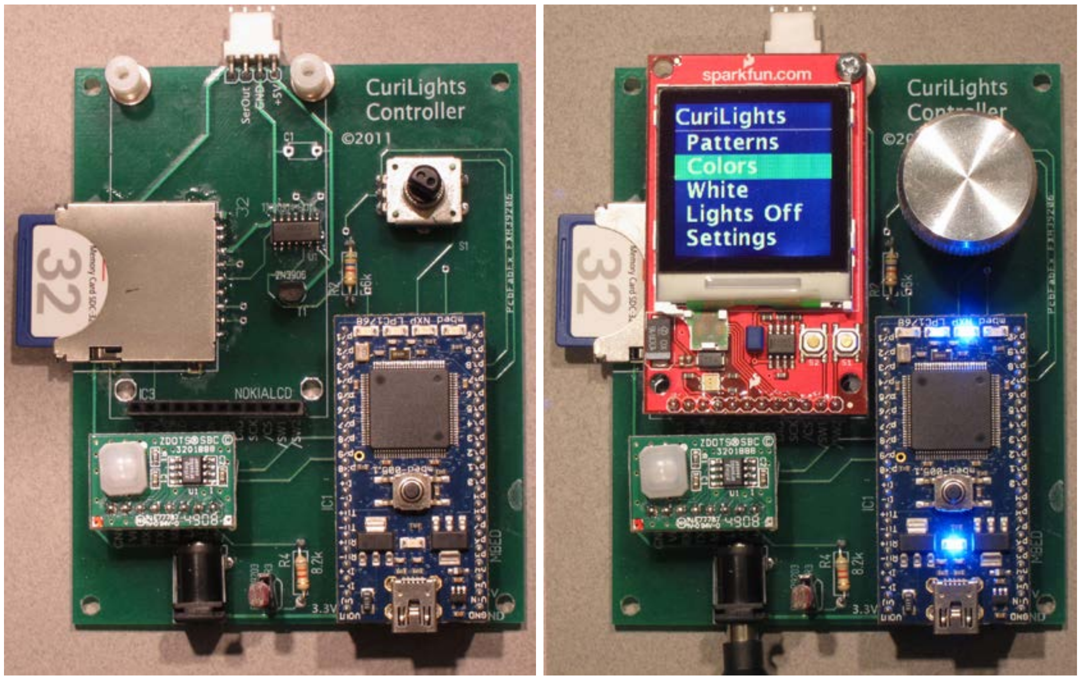


Figure 4 - a) System Board, b) With Display Attached

The heart of the system, of course, is the mbed development board, based on the NXP LPC 1768 MCU. The rich set of on-board peripherals, timers, interfaces and memory all find use in this project.

The LCD display is an Epson LCD originally manufactured for Nokia 6100 series phones. Because of the tremendous volumes phones are manufactured in, these displays are now relatively inexpensive. This display comes mounted on a breakout board from SparkFun electronics. The breakout board simplifies the physical assembly and provides some additional circuitry for powering the display's backlight.

The display communicates with the MCU via an SPI interface using mbed pins 11 and 13, pin 14 is used to drive the display's CS (Chip Select) line (note the display is a "write only" device). The SparkFun breakout board has some additional peripherals including an RGB LED and two pushbutton switches. These are interfaced to the mbed, via GPIO lines. Although the current system doesn't use these, they may become useful as the software design evolves, so it made sense to wire them up.

The display's backlight is driven via one of the mbed's PWM (Pulse Width Modulation) lines. Using a PWM line allows the mbed to gracefully fade the display on and off. Since the backlight requires 5 volts and a significant amount of current, a 2N3906 transistor is used to switch the current for this line. In order to drive the transistor to saturation, the mbed output is run through a TI TXB0104 level converter to take the 3.3V output of the MCU to the 5V level of the backlight supply line.

The motion detector is a Zilog ZEPIR0AAS01SBCG. This is a self contained unit with its own MCU. It communicates to the mbed with a simple serial protocol on pins 27 and 28.

Most of the other components interface very simply to the mbed. The SDCard adapter is connected to the SPI port via pins 5 – 8. The photocell forms a simple voltage divider and is fed into an analog input line, pin 19. The rotary encoder knob uses three digital inputs, one for the pushbutton switch and two for the quadrature – encoded rotation signals. Finally, a serial port on pin 9 is used to drive the CuriLights. Since the CuriLights run on 5V, another channel of the TXB0104 level converter is used to interface to them.

Although the USB port is integral to the mbed module, it also forms a key piece of the overall hardware system. The serial input is used to pass data from any attached computer on through to the CuriLights, allowing the controller to operate as a USB to serial interface. And lighting pattern files downloaded to the mbed 's USB storage are automatically transferred at system start to the SD Card, where they're used by the system.

User Interface

The basic structure of the user interface is a menu using the knob to select items, pushing on the knob when the desired selection is reached. This UI is immediately familiar to anybody who's ever used an iPod® or most any digital camera.

Familiarity with these devices has also raised the bar for expectations of the graphics presented on the screen. Blocky "aliased" fonts and graphics look out of place on modern devices, so a set of anti-aliased graphics was developed to get a clean look.

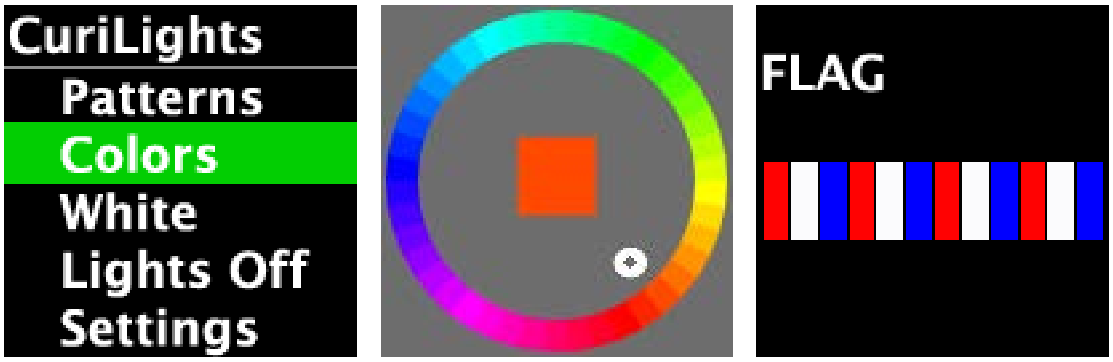


Figure 5 - Sample Screenshots; a) Home menu, b) Color select, c) Pattern select

The home screen is shown in Figure 5a. Choosing the "Colors" option results in a screen like Figure 5b. When you turn the knob, the white cursor moves about the wheel and the center color (and the string of lights) change as you turn it. Clicking the knob button sets the light color and returns you to the home screen. Likewise, selecting "Patterns" in the home menu takes to the screen in Figure 5c. As you turn the knob, the different patterns stored on the SD Card are displayed on the lights, along with the name and a sample of the pattern on the screen. Clicking the knob button selects the pattern and returns you to the home screen.

The "White" menu item sets the string of lights to white, and allows you to choose a brightness level. Clicking "Lights Off" in the home menu shuts the lights off. This changes the menu item to "Lights On", selecting that turns the lights back on.

Finally, "Settings" is reserved for future screens to set various system parameters, such as how long the lights stay on after motion is detected, and enable or disabling the motion and light sensors.

System Software

The software is implemented in C++, using the tools and libraries provided on the `mbed.org` web site. Extensive use was made of existing libraries for SPI and Serial communication, supporting the file system on the SD Card, and responding to digital inputs with appropriate denouncing.

Graphics

The basic support for the LCD screen was based on some code found on the SparkFun web site, but it was substantially modified and enhanced. In particular, routines were added to support rendering anti-aliased graphics. Since rendering quality graphics on the MCU requires the development of a very sophisticated software library, another approach was used. A set of anti-aliased fonts and graphics was prepared on a PC using Adobe Photoshop®. Scripts were used to convert these resources to C++ source code. They are stored as constants, and reside in Flash memory, not RAM.

Some items, like the color selector screen, were developed using a simple JavaScript program to create the color wheel graphic. A screen capture of this was converted into a PNG file in Photoshop, and then a Python script was used to convert the PNG file into C code defining the constant data.

Similarly, an ExtendScript program was used to program Photoshop to generate individual image files for each of the characters used in the display, thus taking advantage of the nice anti-aliased fonts in Photoshop. Similar Python scripts were used to convert these into C code defining constant data for the sprites.

These "sprites" define a mask, where the white pixels define a foreground color, the black ones the background color, and the gray ones define the blend between the two. Thus a compact, monochrome sprite can be used for any combination of foreground and background colors.

This approach offloads the production of high quality graphics to existing PC tools well suited to the job, rather than trying to re-create them in the constrained MCU environment.

UI System

The user interface is implemented with a set of classes to help organize and simplify the design. The `PushKnobUI` class manages the hardware controls for the UI, i.e., turning the knob and pushing it. All classes using these devices descend from this class, but only one "owns" the controls at a time. This ownership is controlled by the `SwitchTo` method. The menu classes and the `LightController` class descend from `PushKnobUI.` Since the `PushKnobUI` class monitors the user activity, it also takes care of turning on the LCD screens backlight. Any activity calls a `Wake` method, which starts a timer. Other calls to `Wake` while the timer is running reset it, but when the timer expires the `Sleep` method is called to fade out the screen's backlight.

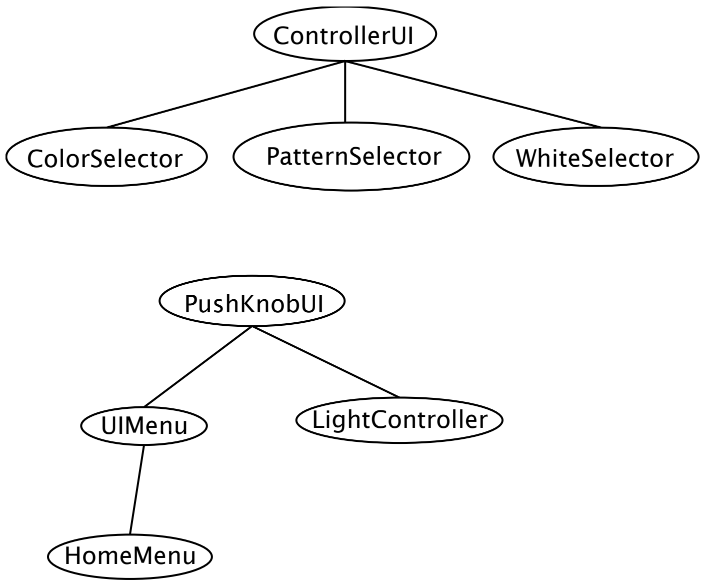


Figure 6 - UI Class Hierarchy

The menus descend from the `UIMenu` class, which implements all the screen activity and selection. When the `KnobPushed` method is called, the menu subclass acts upon the current selection.

The three screens that actually set the light patterns are managed by the `LightController` class. This class has three possible controllers, one for Patterns, another for Colors, and a third for White (brightness). These all descend from the `ControllerUI` base class. When the home menu selects one of these, the `LightController` sets an active Controller, and routes all of the activity (knob movement, turning the lights on or off, etc.) through the active controller.

These user interface classes are interrupt-driven; they are invoked by GPIO interrupts when the knob is turned or the button is pushed. Simply instantiating the `HomeMenu` in `main`() is sufficient to activate them. The light and motion sensors, however, are polled in a loop at the end of main(). These call into the `HomeMenu` if they detect the lights should be turned off or back on.

Pattern Files

Pattern files are used to store various patterns displayed on the lights. These are downloaded to the on-board storage via USB, or placed on the SD Card (files on the `mbed` storage are moved onto the SD Card at system reset).

Since the CuriLights support a nine bit color specification (three bits each for red, green and blue), a convenient shorthand for representing light colors is a three digit number, where the first (hundreds) digit represents red, the second (tens) digit represents green, and the last digit represents blue. Pattern files are simply lists of these numbers, one per line, for each of the lights. If the pattern has fewer lines than there are lights, the pattern is simply repeated for the rest of the lights. For example, to have string of red, white and blue lights, the pattern file contains:

```
700
777
007
```

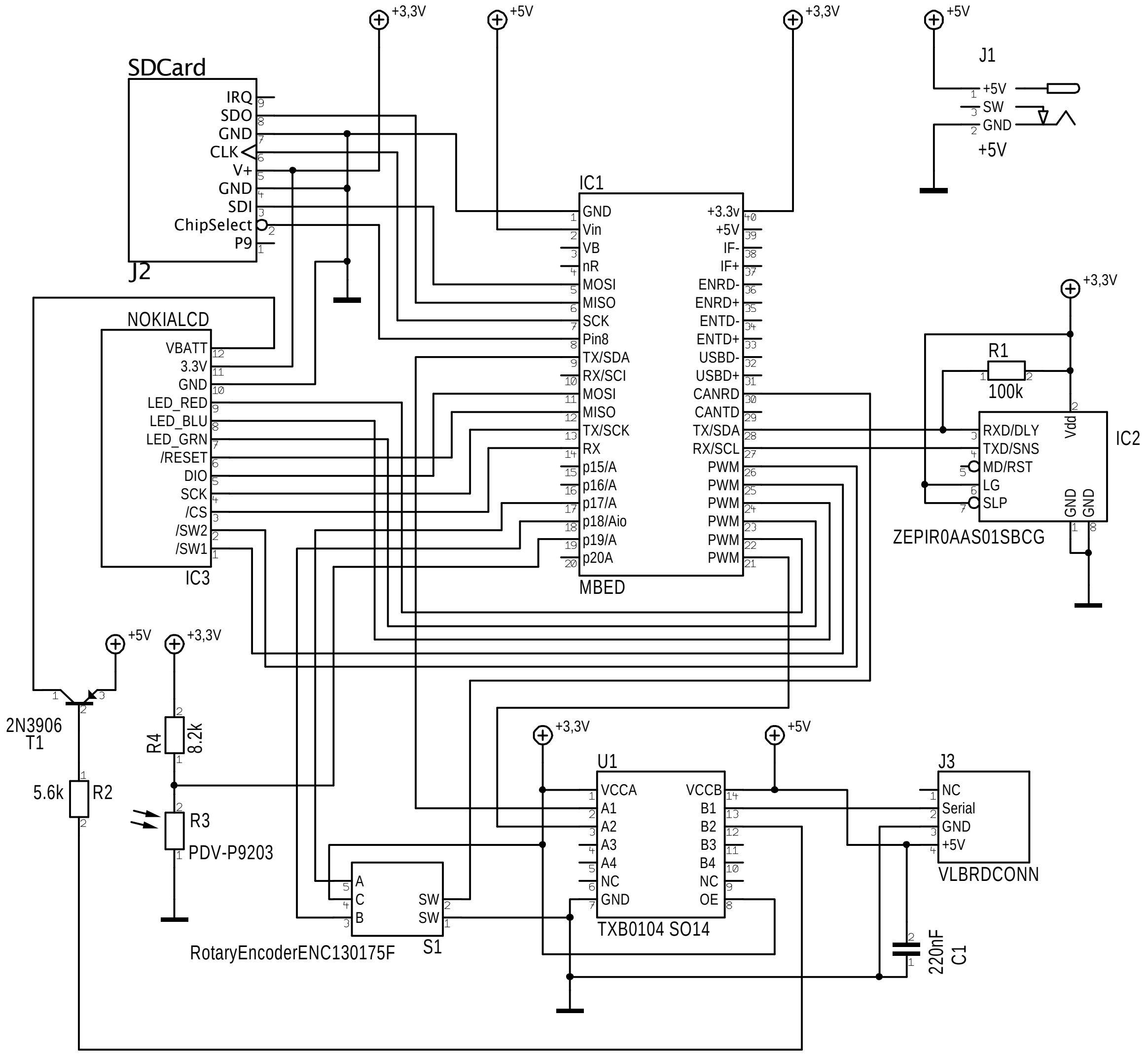
As a tool for creating more intricate patterns, a script was written to convert PNG image files into pattern files. This allows you to use tools like Photoshop as a pattern editor, by creating one-pixel high images.

Conclusion

The CuriLights Controller provides an easy to use, simple control for getting the most out of a string of colorful LED lights. It works well as both a stand-alone light string controller, as well as attached to a nearby computer.

The LPC1768 provides enough resources for a high-quality color user interface; meeting the expectations you have in a world where every new gadget has a bright color screen. The mbed system worked extremely well for this project. The tools and the libraries were easy to learn, and both software and hardware development progressed very smoothly.

Schematic


SDCard
J2
IRQ
SDO
GND
CLK
V+
GND
SDI
ChipSelect
P9
+3,3V
+5V
IC1
MBED
GND
Vin
VB
nR
MOSI
MISO
SCK
Pin8
TX/SDA
RX/SCI
MOSI
MISO
TX/SCK
RX
p15/A
p16/A
p17/A
p18/Aio
p19/A
p20A
+3.3v
+5V
IF-
IF+
ENRD-
ENRD+
ENTD-
ENTD+
USBD-
USBD+
CANRD
CANTD
TX/SDA
RX/SCL
PWM
J1
SW
GND
NOKIALCD
IC3
VBATT
3.3V
LED_RED
LED_BLU
LED_GRN
/RESET
DIO
SCK
/CS
/SW2
/SW1
R1
100k
IC2
ZEPIR0AAS01SBCG
RXD/DLY
TXD/SNS
MD/RST
LG
SLP
Vdd
2N3906
T1
5.6k
R2
R4
8.2k
R3
PDV-P9203
RotaryEncoderENC130175F
S1
A
C
B
SW
U1
TXB0104 SO14
VCCA
A1
A2
A3
A4
NC
VCCB
B1
B2
B3
B4
OE
J3
VLBRDCONN
NC
Serial
220nF
C1