Application Note

Linux Audio on Sitara Socs

Suren Porwar

ABSTRACT

McASP is TI's Multichannel Audio Serial Port. The McASP functions as a general-purpose audio serial port optimized for the needs of multichannel, multi-zone audio applications.

It has been designed into many audio products, such as Audio/Video Receivers, sound bars, automotive amplifiers, and infotainment systems.

This application report is intended for engineers who are new to designing audio systems featuring McASP using Linux as the HLOS (High Level Operating System).

Table of Contents

Trademarks

All trademarks are the property of their respective owners.

1 Introduction

One of the most useful things about McASP is its flexible clocking scheme, which allows for complete independence between the receive and transmit ports, but that flexibility means that the engineers implementing an audio system with McASP will have to make some important design choices.

The primary goal of this document is to make it easier for an engineer to determine how to connect a McASP (or multiple McASPs) to audio devices in their system.

The audio subsystem present on various TI SoCs consists of two major components:

1. Multi-channel Audio Serial Port (McASP) - Provides a full-duplex serial interface between the host processor and external audio peripherals like codecs over industry-standard protocols like Inter-IC sound (I2S).
2. System DMA engine - Provides McASP with direct access to system memory to read audio samples from (for playback) or store audio samples to (for capture).

Along with the above, most TI EVMs and Starter Kits have line input/output jack(s) wired to an on-board codec that can convert between the analog signals and the digital protocol supported by McASP.

Fiirst, a few disclaimers:

- McASP is typically used to interface with devices using a time division-multiplexed (TDM) protocol, and in most cases those devices will be using a specific configuration of TDM called Inter-IC Sound (I2S). It is assumed that I2S is being used in all examples provided in this document, but that is somewhat arbitrary. The physical hookup of McASP to devices using multi-slot TDM or using I2S is the same.
- This document is not meant to be a detailed specification. Further, this document is intended to cover McASP in general, rather than McASP on a specific TI device. It covers the key characteristics of McASP at a high level, but the device-specific Technical Reference Manual (TRM) is still the best place to go for architectural and chip-level details.

2 Software Architecture

The Advanced Linux Sound Architecture (ALSA), now the most popular architecture in Linux system, provides audio and MIDI functionality to the Linux operating system.

ALSA has the following significant features:

- Efficient support for all types of audio interfaces, from consumer sound cards to professional multichannel audio interfaces.
- Fully modularized sound drivers.
- SMP and thread-safe design.
- User space library (alsa-lib) to simplify application programming and provide higher level functionality.
- Support for the older Open Sound System (OSS) API, providing binary compatibility for most OSS programs.

ALSA System on Chip (ASoC) layer is designed for SoC audio. The overall project goal of the ASoC layer provides better ALSA support for embedded system on chip processors and portable audio CODECs.

The ASoC layer also provides the following features:

- CODEC independence. Allows reuse of CODEC drivers on other platforms and machines.
- Easy I2S/PCM audio interface setup between CODEC and SoC. Each SoC interface and CODEC registers its audio interface capabilities with the core.
- Dynamic Audio Power Management (DAPM). DAPM is an ASoC technology designed to minimize audio subsystem power consumption no matter what audio-use case is active. DAPM guarantees the lowest audio power state at all times and is completely transparent to user space audio components. DAPM is ideal for mobile devices or devices with complex audio requirements.
- Pop and click reduction. Pops and clicks can be reduced by powering the CODEC up/down in the correct sequence (including using digital mute). ASoC signals the CODEC when to change power states.
- Machine-specific controls. Allow machines to add controls to the sound card, for example, volume control for speaker amp.

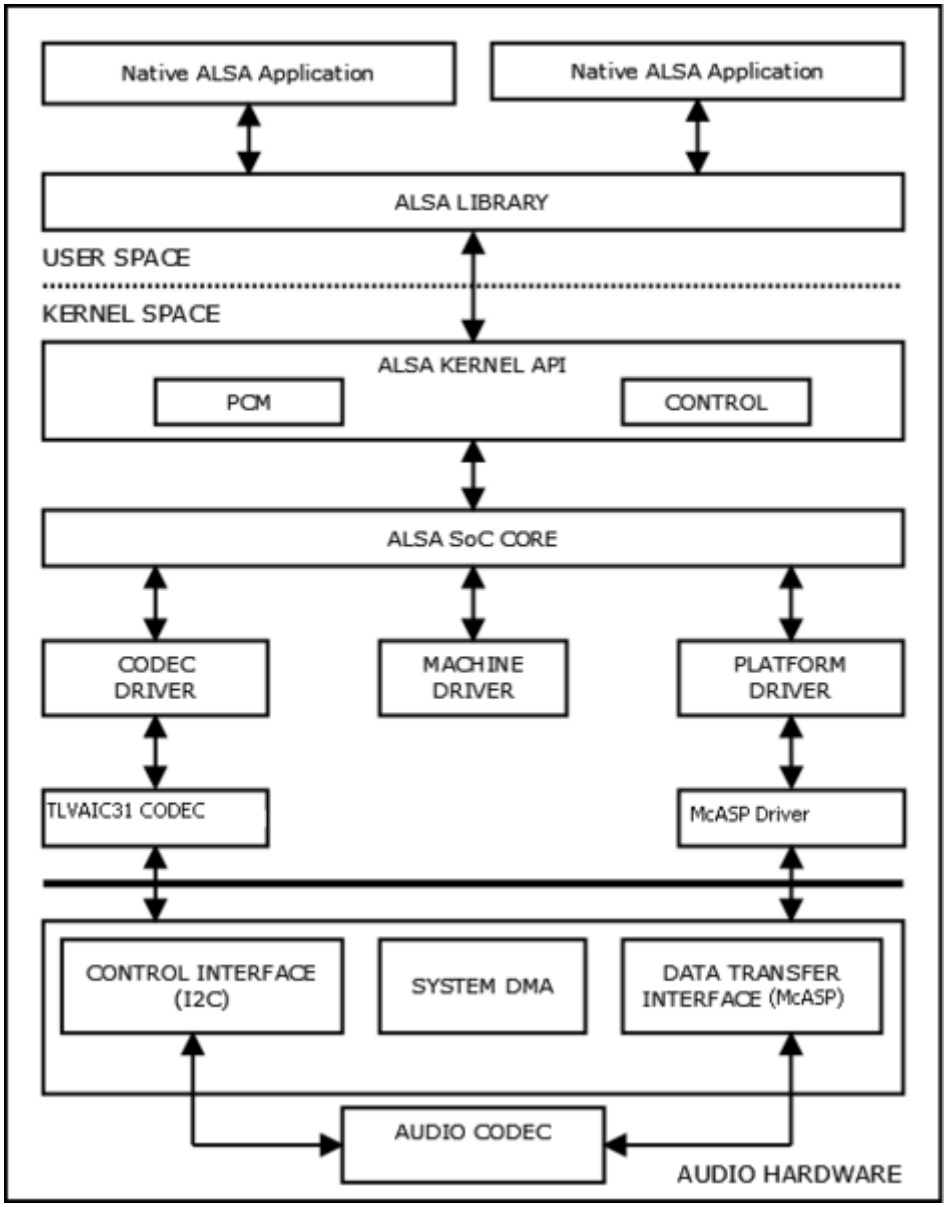

Figure 2-1. ASOC Architecture

ASoC splits an embedded audio system into multiple re-usable component drivers:

- Codec class drivers: The codec class driver is platform independent and contains audio controls, audio interface capabilities, codec DAPM definition and codec IO functions. This class extends to BT, FM and MODEM ICs if required. Codec class drivers should be generic code that can run on any architecture and machine.
- Platform class drivers: The platform class driver includes the audio DMA engine driver, digital audio interface (DAI) drivers (e.g. I2S, AC97, PCM) and any audio DSP drivers for that platform.
- Machine class driver: The machine driver class acts as the glue that describes and binds the other component drivers together to form an ALSA “sound card device”. It handles any machine specific controls and machine level audio events (e.g. turning on an amp at start of playback).

More detailed information about ASoC can be found in the Linux kernel documentation in the Linux OS source tree at linux/Documentation/sound/alsa/soc and at www.alsa-project.org/main/index.php/ASoC.

3 Sound Card Information

The registered sound card information can be listed as follows using the commands aplay -l and arecord -l.

For example, the stereo sound card is registered as card 0.

```
root@am62axx-evm:~# aplay -l
**** List of PLAYBACK Hardware Devices ****
card 0: AM62AxSKEVM [AM62Ax-SKEVM], device 0: 2b10000.audio-controller-tlv320aic3x-hifi tlv320aic3x-
hifi-0 [2b10000.audio-controller-tlv320aic3x-hifi tlv320aic3x-hifi-0]
  Subdevices: 1/1
  Subdevice #0: subdevice #0
```

Alsa mixer controls to set playback volume and gains on the Record path:

```
amixer -c 0 cset numid=71 on
amixer -c 0 cset numid=70 on
amixer -c 0 cset numid=64 on
amixer -c 0 cset numid=71 on
amixer -c 0 cset numid=70 on
amixer -c 0 cset numid=64 on
amixer -c 0 cset numid=65 on
amixer -c 0 cset numid=15 127
```

4 McASP - External Signals

- Data Pins: McASP may have up to 16 serializers. These serializers are connected to data pins, referred to as AXR pins. They are named as such ("audio transmit/receive" = "AXR") because any McASP data pin can be configured to functions as an input or as an output. Note that if an AXR pin is running as a transmitter, it is necessary to re-initialize the McASP in order to re-configure it as a receiver. Dynamically switching direction during operation is not supported. The McASP data pin hookup is straightforward. Since any McASP AXR pin can be configured for either transmit or receive, the designer is free to choose whichever pins are most convenient for the system.
- Mute Pins: McASP can have up to two mute pins:
- AMUTEIN – This is an input. Some external devices have a mute output pin; such a signal can be connected to AMUTEIN. McASP can be configured to mute I2S output under this condition.
- AMUTE – This is an output that McASP can be configured to drive under specific error conditions. For more details, see the device-specific TRM. Mute pin hookup is also straightforward. Configuration of the AMUTE pin's behavior takes some planning (see your device-specific TRM), but it is easy to figure out where to connect it. If the downstream device has a mute input pin, that is what AMUTE should be connected to.
- Clock Pins: McASP may have up to six clock pins, all (and any) of which can be generated internally or sourced externally. There are three types of clock signals on McASP: high frequency (AHCLK), bit (ACLK), and frame sync (AFS). However, each has a transmit version (X, for example, AHCLKX) and a receive version(R, for example, AFSR). Every McASP has a receive clocking section and a transmit clocking section. These are sometimes referred to as the receive port and the transmit port, and each clock port constitutes a clock zone. Therefore, each McASP has two potential clock zones. These can be run asynchronously with respect to each other, but in some cases, synchronous operation is appropriate.
- AHCLKX and AHCLKR – These are high-frequency clocks pins, sometimes referred to as master clocks (often referred to on audio codecs as MCLK). McASP uses a master clock for one purpose: to divide it down and generate a bit clock. There are several cases where a master clock is not required.
- ACLKX and ACLKR – These are bit clocks, often referred to on audio devices as BCK. Data is clocked in and out with respect to bit clock edges. Furthermore, much of McASP's internal logic (state machines, and so forth) runs off of the bit clock, so a bit clock is ALWAYS required.
- AFSX and AFSR – These are the frame sync clocks, often referred to as word clocks, or more commonly as left-right clocks (LRCK). The "left-right" terminology comes from stereo audio in the I2S format, in which the edges of the frame sync clock denote the bits corresponding to the left and right channels. The frame sync clocks run at the audio stream's sample rate.

The following figure is a simplified representation of McASP's external signals, omitting the mute pins.

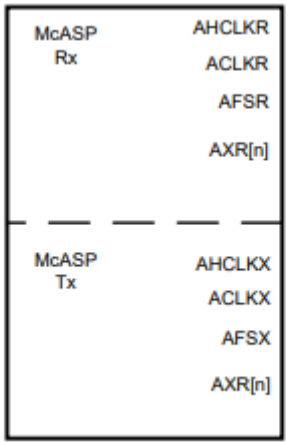

Figure 4-1. MCASP External Signals

5 MCASP Clock Generation and Configurations

In some cases, MCASP is required to act as clock master and must generate clocks.

The following figure is a simplified block diagram of MCASP's clock generation architecture. For more detail, see the device-specific TRM.

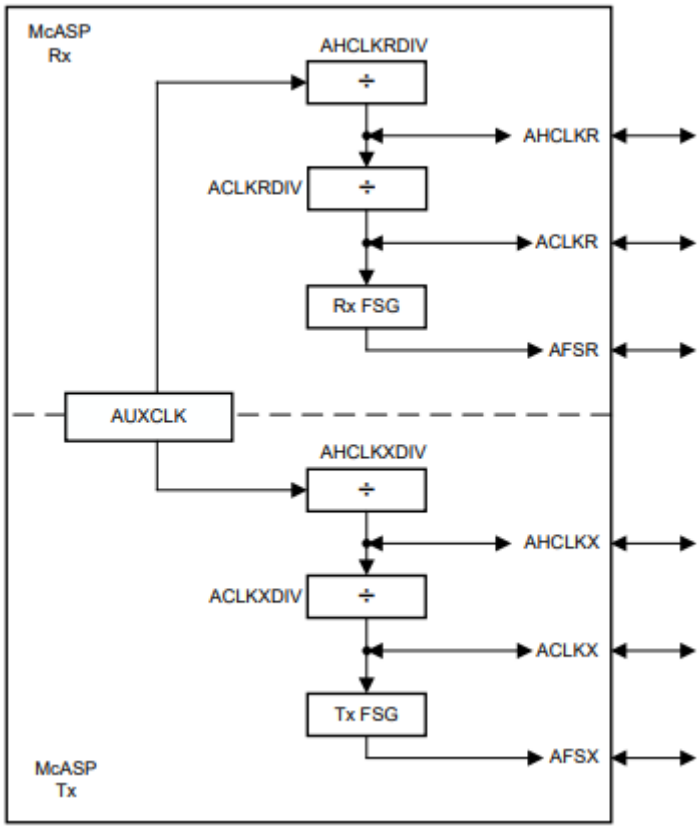

Figure 5-1. MCASP Clock Generation Architecture

The block marked AUXCLK in Figure 2 is a clock source that is available to both the Rx and Tx sections of McASP. The source of AUXCLK varies, depending on the device, but it is often derived directly from the device's main system clock source (usually an on-board oscillator or an externally generated square wave).

For both the receive and transmit sections, the AUXCLK is fed into an integer high-frequency clock divider: AHCLKRDIV or AHCLKXDIV. This clock signal can be divided down to generate the McASP's Rx or Tx master clock: AHCLKR or AHCLKX. This signal can be driven out on its corresponding pin, if configured to do so. It also feeds the next clock divider: ACLKRDIV or ACLKXDIV. Similarly, the integer bit clock divider ACLK[R/X]DIV generates the McASP's bit clock: ACLKR or ACLKX. That signal can be driven out on the corresponding pin. Note that the bit clock divider can instead be fed by the AHCLK[R/X] pin. The next stage of clock generation looks similar. These are the receive and transmit Frame Sync Generators (FSGs). The output of the FSGs will be AFSR and AFSX. These can also be driven out of their corresponding clock pins. The frame sync generator can be fed by an external bit clock provided to the ACLK[R/X] pin. This is not common, as the bit clock and frame sync are usually both internally generated, or both externally sourced.

A key point about the McASP's clock generation architecture is that McASP has integer dividers (and frame sync generators) that can be used to generate internal clocks, and those internal clocks may be driven out of their corresponding device pins.

6 Dummy Sound Card DTS Changes

For using Dummy codec in DTS, you will need to apply the following patch:

https://patchwork.kernel.org/project/alsa-devel/patch/5652E348.8080002@invoxia.com/

Device tree changes to include dummy codec and register as dummy sound card.

```
codec_test: codec_test {
compatible = "linux,snd-soc-dummy";
#sound-dai-cells = <0>;
status="okay";
};

codec_test: codec_test {
                compatible = "linux,snd-soc-dummy";
                #sound-dai-cells = <0>;
                status="okay";
        };



codec_audio: sound {

                compatible = "simple-audio-card";
                simple-audio-card,name = "AM62X-DUMMY";
                simple-audio-card,format = "i2s";
                simple-audio-card,bitclock-master = <&sound_master0>;
                simple-audio-card,frame-master = <&sound_master0>;



                sound_master0: simple-audio-card,cpu {
                sound-dai = <&mcasp1>;
                system-clock-direction-out;
                };



                simple-audio-card,codec {
                        sound-dai = <&codec_test>;
                };
        };
```

7 Single DAI Link or a Single Sound Card

```
-----------------------------------------
Example 1 - single DAI link:
-----------------------------------------

sound {
    compatible = "simple-audio-card";
    simple-audio-card,name = "VF610-Tower-Sound-Card";
    simple-audio-card,format = "left_j";
    simple-audio-card,bitclock-master = <&dailink0_master>;
    simple-audio-card,frame-master = <&dailink0_master>;
    simple-audio-card,widgets =
        "Microphone", "Microphone Jack",
        "Headphone", "Headphone Jack",
        "Speaker", "External Speaker";
    simple-audio-card,routing =
        "MIC_IN", "Microphone Jack",
        "Headphone Jack", "HP_OUT",
        "External Speaker", "LINE_OUT";

    simple-audio-card,cpu {
        sound-dai = <&sh_fsi2 0>;
    };

    dailink0_master: simple-audio-card,codec {
        sound-dai = <&ak4648>;
        clocks = <&osc>;
    };
};
```

8 Multiple DAI Links - Single Card but Multiple Sub-Devices

```
-----------------------------------------
Example 2 - many DAI links:
-----------------------------------------

sound {
    compatible = "simple-audio-card";
    simple-audio-card,name = "Cubox Audio";

    simple-audio-card,dai-link@0 {      /* I2S - HDMI */
        reg = <0>;
        format = "i2s";
        cpu {
            sound-dai = <&audio1 0>;
        };
        codec {
            sound-dai = <&tda998x 0>;
        };
    };

    simple-audio-card,dai-link@1 {      /* S/PDIF - HDMI */
        reg = <1>;
        cpu {
            sound-dai = <&audio1 1>;
        };
        codec {
            sound-dai = <&tda998x 1>;
        };
    };

    simple-audio-card,dai-link@2 {      /* S/PDIF - S/PDIF */
        reg = <2>;
        cpu {
            sound-dai = <&audio1 1>;
        };
        codec {
            sound-dai = <&spdif_codec>;
        };
    };
};
```

Reference: https://www.kernel.org/doc/Documentation/devicetree/bindings/sound/simple-card.txt

9 MCASP - Practical Examples

McASP typically connects to audio devices that have the same types of clock pins (master, bit, and framesync) mentioned in Section 3, so it is fairly easy to figure out which kind of McASP clock pins to a connect to another audio device's clock pins: bit clocks go to bit clocks, frame syncs go to frame syncs, and so forth. Most questions regarding McASP are about which clock port to use (receive or transmit) and which direction (input or output) the pins should be configured for. The easiest way to clear up this aspect of audio designs with McASP is to use some real-world examples.

An engineer that is new to digital audio system design might have to think twice about whether to hook their audio device to a McASP's receive or transmit port. The first thing to consider is which direction the data is going. If the McASP will receive data from the audio device, then that device's clock pins should be connected to the McASP's receive clock pins.

10 McASP as a Receiver

It is very common for an audio system, such as a soundbar or a car stereo, to have an analog input. In all such cases, an audio ADC is required. Once the A-to-D conversion has occurred, the resulting digital data stream must be fed to a McASP. Since the McASP receives data from the ADC, the McASP's Rx clock pins are connected to the ADC's clock pins.

10.1 ADC or Codec as Clock Master

If the ADC must be configured as a clock master, then it will be the device generating all clocks. It has been established that the device will be connected to McASP's Rx clock pins, as shown in the following figure.

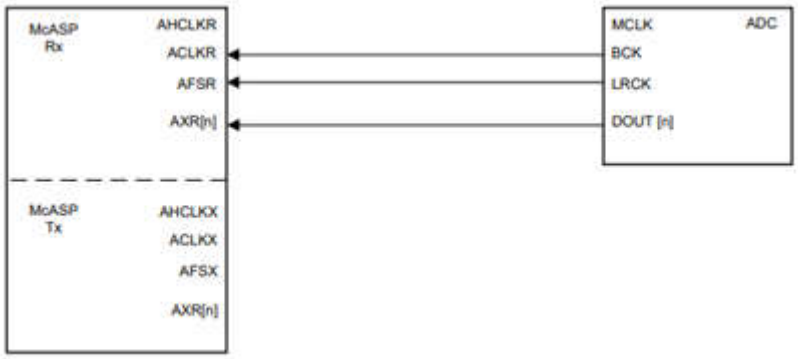

Figure 10-1. ADC as Clock Master

10.2 Device Tree Changes - Codec as Master and MCASP as Slave

```
tlv320_mclk: clk-0 {
    #clock-cells = <0>;
    compatible = "fixed-clock";
    clock-frequency = <12288000>;
};

codec_audio: sound {
    compatible = "simple-audio-card";
    simple-audio-card,name = "AM62x-SKEVM";
    simple-audio-card,widgets =
        "Headphone",    "Headphone Jack",
        "Line",     "Line In",
        "Microphone",   "Microphone Jack";
    simple-audio-card,routing =
        "Headphone Jack",   "HPLOUT",
        "Headphone Jack",   "HPROUT",
        "LINE1L",       "Line In",
        "LINE1R",       "Line In",
        "MIC3R",        "Microphone Jack",
        "Microphone Jack",  "Mic Bias";
    simple-audio-card,format = "dsp_b";
    simple-audio-card,bitclock-master = <&sound_master>;
    simple-audio-card,frame-master = <&sound_master>;
    simple-audio-card,bitclock-inversion;

    simple-audio-card,cpu {
        sound-dai = <&mcasp1>;
    };

    sound_master: simple-audio-card,codec {
        sound-dai = <&tlv320aic3106>;
        clocks = <&tlv320_mclk>;
    };
};

tlv320aic3106: audio-codec@1b {
    #sound-dai-cells = <0>;
    compatible = "ti,tlv320aic3106";
    reg = <0x1b>;
```

```
    ai3x-micbias-vg = <1>;  /* 2.0V */
};
```

Reference: https://git.ti.com/cgit/ti-linux-kernel/ti-linux-kernel/tree/arch/arm64/boot/dts/ti/k3-am62p5-sk.dts?h=ti-linux-6.12.y

Important things to note:

- McASP's receive clock pins, ACLKR and AFSR, are operating as inputs. The ADC, as the clock master, is driving those clock lines. McASP must be configured as the clock slave, which means that the ACLKR and AFSR pins must be configured as inputs, and must be configured for an EXTERNAL clock source. These two things sound equivalent, but they are not. McASP pin direction and clock source are two different settings. For more information on how to configure this, see the device-specific TRM.
- AHCLKR is not used, only bit clock and frame sync are required. AHCLKR is only necessary if the McASP's Rx section is fed a high-frequency clock and must then divide it down in order to generate bitclock and frame sync. In the above example, this is not necessary, since there is no need for McASP to generate Rx clocks.

```
main_mcasp1_pins_default: main-mcasp1-default-pins {
        pinctrl-single,pins = <
            AM62PX_IOPAD(0x0090, PIN_INPUT, 2) /* (U24) GPMC0_BE0n_CLE.MCASP1_ACLKX */
            AM62PX_IOPAD(0x0098, PIN_INPUT, 2) /* (AA24) GPMC0_WAIT0.MCASP1_AFSX */
            AM62PX_IOPAD(0x008c, PIN_OUTPUT, 2) /* (T25) GPMC0_WEn.MCASP1_AXR0 */
            AM62PX_IOPAD(0x0084, PIN_INPUT, 2) /* (R25) GPMC0_ADVn_ALE.MCASP1_AXR2 */
        >;
    };

&mcasp1 {
    status = "okay";
    #sound-dai-cells = <0>;

    pinctrl-names = "default";
    pinctrl-0 = <&main_mcasp1_pins_default>;

    op-mode = <0>;          /* MCASP_IIS_MODE */
    tdm-slots = <2>;

    serial-dir = <  /* 0: INACTIVE, 1: TX, 2: RX */
           1 0 2 0
           0 0 0 0
           0 0 0 0
           0 0 0 0
    >;
};
```

11 MCASP as Transmitter

11.1 Device Tree Changes - With Codec as Slave and MCASP as Master

Many ADCs do not have the ability to generate clocks themselves and must be operated as clock slaves. Since McASP receives data from the ADC, it is appropriate to configure its Rx clock pins as outputs, and generate clocks that will be driven into the ADC's clock pins. Recall that McASP has integer clock dividers. If the device's AUXCLK runs at a frequency that can be used to generate all of the necessary McASP Rx clocks at the desired frequencies, then AHCLKR, ACLKR, and AFSR will all be generated internally and all configured as outputs, as shown in the following figure.

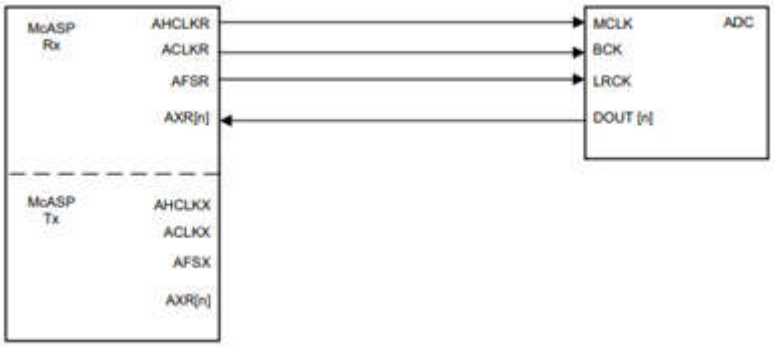

Figure 11-1. ADC as Clock Slave, All McASP Clocks Generated Internally

Important things to note:

- All McASP Rx clocks are outputs.
- AHCLKR is driven out of McASP. This is because ADCs usually require a high-frequency clock for operation. For more information, see the device-specific ADC data manual.

```
clocks {
                /* 25MHz reference crystal */
                ref25: oscillator {
                        compatible = "fixed-clock";
                        #clock-cells = <0>;
                        clock-frequency = <24576000>;
            clock-output-names = "mclk_24m576";
                };
        };

        dep_audio: sound-tac5112 {
                compatible = "simple-audio-card";
                simple-audio-card,name = "AM62x-TAC5112-DEP";
                simple-audio-card,format = "i2s";
                simple-audio-card,bitclock-master = <&master>;
                simple-audio-card,frame-master  = <&master>;
        simple-audio-card,bitclock-inversion;

        simple-audio-card,widgets =
            "Line", "Line In",
            "Line", "Line Out";
        simple-audio-card,routing =
            "AIN1", "Line In",
            "Line Out", "OUT1";

                master: simple-audio-card,cpu {
                        sound-dai = <&mcasp0>;
            system-clock-direction-out;
                };

            master1: simple-audio-card,codec {
                        sound-dai = <&dep_codec>;
                };
        };
};

    main_mcasp0_pins_default: main-mcasp0-pins-default {
        pinctrl-single,pins = <
```

```
            /* Currently unused on the daughtercard */
             AM62X_IOPAD(0x01A8, PIN_OUTPUT, 0)  /* (D20) MCASP0_AFSR */
             AM62X_IOPAD(0x01A4, PIN_OUTPUT, 0)  /* (B20) MCASP0_ACLKR */
             AM62X_IOPAD(0x01A0, PIN_OUTPUT, 0)  /* (E18) MCASP0_AXR0 */
             AM62X_IOPAD(0x019C, PIN_INPUT, 0)  /* (B18) MCASP0_AXR1 */
        >;
    };

&mcasp0 {
    status = "okay";
    #sound-dai-cells = <0>;

    pinctrl-names = "default";
    pinctrl-0 = <&main_mcasp0_pins_default>;

    op-mode = <0>;          /* MCASP_IIS_MODE */
    tdm-slots = <2>;

    auxclk-fs-ratio = <2177>;

    serial-dir = <  /* 0: INACTIVE, 1: TX, 2: RX */
           1 2 0 0
           0 0 0 0
           0 0 0 0
           0 0 0 0
    >;
};
```

In the above case, the clock generation is shown in the following figure:

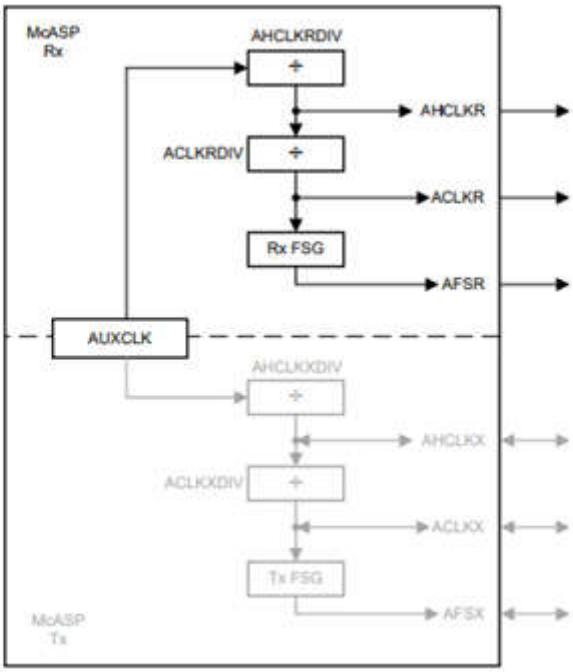

Figure 11-2. ADC as Clock Slave, Rx Clocks Generated Internally

The AUXCLK feeds the AHCLKRDIV, which feeds ACLKRDIV, which feeds the Rx FSG.

Ref: k3-am62-main.dtsi

```
mcasp0: audio-controller@2b00000 {
        compatible = "ti,am33xx-mcasp-audio";
        reg = <0x00 0x02b00000 0x00 0x2000>,
              <0x00 0x02b08000 0x00 0x400>;
        reg-names = "mpu", "dat";
        interrupts = <GIC_SPI 236 IRQ_TYPE_LEVEL_HIGH>,
                     <GIC_SPI 235 IRQ_TYPE_LEVEL_HIGH>;
        interrupt-names = "tx", "rx";
```

```
        dmas = <&main_bcdma 0 0xc500 0>, <&main_bcdma 0 0x4500 0>;
        dma-names = "tx", "rx";

        clocks = <&k3_clks 190 0>;
        clock-names = "fck";
        assigned-clocks = <&k3_clks 190 0>;
        assigned-clock-parents = <&k3_clks 190 2>;
        power-domains = <&k3_pds 190 TI_SCI_PD_EXCLUSIVE>;
        status = "disabled";
};

mcasp1: audio-controller@2b10000 {
        compatible = "ti,am33xx-mcasp-audio";
        reg = <0x00 0x02b10000 0x00 0x2000>,
              <0x00 0x02b18000 0x00 0x400>;
        reg-names = "mpu", "dat";
        interrupts = <GIC_SPI 238 IRQ_TYPE_LEVEL_HIGH>,
                     <GIC_SPI 237 IRQ_TYPE_LEVEL_HIGH>;
        interrupt-names = "tx", "rx";

        dmas = <&main_bcdma 0 0xc501 0>, <&main_bcdma 0 0x4501 0>;
        dma-names = "tx", "rx";

        clocks = <&k3_clks 191 0>;
        clock-names = "fck";
        assigned-clocks = <&k3_clks 191 0>;
        assigned-clock-parents = <&k3_clks 191 2>;
        power-domains = <&k3_pds 191 TI_SCI_PD_EXCLUSIVE>;
        status = "disabled";
};

mcasp2: audio-controller@2b20000 {
        compatible = "ti,am33xx-mcasp-audio";
        reg = <0x00 0x02b20000 0x00 0x2000>,
              <0x00 0x02b28000 0x00 0x400>;
        reg-names = "mpu", "dat";
        interrupts = <GIC_SPI 240 IRQ_TYPE_LEVEL_HIGH>,
                     <GIC_SPI 239 IRQ_TYPE_LEVEL_HIGH>;
        interrupt-names = "tx", "rx";

        dmas = <&main_bcdma 0 0xc502 0>, <&main_bcdma 0 0x4502 0>;
        dma-names = "tx", "rx";

        clocks = <&k3_clks 192 0>;
        clock-names = "fck";
        assigned-clocks = <&k3_clks 192 0>;
        assigned-clock-parents = <&k3_clks 192 2>;
        power-domains = <&k3_pds 192 TI_SCI_PD_EXCLUSIVE>;
        status = "disabled";
};
```

In many cases, the AUXCLK is running at a frequency that cannot be divided down to the desired bit clock and frame sync rates (recall that McASP only has integer dividers - sometimes the math does not workout). In that case, an appropriate external clock source of some kind must be fed to the AHCLKR pin and then divided down to generate bit clock and frame sync. Figure 7 illustrates this scenario.

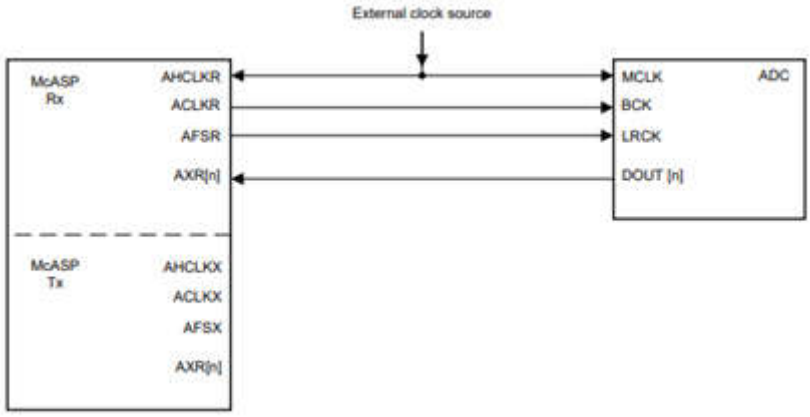

Figure 11-3. ADC as Clock Slave, External Master Clock

The external clock feeds both the ADC's MCLK and the AHCLKR pin. Clock generation for this scenario is illustrated in the following figure.

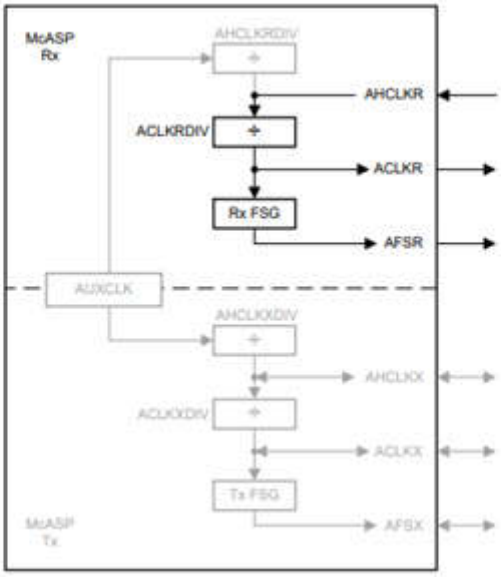

Figure 11-4. ADC as Clock Slave, External Master Clock Used for Rx Clock Generation

In this case, AUXCLK is not used; therefore, AHCLKRDIV is not used. The external master clock source is used to generate the ACLKR and AFSR signals, which are then fed to the ADC's BCK and LRCK pins, satisfying the ADC's requirement to act as a clock slave.

Assuming that this external clock source is providing the clocks to AUDIO_EXT_REFCLK0 and using MCASP2 on AM62x then the Device tree needs to add the following lines:

```
    fixed_audio_refclk: clk-0 {
        status = "okay";

        #clock-cells = <0>;
        compatible = "fixed-clock";
        clock-frequency = <24576000>;
    };

    pinctrl-0 = <&board_pins_refclk>;

    board_pins_mcasp2: board-pins-mcasp2 {
        pinctrl-single,pins = <
            AM62X_IOPAD(0xf41d0, PIN_OUTPUT, 8) /* [Speaker_Audio_FSYNC] (A15)
UART0_CTSn.MCASP2_AFSX */
            AM62X_IOPAD(0xf41d4, PIN_OUTPUT, 8) /* [Speaker_Audio_CLK] (B15)
UART0_RTSn.MCASP2_ACLKX */
            AM62X_IOPAD(0xf41d8, PIN_OUTPUT, 8) /* [Speaker_Audio_OUT] (C15) MCAN0_TX.MCASP2_AXR0 */
        >;
    };

    board_pins_refclk: board-pins-refclk {
        pinctrl-single,pins = <
            AM62X_IOPAD(0xf4190, PIN_INPUT, 2) /* [Audio_EXT_Refclk0] (AE22)
RGMII2_RD3.AUDIO_EXT_REFCLK0 */
            AM62X_IOPAD(0xf41f0, PIN_INPUT, 0) /* [Ext_Refclk1] (A18) EXT_REFCLK1.EXT_REFCLK1 */
            AM62X_IOPAD(0xf413c, PIN_OUTPUT, 7) /* [Audio_Refclk_Enable] (AE18) RGMII1_TD2.GPIO0_77
*/
        >;
    };

&mcasp2 {
    status = "okay";
    #sound-dai-cells = <0>;

    /* CLOCKS:
     * BOARD_AUDIO_EXT_REFCLK0 <192 12> --> AHCLKR IN <192 9>
```

```
     * BOARD_AUDIO_EXT_REFCLK0 <192 18> --> AHCLKX IN <192 15>
     * REFCLK runs at 24,576 MHz
     */
    assigned-clocks        = <&k3_clks 192 9>, <&k3_clks 192 15>;
    assigned-clock-parents = <&k3_clks 192 12>, <&k3_clks 192 18>;
    assigned-clock-rates   = <24576000>, <24576000>;

    pinctrl-names = "default";
    pinctrl-0 = <&board_pins_mcasp2>;

    op-mode = <0>;          /* MCASP_IIS_MODE */
    tdm-slots = <2>;

    serial-dir = <  /* 0: INACTIVE, 1: TX, 2: RX */
           1 0 0 0
           0 0 0 0
           0 0 0 0
           0 0 0 0
    >;
    tx-num-evt = <0>;
    rx-num-evt = <0>;
};
```

12 References

ALSA links

1. ALSA SoC Project Homepage
2. ALSA Project Homepage
3. ALSA User Space Library
4. Using ALSA Audio API Author: Paul Davis

Audio hardware codecs

1. TLV320AIC31 - Low-Power Stereo CODEC with HP Amplifier
2. TLV320AIC3104 - Low-Power Stereo CODEC with HP Amplifier
3. TLV320AIC3111 - Low-Power Stereo CODEC with Embedded miniDSP and Stereo Class-D Speaker Amplifier
4. TLV320AIC3106 - Low-Power Stereo Audio CODEC

Linux Kernel bindings

1. https://www.kernel.org/doc/Documentation/devicetree/bindings/sound/simple-card.txt
2. https://www.kernel.org/doc/Documentation/devicetree/bindings/sound/davinci-mcasp-audio.txt
3. https://www.kernel.org/doc/Documentation/devicetree/bindings/clock/fixed-clock.txt
4. https://github.com/devicetree-org/dt-schema/blob/main/dtschema/schemas/clock/clock.yaml

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you fully indemnify TI and its representatives against any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale, TI’s General Quality Guidelines, or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products. Unless TI explicitly designates a product as custom or customer-specified, TI products are standard, catalog, general purpose devices.

TI objects to and rejects any additional or different terms you may propose.

Last updated 10/2025