Application Report
SPRAC09A–December 2015–Revised January 2017

Interfacing DRA7xx Audio to Analog Codecs

Misael Lopez Cruz

ABSTRACT

DRA7xx Infotainment Application Processor offers an audio solution based on the multichannel audio serial port (McASP). McASP is a very flexible interface capable of interfacing with audio devices using protocols like Inter-IC Sound (I2S), PCM, TDM, S/PDIF. DRA7xx audio solution at Linux® kernel level uses ALSA System-on-Chip (ASoC) layer with independent drivers for the DMA engine, CPU-side audio interface and codecs.

This application report describes the DRA7xx audio device driver parameters and configuration needed to interface with new audio devices (analog codecs) at the kernel level.

NOTE: All programming models and use cases presented in this document are provided for educative purposes only and may differ from or be optimized for your applications.

All DRA7xx peripheral devices presented in this document are provided for illustration purposes and may be different from those in your system.

Contents

List of Figures

1 Introduction

DRA7xx infotainment application processor provides eight instances of the McASP. The McASP functions as a general-purpose audio serial port optimized for the requirements of various audio applications.

McASP1 and McASP2 support 16 serializers with independent TX/RX clock and sync domains. McASP3 through McASP8 support four serializers with unified TX/RX clock/sync domains.

2 DRA7xx Audio on Linux/Android Kernel

DRA7xx audio solution uses the ASoC layer in the Linux kernel. ASoC has the following independent components:

- Codec. Platform independent representation of the analog codecs. Codecs expose their Digital Audio interfaces (DAI) (audio/HiFi port, voice port, and so forth) that can be bound with CPU side interfaces
- CPU DAI. CPU-side audio interface. McASP is the CPU DAI in DRA7xx processors. The McASP driver can be found at sound/soc/davinci/davinci-mcasp.c
- Platform: The audio DMA engine. Two possible DMA controllers can be used:
  - sDMA driver which is shared with OMAP processors and can be found at drivers/dma/omap-dma.c and sound/soc/omap/omap-pcm.c
  - eDMA driver which can be found at arch/arm/common/edma.c and drivers/dma/edma.c
- Machine: Device dependent component that glues above components together:
  - The machine driver for TI DRA7 EVM primary card can be found at sound/soc/davinci/davinci-evm.c. This is a shared machine driver for simple McASP based cards
  - The machine driver for the DRA7 JAMR3 card can be found at sound/soc/davinci/dra7xx-jamr3-card.c

The McASP CPU DAI and machine drivers are discussed in more detail in the context of the TI Linux kernel in sections 3 and 4, respectively. sDMA/eDMA platform drivers are out of the scope of this document.

3 Configuring McASP Driver

The McASP device driver supports:

- I2S, PCM, TDM and S/PDIF protocols
- Up to 32-channels/sots per serializer in TDM mode. 384 slots in DIT mode, used for S/PDIF
  - The DIT mode is not actively supported in DRA7xx software releases
- Signed and unsigned 8/16/24/32 Little-Endian formats
- Sample rates from 8 kHz to 96 kHz
- Unified TX and RX clock and sync domains

3.1 Operational Mode

The McASP device driver differentiates two types of operation modes: I2S and DIT. I2S mode more generally refers to time division multiplexing (TDM) mode. Digital Audio Interface Transmission (DIT) mode is used for S/PDIF.

The operational mode is configured through the following Device Tree (DT) parameters:

- `op-mode`
  - 0: I2S mode, used for I2S, PCM and TDM protocols
  - 1: DIT mode, used for S/PDIF protocol
- `tdm-slots`. Number of TDM slots
  - 1: Reserved value, do not use
  - 2: Used for I2S
  - 3-32: Used for PCM and TDM protocols
  - 384: Used exclusively for S/PDIF

The maximum number of slots or channels is 32 per serializer. If the requested number of channels is higher than 32, the McASP driver tries to use as many enabled serializers as needed to satisfy the channel count request.

Single serializer per stream direction is quite common. This mode allows play and capture audio streams with channel counts equal or less than the TDM slots configured through DT, making the unused slots inactive. The state during those inactive slots is logic low for serializers configured in TX mode, and high impedance for those configured in RX mode.

McASP frame sync generator does not allow 1-slot mode, which is a reserved value in the `tdm-slots` property. Mono is still possible but requires at least two slots, the McASP driver automatically sets the first slot as the only active slot, while the rest of slots are put in inactive state. The McASP driver implements this behavior only for single serializer configuration.

3.2 Serializers

The McASP serializers can be used as separate audio links interfacing with different audio devices. Each serializer has its own pin (AXRn), configurable as transmitter or receiver.

For example, an audio link composed of one McASP and N analog stereo codecs can be configured in the following ways:

- *One serializer per stream direction*. McASP data transmit pin is shared with data-in of all N analog stereo codecs. Similarly, McASP data receive pin is shared with data-out of all codecs. TDM protocol is required with a bit clock frequency of:

$$BCLK = N * 2 * SampleRate * BitsPerSample \tag{1}$$

- *N serializers per stream direction*. Each analog codec is connected to different McASP transmit and receive pins. I2S protocol can be used with a bit clock frequency of:

$$BCLK = 2 * SampleRate * BitsPerSample \tag{2}$$

  Bit clock is *N* times less than in single serializer case, but requires more additional hardware connections between McASP and codecs pins.
- *num-serializer*. Number of serializers in the McASP instance, used and unused:
  - 16: McASP1 and McASP2
  - 4: McASP3 through McASP8
- *serial-dir*. Array describing the serializer direction
  - 0: *N*-th serializer is inactive
  - 1: *N*-th serializer is used for transmit
  - 2: *N*-th serializer is used for receive

3.3 Audio FIFO

McASP has two interface ports: configuration (CFG) and data (DAT). Read/write accesses through CFG port are done directly to the transmit/receive buffer (XRBUF) of a given serializer. XRBUF can store only a single audio sample. Read/write accesses through DAT port offer an additional advantage, the McASP Audio FIFO (AFIFO). The AFIFO can store up to 64 audio samples.

There are two AFIFOs per McASP instance: Write FIFO (WFIFO) and Read FIFO (RFIFO). The AFIFOs have an event pacer that is a configurable threshold that reduces the number of DMA transactions. The event pacer is helpful to reduce the effect of bus latency for high sample frequencies or long frames. McASP generates a DMA request when there is room in the WFIFO for threshold samples or if the RFIFO contains at least *threshold* samples.

AFIFO threshold is configured through the following DT parameters:

- `tx-num-evt`. WFIFO threshold. Number of samples between 1 and 64
- `rx-num-evt`. RFIFO threshold. Number of samples between 1 and 64

The AFIFO threshold imposes a constraint on the ALSA period size: the number of samples in the ALSA period has to be an integer multiple of the threshold. The McASP driver refines the originally requested threshold if it does not satisfy this constraint.

3.4 McASP CPU DAI

McASP interface capabilities are exposed via `struct snd_soc_dai_driver` in the CPU DAI driver. The driver exposes two entries: one for I2S/TDM mode named *davinci-mcasp.0* and one for DIT/SPDIF mode named *davinci-mcasp.1*. These two entries actually refer to the same interface so only one can be used at a time.

```
#define DAVINCI_MCASP_PCM_FMTS (SNDRV_PCM_FMTBIT_S8 | \
                                SNDRV_PCM_FMTBIT_U8 | \
                                SNDRV_PCM_FMTBIT_S16_LE | \
                                SNDRV_PCM_FMTBIT_U16_LE | \
                                SNDRV_PCM_FMTBIT_S24_LE | \
                                SNDRV_PCM_FMTBIT_U24_LE | \
                                SNDRV_PCM_FMTBIT_S24_3LE | \
                                SNDRV_PCM_FMTBIT_U24_3LE | \
                                SNDRV_PCM_FMTBIT_S32_LE | \
                                SNDRV_PCM_FMTBIT_U32_LE)

#define DAVINCI_MCASP_RATES     SNDRV_PCM_RATE_8000_192000

static struct snd_soc_dai_driver davinci_mcasp_dai[] = {
{
    .name       = "davinci-mcasp.0",
    .playback   = {
                .channels_min   = 1,
                .channels_max   = 32 * 16,
                .rates          = DAVINCI_MCASP_RATES,
                .formats        = DAVINCI_MCASP_PCM_FMTS,
    },
    .capture    = {
                .channels_min   = 1,
                .channels_max   = 32 * 16,
                .rates          = DAVINCI_MCASP_RATES,
                .formats        = DAVINCI_MCASP_PCM_FMTS,
    },
    .ops        = &davinci_mcasp_dai_ops,
},
{
    .name       = "davinci-mcasp.1",
    .playback   = {
                .channels_min   = 1,
                .channels_max   = 384,
                .rates          = DAVINCI_MCASP_RATES,
                .formats        = DAVINCI_MCASP_PCM_FMTS,
    },
    .ops       = &davinci_mcasp_dai_ops,
},
};
```

The `channel_max` value of 512 in the TDM DAI entry above refers to the total number of channels in the McASP instances that have 16 serializers, assuming all serializers are connected. Those interfaces are McASP1 and McASP2. McASP3 through McASP8 have four serializers and the max channel count is 128 when all serializers are in use.

3.5 Examples

3.5.1 McASP3 in I2S Mode

This example shows McASP3 instance configured in I2S mode (`op-mode`), which consists of two slots/channels (`tdm-slots`).

McASP3 instance has 4 serializers (`num-serializer`), but only two are used: AXR0 is used for transmit and AXR1 is used for receive (`serial-dir`), which are driven low during inactive slots. AXR2 and AXR3 are not used.

```
&mcasp3 {
       status = "okay";

       op-mode = <0>;   /* MCASP_IIS_MODE */
       tdm-slots = <2>;
       /* 4 serializer */
       serial-dir = <  /* 0: INACTIVE, 1: TX, 2: RX */
            1 2 0 0
       >;
       tx-num-evt = <8>;
       rx-num-evt = <8>;
};
```

3.5.2 McASP6 in TDM Mode

This example shows McASP6 configured in 8-slots (`num-slots`) TDM mode (`op-mode`).

McASP6 instance also has four serializers (`num-serializer`), only the first two are used (`serial-dir`) as shown in the previous example.

WFIFO and RFIFO are active, accessible through DAT port with a threshold of eight samples (`tx-num-evt`, `rx-num-evt`).

```
&mcasp6 {
       status = "okay";

       op-mode = <0>;   /* MCASP_IIS_MODE */
       tdm-slots = <8>;
       /* 4 serializer */
       serial-dir = <   /* 0: INACTIVE, 1: TX, 2: RX */
              1 2 0 0
       >;
       tx-num-evt = <8>;
       rx-num-evt = <8>;
};
```

4 Configuring Analog Codec Driver

Analog codec capabilities are declared in the driver's `snd_soc_dai_driver` structure. Example below shows the DAI capabilities of the tlv320aic3x codec. In this case, the tlv320aic3x driver exposes one DAI named *tlv320aic3x-hifi* that supports playback and capture and requires symmetric rates in full duplex mode. The DAI link name will be used later in the machine driver's DAI link array to specify what codec DAI is being connected.

```
#define AIC3X_RATES         SNDRV_PCM_RATE_8000_96000
#define AIC3X_FORMATS       (SNDRV_PCM_FMTBIT_S16_LE | SNDRV_PCM_FMTBIT_S20_3LE | \
                              SNDRV_PCM_FMTBIT_S24_3LE | SNDRV_PCM_FMTBIT_S32_LE)

...

static struct snd_soc_dai_driver aic3x_dai = {
    .name = "tlv320aic3x-hifi",
    .playback = {
        .stream_name = "Playback",
        .channels_min = 2,
        .channels_max = 2,
        .rates = AIC3X_RATES,
        .formats = AIC3X_FORMATS,},
    .capture = {
        .stream_name = "Capture",
        .channels_min = 2,
        .channels_max = 2,
        .rates = AIC3X_RATES,
        .formats = AIC3X_FORMATS,},
    .ops = &aic3x_dai_ops,
    .symmetric_rates = 1,
};
```

Once the codec DAI is connected to a CPU DAI in a DAI link, the DAI link will support only the common values of channels, rates and formats individually supported by the codec DAI and the CPU DAI.

Other analog codec driver parameters may be configured through DT/OF and may vary from codec to codec. For further details, see the codec driver documentation located in the Linux kernel device-tree bindings directory.

5 Configuring Machine Driver

The machine driver implements the system level settings such as the digital audio interface links definition and configuration, setup power supplies and clocks. It can also declare machine-level DAPM widgets and connect them with codec widgets. Digital Audio Power Management (DAPM) is out of the scope of this document.

5.1 *Digital Audio Interface Links*

The sound card can be composed of one or multiple DAI links. Each DAI link connects the CPU side with the codec side.

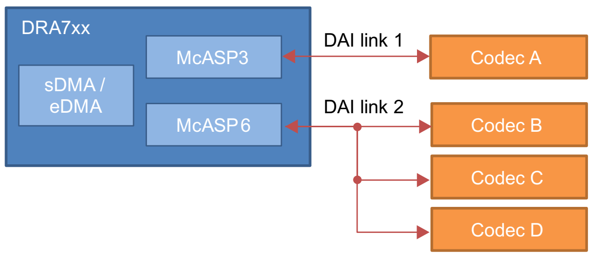

Figure 1. McASP-Based DAI Links in DRA7xx

All DAI links in the sound card are declared as elements in a `struct snd_soc_dai_link` array.

The DAI link array shown below was taken from the dra7evm primary sound card. It shows the declaration of a single link, which connects McASP3 with tlv320aic3106 codec using DSP_B protocol. This link is similar to the *DAI link 1* in Figure 1.

```
static struct snd_soc_dai_link dra7xx_evm_link = {
    .name           = "TLV320AIC3X",
    .stream_name    = "AIC3X",
    .codec_dai_name = "tlv320aic3x-hifi",
    .ops            = &dra7xx_ops,
    .init           = evm_dra7xx_init,
    .dai_fmt        = SND_SOC_DAIFMT_DSP_B | SND_SOC_DAIFMT_CBS_CFS |
                      SND_SOC_DAIFMT_IB_NF,
};
```

The DAI link is composed of:

- `name`/`stream_name`. The DAI link stream name and the codec DAI name define the PCM device name.

```
# cat /proc/asound/pcm
00-00: AIC3X tlv320aic3x-hifi-0 :  : playback 1 : capture 1
```

- `codec_dai_name`. Name of the codec DAI that this link connects. Codecs may have one or multiple DAIs. Refer to the codec driver to identify the DAI names.
- `codec_name`/`codec_of_node`. Name or DT reference to the analog codec node. Either the codec_name or the codec_of_node can be provided, but not both.
- `platform_name`/`platform_of_node`. Name or DT node reference to the platform node. The McASP driver registers the platform drivers it supports (sDMA and/or eDMA), so passing a DT reference to the McASP device node is also a valid option for the platform node.
- `cpu_dai_name`. Name of the CPU DAI that this link connects. McASP exposes two DAIs: *davinci-mcasp.0* for I2S/TDM and *davinci-mcasp.1* for DIT/SPDIF.
- `ops`. Machine-level PCM operations of the DAI link: startup, hw_params, prepare, trigger, hw_free, shutdown. For further details on stream life cycle, see the ALSA documentation **(located where?)**.
- `init`. DAI link initialization. Typically used to add DAI link specific widgets or other type of one-time settings.

The DAI link example shown above uses DT/OF nodes for codec and CPU DAI, these fields are populated dynamically when machine driver parses its own DT parameters.

Fully described DAI links are registered during machine driver *probe()* when the sound card is registered through `snd_soc_register_card()`. ASoC binds the DAI link components during sound card registration, that is, it associates the appropriate codec, CPU DAI and platform drivers needed for each link. A new PCM device is created for each successfully probed DAI link.

The *DAI link 2* in Figure 1 shows the connection between one McASP interface and three analog codecs (Codec B through D). This DAI multi-codec configuration can be achieved using the TDM protocol. The CPU side of the DAI link has a single component (McASP6 in this example), therefore, it generates a single PCM device once the DAI link is bound by ASoC. The *DAI link 2* uses the "multi-codec" add-on feature which was developed by TI and can be found only in kernels 3.8 and 3.14.

5.2 Data Format and Clocking

DAI link data format is configured during *startup()* or *hw_params()* operations. *hw_params()* offers the advantage of receiving stream parameters (channel count, format, sample rate, and so forth).

The data format configuration should be consistent between CPU DAI and the analog codec. McASP supports the following configuration parameters:

- Format. Configured through *snd_soc_dai_set_fmt()*:
  - Clock master/slave
    - `SND_SOC_DAIFMT_CBM_CFM`. McASP is slave for bit clock and frame sync
    - `SND_SOC_DAIFMT_CBS_CFS`. McASP is master for bit clock and frame sync
    - `SND_SOC_DAIFMT_CBM_CFS`. McASP is slave for bit clock and master for frame sync
  - Audio format
    - `SND_SOC_DAIFMT_DSP_A`. Frame sync is one bit-clock wide, 1-bit delay
    - `SND_SOC_DAIFMT_DSP_B`. Frame sync is one bit-clock wide, 0-bit delay. This format can be used for TDM protocol
    - `SND_SOC_DAIFMT_I2S`. Frame sync is one audio word wide, 1-bit delay
  - Signal inversion
    - `SND_SOC_DAIFMT_NB_NF`. Normal bit clock, normal frame sync.
      - McASP transmitter shifts data out on the falling edge of bit clock, receiver samples data on the rising edge
      - McASP frame sync generator starts the frame on the rising edge of frame sync
      - This parameter is recommended for I2S on McASP side
    - `SND_SOC_DAIFMT_NB_IF`. Normal bit clock, inverted frame sync
      - McASP transmitter shifts data out on the falling edge of bit clock, receiver samples data on the rising edge
      - McASP frame sync generator starts the frame on the falling edge of frame sync
    - `SND_SOC_DAIFMT_IB_NF`. Inverted bit clock, normal frame sync
      - McASP transmitter shifts data out on the rising edge of bit clock, receiver samples data on the falling edge
      - McASP frame sync generator starts the frame on the rising edge of frame sync
      - McASP frame sync generator starts the frame on the rising edge of frame sync
    - `SND_SOC_DAIFMT_IB_IF`. Inverted bit clock, inverted frame sync
      - McASP transmitter shifts data out on the rising edge of bit clock, receiver samples data on the falling edge
      - McASP frame sync generator starts the frame on falling edge of frame sync
      - This configuration can be used for PCM mode (bluetooth, modem)
- Clock source. Configured through *snd_soc_dai_set_sysclk()*:
  - Direction
    - `SND_SOC_CLOCK_IN`. McASP high-frequency clock is received externally from AHCLKX/AHCLKR. This clock is external to McASP but is still internal to DRA7xx. The AHCLKX/AHCLKR external clocks are mapped to the PRCM MCSPIi_AHCLKX/AHCLKR clocks.
    - `SND_SOC_CLOCK_OUT`. McASP high-frequency clock is generated internally. The AHCLKX/AHCLKR internal clocks are mapped to the MCASP_AHCLKX/MCASP_AHCLKR pins.
    - Other parameters to *snd_soc_dai_set_sysclk()* are ignored by McASP driver

- Clock divider. Configured through *snd_soc_dai_set_clkdiv()*:
  - Clock id
    - 0: Transmit/receive high-frequency clock dividers. High-frequency clock is sourced from AUXCLK which is tied to MCASPi_AUX_GFCLK
    - 1: Transmit/receive bit clock dividers. These dividers are active if bit clock is sourced internally as in `SND_SOC_DAIFMT_CBS_CFS`.
    - 2: BCLK-to-FSYNC ratio. This ratio must be set when the bit clock rate is higher than the minimum BCLK:

      $$Min\ BCLK = SampleRate * Channels * BitsPerSample \quad (3)$$

      For example, for 44.1 kHz, 16-bits stereo audio, the min BCLK is 1.4112 MHz. That is, one FSYNC period has 32 BCLKs. The BCLK-to-FSYNC ratio can be set to 32, but it is optional in this case.

      If the BCLK were instead 2.8224 MHz, then one FSYNC period has 64 BCLKs. The BCLK-to-FSYNC ratio must be set to 64.

5.3 Example: McASP3 in I2S Mode

This example shows the McASP3 configuration required to interface it with an analog codec in I2S mode (`SND_SOC_DAIFMT_I2S`). McASP3 is master (`SND_SOC_DAIFMT_CBS_CFS`).

McASP3 clock is sourced externally through AHCLKX pin (*snd_soc_dai_set_sysclk()*) and divided internally (*snd_soc_dai_set_clkdiv()*) to produce bit clock frequency shown in Equation 4.

$$BCLK = 2 * SampleRate * BitsPerSample \quad (4)$$

```
static int dra7_snd_media_hw_params(struct snd_pcm_substream *substream,
                         struct snd_pcm_hw_params *params)
{
    struct snd_soc_pcm_runtime *rtd = substream->private_data;
    struct snd_soc_dai *cpu_dai = rtd->cpu_dai;
    struct snd_soc_card *card = rtd->card;
    struct dra7_snd_data *card_data = snd_soc_card_get_drvdata(card);
    unsigned int bclk_freq;
    int ret;

    bclk_freq = dra7_get_bclk(params, card_data->media_slots);

    /* McASP driver requires inverted frame for I2S */
    ret = snd_soc_dai_set_fmt(cpu_dai, SND_SOC_DAIFMT_I2S |
                                   SND_SOC_DAIFMT_CBS_CFS |
                                   SND_SOC_DAIFMT_NB_NF);
    if (ret < 0) {
           dev_err(card->dev, "can't set CPU DAI format %d\n", ret);
           return ret;
    }

    /* Set McASP BCLK divider (clkid = 1) */
    ret = snd_soc_dai_set_clkdiv(cpu_dai, 1,
                       card_data->media_mclk_freq / bclk_freq);
    if (ret < 0) {
           dev_err(card->dev, "can't set CPU DAI clock divider %d\n", ret);
           return ret;
    }

    /* Set McASP sysclk from AHCLKX sourced from ATL */
    ret = snd_soc_dai_set_sysclk(cpu_dai, 0,
                       card_data->media_mclk_freq,
                       SND_SOC_CLOCK_IN);
    if (ret < 0) {
           dev_err(card->dev, "can't set CPU DAI sysclk %d\n", ret);
           return ret;
    }

    ...

    return ret;
}
```

Revision History

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

Changes from Original (December 2015) to A Revision **Page**

- Updates were made in Section 2. 2
- Updates were made in Section 3.5.1. 5
- Updates were made in Section 3.5.2. 5
- Update was made in Section 4. 6
- Update were made in Section 5.1. 7
- Updates were made in Section 5.2. 8