Power(full) framework based on Driver Model

Przemysław Marczak
Samsung R&D Institute Poland

Embedded Linux Conference Europe
Düsseldorf, 13-th October, 2014
A few words about me...

Przemysław Marczak
p.marczak@samsung.com

The programming adventure was started about 2 years ago at Samsung R&D Institute Poland

Work activities
• Focus on the bootloader development for mobile devices
  – Started with some private projects
  – Now continue with the Open Source
• U-Boot development
  – Mainline – Maintainer of boards:
    • Odroid U3 (Exynos4412)
    • Universal C210 (Exynos4210)
  – Tizen.org – Adding features and sync with mainline
Agenda

• Introduction
  – The Aim
  – Project status

• The power requirements
  – Framework? What for?

• The present PMIC Framework
  – The story of improvements
  – Design and features

• The new PMIC Framework
  – Design assumptions
  – Usage of Drivel Model and Device Tree
  – Implementation

• The future plans

• Functionality
  – Quick comparison
  – User interface screens

13/10/2014
The aim

Today

• Present my conception of the PMIC Framework
• Get the feedback from the listeners

In the future

• Introduce a common and functional framework architecture
• Improve the functionality
• Port the drivers
• Remove old the PMIC Framework
The project

Status

• Still under the construction

• The basic functionality is achieved
  – Device classes:
    • PMIC – For Read/Write operations
    • REGULATOR – For regulator specific operations
  – Ported Drivers: MAX77686 (both class instances)
  – Ported Boards: OdroidU3 and Trats2

Code review – is welcome!

• The RFC patch set was send to list, a few days ago and is also available here:
  – https://github.com/bobenstein/u-boot.git
The Mobile Devices power requirements

- Stability and precision
- Many power source lines
- Specific Voltage values
- Control the power consumption
- Protection: Voltage and Thermal

source: samsung.com
There are lots of various PMIC types in the Embedded Systems
• Common functionality – Framework
• Control in a specific way – Drivers
Features

- **Regulator**
  - 9 of BUCK, with up to **5A** of load
  - 26 of LDO, with up to **450mA** of load

- **Voltage source**
  - Range: **0.8–3.95V**
  - Step: **25/50mV**

- **Operation mode**
  - Low Power Mode (LPM) – 1uA per LDO
  - ON/OFF/STANDBY

- **Integrated peripherals**
  - RTC with the calendar
  - Backup battery charger
  - Buttons: Power/Reset

- **Interface**: I2C
The PMIC Framework – What for?

- Common Framework API – Other Frameworks can use it!
- Clear driver coding style
- Drivers API – one per driver class
- User interface – one per driver class
  - For developers – magic commands
  - For users – clear commands
- Easy to extend by the new features
The present PMIC Framework - history

• October 2011 - The first version was introduced by Łukasz Majewski
  – Supports SPI or I2C – but not both
  – Only one device instance
  – Register read/write
  – Set LDO output
  – PMIC I/O command

• 2011/2012 – Porting the drivers
  – TWL6030
  – TWL4030
  – MC13892
  – MC34704
  – and more ...
November 2012 – Significant changes

- Improve the I2C interface support
  - Two bytes transmission
  - Sensor byte order [ BIG/LITTLE ]
- Multiple PMIC devices
- Change the directory: „misc” -> „power”
- Charger/Battery support
The present PMIC Framework - Architecture

- ALLOC
- GET
- PROBE
- READ
- WRITE

Power(full) framework based on Driver Model
The present PMIC Framework – Design for Trats2

Powerfull framework based on Driver Model
The present PMIC Framework - Strong points

• Supports I2C and SPI interfaces
• Provides many interface options
• Provides pmic command
• Tested for a long time – works fine!
The present PMIC Framework - Limitations

- One interface type (chosen at compilation)
- Only one device driver instance
- Architecture is strictly defined (*struct pmic*)
- Provides only raw read/write operations for PMIC regulators
Design assumptions

• Keep the current functionality
• Use the Driver Model
• Use the Device Tree
• Clear Architecture – one class – one functionality
• Add new features
• Extend the User Interface
Usage of the Driver Model

**Driver Model**
- Common driver API
- One class type - single device
- Many device instances

**Device Driver side:**
- Usage of common API
- Easy to implement (at least one func)

**User Interface:**
- Depends on common API
- Easy to implement
- Many feature possibilities
Usage of the Device Tree

Exynos4412-odroid .dts

[...] max77686: max77686@09 {
  compatible = "maxim,max77686_pmic";
  interrupts = <7 0>;
  reg = <0x09 0 0>;
  #clock-cells = <1>;

  voltage-regulators {
    ldo1_reg: ldo1 {
      regulator-compatible = "LDO1";
      regulator-name = "VDD_ALIVE_1.0V";
      regulator-min-microvolt = <1000000>;
      regulator-max-microvolt = <1000000>;
    }
    ldo1_reg: ldo2 {
      regulator-compatible = "LDO2";
      regulator-name = "VDDQ_VM1M2_1.2V";
      regulator-min-microvolt = <1200000>;
      regulator-max-microvolt = <1200000>;
    }
    ldo1_reg: ldo3 {
      regulator-compatible = "LDO3";
      regulator-name = "VCC_1.8V_AP";
      regulator-min-microvolt = <1800000>;
      regulator-max-microvolt = <1800000>;
    }
  }
[...]

Exynos4412-odroid .dts

Power(full) framework based on Driver Model
Regulator operation mode and descriptors

**LDO2CTRL1**
Control OUT2 (LDO2) ON/OFF and Programmable Output

<table>
<thead>
<tr>
<th>Address (hex)</th>
<th>MODE</th>
<th>Type: O</th>
<th>RESET:0xD0</th>
</tr>
</thead>
<tbody>
<tr>
<td>41</td>
<td>R/W</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>BIT</th>
<th>Name</th>
<th>POR</th>
<th>Description</th>
</tr>
</thead>
</table>
| 7:6 | OPMODE | 11 | 00: OFF (regardless of PWRREQ)  
01: Output ON/OFF controlled by PWRREQ  
PWRREQ=H (1) : Output ON in Normal Mode  
PWRREQ=L (0) : Output OFF  
10: Output ON with Low Power Mode by PWRREQ  
PWRREQ=H (1) : Output ON in Normal Mode  
PWRREQ=L (0) : Output ON in Low Power Mode  
11: ON in normal mode (Regardless of PWRREQ) |

Source: MAXIM MAX77686 datasheet
The implementation

• **New class types:**
  – PMIC
  – REGULATOR

• **Framework early init function**
  – Search for PMIC and REGULATOR drivers in FDT and bind

• **Framework API**
  – Class PMIC - basic and common
    • Device: [GET] (by name or interface)
    • Register: [READ] [WRITE]
  – Class REGULATOR:
    • Voltage: [GET] [SET] [DESCRIPTOR]
    • Operation MODE: [GET] [SET] [DESCRIPTORS]

• **Single PMIC Device class drivers**
  – PMIC [PARENT] – register I/O operations
  – REGULATOR [CHILD] – specified operations
  – CHARGER/MUIC/ETC.. [CHILDS] – specified operations – not implemented yet
The difference of a design

Present framework

New framework

13/10/2014

Power(full) framework based on Driver Model
The new PMIC Framework – sample design

U-Boot Driver Model

The new PMIC Framework API

COMMON
- GET (by name)
- GET (by bus&addr)
- PROBE
- READ
- WRITE

REGULATOR SPECIFIC
- COUNT
- VOLTAGE DESCRIPTOR
- MODE DESCRIPTORS
- GET VAL
- SET VAL
- GET MODE
- SET MODE

CHARGER SPECIFIC
- ENABLE
- DISABLE
- STATE

MUIC SPECIFIC
- CABLE TYPE
- SWITCH PATH

NOT IMPLEMENTED YET

13/10/2014

Power(full) framework based on Driver Model
## The new PMIC Framework – sample design

<table>
<thead>
<tr>
<th>U-Boot Driver Model</th>
<th>The new PMIC Framework API</th>
</tr>
</thead>
</table>

13/10/2014

Power(full) framework based on Driver Model
The new PMIC Framework – sample design

U-Boot Driver Model

The new PMIC Framework API

UCLASS DEVICE

NAME

PLATFORM DATA

INTERFACE

DEVICE DRIVER

PMIC
PMIC PARENT
RAW I/O

IMPLEMENTED

UCLASS ID
PMIC

OPERATIONS:
NULL

13/10/2014

Power(full) framework based on Driver Model
The new PMIC Framework – sample design

U-Boot Driver Model

The new PMIC Framework API

UCLASS DEVICE

NAME

PLATFORM DATA

INTERFACE

DEVICE DRIVER

IMPLEMENTED

PMIC

PMIC PARENT

RAW I/O

UCLASS ID

PMIC

OPERATIONS:

NULL

13/10/2014

Power(full) framework based on Driver Model
The new PMIC Framework – sample design

U-Boot Driver Model

The new PMIC Framework API

UCLASS DEVICE

IMPLEMENTED

NAME

PLATFORM DATA

INTERFACE

DEVICE DRIVER

UCLASS ID

PMIC

OPERATIONS: NULL

PMIC PARENT

RAW I/O

UCLASS DEVICE

IMPLEMENTED

NAME

SEQ ID

PARENT DEVICE

FDT OFFSET

PLATFORM DATA

INTERFACE

DEVICE DRIVER

UCLASS ID

REGULATOR

UCLASS ID OPERATIONS

DESCRIPTIONS

REG COUNT

PDO VAL

PDO MODE

BUCK VAL

BUCK MODE

REGULATOR

PMIC CHILD

SPECIFIED OPS

Power(full) framework based on Driver Model
The new PMIC Framework – sample design

U-Boot Driver Model

The new PMIC Framework API

Power(full) framework based on Driver Model
The new PMIC Framework – The future plans

- Wait for I2C device class
  - PMIC SPI drivers – bind automatically
  - PMIC I2C drivers – bind manually by FDT „alias”

- Introduce a common and functional framework architecture

- Add new UCLASS types: charger, battery, muic, etc.

- Improve the functionality

- Port the drivers (Help is welcome 😊)

- Remove old the PMIC Framework (half a year)
### Functionality comparison

<table>
<thead>
<tr>
<th>Device</th>
<th>Operation Type</th>
<th>Present Framework</th>
<th>New Framework</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMIC</td>
<td>Register</td>
<td>R/W</td>
<td>Raw I/O</td>
</tr>
<tr>
<td>Regulator</td>
<td>Voltage</td>
<td>SET/GET</td>
<td>Raw I/O</td>
</tr>
<tr>
<td></td>
<td>State</td>
<td>EN/DIS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mode</td>
<td>SET/GET</td>
<td>Common API and Commands</td>
</tr>
<tr>
<td>Muic</td>
<td>Path</td>
<td>SET/GET</td>
<td></td>
</tr>
<tr>
<td>Other</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Battery</td>
<td>State</td>
<td>GET</td>
<td>Common API</td>
</tr>
</tbody>
</table>

Power(full) framework based on Driver Model
The difference for the user

- Magic commands: `pmic „XXX“ write 0x10 0xc3`
- Framework: Read/Write only
- Device Driver
- I/O (SPI/I2C)
- PMIC Hardware

Clear command: `regulator set ldo4 1500mV`
- Framework: Specified class ops
- Class Device
- Device Driver
- I/O (SPI/I2C)
- PMIC Hardware

Which one would you like to use?

13/10/2014
Power(full) framework based on Driver Model
Odroid U3 - example of PMIC usage
Command: „pmic”

pmic list
- list available PMICs
pmic dev <id>
- set id to current pmic device
pmic dump
- dump registers
pmic read <reg>
- read register
pmic write <reg> <value>
- write register

Odroid # pmic list
<table>
<thead>
<tr>
<th>Id</th>
<th>Uclass</th>
<th>Driver</th>
<th>Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>13@pmic</td>
<td>max77686 pmic</td>
<td>I2C0@0x9</td>
</tr>
</tbody>
</table>

Odroid # pmic dev 2
Device: 2 not found
PMIC dev is not set!
Error: No such device

Odroid # pmic dev 9
PMIC: max77686@09

CTRL-A Z for help | 115200 8N1 | NOR | Minicom 2.7 | VT102 | Offline | ttyUSB1
Command: „regulator”

```
Odroid #
Odroid #
Odroid # regulator
regulator - PMIC Regulator

Usage:
regulator list
  - list UCLASS regulator devices
regulator dev [id]
  - show or set operating regulator device
regulator dump
  - dump registers of current regulator
regulator [ldo/buck][N] [name/state/desc]
  - print regulator(s) info
regulator [ldoN/buckN] [setval/setmode] [mV/modeN] [-f]
  - set val (mV) (forcibly) or mode - only if desc available

Example of usage:
First get available commands:
  # regulator
Look after the regulator device 'Id' number:
  # regulator list
Set the operating device:
  # regulator dev 'Id'
List state of device ldo's:
  # regulator ldo state
List descriptors of device ldo's:
  # regulator ldo desc
Set the voltage of ldo number 1 to 1200mV
  # regulator ldo1 setval 1200
Use option: '-f', if given value exceeds the limits (be careful!):
  # regulator ldo1 setval 1200 -f
Set the mode of ldo number 1 to '5' (force not available):
  # regulator ldo1 setmode 5
Odroid #
```
Command: „regulator list/dev”
Command: „regulator name”

```
Odroid # regulator ldo name
RegN: name
LDO01: VDD_ALIVE_1.0V
LDO02: VDDQ_VM1M2_1.2V
LDO03: VCC_1.8V_AP
LDO04: VDDQ_MMC2_2.8V
LDO05: VDDQ_MMC0/1/3_1.8V
LDO06: VMPLL_1.0V
LDO07: VPLL_1.1V
LDO08: VDD_MIPI/HDMI_1.0V
LDO09: nc
LDO10: VDD_MIPI/HDMI_1.8V
LDO11: VDD_ABB1_1.8V
LDO12: VDD_UOTG_3.0V
LDO13: VDD_C2C_1.8V
LDO14: VDD_ABB02_1.8V
LDO15: VDD_Hsic/OTG_1.0V
LDO16: VDD_Hsic_1.8V
LDO17: VDDQ_CAM_1.2V
LDO18: nc
LDO19: nc
LDO20: VDDQ_EMMC_1.8V
LDO21: TFLASH_2.8V
LDO22: VDDQ_EMMC_2.8V
LDO23: nc
LDO24: nc
LDO25: VDDQ_LCD_3.0V
LDO26: nc
Odroid # regulator ldo20 name
RegN: name
LDO20: VDDQ_EMMC_1.8V
```

CTRL-A Z for help | 115200 8N1 | NOR | Minicom 2.7 | VT102 | Offline | ttyUSB1
Command: „regulator setval”

Odroid # regulator ldo26 state
RegN:  val mV  mode
LD026: 3000 mV  @OFF
Odroid # regulator ldo26 desc
RegN:  @descriptors data
LD026:
  @name:  nc
  @Vout:  3000 mV
  @Vmin:  3000 mV
  @Vmax:  3000 mV
  @mode:  0 (OFF) (active)
  @mode:  1 (LPM)
  @mode:  3 (ON/LPM)
  @mode:  4 (ON)
Odroid # regulator ldo26 setval 2900
Value: 2900 mV exceeds descriptor limits:
LD026:
  @name:  nc
  @Vout:  3000 mV
  @Vmin:  3000 mV
  @Vmax:  3000 mV
Error: Operation not permitted
Odroid # regulator ldo26 setval 2900 -f
Set LD026 val: 2900 mV (force)
Command: „regulator setmode”

```
Odroid # regulator ldo26 setmode
Expected positive value!
Error: Invalid argument
Odroid # regulator ldo26 setmode 8
Mode:8 not found in the descriptor:
LD026:
@mode: 0 (OFF) (active)
@mode: 1 (LPM)
@mode: 3 (ON/LPM)
@mode: 4 (ON)
Error: Operation not permitted
Odroid # regulator ldo26 setmode 1
Set LD026 mode: 1
Odroid # regulator ldo26 state
RegN: val mV mode
LD026: 2900 mV 1@LPM
Odroid #
```

CTRL-A Z for help | 115200 8N1 | NOR | Minicom 2.7 | VT102 | Offline | ttyUSB1
The discussion

Share your opinion 😊
Dear listeners

THANK YOU!