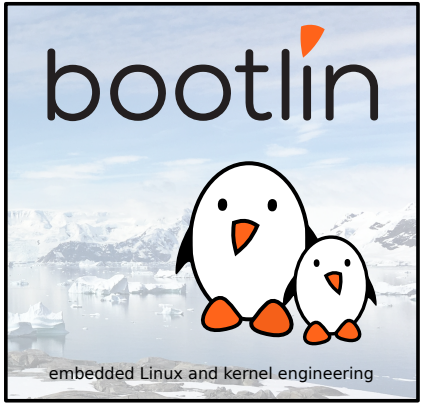
Introduction to DAPM: Linux power management for embedded audio devices

Luca Ceresoli
luca.ceresoli@bootlin.com



© Copyright 2004-2024, Bootlin.
Creative Commons BY-SA 3.0 license.
Corrections, suggestions, contributions and translations are welcome!

Luca Ceresoli

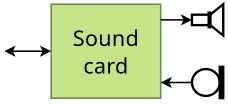
- Embedded Linux engineer at Bootlin
  - Embedded Linux **expertise**
  - **Development**, consulting and training
  - Strong open-source focus
- Linux kernel device driver developer
- Bootloaders, Buildroot and Yocto integration
- Open-source contributor
- Living in **Bergamo**, Italy
- luca.ceresoli@bootlin.com

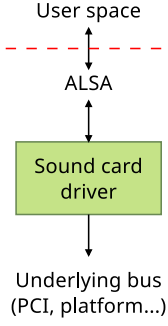
https://bootlin.com/company/staff/luca-ceresoli/

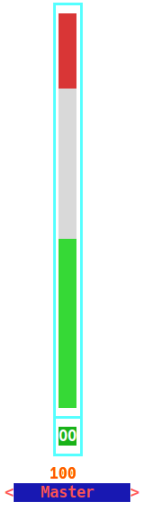
Background (ALSA, ASoC, DAPM)

ALSA = Advanced Linux Sound Architecture

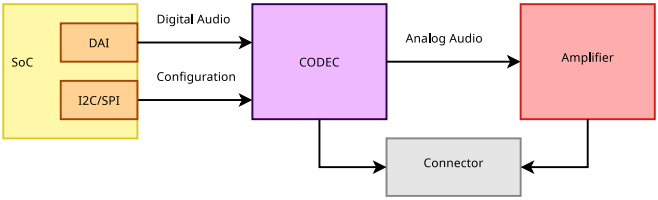
- Merged in v2.5, 2002
- 1 sound card = 1 device = 1 driver
- Consistent user space API based on
  - Streams: capture, playback
  - kcontrols to change settings
- User space API still in use today
- Hard to reuse code for components used on different cards







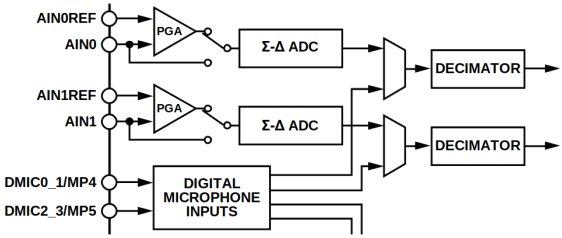
ASoC = ALSA System on Chip



- Merged in 2006
- Additional ALSA layer “to provide better ALSA support for embedded System-on-Chip processors” (https://docs.kernel.org/sound/soc/overview.html)
  - Great for embedded systems where different SoCs, codecs and other components are mixed and matched
- 1 sound card = N components and their interconnections + glue
- Each component has a separate driver
- Same user space API

Power management with ASoC

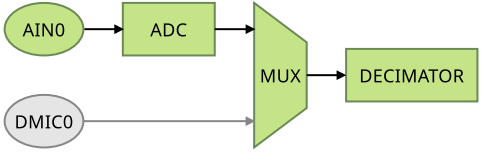
- Many components allow flexible routing
- Different routings require different components to be turned on
- Many combinations: code to enable only what is needed tends to be complex and not easy to maintain

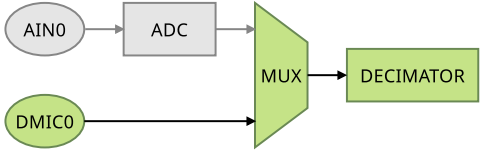


(https://www.analog.com/media/en/technical-documentation/data-sheets/ADAU1372.pdf)

DAPM: Dynamic Audio Power Management

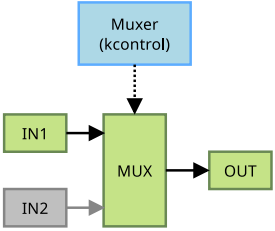
- The power management component of ASoC
- LinuxRuntime PM works at the device level, → not suitable
- DAPM is independent from kernel Runtime PM, and co-existing
- Transparent to user space applications
- Describes every power-related element as a node of a graph
- Every power control is called a DAPM **widget** (graph node)
- Every connection between widgets is called a DAPM **route** (directed graph edge)
- DAPM automatically enables widgets based on active routes





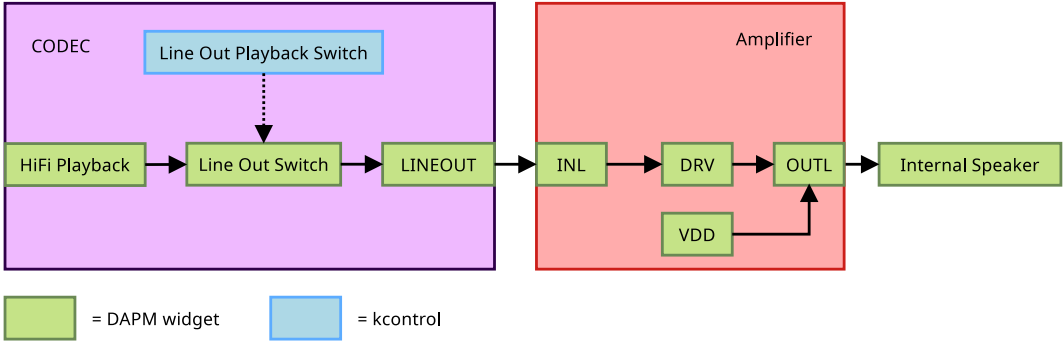
DAPM: Dynamic Audio Power Management

- DAPM widgets can be controlled by a regular kcontrol



DAPM: Dynamic Audio Power Management

- The DAPM tree spans the whole card
  - In-component widgets and routes are implemented by the component driver
  - Border widgets and cross-component routes are added by the card



Documentation

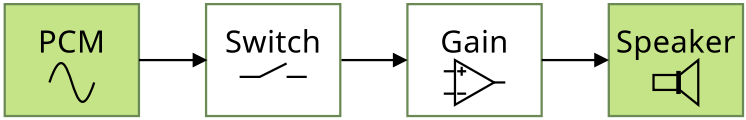
- Documented in the official kernel docs
- https://docs.kernel.org/sound/soc/dapm.html
- Proposed improvement: https://lore.kernel.org/all/20240416-dapm-docs-v1-0-a818d2819bf6@bootlin.com/

References

- Audio on Linux: The End of a Golden Age?
  Lars-Peter Clausen, ELCE 2016
  Slides: https://elinux.org/images/e/e7/Audio_on_Linux.pdf
  Video: https://www.youtube.com/watch?v=6oQF2TzCYtQ
- Making the Most of Dynamic Audio Power Management
  Lars-Peter Clausen, ELCE 2015
  Slides: https://elinux.org/images/c/c1/Dapm_clausen.pdf
  Video not available

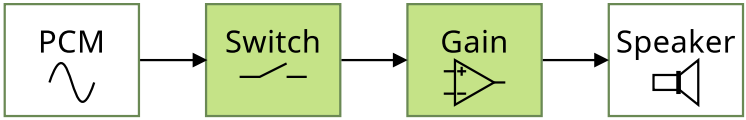
DAPM widgets

Endpoint widgets



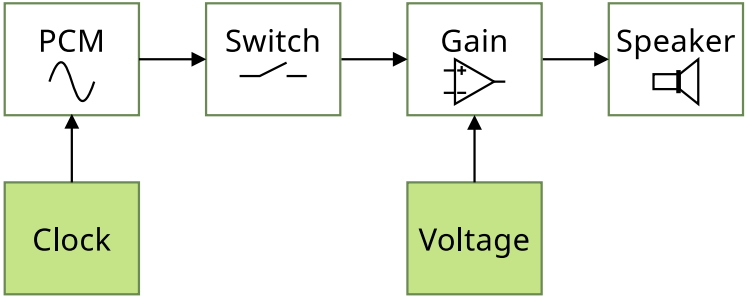
- Widgets where the sound stream originates from or terminates at
- ADC, DAC (PCM waveform to/from memory)
- Speaker, Line out, Microphone, ...

Pass-through widgets



- Widgets on a route between other widgets
- Sound modifiers (PGA, Effect)
- Routing: Mixer, Mux, Demux, Switch

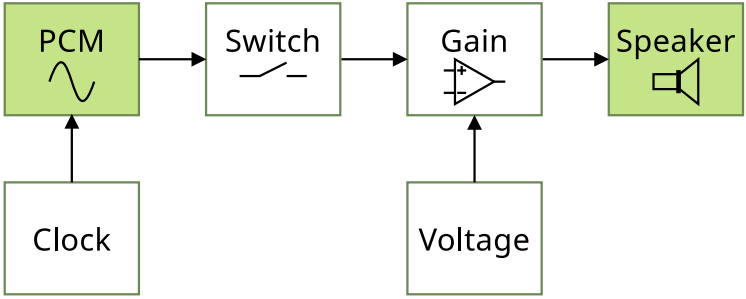
Supply widgets



- Suppliers to other widgets
- Clock, current, voltage

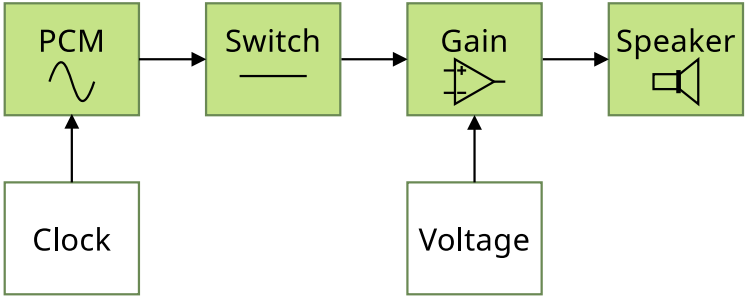
DAPM in action

Phase 1: determining power state



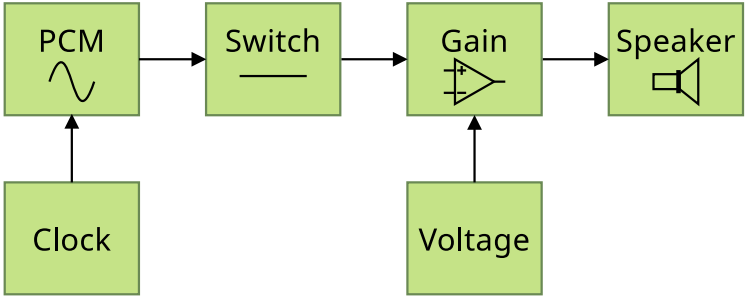
- Source widgets are powered if they are active (used by a stream) and have a route to an active sink widget
- Sink widgets are powered if they are active (used by a stream) and have a route to an active source widget

Phase 1: determining power state



- Pass-through widgets are powered if they are on the route between two powered endpoint widgets
- Computed by DAPM automatically

Phase 1: determining power state



- Supply widgets are powered if they have a path to a powered widget
- Computed by DAPM automatically

Phase 2: Powering sequence

1. Compute difference between previous and new configurations
2. Power down newly-disabled widgets
3. Apply routing changes
4. Power up newly-enabled widgets

Using DAPM in device drivers

Defining DAPM widgets

- A widget is defined by `struct snd_soc_dapm_widget`

include/sound/soc-dapm.h

```
struct snd_soc_dapm_widget {
    enum snd_soc_dapm_type id;
    const char *name;                           /* widget name */
    const char *sname;                          /* stream name */
    struct snd_soc_dapm_context *dapm;
    /* ... */
    struct pinctrl *pinctrl;                    /* attached pinctrl */
    /* ... */
    int reg;                                    /* negative reg = no direct dapm */
    unsigned char shift;                        /* bits to shift */
    unsigned int mask;                          /* non-shifted mask */
    unsigned int on_val;                        /* on state value */
    unsigned int off_val;                       /* off state value */
    /* ... */
    unsigned short event_flags;                 /* flags to specify event types */
    int (*event)(struct snd_soc_dapm_widget*, struct snd_kcontrol *, int);
};
```

Defining DAPM widgets

- Do **not** use `struct snd_soc_dapm_widget` directly!
- Use the `SND_SOC_DAPM_*()` macros defined in `include/sound/soc-dapm.h`
- Each macro fills a `struct snd_soc_dapm_widget`

sound/soc/codecs/adau1372.c

```
static const struct snd_soc_dapm_widget adau1372_dapm_widgets[] = {
    SND_SOC_DAPM_INPUT("AIN0"), /* An input pin */
    SND_SOC_DAPM_SUPPLY("MICBIAS0", ADAU1372_REG_MICBIAS, 4, 0, NULL, 0),
    SND_SOC_DAPM_PGA("PGA0", ADAU1372_REG_PGA_CTRL(0), 6, 1, NULL, 0),
    SND_SOC_DAPM_ADC("ADC0", NULL, ADAU1372_REG_ADC_CTRL2, 0, 0),
    SND_SOC_DAPM_SUPPLY("ADC0 Filter", ADAU1372_REG_DECIM_PWR, 0, 0, NULL, 0),
    SND_SOC_DAPM_SUPPLY("Output ASRC0 Decimator", ADAU1372_REG_DECIM_PWR, 4, 0, NULL, 0),
    SND_SOC_DAPM_MUX("Decimator0 Mux", SND_SOC_NOPM, 0, 0, &adau1372_decimator0_1_mux_control),
    SND_SOC_DAPM_OUTPUT("HPOUTL"), /* An output pin */
    /* ... */
};
```

regmap

- Widget macros take register offset but no base
- Each widget can set the correct register thanks `regmap`
- Regmap abstracts register access from the underlying bus
  - Originated from ASoC, now a generic kernel feature
  - Allows a single driver for dual (I^2C/SPI) CODECs
  - Optimizes access via register cache
  - and much more
- `regmap` is recommended for register access in ASoC
  - In ASoC, a `regmap` is automatically added
  - By `snd_soc_add_component()`
  - In `widget->component->regmap`

Defining DAPM routes

- A route is defined by `struct snd_soc_dapm_route`

include/sound/soc-dapm.h

```
struct snd_soc_dapm_route {
    const char *sink;
    const char *control;
    const char *source;
    /* ... */
};
```

Defining DAPM routes

```
static const char * const adau1372_decimator_mux_text[] = { "ADC", "DMIC", };
static SOC_ENUM_SINGLE_DECL(adau1372_decimator0_1_mux_enum, ADAU1372_REG_ADC_CTRL2,
                            2, adau1372_decimator_mux_text);

static const struct snd_soc_dapm_route adau1372_dapm_routes[] = {
    { "PGA0",           NULL,   "AIN0"        },
    { "ADC0",           NULL,   "PGA0"        },
    { "Decimator0 Mux", "ADC",  "ADC0"        },
    { "Decimator0 Mux", "DMIC", "DMIC0_1"     },
    { "HPOUTL",         NULL,   "OP_STAGE_LP" },
    { "HPOUTL",         NULL,   "OP_STAGE_LN" },
    /* ... */
};
```

- Control is a standard ALSA kcontrol for selection of mux input, demux output, mixer levels, PGA gain, ...
- Matching based on strings

Adding DAPM widgets and routes

- Just point to the defined arrays in `struct snd_soc_component_driver`

```
static const struct snd_soc_component_driver adau1372_driver = {
    .set_bias_level = adau1372_set_bias_level,
    .controls = adau1372_controls,
    .num_controls = ARRAY_SIZE(adau1372_controls),
    .dapm_widgets = adau1372_dapm_widgets,
    .num_dapm_widgets = ARRAY_SIZE(adau1372_dapm_widgets),
    .dapm_routes = adau1372_dapm_routes,
    .num_dapm_routes = ARRAY_SIZE(adau1372_dapm_routes),
};
```

Adding DAPM widgets and routes dynamically

- DAPM routes can be added dynamically, e.g. based on codec model

```
static const struct snd_soc_dapm_widget wm8994_dapm_widgets[] = { ... };
static const struct snd_soc_dapm_route wm8994_intercon[] = { ... };

static int wm8994_component_probe(struct snd_soc_component *component)
{
    /* ... */
    switch (control->type) {
    case WM8994:
        snd_soc_dapm_new_controls(dapm, wm8994_specific_dapm_widgets,
                                  ARRAY_SIZE(wm8994_specific_dapm_widgets));
        /* ... */
        snd_soc_dapm_add_routes(dapm, wm8994_intercon,
                                ARRAY_SIZE(wm8994_intercon));
        /* ... */
    }
}
```

Connecting widgets to the DAI stream

- The endpoints of the DAPM graph are
  - The external input/output pins
    - `SND_SOC_DAPM_INPUT()`, `SND_SOC_DAPM_OUTPUT()`
  - The DAI (digital audio interface)
    - Via the stream name defined by the DAI driver, using (sub)string-based matching

```
static const struct snd_soc_dapm_widget wm9705_dapm_widgets[] = {
    SND_SOC_DAPM_DAC("Left DAC",  "Left HiFi Playback",  SND_SOC_NOPM, 0, 0),
    SND_SOC_DAPM_DAC("Right DAC", "Right HiFi Playback", SND_SOC_NOPM, 0, 0),

...

static struct snd_soc_dai_driver wm9705_dai[] = {
    {
        .name = "wm9705-hifi",
        .playback = {
            .stream_name = "HiFi Playback",
...
```

Inspecting the DAPM state at runtime

Inspecting DAPM

- DAPM is internal, not exposed to user space
- It is managed automatically
- It just works, no need to inspect it! `:-)`
- OK, so you want to learn? Debug? Well...

debugfs

- Each widget is exposed as a file in debugfs
- `/sys/kernel/debug/asoc/${CARD}/${COMPONENT}/dapm/${WIDGET}`
- `/sys/kernel/debug/asoc/${CARD}/dapm/${WIDGET}` for card-level widgets

```
# cat "/sys/kernel/debug/asoc/STM32MP15-DK/cs42l51.0-004a/dapm/Left ADC"
Left ADC: Off  in 1 out 0 - R2(0x2) mask 0x2
 stream Left HiFi Capture inactive
 out  "static" "Capture"
 in  "static" "Left PGA"
#
```

- Widget name can be ambiguous (same widget name in different components)

debugfs: proposed improvement

- Proposed improvements: https://lore.kernel.org/all/20240416-vizdapm-ng-v1-0-5d33c0b57bc5@bootlin.com
- Add widget type
- Add component name for in/out routes

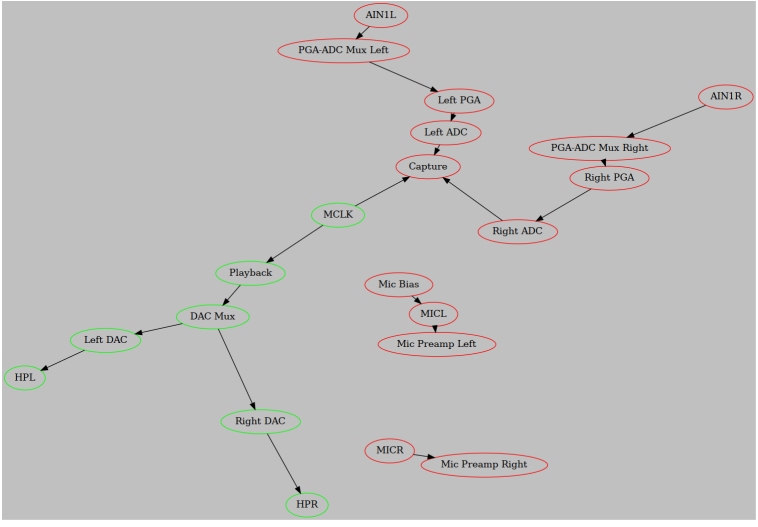
```
# cat "/sys/kernel/debug/asoc/STM32MP15-DK/cs42l51.0-004a/dapm/Left ADC"
Left ADC: Off  in 1 out 0 - R2(0x2) mask 0x2
 stream Left HiFi Capture inactive
 widget-type adc
 out  "static" "Capture" "cs42l51.0-004a"
 in  "static" "Left PGA" "cs42l51.0-004a"
#
```

vizdapm

- Simple shell script developed by Dimitris Papastamos, Wolfson Micro
- Generates a graph of DAPM widgets and routes as a PNG picture
- Based on `dot` from graphviz
- Repository disappeared, still available in some git forks

```
# vizdapm /sys/kernel/debug/asoc/STM32MP15-DK/cs42l51.0-004a/dapm out.png
```

vizdapm


AIN1L
PGA-ADC Mux Left
Left PGA
AIN1R
Left ADC
PGA-ADC Mux Right
Capture
Right PGA
MCLK
Right ADC
Playback
Mic Bias
DAC Mux
MICL
Left DAC
Mic Preamp Left
HPL
Right DAC
MICR
Mic Preamp Right
HPR

dapm-graph: new proposed tool

- Inspired by vizdapm and also based on `dot` from graphviz
- Yet another shell script but more powerful and simpler to use
- Shows all components and their connections
- Works with BuyBox shell
- Basic usage:
  - `dapm-graph -o dapm.svg -c STM32MP15-DK`
- Remote mode:
  - `dapm-graph -o dapm.svg -c STM32MP15-DK -r root@192.168.0.1`
  - Gets the status from target, processes on the host
- And more
- Proposed for kernel inclusion in the same series:

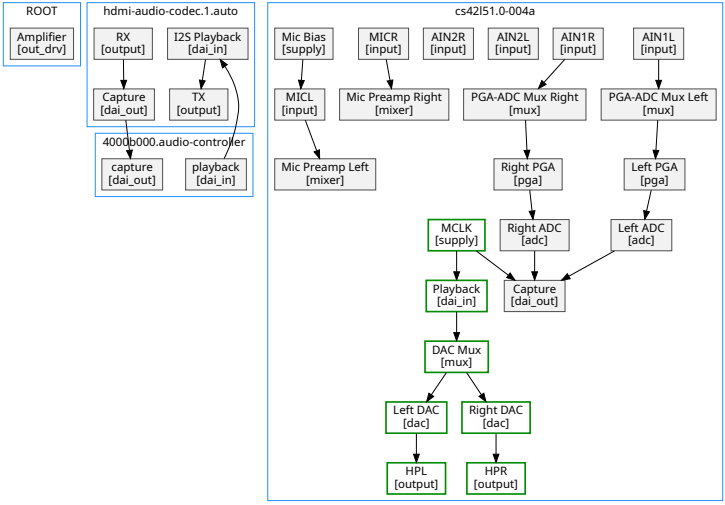
  https://lore.kernel.org/all/20240416-vizdapm-ng-v1-0-5d33c0b57bc5@bootlin.com

dapm-graph: new proposed tool

```
Usage:
    dapm-graph [options] -c CARD                  - Local sound card
    dapm-graph [options] -c CARD -r REMOTE_TARGET - Card on remote system
    dapm-graph [options] -d STATE_DIR             - Local directory

Options:
    -c CARD             Sound card to get DAPM state of
    -r REMOTE_TARGET    Get DAPM state from REMOTE_TARGET via SSH and SCP
                        instead of using a local sound card
    -d STATE_DIR        Get DAPM state from a local copy of a debugfs tree
    -o OUT_FILE         Output file (default: dapm.dot)
    -D                  Show verbose debugging info
    -h                  Print this help and exit
```

dapm-graph


ROOT
Amplifier [out_drv]
hdmi-audio-codec.1.auto
RX [output]
I2S Playback [dai_in]
Capture [dai_out]
TX [output]
4000b000.audio-controller
capture [dai_out]
playback [dai_in]
cs42l51.0-004a
Mic Bias [supply]
MICR [input]
AIN2R [input]
AIN2L [input]
AIN1R [input]
AIN1L [input]
MICL [input]
Mic Preamp Right [mixer]
PGA-ADC Mux Right [mux]
PGA-ADC Mux Left [mux]
Mic Preamp Left [mixer]
Right PGA [pga]
Left PGA [pga]
MCLK [supply]
Right ADC [adc]
Left ADC [adc]
Playback [dai_in]
Capture [dai_out]
DAC Mux [mux]
Left DAC [dac]
Right DAC [dac]
HPL [output]
HPR [output]

Inspecting the internals with ftrace

- Using ftrace

```
# trace-cmd record -e 'snd_soc_*' -l 'rk3308_codec*' -p function_graph play ...
# trace-cmd report
```

- `rk3308_codec_hw_params()`
- `dapm_power_widgets()`
  - `snd_soc_dapm_start: card=rk3308card event=1`
    - `snd_soc_dapm_widget_power` fills list of widgets to be powered on/off
    - `snd_soc_dapm_path` propagates state through routes
    - `snd_soc_dapm_walk_done: rk3308card: checks 34 power, 26 path, 36 neighbour`
    - `snd_soc_bias_level_start/done` for each component (in separate kthreads) and `snd_soc_dapm_widget_event_start/done` for widgets having events
  - `snd_soc_dapm_done: card=rk3308card event=1`
- Audio stream...
- Rollback

Questions? Suggestions? Comments?

Luca Ceresoli
luca.ceresoli@bootlin.com

Slides under CC-BY-SA 3.0
https://bootlin.com/pub/conferences/