参考資料

Application Note

Sitara Soc 上の Linux オーディオ

Suren Porwar

概要

McASP は、TI のマルチチャネル オーディオ シリアル ポートです。McASP は、マルチチャネルおよびマルチゾーンのオーディオ アプリケーションの要件に最適化された汎用オーディオ シリアル ポートとして機能します。

オーディオ / ビデオ レシーバー、サウンド バー、車載アンプ、インフォテインメント システムなど、多くのオーディオ製品に採用されています。

このアプリケーション レポートは、HLOS (ハイレベル オペレーティング システム) として Linux を使用し、McASP を採用したオーディオ システムの設計に新規取り組むエンジニアを対象としています。

目次

商標

1 はじめに

McASP の最も有用な特長の一つは、受信ポートと送信ポートを完全に独立させることができる柔軟なクロック構成で。しかし、すその柔軟性ゆえに、McASP を用いたオーディオ システムを実装するエンジニアは、いくつかの重要な設計上の選択を行う必要があります。

このドキュメントの主な目的は、エンジニアが McASP (または複数の McASP) をシステム内のオーディオ デバイスに接続する方法を簡単に決定できるようにすることです。

各種 TI SoC に搭載されているオーディオ サブシステムは、主に次の 2 つのコンポーネントで構成されています:

1. マルチチャネル オーディオ シリアル ポート(McASP):Inter-IC sound(I2S) などの業界標準プロトコルを介して、ホスト プロセッサとコーデックなどの外部オーディオ周辺機器との間にフル デュプレックスのシリアル インターフェイスを提供します。
2. システム DMA エンジン - 再生時にはオーディオ サンプルを読み出し、キャプチャ時にはオーディオ サンプルを書き込むために、McASP へシステム メモリへの直接アクセスを提供します。

上記に加えて、ほとんどの TI 製 EVM およびスタータ キットには、アナログ信号と McASP がサポートするデジタル プロトコルとの間で変換を行うオンボード コーデックに接続されたライン入力 / 出力ジャックが備わっています。

最初にいくつかの免責事項があります:

- McASP は通常、時分割多重 (TDM) プロトコルを使用するデバイスとのインターフェースに用いられ、多くの場合、それらのデバイスは Inter-IC Sound (I2S) と呼ばれる特定の TDM 構成を使用しています。本書で示すすべての例では I2S を使用することを前提としていますが、これはある程度任意の設定です。マルチスロット TDM を使用する場合でも I2S を使用する場合でも、McASP とデバイスとの物理的な接続は同じです。
- このドキュメントは詳細な仕様ではありません。さらに、本書は特定の TI デバイス上の McASP ではなく、McASP 全般を対象としています。本書では McASP の主要な特性を高いレベルで概説していますが、アーキテクチャやチップ レベルの詳細については、デバイス固有のテクニカル リファレンス マニュアル (TRM) を参照するのが最適です。

2 ソフトウェア アーキテクチャ

Advanced Linux Sound Architecture (ALSA) は、Linux システムで最も一般的なアーキテクチャであり、Linux オペレーティング システムにオーディオと MIDI 機能を提供します。

ALSA には次のような特徴があります:

- 民生用サウンドカードから業務用のマルチチャネル オーディオ インターフェイスまで、あらゆる種類のオーディオ インターフェイスを効率的にサポートします。
- フルモジュール化されたサウンド ドライバ。
- SMP とスレッドセーフな設計。
- アプリケーション プログラミングを簡素化し、より高レベルの機能を提供するユーザー空間ライブラリ (alsa-lib)。
- 旧来の Open Sound System (OSS) API をサポートし、多くの OSS プログラムに対してバイナリ互換性を提供します。

ALSA システム オン チップ (ASoC) レイヤは、SoC オーディオ用に設計されています。ASoC レイヤのプロジェクト全体の目標は、チップ プロセッサおよびポータブル オーディオ CODEC 向けに、より優れた ALSA サポートを提供することです。

ASoC レイヤには次の機能もあります:

- CODEC への非依存性。他のプラットフォームやマシンでコーデック ドライバを再利用できます。
- コーデックと SoC の間で、I2S/PCM オーディオ インターフェイスを簡単に設定できます。各 SoC インターフェイスおよび CODEC は、自身のオーディオ インターフェイス機能をコアに登録します。
- ダイナミック オーディオ パワー マネージメント (DAPM)。DAPM は、どのようなオーディオ使用ケースが有効であっても、オーディオ サブシステムの消費電力を最小化するために設計された ASoC の技術です。DAPM は常に最小のオーディオ電力状態を保証し、ユーザー空間のオーディオ コンポーネントに対して完全に透過的です。DAPM は、複雑なオーディオ要件を持つモバイル デバイスやデバイスに最適です。
- ポップ音およびクリック音の低減。コーデックを正しいシーケンスでパワーアップ/ パワーダウン (デジタル ミュートの使用を含む) することで、ポップ音やクリック音を低減できます。ASoC は、電源状態を変更するタイミングをコーデックに通知します。
- 機械固有のコントロール。マシンがサウンド カードに制御項目を追加できるようにします。例えば、スピーカ アンプの音量制御などです。

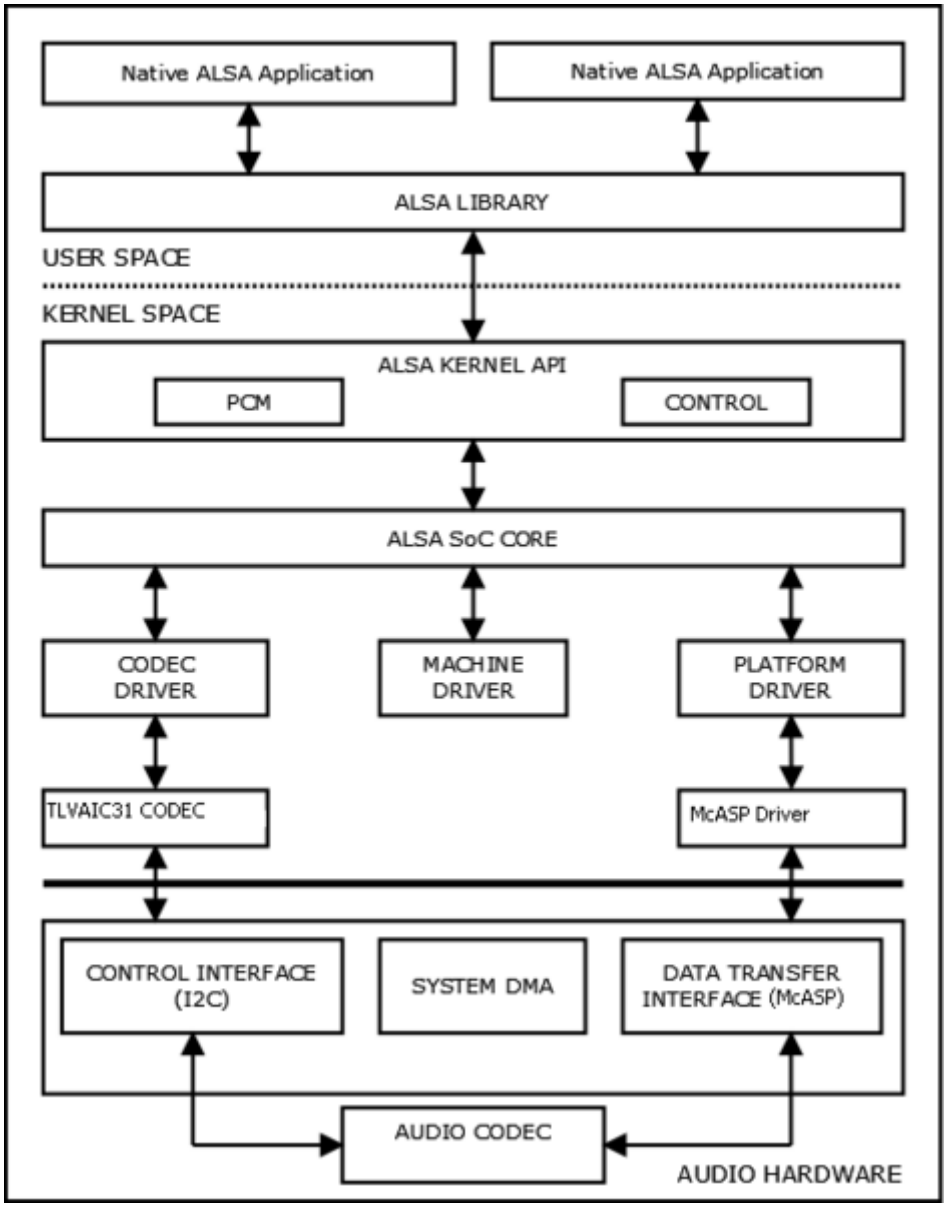

図 2-1. ASOC アーキテクチャ

ASoC は、組み込みオーディオシステムを複数の再利用可能なコンポーネントドライバに分割します:

- コーデック クラスドライバ:コーデック クラスドライバはプラットフォームに依存せず、オーディオ制御、オーディオ インターフェイス機能、コーデック DAPM 定義、およびコーデック IO 関数を含みます。このクラスは、必要に応じて BT、FM、およびモデム IC に拡張されます。コーデック クラスドライバは、任意のアーキテクチャとマシンで実行できる汎用コードである必要があります。
- プラットフォーム クラスドライバ:プラットフォーム クラスドライバには、そのプラットフォーム向けのオーディオ DMA エンジンドライバ、デジタル オーディオ インターフェイス (DAI) ドライバ (I2S、AC97、PCM など)、およびオーディオ DSP ドライバが含まれます。
- マシン クラスドライバ:マシンドライバ クラスは、他のコンポーネントドライバを記述して結び付け、ALSA の「サウンド カード デバイス」を構成するための接着剤として機能します。再生開始時にアンプをオンにするなど、マシン固有の制御やマシン レベルのオーディオ イベントを処理します。

ASoC に関するより詳細な情報は、Linux OS ソース ツリー内の Linux カーネル ドキュメント linux/Documentation/sound/alsa/soc および www.alsa-project.org/main/index.php/ASoC で確認できます。

3 サウンド カード情報

登録されているサウンド カードの情報は、aplay -l および arecord -l コマンドを使用して以下のように一覧表示できます。

たとえば、ステレオ サウンド カードはカード 0 として登録されます。

```
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 ミキサ コントロールを使用して、録音パスの再生音量とゲインを設定します:

```
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 — 外部信号

- データ ピン: McASP は、最大 16 個のシリアライザを搭載できます。これらのシリアライザは、AXR ピンと呼ばれるデータ ピンに接続されています。これらは、任意の McASP データ ピンを入力または出力のいずれとしても設定できるため、「audio transmit/receive」=「AXR」と呼ばれています。AXR ピンがトランスミッタとして動作している場合、それをレシーバとして再設定するには、McASP を再初期化する必要がある点に注意してください。動作中の方向の動的切り替えはサポートされていません。McASP のデータ ピンの接続は簡単です。どの McASP AXR ピンも送信または受信のいずれかに設定できるため、設計者はシステムにとって最も都合のよいピンを自由に選択できます。
- ミュート ピン：McASP は最大 2 つのミュート ピンを搭載できます：
- AMUTEIN - 入力です。一部の外部デバイスにはミュート出力ピンがあり、そのような信号を AMUTEIN に接続できます。この状況では、McASP は I2S 出力をミュートするように構成できます。
- AMUTE – これは、特定のエラー条件で McASP を駆動するように構成できる出力です。詳細については、各デバイスの TRM をご覧ください。ミュート ピンの接続も簡単です。AMUTE ピンの動作設定にはある程度の検討が必要です (デバイス固有の TRM を参照してください) が、どこに接続すべきかを判断するのは容易です。ダウンストリーム デバイスにミュート入力ピンがある場合、AMUTE はそのピンに接続します。
- クロック ピン：McASP には最大 6 本のクロック ピンがあり、内部で生成するか、外部から供給することができます。McASP には、高周波 (AHCLK) 、ビット (ACLK) 、フレーム同期 (AFS) の 3 種類のクロック信号があります。ただし、それぞれには送信バージョン (X、AHCLKX など) と受信バージョン (R、AFSR など) があります。すべての McASP には、受信クロック セクションと送信クロック セクションがあります。これらは、受信ポートおよび送信ポートと呼ばれることがあり、各クロック ポートは 1 つのクロック ゾーンを構成します。したがって、各 McASP には 2 つの潜在的なクロック ゾーンがあります。これらは互いに非同期に実行できますが、場合によっては同期動作が適切です。
- AHCLKX および AHCLKR – これらは高周波クロックピンであり、マスタ クロックと呼ばれることもあります (オーディオ コーデックでは MCLK と呼ばれることもあります)。McASP は、マスタ クロックを分周してビット クロックを生成するという 1 つの目的のためにマスタクロックを使用します。マスタ クロックが必要ない場合もいくつかあります。
- ACLKX および ACLKR – これらはビット クロックであり、オーディオ デバイスでは BCK と呼ばれます。データは、ビット クロック エッジを基準にして入出力されます。さらに、McASP の内部ロジック (ステート マシンなど) の多くはビット クロックで動作するため、ビット クロックは常に必要です。
- AFSX および AFSR：これらはフレーム同期クロックで、ワード クロック、またはより一般的には左右クロック (LRCK) と呼ばれます。「左-右」という用語は、I2S 形式のステレオ オーディオに由来しており、フレーム同期クロックのエッジが左チャネルおよび右チャネルに対応するビットを示します。フレーム同期クロックは、オーディオ ストリームのサンプル レートで動作します。

次の図は、McASP の外部信号の概略図で、ミュート ピンを省略しています。

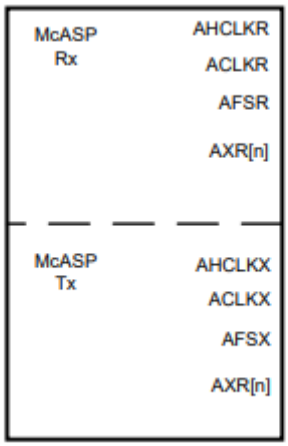

図 4-1. MCASP 外部信号

5 MCASP クロックの生成と構成

場合によっては、MCASP がクロック マスタとして動作し、クロックを生成する必要があります。

次の図は、MCASP クロック生成アーキテクチャの概略ブロック図を示しています。詳細については、各デバイスの TRM をご覧ください。

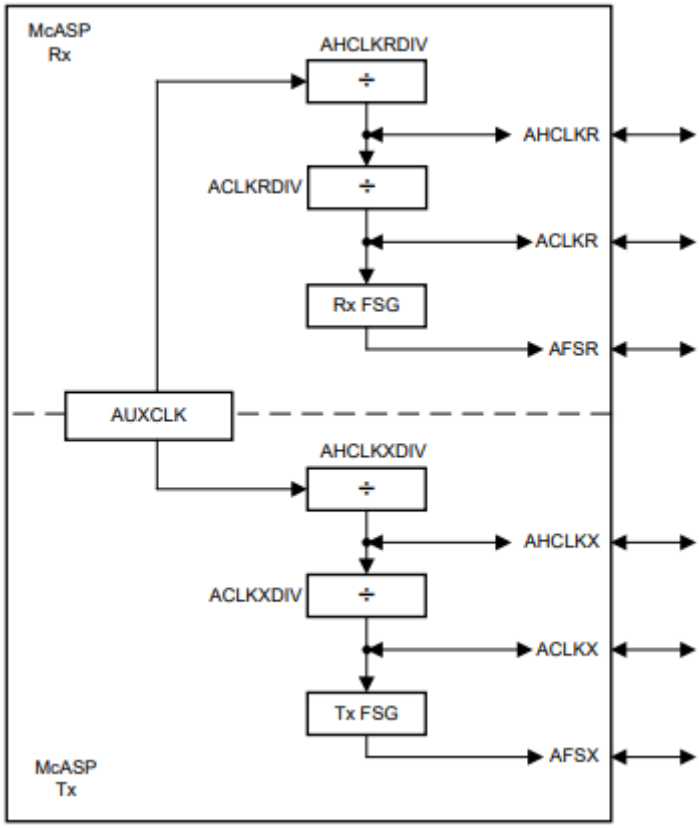

図 5-1. MCASP クロック生成アーキテクチャ

図 2 で AUXCLK と示されているブロックは、McASP の Rx および Tx の両方で使用可能なクロック ソースです。AUXCLK のクロック ソースはデバイスによって異なりますが、多くの場合、デバイスのメイン システム クロック ソース (通常はオンボード発振器または外部生成された矩形波) から直接生成されます。

受信および送信の両セクションにおいて、AUXCLK は整数の高周波クロック デバイダに入力されます:AHCLKRDIV または AHCLKXDIV。このクロック信号を分周して、McASP の Rx または Tx マスタクロックを生成できます:AHCLKR または AHCLKX。この信号は、設定により対応するピンから出力することができます。また、次のクロック デバイダも供給します:ACLKRDIV または ACLKXDIV。同様に、整数ビット クロック デバイダ ACLK[R/X] DIV は、McASP のビット クロックを生成します:ACLKR または ACLKX。この信号は、対応するピンで駆動できます。なお、ビットクロック デバイダには、代わりに AHCLK[R/X] ピンからクロックを入力することもできます。クロック生成の次の段階は、同様のものになります。これらは、受信および送信のフレーム同期ジェネレータ (FSG) です。FSG の出力は AFSR と AFSX になります。これらは、対応するクロック ピンから駆動することもできます。フレーム同期ジェネレータは、ACLK[R/X] ピンに供給される外部ビット クロックによって供給できます。これは一般的ではありません。通常、ビット クロックとフレーム同期は両方とも内部で生成されるか、または両方とも外部から供給されます。

McASP のクロック生成アーキテクチャにおける重要なポイントは、McASP が内部クロックを生成するための整数デバイダ (およびフレーム同期ジェネレータ) を備えており、それらの内部クロックを対応するデバイス ピンから出力できる点です。

6 ダミー サウンド カード DTS の変更

DTS でダミー コーデックを使用するには、次のパッチを適用する必要があります:

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

デバイス ツリーがダミー コーデックを含むように変更され、ダミー サウンド カードとして登録されます。

```
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 シングル DAI リンクまたはシングル サウンド カード

```
-----------------------------------------
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 複数の DAI リンク - 単一カードで複数のサブデバイス

```
-----------------------------------------
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>;
        };
    };
};
```

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

9 MCASP - 実例

McASP は通常、セクション 3 で述べたマスタ、ビット、フレーム同期といった同種のクロック ピンを備えたオーディオ デバイスに接続されるため、どの McASP のクロック ピンを相手デバイスのどのクロック ピンに接続するかは比較的容易に判断できます。つまり、ビット クロックはビット クロックへ、フレーム同期はフレーム同期へ、という具合です。McASP に関する質問の多くは、どのクロック ポート (受信または送信) を使用するか、また各ピンを入力と出力のどちらに設定すべきか、という点に集中しています。McASP を使用してオーディオ設計のこの側面を解消する最も簡単な方法は、いくつかの実際的な例を使用することです。

デジタル オーディオ システム設計に不慣れなエンジニアは、オーディオ デバイスを McASP の受信ポートと送信ポートのどちらに接続すべきかについて、慎重に考える必要があるかもしれません。最初に考慮すべきことは、データがどの方向に向かっているかということです。McASP がオーディオ デバイスからデータを受信する場合、そのデバイスのクロック ピンを McASP の受信クロック ピンに接続する必要があります。

10 McASP をレシーバとして使用

サウンド バーやカー ステレオなどのオーディオ システムでは、アナログ入力を備えていることが非常に一般的です。いずれの場合も、オーディオ ADC が必要です。A/D 変換が発生した後、結果として得られるデジタル データ ストリームを McASP に供給する必要があります。McASP は ADC からデータを受信するため、McASP の Rx クロック ピンは ADC のクロック ピンに接続されます。

10.1 クロック マスタとしての ADC またはコーデック

ADC をクロック マスタとして設定する必要がある場合、すべてのクロックはこのデバイスによって生成されます。次の図に示すように、デバイスを McASP の Rx クロック ピンに接続することが確立されています。

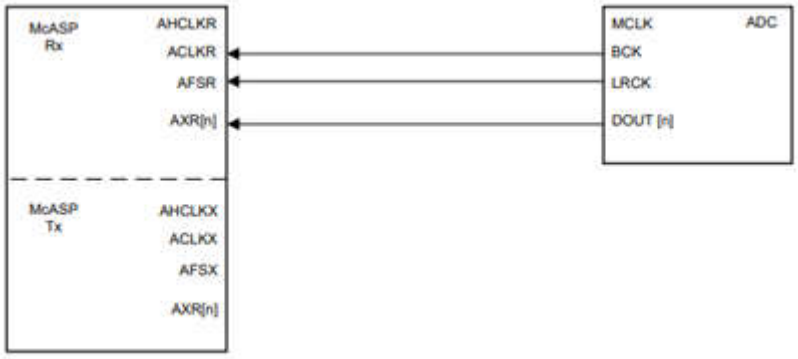

図 10-1. クロック マスタとしての ADC

10.2 デバイス ツリーの変更 — コーデックをマスタ、MCASP をスレーブとして使用

```
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 */
};
```

参照: 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

注意すべき重要事項:

- McASP の受信クロック ピン ACLKR および AFSR は入力として動作します。クロック マスタとして ADC はクロック ラインを駆動しています。McASP はクロック スレーブとして設定する必要があり、そのため ACLKR および AFSR ピンは入力として設定し、外部クロック ソースを使用するように構成する必要があります。この 2 つは同じ意味に聞こえますが、実際には異なります。McASP ピンの方向とクロック ソースは 2 つの異なる設定です。この設定方法の詳細については、各デバイス固有の TRM を参照してください。
- AHCLKR は使用されず、ビット クロックとフレーム同期のみが必要です。AHCLKR は、McASP の Rx セクションに高周波クロックが入力され、それを分周してビット クロックとフレーム同期を生成する必要がある場合にのみ必要です。上記の例では、McASP で Rx クロックを生成する必要がないため、これは不要です。

```
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 をトランスミッタとして使用

11.1 デバイス ツリーの変更 — コーデックをスレーブ、MCASP をマスタとする

多くの ADC は自らクロックを生成する機能を持たないため、クロック スレーブとして動作させる必要があります。McASP は ADC からデータを受信するため、Rx クロック ピンを出力として設定し、ADC のクロック ピンに供給するクロックを生成する構成が適切です。McASP には整数クロック デバイダがあることに注意してください。デバイスの AUXCLK が、必要なすべての McASP Rx クロックを所望の周波数で生成できる周波数で動作している場合、AHCLKR、ACLKR、および AFSR はすべて内部で生成され、次の図に示すように、すべて出力として設定されます。

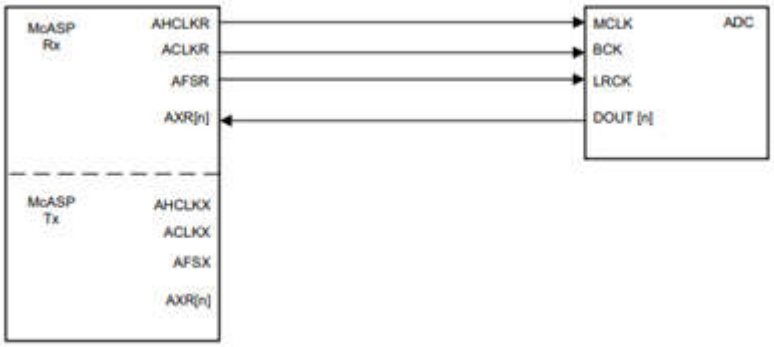

図 11-1. ADC をクロック スレーブとして使用し、すべての McASP クロックを内部で生成

注意すべき重要事項:

- すべての McASP Rx クロックは出力です。
- AHCLKR は McASP から駆動されます。これは、ADC は通常、動作に高周波クロックを必要とするためです。詳細については、デバイス固有の ADC データ マニュアルを参照してください。

```
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
    >;
};
```

上の例では、クロック生成を次の図に示します:

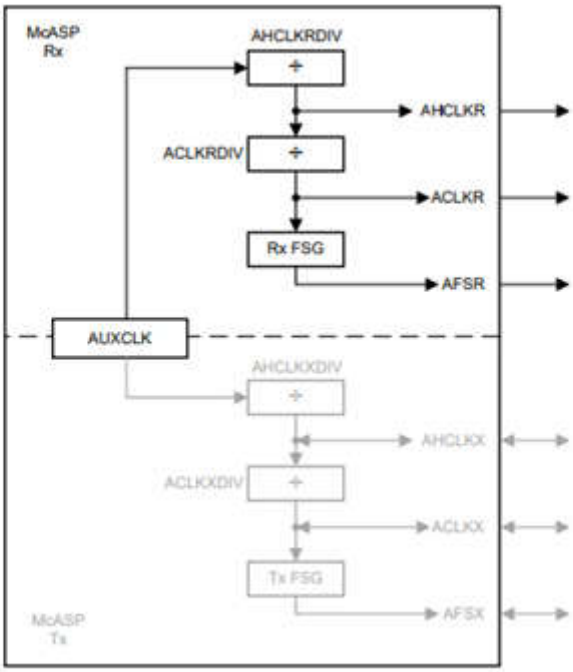

図 11-2. ADC をクロック スレーブとして使用し、Rx クロックを内部で生成

AUXCLK は AHCLKRDIV を供給し、ACLKRDIV が供給されて、Rx FSG が供給されます。

参考文献: 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";
};
```

多くの場合、AUXCLK は、目的のビット クロックおよびフレーム同期レートに分周することはできない周波数で動作します (McASP には整数分周器のみがあることに注意してください。時には計算がうまくいかないこともあります)。その場合は、適切な外部クロック ソースを AHCLKR ピンに入力し、そのクロックを分周してビット クロックとフレーム同期を生成する必要があります。図 7 はこのシナリオを図示したものです。

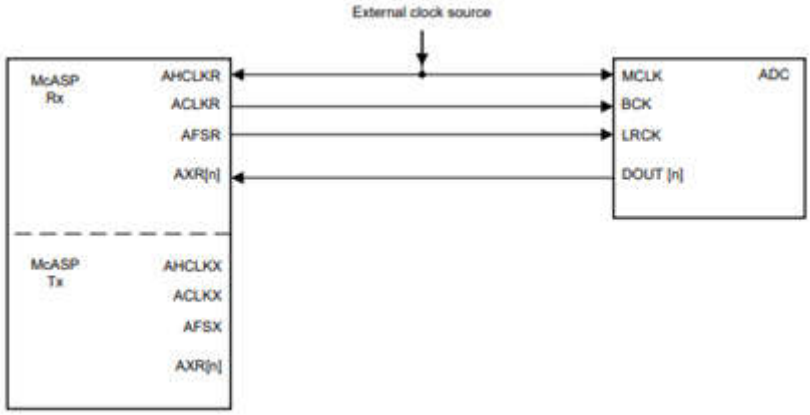

図 11-3. クロック スレーブとして動作する ADC (外部マスタ クロックを使用)

外部クロックは、ADC の MCLK と AHCLKR ピンの両方に信号を供給します。このシナリオのクロック生成を次の図に示します。

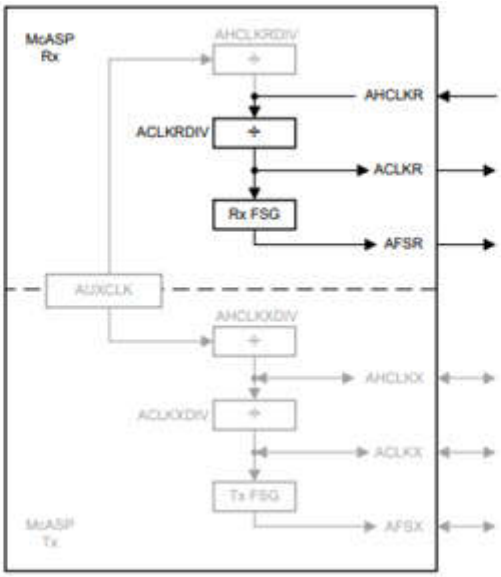

図 11-4. クロック スレーブとして ADC、Rx クロック生成に使用される外部マスタ クロック

この場合、AUXCLK を使用しないため、AHCLKRDIV は使用されません。外部マスタ クロック ソースを用いて ACLKR および AFSR 信号を生成し、それらを ADC の BCK ピンと LRCK ピンに入力することで、ADC がクロック スレーブとして動作するための要件を満たします。

この外部クロック ソースが AUDIO_EXT_REFCLK0 にクロックを供給し、AM62x で MCASP2 を使用する場合、デバイス ツリーに以下の行を追加する必要があります:

```
    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 参考資料

ALSA リンク

1. ALSA SoC プロジェクト ホームページ
2. ALSA プロジェクト ホームページ
3. ALSA ユーザ スペース ライブラリ
4. ALSA Audio API Author の使用: Paul Davis

オーディオ ハードウェア コーデック

1. TLV320AIC31 - HP アンプ内蔵の低消費電力ステレオ コーデック
2. TLV320AIC3104 - HP アンプ内蔵の低消費電力ステレオ コーデック
3. TLV320AIC3111 - miniDSP およびステレオ Class-D スピーカ アンプを内蔵した低消費電力ステレオ コーデック
4. TLV320AIC3106 - 超低消費電力ステレオ オーディオ コーデック

Linux カーネル バインディング

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

重要なお知らせと免責事項

TI は、技術データと信頼性データ (データシートを含みます)、設計リソース (リファレンス デザインを含みます)、アプリケーションや設計に関する各種アドバイス、Web ツール、安全性情報、その他のリソースを、欠陥が存在する可能性のある「現状のまま」提供しており、商品性および特定目的に対する適合性の黙示保証、第三者の知的財産権の非侵害保証を含むいかなる保証も、明示的または黙示的にかかわらず拒否します。

これらのリソースは、TI 製品を使用する設計の経験を積んだ開発者への提供を意図したものです。(1) お客様のアプリケーションに適した TI 製品の選定、(2) お客様のアプリケーションの設計、検証、試験、(3) お客様のアプリケーションに該当する各種規格や、その他のあらゆる安全性、セキュリティ、規制、または他の要件への確実な適合に関する責任を、お客様のみが単独で負うものとします。

上記の各種リソースは、予告なく変更される可能性があります。これらのリソースは、リソースで説明されている TI 製品を使用するアプリケーションの開発の目的でのみ、TI はその使用をお客様に許諾します。これらのリソースに関して、他の目的で複製することや掲載することは禁止されています。TI や第三者の知的財産権のライセンスが付与されている訳ではありません。お客様は、これらのリソースを自身で使用した結果発生するあらゆる申し立て、損害、費用、損失、責任について、TI およびその代理人を完全に補償するものとし、TI は一切の責任を拒否します。

TI の製品は、TI の販売条件 、 TI の総合的な品質ガイドライン 、 ti.com または TI 製品などに関連して提供される他の適用条件に従い提供されます。TI がこれらのリソースを提供することは、適用される TI の保証または他の保証の放棄の拡大や変更を意味するものではありません。 TI がカスタム、またはカスタマー仕様として明示的に指定していない限り、TI の製品は標準的なカタログに掲載される汎用機器です。

お客様がいかなる追加条項または代替条項を提案する場合も、TI はそれらに異議を唱え、拒否します。

最終更新日：2025 年 10 月