Radio Controlled 120V Outlet Strip

Our project is a remotely controlled outlet strip, to create simple light shows. It was built with tw microcontrollers on their own circuit boards, communicating over a 433MHz radio channel. The relays to switch the outlets on and off, as the user desires. The relays we chose can only hand that's plenty for decorative and most desk or floor lamps.

Home | High-Level Design | Hardware Design | Software Design | Results | Conclusions | Pictures | Appendices

Problems? Questions? Contact sleepdeprived@cornell.edu.
Last updated: 04/29/04.

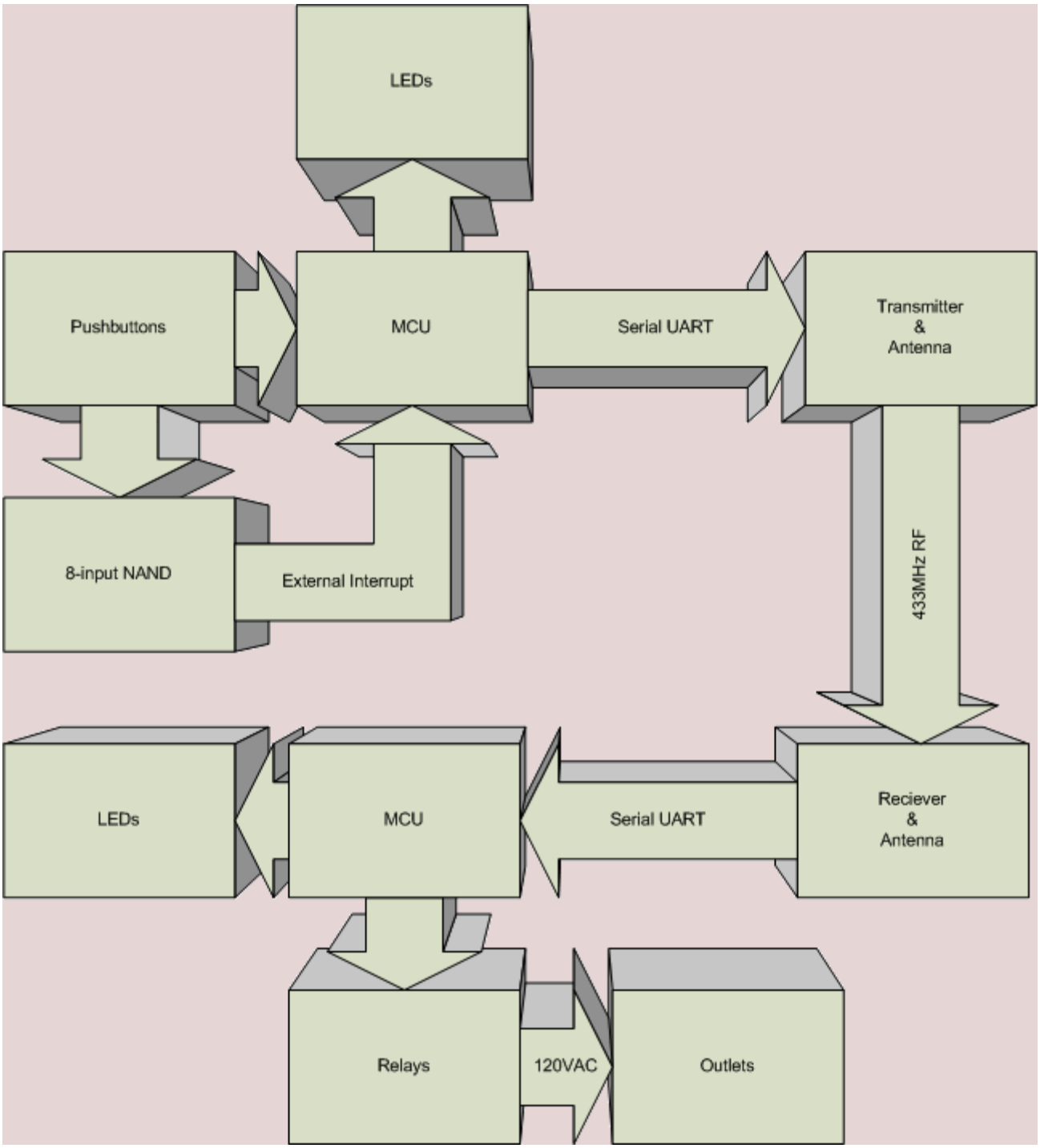
Rationale and Sources of Project Idea

Dan has a lot of decorative lightings in his dorm room, and he finds it troublesome to have to reach for switches or plugs that are not very accessible when he wants his room to look pretty. So we thought that our project idea could simplify such tasks for people that experience similar problems. The outlet strip has 4 outlets which can be controlled individually or as a group, without needing direct access.

Logical Structure

Our project creates a simple light show from a 120VAC power outlet strip which encloses four solid-state relays from Omron, capable of powering four 2A loads (lamps). We have two circuit boards, each with an ATMega16L microcontroller, which communicate via radio transmission at 433.92MHz. The transmitter and receiver came as a pair, prefabricated, and we used a 17cm piece of wire as the antenna (1/4 wavelength). There are status LEDs on both boards to indicate the state of the system, and that everything is functioning normally.

Logical Block Diagram:



The transmitter is normally in a power-down sleep state, to conserve battery life. All components were selected to have low-quiescent current, to minimize the power wasted when nothing is happening. The pushbuttons cause an external interrupt, which wakes up the MCU, and reads the port. It then sends the port state through the UART to the RF transmitter if only one button was pressed. This makes detecting data errors simpler, since the data byte must have seven 1s and one 0.

The receiver gets the data in through the UART, and decodes it. Only if a certain set of error correction steps are met does the MCU actually change the state of the system. That error correction basically consists of frame errors, parity errors, a specific start and end byte, and

the data byte as described above. Concurrently with the data reception, the MCU is processing blink modes, and toggling LEDs and the relays appropriately.

Since we don't have two way communication, the transmitter does not know the state of the receiver, and if the desired result isn't achieved (signal loss), the user just pushes the button again. This is very similar to a TV remote control - the user pushes power, which toggles the power state - he doesn't push on or off. The remote doesn't know the state of the base.

Hardware/Software Tradeoffs

Our project was much more heavily biased towards hardware complexity rather than software. The nature of the project, controlling high-voltage loads, lends itself to it much more. The software is basically error correction on the data transmission and relatively simple state machines to toggle the outlets. We chose to sleep the transmit microcontroller, and used an 8-input NAND gate to edge trigger the external interrupt when a button is pushed. This was more responsive and used far less power than periodically waking up the MCU to check for buttons, which wouldn't work very well. It's also much better, power-wise, than leaving the MCU always running. Neither end of the system needed much memory or horsepower in the microcontrollers - they're basically glorified UARTs with error correction and debouncing.

Relationship to Standards

Our project used two main standards. One was the 60Hz nature of standard wall power outlets. The other was the FCC regulations governing unlicensed radio transmission.

The 60Hz alternating current power entered our circuit through a wall plug, and normally we didn't have to worry about it. The only time it would matter is during a state transition, on to off. There could be a large power spike if we cut the power in the middle of the wave. To mitigate this, we used relays that had zero-cross circuitry, and only switched when the incoming signal crossed the zero point -- basically ensuring there wouldn't be a power spike. Our loads are designed to be primarily resistive, so there shouldn't be any lag. If anything was not as expected, the relays can handle a bit of a jolt anyway.

The main standard that we were concerned about was the FCC regulation to have our transmitter operate at low range and at a frequency that does not interfere with other radio communications such as aircraft radio navigation, radio astronomy, and search and rescue operations. Our transmitter and antenna complies with Part 15 of the FCC code, outputting an acceptable level of radiation. The antenna we use is just a piece of wire we cut to a quarter of a wavelength long (17cm), which is low powered. The transmitter we use is low-powered also and operates at 433.92MHz. In that unlicensed band, our output is well below the limit for intermittent control signals.

For detailed info, see the FCC file in the appendices.

Existing Patents, Copyrights, and Trademarks

There weren't any. Our design isn't patentable, nor did we use any copyrighted code. Nothing we used violated any trademarks either.

From the beginning, we were dealing with a remote control project, which inherently involves two separate systems. The receiver, which interfaces with the high voltage hardware, and the transmitter, which interfaces with the user. Each end of project had it's own challenges and design decisions. We'll outline most of our decisions, but for detailed trials and tribulations, see Dan's progress log in the appendix.

Transmitter

The job of this part is to gather information about the user's desires, and communicate them to the other end. We decided to build a circuit board that had eight buttons, since that's the size of a data byte, and give it a simple interface. Everything else is just there to support those buttons. We also wanted to make the battery last as long as possible, so we attempted to chose parts that use little quiescent current, and minimize the current draw while active. We also wanted to make the boards easy to build, so we attempted to minimize components when possible. For example, using the internal pull up resistors in the MCU.

We started out with an ATMega16L, because it was cheaper than the Mega32. We didn't really need the extra memory of the 32, and using an L part meant we could run at a lower Vcc. It also forced us down to 8MHz, but that's fine -- it uses less power and we don't need the extra cycles. We have the buttons hooked up to PortC, so we can read them in a single operation.

We added an 74HCT30 NAND gate, to generate an edge triggered interrupt whenever a button is pushed. Because the buttons are active low, the NAND normally has 8 high inputs, and outputs low. Whenever a button is pressed, one input goes to zero, and the microcontroller sees a rising edge on INT2. This serves to wake up the microcontroller from power down sleep. In that mode, it uses a few microamps, but takes a while to wake up. Basically, all clocks are halted and only external interrupts can wake it up. This wasn't necessary for basic operation, but dramatically increases battery life. We chose the HCT family because it is level compatible with our circuitry, and consumes a few microamps as well. CMOS technology has less leakage current than bipolar or schottky families, and thus is better for us. We hooked an LED up to the output, to verify that the gate was operating. This was in case we pushed a button and nothing happened -- we'd know if the NAND was busted, or if there were a more serious MCU issue.

Initially, we were using an LM340 voltage regulator to drop the 9V from the battery down to 5V for all the CMOS circuitry. It can handle a lot of current, and is rather robust. It is simple to use, and doesn't require much external circuitry (capacitors on input and output, if you choose). However, it has 5mA of quiescent current, which will deplete our battery in less than a week, even if the microcontroller is in power down sleep. We ordered better parts from Analog Devices, but they didn't arrive by the time of the demonstration. We did put it on afterwards though, because then the battery will last for nearly a year. We also tried to build a simple one ourselves from a Zener diode, bipolar transistor, and resistor, but decided it wasn't worth the effort after playing around with it on a breadboard.

To determine the battery level, we use a large resistive voltage divider between the 9V terminal of the battery and ground. It consists of a 1MΩ resistor and 330kΩ resistor. The output is hooked to an analog input, and is compared to the internal bandgap voltage reference. That reference is very constant, barely changing over wide ranges of temperature and Vcc. The output is 0.24182 times the input, for a max of 2.109V when the battery is at

9V. This is well below the 2.60V reference, just in case the battery starts out at 9.5V or something. When the battery is nearly completely depleted, at 7V, the output is 1.7368V. The battery status is indicated by two LEDs -- a yellow one for almost dead, and a red one for dead. The divider uses 6.7 µA, and will basically function as long as the battery has enough juice to wake up the MCU.

Another way we save power is by physically disconnecting the power from the voltage regulator when not in use. We put a jumper on the input wire of the battery, which allows easy access. In addition, it allows us to measure supply current rather simply. We remove the jumper and hook an ammeter between the pins.

The transmitter itself is a very small surface mount component, which is designed for handheld battery powered applications. It has no error correction, and uses OOK modulation to transmit the data. A 0 is the lack of carrier signal, while a 1 is the presence of it. This causes some software headaches, but is good for power - when it's "transmitting" a 0, it's using about 100µA. We were considering powering it from an op-amp, which we could turn off with a port pin, to eliminate that power also. But we decided that this power pinching was getting excessive.

We tried lots of ways to increase the range of the RF transmission, since it was not very good to begin with. One thing that helped was running the transmitter at 9V (or whatever the battery was putting out). Running at a higher voltage puts more power into the transmission, which increases the range. However, the input of the transmitter needed to go up to 9V as well. To that end, we used an LMC7111 op-amp with 20kΩ resistors to create a gain of 2. The UART signal would rail out, and we'd get our 9V. However, the 7111 has a very low slew rate, and the signal coming out was not a square wave by any stretch of the imagination. We considered using a Schmitt trigger buffer to clean it up, but couldn't find one in the lab that would operate up to 9V. Luckily, the transmitter seemed to work alright at 1200bps without it. Other op-amps used more quiescent current to get better amplification, and not all were rail-to-rail. The 7111 was easily available and good enough. We tried various antenna designs, straight wires of varying lengths, helical wires, with and without ground planes, etc. The best result was hooking up an alligator clip and wire to the existing antenna and forming a circle with it, and then touching an exposed part to use our body as a tuned element. Needless to say we gave up and just used a 17cm (1/4 wavelength) straight wire, like was recommended.

Receiver

This part is twofold -- it consists of the circuit board with the MCU and RF receiver, as well as the relays inside the outlet strip housing. They're connected with Cat5 twisted cable, each wire pair consisting of a signal wire, and a ground wire. This minimizes crosstalk and signal losses. However, our signal isn't high enough frequency or traveling over long enough distances for this to matter.

Within the housing, we hooked up four Omron G3MB-202P relays to control each outlet individually. All the neutral wires were hooked together, and the hot wires individually went to the relays. The relays were in turn hooked up to the 15A circuit breaker, which then connected to the outside world. We selected those relays due to a few factors. First off, they're really small. They are solid state devices, with no mechanical parts to break. Their housings act as heat sinks, and we push them against the metal housing as an even bigger heat sink. They can handle up to 2A at 240VAC, which is plenty for most home lighting applications. They

have an internal input resistor, so they can be hooked directly to the MCU port pins. In addition, they have zero-cross circuitry, which prevents them from switching when the AC signal is not at zero. Because of this, we don't have to worry about inductive spikes from cutting the power at the wrong time. We tried to get samples of them, since they were relatively expensive in small quantities, but did not have any luck with that.

We attempted to put a small 5V switching power supply also within the housing, since we'll need the power for the circuitry outside the box. It worked well for about 30 seconds, and then blew itself up. We're still not quite sure how it happened, but decided to just wire up a connector, and have the actual power supply outside the box. That way, if we blow another one up (which we hoped we wouldn't), the box could remain closed.

To connect the outlets to the microcontroller board, we soldered the Cat5 cable to a pin header, and hot-glued it to prevent it from breaking. We keyed the connector so it could only be inserted one way. Basically, we filled in one of the pin holes, and removed that pin from the header. It was important to do that, since reversing the polarity on the relays for even a moment can destroy them. It would be very bad if our project would cook itself from inadvertently inserting the connector backwards.

The board itself consists of a few components. First off, there's a full wave bridge rectifier from the input of the power supply before the voltage regulator. This will allow us to use various power supplies with our circuit -- it won't matter if the center is (+) or (-) polarity. We use the same Mega16L chip as in the transmitter, for code consistency and price. We also have an array of LEDs to mirror the state of the relays. They are powered from the upper half of PortC, the same port the relays are controlled from. This made the software simpler to write. We were considering running them off the same pin as the relay, but weren't sure if putting the resistance to power or ground would affect the relay. We tested it briefly, and it still worked, but weren't sure if it was detrimental to the life of the device.

Finally, there's the receiver itself. This was a prefabricated component, designed to work together with the transmitter. It has auto-gain control, which will turn random noise into a signal if the transmitter hasn't sent data in about 5-10 seconds. One problem we noticed was that the rising edge of the signal was delayed from the transmitter to the receiver by a significant amount. We considered using a retriggerable one-shot to fix the pulse width ourselves, but decided that the hardware wasn't worth the effort. Instead, we slowed down the baud rate of the transmitter to 1200bps, so the receiver's level detection would have a better chance of sampling the correct part of the waveform. We have an LED to indicate successful data reception, which only comes on if the complete data packet arrives without errors.

Just like the hardware section, doing a remote control project involves two separate pieces working together to achieve the goal. The software on the transmitter functioned to read the buttons and safely get that data to the receiver. The receiver's code interpreted the signal constantly coming in from the RF receiver, decoding a data byte when it exists, and running simple state machines to control the outlets.

Transmitter

This code serves to read the buttons when they change, send the data, and then be asleep until something happens again. To this end, we use the external edge triggered interrupt INT2 coming from the NAND gate discussed in the hardware section. If no buttons are pushed for 5 seconds, we turn off all peripherals (UART and ADC), enable the external interrupt, and put the MCU into power down sleep. When it wakes up, we disable that external interrupt, re-enable peripherals, and transmit the state of the port. The wake-up sequence is long enough from that step that the buttons are no longer bouncing. We also preload the debounce state machine to have that value of the port, such that it doesn't transmit the same data twice.

Our heartbeat LED blinks with a 50% duty cycle, every second. That indicates that the MCU is powered up, and executing code. When it's off, either something is stuck, or it's powered down. If pushing a button starts it up again, we can be pretty sure it was powered down, as well as watching the supply current change. We also read the analog input every half a second, to ascertain the level of charge on the battery. When it falls below 8V, we light the yellow LED, indicating the transmitter range may begin to suffer. Below 7.75V, we light the red LED, indicating the MCU and transmitter will soon stop operating, and it's time for a new battery. If a button is pushed and perhaps the NAND LED lights, but none of the MCU status LEDs do, it could be due to three main factors. Either the MCU is stuck in an infinite loop somewhere, the MCU is broken, or the battery is dead. Hopefully the user was paying attention to the yellow and red LEDs to notice if the problem is the battery.

While the MCU is awake, we check the button state every 30ms. We use the same global debounce scheme we've been using for the last few labs in class. When a button is pushed, we wait for it to stop bouncing, and then run the transmit routine. That routine makes sure only one button was pressed, since that's part of our error correction scheme. If that condition is satisfied, it sends out a stream of 0xaa bytes to synchronize the receiver's automatic gain control to the correct levels, and give it enough chances to resolve frame errors and find the actual start of the data byte. We then send our start byte, data byte, and stop byte. These three bytes compose our data packet. The receiver will be constantly checking for it, to ensure it doesn't think data from someone else's project (or other stray noise) is a valid signal. We don't use the transmit interrupt, since we want the operation to be blocking. Transmitting at 1200bps is fast enough that we can push buttons in succession and not notice that the MCU was waiting for the data to transmit for most of the time.

Receiver

This code is constantly watching the receive UART for a valid data packet, and when it gets one, updates the state machines controlling the relays. As mentioned in the high-level design section, our error detection scheme isn't perfect, but is rather robust. Through the combination of parity errors, frame errors, start and stop bytes, and one-hot encoding, we can determine if another signal interfered with our reception. We cannot correct the error, but at least we can

recover from bad data and not change the relays when that operation is not desired. When good data is received, we light an LED for half a second to indicate so. We also have a heartbeat LED with 10% duty cycle every second to indicate that code is correctly running.

The way the port is connected on the transmitter is as follows:

7	6	5	4	3	2	1	0
all off	speed	toggle 2nd	toggle 4th	all on	mode	toggle 1st	toggle 3rd

We decided it was simplest to send that data byte directly as wired, and do the decoding at the receiving end. So, when the data packet arrives, we decode the data byte according to that description. We use the receive interrupt, since the receiver will constantly be sending data to the UART, and we wouldn't want to miss anything that might be our data.

We have a variable which stores the "active" state of the outlets. Normally, the active lights are on, the inactive ones are off. However, in some modes of operation, that is not so. Depending on the speed chosen from {125, 250, 500, 1000, 2000, 5000} ms, the blink state machine updates itself at that interval. The different modes are normal (as just described), blink active on/all off, blink all on/inactive off, alternate active/inactive, scroll active, and marquee all. At any point in these states, the active set can be altered, which will change the pattern displayed. It's rather simple to use, yet offers a good range of possibilities.

As mentioned in the hardware section, we had to reduce the baud rate to 1200bps because of losses in the transmission channel. That speed was plenty fast for our purposes though. We did get held up for quite a while by frame errors, however. Regardless of the amount of 0xaa or 0xf0 or 0x0f or anything we sent, the frame errors would never resolve themselves completely. There would be a varying amount of them before the data byte, but the data byte itself would always have a frame error. We were able to monitor this by outputting the frame error to a pin, and watching the receiver output at the same time as that pin on the oscilloscope. What we figured out eventually was that all the received bytes except for the data byte had an even number of 1's, while the data byte always had an odd number. We were attempting to use odd parity, so all the other bytes' parity bits would be a high level, while the data byte's would be low. However, the receive UART was not properly set, so it was not expecting parity. It saw the parity bit as the stop bit, which is supposed to be high. So even though parity was being transmitted, it wasn't interpreted as such. Only the data byte would consistently have frame errors, since the stop bit wasn't a high level. This error was caused by an idiosyncrasy of UBRRH and UCSRC, both of which we were using. UBRRH we needed to slow transmission down to 1200bps, and UCSRC to enable parity checking and use two stop bits. However, they share the same I/O address. They way the MCU determines which one you meant to access is by the most significant bit -- 0 means UBRRH, 1 means UCSRC. Somehow we missed this, and never were writing to UCSRC. The importance of reading and rereading datasheets is second only to starting on the project early. Even though we were using a different microcontroller, which should mean we have to be extra careful, this would have caused trouble with the Mega32 as well.

It's awesome.

Speed and Accuracy

We did not observe much latency or delays in the communication between the transmitter and the receiver. Although we're only transmitting at 1200bps, it's still very responsive. We could achieve a range of about 60 feet and still have the receiver acquire the signal. That is, as long as the planets are aligned just right, there's a full moon, and we ask really nicely. Transmission at 30 feet was much more reliable. A problem we were not able to debug was that sometimes the blinking speed gets reset for some reason, so a temporary visual change in frequency of the lights can easily be detected. It's rather rare once the states aren't transitioning, however. Besides that, everything works the way we wanted it to.

RF Interference

To avoid the RF interference from other lab groups in the room, we send a start byte before our data and a stop byte after it. This ensures we are decoding the correct data packet. We also check for parity and frame errors, as well as our data being an active-low one-hot decoding. So we had no trouble rejecting weird data that didn't belong to us. We also only transmit when the user desires a change in operating modes, so we don't cause much interference either.

Safety

With the large amount of voltage and power that we have to deal with in this project, especially within the power strip, safety is a critical factor in our design. In order to minimize the danger of any lethal electrical shock, we enclose all our high voltage AC circuit components in a grounded steel box. All connections are secured, and everything is properly insulated and sealed inside so that nothing is exposed. Only a small hole is drilled through the box for signal wires to come out so that the rest of the controls can be done outside of the box. In addition, we have a 15A circuit breaker between the wall plug and the relays. If anything were to short out, the breaker would trip, and excess current would be channeled through the earth ground.

Another safety concern for our project may be the possible health hazards of radio frequency that we will be transmitting. Studies have shown that exposure to very high intensity RF causes thermal effects that are harmful to humans. The harm is that it increases body temperature and heats biological tissues rapidly. However, experimental results show no evidence that low level RF radiation, which we will be using, poses any dangerous biological effects at all. We experience the same level of radio frequency in our every day life from cell phones already, so using radio as our means of communication to control the power strip should not be an issue.

Usability

We labeled all our buttons in a simple manner, so users can play around with them and learn the different modes and speeds available. Our design should be pretty user-friendly. Each light is individually controllable, they're all controllable as a group, and the mode/speed buttons cycle through available options. If the user hasn't connected anything to the outlets, he can watch the LEDs on the receiver board to see what is happening.

Most of our time was spent on hardware. It took a long time to select the right components for various parts of the circuit, lay everything out, and then build it. The most time consuming part of the project was probably the tedious work of soldering our circuit because the wiring was cramped in the little space we had. It would have been easier if we could afford a printed circuit board, but that's not very good for prototyping anyway. The software part of the project was relatively straightforward, except that we had a little trouble with having the receiver correctly read the data sent by the transmitter. Some debugging was needed for the code, but the work wasn't as troublesome as the hardware part of the project.

Starting early is so important, it cannot be emphasized enough. In the first few weeks, the lab is usually half empty, and it's really easy to grab a lab station and soldering iron. It's also easier to get help from the TAs or Prof. Land. If you're doing a radio project, nobody's around to interfere with your signal. Later on, all the lab benches are taken, the soldering irons have mile-long wait lists, and once you do get one, the tip is used up. The TAs also have less time to devote to individual problems, since everyone needs them. We finished essentially a week early, and that made everything so much easier.

We completed the project the way we expected it to turn out. Everything reasonable we mentioned that we would do in the proposal was completed. We initially were considering two-way communication with an LCD on the transmitter, but decided that's way overkill for the principle of the design. We had some difficulty fitting all the components inside the metal case, but Dan managed to squeeze everything in and seal the box. If we had more time, we could perhaps get a pair of real antennas to enhance our signal and transmission range. Or at least design something better than a piece of wire. We have some cold solder joints and at times we feared that we might have some bad connections because our wiring was so cramped. Luckily that did not happen. Everything seems rather robust and no wires are coming free.

As mentioned in the high-level design, we adhered to applicable FCC regulations concerning low-power unlicensed transmission. We don't expect any federal officials knocking on our door anytime soon... well not for this project, anyway.

Intellectual property considerations

We didn't have any problem with any patent or trademark issues. All of the hardware and software were done by group members. However, the design doesn't involve anything non-obvious, and thus is not patentable.

Ethical Considerations

Since we were dealing with potentially dangerous voltages, we had to ensure we were adequately protecting our users. We could have skimped on safety to make the project easier, for example, it was really hard to squeeze everything into that box. However, we decided it would be better to avoid killing people, so we did everything we could to insulate, ground, and isolate dangerous components from the outside world.

IEEE Code of Ethics

1. To accept responsibility in making engineering decisions consistent with the safety, health and welfare of the public, and to disclose promptly factors that might endanger the public or

the environment

Our project uses large voltages, namely the 120VAC from the power strip, which may pose danger to the safety, health, and welfare of the public. So we minimize the danger by enclosing that circuitry inside the metal casing of our power strip. In addition, we have a circuit breaker that prevents the circuit from blowing up and hurting anyone if there were to be a short circuit. All circuitry within the casing is well insulated and hooked to earth ground.

4. To reject bribery in all its forms

There was no bribery whatsoever in the development of our project. Most of our parts were scrounged from Dan's home, given by Professor Land, or purchased from retailers. We didn't accept any free lunches, suitcases full of non-consecutive $20s, all-expenses-paid trips to Hawaii, or unsolicited sample parts.

7. To seek, accept, and offer honest criticism of technical work, to acknowledge and correct errors, and to credit properly the contributions of others

We posed questions to TAs and Professor Land for anything that we were not sure about. We asked for suggestions and accepted criticisms on our work. Whenever something wasn't working as expected, usually someone in lab could help us out. We invited criticisms against our user interface, in order to improve it. For example, when some users complained about the signal range, Vlad (TA) suggested the 9V battery could power the transmitter instead of the 5V from the regulator. It was a bit more complicated to amplify the UART signals and harder to regulate the power from the battery, but was worth the effort.

9. To avoid injuring others, their property, reputation, or employment by false or malicious action

Proper safety and lab rules were carried out. Classmates were friendly and helped each other out on their projects by giving ideas. For example, when other groups would ask how we accomplished certain tasks, we'd tell the truth and attempt to assist with their problems. When using the soldering iron, we were all careful with it so that we don't injure others, as well as trying to keep it in good condition so other groups had the same opportunity to use it that we did.

10. To assist colleagues and co-workers in their professional development and to support them in following this code of ethics.

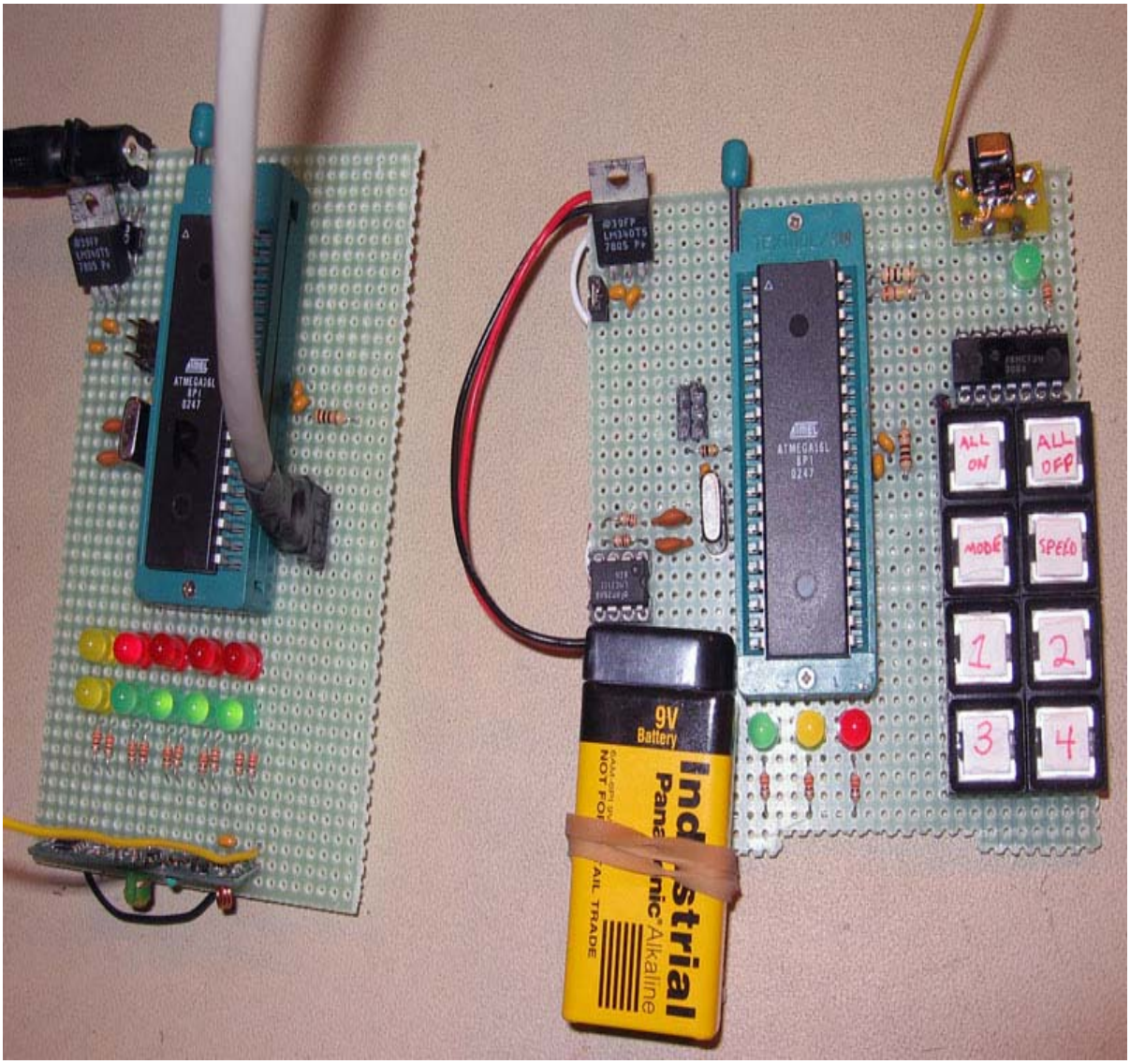
Whenever another group seemed like they were having trouble in a field we understood, we'd offer our assistance. Most of the radio control groups discussed transmission and reception range, and how to improve the robustness of the signal. When one of us found a good antenna design, we'd share it with the rest. When we discovered a delay in the rising edge of the receiver's signal, we told the group doing PWM, hoping it would help them more accurately decode their signal.

Special Thanks

Oh wow, we never thought we'd make it this far. We couldn't have done it without the love and support of our families, the academy, my goldfish Nemo, may he rest in peace, and so many others along the way. We'd also like to thank Prof. Bruce Land for always being around to support our crazy ideas and help us design our project. He was very helpful and patient, and made the project a lot more enjoyable. Our TAs, David Li and Jeannette Lukito have also been great, putting up with our nagging and helping out whenever we had problems. Dave was also cool enough to bake us a cheesecake once, and take us to Burger King late one night. How many TAs would do that for you? The list is short. We'd also like to thank Vlad Kozitsky for helping us with radio stuff and sneaking into the superlab to borrow resistors and a solder sponge. He didn't know about that though. =Þ

The circuit breaker on the left protected the rest of the world in case something within the box s
relays on the right were hooked to the microcontroller with Cat5 twisted pair cable, and to the ou
gauge stranded wire.

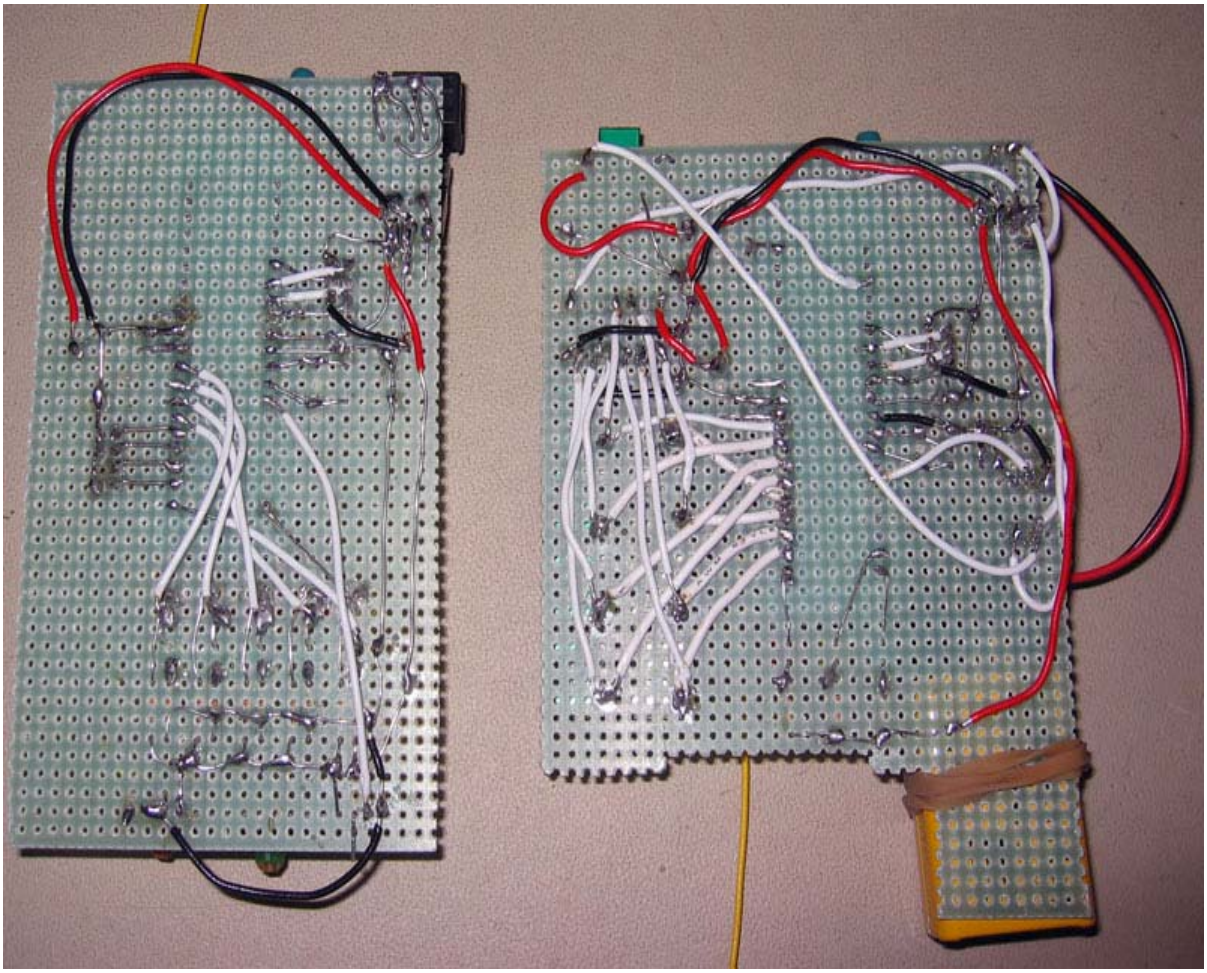
Here's the inside of the strip right before being closed up. There used to be a small switching p
jammed in there too, but somehow it blew up. So, instead there's a wire coming out that hooks
supply. The relays are covered with electrical tape, right next to the main wire bundle. Interesti
earth ground wires of the outlets weren't initially hooked to anything, but we fixed that.



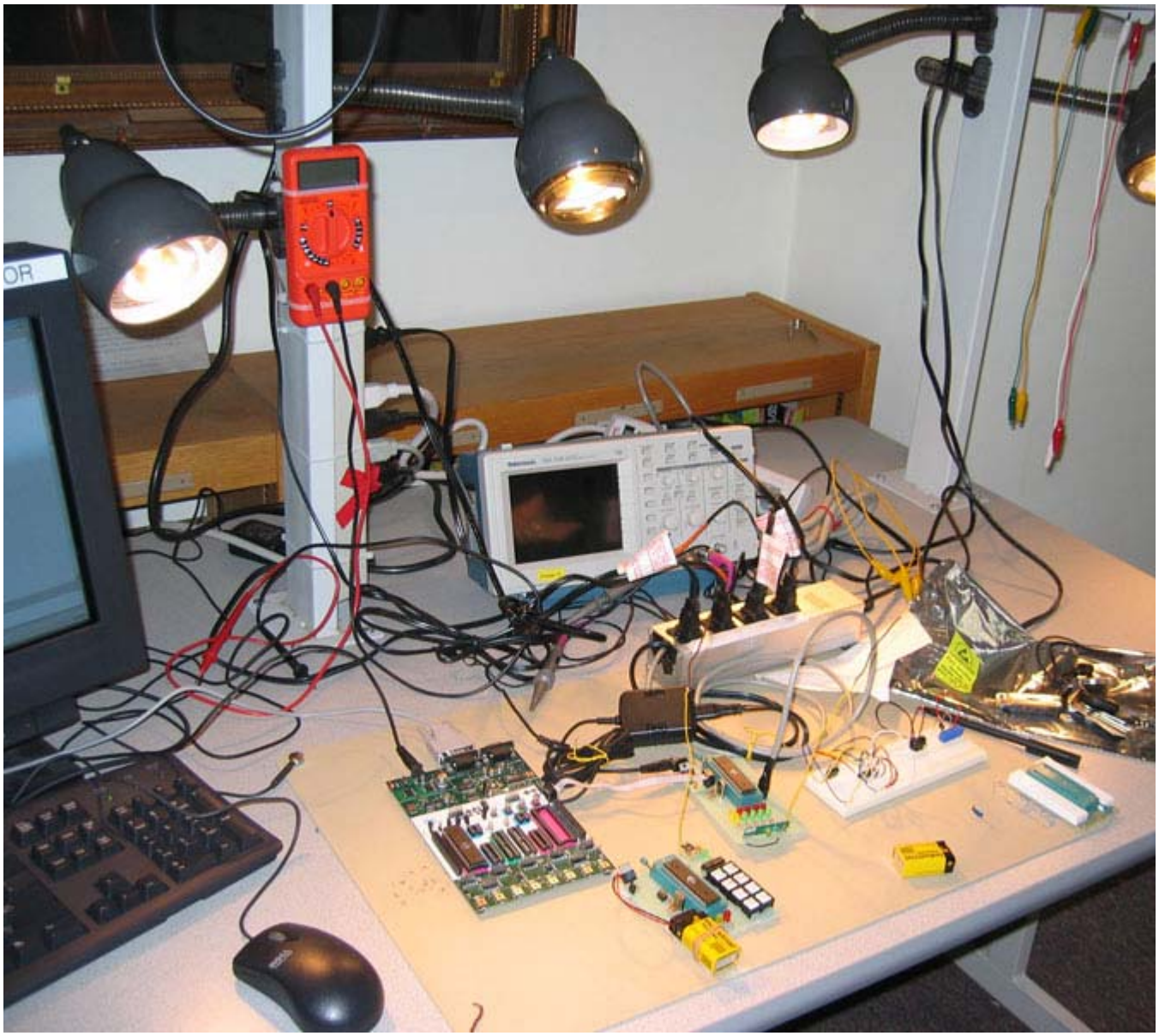
Here are the two circuit boards, transmitter on right and receiver on left. Both used a 17cm piec antenna.

Transmitter stuff... The battery fits in nicely, so we rubber banded it in. The pin header to the le for programming. The jumper in the upper left corner physically disconnects it from the power s This was also useful for measuring supply current. The 3 LEDs at the bottom are heartbeat, ba battery dead. This was detected by the resistive voltage divider hooked to the analog input. Th wired to the port, using internal pull-up resistors. They're also connected to the 8-input NAND, to the external edge triggered interrupt. The LED next to it lights whenever a button is pressed, NAND is functional. The thing in the top right corner is the 433MHz transmitter. Directly above op-amp, bringing the UART signal up to the 9V for the transmitter.

Receiver stuff... In the top left corner, there's a full wave bridge rectifier before the voltage regul plug in any type of 9V supply. The Cat5 cable connected to the pin header on the right goes to top yellow LED is the heartbeat, and the bottom indicated when a data packet was successfully rest of the LEDs mirror the status of the relays, green for on and red for off. The thing along the 433MHz receiver.



This is the bottom of the two boards. Soldering these two boards was rather tedious and difficu connections are solid and the wires aren't going anywhere. The only real way to neaten it up is printed circuit boards, but that's a little advanced for this early prototyping stage.



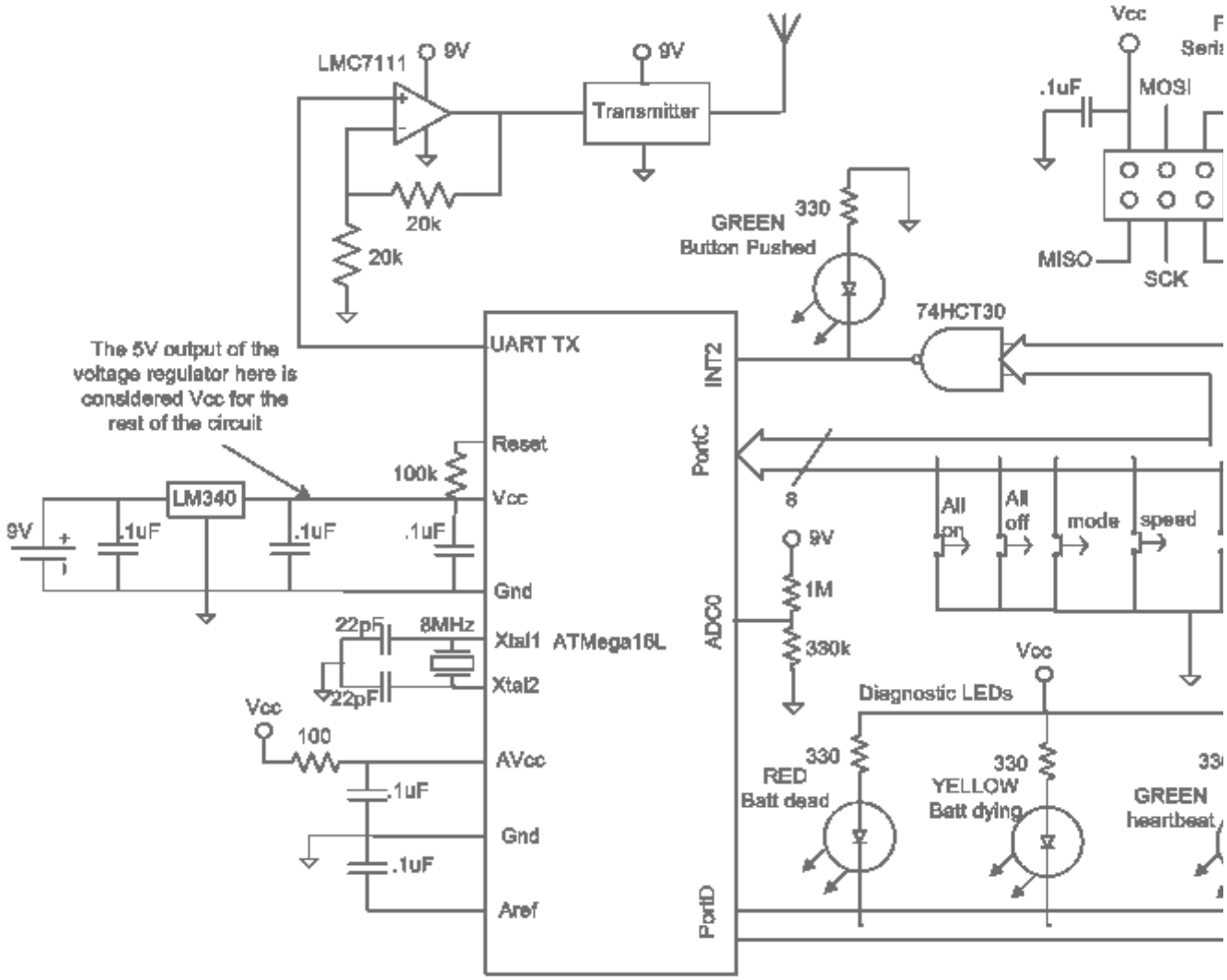
Here's a shot of it all put together. The STK500 is hooked up for programming, and there's a br
to test out some new wacky circuit. We used the 60W lamps in the lab to test the outlets' functi
were easy to come by, and made a good display of the project's capabilities.

Program Code Listings

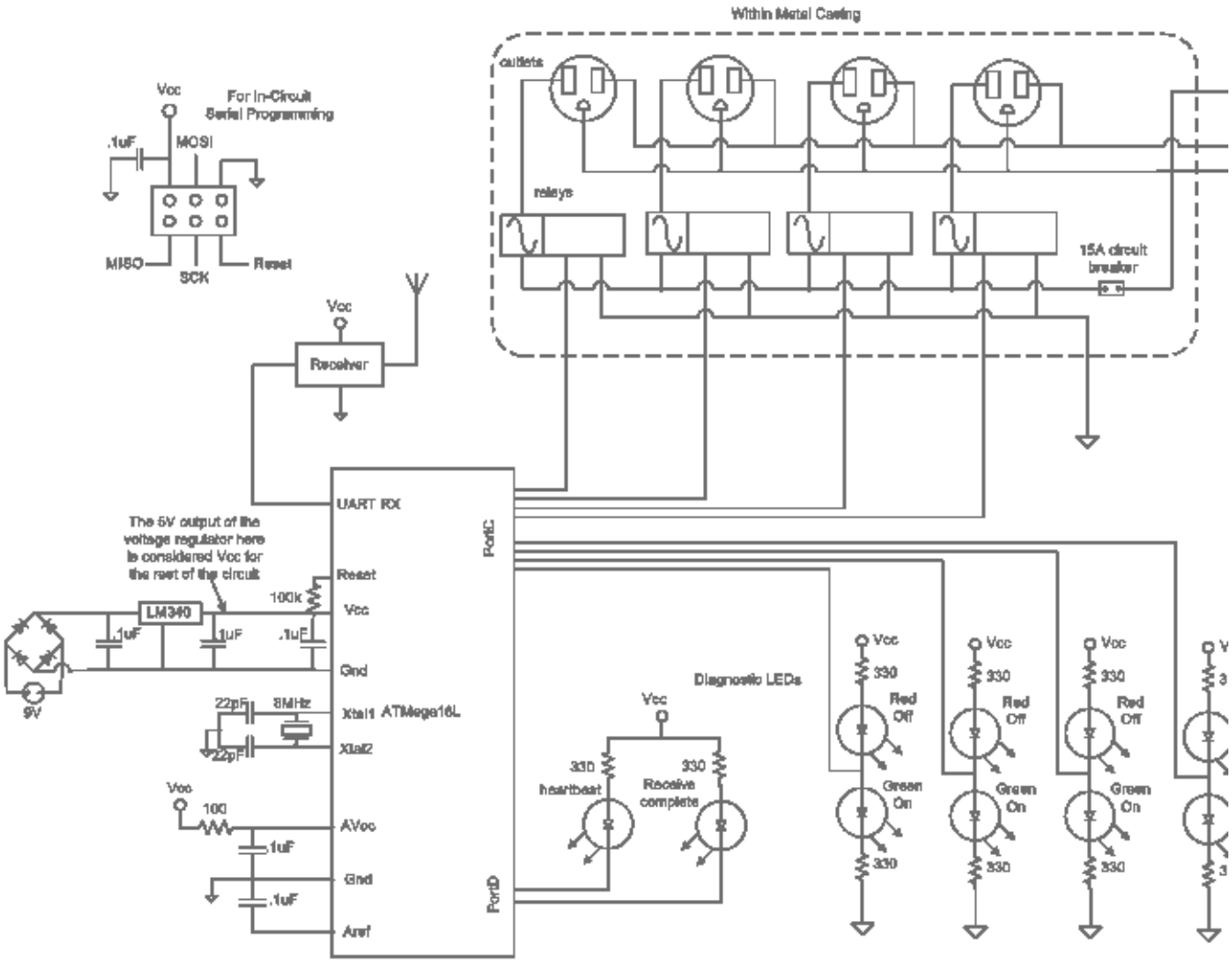
- Transmitter
- Receiver

Schematics

Transmitter:



Receiver:



Data Sheets

- Microcontrollers: Atmel ATMega16L-8PI
- Transmitter: Radiotronix RCT-433-AS
- Receiver : Radiotronix RCR-433-RP
- Relays: Omron G3MB-202P-DC5
- 8-input NAND: Texas Instruments CD74HCT30E
- Op-Amp: National Semiconductor LMC7111BIN
- RX Voltage Regulator: National Semiconductor LM340T-5.0
- TX Voltage Regulator: Analog Devices ADP667AN
- FCC Regulations governing unlicensed 433MHz transmission

Cost Breakdown

We used free stuff whenever possible, since we were constantly pushing the limits of our budge
we couldn't afford to use the nicer solder boards that were available at a cost of $5 for two of th

Item	Manufacturer	Part #	Quantity	Ur
Relays	Omron	G3MB-202P-DC5	4	$4
Microcontrollers	Atmel	ATMega16L-8PI	2	$4
RF Transmitter	Radiotronix	RCT-433-AS	1	$4
RF Receiver	Radiotronix	RCR-433-RP	1	$5
Voltage Regulator	Nat'l Semiconductor	LM340T-5.0	2	$0
Op-Amp	Nat'l Semiconductor	LMC7111BIN	1	$0

Total: $36

Other parts we didn't cost because they were "scrounged" by various means (Dan's basement, bottomless drawers of stuff, analog or digital lab):

(2) Perforated circuit boards
(1) 4-outlet power strip with circuit breaker
(2) 8MHz crystals
(14) LEDs of varying colors
(16) capacitors of varying capacitance
(20) resistors of varying resistance
(2) set of 4 momentary on pushbuttons
(4) 1N4001 power diodes
(2) 40-pin ZIF sockets
(1) 9V battery clip
(1) 9V power supply
(1) power connector
(1) 74HCT30 8-input NAND gate
cut up pin headers and sockets
wire of various gauges

Unfortunately, our sampled ATMega32L-8PI chips didn't arrive in time. The Analog Devices vo didn't arrive in time for the demo, but we did put them on afterwards, since the circuit is better th plans on using it.

Tasks by each group member

- Daniel - see progress log
  - be in the lab whenever it's open
  - component selection and circuit design
  - board layout and soldering
  - coding and debugging
  - website design and documentation
- Heidi
  - soldered some parts onto circuit boards
  - programming initial transmitter and receiver code
  - documentation and testing

- Earlier, before keeping track of dates
  - Looked up various relays … decided on G3MB-202P-5DC from Omron
  - Scrounged around for random parts
    - Romex cable, CAT5 cable, the surge strip, buttons, etc
  - Sent emails about sample parts
  - Drew up preliminary schematics
- Friday 4/9
  - Started physical assembly
    - "Borrowed" a sponge and solder wick from 315, but the irons in 476 suck
    - De-soldered ZIF sockets from old boards
    - De-soldered plug from 5V power supply
      - Might be issues with continuity between solder pads and lines
    - Attempted to connect neutral wires for outlet
      - Bent one of them too far… may cause trouble later – weak connection
  - Gathered parts
    - Perfboard, relays, voltage regulator, etc
- Saturday 4/10
  - Continued assembly
    - Soldered wires to power supply, wrapped in electrical tape
      - Tested it – it's good
    - Started soldering relays to perfboard
      - Hard to find a place for it to fit in the box… should be good by input wires
      - Got cat5 wires hooked to inputs, common 120VAC input soldered
- Monday 4/12
  - Met with Bruce to get transmitter soldered and notch cut in end cap
    - Discussed methods of detecting dying battery – Zener diode at 8V, using internal bandgap of ADC on chip with voltage divider
  - Completely assembly

        - Hooked all outlets to ground and corresponding output of relays
        - Fit everything in (relay block, circuit breaker, power supply, wires), taped up anything that might contact something else, put wire caps on the main 120VAC inputs, closed the box
    - Tested it
        - Hooked up 4 lamps and pushbuttons to the relay inputs
        - It worked!! For about 30 sec, and then died
        - The 5V power supply was busted… looked and smelled like something was burnt, continuity between output +5 and Gnd
        - Tested everything else using 9V battery and LM340 – relays still good
    - Found a 74HCT30 and tested it – it's good
    - Looked through data sheets for pins to use for various things
    - Started soldering circuit together on perfboard
        - Extremely tedious, annoying, pointless, sensation of burning
        - Decided to just do it on the breadboard, maybe move to soldering later if time
        - Can't afford nicer solder boards – other stuff was scrounged
    - Heidi worked on the transmitter code a bit, and helped with the soldering

- Wednesday 4/14
    - Took the box apart and wired in a new 9V power supply – this one has a separable plug and transformer section, so if another one gets blown up, we won't have to open the box again. Besides, it was a little bigger than the previous one, and didn't fit
    - It was at that point that I was reminded of the importance of unplugging your circuit before grabbing leads that connect to 120VAC
    - Made a 5V supply from a full wave bridge rectifier (4x 1N4001 and LM340)
    - Made a clock circuit for the breadboard (8MHz crystal and 2x 22pF capacitors)
    - Made a connector for the cat 5 cable connected to the relays, but then realized we should make it keyed, to prevent accidental insertion backwards, which would destroy the relays
    - Hooked up LEDs to the same pin as the relays, to see if the parallel resistance would matter – didn't seem to, but hard to tell
    - Started to reconsider the decision to breadboard stuff instead of soldering… the soldering really isn't that complex for the receiver, and half of the transmitter is done

- Thursday 4/15
    - Fixed the connector to the relays such that it is keyed – put two grounds to the same pin, and filled in that hole – can only put it in the right way

- Decided soldering is annoying, but worth it – takes forever though
  - First prototype a circuit on the breadboard, and once it's good, copy it
  - Did some work on the receiver – everything worked
    - Put the 5V supply circuit on the perfboard and soldered it in
    - Put the status LEDs and resistors, and soldered those
    - Started to solder basic functionality stuff (capacitors between ARef/Gnd and AVcc/Gnd)

- Friday 4/16
  - More soldering
    - Random capacitors and resistors to make the microcontroller function
    - Programming port
    - Connected LEDs to microcontroller
    - Relay output port
  - Tested it
    - Preliminary code to blink various LEDs and relays – worked properly the first time – relays successfully controlled by microcontroller
  - Started coming up with communication scheme – what to use bits for, data frame, error correction, synchronization issues, etc
  - Figured out how to detect low voltage on battery of transmitter
    - Use internal 2.56V bandgap ARef, and connect very large resistive voltage divider across battery
    - when it falls below a certain level, light an LED

- Saturday 4/17
  - Heidi did some soldering of random capacitors, resistors, and the crystal

- Sunday 4/18
  - We soldered the socket for the NAND gate into place
  - Figured out resistors for battery check
    - Maximum (at 9V) has to be below ~2.4V, to account for variations between bandgaps of different microcontrollers
    - 1MΩ and 330kΩ
      - 9V => 2.33V, 8V => 1.98V, 7V => 1.74V
      - 6.7 µA current through the divider – practically nothing

- Monday 4/19
  - Finished preliminary construction
    - Soldered receiver and transmitter into place
    - Measured and cut antenna wires
    - Status LEDs for transmitter
    - Power supply section for transmitter
    - Battery voltage monitoring resistors
      - Discovered bandgap reference does change slightly with V_{CC}, but it shouldn't be too much of a big deal if we choose appropriate transitions
  - Started software testing
    - Plugged it in, programmed both boards… and nothing happened
    - Realized we needed a more definite specification of what was being transmitted, and different states to be in – started defining specs better
    - Started verifying bit patterns in status registers… some were incorrect
    - Code needs a lot of work before it'll do something useful
- Wednesday 4/21
  - Spoke with Atmel sales contact
    - Will sample two ATMega32L-8PI chips… may take 2 weeks though
  - Added a power jumper to the transmitter board
    - Another way to save battery life, as well as monitor current usage
    - At first, it was drawing ~75mA active, 40mA powered down
    - Then, after some modification to the code, 35mA active, 5mA powered down
    - We'll see what we can do about that 5mA – should be well under a milliamp
  - Worked on code a lot
    - Sometimes bugs would be bad enough that a new program was necessary
    - Just write a shell of a program, to try to isolate the problem
  - Rewrote transmitter code
    - Random things going wrong, debugging excessive power usage
    - Only transmit if a single button is pressed

- Transmit the port when waking up, otherwise use debounce state machine
- Not transmitting continuously, so use a data frame, with parity
- Turns out the microcontroller USART is at 5V for idle, not 0V as initially expected
  - Turn off the UART transmit and zero the port when not in use to save power in the radio transmitter
- Eventually got something working, saw the waveform show up at the radio transmitter input, but couldn't tell if it was actually doing anything

- Started rewrite of receiver code
  - Huge mess – started over
  - Wrote receive interrupt to handle data frames – hopefully robust, but hard to tell – ignores input if any parity or frame errors, or if missing start or stop tags
  - No heartbeat LED for a while, nothing appeared to be working
  - Turned out the USART was initialized improperly – constantly taking an interrupt that had no routine
  - Used oscilloscope – data transmitted is actually showing up on receiver output
    - Incredible! It's like magic
    - Although, it's pretty noisy, and often is a gigantic mess
    - Probably will need low-pass filtering
  - Data not successfully received though – needs more work
  - Will have to figure out state machines for all the various modes of operation

- Thursday 4/22
  - Figured out the source of the powered down current draw – it's the voltage regulator
    - LM340 has 5mA quiescent current – not acceptable
    - 74HCT30 = 2µA, MCU = 5µA, transmitter = 100µA
    - Tried alternate circuit with a 5.6V zener diode, 10K pot, and npn transistor – lower quiescent current, but hard to tune the output correctly
    - Found better parts by Analog Devices, ordered samples – overnight delivery
      - 17µA quiescent, 0.2µA standby (probably won't use)
      - Require 10µF capacitor
      - Can't use the 74HCT30 to power down the supply regulator, since it only operates from 4.5V to 5.5V
  - Worked on the radio link

        - Low pass filtering won't help – not that type of problem
        - At 2400bps, the reception seems to have a delay on the rising edge
        - Slowed it down to 1200bps, so it wouldn't be as much of an issue
        - Tried transmitting with 2 stop bits
        - Getting frame errors in strange places, always on the one data byte we need
        - Haven't found a byte sequence to get the receiver synched up – tried varying combinations of 0xaa, 0xff, 0x00, 0x0f, different start and stop bytes, etc.
        - After transmitting a constant high (idle) for a bit, the receiver drops out, and thinks it's a zero – frame error
        - Found an error in UCSRC setting - was incorrectly writing a 0 to a field that needs to have 1 written to it always (register select) – found after lab closed – will have to test tomorrow
        - Outlook pretty bleak – Bruce and the TAs can't figure it out, other groups don't have this problem
        - Range of reception rather small
    - Other software stuff
        - More or less put on hold until radio link solved
        - If radio not solved soon, may have to disconnect it and just connect a wire between the UARTS, pretend it was the radio link
        - Changed power down code to start up the debounce state machine in the pushed state – make sure same button not sent twice

- Friday 4/23
    - Changed that UCSRC thing – and it fixed the problem
        - It was interpreting the parity bit as the stop bit, and the data byte's odd parity was 0, while all the synchronization bytes were 1
        - Shortened the synchronization to an acceptable level
    - Found errors in the data byte decoding – was active high instead of active low
    - IT'S ALIVE
        - MWAHAHAHAHAHAHAHA
    - Made all on, all off, 1, 2, 3, 4 buttons work individually
    - Added speed and mode control
        - Modified function of other buttons to change the active set of outlets, instead of the port directly
        - Started coming up with different modes – need a few more
    - Timing of blinking seems non-constant

        - Sometimes it won't be on for long enough, or won't be off for long enough
        - Seems random, and doesn't occur very often – hard to capture on the scope or find a cause
    - Sampled microcontroller probably won't arrive before the project is due
    - Sampled voltage regulators won't come until later in the week – being shipped from the Philippines – LM340 still works, but draws a little more current that desired
        - Considered disabling the transmitter completely (cutting off Vcc) when powered down to save the 100μA, but decided that's getting excessive
        - Considered using the new voltage regulator to indicate when the battery is dead, since if the battery is really dead, the microcontroller won't power up enough to indicate that
            - New part won't be here soon enough to test though
            - Another analog circuit that needs tuning
            - Not worth the trouble – nothing happens, try a new battery
    - Started working on the webpage
        - Haven't done one in a few years
        - Figured out FrontPage again (new version)
        - Made a basic template, started filling some silly things in
        - Reworked block diagram slightly

- Saturday 4/24
    - More work on the webpage
    - Cropped and resized some pictures to include in report
- Sunday 4/25
    - Investigated alternate antennas
        - Couldn't come up with any better designs for homebuilt ones, other than putting the ¼ wavelength wire perpendicular to a ground plane
        - Conditions in lab would probably adversely affect reception, but other groups have seemed to have better luck with range
        - Purchasing tuned antennas cost ~$10 each
- Monday 4/26
    - Decided to make the transmitter run at 9V to get more power out
        - Hooked its Vcc up to the battery directly, without a regulator
        - Apparently, it needs the input to be Gnd-Vcc, not to 5V

        - Tried a few circuits, decided on a rail-to-rail op-amp with gain 2
            - Tried a few op-amps, but went with LMC7111
            - It had lowest quiescent current, in a small package
            - However, it had a really slow slew rate, so the signal out was very sloped – thought it might be a problem, surely can't run UART faster
            - Looked for non-inverting Schmitt triggers, but couldn't find any in lab that would run up to 9V… it works anyway though
        - Transmission range with a fresh battery did increase slightly – yay
    - Tried putting a ground plane below the transmitter antenna – no noticeable effect

- Wednesday 4/28
    - Tried more antenna shapes, but nothing seemed better than the straight 17cm wire
        - Helical was very bad, despite the luck other groups were having
        - Sometimes putting an alligator clip on in a big loop helped
        - Sometimes touching the metal and using your body as a tuned element helped
        - Decided what we have now is good enough – works most of the way across the lab
    - More work on the website
        - Fixed up schematics to be consistent with final circuitry
        - Wrote stuff for the intro and high level design
        - Started writing hardware design
    - No word on the sampled ATMega32L-8PI chips – probably won't come in time
    - Analog Devices voltage regulators shipped overseas last Friday, should arrive this Friday

- Thursday 4/30
    - Took more pictures of the project, and a video
    - Edited the video to smoothly work as an infinite loop
    - Wrote a lot of text for the website, and finished the formatting
    - Got the website ready for demo