Dynamic Audio Power Management for Portable Devices

Description

Dynamic Audio Power Management (DAPM) is designed to allow portable Linux devices to use the minimum amount of power within the audio subsystem at all times. It is independent of other kernel PM and as such, can easily co-exist with the other PM systems.

DAPM is also completely transparent to all user space applications as all power switching is done within the ASoC core. No code changes or recompiling are required for user space applications. DAPM makes power switching decisions based upon any audio stream (capture/playback) activity and audio mixer settings within the device.

DAPM spans the whole machine. It covers power control within the entire audio subsystem, this includes internal codec power blocks and machine level power systems.

There are 4 power domains within DAPM

Codec bias domain

VREF, VMID (core codec and audio power)

Usually controlled at codec probe/remove and suspend/resume, although can be set at stream time if power is not needed for sidetone, etc.

Platform/Machine domain

physically connected inputs and outputs

Is platform/machine and user action specific, is configured by the machine driver and responds to asynchronous events e.g when HP are inserted

Path domain

audio subsystem signal paths

Automatically set when mixer and mux settings are changed by the user. e.g. alsamixer, amixer.

Stream domain

DACs and ADCs.

Enabled and disabled when stream playback/capture is started and stopped respectively. e.g. aplay, arecord.

All DAPM power switching decisions are made automatically by consulting an audio routing map of the whole machine. This map is specific to each machine and consists of the interconnections between every audio component (including internal codec components). All audio components that effect power are called widgets hereafter.

DAPM Widgets

Audio DAPM widgets fall into a number of types:-

Mixer
Mixes several analog signals into a single analog signal.

Mux
An analog switch that outputs only one of many inputs.

PGA
A programmable gain amplifier or attenuation widget.

ADC
Analog to Digital Converter

DAC
Digital to Analog Converter

Switch
An analog switch

Input
A codec input pin

Output
A codec output pin

Headphone
Headphone (and optional Jack)

Mic
Mic (and optional Jack)

Line
Line Input/Output (and optional Jack)

Speaker
Speaker

Supply
Power or clock supply widget used by other widgets.

Regulator
External regulator that supplies power to audio components.

Clock
External clock that supplies clock to audio components.

AIF IN
Audio Interface Input (with TDM slot mask).

AIF OUT
Audio Interface Output (with TDM slot mask).

Siggen
Signal Generator.

DAI IN
Digital Audio Interface Input.

DAI OUT
Digital Audio Interface Output.

DAI Link
DAI Link between two DAI structures

Pre
Special PRE widget (exec before all others)

Post
Special POST widget (exec after all others)

Buffer
Inter widget audio data buffer within a DSP.

Scheduler
DSP internal scheduler that schedules component/pipeline processing work.

Effect
Widget that performs an audio processing effect.

SRC
Sample Rate Converter within DSP or CODEC

ASRC
Asynchronous Sample Rate Converter within DSP or CODEC

Encoder
Widget that encodes audio data from one format (usually PCM) to another usually more compressed format.

Decoder
Widget that decodes audio data from a compressed format to an uncompressed format like PCM.

(Widgets are defined in include/sound/soc-dapm.h)

Widgets can be added to the sound card by any of the component driver types. There are convenience macros defined in soc-dapm.h that can be used to quickly build a list of widgets of the codecs and machines DAPM widgets.

Most widgets have a name, register, shift and invert. Some widgets have extra parameters for stream name and kcontrols.

Stream Domain Widgets

Stream Widgets relate to the stream power domain and only consist of ADCs (analog to digital converters), DACs (digital to analog converters), AIF IN and AIF OUT.

Stream widgets have the following format:-

```
SND_SOC_DAPM_DAC(name, stream name, reg, shift, invert),
SND_SOC_DAPM_AIF_IN(name, stream, slot, reg, shift, invert)
```

NOTE: the stream name must match the corresponding stream name in your codec snd_soc_codec_dai.

e.g. stream widgets for HiFi playback and capture

```
SND_SOC_DAPM_DAC("HiFi DAC", "HiFi Playback", REG, 3, 1),
SND_SOC_DAPM_ADC("HiFi ADC", "HiFi Capture", REG, 2, 1),
```

e.g. stream widgets for AIF

```
SND_SOC_DAPM_AIF_IN("AIF1RX", "AIF1 Playback", 0, SND_SOC_NOPM, 0, 0),
SND_SOC_DAPM_AIF_OUT("AIF1TX", "AIF1 Capture", 0, SND_SOC_NOPM, 0, 0),
```

Path Domain Widgets

Path domain widgets have a ability to control or affect the audio signal or audio paths within the audio subsystem. They have the following form:-

```
SND_SOC_DAPM_PGA(name, reg, shift, invert, controls, num_controls)
```

Any widget kcontrols can be set using the controls and num_controls members.

e.g. Mixer widget (the kcontrols are declared first)

```
/* Output Mixer */
static const snd_kcontrol_new_t wm8731_output_mixer_controls[] = {
SOC_DAPM_SINGLE("Line Bypass Switch", WM8731_APANA, 3, 1, 0),
SOC_DAPM_SINGLE("Mic Sidetone Switch", WM8731_APANA, 5, 1, 0),
SOC_DAPM_SINGLE("HiFi Playback Switch", WM8731_APANA, 4, 1, 0),
};

SND_SOC_DAPM_MIXER("Output Mixer", WM8731_PWR, 4, 1, wm8731_output_mixer_controls,
      ARRAY_SIZE(wm8731_output_mixer_controls)),
```

If you don't want the mixer elements prefixed with the name of the mixer widget, you can use SND_SOC_DAPM_MIXER_NAMED_CTL instead. the parameters are the same as for SND_SOC_DAPM_MIXER.

Machine domain Widgets

Machine widgets are different from codec widgets in that they don't have a codec register bit associated with them. A machine widget is assigned to each machine audio component (non codec or DSP) that can be independently powered. e.g.

- Speaker Amp
- Microphone Bias
- Jack connectors

A machine widget can have an optional call back.

e.g. Jack connector widget for an external Mic that enables Mic Bias when the Mic is inserted:-:

```
static int spitz_mic_bias(struct snd_soc_dapm_widget* w, int event)
{
      gpio_set_value(SPITZ_GPIO_MIC_BIAS, SND_SOC_DAPM_EVENT_ON(event));
      return 0;
}

SND_SOC_DAPM_MIC("Mic Jack", spitz_mic_bias),
```

Codec (BIAS) Domain

The codec bias power domain has no widgets and is handled by the codecs DAPM event handler. This handler is called when the codec powerstate is changed wrt to any stream event or by kernel PM events.

Virtual Widgets

Sometimes widgets exist in the codec or machine audio map that don't have any corresponding soft power control. In this case it is necessary to create a virtual widget - a widget with no control bits e.g.

```
SND_SOC_DAPM_MIXER("AC97 Mixer", SND_SOC_DAPM_NOPM, 0, 0, NULL, 0),
```

This can be used to merge to signal paths together in software.

After all the widgets have been defined, they can then be added to the DAPM subsystem individually with a call to snd_soc_dapm_new_control().

Codec/DSP Widget Interconnections

Widgets are connected to each other within the codec, platform and machine by audio paths (called interconnections). Each interconnection must be defined in order to create a map of all audio paths between widgets.

This is easiest with a diagram of the codec or DSP (and schematic of the machine audio system), as it requires joining widgets together via their audio signal paths.

e.g., from the WM8731 output mixer (wm8731.c)

The WM8731 output mixer has 3 inputs (sources)

1. Line Bypass Input
2. DAC (HiFi playback)
3. Mic Sidetone Input

Each input in this example has a kcontrol associated with it (defined in example above) and is connected to the output mixer via its kcontrol name. We can now connect the destination widget (wrt audio signal) with its source widgets.

```
/* output mixer */
{"Output Mixer", "Line Bypass Switch", "Line Input"},
{"Output Mixer", "HiFi Playback Switch", "DAC"},
{"Output Mixer", "Mic Sidetone Switch", "Mic Bias"},
```

So we have :-

- Destination Widget <=== Path Name <=== Source Widget, or
- Sink, Path, Source, or
- `Output Mixer` is connected to the `DAC` via the `HiFi Playback Switch`.

When there is no path name connecting widgets (e.g. a direct connection) we pass NULL for the path name.

Interconnections are created with a call to:-

```
snd_soc_dapm_connect_input(codec, sink, path, source);
```

Finally, snd_soc_dapm_new_widgets(codec) must be called after all widgets and interconnections have been registered with the core. This causes the core to scan the codec and machine so that the internal DAPM state matches the physical state of the machine.

Machine Widget Interconnections

Machine widget interconnections are created in the same way as codec ones and directly connect the codec pins to machine level widgets.

e.g. connects the speaker out codec pins to the internal speaker.

```
/* ext speaker connected to codec pins LOUT2, ROUT2  */
{"Ext Spk", NULL , "ROUT2"},
{"Ext Spk", NULL , "LOUT2"},
```

This allows the DAPM to power on and off pins that are connected (and in use) and pins that are NC respectively.

Endpoint Widgets

An endpoint is a start or end point (widget) of an audio signal within the machine and includes the codec. e.g.

- Headphone Jack
- Internal Speaker
- Internal Mic
- Mic Jack
- Codec Pins

Endpoints are added to the DAPM graph so that their usage can be determined in order to save power. e.g. NC codecs pins will be switched OFF, unconnected jacks can also be switched OFF.

DAPM Widget Events

Some widgets can register their interest with the DAPM core in PM events. e.g. A Speaker with an amplifier registers a widget so the amplifier can be powered only when the spk is in use.

```
/* turn speaker amplifier on/off depending on use */
static int corgi_amp_event(struct snd_soc_dapm_widget *w, int event)
{
      gpio_set_value(CORGI_GPIO_APM_ON, SND_SOC_DAPM_EVENT_ON(event));
      return 0;
}

/* corgi machine dapm widgets */
static const struct snd_soc_dapm_widget wm8731_dapm_widgets =
      SND_SOC_DAPM_SPK("Ext Spk", corgi_amp_event);
```

Please see soc-dapm.h for all other widgets that support events.

Event types

The following event types are supported by event widgets.

```
/* dapm event types */
#define SND_SOC_DAPM_PRE_PMU  0x1     /* before widget power up */
#define SND_SOC_DAPM_POST_PMU 0x2             /* after widget power up */
#define SND_SOC_DAPM_PRE_PMD  0x4     /* before widget power down */
#define SND_SOC_DAPM_POST_PMD 0x8             /* after widget power down */
#define SND_SOC_DAPM_PRE_REG  0x10    /* before audio path setup */
#define SND_SOC_DAPM_POST_REG 0x20    /* after audio path setup */
```