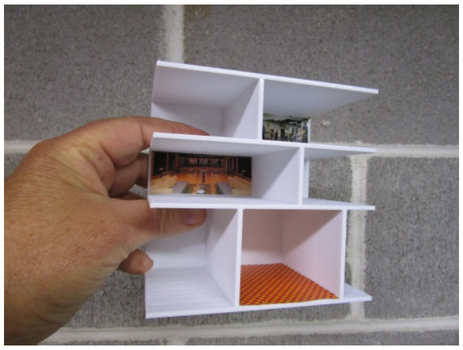
Project 3 - Lighting a Building and Adding Sparkles to Water

July, 2020
David Ackmann – Gateway Division NMRA

This project is all about using an Arduino microprocessor to randomly turn LEDs on and off in several similar environments: first as lighting inside of a building, and next as sparkles in a water feature. Additionally, for the first time we are adding audio to a project in the form of the sound of moving water to the water feature.

Wait a minute! How are these projects similar? Well, lights in a building are turned on when someone is inside a room. They stay on for a while then are turned off. They do so in a random fashion, as people come and go through different rooms, and they stay on for random times. And sparkles in a water feature also come and go randomly, just much more quickly. So it might be possible to control both lights in a building and sparkles in water with the same Arduino code, but with different timings. Two projects in one? Let's see if it can be done.

I'll start with the building lighting project. I will be using a small DPM building I already had on my layout ("B. Moore Catalog Showroom", Walthers 243-10400, $21). The electronics will require an Arduino Nano from LAVFIN (ASIN: B07G99NNXL), a Nano expansion board (AMAZON ASIN: B07MGVC18K), a 12V power supply and five pieces of white "LED strips", (AMAZON ASIN: B07SZMBPFT; the second letter in the ASIN is a zero, not an "O", $13 for a roll of 120 LEDs).



I am not the first model railroader to make random lighting in the rooms of a building (Geoff Bunza did a nice treatment of this in Model Railroad Hobbyist about 5 years ago), but I do think this project has some unique features. Instead of using styrene to build the rooms, I used "TinkerCAD" software and a 3D printing service bureau ("makeXYZ.com") to make the interior. Through careful measuring and some trial and error, I made an interior which fits snugly, yet can be easily removed for modifications. You can see that I have already glued some photographs into the interior, and eventually all interior walls will have pictures captured from the web for their walls and flooring. Secondly, the lighting is controlled by the Arduino in such a way as to give controlled, yet random illumination. More about that later.

Each LED can be separated from the next one with a scissors. The LED is contained in a “pad”, and the back of the pad contains soldering points for its anode and cathode. The rounded top of the pad is marked with a plus and minus sign, indicating which soldering point is which. The soldering points are tiny and the LED tends to roll when upside down, so I used masking tape to secure it to my soldering plate. I started by tinning each of the soldering points with a dab of solder. Using 28 gauge wire, I tinned the ends of the wires and soldered a white wire to the positive point and black to the negative point; the white wire is about 10” long, and the black about 8”. Soldering onto the small pads using light wire is tedious, so take your time. Once attached, I temporarily covered the LEDs with a piece of masking tape and painted about 2” of each wire with a white spray paint so they would be less visible on the ceiling.

Into the back wall of each room I drilled a 1/8” hole just below the ceiling of each room to allow the wires from the LEDs to pass through the back wall.

The back of the LED pad is covered with a thin plastic sheet protecting some adhesive; I removed the protective covering and placed the pad firmly on the ceiling. The adhesive is rather weak, so sometimes I added a good size drop of CA glue before pressing the LED into position. The LEDs put out a nice white light. I did not try to paint them a different color.

One of the nice things about these LEDs is that they run off 5V, such as is supplied by the Arduino’s digital output pins, but I did use a 10Ω resistor just in case the LEDs were a bit out of specification, But they are really brighter than I would like, and they consume about 40 milliamps of current, meaning only 5 could be running at full power before the 200 milliamp current limit for an Arduino is reached. How do I know that? Let’s do the math.

The specifications for these LEDs don’t say how much current each bulb consumes, but we can calculate it. The specification does say that a meter of bulbs, 60 of them, consume 12 Watts. Dividing 12 Watts by 60 bulbs, we calculate that each bulb consumes 12/60 watts, or 0.2 watts. Each bulb is running at 5V, and since Watts=Volts*Amps, we can calculate the current that each bulb consumes by dividing 0.2W by 5 volts, which tells us that each LED consumes 0.04 amps, or 40 milliamps. Each Arduino pin can deliver up to 40 milliamps, so we are right at the limit for the pin, but remembering that the total current for all Arduino pins is limited to 200 milliamps, it means that if any more than 5 LEDs were running at full power at the same time, we would be consuming a more than 200 milliamps, and we would overload the Arduino.

But what would happen if we added dropping resistors to the LEDs? We could dim them a bit (they are pretty bright to begin with) and have more of them. We know from Ohm’s Law, that Volts divided by Amps equals Resistance, so 5V/0.04A = 125Ω; this is how much resistance these LEDs are supplying at full power. If we added in another 125Ω of resistance in the form of a 125 ohm resistor, we would cut their current requirements in half, which would bring the total current draw to 200ma or less; we could run 10 of them and still be safe! The point is that these guys are about twice as power-hungry as standard LEDs, so if you need more than 5 in any application, you will need to reduce their current draw with resistors, or run multiple Arduinos. Whatever you choose, make sure your “wall wart” can handle the load as well.

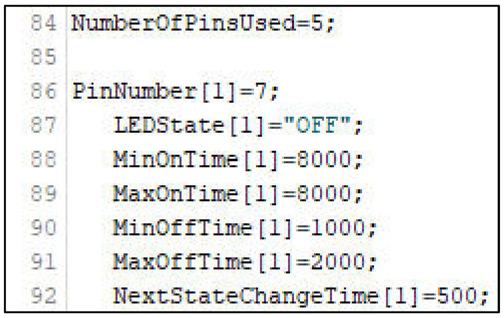
Even though I could run all 5 lights at full power, I went back and added some dropping resistors as an experiment. I discovered that even when I used a 3.9KΩ resistor, the light was still visible, but I was most happy when I used a resistor of 1KΩ or smaller. In the end, I decided to use the 10K resistor on the bottom floor (the motorcycle shop), a 220Ω resistor to the left room on the middle floor (the bowling alley), a 510Ω to the right room on the middle floor (the store room), a 1KΩ in the left room of the top floor (the bar), and a 1500Ω to the right room of the top floor (the ballroom). Doing the math, a 220Ω resistor will drop the current draw to under 15 milliamps, while still providing more than enough light for a small room, and larger resistors reduce the current draw even further. Feel free to experiment yourself.

I took the ground wires from the top floor LEDs, wrapped them together and soldered on a “pigtail” wire. I wrapped the joint in heat-shrink and soldered a spade connector to the pigtail. I did the same to the ground wires from the second floor and bottom floor. I attached both spade connectors to the ground point of the barrier strip. I tinned the ends of the white wires and inserted them into sockets D2 to D7 of the expansion board. I used nylon ties to keep things neat.

Yes, I could have used standard LEDs that require less current, but they would not have been as bright, they would have been harder to attach to the ceilings of the building, and I wouldn’t have learned something new. Your choice.

The final step of the hardware installation is to connect a power supply to the barrier strip, and then on to the expansion board. I prefer a 12VDC 2A “wall wart” connected to a barrier strip, and from there to the VIN and Ground sockets of the expansion board. And since bigger power supplies cost about the same as smaller ones, I buy power supplies with enough capacity to power 10 Arduinos at once (or perhaps power a few ancillary components such as MP3 players, Audio Amplifiers, Motors, or “whatever”). But for this application, if you have a 9VDC, 500ma power supply lying around, you should be OK.

The code for lighting the building is found in the “Project 3” area at https://daackm.github.com. As before, click on the entry and copy and paste the code into a new sketch, overlaying the default “setup” and “loop” functions. Click on the download button and watch the lights come on and off in a random sequence.



```
84 NumberOfPinsUsed=5;
85
86 PinNumber[1]=7;
87    LEDState[1]="OFF";
88    MinOnTime[1]=8000;
89    MaxOnTime[1]=8000;
90    MinOffTime[1]=1000;
91    MaxOffTime[1]=2000;
92    NextStateChangeTime[1]=500;
```

But how do these lights go on and off at random? Take a look at lines 86-92 where you can see the "parameters" for what I call a "light block". Line 86 says the for light block number one, the positive wire of the LED goes to pin D7 of the Arduino. Line 87 says that when the Arduino passes the initial Power Up test, this particular LED is off. Lines 88 and 89 say that the minimum on time is 8000 milliseconds, or 8 seconds, and the maximum on time is also 8 seconds; these values are usually different from each other and the actual time on is some random value between the minimum and maximum, but for this room I wanted them to be the same. Lines 90 and 91 give the minimum and maximum values for the off time. Line 92 gives the elapsed time when the state of the LED will next change. There are blocks for up to 9 LEDs, and the code is written to cause them each to change as desired. Additionally, line 84 specifies how many of the blocks are actually used in any particular application.

Further down in the code is a function called CheckStatus (too big to include here). Each time through the loop function, CheckStatus is called for each "light block". CheckStatus looks to see if that light block's NextStateChangeTime has elapsed (kind of like a timer going off) and changes its LEDState. If the new LEDState is "ON", it calculates a time that it should remain on by generating a random number between the MinOnTime and MaxOnTime, and then adds that number to the current value of the "millis()" clock, to calculate a new NextStateChangeTime. If the new LEDState is "OFF", well, you get the idea.

To see the animation in action, visit http://daackm.github.io/Trailers.html and click on the "Randomized LED" entry.

So, upload the code to the Arduino, cover the interior with the building's shell and enjoy.

Now, on to the water feature; this was the first water feature I had ever done. Another hobby of mine is woodturning on a lathe, particularly pens. Some of mine are indeed wood, but some are polyester resin ("PR"), and some are even combinations of wood and PR. Since I had PR on hand, and didn't want to unnecessarily buy something else, I ran a few tests and discovered that I could create a sparkling water feature with standard LEDs, and the heat generated during the curing process would not harm the LEDs. I was off and running!

I am going to describe the "final process" that worked for me to create my canyon, but if you are a "water features newbie" as I was, you might want to make some trial runs on 6" by 8" features before committing to a larger form. It's a learning process.

I used Woodland scenic “shaper sheet” to make a small hillside canyon, and applied plaster to stiffen the form. I let it dry for 24 hours and then gave it a bit of gray/black color, but left the area where the water would flow unpainted for the time being. In the white waterfall area I painted it with different colors of Woodland Scenics, Deep Blue, Navy Blue, or whatever looks good to you, but probably placing the darker colors in the middle of the stream.

After drying, I drilled nine 1/8” holes in the blue area to accommodate the LEDs, then covered the holes securely with several patches of green “Frog Tape” (painters edging tape). I turned the form over and inserted standard 3mm LEDs, pre-wired and with 150Ω dropping resistors, into the holes allowing the Frog Tape to hold them in place; my wife thought they were too bright, so you might want to “dry fit” some into the form and see if larger resistors look better to your eyes (to me,they didn’t seem to lose much intensity when subsequently covered with resin). I experimented with different colors of LEDs, standard white and blue, clear ones painted with Woodland Scenics blue paints, and Tamiya clear paints, gray, and with the exception of the Tamiya clear I thought none of them were that much better than the clear and blue standard 3m LEDs, but then I am substantially color-blind. To the back of the form I used a generous amount of clear silicone caulk to the LEDs to seal them into place, making certain that they extended fully into the form. I let the silicone cure overnight. In addition to holding the LEDs in place, the silicone served to prevent the resin from leaking through the form.

Clear Polyester Casting Resin
Resina de poliéster transparente para coladas
Liquid Plastic
Plástico Líquido
Castin'Craft
Net 16 fl oz / Neto 16 oz fl / 473 mL

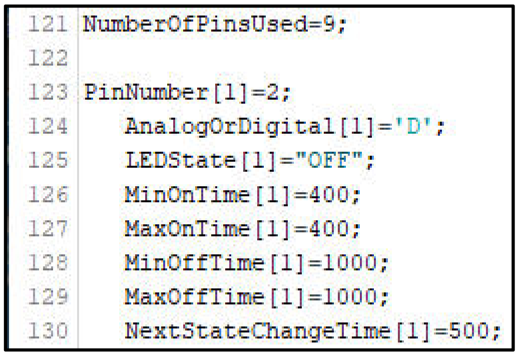
The polyester resin I use is “Clear Polyester Casting Resin” by “Castin Craft” from Michaels or Hobby Lobby; I recommend that you buy the 16 ounce size for $23.99 because this size comes with the catalyst to transform the messy syrupy liquid into a solid, whereas the quart size does not, however I was able to find the MEKP (Methyl Ethyl Ketone Peroxide) catalyst in the Radio Controlled Airplane section of my local hobby shop; also be aware the both stores have coupons on their web site which will bring the cost down a bit, and Michaels also accepts coupons from Hobby Lobby, which are often more generous than their own.

Initially I thought that because the waterfall and the river were in two planes, one mainly vertical and one mainly horizontal, I should pour the resin in two “pours”, but I changed my mind. I used Frog Tape to “dam off” any area where I thought the resin might escape, and mixed up three ounces of resin in a 5 ounce Dixie Cup, then added 5 drops of the MEKP for each ounce (exactly 15 drops total, this needs to be in these exact proportions). I used a postal scale and get the proportions exactly 5 drops per ounce, and covered my scale with some craft paper to keep it clean; don’t ask how I learned that hint. I added a small amount of PearlEx powder to the mixture to give the resin a lustrous color and finish; I like a Sapphire Blue. I then waited about 13-14 minutes for the resin to start to harden to the consistency of syrup before pouring, then quickly gave it a pour into the entire area; you will have only about a minute from the time the resin is in the syrup stage until it turns into the “Strawberry Preserves” stage, so work quickly and if necessary twist the form around to ensure even coverage; if necessary, you can let the resin dry overnight and pour another layer over the first. Be advised, Polyester Resin when curing has quite an odor, so you will want to do the resin work in a well ventilated area, and for “full disclosure” Polyester Resin is known to the State of California to be carcinogenic. Always follow label instruction, wear safety glasses and disposable gloves. If you are a “water feature veteran”, use some of your own techniques to get waves or other advanced effects.

The next day I made another shallower pour of clear resin, just for a bit more depth, and I used the opportunity to add some twigs and rocks to the scene. I waited overnight for the resin to cure. After waiting overnight, you may want to heat the form gently with a hair dryer or heat gun to get a rock-solid finish, and even then, let it sit overnight for another night.

But falling water is never silent, so I decided to add audio to this project. There is no need to go into the great detail for this aspect of the project because I documented all you need to know in “Chapter 9” of my web site, so please reference how this is done there. Make sure that you run the wire from pin16 of the MP3 Player to pin 13 of the Arduino. I also included a MP3 file for falling water that I liked, but feel free to search for audio that matches what you build.

There is code for the waterfall under the “Project 3” heading of the web site at http://daackm.github.io. As with the lighted building, I used “light blocks” to control the timing. Sparkling water cycles on and off much faster than lights in a man-made structure, so I altered the timing values to suit my liking. Also, be aware that as long as you ran a wire from pin 16 of the MP3 player to pin


```
NumberOfPinsUsed=9;

PinNumber[1]=2;
   AnalogOrDigital[1]='D';
   LEDState[1]="OFF";
   MinOnTime[1]=400;
   MaxOnTime[1]=400;
   MinOffTime[1]=1000;
   MaxOffTime[1]=1000;
   NextStateChangeTime[1]=500;
```

That’s about it for Randomizing LED, and I hope you give these projects a try. Thanks for stopping by the site.