

# PMIC: First one to turn on and the last one to turn off

**Embedded Linux Conference 2019**

**Keerthy Jagadeesh**

# Agenda

- Introduction to PMIC
- Why do we need PMIC?
- Main components of PMIC
- Regulator driver overview
- Types of regulators
- Other common components
- I2C Debugging in u-boot
- I2C Debugging in kernel
- Hardware debug

# Introduction to PMIC

- Power management integrated circuits.
- Communicates with the host Processor via I2C or SPI.
- A PMIC is most commonly used in mobile phones & portable media players to decrease the amount of space required.
- Primary responsibility for implementing power on/off sequence.
- Feeding multiple voltage sources to SoCs
- Power supply types: LDOs, SMPS, DC/DC convertors
- May also provide RTC, GPIO, ADC, Watchdog & even thermal sensor support.
- Some PMICs also include Battery charging support.

# Why PMIC

- One stop solution for all the voltage requirements for the whole system including the SoC, RF, external codecs, modem etc.
- Even when multiple regulators are present PMICs offer granular voltage control. Ex: TPS65218.
- Enables deeper power management.
- ADC can be used to monitor variety of voltage and current parameters.
- Thermal sensor can be used to monitor the device temperature.
- RTC – Real time clock help keep track of time.

# Example: TPS65218

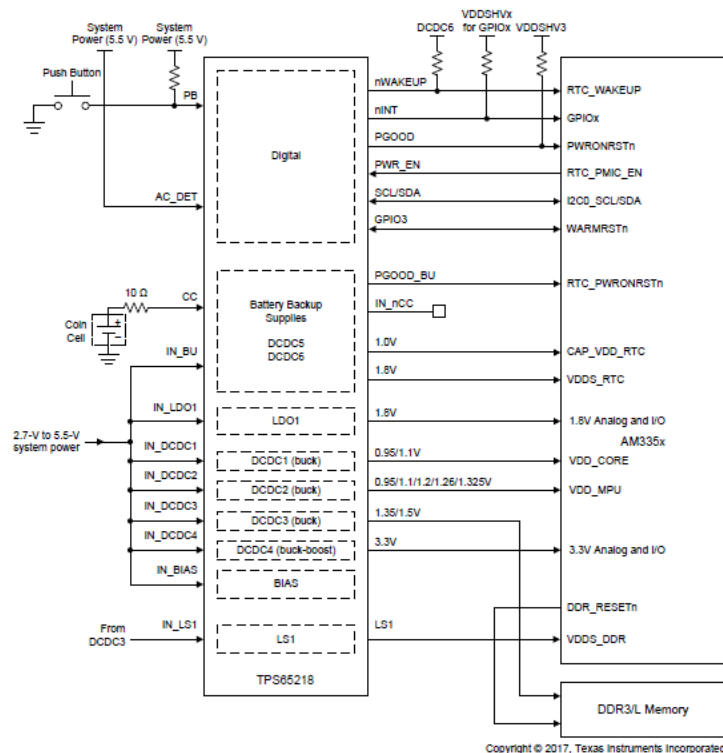
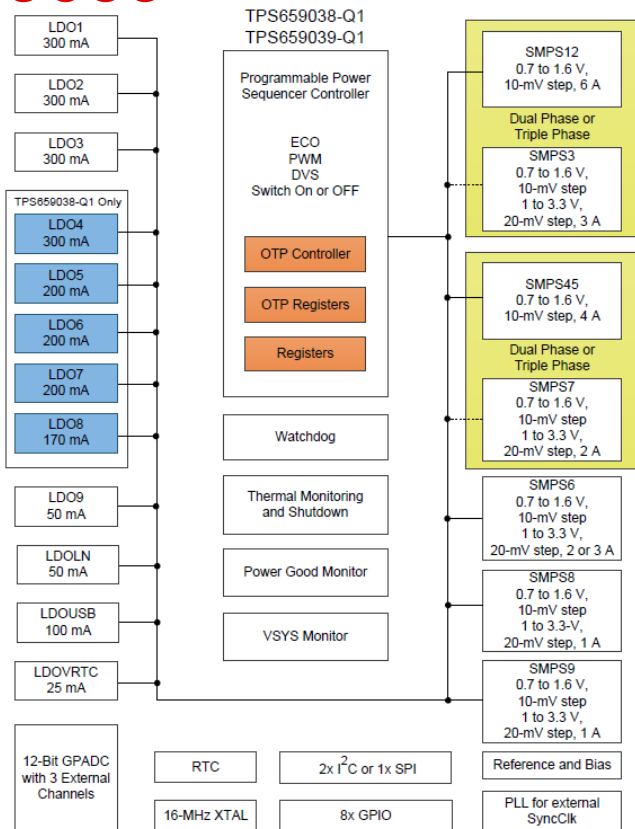


Figure 1. Connection Diagram for TPS65218 and AM335x

# Example: TPS659039



Copyright © 2017, Texas Instruments Incorporated

# DRA7 Power Tree

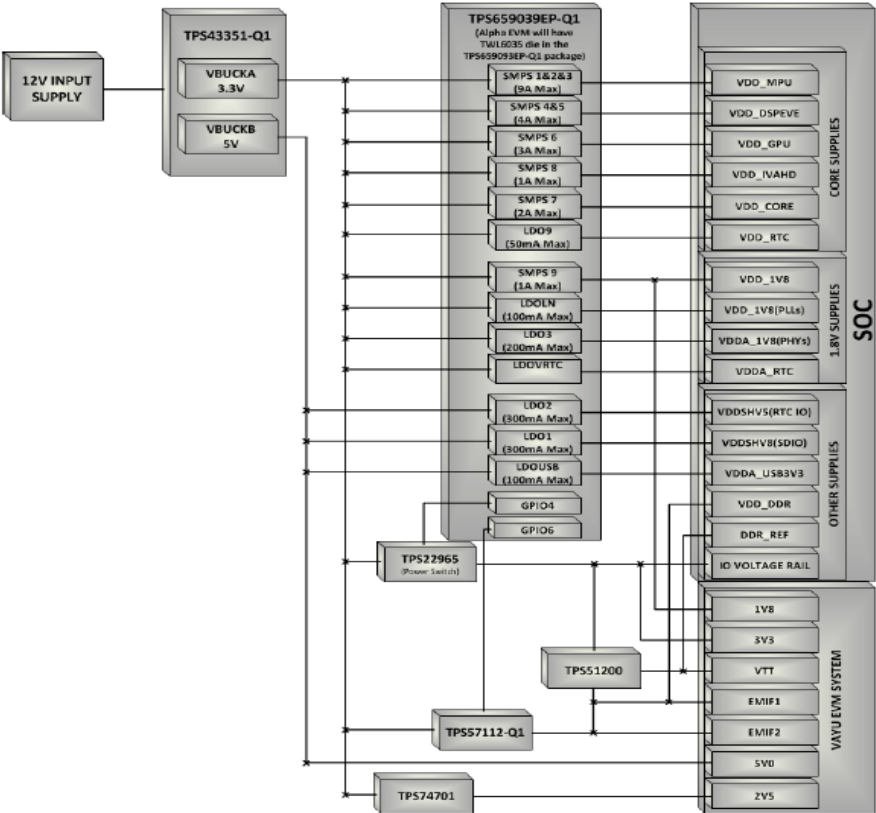


Figure 18. Power Blocks

# Types of voltage regulators

- **LDO - A low-dropout linear regulator:**

- LDOs are simple to regulate, create very little noise
- Requires no inductor for operation
- Produce a highly accurate output voltage
- Use of a pass transistor creates excessive heat, leading to reliability issues and low efficiency levels
- LDOs are most often used in low-power applications where these drawbacks are minimized by the nature of the application

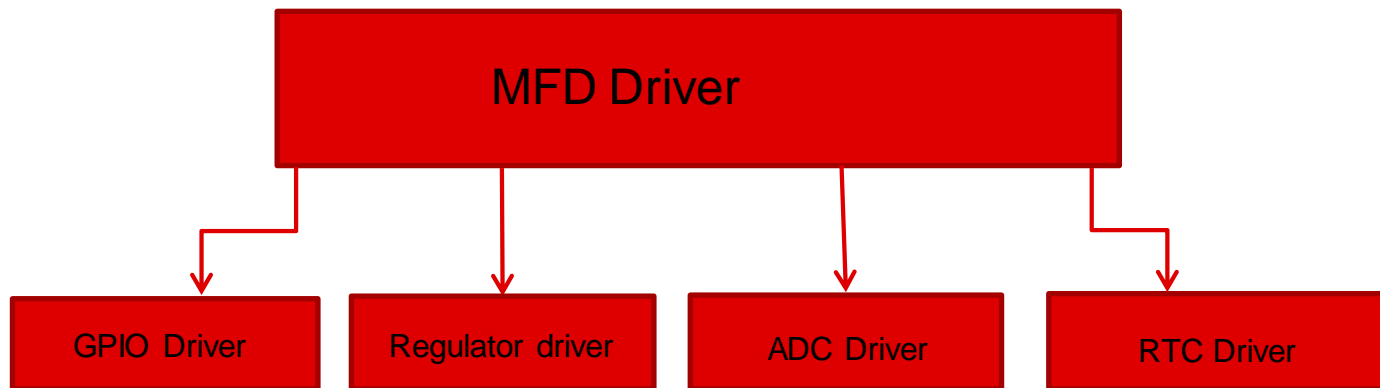
- **Switch-mode power supplies (SMPS):**

- Most popular power supply today because of their high efficiency
- Store energy in a capacitor or inductor, and repetitively switches its transistor on and off
- Low heat levels & SMPSs typically have efficiency levels above 80%
- High cost and complex design
- SMPSs can create high levels of noise and ripple, which decreases the accuracy of output voltage
- SMPSs are usually used in high power applications



# Linux modeling of PMIC

- MFD: It is very natural to model the PMIC as a multi function device. As it caters to Voltage, ADC, GPIO, thermal, RTC etc.
- Most certainly has regulator drivers.
- ADC/GPIO/Thermal/RTC drivers can also be present based on the HW.



# MFD Driver nuances

- Initialize the regmap - `devm_regmap_init_i2c`
- Register the interrupt handler & configure the interrupt controller as PMIC can potentially have multiple sources that can generate interrupts.
- Enable probing of child device drivers - `mfd_add_devices`
- Perform any PMIC specific pad configs
- Set any client specific data when multiple versions are supported.

# Device Tree Modelling with an example

```
&i2c0 {
    pinctrl-names = "default";
    pinctrl-0 = <&i2c0_pins>;
    clock-frequency = <100000>;

    tps65218: tps65218@24 {
        reg = <0x24>;
        compatible = "ti,tps65218";
        interrupts = <GIC_SPI 7 IRQ_TYPE_LEVEL_HIGH>; /* NMI in */
        interrupt-controller;
        #interrupt-cells = <2>;

        dcdc1: regulator-dcdc1 {
            regulator-name = "vdd_core";
            regulator-min-microvolt = <912000>;
            regulator-max-microvolt = <1144000>;
            regulator-boot-on;
            regulator-always-on;
        };
    };
};
```

# Regulator Driver nuances

- `devm_regulator_register`: Register all the regulators to the regulator framework.
- Populate necessary hooks for enable/disable. This is turn on/off the regulators.
- Populate get/set voltage hooks to read and write the voltage values.
- One can also set/get current limit hooks.
- Make use of `regulator_linear_range` to provide `vsel` to voltage conversion and vice versa.
- `map_voltage` to map the voltages
- `set_ramp_delay` : To take care of the slew rate that allows regulators ramp up and ramp down time.

# Types of regulators in Linux

- Fixed regulators: Voltage is fixed. Only on/off support.
- Variable voltage DC-DC convertors, SMPS, LDO regulators.
- Variable regulators cater to a range of voltages with fixed number of steps.
- Allows DVFS – dynamic voltage frequency scaling.
- Allows to lower voltages during low power modes.
- Variable voltage regulators are also used as cooling agents.
- GPIO Regulators: GPIOs can also be modelled as regulators where in the high value corresponds to a discrete value of voltage and the low corresponds to another discrete value.

# Regulator Device Tree nodes

```
dcadc1: regulator-dcac1 {  
    regulator-name = "vdd_core";  
    regulator-min-microvolt = <912000>;  
    regulator-max-microvolt = <1144000>;  
    regulator-boot-on;  
    regulator-always-on;  
};
```

```
ldo1: regulator-ldo1 {  
    regulator-min-microvolt = <1800000>;  
    regulator-max-microvolt = <1800000>;  
    regulator-boot-on;  
    regulator-always-on;  
};
```

# Regulator Device Tree nodes - 2

```
vpo_sd_1v8_3v3: gpio-regulator-TPS74801 {  
    compatible = "regulator-gpio";  
    regulator-name = "vddshv8";  
    regulator-min-microvolt = <1800000>;  
    regulator-max-microvolt = <3300000>;  
    regulator-boot-on;  
    vin-supply = <&evm_5v0>;  
    gpios = <&gpio7 11 GPIO_ACTIVE_HIGH>;  
    states = <1800000 0x0  
              3300000 0x1>;  
};
```

# GPIO Driver nuances

- `devm_gpiochip_add_data`: One time registration with the gpio framework
- `Gpio set/get` direction to either configure as input or output.
- `GPIO set get` to retrieve the value of gpio level.
- `Set_config` enables more config options.
- `Gpio_request`: optional hook for chip-specific activation, such as enabling module power and clock; may sleep
- `Gpio_free`: optional hook for chip-specific deactivation, such as disabling module power and clock; may sleep
- `Set_multiple`: Assigns output values for multiple signals



# TPS659038 GPIO Device Tree

```
• &i2c1 {
    status = "okay";
    clock-frequency = <400000>;

    tps659038: tps659038@58 {
        compatible = "ti,tps659038";
        reg = <0x58>;
        interrupt-parent = <&gpio1>;
        interrupts = <0 IRQ_TYPE_LEVEL_LOW>;
        #interrupt-cells = <2>;
        interrupt-controller;
        ti,system-power-controller;
        ti,palmas-override-powerhold;

        tps659038_gpio: tps659038_gpio {
            compatible = "ti,palmas-gpio";
            gpio-controller;
            #gpio-cells = <2>;
        };
    };

    gpio_fan: gpio_fan {
        /* Based on 5v 500mA AFB02505HHB */
        compatible = "gpio-fan";
        gpios = <&tps659038_gpio2 GPIO_ACTIVE_HIGH>;
        gpio-fan,speed-map = <0 0>, <13000 1>;
        #cooling-cells = <2>;
    };
};
```

# RTC Driver nuances

- Clear all the prior rtc interrupts
- Populate the rtc dev ops
  - .read\_time
  - .set\_time
  - .read\_alarm
  - .set\_alarm
  - .alarm\_irq\_enable
- devm\_rtc\_device\_register
- Register the rtc interrupt using devm\_request\_threaded\_irq

# TPS659038 RTC Device Tree

```
• &i2c1 {  
    status = "okay";  
    clock-frequency = <400000>;  
  
    tps659038: tps659038@58 {  
        compatible = "ti,tps659038";  
        reg = <0x58>;  
        interrupt-parent = <&gpio1>;  
        interrupts = <0 IRQ_TYPE_LEVEL_LOW>;  
  
        #interrupt-cells = <2>;  
        interrupt-controller;  
  
        ti,system-power-controller;  
        ti,palmas-override-powerhold;  
  
        tps659038_rtc: tps659038_rtc {  
            compatible = "ti,palmas-rtc";  
            interrupt-parent = <&tps659038>;  
            interrupts = <8 IRQ_TYPE_EDGE_FALLING>;  
            wakeup-source;  
        };  
    };  
};
```

# Debugging

- U-boot & kernel prints to start with to trace the i2c register values
- `Cat /sys/class/regulator/regulator*/*`
  - Microvolts gives the actual voltage value
  - State tells whether regulator is turned on or not
  - Microamps gives current drawn
  - Other sysfs nodes that give out a lot of information about regulator.
- Enable `CONFIG_CMD_REGULATOR` in u-boot to get the regulator state and voltage.

# U-boot I2C debug tools

- First need to assign i2c device using – i2c dev 0
- One can dump the i2c registers of a particular Slave ID using: i2c md 'slave\_id'
- One can modify the registers using any of the i2c mw or i2c mm
- Example: i2c dev 0

Setting bus to 0

```
i2c md 0x58 0x20
```

```
0020: 11 00 be 3a 00 00 be 3e 00 00 00 00 11 00 c7 32  ....>.....2
```

```
i2c mw 0x58 0x23 0x3B
```

```
i2c md 0x58 0x20
```

```
0020: 11 00 be 3b 00 00 be 3e 00 00 00 00 11 00 c7 32  ...;>.....2
```

```
i2c speed
```

```
Current bus speed=400000
```

# Kernel I2C debug tools

- `i2cdump [-f] [-y] [-r first-last] I2CBUS ADDRESS`

Command: `i2cdump -f -y 0x0 0x58`

Output: No size specified (using byte-data access)

0 1 2 3 4 5 6 7 8 9 a b c d e f 0123456789abcdef

00: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....

- `i2cget [-f] [-y] I2CBUS CHIP-ADDRESS [DATA-ADDRESS [MODE]]`

Command: `i2cget -f -y 0x0 0x58 0x23`

Output: 0x3b

# Kernel I2C debug tools - 2

- `i2cset [-f] [-y] [-m MASK] [-r] I2CBUS CHIP-ADDRESS DATA-ADDRESS [VALUE] ... [MODE]`

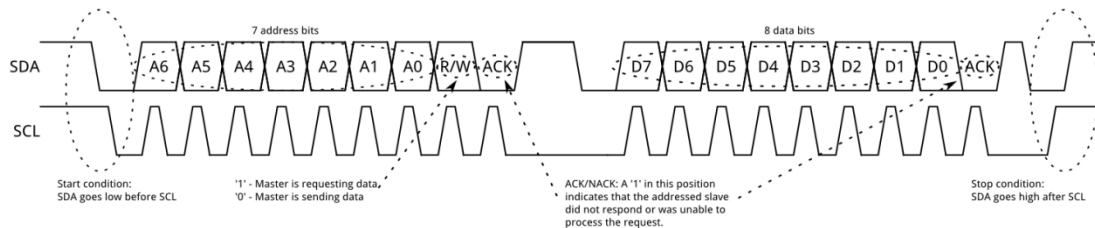
Command: `i2cset -f -y 0x0 0x58 0x23 0x3b`

Command: `i2cdump -f -y 0x0 0x58`

Output: No size specified (using byte-data access)

0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
00:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
10:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
20:	11	00	be	3b	00	00	be	3e	00	00	00	00	11	00	c7	32	?	?	;	..	?	>	....	?	?	2					

# I2C Protocol fundamentals

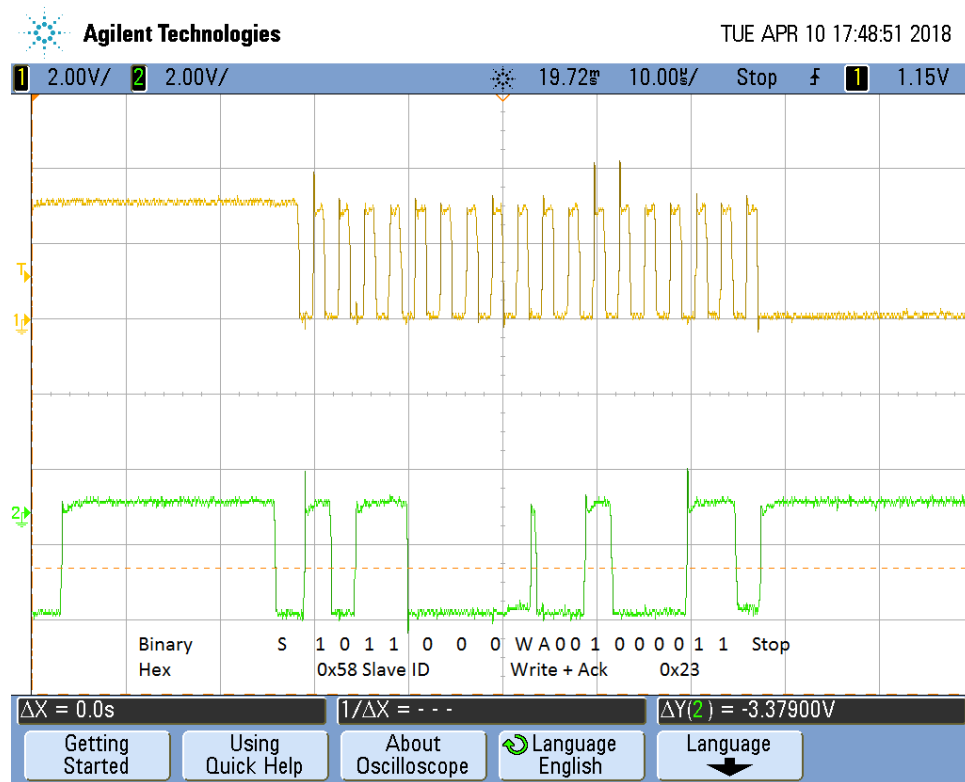


- **Start:** the master device leaves SCL high and pulls SDA low
- **Address Frame:** 7-bit address
- **R/W:** a R/W bit indicating whether this is a read (1) or write (0) operation
- **NACK/ACK bit:** receiving device pulls the SDA line low before the 9th clock pulse means ACK otherwise it is a NACK.
- **Data Frames:** The master will simply continue generating clock pulses at a regular interval, and the data will be placed on SDA by either the master or the slave, depending on whether the R/W bit indicated a read or write operation
- **Stop:** Stop conditions are defined by a 0->1 (low to high) transition on SDA after a 0->1 transition on SCL, with SCL remaining high



# I2C Hardware debug - 2

Example showing a normal i2c write sequence followed by Ack



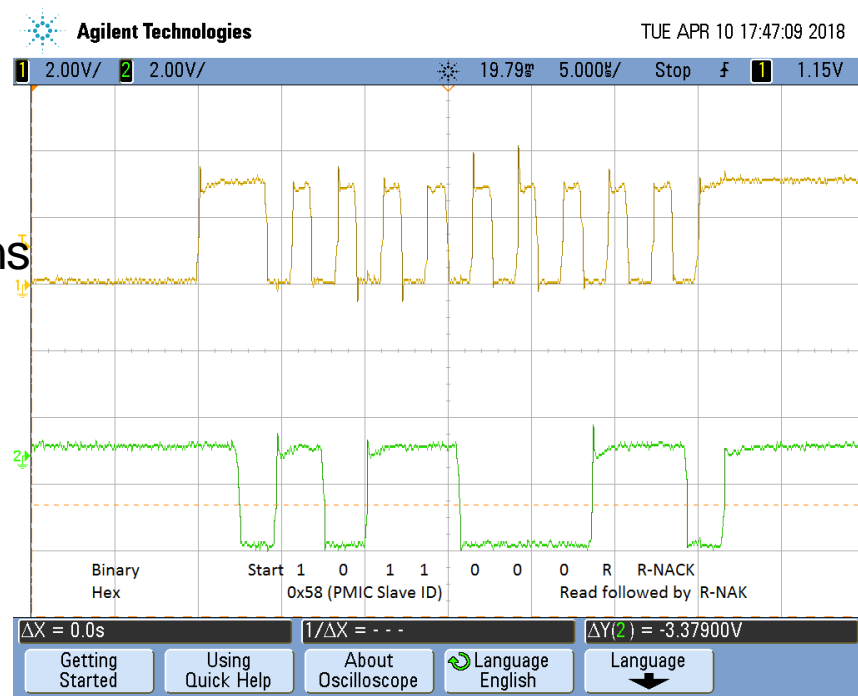
# I2C Hardware debug

Probe the I2C Clock and Data lines

Check with the standard expected patterns for read/write

Any discrepancy means HW is not behaving as expected

For example Read Nack



# I2C Hardware debug - 3

Total phase  
Aardvark I2C  
Protocol analyzer.

A simple tool that  
tracks every  
transaction.

Expects i2c clock  
and data lines  
connected.

Snoops on that two  
lines. Helps catch  
the buggy  
transaction. Ex:  
The one in the end  
has no data.

The screenshot displays the Total Phase Data Center v6.73.007 application window. The main pane shows a list of I2C transactions with columns for Index, mcs.ms.us, Dur, Len, Err, S/P, Addr, Record, and Data. The transactions are listed from 1691841 to 1691872. The right sidebar contains the Capture Control panel, the Navigator panel showing the I2C Bus structure, and the Statistics panel. The bottom status bar indicates the protocol lens is set to I2C, the bus is LiveFilter, and the info is displayed. The system tray shows the date and time as 4:15 PM on 4/10/2018.

Index	mcs.ms.us	Dur	Len	Err	S/P	Addr	Record	Data
1691841	10:20:862.748	56.6 us	1 B		S	58	Read Transaction	4F*
1691842	10:20:862.827	80.5 us	1 B		S	58	Write Transaction	23
1691843	10:20:862.908	56.6 us	1 B		S	58	Read Transaction	4F*
1691844	10:20:863.000	81.7 us	2 B		S	58	Write Transaction	23 37
1691845	10:20:866.950	85.6 us	1 B		S	58	Read Transaction	23
1691846	10:20:867.036	56.3 us	1 B		S	58	Write Transaction	37*
1691847	10:20:867.117	81.8 us	1 B		S	58	Read Transaction	20
1691848	10:20:867.199	56.1 us	1 B		S	58	Write Transaction	11*
1691849	10:20:867.278	80.6 us	1 B		S	58	Read Transaction	23
1691850	10:20:867.359	56.3 us	1 B		S	58	Write Transaction	37*
1691851	10:20:867.438	80.5 us	1 B		S	58	Read Transaction	23
1691852	10:20:867.518	56.1 us	1 B		S	58	Write Transaction	37*
1691853	10:20:867.597	80.7 us	1 B		S	58	Read Transaction	23
1691854	10:20:867.678	56.6 us	1 B		S	58	Write Transaction	37*
1691855	10:20:867.764	81.6 us	2 B		S	58	Read Transaction	23 40
1691856	10:20:870.868	81.0 us	1 B		S	58	Write Transaction	23
1691857	10:20:870.949	56.6 us	1 B		S	58	Read Transaction	40*
1691858	10:20:871.027	77.9 us	1 B		S	58	Write Transaction	20
1691859	10:20:871.104	56.6 us	1 B		S	58	Read Transaction	11*
1691860	10:20:871.179	77.4 us	1 B		S	58	Write Transaction	23
1691861	10:20:871.257	56.5 us	1 B		S	58	Read Transaction	40*
1691862	10:20:871.344	77.9 us	1 B		S	58	Write Transaction	23
1691863	10:20:871.422	56.5 us	1 B		S	58	Read Transaction	40*
1691864	10:20:871.498	78.0 us	1 B		S	58	Write Transaction	23
1691865	10:20:871.576	56.5 us	1 B		S	58	Read Transaction	40*
1691866	10:20:871.661	81.9 us	2 B		S	58	Write Transaction	23 4F
1691867	10:20:874.299	85.2 us	1 B		S	58	Read Transaction	23
1691868	10:20:874.384	56.6 us	1 B		S	58	Write Transaction	4F*
1691869	10:20:874.467	82.4 us	1 B		S	58	Read Transaction	20
1691870	10:20:874.549	56.7 us	1 B		S	58	Write Transaction	11*
1691871	10:20:874.629	81.6 us	1 B		S	58	Read Transaction	23
1691872	10:20:874.710	29.1 us	0 B		S	58	Write Transaction	

# Debug Summary

- **We Don't need a sledge hammer to break open the eggs.**
- Start with simple prints, sysfs entries & software debug tools.
- Only if the kernel is crashing get on to multimeter & oscilloscope.
- I2c protocol analyzers only when we suspect a hardware bug.

# Thank You

# References

- TPS65218 Public Data Sheet:  
<http://www.ti.com/lit/ds/symlink/tps65218.pdf>
- <http://www.ti.com/lit/ug/sprui50/sprui50.pdf>
- <https://www.kernel.org/> Linux Mainline kernel
- <http://events17.linuxfoundation.org/sites/events/files/slides/schulz-pmics-keep-power-in-your-hands.pdf>
- <https://www.motioncontrolonline.org/blog-article.cfm/LDOs-vs-SMPS-Power-Supplies-for-Industrial-Automation-Equipment/44>
- <https://manpages.debian.org/unstable/i2c-tools/i2cset.8.en.html>

# Back Up slides

- **-f** : Force access to the device even if it is already busy. By default, i2cset will refuse to access a device which is already under the control of a kernel driver. Using this flag is dangerous, it can seriously confuse the kernel driver in question. It can also cause i2cset to silently write to the wrong register. So use at your own risk and only if you know what you're doing.
- **-y**: Disable interactive mode. By default, i2cset will wait for a confirmation from the user before messing with the I2C bus. When this flag is used, it will perform the operation directly. This is mainly meant to be used in scripts.