Using the Raspberry Pi to Control Decorative Nightlights

I am now retired and I have a single-family home in Long Beach, California, that not only looks beautiful during the day, but also looks quite lovely at night. The decorative nightlights I use consist of four LED hardscape lamps on 12-inch risers, six LED spotlights for the trees, and five halogen globes for the flowerbeds. I currently use the built-in timer control in the power unit to turn the nightlights on at dusk and off about eight hours later. However, the time of dusk varies throughout the year and I need to periodically adjust the timer control or the nightlights will turn on too early or too late: there is not a best setting. It would be nice if the nightlight would stay on longer during the winter months, but that would require yet another manual adjustment.

The Raspberry Pi is an affordable computer capable of hosting the Linux operating system, compiling, linking, and executing C code, connecting to the Internet via Wi-Fi, and servicing a GPIO port. It is the perfect solution that can calculate sunset and sunrise, and turn a GPIO port on and off respectively. A rather elementary circuit consisting of an optoisolator triac driver, a triac, and a few resistors is all that is needed to interface a Raspberry Pi GPIO port to the decorative nightlight power unit. This document, circuit, software, and pictures of the completed project may be downloaded from the lightcontrol directory at https://www.mediafire.com/folder/j5i7edf6ezqo7/Public.

In order to calculate sunset and sunrise for one's location on earth, one needs to know one's time zone, latitude, and longitude. I prefer to use the "official" value for the sun's zenith, or 90 degrees 50 minutes, and I want the turn-on time to be the time of sunset plus 18 minutes and the turn-off time to be the time of sunrise minus 36 minutes. The algorithm to calculate sunset and sunrise from this information is readily available from the Internet, and the Raspberry Pi's C code compiler and linker have the required library routines and for handling transcendental numbers and using trigonometric functions.

Even though my University studies were in Electrical Engineering, I managed to direct my professional career towards real time software development, specifically in the area of tactical radar systems. I designed and developed very, very high-speed data collection systems for tactical radar and sensor data, and I wrote all the processing software, data handling algorithms, direct I/O data management, and user interfaces. I believe and maintain that ANSI C is the only language, other than assembly, capable of handling the quantity of data and data manipulations I required for real time processing. Of course, C code on the Raspberry Pi can quite nicely handle the requirements set forth in calculating sunset and sunrise.

When the Raspberry Pi powers on and boots the Linux operating system and connects to the Internet, it launches the lightcontrol program. This power on may be due to an intentional power on or due to a power on after a power outage. Therefore, the first time the software traverses the main processing loop, it needs to figure out if the current time is before turn-off time or after turn-on time and turn the GPIO port ON, or neither and

turn the GPIO port OFF. It then calculates the number of seconds to the next event, sunset or sunrise, and goes to sleep for those many seconds. Normal processing will then occur when the task wakes up, turn the GPIO port ON or OFF accordingly, and calculate the number of seconds to sleep until the next event. And that is basically what this Raspberry Pi will do for the rest of its existence. Pretty cool, don't you think?

The GPIO port interface circuit is courtesy of Fairchild. I chose to use the MOC3031 optoisolator triac driver because it is inexpensive, readily available, and it incorporates an internal zero-crossing circuit to start the Triac conducting at or near zero volts in the AC cycle. Pins 1 and 2 connect to an internal GaAs infrared emitting diode, so direct connection to the Raspberry Pi using R1 is possible and practical. Approximately 5 mA is drawn from the GPIO port when it is at logic 1. The Q4008L4 Triac is inexpensive, readily available, can handle up to 8 amps, and is packaged in an isolated TO-220 case. It is important that the case is isolated from the inputs because their voltage levels and currents could cause difficulties in choosing and mounting a required heat sink. R4 and C1 effectively snub the Triac from an inductive load. If the inductive load is very high, say with a power factor less than 0.5 for example, change R4 to 360 ohms. The values shown for R2 and R3 are to be used when the input Hot is 115 VAC. The values shown in parenthesis for these resistors are to be used when the input Hot is 240 VAC. Certainly most any similar Triac driver and Triac may be substituted.

The software I wrote has a lightconfig program and the lightcontrol program. The lightconfig program is used to input the selected GPIO port, sunrise offset, sunset offset, time zone, zenith, latitude, and longitude. These values are saved to a configuration file read by the lightcontrol program. Every time the lightcontrol program wakes up several entries are made into its log file. You may at any time view the contents of that log file. Whenever the Raspberry Pi boots, a new log file is created so that previous entries are never lost. Create a directory to save the software. Download the lightcode.tar file and execute the UNIX command "tar xvf lightcode.tar". Edit the parameterDefs.h file and change the value for LIGHTCONTROL_PATH to the path of the directory you just created. Compile and link the software using two UNIX commands: "make" and "sudo make root". Edit the /etc/rc.local file (you must be root or use sudo) and add the following lines before the "exit 0" line (which must be the last line of this file):

```
#
# Launch the lightcontrol program.
lightcontrol directory path/lightcontrol &
```

It is critical that you include the ampersand or the boot process will wait forever for the rc.local file to complete its processing. It is not necessary to "sudo" this command because the executable file already has root privileges (the second "make" did that).

I built my lightcontrol hardware into a 3.5"x2.0"x6.0" plastic box. There are a couple of pictures showing where I placed all the components. The Raspberry Pi is under its power supply. I built this before I knew anything about the Adafruit Perma-Proto HAT (no EEPROM) which might have made the project simpler and more compact to build. But I

wonder how safe it would be to bring 115 VAC or 240 VAC to the HAT. The Adafruit Ultimate GPS HAT would provide the time zone, latitude, and longitude automatically as well as provide a convenient place for the electrical components. The possibilities seem almost endless. I hope this project inspires you to make use of the Raspberry Pi in a useful and productive way. I think it's time to dust off my American Flyer trains and see what I can do with another Raspberry Pi!