Boundary/Condition

The repairing and hacking of a Dell J1KND (BQ8050) laptop battery

2020.01.18 Uncategorized

There's so many great laptop battery hacking content on the internet yet so little love goes to the BQ8050 fuel gauge IC. I'm not sure why but I guess it's for a good reason. For starters these ICs go by two labels: one for the blank version and one for the factory programmed version. Example – blank: BQ8030, factory programmed: BQ20Z90. Source.

Reviewing the great content of this source I realized that the BQ8050 IC might probably be the blank version of a more popular IC. Meaning the manufacturer (SANYO) wanted to flash its own firmware to the IC.

Alright so what happened? My original battery got old in the past 10 years and not knowing anything about Smart Battery Systems I quickly jumped into buying new 18650 Li-Ion cells to replace the old ones. I even got a better idea: lets buy higher capacity cells to extend the original battery life. What a genius idea. I wonder why more people don't do this?

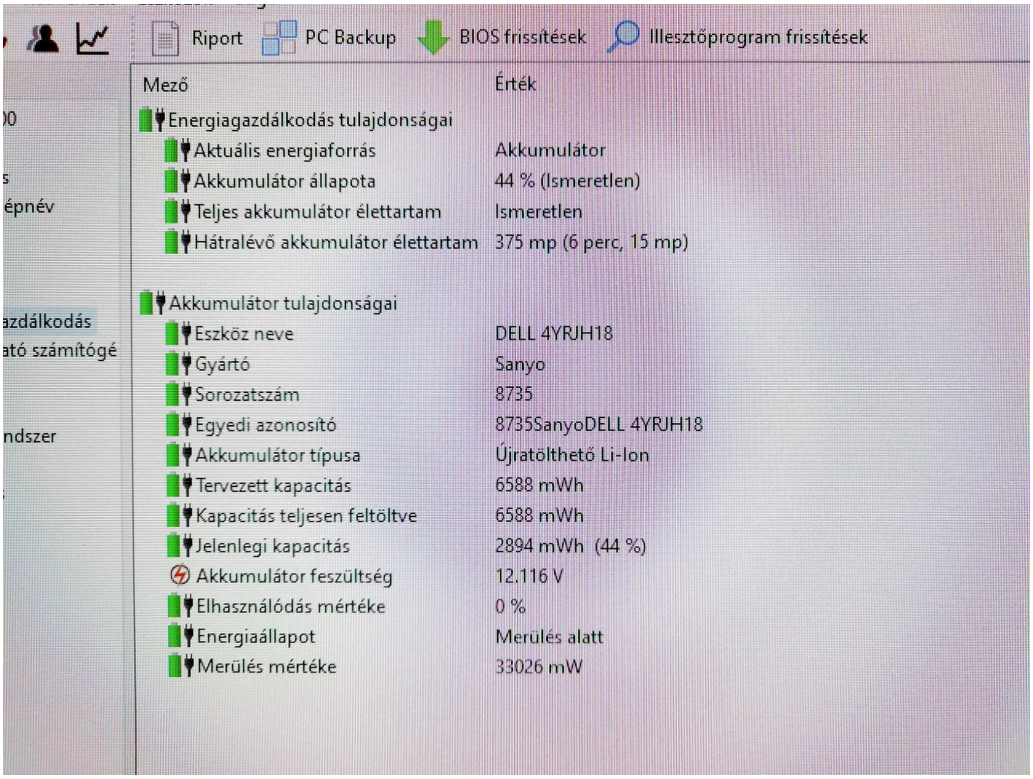
After carefully taking apart the casing (breaking only a small amount of clips) I was presented with 6 Li-Ion cells.

Original: **SANYO UR18650A** (2200 mAh)
Replacement: **SAMSUNG INR18650-35E** (3500 mAh)

I re-used the original spot-welded nickel strips and soldered them back using phosphoric acid surface wetter (75% H_3PO_4 / 25% H_2O; the cell's stainless steel cover requires special wetting for solder to properly melt on it). Before soldering I charged all 6 cells one-by-one to 4.20V.

I won't show the wiring. In fact you shouldn't do this at home at all. Li-Ion cells are extremely dangerous. I accidentally damaged the pink protection layer on one of the cells with my soldering iron and shorted the two poles… biiig spark ensues. Don't do this at home. Really. The image above doesn't show but I later added thick pieces of paper below the nickel strips to avoid further shorts.

As you imagined the story doesn't end here. After modifying the case to accommodate the longer cell pack and assembling the case the laptop did not start up at first. I plugged in the charging cable for a minute which revived the battery and the laptop started… for 5 minutes. The battery monitor showed that something is wrong.



Even though the cells were fully charged some factory values were automatically changed. The old cells or my way of disassembly probably messed up the BQ8050. Both Design Capacity and Full Charge Capacity were dropped from ~48000 mWh to 6588 mWh.

Meanwhile I learned that these batteries are "smart" and they have an I2C-like communication protocol (SMBus). All of them must conform to the Smart Battery Specifications. All right, this looks bad but in a good way. Now I get the idea why people don't just repair their batteries. The ultimate goal would be to rewrite capacity and cycle values to make the IC think it is in a brand new battery.

After reviewing the electrical requirements for SMBus I decided to connect my Raspberry Pi 2 Model B to the battery. The RPi2 has a 3.3V I/O level so it's perfect for this purpose. Although as per requirements I/O levels are good up to 5.5V.

Some configuration is necessary. First the I2C-bus has to be enabled. One way to do it is to modify two files. Open a terminal and edit the config.txt file with nano:

```
sudo nano /boot/config.txt
```

Add this line to the end of the file (or un-comment if present):

```
dtparam=i2c_arm=on,i2c_arm_baudrate=100000
```

Notice that there are two commands in the same line, one which enables the I2C-bus and after the comma the clock frequency is set to 100 kHz. Important: don't go above 100 kHz because SMBus doesn't support it. Hit Ctrl+X to exit nano, press Y and Enter/Return key to save the changes and exit.

Important: newer RPi boards handle frequencies differently. Check this it this applies to you.

Now you need to edit another file:

```
sudo nano /etc/modules
```

Add this line to the end of the file:

```
i2c-dev
```

Save and exit just like before.

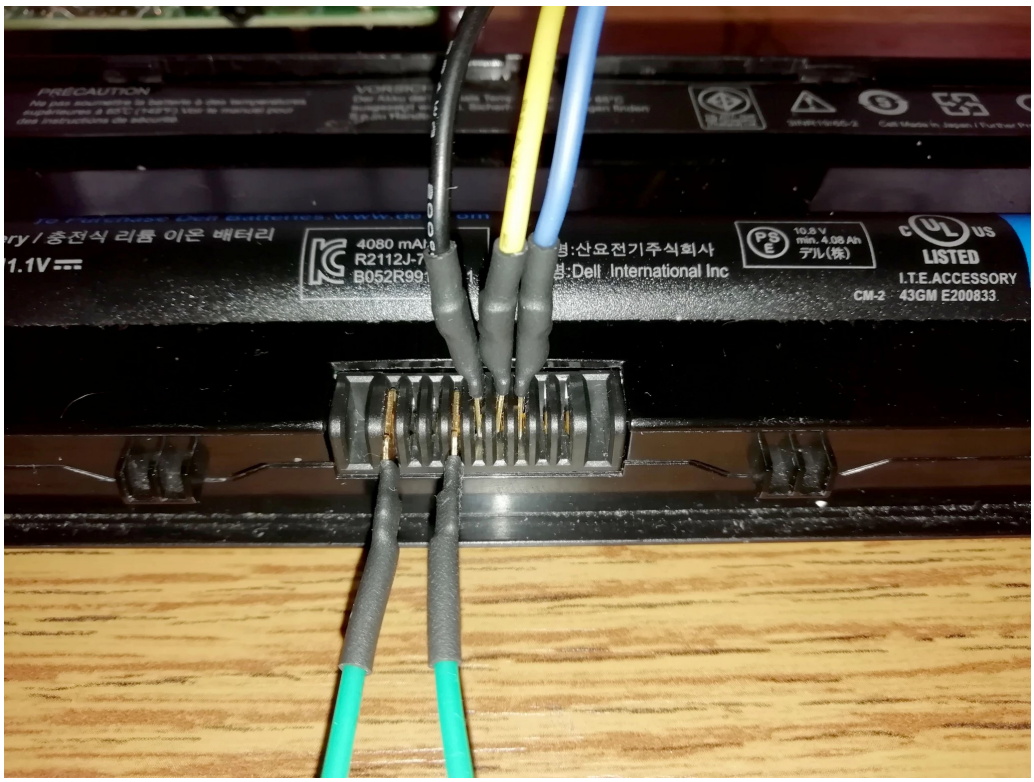
Finally reboot the RPi:

```
sudo reboot
```

After reboot make sure that the I2C-tools package is installed:

```
sudo apt-get update
sudo apt-get install i2c-tools
```

Now for the wiring. Laptop batteries can have different pinouts. This is how Dell J1KND batteries have their pins laid out:

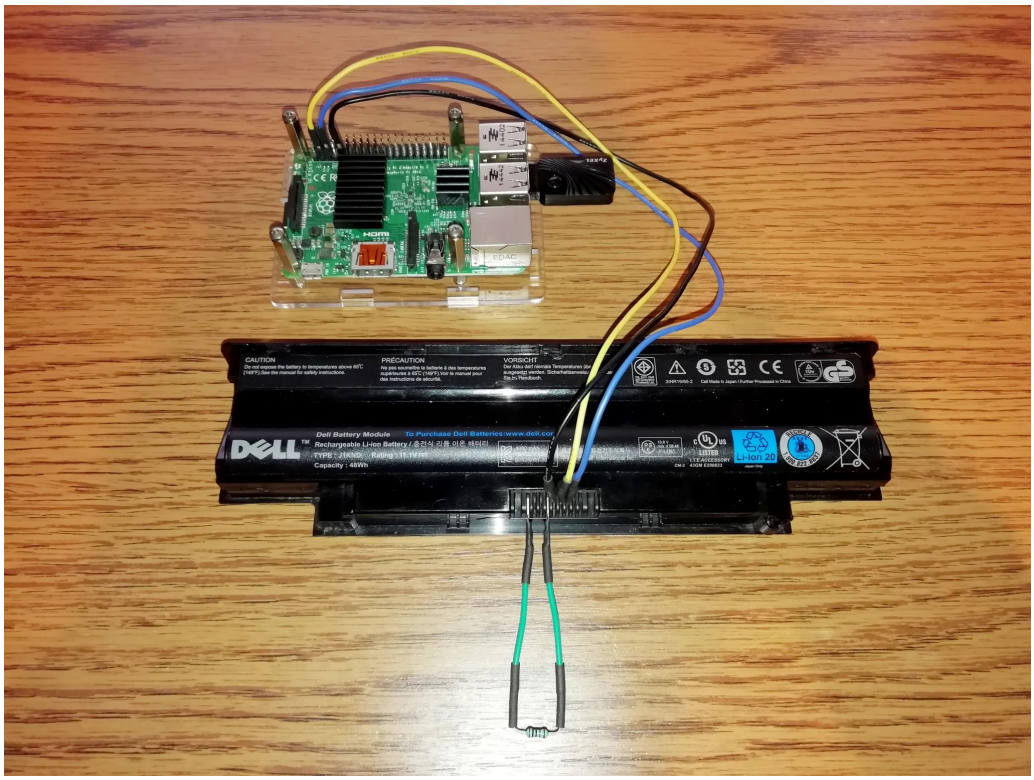


From left to right:

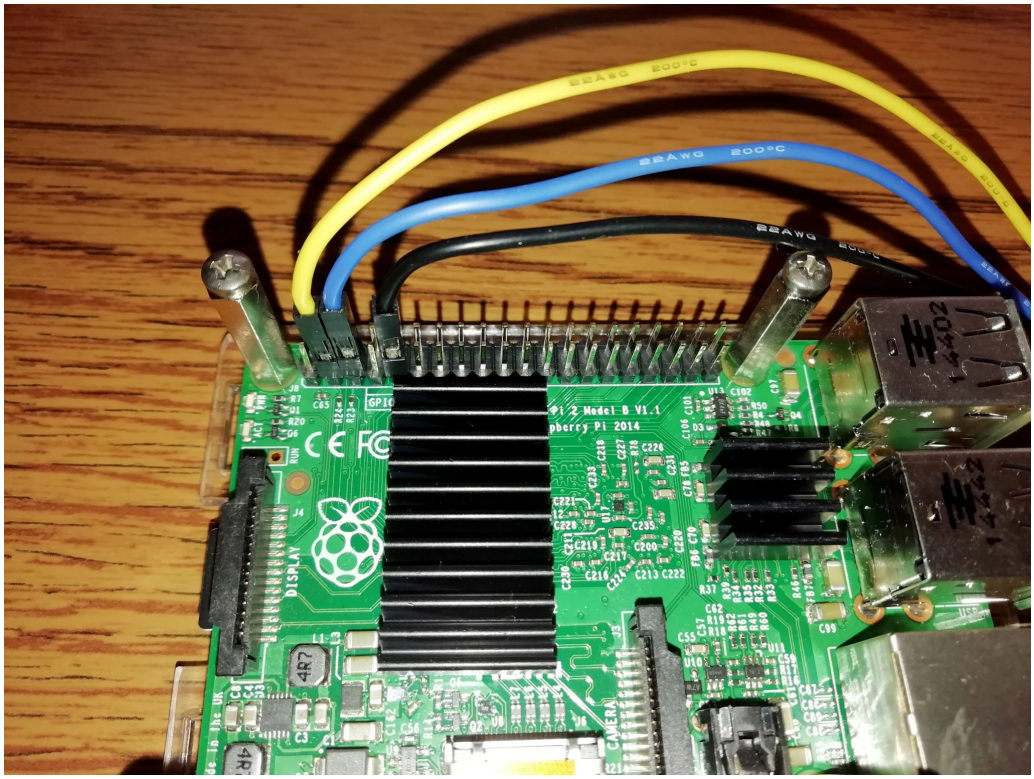
```
1: BAT- (GND)
2: BAT- (GND)
3: THERMISTOR? (probably)
4: SYSTEM PRESENT
```

5: GND
6: SDA
7: SCL
8: BAT+
9: BAT+

To wake up the battery the SYSTEM PRESENT pin has to be connected to ground. For this purpose I made a jumper cable with a 1 kOhm resistor in series (100-1000 Ohm is okay to use):

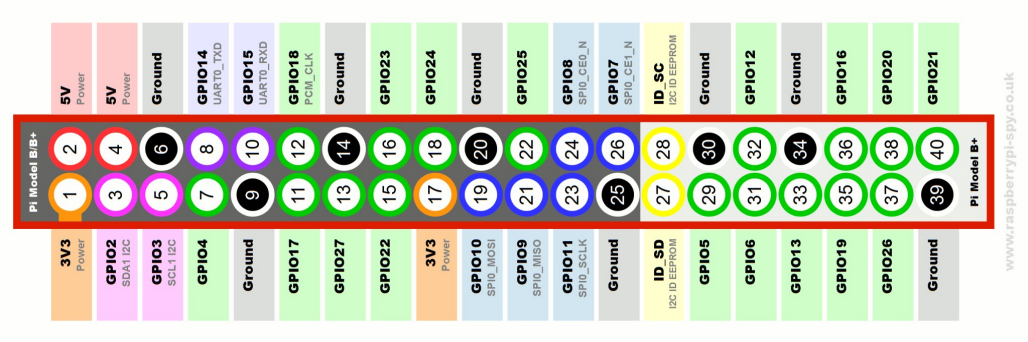


The connection on the RPi2 side is as follows:



Three wires are necessary: SDA, SCL and GND. No external pullup resistors are needed because the

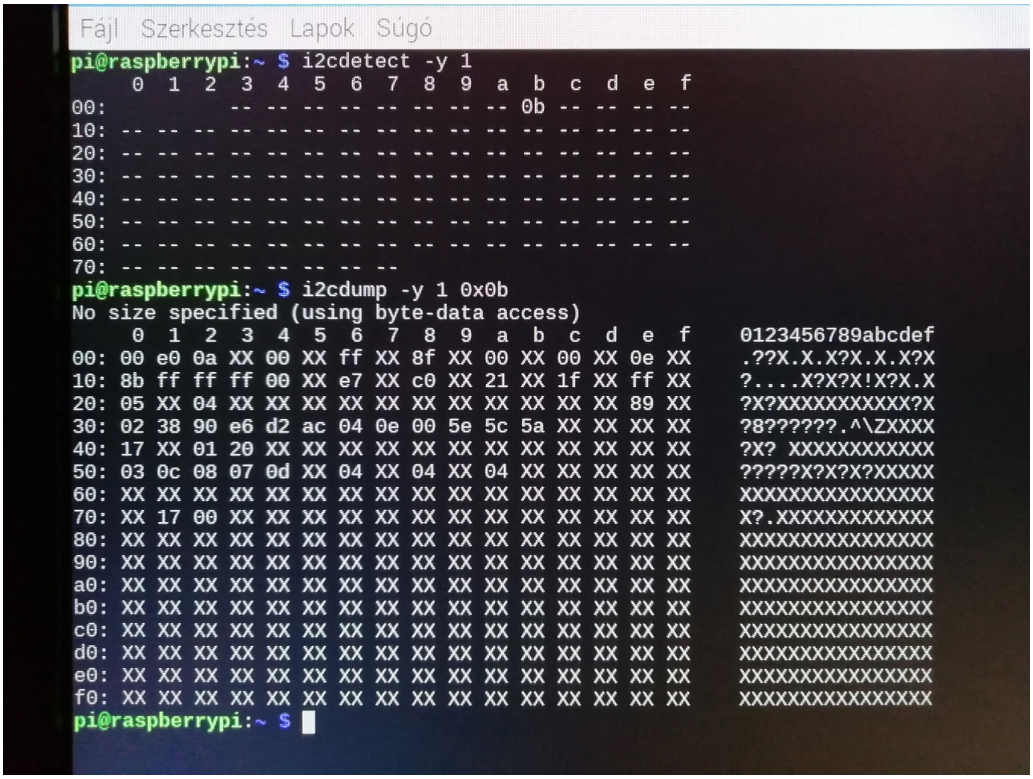
RPi2 has them onboard (R23 and R24 right next to the connector). Raspberry Pi 2 Model B pinout:



Now lets see if the RPi2 sees the battery. Open a terminal and issue this command:

```
sudo i2cdetect -y 1
```

This tool searches for responding I2C/SMBus devices. The search shouldn't take longer than a blink of an eye. If the tool slowly scans through the addresses or returns every address as available then check your wiring (I've been there).
In my case it found the battery at address `0x0b`:



As a quick probe you can request values from the device's memory addresses with this command:

```
sudo i2cdump -y 1 0x0b
```

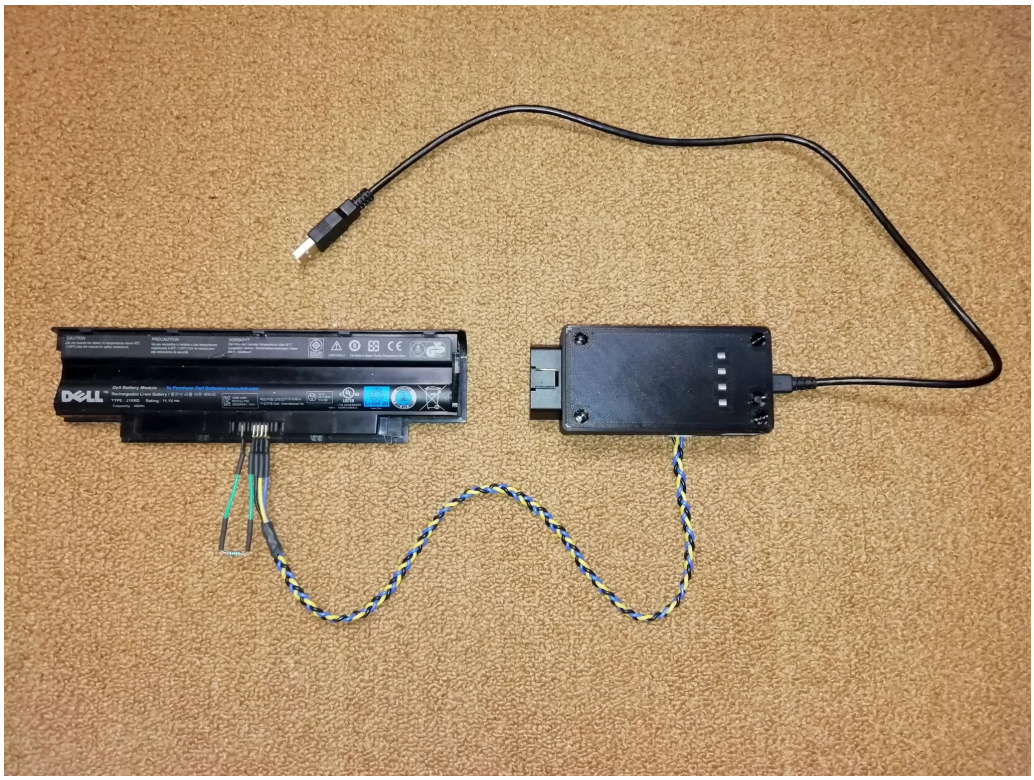
where the -y flag makes the task smooth, 1 is the I2C-bus number and 0x0b the I2C-device address.

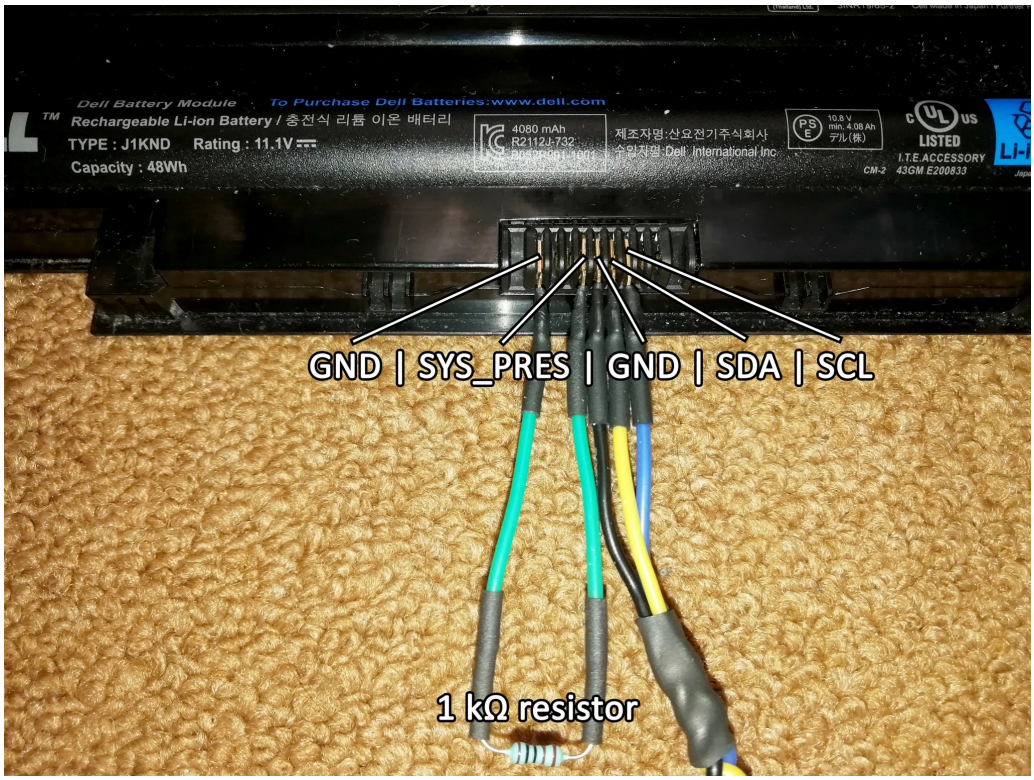
The BQ8050 usually returns more than 1 byte as a response but this with these flags the tool only reads 1 byte. Good enough to see what's what.

As it turns out this battery doesn't have a separate EEPROM chip where settings are stored. BQ8050 has both the firmware FLASH and EEPROM.

To be able to modify dynamic data we have to put the IC in unsealed mode. Now this will be a challenge.

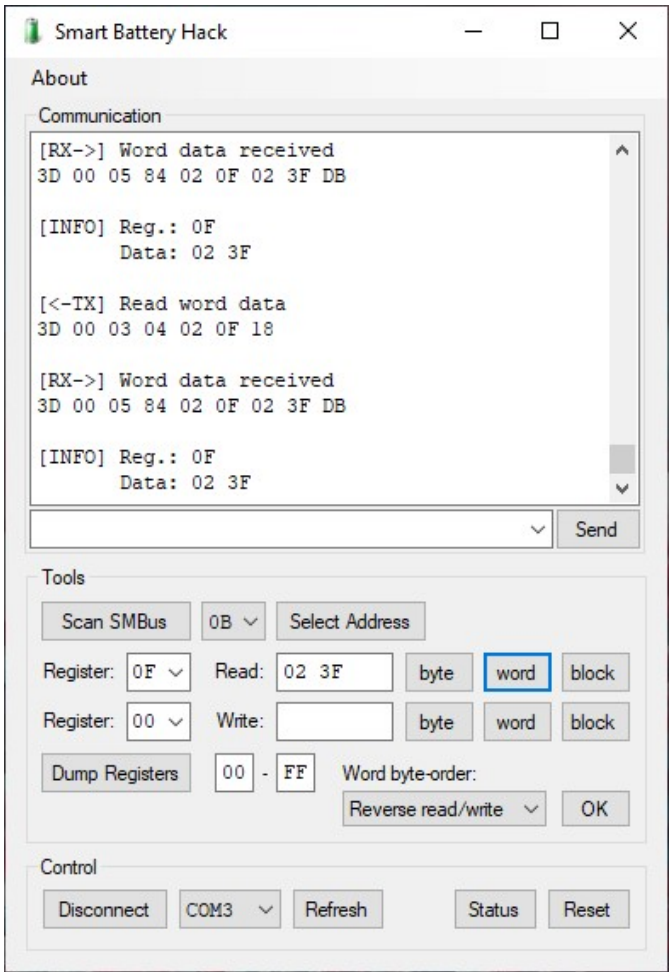
Alright, I won't push the RPi2/Python3 approach, instead I'm using an Arduino project of mine (Chrysler CCD/SCI Scanner) that has an I2C-connector with pullup resistors to +5V.

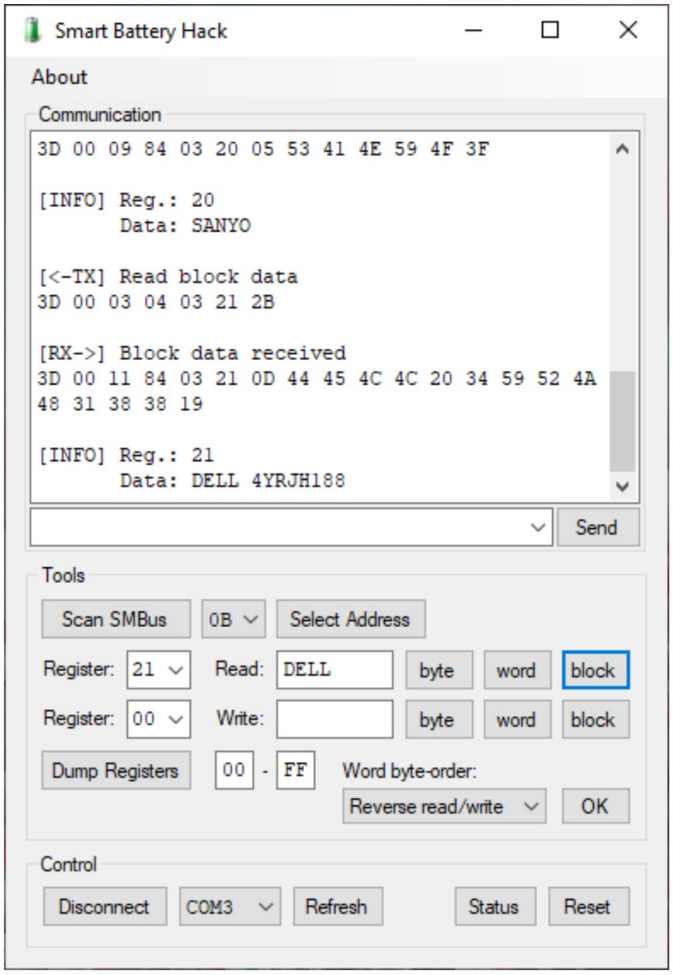




This Arduino code here should run on Uno/Mega boards:
https://github.com/laszlodaniel/SmartBatteryHack

In the GUI application you should be able to connect to the Arduino board and send commands through USB link.





If you're interested in the USB messaging protocol then you can check it out here. I used my already developed protocol with slight changes. The data code byte's highest bit is zero if the packet is sent from an external computer to the Arduino and it's one if the packet is sent from the Arduino to an external computer.

91 thoughts on "The repairing and hacking of a Dell J1KND (BQ8050) laptop battery"

2020.02.07 at 01:19

Very well written information. It will be supportive to anyone who utilizes it, including me. Keep up the good work – can'r wait to read more posts.

Hairstyles

★ Like

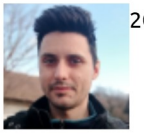
Pingback: Green Cell Pro (DE01PRO) J1KND laptop battery review | Boundary/Condition

2020.03.04 at 17:46

5ch4um1

Had a super strange experience with one of these on a windoze10 machine. I tried to poke around on it, writing 0x0214 to 0x71 unlocked some new commands. i tried the “read from 0x73, substract that from 10000 and write it to 0x71, which doesn’t seem to work. I then tried writing all possible values from 0000 up to ffff to 0x71, and monitoring the return value of a read from 0x54 to see if something changes. So by brute force i was able to turn that ff0d into a 000d. the word write needed seems to be different every time though, so it probably still depends on 0x73, but i couldn’t figure out the logic behind that yet. I couldn’t get any further, so i put it back into the laptop, still didn’t charge. Then, a few hours later, not even thinking about the battery, i was using an external keyboard, so i installed the spanish language pack and made the spanish keyboard the default. And it seems that this somehow convinced the battery to charge again. Then i tried with a second battery, turning ff0d into 000d and inserted it, e voila, it charges… no idea why or how, i’d love to get a logic analyzer to see what is actually happening… 😀

★ Like



★Daniel Laszlo

2020.03.06 at 19:44

Check if the data you read from 0x73 is constantly changing. For me it’s a milliseconds counter and it updates every 200 ms if I remember correctly. I’m not sure how to proceed from there. The turning of ff0d to 000d seems random to me. It changed for me when I did completely unrelated reads/writes.

★ Like

5ch4um1

2020.03.07 at 19:12

Yes, it changes, that’s what i refered to with ” the word write needed seems to be different every time though, so it probably still depends on 0x73”
and again yes, seems to be between 230 and 270 miliseconds.
and no idea if random or not, i just wrote every possible value, starting from 0000…
the write that changed ff0d to 000d in 0x54 was a different one each time.
and no idea if 0x54 is relevant to this at all, or what happend in general… 😀

Just got 2 more batteries to play with. Interesting thing: they both have different cells and a different pcb, obviously different from yours as well.
They both use the bq29330 protection ic, and a bq20889 as the gas gauge?
Btw, to which of them am i talking actually? (the bq29330 seems to have a i2c interface as well?) and, do they have their eeprom integrated, like the bq8030? or could these two other rather huge ass chips on the pcb acutally be eeprom and if so, could i probably talk to it directly by attaching wires to it?
And another funny thing, one of those that i could ‘repair’ had this string somewhere inside it’s memory: DELL 4YRJH
the case said J1KND… no idea, i bet it had yet another pcb…