SparkFun Electronics 

Lessons from Rebuilding Illumitune

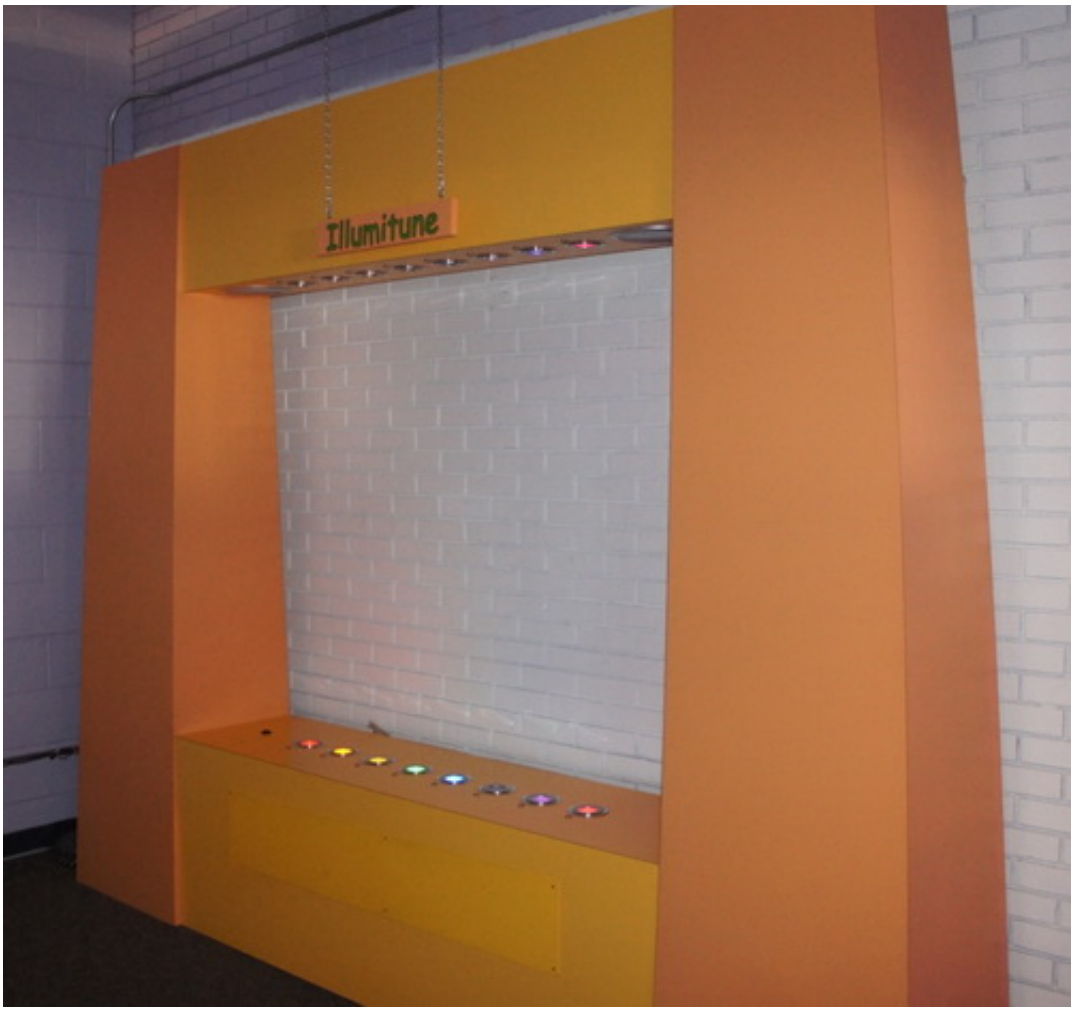
by Nate | July 06, 2011 | *55 comments* Skill Level: Intermediate

---

A few SparkFun engineers have been volunteering their time and skills at a local kid's museum called World of Wonder (WOW). It's a great challenge: Building electronics is hard, building electronics to last a week at a kid's museum is nearly impossible. I have a new found respect for exhibit builders and museum caretakers. Maintaining these exhibits is a full time job.

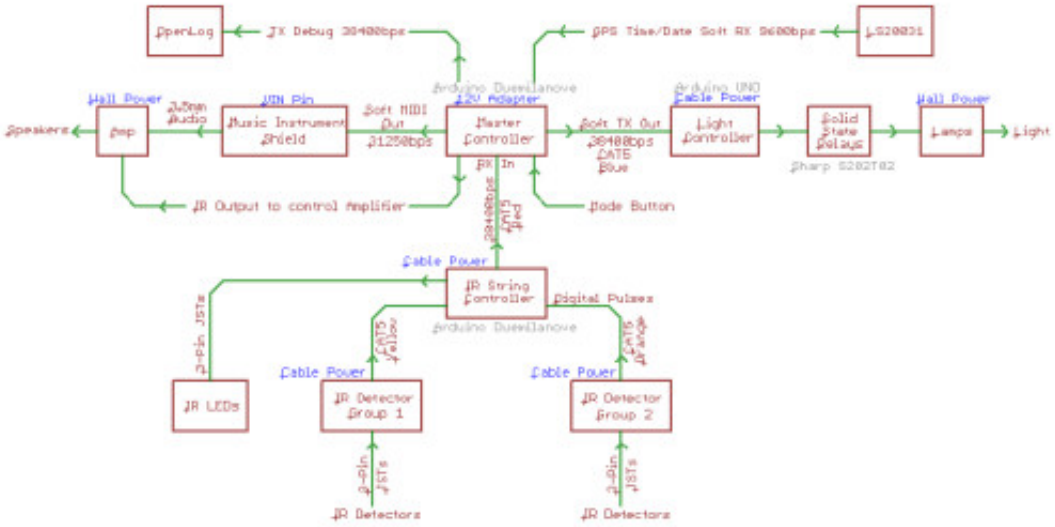
When I first started a few months ago, the museum staff didn't really understand what skills we brought to the table. My first assignments were to fix simple things: Can you make this button work? Can you take a look at this broken train whistle? Yes! I can help with that.

Once the museum staff saw how we operated, they got more bold with their requests: Can you fix this gigantic exhibit that plays music based on some sort of invisible beam and displays different colors? We don't have any schematic, firmware, contact information, or supporting materials. Uh, sure! I can take a look...

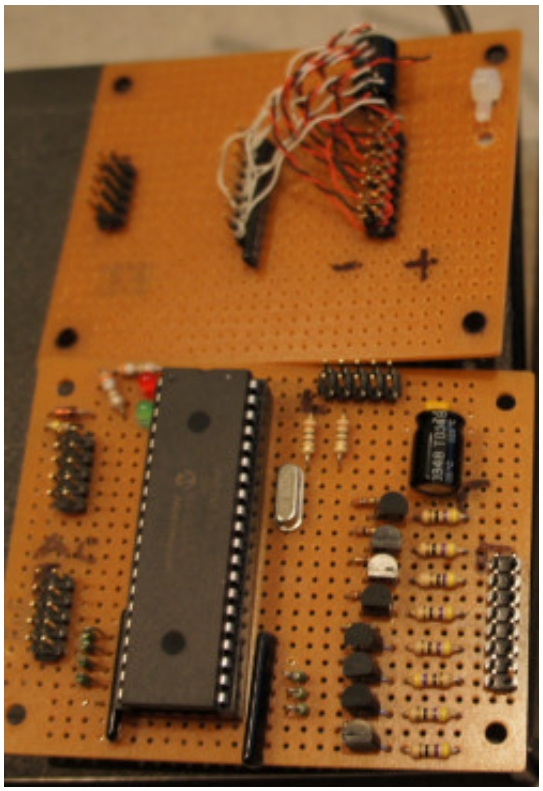


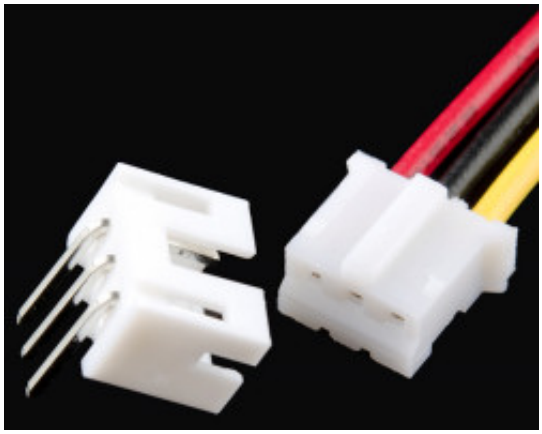
Here is the exhibit in all its glory. The following is the story of the 3 month adventure to fix the Illumitune Infrared harp. Let me start with some of the lessons learned:

1. **Document your work**
2. **Use polarized connectors**
3. **Buy pre-made cables**
4. **Label stuff**
5. **Leave the debug cable in the unit**
6. **Give me some debug information**
7. **Own a logic analyzer**



1) Document your work. Realize you won't be around next month so please leave behind everything you can to help the next person support (repair) your work. Things I now leave inside the actual exhibit: clear schematics, a wiring diagram, an overall layout, all the firmware on a jump drive, operation instructions and maintenance information for the museum staff, and contact details (my name, my email, cell # if needed, etc).

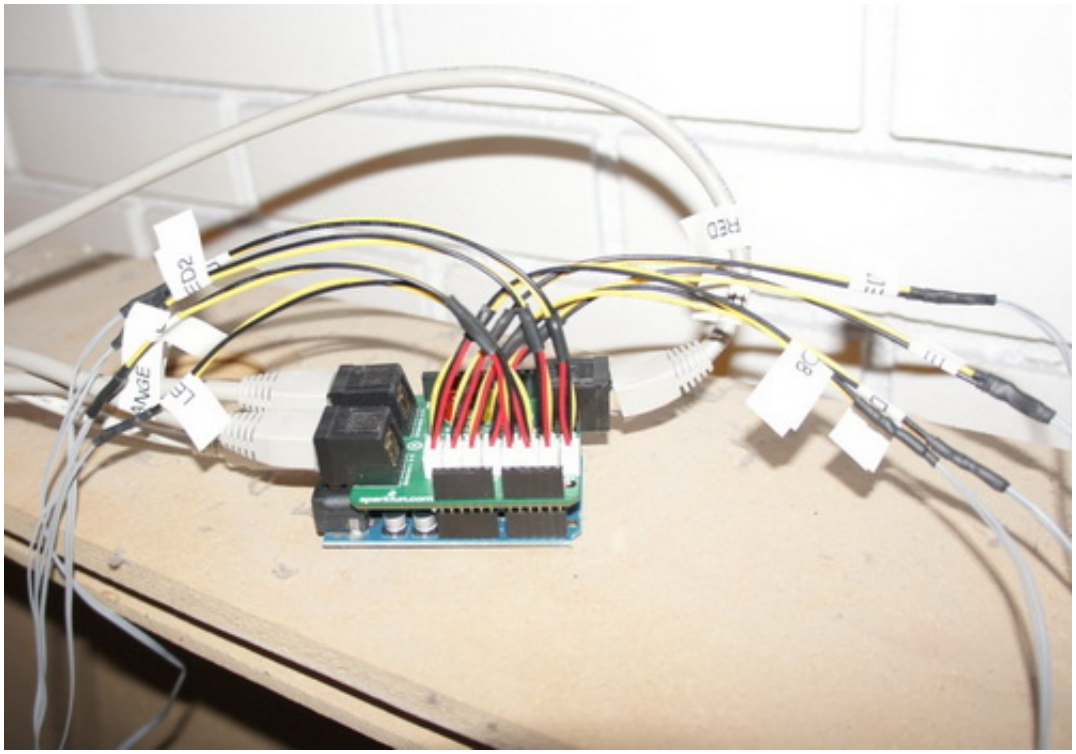




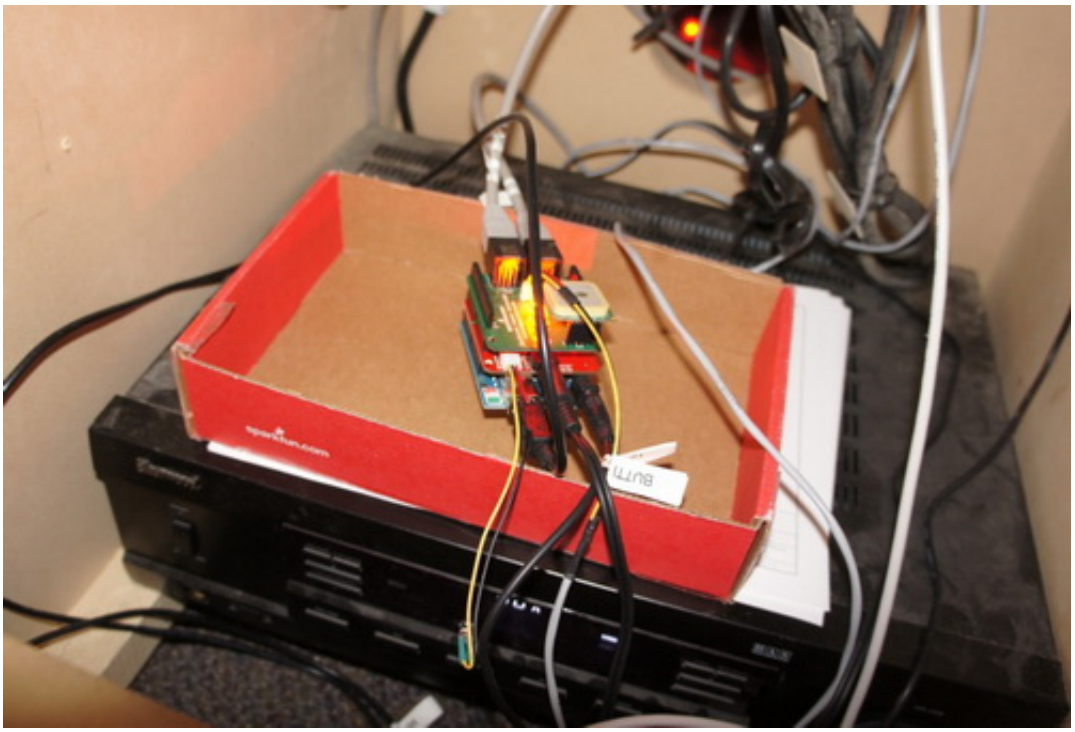
2) Use polarized connectors. This may seem silly but I've seen a lot of projects where the creator soldered wires point to point. See the clump of black, red, and white wires in the left image? Those wires originally connected the IR receivers to that perf board - wire wrapped then soldered. Not only is this slow, it is unmaintainable. You should *always* be able to remove a board or sensor by unplugging the various wires. This makes it possible to swap out parts to determine what might be broken, increasing the speed of debugging. Furthermore, you should always use *polarized* connectors. If the staff need to relocate a piece to

a different part of the museum, asking them to decipher and follow a thin sharpie mark on a perf board (see the lower left image? Where's pin 1?) is dangerous. Polarized connectors will help prevent your piece from sitting unused in the back room.

3) Buy pre-made cables. This goes hand in hand with the polarized connector lesson. You should not be running down gremlins (aka problems) in your wiring harness. Use premade cables that you can trust. I personally love to use CAT5 cable: it's cheap, readily available in many different lengths, and you can even get different colors! If multiple of the same type of cable run into a board then throw the cable color ("Red") or cable name ("IR2") on the PCB for easy identification. I also use the SparkFun JST assemblies by the handful. Buying these pre-made cables and connectors allowed me to concentrate on bigger problems then getting my crimps correct.



4) Label stuff. This is perhaps the greatest lesson I have learned from working on Illumitune. On a complex piece there will be large amounts of wires running everywhere. A $30 label printer will save your sanity and the person who looks at the piece after you. Label *everything*. This is obviously a USB cable, but what does it go to? **Main controller debug 38400bps** would be a great label. **"IR LED 4"**, **"Red to Detectors 1-4"**, **"To Light Controller"** are all great cable labels.



5) Leave the debug cables in the unit. Illumitune had a total of 4 microcontrollers. At any given time I needed to reconfigure two of them regularly. I had the idea of running USB cables from the boards down to the side access panel. I left these USB cables permanently installed in the exhibit. This was awesome! It saved me from having to climb a ladder and plug in a USB cable every time I needed to debug the system. By leaving the 'debug cable' in your exhibit you also increase the odds the next person to maintain your exhibit will be able to connect to it. Please don't assume that next user will be able to connect to the three odd spots on your perf board to get to TX/RX/GND. By leaving a standard USB A-to-B cable in the exhibit I assume that USB ports will be supported for the next 3-5 years.

```
04/27/2011, 12:16:20, Note,2
04/27/2011, 12:16:22, Note,1
04/27/2011, 12:16:22, Note,7
04/27/2011, 12:16:22, Note,2
04/27/2011, 12:16:22, Note,6
04/27/2011, 12:16:22, Note,3
04/27/2011, 12:16:22, Note,6
04/27/2011, 12:16:22, Note,4
04/27/2011, 12:16:23, Switching to instrument 0
04/27/2011, 12:16:23, Note,8
04/27/2011, 12:16:23, Note,7
04/27/2011, 12:16:23, Note,6
04/27/2011, 12:16:24, Note,5
04/27/2011, 12:16:24, Note,4
04/27/2011, 12:16:24, Switching to instrument 1
04/27/2011, 12:16:24, Note,3
04/27/2011, 12:16:24, Note,2
04/27/2011, 12:16:25, Switching to instrument 2
04/27/2011, 12:16:26, Switching to instrument 3
04/27/2011, 12:16:26, Note,1
04/27/2011, 12:16:26, Note,2
04/27/2011, 12:16:27, Switching to instrument 4
04/27/2011, 12:16:28, Switching to noises
04/27/2011, 12:16:29, Note,1
04/27/2011, 12:16:29, Note,2
04/27/2011, 12:16:29, Note,8
04/27/2011, 12:16:29, Note,3
04/27/2011, 12:16:30, Note,5
04/27/2011, 12:16:30, Note,5
04/27/2011, 12:16:30, Note,4
04/27/2011, 12:16:30, Note,3
04/27/2011, 12:16:32, Switching to instrument 0
04/27/2011, 12:16:32, Switching to instrument 1
04/27/2011, 12:16:32, Switching to instrument 2
```

6) Give me some debug information. Make the debug port obvious. I should be able to walk up to a piece and within a few minutes be able to plug in a netbook and see something coming out. Ideally the output would tell me what I'm looking at and looking for. For example, when you power up Illumitune, it states 'IR Controller Online' then 'Light Controller Online' then it displays '$3#' when beam 3 is broken.

7) Own a logic analyzer. I wasted about 8 hours of painful troubleshooting because my IR timing was off. 15 minutes after hooking up my logic analyzer and it was obvious what was wrong and I had a solution in place. I usually do all of my debugging with print statements. This works for much of the time, but for the situations where one is needed (troubleshooting I2C, SPI, or carefully timed procedures like IR

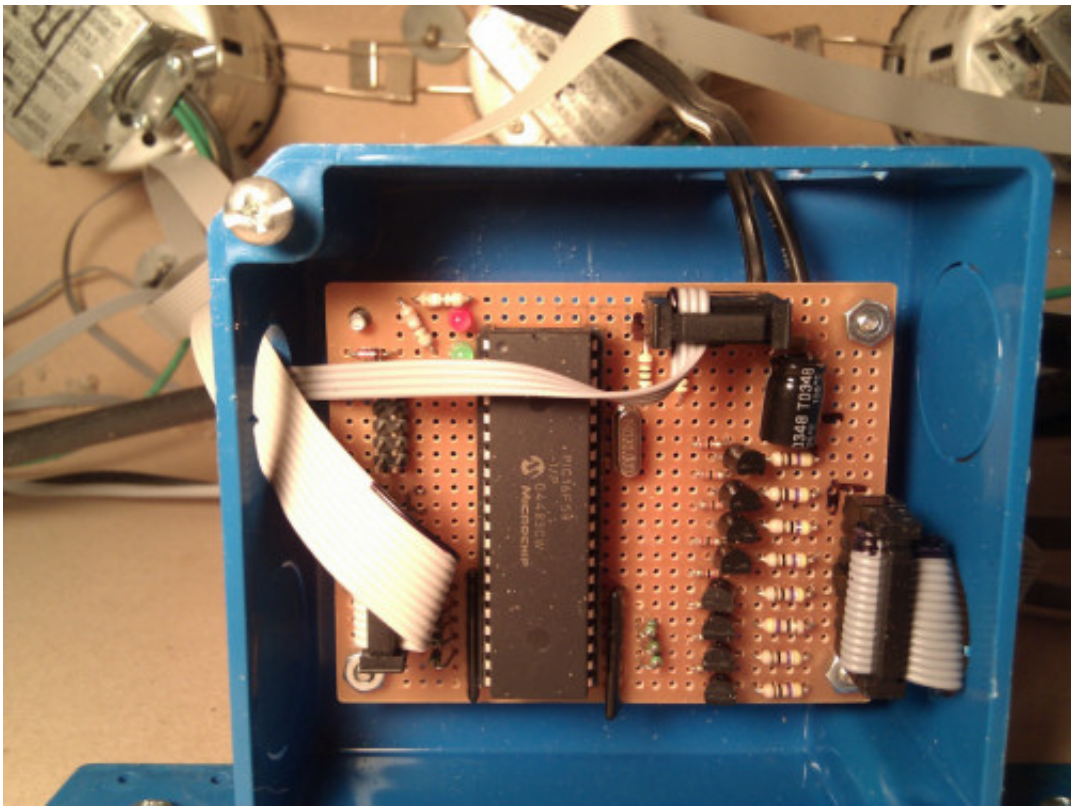
transmission) a logic analyzer is worth its weight in platinum. No really, get one. Your sanity will thank me later.

Here's all the files:

- Illumitune master plan, source file
- Main Board schematic, source files
- Main Board firmware
- IR Controller Board schematic, source files
- IR Controller Board firmware
- IR Detector Breakout Board schematic, source files
- Light Controller Board schematic, source files
- Light Controller Board firmware

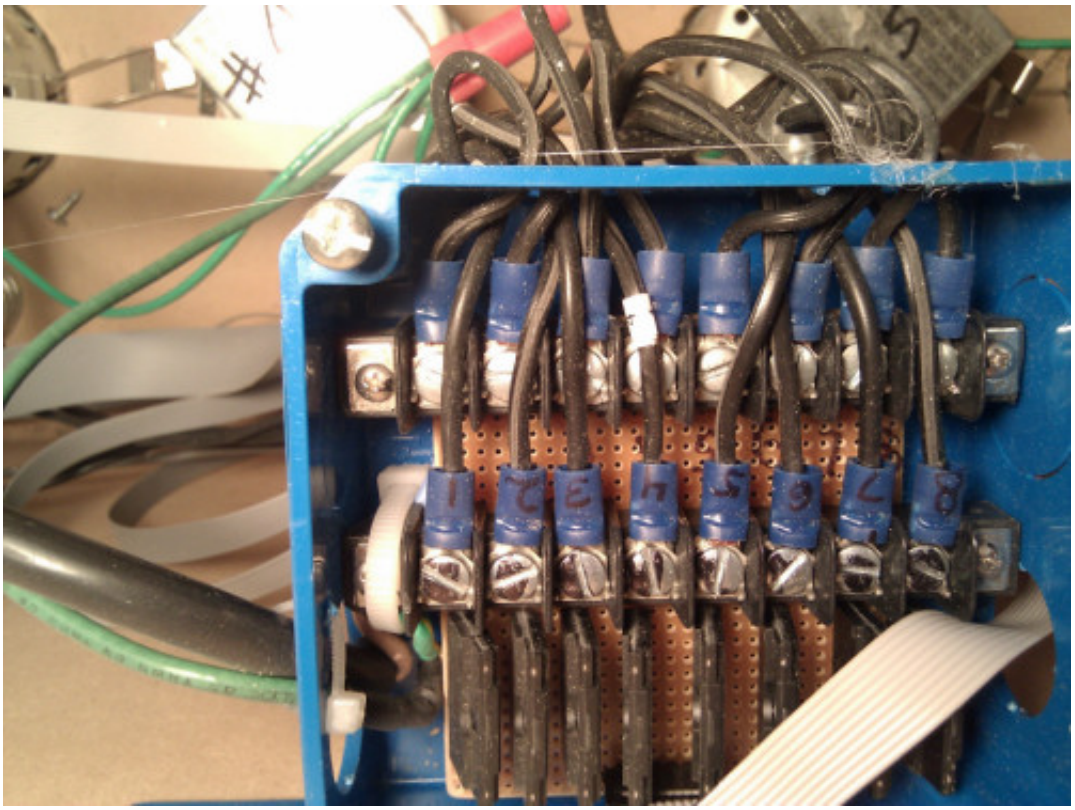
When I was first presented with 'making Illumitune work', I began by carefully inspecting the unit. If it's not obvious, Illumitune is an exhibit that plays music and illuminates lights when an infrared beam is broken. Think of it as an IR based harp. It took me a few days to figure out how the original system was designed to work. Slowly pulling apart the system, I found the 'main' board was a PIC 16F628. The date code is 0131 - meaning it was made the 31st week of 2001. Yikes this is old! The 'IR' board is a PIC 16F59 with date code from the 48th week in 2004. While the Illumitune could have been built any time after 2004, it was probably installed around 2005. I don't have any access to the original creator or the original source code so I decided to start from scratch.

Here are the basic blocks:

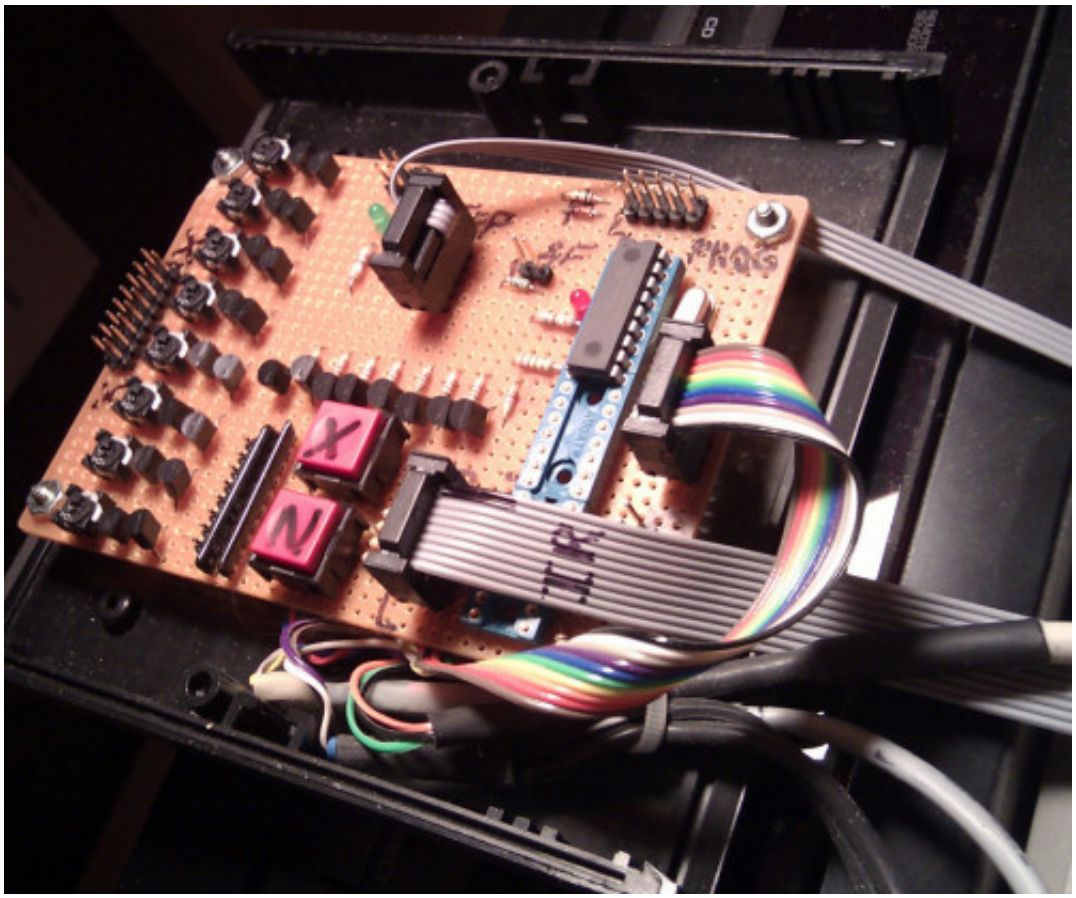


1) This board is located within the top of Illumitune above the halogen lamps (seen in the background) and IR LEDs. It was responsible for turning on/off the IR LEDs and controlling halogen lamps. There are 8

infrared LEDs on the top on the unit and 8 IR receivers built into the base of the exhibit. When these IR beams are broken, the system plays a note and turns on the appropriate light. I assume this is how it originally worked but I never saw it functional.



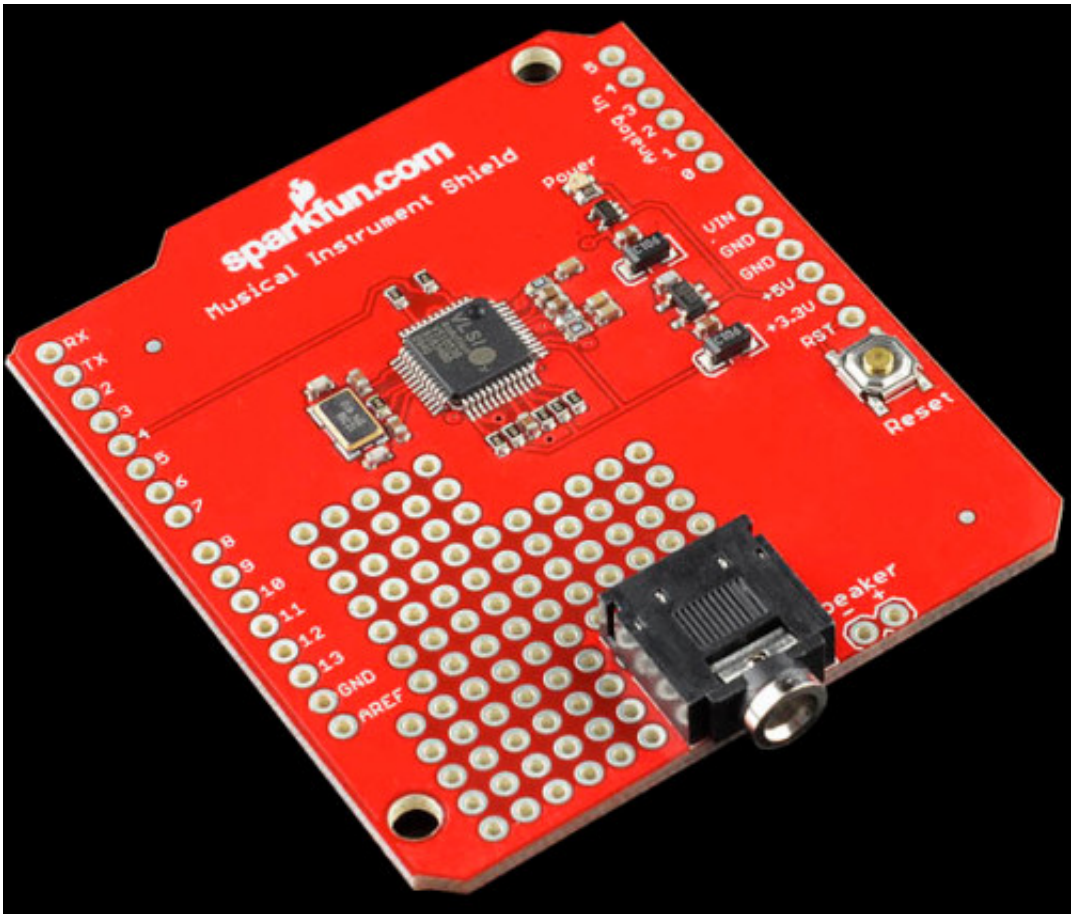
2) The lights are halogen lamps (seen in the background) and are driven by 110V AC. Fun! This control board has 8 solid state relays (SSRs) with a 10-pin IDC control ribbon. 2 wires are for ground, 8 wires control one light each via TTL level logic.



3) The 'main' board was located on the left side behind a panel. It was sitting on top the MIDI synthesizer and the large amplifier. It is connected to the 8 IR detectors as well as the larger IR board. There's far too many unlabeled pins, buttons, and trimpots to try to figure out what's going on. Easier to start from scratch...

4) The music is generated by a large MIDI synthesizer, which was in turn attached to an amplifier (seen sitting underneath) to boost the volume.

After pulling apart the system, I then set out the replace each piece.



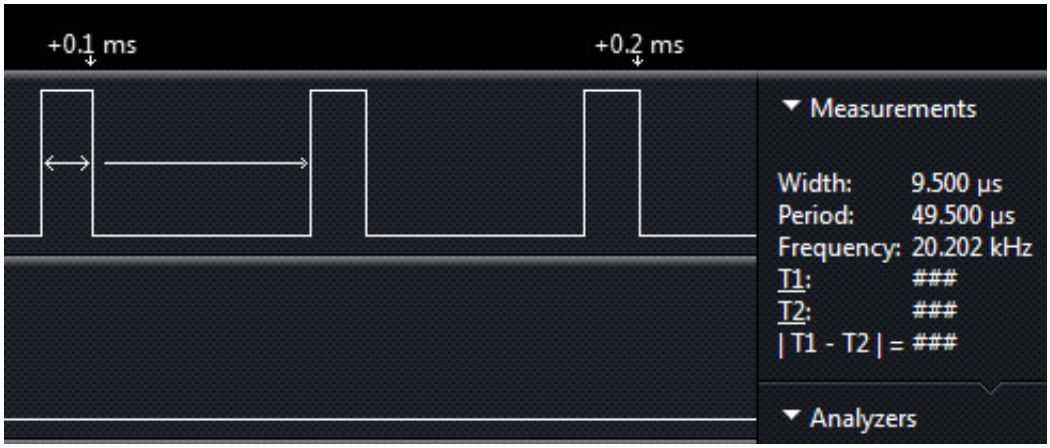
Step 1: Get music working. This was my first project using MIDI. I had learned that the VS1053 could do MIDI synthesis so I did some experimenting with the VS1053 breakout board. Then I sat down and designed the Musical Instrument Shield specifically for this project. This allowed me to completely remove the large and expensive "SoundEngine Music Module" that was previously in the system and had no documentation of its own. The Musical Instrument shield is simple - it receives serial commands from the Arduino ATmega328 and outputs musical sounds, even polyphonic notes and various sound effects.

Step 2: Get IR beams working. This was also my first project using IR. Initially I thought I could turn on *LED-4* and then check *IR-Receiver-4* to see if the beam was viewable or not. If the LED was not seen then the beam is broken and we should play a note. The problem with this is the IR detectors have a demodulator built in. This means that unless the LED is blinking at ~38kHz, the receiver will not detect the LED is on. This is great for protecting TVs and stereo systems from IR noise, but makes the Illumitune a bit more complex.

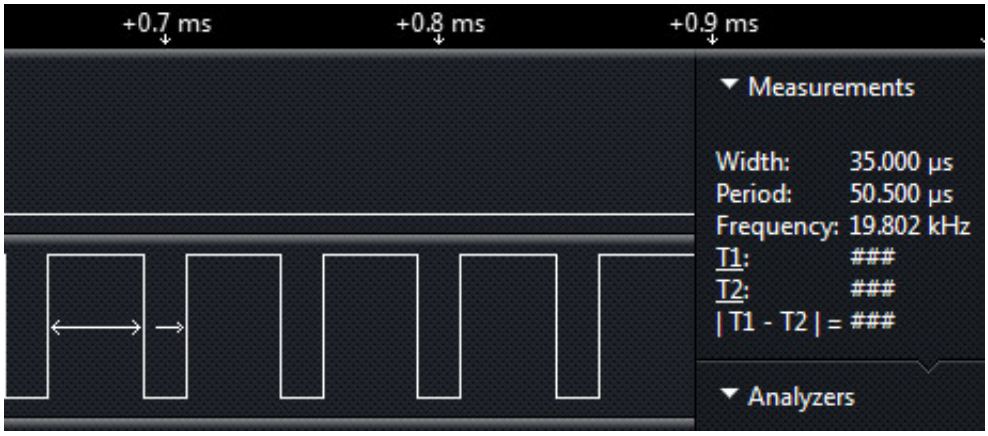
So we need to blink the IR LEDs at 38kHz. This was a simple concept that I screwed up badly.

Lesson 1: IR LEDs have a maximum current of 50mA. This is much more than normal LEDs that have a max of 20mA. This means that if you want to drive them at their max (and I did), you can't drive them directly from the microcontroller (most microcontrollers have a max of about 20mA output current per pin). You have to use a NPN transistor to turn them on/off. This is not a huge barrier but it originally caused issues because my IR beams were very weak.

Lesson 2: Timing is everything. I quickly threw together some code to pulse the (weak) IR LEDs. I used the Arduino delayMicrosecond() routine along with some if statements to check each of the 8 channels. I assumed my IR LEDs were turning on/off at the correct rate. I then tore my hair out for a couple weeks while I tweaked the code and hardware in Illumitune. When channels 1 through 4 would work, channel 6 would go completely haywire. And when I changed bits of code to get channel 6 to work, channel 3 would start to have 'phantom' actuation. It was maddening. Finally, I sat down and inspected the IR LEDs and IR detectors with a logic analyzer. What I found was that the LEDs had some really awful timing:

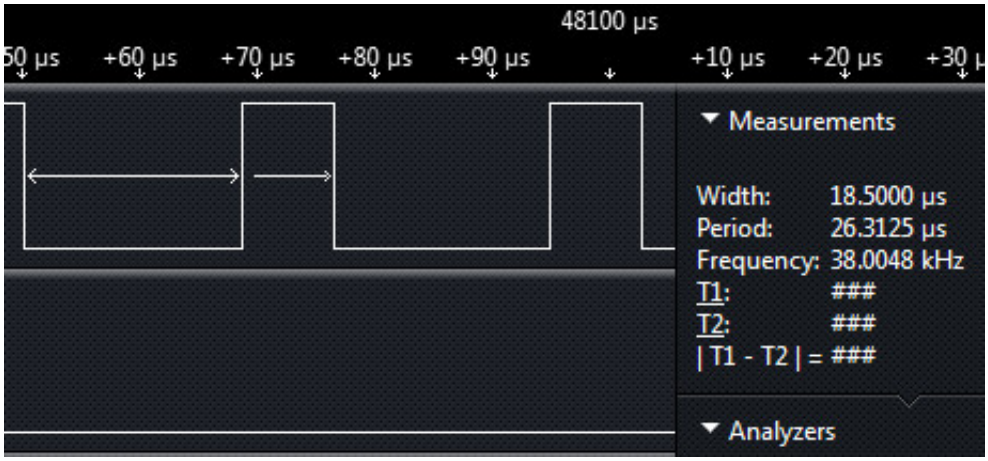


This is channel 1, turning on/off at 20kHz.



This is channel 7. It too is blinking at 20kHz but note the on time is 3 times as long as channel 1! What in the world? Here's the thing - IR receivers/detectors have many filters built into them. They filter out all light that is not infrared. They also demodulate or 'look for' a signal that is coming in at a certain frequency. The IR receivers I was using are 38kHz receivers meaning all signals that don't pulse at 38kHz it will ignore. The light I was sending was 20kHz. This was obviously really bad.

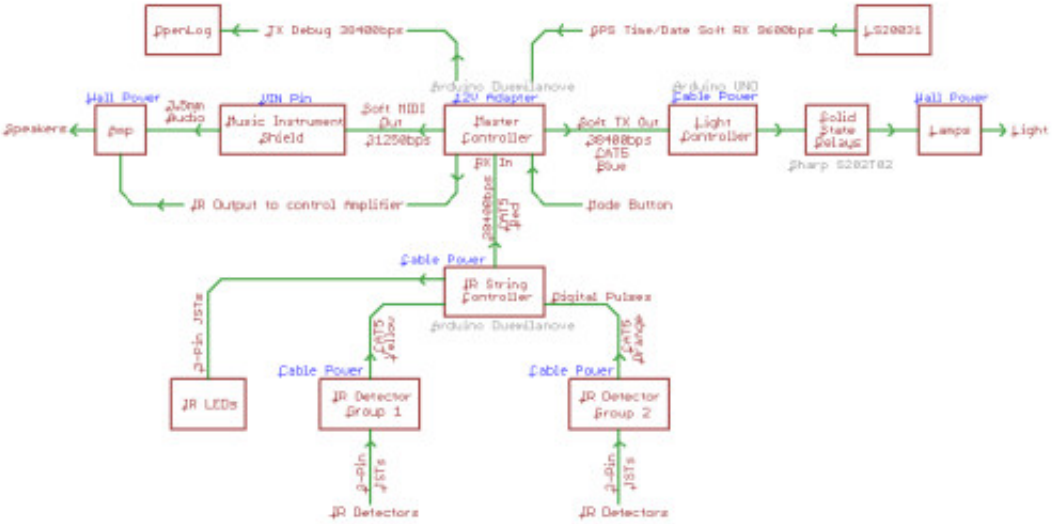
What was happening is that I was depending too heavily on Arduino's delayMicroseconds() and on the switch timing of my code. Different channels (channel 6 for example) would take a few extra instruction cycles to get to the point where it would turn on LED 6. Once I saw the LED timing on the logic analyzer I quickly re-wrote the code using switch statements and *__asm__("nop\n\t");* (also known as no-ops or nops) to really tweak and trim the timing for the LED pulsing to get exactly 38kHz. All 8 channels now had deterministic code which lead to the same timing for each channel.



Here is the new and improved timing. All channels now look identical, operate at nearly perfect 38kHz, and the code mimics the RC-5 standard of 32 'ticks'. The IR receivers now respond much more predictably and the IR beams are much more sensitive.

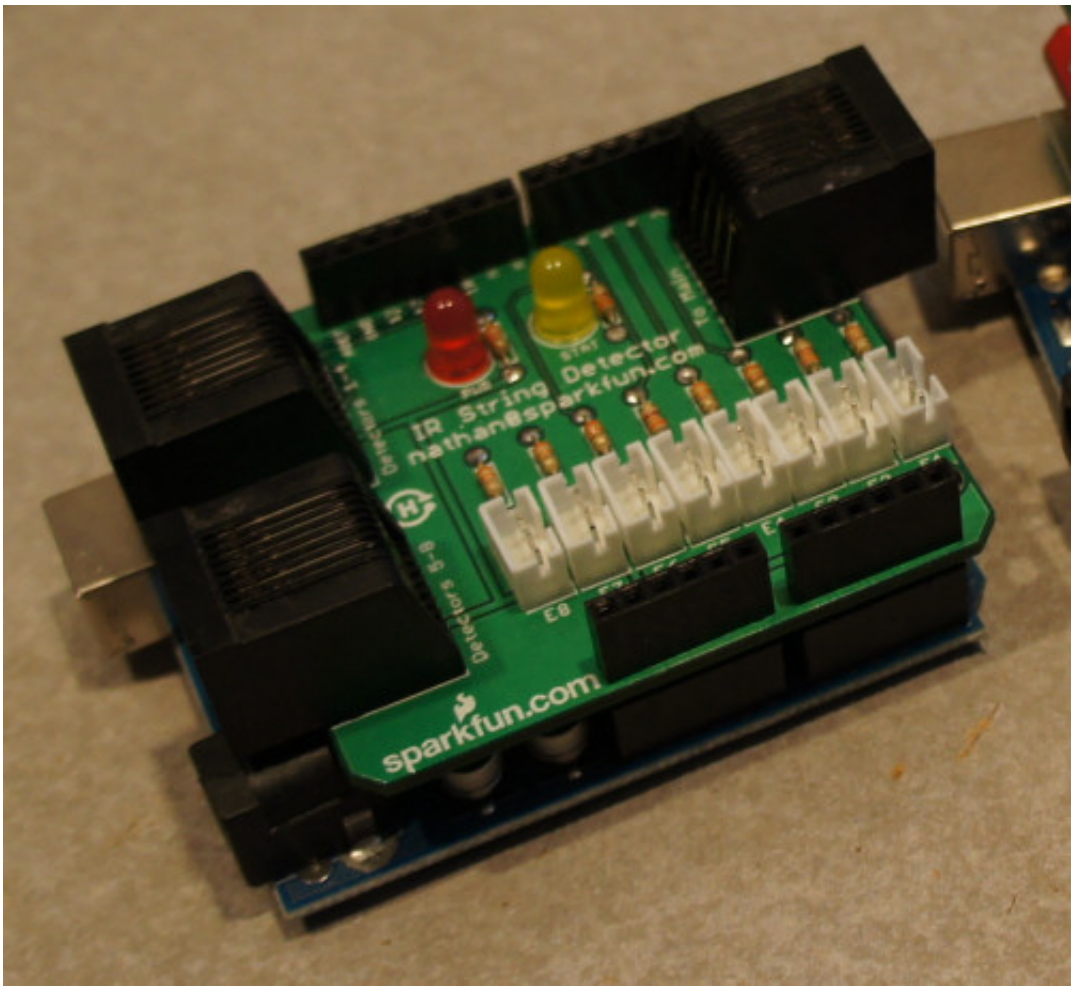
Lesson 3: IR LEDs are viewable with a cell phone. Something is broken, is it the hardware or the firmware? When in doubt, pull our your cell phone to check the see if the LEDs are working. This is a great sanity check to be sure the system is correctly wired.

With these lessons learned, here is how the final system looks:

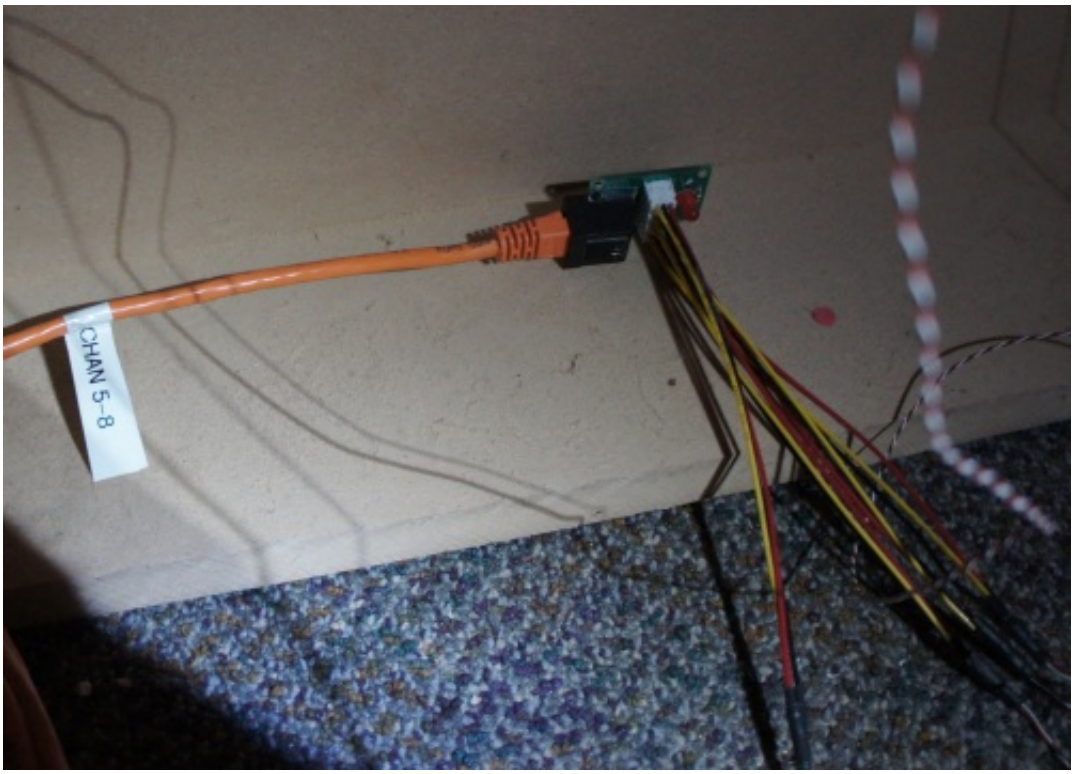


Use Arduinos with shields. Yes, I could have built this entire system on 1 PCB with one microcontroller. The problem is that I knew I was going to have a lot of hardware changes as I learned. It would be cheaper and easier to revise a shield PCB than it would to build up a monster board multiple times. Remember, maintainability is one of the highest priorities. A custom board with a specialty micro requires maintenance done by only the creator. There are a significant number of people in this world who can read a schematic and troubleshoot Arduino code.

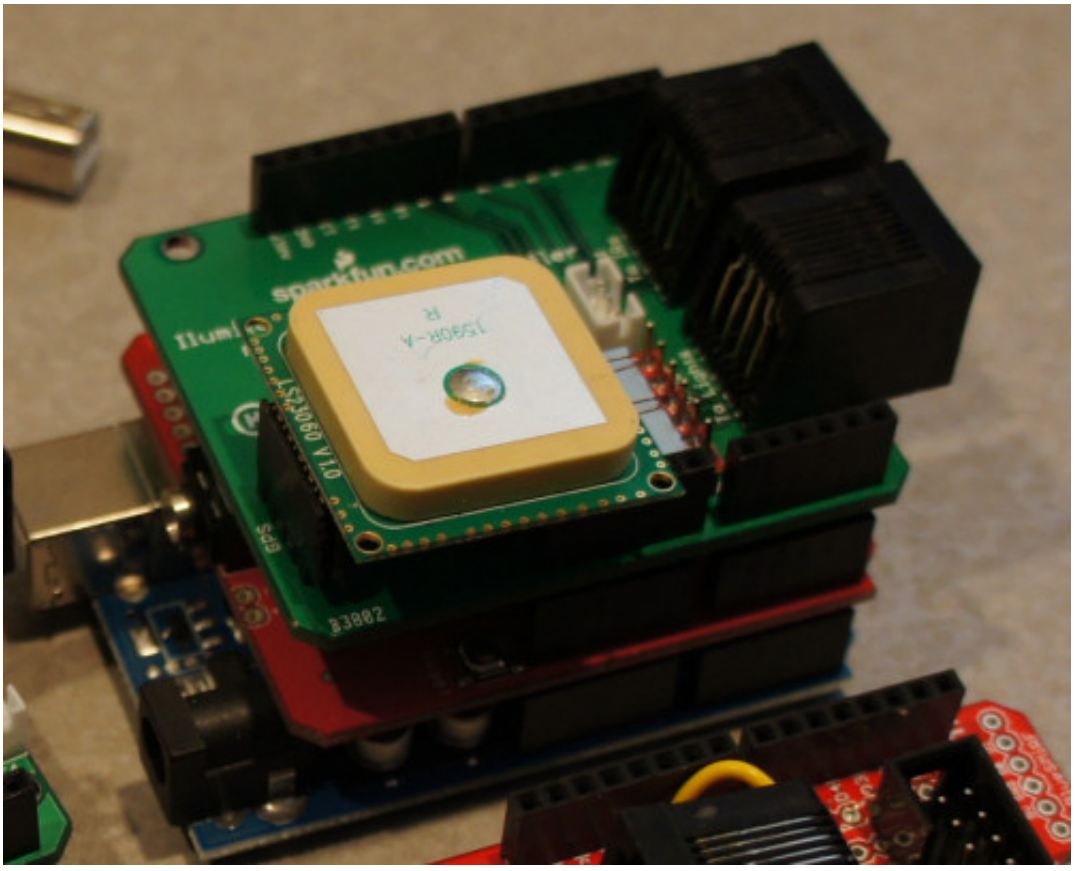
By splitting the boards up the project was a much easier problem to attack. For example, creating a board that did nothing but light control has a clearly defined set of requirements. Having independent boards also aids in debugging: it's easy to tell if each piece of the system is alive and working.



IR Controller Board is responsible for turning on 8 LEDs and monitoring the 8 receivers. This would take up nearly all of the GPIOs on the Arduino. This board would constantly be scanning the IR beams and would report any beam breaks over serial to the main board.

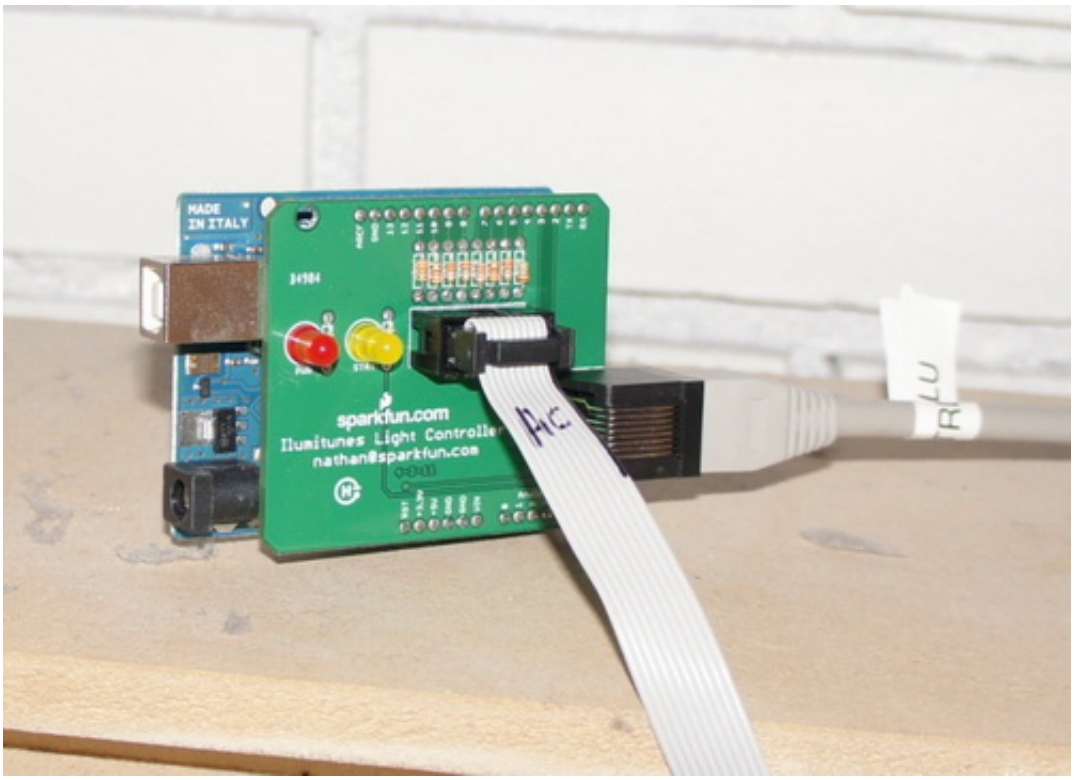


IR Detector Breakout Board is a small board that interfaces the 3-pin JST connectors of the IR detector and converts it to an RJ45 connector so that we can send the signals back to the IR Controller Board.

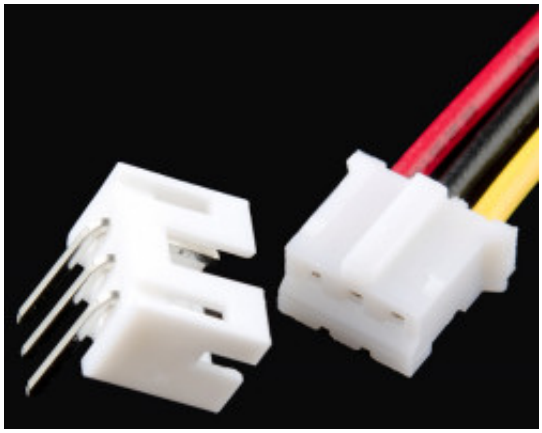


Main Controller Board is responsible for receiving beam breaks, playing notes, and sending light commands to the Light Controller Board. The main board also has an OpenLog attached. This served as a

troubleshooting logger - if the exhibit broke, I can re-create how it failed by viewing the logs. I also really wanted to see how much use the Illumitune received by the kids and what parts of the day saw the most usage. If I have a datalogger then I need the correct date and time stamp for the records so I also attached a LS20031 GPS module. Overkill I know, but RTCs require too much babysitting - you have to program the thing, make sure it has battery backup (Illumitune is turned off every night), and any RTC will have accuracy issues. GPS doesn't have any of these problems and I was confident we could get at least a few satellites indoors (in fact it gets 7 satellites and a full location lock!). The main board also reads the status of a basic button. When pressed, the instrument bank changes.



Light Controller Board is responsible for controlling the halogen lamps. It receives serial commands from the main board, turns on a lamp, and then after a short amount of time (250ms) it turns off the lamp.



Rewiring Illumitunes was straightforward. I spent a few days cutting out the old wiring and installing the new. It was a tremendous amount of work re-terminating 8 LEDs and 8 IR detectors but once all the parts were terminated with new connectors it was quite fun to quickly snap in all the new hardware. I used tons of 3-pin JST assemblies. These simple, small connectors made it possible to easily attach or detach the IR

LEDs and IR detectors to the various control PCBs. Did I mention I love RJ45 cable? The entire system uses color coded RJ45 cables. The RJ45 connector made plugging the system together extremely easy.

A cordless soldering iron can save you a lot of time as well. I spent a fair amount of time on a ladder working on the top of Illumitune. Plenty of heat shrink and a heat gun are now part of my standard set of tools as well.

Once the new hardware and wiring was installed I had a few more days of hammering on various bugs. After 3 months of work Illumitune started to make sound again! There are still a few small bugs to work out but in general the system works great! I look forward to checking the system and reviewing the logs over the coming months.

I hope these lessons help you with your next project.

SparkFun Electronics ® | Boulder, Colorado | Customer Service