iPhone DFU MODE Troubleshooting

By DeeGee

This article is about DFU Mode Trouble shooting.

To some, this mode is a god send but to others it's just a pain so let's get our heads around DFU and what role it plays in our repairs....

THIS IS NOT A PLZ BRO SOLUTON!!!

I am providing the information to allow you to understand these circuits and once understood you can then work out for yourself where the faults are....

Let's Go.....

DFU Mode is a mode in which a device's Boot ROM code waits to be recovered over USB.

```
# iPhone7,x iBoot bootloader build config
PLATFORM        : =     t7000
SUB_PLATFORM    :=      t7000
ARCH            : =     arm64
BOOT_CONFIG             :=      nand
HW_TIMER        :=      architected
AMC_REG_VERSION :=      4
AMC_FILE_VERSION :=     2
ADBE_VERSION    :=      2

# iPhone8,x iBoot bootloader build config
PLATFORM        : =     s8000
ARCH            : =     arm64
BOOT_CONFIG             :=      nand
HW_TIMER        :=      architected
ADBE_VERSION    :=      2
TARGET          : =     n69
```

First off we need to know some more about what the DFU mode is so let's gets the boring out of the way first.

The reason we need this information is this will allow us to determine software or hardware or both...

So we have iPhone Modes of operation as follows:

NORMAL = Once the device is powered on it boots normally into the iOS using the Apple Secure Boot chain sequence.

This secure boot chain and all apps are checked for Apple signatures to verify its integrity.

The SECURE BOOT ROM is normally the first part to be loaded/executed in normal mode.

Sequence:

1) The Secure BootRom contains Apples CA root public key which is used to verify the signature prior to the next stage being loaded.

2) The AP executes the code from the Secure BootRom

3) The Secure BootRom then verifies the LLB (low level bootloader) signature and if verified then loads the next stager.

4) The LLB then runs its code and carries out its duties before verifying the next stage for loading this will be the SSBL / iBoot

5) SSBL /iBoot will then verifies its section and then loads the iOS kernel.

6) The iOS kernel then runs the iOS Applications.

The Secure Boot Chain is there to ensure that the iOS only runs or boots from signed Apple Media / storage / devices

When the iOS is in this NORMAL state it's possible to recover ALL data that would be available to the user.

So this means that if any part of the sequence fails or fails to validate then the secure boot chain of events halts and stops, this in turn then let's the AP know to send a notification image on to the screen, this image is the recovery screen.

This recovery screen (connect to iTunes) is to allow the user to upgrade the iOS or fully restore the iOS from its downloaded signed *.ipsw file.
So in this mode we can only upgrade or restore the devices iOS data.

This restore mode has issues of its own by way of a restore boot loop that can arise from Jailbreaking or data

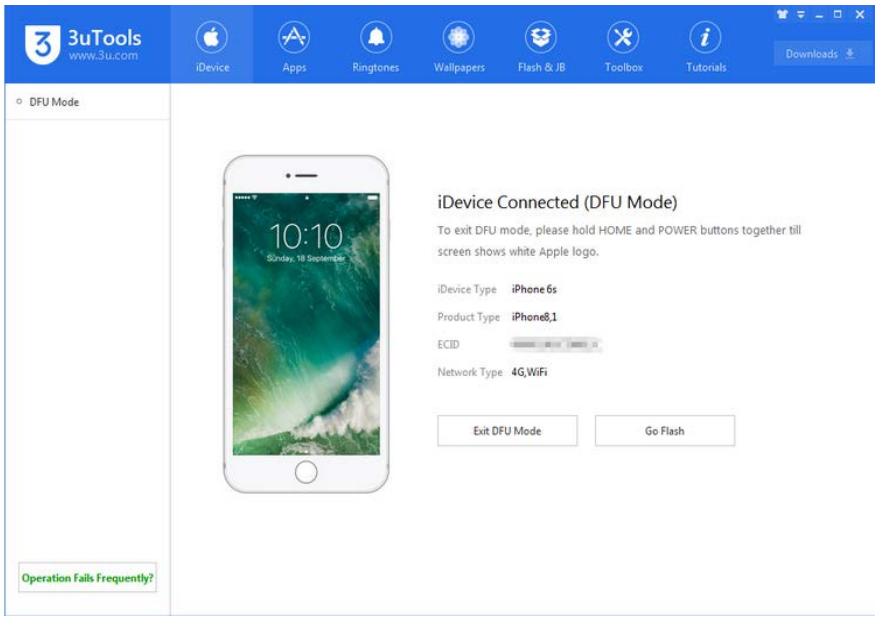
corruption of the eNAND Storage.

The only repair in this state is eNAND backup-restore only. Of underlying data.

So we have covered NORMAL mode and RESTORE mode, now for DFU mode....

DFU MODE:

During the Secure Boot Chain sequence if the Secure Boot ROM cannot be verified or accessed due to corruption then the AP defaults the device to DFU Mode (Development Firmware Upgrade/Update).



This is a low level diagnostics mode for performing Firmware upgrades to the iOS and the bootloader sections if corrupt.

Basically the DFU mode allows for the download and transfer/repair to the device over USB of Firmware files and ROM files for the SECURE BOOT sequence.

The boot sequence is slightly different for DFU mode:

1*) The Secure BootRom is executed and verifies stage 2*

2) Once verified then the iBSS and iBSEC are both loaded

iBSS is a custom version of iBoot which verifies the iBSEC loader for loading

3) Once these are loaded the iBSEC then verifies and loads the kernel.

4) The kernel then verifies once more and loads the RAMDISK into memory.

DFU Mode is the number one mode for forensic data acquisition and therefore should be for data recovery. (Not simple file backups)

(i.e.) UFED / CELLEBRITE specify DFU mode.

Using this mode an investigator prior to iPhone 4s could side load any usb media and run its own live cd / bootloader and access the Devices storage just like an usb drive. That why there is a secure signed boot chain.

SOFWARE CORRUPTION BEHAVIOUR:

Software failure of the BootROMS are as follows:

1) Device

2) BootRom
----------> Failure = DFU
3) LLB
----------> Failure = RESTORE
4) iBoot
----------> Failure = RESTORE
5) Kernel (iOS)

Please also note that the Baseband Modem has its own secure boot chain provided by either Qualcomm / Intel TSMC and therefore has its own BOOTLOADERS.

HARDWARE RELATED TESTING:

So thus far we know that DFU mode kicks in if we have a corrupt secure boot chain process or a corrupt IPSW/iOS system files but wait there's more….

We also have missing voltages, shorts, bridges, hardware IC failure.

Let's take a look at some failures on different devices and see how they compare if at all and what lines failing cause a default DFU Mode instance.

iPhone 7 Series DFU MODE Testing Schedule:

There are 3 main areas where these components or circuits attached to them cause DFU MODE:

1) AP/CPU/RAM

2) NAND

3) I2C

This may be by a short to ground, dropped pads, open circuit, low/high pull ups (unwanted), and supply failure (PMIC) or just a faulty component.

This is a hardware based test, the software boot sequence faults can be deduced by the device behavior and errors.

The checks to follow will cover most if not all of the above three.

Prelim: Known good Dock, Battery and so on trying to reduce our fault variables.

1) Check your DFU logs in iTunes or 3Utools for any information relating to the DFU mode instance.

2) Check ICs that can affect voltages across the board:

*U0700
*U1801
*U4001
*U2101
*U2301

3) Check Main power supply resistances and voltages: SHORTS/BRIDGES/OL

DFU MODE TESTING SCHEDULE:

LINE	VOLTAGE (v)	OHMS	DIODE	TEST POINT
90_TRISTAR_DP1_CONN_P	0	>2.5M	0.7	**TP0402**
90_TRISTAR_DP1_CONN_N	0	>2.5M	0.7	**TP0403**
90_TRISTAR_DP2_CONN_P	3	>2.5M	0.7	**TP0404**
90_TRISTAR_DP2_CONN_N	3	>2.5M	0.7	**TP0405**
PP_TRISTAR_ACC1	3	>1M	0.5	**TP0406**
PP_TRISTAR_ACC2	0.28	>1M	0.5	**TP0407**
PMU_TO_SYSTEM_COLD_RESET_L	1.8(h)	0.100M	0.45	**C2001**
AP_TO_PMU_SOCHOT_L	1.8(h)	69K	0.5	**C2012**

PP5V0_USB	5	0.2M	OL	**C2101**
PP5V0_USB_RVP	4.9	0.25M	0.67	**C4006**

PP_BATT_VCC	2.9	>1M	0.55	**TP0415**

PP_VDD_MAIN	0.25	>1M	0.55	**C1857**

PP_CPU_VAR	0.8	8	0.007	**PP1401**
PP_GPU_VAR	0	8	0.006	**PP1402**
PP_SOC_VAR	0.7	9	0.01	**C1822**
PP1V8_SDRAM	1.8	50K	0.3	**C1823**
PP1V8	1.8	48K	0.35	**C1602**
PP1V1_SDRAM	1.1	>500	0.3	**C1874**
PP1V1	1.1	0.40K	0.3	**C1528**
PP0V9_SOC-FIXED	0.9	30	0.02	**C1840**
PP_CPU_SRAM_VAR	0.8	80	0.35	**C1458**
PP_GPU_SRAM_VAR	0	380	0.38	**C1439**
PP1V2_SOC	1.2	60K	0.5	**C1927**
PP1V2_REF	1.2	60K	0.6	**C1932**
PP1V1_XTAL	1.1	0.318K	0.3	**C0704**

PP1V2_SOC_PCIE_REFBUF	1.2	60K	0.5	**C0802**

PP1V25_BUCK	1.25	50K	0.6	**C1804**
PP3V0_TRISTAR_ANT_PROX_	3	80K	0.45	**C1933**
PP1V8_ALWAYS	1.8	35K	0.6	**C1925**

PP3V0_NAND	3	>1M	0.45	**C1719**
PP0V9_NAND	0.9	>1M	0.3	**C1704**
PP1V8_VA	1.8	200K	0.3	**C3209**

PPVDD_BOOST	4.27	>1.5M	0.3	**C2303**
PP1V8_TOUCH	1.8	500K	0.55	**C3932**

PMU_VPUMP	4.5	>13M	0.78	**C1902**
PP0V9_SOC-FIXED_PCIE	0.9	35	0.06	**C0804**
PP1V1_DDR_PLL	1.1	2.2K	0.35	**C1523**
PP1V2_PLL_CPU	1.2	440K	0.54	**C1606**
PP1V2_PLL_SOC	1.2	440K	0.52	**C1609**

FORCED DFU BY HARDWARE
TEST POINT

PMU_TO_AP_FORCE_DFU	1.8	/		**TP0414**

DFU REQUESTS 1/2

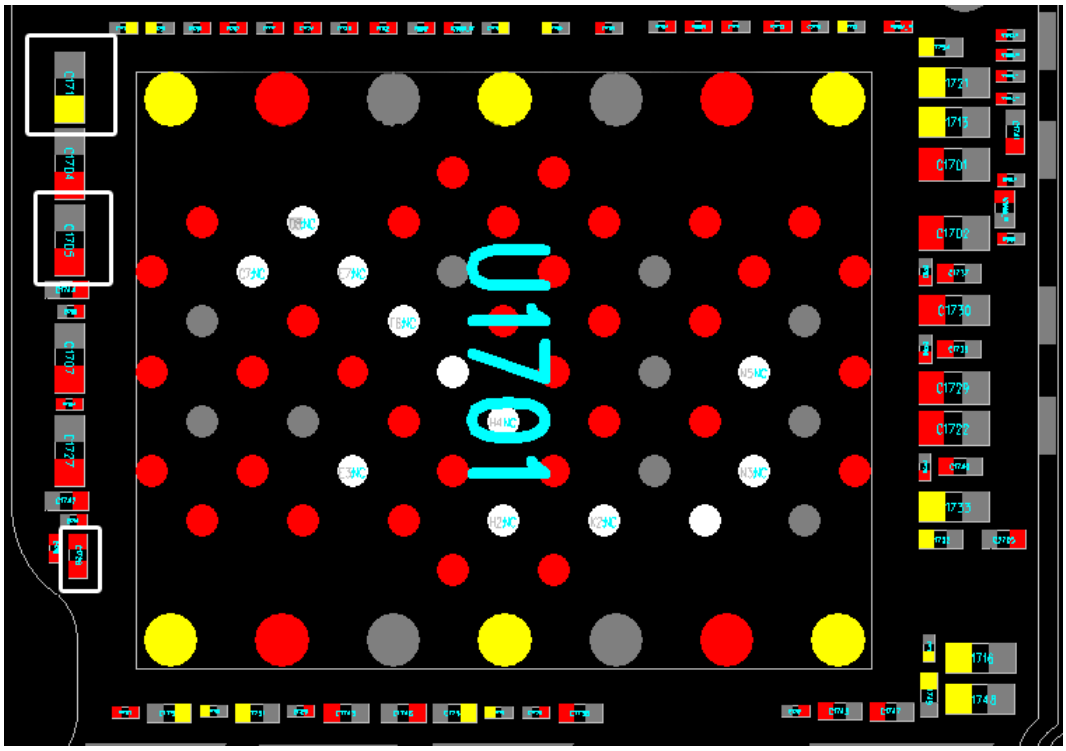
PMU_TO_AP_BUF_POWER_KEY_L	/	/	/	/
PMU_TO_AP_BUF_VOL_DOWN_L	/	/	/	/

If you find abnormal readings or voltages check the main PMIC is outputting correctly.

Please be aware that ANY other line attached can cause the fault on PMIC side including under the PMIC

4) Check eNAND power supply resistance and voltages: SHORTS/BRIDGES/OL

*PP3V0_NAND = 3.0V @ C1759
*PP0V9_NAND = 0.9V @ C1704
*PP1V8 = 1.8V @ C3209



What can happen to the above three voltages to create an instance of DFU mode kicking in?

1) Any Short circuit to ground of the 3V0_NAND supply will default to DFU

2) Bridging of the 3v supply to 1.8V supply will cause the 1V8 to be pulled high to 3V and the i2C bus cannot pull low enough to operate and can default to DFU Mode.

3) Missing 1V8, a whole host of issues there as VDD Main will not be produced for starters.

4) There are also eNand detection resistors failing that will cause NO DETECT of eNAND and default to DFU mode.

Then there is also the Touch circuit of 1.8V that can also pull down the i2C bus resulting in 50mA DFU mode.

There are BUCK lines that can pulled too low to allow the inductor to boost it correctly and in that case the inductor needs testing on BOTH sides to find out where the fault lies.

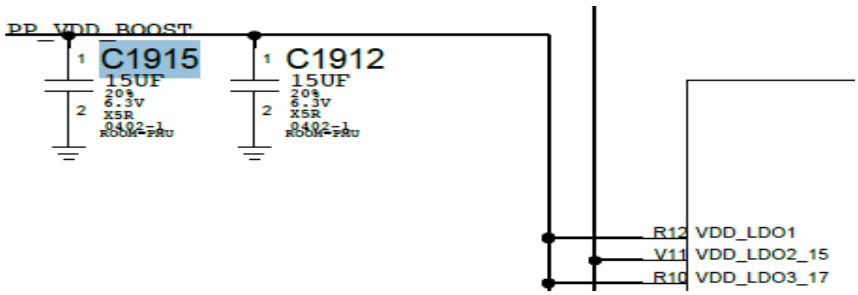
Example:

1V8 SDRAM pulled low we need to remove the inductor and test both pads to check if it's a PMIC failure.

If it's ok then you can provide a proper 1V8 tap-off from another source.

VDD_BOOST issues can be checked by testing U2301 for poor connection, shorts etc.

*C1915
*C1912

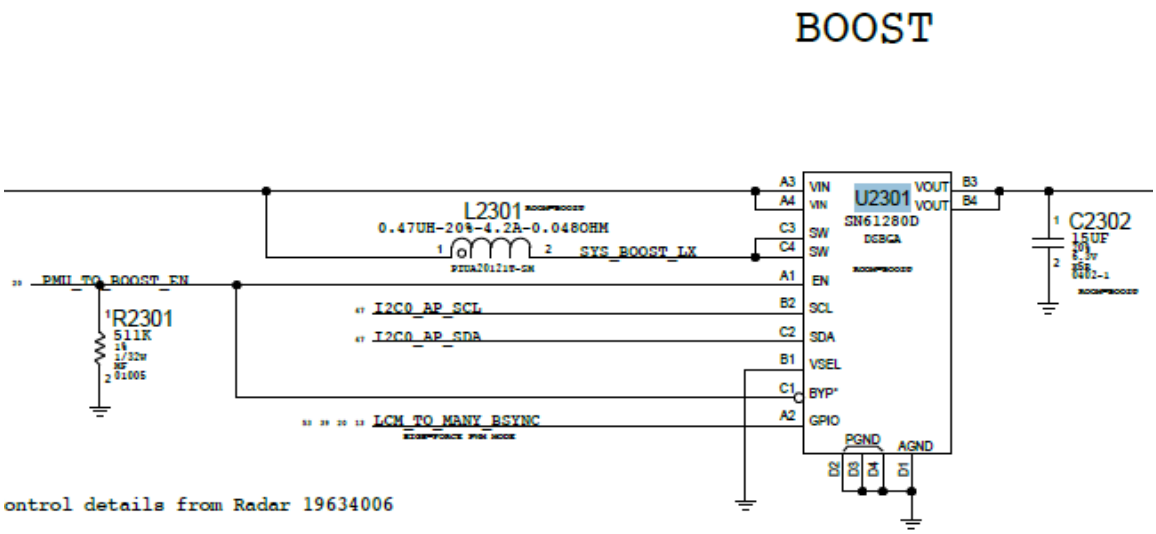


If VDD BOOST is abnormal then the BUCK line outputs will be affected. These are calculated for induction in design.

VDD BOOST handles LDO:

LD01
LD03-17
LDO5
LDO6
LD07-8
LDO11-13

U2301:



When VDD Main is at 3.4V U2301 is at the same voltage as VDD MAIN once triggered the impedance between the VDD BOOST and VDD MAIN is very low almost a short and the voltage is boosted by inductor L2301 switched boost operation.

So the U2301 will be at a higher voltage once triggered.

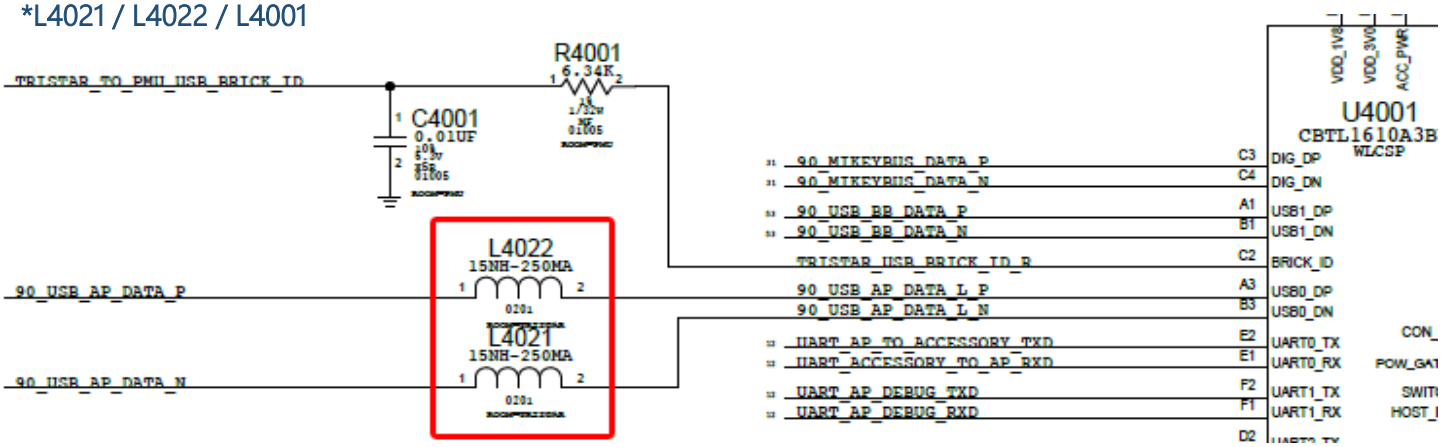
So VDD VDD_BOOST boosts the VDD MAIN voltage if its low at say 2.7v then the inductor will boost to 3.4v to power system.

ANY LOWER THAN 2.7V AND IT WON'T WORK CORRECTLY.

DP1 LINES:

Test inductors: For short or open circuit, these can become shorted from an attached line that IS short to ground.

*L4021 / L4022 / L4001



PMU TESTS: VDD MAIN PRESENT

1) PMU_IREF = 1.2V @ R0211

2) PMU_VREF = 1.2V @ C2006

3) TIGRIS_TO_PMU_INT_R_L = @R2103

4) PMU_VDD_RTC = 1.55V @ C2002

5) TRISTAR_TO_PMU_HOST_RESET

If the above lines are ok then replace the PMIC

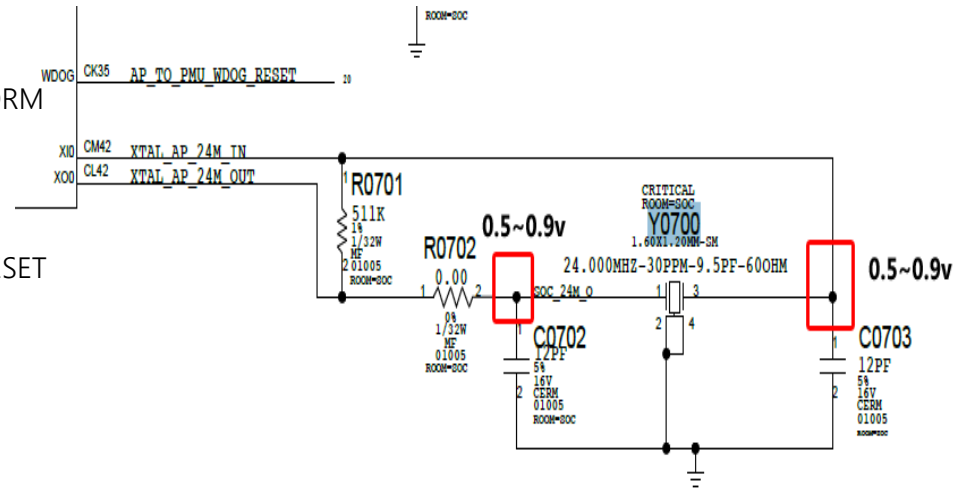
VOLTAGE JUMP/FLUCTUATION: VDD_MAIN PRESENT.

TEST:

1) Y0700 0.5-0.9V / WAVEFORM

2) FL0700

3) PMIC_TO_PMU_HOST _RESET



If any of these fail there will be voltage jump/fluctuations and the failure of "TRISTAR_TO_PMU_HOST_RESET" can result In the PMIC not being able to complete a reset and turns ON/OFF repeatedly.

No VCC_MAIN voltage: PWRPON / TRIGGER TEST:

Turn on the PWRON Button "BUTTON_PWR_KEY_L" circuit and check for VDD Main voltage at inductor L4001

U4001 TESTS:

1) Dock transmits low level signal to U4001 on TRISTAR_CONN_DET_L.

2) At this stage U4001 opens the 5V signal on to L4001 via PP5V0_USB_RVP line.

3) U4001 internally generates its own 3V LDO output from the TRISTAR BYPASS line.

4) ACC1/+USB and ACC2/USB_REV_CON communicate with the AP/CPU

5) Once authentication has been established U4001 then sends a low level signal to U2101 controller to open the 5V line to always boot.

This signal is the "TRISTAR_TO_TIGRIS_TBUS_OFF"

Test the above for any abnormalities in relation to voltage jumps/fluctuation.

We also have the following resistors to check for our clocks:

24M

Y0700 - C0702-C0703 IN/OUT

R0701 - XTAL_AP_24M_IN

R0702 - XTAL_AP_24M_OUT

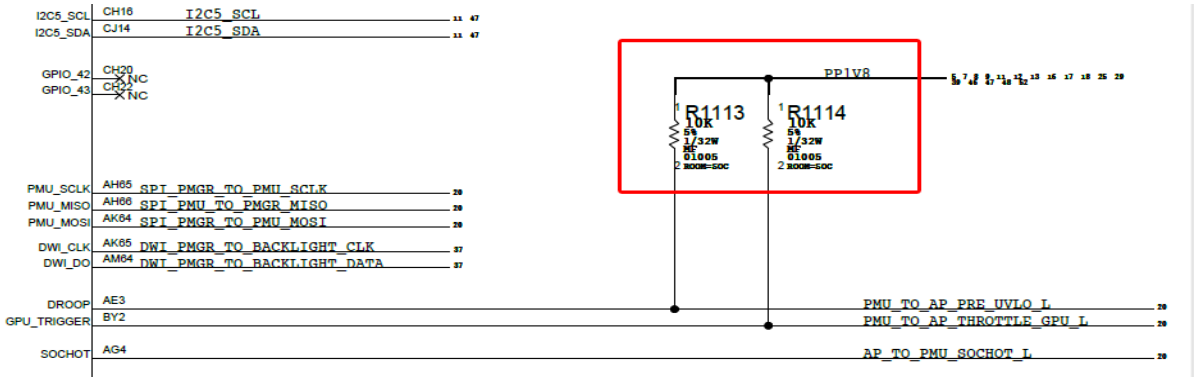
32M

Y2001 - C2003-C2004 IN/OUT

CPU DROP FUNCTION SWITCHING RESISTORS: (FOR U07000) CPU THROTTLING

R1113 - PP1V8 (PMU_TO_AP_PRE_UVLO_L)

R1114 - PP1V8 (PMU_TO_AP_THROTTLE_GPU_L)



There are also the DFU request shortcut command sequence lines to check:

DFU REQUESTS 1/2

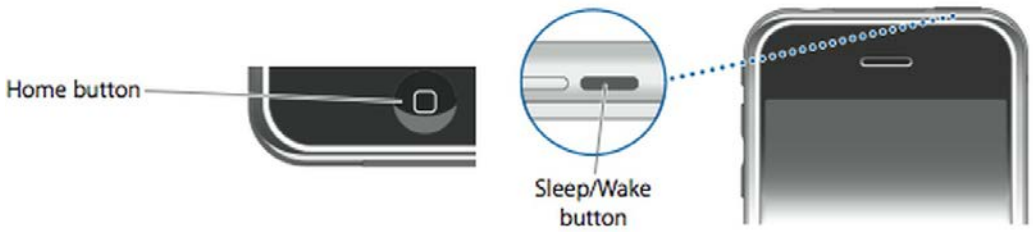
PMU_TO_AP_BUF_POWER_KEY_L

PMU_TO_AP_BUF_VOL_DOWN_L

If the above is faulty in any way then you will not be able to put the device manually into DFU mode as these lines are responsible for the power/home combo for DFU.

```
/* Check for Force DFU Mode Pin */
        dprintf(DEBUG_INFO, "Checking for Force DFU Mode Pin: %x / %x\n", force_dfu, force_reset);
        if (force_dfu && !platform_get_usb_cable_connected())
                force_dfu = false;

        /* Check for Force DFU Mode request pins. Requesting DFU mode is done
           by asserting both REQUEST_DFU1 and REQUEST_DFU2, holding them for
           the required time, then deasserting REQUEST_DFU1, followed by
           deasserting REQUEST_DFU2 after another required hold time.
```



Example 1:

So in this example we have a 6SP stuck in DFU mode and stuck at 50mA after PWRON / Trigger

From what we saw in the previous i7 series example let's take a look at this device against our schedule of tests.

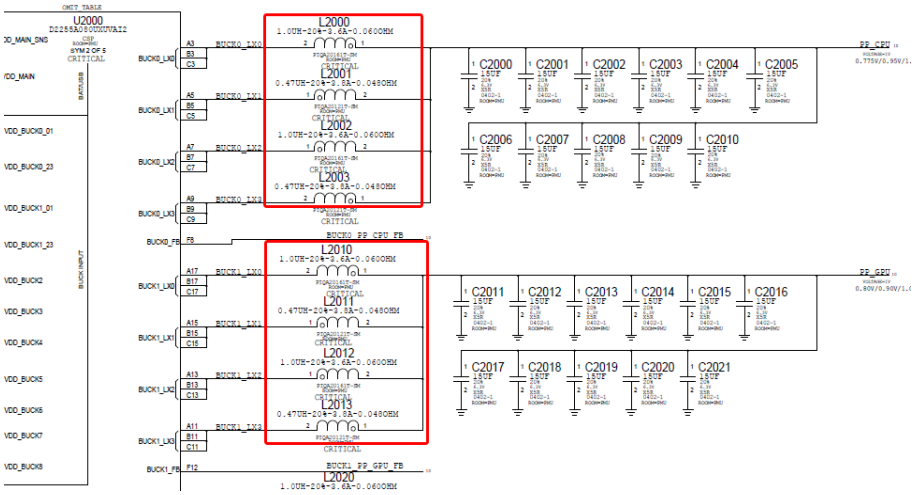
1) CPU

2) NAND

3) I2C

CPU:

1) Check Buck output inductors to CPU:



PP_CPU = 0.89V [PASS]

L2000
L2001
L2002
L2003

PP_GPU = 0V [PASS] C2016

L2010
L2011
L2012
L2013

PP_SOC = 0.825V [PASS] C2026

L2020
L2021

PP1V8_SDRAM = 1.8V [PASS] C2032

L2030

PP1V1_SDRAM = 1.1 [PASS] C2044

L2040
L2041

PP_FIXED = 0.84V [PASS] C2050

L2050

PP_CPU_SRAM = 0.79V [PASS] C2070

PP_GPU_SRAM = 0V [PASS] C20850

PP1V2 = 1.2V [PASS] C2114

FL3220 = 1.8V

C3220 = 1.8V

NAND CHECKS:

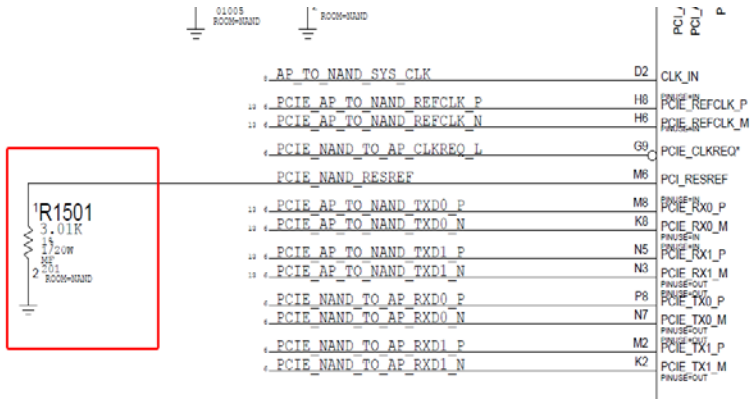
PP3V0_NAND = 3.0V [PASS] C1503

PP0V9_NAND = 0.9V [PASS] C1450

PP1V8 = 1.8V [PASS] C1530/R1530

NAND DETECTION:

RESREF resistor pulled down = 1.8/0V [PASS]



I2C CHECKS: (INC PP1V8 CONNECTION FEED)

R0900 SCL = 1.8V / 0.3V [FAIL] this line is pulled low and should be 1.8V on both incoming and outgoing ends of R0900.

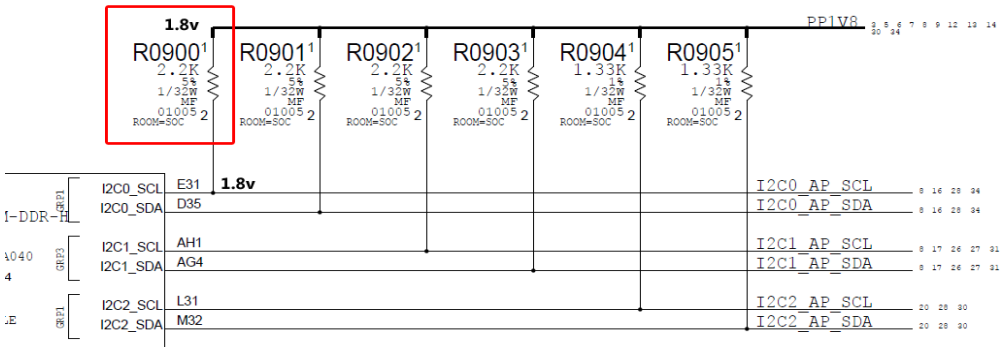
R0901 SDA = 1.8V / 1.79V [PASS]

R0902 SCL = 1.8V / 1.78V [PASS]

RO903 SDA = 1.8V / 1.8V [PASS]

R0904 SCL = 1.8V / 1.79V [PASS]

R0905 SDA = 1.8V / 1.8V [PASS]



Replace R0900 and retest on DCPS for current.

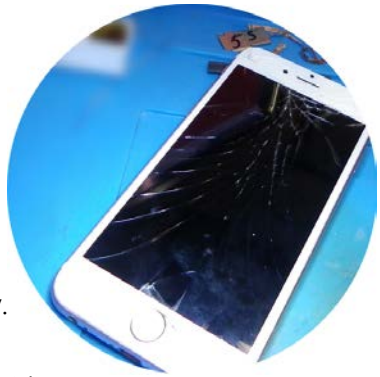
30mA/50mA/240mA/1A/800mA [PASS]

Connect to iTunes or 3uTools to test mode of operation. Phone is now in normal mode.

The fault was an I2C Line resistor putting the device into DFU MODE.

Example 2:

This 6s came in stuck in DFU mode and always gives the error "unknown error 14"



We couldn't kick it out of DFU so we looked further into it.

We know the CPU is working ok and we have voltages present.

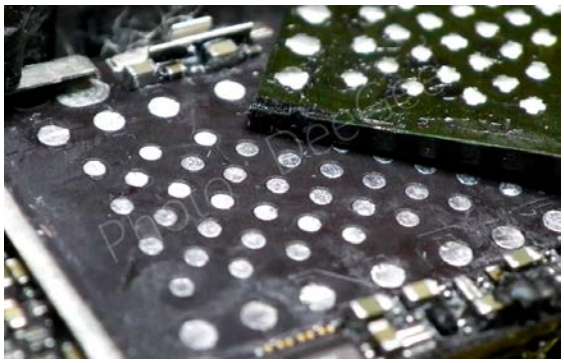
This error is related to eNAND flash storage failure.

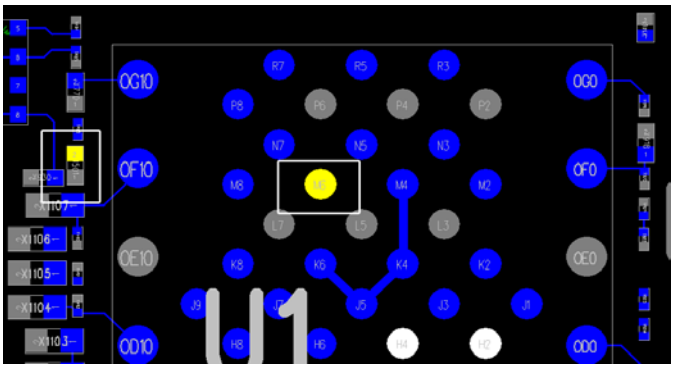
We check the power supplies and they are all good but still no joy.

Let's look at the NAND schematic and see what we have to work with.....

As there are tons of circuit lines to check the best way is to remove the eNAND and test each pad for resistance.

Picture shows iPhone 6 Nand removal (Quick 861DW – 300c / 50 = 125 seconds)





Picture shows 6s M6 and R1501

After testing we find that the pad M6 is open OL which is not right for that line. This line detects the NAND is present and if faulty will default the device to DFU Mode.

This line is pulled low to ground and the resistor is faulty and replaced.

The restore process can now be carried out in restore mode and no errors....

Example 3:

Here we have another 6s but this device is stuck in DFU but pulling PWRON / Trigger current of 100mA and stays there but no boot.

Plug the lightning cable into PC to check usb comms and we are in DFU mode according to 3uTools.

This is not normal current consumption for DFU Mode.

So if we remember our test schedule for DFU we have:

1) AP/CPU [PASS]

2) eNAND [FAIL]

3) I2C [NOT TESTED]

Well we know the CPU is ok as it carries out the partial restore and gives us an error.

Let's check the eNAND supplies...

* 3V0_NAND = 2.99v
* 0V9_NAND = 0.845V
* PP1V8 = 1.79V

All of the above are fine so we have good supplies into our eNAND so we either have dropped pads, corrupt data, or it could be an I2C issue still, we just don't know at this stage…

Again we remove the eNAND and test the pads all the resistances are normal and within range, so this is a physical eNAND failure.

We need to see if we can read the underlying data and we can but the eNAND is only readable so we copy this data before writing to the new chip.

That was our error while the restore was trying to write to the eNAND it couldn't due to this fault.

We write our new data to our replacement eNAND with our default codes which seem to increase over iOS updates. From 3 to 5 including battery and cameras.

This corruption of underlying data can also cause camera issues being reversed and a whole host of other issues, battery errors and so on so this underlying data integrity is very important.

eNANDs can take up to 400c for nearly 10 minutes before any data is at risk so these chips are quite strong protecting the data and most of the time its data corruption from bad .ipsw or USB comms from poor cables during restore causing corruption of the data transfer.

Example 4:

Here we have a 6 Plus with black screen won't turn on and consuming 60mA after PWRON / Trigger.

Again we plug into PC and we get detected as in DFU Mode.

Test Schedule:

1) AP/CPU [PASS]

2) eNAND [PASS]

3) I2C [FAIL] 60mA is I2C Related.

We measure the supplies for the AP and eNAND and all are present and within range.

Now do we remove the eNAND well not quite yet as the current is telling us that the eNAND is not active yet so let's move on to the I2C lines?

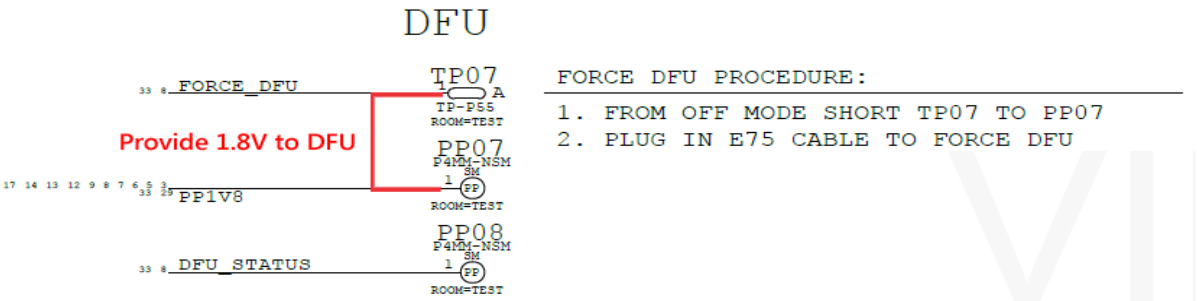
We test the I2C lines one by one from I2C0 first and methodically move our way on to the next BUS circuit until we have confirmed all is normal on I2C.

We find that the AP_BI_I2C0_SDA is at fault. Test Point PP0301

The resistor R0303 is to blame as we have 1.8v in and 0.7v out to the BUS circuit and it should read 1.8v also unless the bus is already pulled down in which case 0.7v is still too high for pull down.

Resistor Replaced and the phone boots normally and is now at 800mA and charging.

Before we finish we can't forget the FORCED DFU for us Board Repairers.



By providing 1.8V to the DFU Test Point we can FORCE the device into DFU mode.

Hope this helps a few.

Regards Dee