Dynamic PCM

Description

Dynamic PCM allows an ALSA PCM device to digitally route its PCM audio to various digital endpoints during the PCM stream runtime. e.g. PCM0 can route digital audio to I2S DAI0, I2S DAI1 or PDM DAI2. This is useful for on SoC DSP drivers that expose several ALSA PCMs and can route to multiple DAIs.

The DPCM runtime routing is determined by the ALSA mixer settings in the same way as the analog signal is routed in an ASoC codec driver. DPCM uses a DAPM graph representing the DSP internal audio paths and uses the mixer settings to determine the path used by each ALSA PCM.

DPCM re-uses all the existing component codec, platform and DAI drivers without any modifications.

Phone Audio System with SoC based DSP

Consider the following phone audio subsystem. This will be used in this document for all examples :-

```
| Front End PCMs    |  SoC DSP  | Back End DAIs | Audio devices |

                    *************
PCM0 <------------> *           * <----DAI0-----> Codec Headset
                    *           *
PCM1 <------------> *           * <----DAI1-----> Codec Speakers
                    *   DSP     *
PCM2 <------------> *           * <----DAI2-----> MODEM
                    *           *
PCM3 <------------> *           * <----DAI3-----> BT
                    *           *
                    *           * <----DAI4-----> DMIC
                    *           *
                    *           * <----DAI5-----> FM
                    *************
```

This diagram shows a simple smart phone audio subsystem. It supports Bluetooth, FM digital radio, Speakers, Headset Jack, digital microphones and cellular modem. This sound card exposes 4 DSP front end (FE) ALSA PCM devices and supports 6 back end (BE) DAIs. Each FE PCM can digitally route audio data to any of the BE DAIs. The FE PCM devices can also route audio to more than 1 BE DAI.

Example - DPCM Switching playback from DAI0 to DAI1

Audio is being played to the Headset. After a while the user removes the headset and audio continues playing on the speakers.

Playback on PCM0 to Headset would look like :-

```
                    *************
PCM0 <============> *           * <====DAI0=====> Codec Headset
                    *           *
PCM1 <------------> *           * <----DAI1-----> Codec Speakers
                    *   DSP     *
PCM2 <------------> *           * <----DAI2-----> MODEM
                    *           *
PCM3 <------------> *           * <----DAI3-----> BT
                    *           *
                    *           * <----DAI4-----> DMIC
                    *           *
                    *           * <----DAI5-----> FM
                    *************
```

The headset is removed from the jack by user so the speakers must now be used :-

```
                    *************
PCM0 <============> *           * <----DAI0-----> Codec Headset
                    *           *
PCM1 <------------> *           * <====DAI1=====> Codec Speakers
                    *   DSP     *
PCM2 <------------> *           * <----DAI2-----> MODEM
                    *           *
PCM3 <------------> *           * <----DAI3-----> BT
                    *           *
                    *           * <----DAI4-----> DMIC
                    *           *
                    *           * <----DAI5-----> FM
                    *************
```

The audio driver processes this as follows :-

1. Machine driver receives Jack removal event.
2. Machine driver OR audio HAL disables the Headset path.
3. DPCM runs the PCM trigger(stop), hw_free(), shutdown() operations on DAI0 for headset since the path is now disabled.
4. Machine driver or audio HAL enables the speaker path.
5. DPCM runs the PCM ops for startup(), hw_params(), prepare() and trigger(start) for DAI1 Speakers since the path is enabled.

In this example, the machine driver or userspace audio HAL can alter the routing and then DPCM will take care of managing the DAI PCM operations to either bring the link up or down. Audio playback does not stop during this transition.

DPCM machine driver

The DPCM enabled ASoC machine driver is similar to normal machine drivers except that we also have to :-

1. Define the FE and BE DAI links.
2. Define any FE/BE PCM operations.
3. Define widget graph connections.

FE and BE DAI links

```
| Front End PCMs    |  SoC DSP  | Back End DAIs | Audio devices |

                     *************
PCM0 <------------> *           * <----DAI0-----> Codec Headset
                     *           *
PCM1 <------------> *           * <----DAI1-----> Codec Speakers
                     *   DSP     *
PCM2 <------------> *           * <----DAI2-----> MODEM
                     *           *
PCM3 <------------> *           * <----DAI3-----> BT
                     *           *
                     *           * <----DAI4-----> DMIC
                     *           *
                     *           * <----DAI5-----> FM
                     *************
```

For the example above we have to define 4 FE DAI links and 6 BE DAI links. The FE DAI links are defined as follows :-

```
static struct snd_soc_dai_link machine_dais[] = {
      {
              .name = "PCM0 System",
              .stream_name = "System Playback",
              .cpu_dai_name = "System Pin",
              .platform_name = "dsp-audio",
              .codec_name = "snd-soc-dummy",
              .codec_dai_name = "snd-soc-dummy-dai",
              .dynamic = 1,
              .trigger = {SND_SOC_DPCM_TRIGGER_POST, SND_SOC_DPCM_TRIGGER_POST},
              .dpcm_playback = 1,
      },
      .....< other FE and BE DAI links here >
};
```

This FE DAI link is pretty similar to a regular DAI link except that we also set the DAI link to a DPCM FE with the `dynamic = 1`. The supported FE stream directions should also be set with the `dpcm_playback` and `dpcm_capture` flags. There is also an option to specify the ordering of the trigger call for each FE. This allows the ASoC core to trigger the DSP before or after the other components (as some DSPs have strong requirements for the ordering DAI/DSP start and stop sequences).

The FE DAI above sets the codec and code DAIs to dummy devices since the BE is dynamic and will change depending on runtime config.

The BE DAIs are configured as follows :-

```
static struct snd_soc_dai_link machine_dais[] = {
      .....< FE DAI links here >
      {
              .name = "Codec Headset",
              .cpu_dai_name = "ssp-dai.0",
              .platform_name = "snd-soc-dummy",
              .no_pcm = 1,
              .codec_name = "rt5640.0-001c",
              .codec_dai_name = "rt5640-aif1",
              .ignore_suspend = 1,
              .ignore_pmdown_time = 1,
              .be_hw_params_fixup = hswult_ssp0_fixup,
              .ops = &haswell_ops,
              .dpcm_playback = 1,
              .dpcm_capture = 1,
```

```
        },
        .....< other BE DAI links here >
};
```

This BE DAI link connects DAI0 to the codec (in this case RT5460 AIF1). It sets the `no_pcm` flag to mark it has a BE and sets flags for supported stream directions using `dpcm_playback` and `dpcm_capture` above.

The BE has also flags set for ignoring suspend and PM down time. This allows the BE to work in a hostless mode where the host CPU is not transferring data like a BT phone call :-

```
                     *************
PCM0 <-------------> *           * <----DAI0-----> Codec Headset
                     *           *
PCM1 <-------------> *           * <----DAI1-----> Codec Speakers
                     *    DSP    *
PCM2 <-------------> *           * <====DAI2=====> MODEM
                     *           *
PCM3 <-------------> *           * <====DAI3=====> BT
                     *           *
                     *           * <----DAI4-----> DMIC
                     *           *
                     *           * <----DAI5-----> FM
                     *************
```

This allows the host CPU to sleep while the DSP, MODEM DAI and the BT DAI are still in operation.

A BE DAI link can also set the codec to a dummy device if the codec is a device that is managed externally.

Likewise a BE DAI can also set a dummy cpu DAI if the CPU DAI is managed by the DSP firmware.

FE/BE PCM operations

The BE above also exports some PCM operations and a `fixup` callback. The fixup callback is used by the machine driver to (re)configure the DAI based upon the FE hw params. i.e. the DSP may perform SRC or ASRC from the FE to BE.

e.g. DSP converts all FE hw params to run at fixed rate of 48k, 16bit, stereo for DAI0. This means all FE hw_params have to be fixed in the machine driver for DAI0 so that the DAI is running at desired configuration regardless of the FE configuration.

```
static int dai0_fixup(struct snd_soc_pcm_runtime *rtd,
                      struct snd_pcm_hw_params *params)
{
      struct snd_interval *rate = hw_param_interval(params,
                      SNDRV_PCM_HW_PARAM_RATE);
      struct snd_interval *channels = hw_param_interval(params,
                                              SNDRV_PCM_HW_PARAM_CHANNELS);

      /* The DSP will convert the FE rate to 48k, stereo */
      rate->min = rate->max = 48000;
      channels->min = channels->max = 2;

      /* set DAI0 to 16 bit */
      params_set_format(params, SNDRV_PCM_FORMAT_S16_LE);
      return 0;
}
```

The other PCM operation are the same as for regular DAI links. Use as necessary.

Widget graph connections

The BE DAI links will normally be connected to the graph at initialisation time by the ASoC DAPM core. However, if the BE codec or BE DAI is a dummy then this has to be set explicitly in the driver :-

```
/* BE for codec Headset -  DAI0 is dummy and managed by DSP FW */
{"DAI0 CODEC IN", NULL, "AIF1 Capture"},
{"AIF1 Playback", NULL, "DAI0 CODEC OUT"},
```

Writing a DPCM DSP driver

The DPCM DSP driver looks much like a standard platform class ASoC driver combined with elements from a codec class driver. A DSP platform driver must implement :-

1. Front End PCM DAIs - i.e. struct snd_soc_dai_driver.
2. DAPM graph showing DSP audio routing from FE DAIs to BEs.
3. DAPM widgets from DSP graph.
4. Mixers for gains, routing, etc.
5. DMA configuration.
6. BE AIF widgets.

Items 6 is important for routing the audio outside of the DSP. AIF need to be defined for each BE and each stream direction. e.g for BE DAI0 above we would have :-

```
SND_SOC_DAPM_AIF_IN("DAI0 RX", NULL, 0, SND_SOC_NOPM, 0, 0),
SND_SOC_DAPM_AIF_OUT("DAI0 TX", NULL, 0, SND_SOC_NOPM, 0, 0),
```

The BE AIF are used to connect the DSP graph to the graphs for the other component drivers (e.g. codec graph).

Hostless PCM streams

A hostless PCM stream is a stream that is not routed through the host CPU. An example of this would be a phone call from handset to modem.

```
                     *************
PCM0 <-------------> *           * <----DAI0-----> Codec Headset
                     *           *
PCM1 <-------------> *           * <====DAI1=====> Codec Speakers/Mic
                     *    DSP    *
PCM2 <-------------> *           * <====DAI2=====> MODEM
                     *           *
PCM3 <-------------> *           * <----DAI3-----> BT
                     *           *
                     *           * <----DAI4-----> DMIC
                     *           *
                     *           * <----DAI5-----> FM
                     *************
```

In this case the PCM data is routed via the DSP. The host CPU in this use case is only used for control and can sleep during the runtime of the stream.

The host can control the hostless link either by :-

1. Configuring the link as a CODEC <-> CODEC style link. In this case the link is enabled or disabled by the state of the DAPM graph. This usually means there is a mixer control that can be used to connect or disconnect the path between both DAIs.
2. Hostless FE. This FE has a virtual connection to the BE DAI links on the DAPM graph. Control is then carried out by the FE as regular PCM operations. This method gives more control over the DAI links, but requires much more userspace code to control the link. Its recommended to use CODEC<->CODEC unless your HW needs more fine grained sequencing of the PCM ops.

CODEC <-> CODEC link

This DAI link is enabled when DAPM detects a valid path within the DAPM graph. The machine driver sets some additional parameters to the DAI link i.e.

```
static const struct snd_soc_pcm_stream dai_params = {
      .formats = SNDRV_PCM_FMTBIT_S32_LE,
      .rate_min = 8000,
      .rate_max = 8000,
      .channels_min = 2,
      .channels_max = 2,
};

static struct snd_soc_dai_link dais[] = {
      < ... more DAI links above ... >
      {
              .name = "MODEM",
              .stream_name = "MODEM",
              .cpu_dai_name = "dai2",
              .codec_dai_name = "modem-aif1",
              .codec_name = "modem",
              .dai_fmt = SND_SOC_DAIFMT_I2S | SND_SOC_DAIFMT_NB_NF
                              | SND_SOC_DAIFMT_CBM_CFM,
              .params = &dai_params,
      }
      < ... more DAI links here ... >
```

These parameters are used to configure the DAI hw_params() when DAPM detects a valid path and then calls the PCM operations to start the link. DAPM will also call the appropriate PCM operations to disable the DAI when the path is no longer valid.

Hostless FE

The DAI link(s) are enabled by a FE that does not read or write any PCM data. This means creating a new FE that is connected with a virtual path to both DAI links. The DAI links will be started when the FE PCM is started and stopped when the FE PCM is stopped. Note that the FE PCM cannot read or write data in this configuration.