*Projects and Stuff*

Chameleon

Project Log

Contents

[2012/05/06 3](#_Toc327121667)

[2012/05/07 3](#_Toc327121668)

[2012/05/12 5](#_Toc327121669)

[2012/05/13 8](#_Toc327121670)

[2012/05/15 8](#_Toc327121671)

[2012/05/16 9](#_Toc327121672)

[2012/05/19 10](#_Toc327121673)

[2012/06/06 10](#_Toc327121674)

[2012/06/10 10](#_Toc327121675)

# 2012/05/06

I began setting up the bare-bones of the project, starting with creating a new Git repository. I also started simple documentation about what the project goals are.

This project will be based upon the work that I’ve done on other capacitive-sensing projects. The main benefit of capacitive sensors over other methods, like IR LEDs and Diodes/transistors, is that the sensor itself has zero additional cost. While other sensor types require an actual sensor and maybe even an emitter of some sort, which costs money, capacitive sensors are just traces on a PCB. The argument could be made that the resistors and capacitors used with the capacitive sensors also cost money, but in production quantities, it’s an order of magnitude cheaper. All of the external components for a capacitive sensor typically cost less than one cent total in production quantities.

I also began developing the schematics today. Because much of this project is based upon my previous work on capacitive sensors, much of the hard work and research is already done. So placing the major components was simple.

I’m considering which LED driver to use. Since there are 5 RGB LEDs (15 LEDs total), I’m thinking a 16-bit LED driver would be perfect. It will have to be a chip that can adjust the current or PWM duty cycle of each LED separately. I can also add a status LED to the bottom of the board with the remaining output.

# 2012/05/07

I’ve continued working through the schematic today, on a plane ride to Arizona. I’ve switched the display driver from the AS1107 to the PCA9635, which is better suited to optimally driving the smaller number of LEDs. This may not be the final decision on the driver, though, depending on price.

Each LED needs a current limiting resistor. Here’s how I determined the value for each.

**RED**

***Input Voltage = 5V***

***Forward Voltage = 1.8V*** (actually, it ranges from 1.8V to 2.6V, but a drop of only 1.8V means greater current, so I want to choose a resistor value based on this. If I used the 2.6V value, and the actual drop were 1.8V, then I might burn up my LED because the current limiting resistor wasn’t of a high enough value)

***My Desired Maximum Current = 15mA*** (or 0.015A for our calculation) (each LED can handle 30mA max, but 15mA should provide plenty of brightness. I may change this to 20mA if the prototype isn’t bright enough)

As you may recall from Ohm’s law,

To find the value of the resistance needed, you can deduce that

But you have to recall that E in this case is reduced by the forward voltage of the LED, so

And now we can plug everything in:

Since 213.3 Ω isn’t a common resistor value, we’ll go with 220 Ω, which is slightly higher, but inconsequentially so.

**GREEN & BLUE (same forward voltage)**

***Input Voltage = 5V***

***Forward Voltage = 2.8V*** (Actual values are 2.8V-3.6V)

***My Desired Maximum Current = 15mA*** (or 0.015A for our calculation)

Following the same as above, we get

And now we can plug everything in:

Since 146.6 Ω isn’t a common resistor value, we’ll go with 150 Ω, which is slightly higher, but inconsequentially so.

I also added the ATTiny44 as the brains for this device. The ATTiny44 is completely swappable with the ATTiny84 if you need more memory (8k vice 4k). It’s a microcontroller I’ve used in past projects as well. It is compatible with Atmel’s QTouch library, and can control up to 4 QMatrix capacitive sensors. In this case, I only need to control 1.

While I’ve currently got the SPX29150T-L-5-0/TR in the schematic as the regulator, I’m going to have to change this. Since the battery will be providing a nominal 4.2V, I’m considering using a boost converter to operate at 5V.

One thing you’ll notice looking at the schematics is that there are several capacitors associated with the LEDs. The 0.1uF Capacitors are intended to keep the LEDs from interfering with the capacitive sensor when they’re switching on and off. These may not be required, because the PCA9635 “LED outputs programmable to logic 1, logic 0 or ‘high-impedance’ (default at power-up) when ~~OE~~ is HIGH”. I’ll have to read more into this, but as long as the outputs are pulled high or low (and not high-impedance), it shouldn’t have negative effects on the sensor. The other key thing is to keep the LEDs as far away from the sensor as possible.

# 2012/05/12

While I was originally planning to use the PCA9635, I’m now leaning toward the PCA9685 for a couple reasons, directly from the PCA9685 Datasheet:

* *The PCA9685 allows staggered LED output on and off times to minimize current surges. The on and off time delay is independently programmable for each of the 16 channels.*
* *The PCA9685 has 4096 steps (12-bit PWM) of individual LED brightness control. The PCA9635 has only 256 steps (8-bit PWM).*

Also, since we’ll be using a Lithium-Polymer battery with a nominal voltage of 3.7V, I’ve selected the NP1402 Boost regulator. This chip will boost the battery input, which ranges from about 3.2V-4.2V, to an output of 5V.

The NP1402 requires a few external components to operate correctly.

The most important external component is probably the inductor. The datasheet gives a formula to determine the maximum inductor value, and also gives 27uH as the minimum acceptable value. So, using the formula below, we can determine the range of inductors acceptable for our application:

Where:

M = 8\*10-6

So,

The datasheet also notes that as inductor value decreases (toward 27uH), you get the following effects:

* Ability to supply higher output current
* Increased ripple
* Reduced Efficiency

So it’s really a matter of finding a good balance. I performed a search on Digikey for surface mount inductors between 29uH and 33uH, and with a current saturation of at least 200mA (since I’m expecting to see peak currents of no higher than 200mA). I also limited the component height to 2mm, so that I can ensure the final product will fit in its case. Inductors often tend to be tall components, and I can’t afford to place a part that will stick up 5 or 6mm off the board. I’ll be selecting a 33uH inductor, since that’s a common value, and easily procurable. Since the inductor I’m using for input filtering on the AVR microcontroller fits all of these specifications, I’ll just order 2 per board. That works out well.

Next is the diode.

The Boost Regulator datasheet recommends a diode with the following characteristics:

* Small forward voltage, VF < 0.3 V
* Small reverse leakage current
* Fast reverse recovery time/ switching speed
* Rated current larger than peak inductor current
* Reverse voltage larger than output voltage

I was able to find one, using Digikey's parametric search, with the following characteristics, perfectly in line with what's needed:

* Forward voltage = 0.29V @ 250mA
* Reverse leakage current is a maximum of 100uA @ 30V
* Reverse recovery time is 5nS, extremely low
* Rated for up to 750mA, well above our peak current of ~200mA
* Reverse voltage 40V, well above out peak voltage of ~5V

Last are the capacitors. The important factors for the input capacitor are a small ESR and a value around 10uF.

The output capacitor requirements are a bit more strict. The datasheet recommends a tantalum capacitor around 47uF to 68uF (or using smaller value capacitors in series to achieve the same overall capacitance). A low ESR is also important. I found a 47uF capacitor with an ESR of 90mOhms, well below the specifications recommended.

I also updated the BOM quite a bit. Still more to do, but it’s getting there. The schematics are still a work in progress at this point, but I should have them just about complete by the end of the day. I’m pausing at this point to upload my current progress via Git.

I’ve been continuing work on the schematics, and making good progress. Almost everything is in place now. It’s almost time to annotate the schematics, and move toward the board design.

I was selecting a USB connector today for charging the battery, and found that cheaper isn’t always better. The very cheapest USB jacks don’t have very detailed datasheets, something that I think is important. It’s good to know as much as possible about a part in order to ensure it has the best chance at meeting your needs. For instance, the cheaper models have no information about the current-handling capability of the pins on the connector. This is pretty important if I’m charging something. So I’m going with a connector priced $0.39, rather than the ultra-cheap $0.25 connectors.

Manufacturing is expensive. Right now I’m looking the following:

Components for PCB: ~$5.00

Battery: ~$1.66

Case: ~$1.00

PCB Manufacturing: ~$1.50

Assembly: ~$Unknown yet

Around $10.00 or so per board. This isn’t even including shipping costs.

# 2012/05/13

KiCad can be tricky sometimes. One issue I’ve had in the past is that when I use FreeRoute, there doesn’t seem to be a good way to tell the router what areas to avoid (AKA, Keepout regions). This is particular problematic when dealing with capacitive sensors. You don’t want traces running behind your sensors.

The workaround is to create fill zones, not attached to any net, in areas you want to keep traces away from. Then, when you export a Spectra Design File for use in FreeRoute, you edit the file prior to opening FreeRoute. Here’s a short description of the process: <http://permalink.gmane.org/gmane.comp.cad.kicad.user/5755>

# 2012/05/15

I’ve been working heavily today on placing and routing. This project is actually pretty difficult to route due to a number of factors:

* Overall size of the board compared to number of components. There’s not a lot of free space for tracks
* The shape of the board. The actual area for routing traces is a ring, because we don’t want any traces near the capacitive sensor. This is a big challenge!

I’ve been routing many of the easy traces, and most important traces by hand. While it doesn’t look as neat, I turned off the automatic 45 degree trace bends for hand-routing. Otherwise it’s simply too restricting on this particular board. I’m turning it back on for autorouting.

If the autorouter has problems in a particular area after several passes, I’ll stop the program, move components or route by hand, and then restart. It’s getting there, but this is slow work. You simply cannot leave everything up to an autorouter and expect it all to work perfectly.

One thing I’ve found very useful is this: make all the critical routes by hand. Go through the process several times of letting FreeRoute work for a bit and then adjusting by hand until FreeRoute can complete all of the routes. Then let FreeRoute run overnight on a reasonably powerful computer. The “Batch Optimizer” will run in several passes, and slowly will make things better and better. For instance, in this project, after several initial routing passes, there were about 50-something vias. Vias are a small plated hole that transfers a signal from the front of the board to the back (or to other layers on multi-layer boards), and they’re one of those things that are necessary, but generally should be avoided as much as you’re able to. So, I let the optimizer continue running, and by pass 10, the via count was down to 40. That’s a pretty impressive improvement.

The last thing to keep in mind is that by this point, you should better than the autorouter which important traces should go where, and what unusual circumstances need to be taken into account on your board. For instance, once autorouting was done on this project, there were two traces running under the “Y” QMatrix line of my capacitive sensor, as it routed to the AVR microcontroller. Ideally, you don’t want anything running under “Y” lines, as they tend to greatly decrease sensitivity of the sensors. So, once autorouting was complete and the optimizer had done about as well as it could, I stopped it and pushed those two traces away from the “Y” line.

# 2012/05/16

Today was another day of routing. It’s a repetitive process of testing the autorouter, moving things to better placement, and trying again with the autorouter until things get as good as possible. And I think I’ve about gotten to that point. There are no trace failures, and overall everything looks pretty good.

I’m going to do a quick git push, though, before I Back Import the Spectra Session file from FreeRoute to PCBNew. That way, I notice a problem in the future, I can go back to this point and easily correct it, rather than having to rip up all the traces and start from scratch.

I sent off the Gerber files to PCBCart for manufacture today. I ordered a set of prototype boards with the same specifications that the final boards will have. But first I made a few last edits to part placement and routing. Then I did a “Design Rule Check” to ensure all my pads are connected, all of my trace widths are wide enough, nothing’s crossing that shouldn’t be, etc.

Then I plotted the Gerber and Drill files per PCBCart’s specifications, reviewed the Gerber’s layer by layer for errors. Finding none, I zipped the Gerbers up and sent them away.

Now it’s a matter of waiting for the boards to be made and sent back. In the meantime, I have several other projects to be working on.

# 2012/05/19

I installed Atmel Studio 6 and used the QTouch Project Builder to generate the skeleton for this project. It almost feels like cheating. You input how many sensors, which microcontroller you’ll be using, and which pins you’re using for each channel, and it creates a ready-to-go project. This is very convenient, and will cut probably cut a full day or two of development time, since I don’t have to manually fill all this out in the source files. Wizards can be nice sometimes.

# 2012/06/06

I’ve received the initial prototype boards from PCBCart, and they look fantastic. I assembled the first board, and have applied power, and nothing burnt up. That’s always a good start.

I have already discovered a few changes to make from my original design and plans:

1. The sensor should be placed on the board opposite the SMT components, so that the sensor is flush against the top of the enclosure. This should be very simple to do. Place on opposite side, use 2 vias, and mirror the sensor so that no other movement of traces or parts is required.
2. The USB Micro-B connector should be moved slightly inward on the board, and a slight flat area should be routed off the edge of the circular PCB where the connector sits. This way the PCB will fit properly and snugly in the enclosure.
3. I originally specified the inner height of the enclosure at 10mm, which makes the overall height of the enclosure 16-18mm; quite thick! Instead, if the components are on the bottom of the board, against the battery, the inner height can be reduced significantly. Eyeballing it, it looks like about 5mm, making the overall enclosure height 11-13mm. This is much closer to what someone would expect for a beverage coaster; approximately ½ inch total.

This evening I will be testing the battery, installing it with the board, and testing that the Atmel AVR microcontroller can be programmed.

# 2012/06/10

I got a bit too excited in assembling the initial prototype, and soldered up the whole thing in one go. The microcontroller wouldn’t program, and when I powered the device via USB, the boost converter burnt up. Not nice.

The better way to do this sort of thing is to solder up one ‘block’ at a time. For instance, in my second attempt, I first soldered up the microcontroller block, which includes only:

* Microcontroller (IC1)
* Input inductor (L1)
* Decoupling Capacitor (C1)
* Reset line components (R5, S2, and C3)
* ISP Header (P1)

By assembling only these components, I was able to check that the microcontroller was wired up correctly and could be programmed. Once confirmed, I added the crystal and crystal capacitors, and checked again that the AVR Microcontroller was programmable.

Then I moved on, block by block, until everything was soldered in.

Knowing that there was something wrong with the power regulation block on the first prototype, I dug a bit to find what the problem might be.

D1, the schottky diode used in the power regulation block, is incorrectly placed on the PCB. The schematic uses a 2-pin representation of the diode, since a diode only actually has two parts, and anode and a cathode. The actual device used, though, has three pins, one of which has no electrical connection. Basically, the cathode was unconnected, which caused some significant problems. I corrected this in the prototype by canting the diode to the side so that the anode and cathode are both connected. The only real problem with this, other than aesthetics, is that that third pin provides additional physical stability for the chip. I don’t foresee this being a problem with the prototyping, but it will clearly need to be corrected in the final product.

On the firmware side of things, I’m using Donald R. Blake’s USI TWI Slave driver, which is based on Atmel’s Application Note AVR312: Using the USI Module as an I2C slave. I may rewrite/refine his code in the future, but for initial testing, there’s no point in reinventing the wheel.

Another trip-up: QMatrix can be used with ATTiny44**A**, or ATTiny84, but not ATTiny44, which is what I ordered. Whoops. New chip on the way.

A few further changes I plan to make on the next iteration of Chameleon:

* Place sensor and LEDs on one side of the board, remaining components on the other
* Improve connection of battery leads to board. Use double-holes for strain-relief
* Correct D1 placement
* Add TP3, which will be VCC