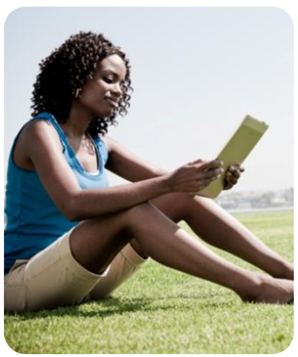
Open *Source* | Open *Possibilities*



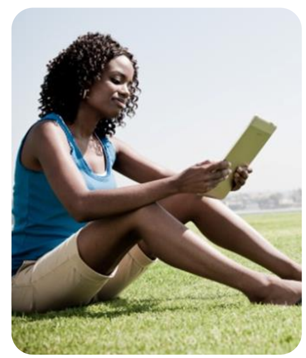
SPMI: System Power Management Interface

Presented by: Josh Cartwright

Presentation Date: 4/30/14

QuIC
QUALCOMM INNOVATION CENTER, INC.

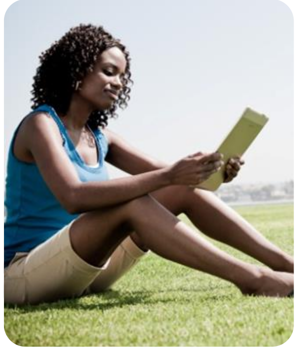
What is SPMI?



Agenda

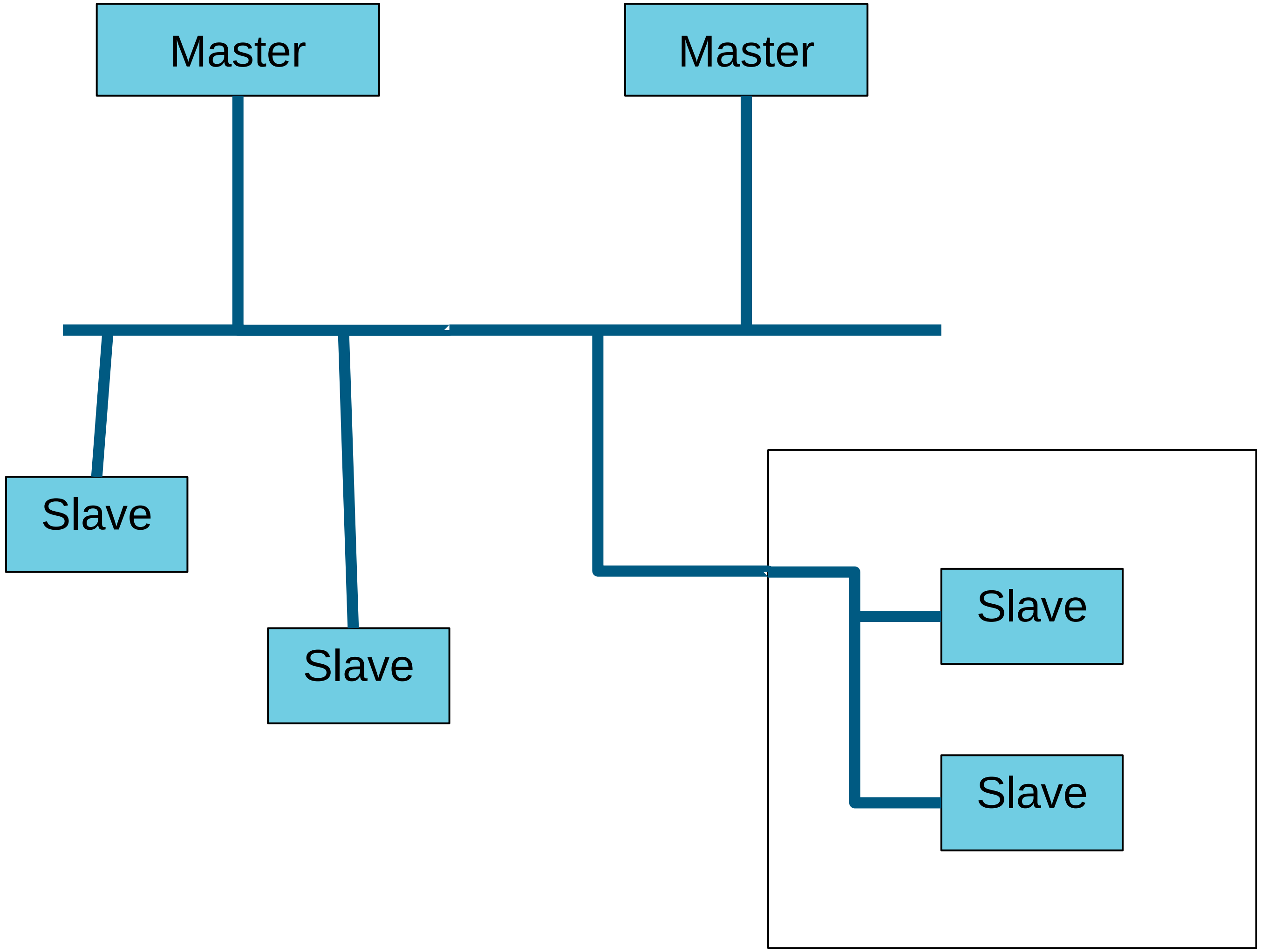
- Architectual Overview
  - Components
  - Addressing
  - Sequences and Arbitration
  - Command set
- Linux Kernel API
- Real World Example

Architecture



Components

- Master
  - At least one master, up to four masters
  - One Master designated Bus Owner Master (BOM)
  - All Masters can initiate Requests
- Slave
  - Up to 16 slaves
  - Slaves can optionally be Request Capable (RCS)


Master
Master
Slave
Slave
Slave
Slave

Addressing

- Master Identifier (MID) – 2-bits
- Unique Slave Identifier (USID) – 4-bits
- Group Slave Identifier (GSID) – 4-bits

Enumeration

- Addressing scheme designed by “System Integrator”

Sequences

- Bus Arbitration
- Start condition
- One or more Frames
  - Command Frame
  - Data Frame
  - No response frame
- Bus Park Cycle

Bus Arbitration

- Responsibility of the current Bus Owner Master (BOM)
- Sequences prioritized in the following levels:
  - Priority Request Capable Slave initiated
  - Priority Master initiated
  - Secondary Request Capable Slave initiated
  - Secondary Master initiated
- Within each level:
  - Slaves are prioritized based on Unique Slave Identifier (USID)
  - Masters are prioritized using round robin scheme
    - Also results in transition of BOM

Command Set

- 17 defined Commands
- State Management
- Master register access
- Slave register access

Slave State Machine

- STARTUP
  - Entered on reset
  - Regulators must be off
- ACTIVE
  - Normal operating state
  - Regulator state user/manufacturer defined
- SLEEP
  - Lower power state
  - Regulator state user/manufacturer defined
- SHUTDOWN
  - Entered via command
  - Regulators must be off

Command Set : State Management

- Reset
  - Puts slave into STARTUP state
- Sleep
  - Puts slave into SLEEP state
- Shutdown
  - Puts slave into SHUTDOWN state
- Wakeup
  - Takes slave out of SLEEP into ACTIVE state

Command Set : Register Access

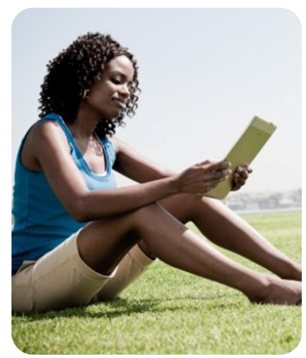
- Register Read/Write
  - 5-bit address, 8-bit data
- Register 0 Write
  - 8-bit data (address assumed 0)
- Extended Register Read/Write
  - 8-bit address, 16 bytes data
- Extended Register Read/Write Long
  - 16-bit address, 8 bytes data

Command Set : Register Access (Master)

- Master Read/Write
  - 8-bit address, 8-bit data

Open *Source* | Open *Possibilities*

Linux Kernel API



QUALCOMM INNOVATION CENTER, INC.

Tree layout

- drivers/spmi/*
  - Contains SPMI "core" (spmi.c)
  - Contains SPMI controller implementations
- Include/linux/spmi.h
  - Contains SPMI data structure definitions/function prototypes
- drivers/base/regmap/regmap-spmi.c
  - Regmap implementation for SPMI devices
- Documentation/devicetree/bindings/spmi/*
  - Generic SPMI device tree binding documentation
  - SPMI controller-specific device tree binding
- (Landed in v3.15 merge window)

Data Structures

- struct spmi_controller;
  - Represents a hardware block capable of acting as a Master on an SPMI bus
- struct spmi_device;
  - Represents an individual unique slave on the SPMI bus
- struct spmi_driver;
  - May be attached to one or more spmi_device objects, implements slave-specific logic

struct spmi_controller

```
struct spmi_controller {
          struct device                    dev;
          unsigned int                     nr;
          int        (*cmd)(struct spmi_controller *ctrl, u8 opcode, u8 sid);
          int        (*read_cmd)(struct spmi_controller *ctrl, u8 opcode,
                                    u8 sid, u16 addr, u8 *buf, size_t len);
          int        (*write_cmd)(struct spmi_controller *ctrl, u8 opcode,
                                     u8 sid, u16 addr, const u8 *buf, size_t len);
};
```

- First two fields are managed by the SPMI core
  - ‘dev’ hooks the controller into the kernels’ device model
  - ‘nr’ is a unique controller number allocated by the core
- Last three members are called by the SPMI core when software wants to issue a Sequence on the bus
- `spmi_controller_get_drvdata()` for controller private data

struct spmi_controller by example

```
static int my_probe(struct parent_bus_type *pdev)
{
        struct spmi_controller *ctrl;
        struct my_data *my_data;
        int err;

        ctrl = spmi_controller_alloc(&pdev->dev, sizeof(*my_data));
        if (!ctrl)
                /* bail */;

        my_data = spmi_controller_get_drvdata(ctrl);
        /* initialize private my_data */

        ctrl->cmd = my_cmd;
        ctrl->read_cmd = my_read_cmd;
        ctrl->write_cmd = my_write_cmd;

        err = spmi_controller_add(ctrl);
        if (err)
                /* bail, but don't forget to spmi_controller_put()! */;
}
```

struct spmi_controller::read_cmd

```
int     (*read_cmd)(struct spmi_controller *ctrl, u8 opcode,
                        u8 sid, u16 addr, u8 *buf, size_t len);
```

- ctrl: driver's controller object
- opcode: one of the following (defined in include/linux/spmi.h)
  - SPMI_CMD_READ
  - SPMI_CMD_READL
  - SPMI_CMD_EXT_READL
- sid: Slave Identififer (SID)
- addr: register address
- buf: buffer to read into
- len: length of buffer

struct spmi_driver

```
struct spmi_driver {
        struct device_driver driver;
        int     (*probe)(struct spmi_device *sdev);
        void    (*remove)(struct spmi_device *sdev);
};
```

- Simple device driver object
- `probe()` is issued when the SPMI core wishes to attach the driver to a slave
- `remove()` is issued when the SPMI device is to be removed

struct spmi_driver by example

```
static const struct of_device_id my_of_table = {
	{ .compatible = “acme,my_device” },
	{ },
};
MODULE_DEVICE_TABLE(of, my_of_table);

static struct spmi_driver my_spmi_driver = {
	.driver    = {
		.name      = “my_spmi_driver”,
		.of_match_table = my_of_table,
	},
	.probe     = my_spmi_probe,
	.remove    = my_spmi_remove,
};
module_spmi_driver(my_spmi_driver);
```

struct spmi_device

```
struct spmi_device {
        struct device dev;
        struct spmi_controller *ctrl;
        u8 usid;
};
```

- spmi_device objects managed by the SPMI core

struct spmi_device API

```
int spmi_register_read(struct spmi_device *sdev, u8 addr, u8 *buf);
int spmi_ext_register_read(struct spmi_device *sdev, u8 addr, u8 *buf,
                                    size_t len);
int spmi_ext_register_readl(struct spmi_device *sdev, u16 addr, u8 *buf,
                                    size_t len);
int spmi_register_write(struct spmi_device *sdev, u8 addr, u8 data);
int spmi_register_zero_write(struct spmi_device *sdev, u8 data);
int spmi_ext_register_write(struct spmi_device *sdev, u8 addr,
                                    const u8 *buf, size_t len);
int spmi_ext_register_writel(struct spmi_device *sdev, u16 addr,
                                    const u8 *buf, size_t len);
int spmi_command_reset(struct spmi_device *sdev);
int spmi_command_sleep(struct spmi_device *sdev);
int spmi_command_wakeup(struct spmi_device *sdev);
int spmi_command_shutdown(struct spmi_device *sdev);
```

Device Tree Bindings

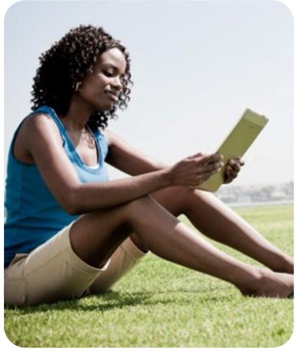
```
spmi@.. {
        compatible = "...";
        reg = <...>;

        #address-cells = <2>;
        #size-cells <0>;

        child@0 {
                compatible = "...";
                reg = <0 SPMI_USID>;
        };

        child@7 {
                compatible = "...";
                reg = <7 SPMI_USID
                       3 SPMI_GSID>;
        };
};
```

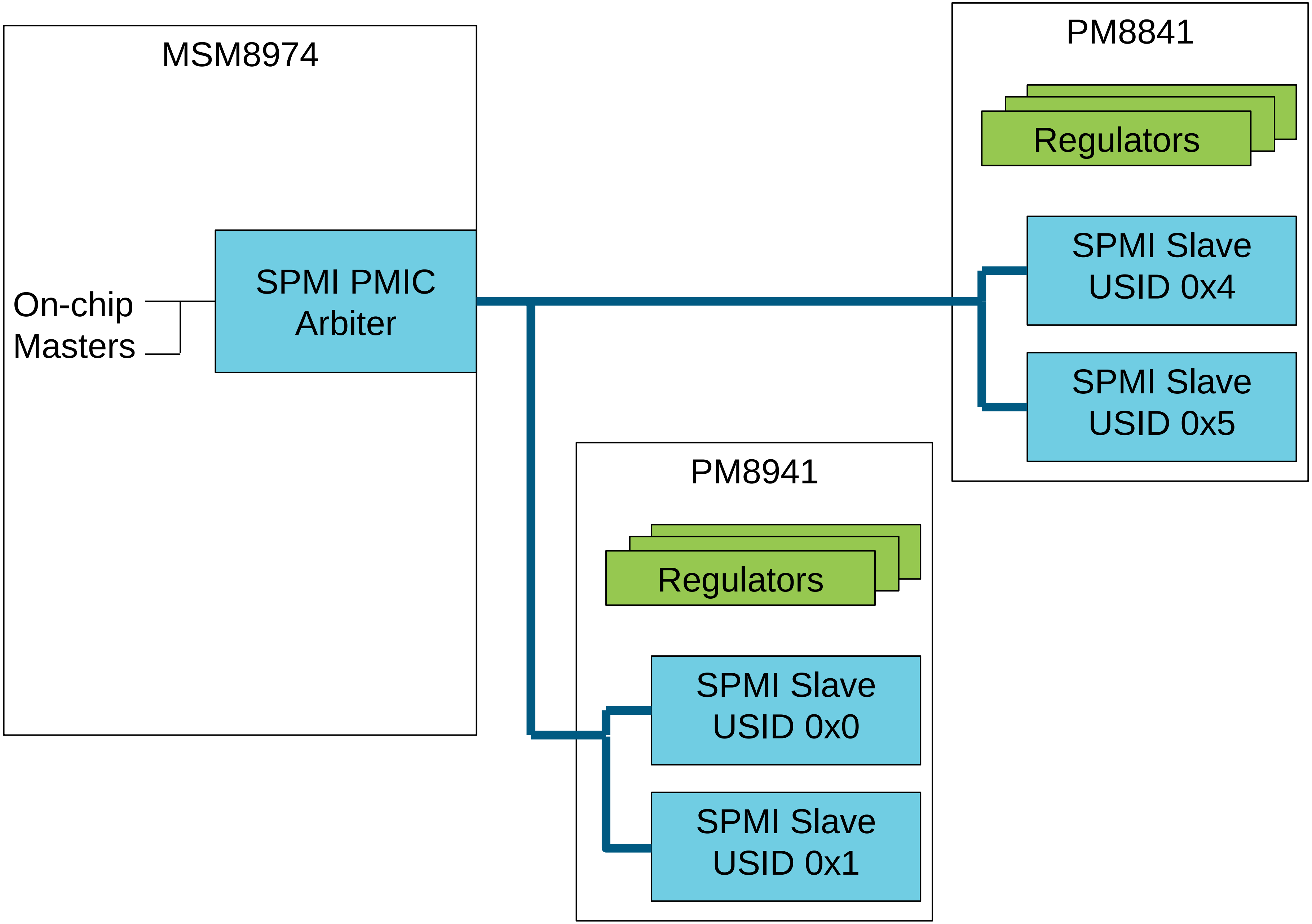
Real World Implementation



QuIC
QUALCOMM INNOVATION CENTER, INC.

SPMI in the wild

- MSM8974
  - Member of Qualcomm Snapdragon 800 Series SoCs
  - Quad-core Krait, Adreno 330 GPU, …
- PM8841 & PM8941
  - Pair of PMICs housing regulators used to power the SoC and peripherals
  - Also responsible for battery management/charging
  - Various misc. functionality, too (GPIOs, RTC, …)
- Communication between SoC and PMIC implemented over SPMI


MSM8974
On-chip
Masters
SPMI PMIC
Arbiter
PM8841
Regulators
SPMI Slave
USID 0x4
SPMI Slave
USID 0x5
PM8941
Regulators
SPMI Slave
USID 0x0
SPMI Slave
USID 0x1

Questions?