



ES6550DHAT-ALSA Driver v1.2.0

User Guide

Contents

History.....	3
1 Introduction	1
2 EV6550DHAT	2
3 Quick Start.....	3
3.1 Quick Install	4
3.2 Test	4
4 Driver Module Installation Details	5
5 Uninstall Procedure	6
6 Raspberry Pi Device Tree	7
6.1 Structure of CMX655D Sound Card	7
7 ALSA Controls	10
7.1 Master Playback Volume	10
7.2 Master Capture Volume	10
7.3 ALC Playback Switch	10
7.4 ALC_Attack Playback Switch	10
7.5 ALC_Gain Playback Volume	11
7.6 ALC_Ratio Playback Switch.....	11
7.7 ALC_Release Playback Switch.....	11
7.8 ALC_Threshold Playback Volume	11
7.9 Cap_HPFS Capture Switch	12
7.10 Cap_SAI Capture Route	12
7.11 Companding_En Switch	12
7.12 Companding_Type Switch	12
7.13 DC_Block Capture Switch	12
7.14 DC_Block Playback Switch	12
7.15 DST.....	13
7.16 DST_EN Playback Switch.....	13
7.17 LOUT_EN Playback Switch	13
7.18 LPF Capture Switch	14
7.19 LPF Playback Switch.....	14
7.20 NG_Attack Capture Switch	14
7.21 NG_Ratio Capture Switch	14
7.22 NG_Release Capture Switch	14
7.23 NG_Threshold Capture Volume.....	15
7.24 Noise_Gate Capture Switch.....	15
7.25 Play_HPFS Playback Switch	16
7.26 Play_SAI Playback Route.....	16
7.27 Pre_Amp Playback Volume.....	16
7.28 SPKR_EN Playback Switch.....	16
7.29 Sidetone Playback Volume	16

7.30	Smooth Playback Switch.....	17
7.31	Soft_Mute Playback Switch	17
8	Application and Additional Information.....	18
8.1	DAPM (Dynamic Audio Power Management)	18
8.2	Click and Pop Management.....	18
8.2.1	Digital Side Tone Source	18
8.2.2	Changing Sample Rate	19
8.3	DKMS (Dynamic Kernel Module Support)	19
8.4	Status Monitoring.....	19
8.4.1	Class-D Over-current.....	19
8.4.2	Class-D Over-temperature	19
9	Troubleshooting	20
9.1	Class-D Amplifier Over-current.....	20
9.2	Class-D Amplifier Over-temperature.....	20
9.3	Device Tree	20
9.4	Software Tools.....	20
9.4.1	DMESG	20
10	Glossary.....	22

History

Version	Changes	Date
1	Initial release	June 2020
2	Section 5: reference to kernel version number removed	May 2021
3	References to RPi5 and 64-bit Bookworm 6.6 added	July 2024

1 Introduction

This document describes a firmware driver and installation method that combines the Raspberry Pi with the EV6550DHAT codec board and enables its use with any audio application that follows the standard Advanced Linux Sound Architecture (ALSA) interface. The current implementation may be used with the Raspberry Pi2, 3 or 4. At time of writing GCC is not available for ARM64 for Pi5 so the ALSA driver could not be tested on a Pi5.

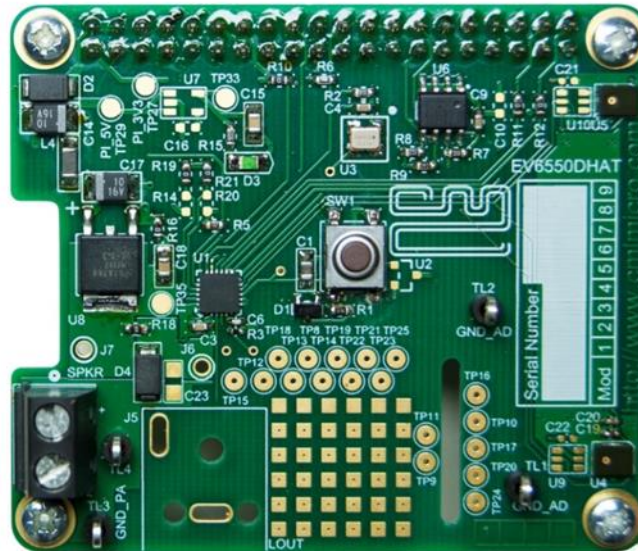


Figure 1 - EV6550DHAT

2 EV6550DHAT

The EV6550DHAT supports the use of the CMX655D Ultra-low Power Voice Codec enabling easy demonstration of the CMX655D and providing improved low power audio capabilities to the Raspberry Pi. This is achieved using common hardware and open-source software, simplifying its use and simplifying software and hardware development.

The CMX655D Ultra-low Power Voice Codec integrates two matched channels supporting a large variety of digital MEMS (Micro-Electromechanical System) microphones and signal processing capability along with a 1 Watt high efficiency Class D speaker amplifier.

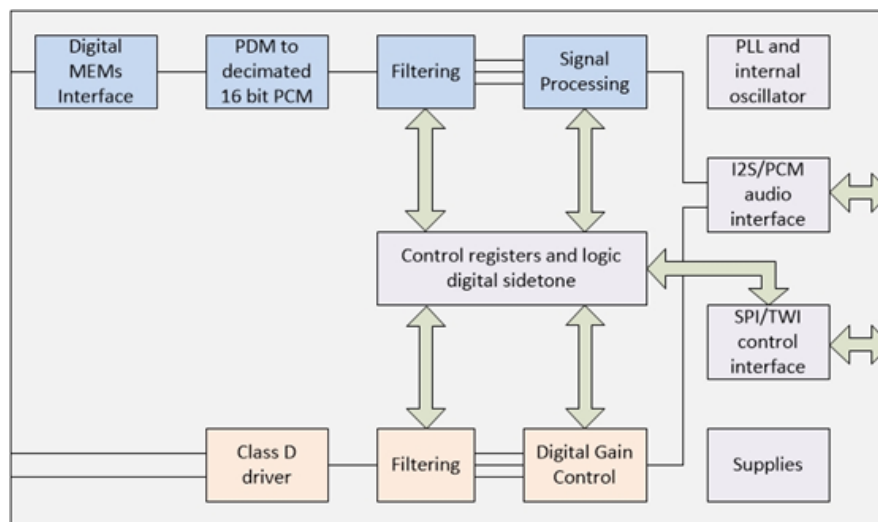


Figure 2 - EV6550DHAT Functional Block Diagram

Warnings

The pre-released EV6550DHAT Raspberry Pi GUI and source code that is available from CML is currently incompatible with the ES6550DHAT driver described by this document and its use will result in unintended results.

The ES6550DHAT driver is written without any knowledge of the EV6550DHAT Reset Button. It is rendered unnecessary with this driver. Should the reset button be depressed during use the driver and therefore the Raspberry Pi will have no knowledge of the new state of the CMX655D codec device. To correct this and enter a known state will require the Raspberry Pi to be restarted.

3 Quick Start

The EV6550DHAT follows the Raspberry Pi Foundation's common HAT format so the card will plug into a Raspberry Pi needing no modification. It is compatible with any Raspberry Pi with a 40-pin extended GPIO connector. With the Raspberry Pi in an unpowered state, the EV6550DHAT can be mounted by pushing the board onto the 40-way GPIO connector. Additionally the EV6550DHAT is supplied with 4 metal standoffs and 4 screws that may be used to support the board via 4 mounting holes positioned to fit the Raspberry Pi board's holes.

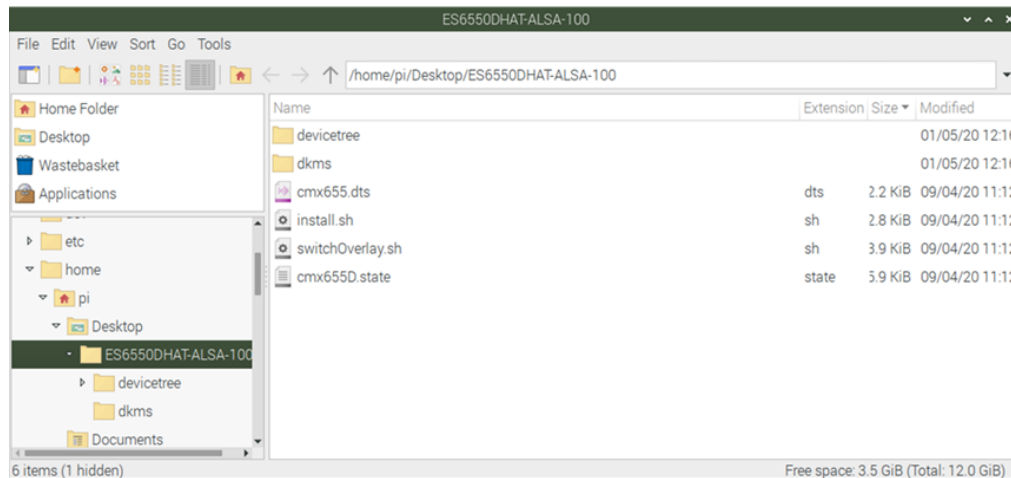


Figure 3 - Files Supplied

To install the EV6550DHAT ALSA driver the following are required:

- EV6550DHAT hardware
- dkms
- sudo
- raspberrypi-kernel-headers
- Latest kernel version.

As this install process involves using the kernel headers it is strongly advised to update the system to ensure that the kernel headers match the running kernel. To do this run (with elevated permissions):

```
apt update && apt upgrade
```

Reboot the Pi to ensure that the latest version of the kernel gets used.

To install dkms, raspberrypi-kernel-headers and sudo, run (with elevated permissions):

```
sudo apt install raspberrypi-kernel-headers dkms
```

It is likely that sudo is already installed on the Raspberry Pi's operating system.

3.1 Quick Install

Download and unpack the driver files to a suitable location in the user's home area. The instructions that follow will use `~/EV6550DHATAlsa`. First change directory to this folder, e.g. `'cd ~/EV6550DHATAlsa'`. Create the `EV6550DHATAlsa` in the home area if necessary. Ensure that the install scripts have execute permissions by running `'chmod +x install.sh switchOverlay.sh'`.

Then run the install script `'./install.sh'`. This script should be run with standard user permission, the script gains elevated permission, where required, via the use of `sudo`.

Reboot the Raspberry Pi. It will be necessary to run `'./install.sh --state'` again to get the default settings for the new soundcard.

Note: Should `'apt update && apt upgrade'` be performed at a later date the EV6550DHAT ALSA driver will require reinstalling.

3.2 Test

To test the ALSA driver connect a 4 to 8Ω speaker to the SPKR (J2) terminal and play an audio file, e.g. `'aplay -D hw:CMX655D <filename.wav>'`. Alternatively a program such as Audacity could be used.

To use Audacity first start the program. Then select the CMX655D sound card as shown in Figure 4.



Figure 4 - Audacity Sound Card Selection

4 Driver Module Installation Details

The install script does three operations:

1. Builds and installs the CMX655-alsa kernel module via dkms.
2. Adds the EV6550DHAT device tree blob 'cmx655D.dtbo' to overlays and enables it.
 - a. This includes disabling the use of the device tree blob overlay located on the EEPROM on the EV6550DHAT.
3. Adds a default setup to ALSA asound.state file.
 - a. If the CMX655D sound card exists it will set the current state to the default.

To run one or more parts of the install process use a combination of the following arguments:

--dkms: Build and install kernel module.
--dt: Install and enable device tree overlay.
--status: Add or update sound card settings.

Supplying no arguments is the same as supplying all arguments. i.e. './install.sh' = './install.sh --dkms --dt --status'. To install without changing the ALSA settings './install.sh --dkms --dt' could be used.

5 Uninstall Procedure

There is no script to uninstall this package included in the release bundle. If uninstallation is required proceed as follows:

1. Remove dkms module: `'sudo dkms remove cmx655-alsa/1.0.0 --all'`
 - a. Note: the version number may be different
2. Remove and disable device tree blob:
 - a. `sudo rm /boot/overlays/cmx655D.dtbo`
 - b. Modify `/boot/config.txt`:
 - i. Remove `'dtoverlay=cmx655D'` to remove loading of local overlay
 - ii. Remove `'dtoverlay='` to enable loading of the HAT's overlay
3. Remove ALSA setup, ALSA saves this at shutdown:
 - a. Reboot after the above steps have been conducted.
 - b. Modify `'/var/lib/alsa/asound.state'`:
 - i. Remove `'state.CMX655D'` section.

6 Raspberry Pi Device Tree

The device tree is a tool used by the Raspberry Pi kernel and its firmware to allow for the complex configuration of kernel modules. It allows for configuration that was previously written into kernel modules to be removed from the module and added at a higher level. This concept allows for one module driver to support multiple situations.

For example, instead of a device driver using macros to define the location of a peripheral, that would then get hard-coded when built, it can get passed a handle by the driver for the peripheral. It is the job of the device tree to define the connection of peripheral driver to the device driver.

The EV6550DHAT ALSA driver contains a device tree configuration that is installed by the install script, see the -dt option for install script in section 4. This device tree configuration constructs a CMX655D sound card as described in section 6.1 below. The device tree configuration included with the EV6550DHAT ALSA driver is written to use the EV6550DHAT with the Raspberry Pi. The CMX655 codec driver should function on other Linux-based platforms however a different device tree configuration may be required.

6.1 Structure of CMX655D Sound Card

The CMX655D sound card is written using ALSA's (Advance Linux Sound Architecture) SoC (System on Chip) layer. This layer builds a sound card from 3 module blocks:

- Codec driver – Controls codec and produces a DAI.
- Platform driver – Controls the platform and produces a DAI.
- Machine driver – Connects the codec and platform drivers together

In the case of the CMX655D sound card:

- The codec driver was written by CML to control the CMX655D via an I²C port. This driver produces a DAI (Digital Audio Interface) that allows the machine driver to set the required parameters such as the CMX655D SAI (Serial Audio Interface) format and sample rate. This codec driver also exposes the majority of the CMX655D features via ALSA controls. This driver is referred to as the CMX655 codec driver in this document.
- The platform driver is the Raspberry Pi's I²S port driver; this is part of the Linux kernel used by the Raspberry Pi. Its job is to setup the Pi's I²S as requested by the machine driver. It also transfers data between the I²S port (connected to the CMX655D's SAI) and the kernel.
- The machine driver is 'simple-card' - it uses the capability of the CMX655 codec driver and I²S platform driver to determine the capability of the sound card (a logical AND of the 2 drivers). The concept is to produce a sound card that looks like a sound card that was written without the ASoC layer.

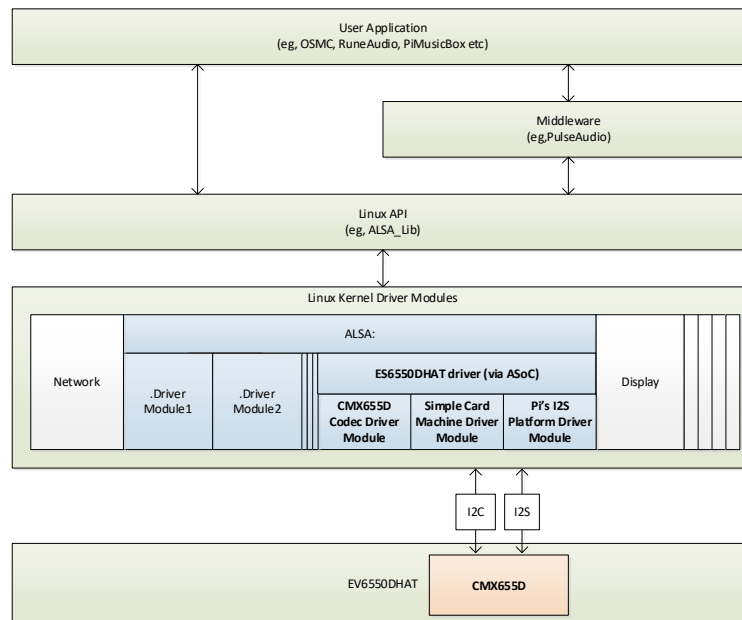


Figure 5 - ES6550DHAT Sound Card

The above listed modules, Codec, machine and Platform are each independent kernel modules with no interconnect defined by themselves. The modules are configured and connected by the device tree.

The CMX655 codec driver requires a connection to an I²C port. This is done by creating a sub-node attached to the Raspberry Pi's I²C device tree node. In the fragment below the sub-node is given the handle 'codec'.

```
target = <&i2c>;
__overlay__ {
    codec: cmx655 {
        reg = <0x54>;
        compatible = "cml,cmx655D";
    };
};
```

The Raspberry Pi's I²S port platform driver is defined by the Raspberry Pi's device tree, this creates a node with the handle 'i2s'.

The simple-card is configured to use the CMX655 driver as the codec and the I²S port as the CPU.

```
target = <&sound>;
__overlay__ {
    compatible = "simple-audio-card";
    simple-audio-card, name = "CMX655D";
    status = "okay";

    simple-audio-card, dai-link@0 {
        format = "i2s";
        bitclock-master = <&codec_dai>;
        frame-master = <&codec_dai>;

        cpu_dai: cpu {
            sound-dai = <&i2s>;
        };
        codec_dai: codec {
            sound-dai = <&codec>;
        };
    };
};
```

The CMX655 codec driver uses the resetN and IRQN pins of CMX655, if the required connections are given. For the EV6550DHAT to operate correctly the resetN line is required as the Raspberry Pi's GPIOs control the level of the resetN. The EV6550DHAT also configures the IRQN line so the CMX655 codec driver can detect over-temperature or over-current events. This is done with the following lines in the CMX655 sub-node:

```
    codec: cmx655 {
        ...
        reset-gpios = <&gpio 24 1>;
        interrupt-parent = <&gpio>;
        interrupts = <25 0x2>; /* falling edge */
        interrupt-names = "irq";
        cmx655, classd-oc-reset-attempts = <5>;
    };

};
```

The 'cmx655, classd-oc-reset-attempts' settings is optional it defaults to 5. See section 9.1 for details on over-current monitoring.

The device tree also allows the function of the Raspberry Pi's GPIOs to be configured. This is done as a sub-node of the GPIO device tree node.

```
target = <&gpio>;
__overlay__ {
    ev6550DHAT_pins: cmx655_pins {
        // Pins resetN, IRQn
        brcm,pins = <24 25>;
        // Out, In
        brcm,function = <1 0>;
        // No pull, pull up
        brcm,pull = <0 2>;
    };
};
```

The function of the I²C and I²S pins is configured by the I2C and I2S device tree nodes respectively.

7 ALSA Controls

7.1 Master Playback Volume

A mono control that sets playback volume. It maps to the CMX655D Playback Volume Register, VOL (\$2A, b0..6).

The volume ranges from -91dB to 0dB in 1dB steps, where -91dB is mute.

The control is mapped as follows:

-91dB	0
-90dB	1
...	
0dB	91

7.2 Master Capture Volume

A stereo control that sets the record level. It maps to the GL and GR controls in the CMX655D Record Level Control Register, LEVEL (\$0F, b4..7 and b0..3) where GL is the left channel and GR is the right channel.

The record level ranges from -12dB to +3dB in 1dB steps.

This control is mapped as follows:

-12dB	0
-11dB	1
...	
+3dB	15

7.3 ALC Playback Switch

A mono control that enables/disables the CMX655D automatic level control block.

This control maps to the EN bit in the CMX655D ALC Control Register, ALCCTRL (\$2B,b7). The mapping is non-inverted.

7.4 ALC_Attack Playback Switch

A mono control that selects the automatic level control attack time. It maps to the ATTACK control in the ALC Time Register, ALCTIME (\$2C, b4..6).

The control is mapped as follows:

1.5ms	0
3ms	1
4.5ms	2
6ms	3
12ms	4
24ms	5
48ms	6
96ms	7

7.5 ALC_Gain Playback Volume

A mono control that sets automatic level control make-up gain. It maps to the GAIN control in the ALC Make-up Gain Register, ALCGAIN (\$2D, b0-3).

The control is mapped as follows:

+0dB	0
+1dB	1
...	
+12dB	12

7.6 ALC_Ratio Playback Switch

A mono control that sets the automatic level control compression ratio. It maps to the RATIO control in the ALC Control Register ALCTRL (\$2B, b5..6).

The control is mapped as follows:

1.5:1	0
2:1	1
4:1	2
Inf:1	3

7.7 ALC_Release Playback Switch

A mono control that sets the automatic level control release time. It maps to the RELEASE control in the ALC Time Register, ALCTIME (\$2C, b0..2).

The control is mapped as follows:

0.06s	0
0.12s	1
0.24s	2
0.48s	3
0.96s	4
1.92s	5
3.84s	6
7.68s	7

7.8 ALC_Threshold Playback Volume

A mono control that sets the automatic level control threshold level. It maps to the THRESH control in the ALC Control Register, ALCCTRL (\$2B, b0..4).

The control is mapped as follows:

-31dB	0
-30dB	1
...	
0dB	31

7.9 Cap_HPFP Capture Switch

A mono control that sets the record high pass filter bandwidth. It maps to the HPSEL control in the Record Voice Filters Register, RVF (\$0C, b0..1).

The control is mapped as follows:

Disabled	0
SRate/320	1
SRate/53.3	2
SRate/26.7	3

7.10 Cap_SAI Capture Route

A mono control that sets the source of data for the SAI record data. It maps to the MIC control in the Serial Audio Interface Mux Register, SAIMUX (\$0A, b0..1).

The control is mapped as follows:

Normal	0
Swapped	1
Left Only	2
Right Only	3

7.11 Companding_En Switch

The ES6550DHAT-ALSA Driver does not support 8-bit audio. 8-bit companding is therefore not supported by the EV6550DHAT.

7.12 Companding_Type Switch

The ES6550DHAT-ALSA Driver does not support 8-bit audio. 8-bit companding is therefore not supported by the EV6550DHAT.

7.13 DC_Block Capture Switch

A mono control that sets the state of the DC blocking filter in the record path. The control maps to the DCBEN bit in the Record Voice Filters Register, RVF (\$0C, b2). The mapping is non-inverted.

7.14 DC_Block Playback Switch

A mono control that sets the state of the DC blocking filter in the playback path. The control maps to the DCBEN bit in the Playback Voice Filter's Register, PVF (\$28, b2). The mapping is non-inverted.

7.15 DST

A mono control that sets the source of the digital side tone. It maps to the DSTS control in the Digital Sidetone Control Register, DST (\$2F, b5..6).

The control is mapped as follows:

Left	0
Right	1
Mean	2
-	3

7.16 DST_EN Playback Switch

A mono control that sets the enable state of the digital side tone. It maps to the EN bit in the Digital Sidetone Control Register, DST (\$2F, b7). The mapping is non-inverted.

7.17 LOUT_EN Playback Switch

The is a virtual switch that allows the power management to disable the line out block if it is set to 0. The result is that the control maps to the LOUT bit in the System Control Register, SYSCTRL (\$32, b4). The

CMX655D control will not be set if there is no requirement for the line out to be activate. See DAPM (section 8.1) for more details.

7.18 LPF Capture Switch

A mono control that sets the state of the low pass filter in the record path. The control maps to the LPFEN bit in the Record Voice Filters Register, RVF (\$0C, b3). The mapping is non-inverted.

7.19 LPF Playback Switch

A mono control that sets the state of the low pass filter in the playback path. The control maps to the LPFEN bit in the Playback Voice Filters Register, PVF (\$28, b3). The mapping is non-inverted.

7.20 NG_Attack Capture Switch

A mono control that selects the noise gate attack time. It maps to the ATTACK control in the Noise Gate Time Register, NGTIME (\$1D, b4..6).

The control is mapped as follows:

1.5ms	0
3ms	1
4.5ms	2
6ms	3
12ms	4
24ms	5
48ms	6
96ms	7

7.21 NG_Ratio Capture Switch

A mono control that sets the noise gate ratio. It maps to the RATIO control in the Noise gate Control Register, NGCTRL (\$1C,b5..6).

The control is mapped as follows:

1:2	0
1:3	1
1:4	2
-	3

7.22 NG_Release Capture Switch

A mono control that sets the noise gate release time. It maps to the RELEASE control in the Noise Gate Time Register, NGTIME (\$1D, b0..2).

The control is mapped as follows:

0.06s	0
0.12s	1
0.24s	2
0.48s	3
0.96s	4
1.92s	5
3.84s	6
7.68s	7

7.23 NG_Threshold Capture Volume

A mono control that sets the noise gate threshold level. It maps to the THRESH control in the Noise Gate Control Register, NGCTRL (\$1C, b0..4).

The control is mapped as follows:

-63dB	0
-62dB	1
...	
-32dB-	63

7.24 Noise_Gate Capture Switch

A mono control that enables/disables the noise gate block. This control maps to the EN bit in the Noise gate Control Register, NGCTRL (\$1C, b7). The mapping is non-inverted.

7.25 Play_HPFP Playback Switch

A mono control that sets the playback high pass filter bandwidth. It maps to the HPSEL control in the Playback Voice Filters Register, PVF (\$28, b0..1). register (bits 0-1).

The control is mapped as follows:

Disabled	0
SRate/320	1
SRate/53.3	2
SRate/26.7	3

7.26 Play_SAI Playback Route

A mono control that sets the SAI data to use in the playback path. It maps to the AMP control in the Serial Audio Interface Mux Register, SAIMUX (\$0A, b2..3).

The control is mapped as follows:

Left	0
Right	1
Mean	2
-	3

7.27 Pre_Amp Playback Volume

A mono control that sets the level of the playback pre-amp gain. It maps to the PRE control in the Playback Preamp Gain Register, PREAMP (\$29, 1..0) register.

The control is mapped as follows:

0dB	0
6dB	1
12dB	2
18dB	3

7.28 SPKR_EN Playback Switch

This is a virtual switch that allows the power management to disable the power amplifier block if it is set to 0.

The result is that this control maps to the PAMP bit in the System Control Register, SYSCTRL (\$32, b3).

The CMX655D control will not be set if there is no requirement for the power amplifier to be activate. See DAPM (section 8.1) for more details.

7.29 Sidetone Playback Volume

A mono control that sets the digital side tone level. It maps to the DSTG control in the Digital Sidetone Control register, DST (\$2F, b0..4).

The control is mapped as follows:

-62dB	0
-60dB	1
...	
-0dB	32

7.30 Smooth Playback Switch

A mono control that enables/disables playback volume smoothing. It maps to the SMOOTH bit in the Playback Volume Register, VOLUME (\$2A, b7). The mapping is non-inverted.

7.31 Soft_Mute Playback Switch

A mono control that enables/disables playback volume soft mute. It maps to the SOFTM bit in the Click and Pop reduction Register, CPR (\$30,b0). The mapping is non-inverted.

8 Application and Additional Information

8.1 DAPM (Dynamic Audio Power Management)

The Raspberry Pi ASoC layer includes a DAPM module. The goal of this module is to keep a sound card in the lowest possible power state. This is done by disabling functional blocks that are not required. Each ASoC driver defines a DAPM map that the DAPM module uses to determine what functional blocks are required. The figure below shows the DAPM map for the CMX655 codec driver.

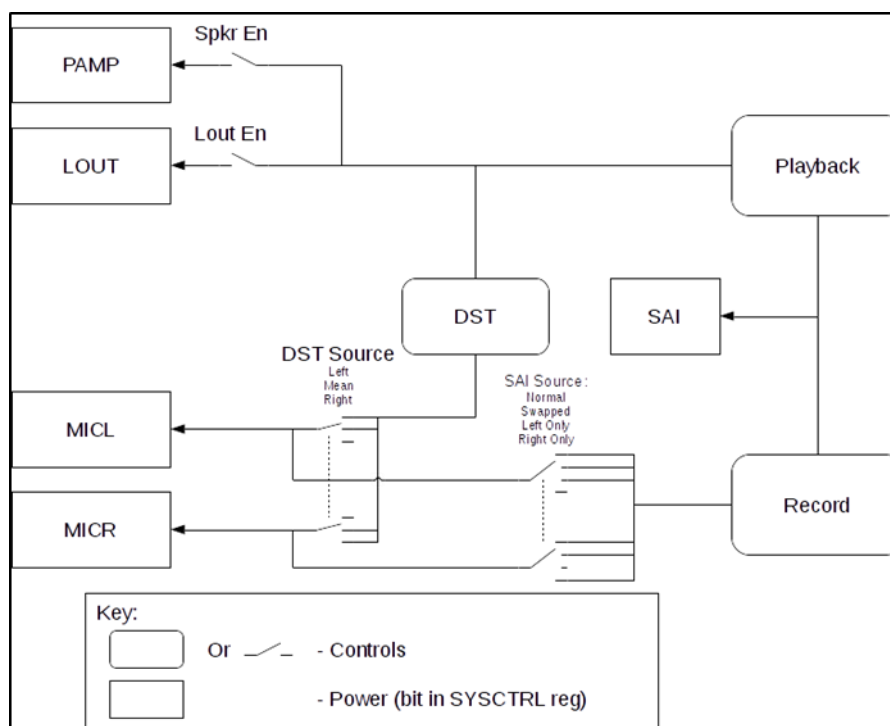


Figure 6 - DAPM Map

The map above shows what effect the controls have on the power requirements for a specific CMX655D functional block.

For example, if the DST is enabled, the source is set to Left and the LOUT switch is enabled, then the MICL and LOUT block require power. In addition if all other controls are disabled then PAMP, MICR and SAI can be safely disabled.

8.2 Click and Pop Management

A side effect of DAPM is that click or pop can be produced by changing power states of functional blocks during capture or playback. This section lists the known causes of click or pops and how to avoid them.

8.2.1 Digital Side Tone Source

Changing the source of the digital side tone can cause unwanted audible artefacts as the power requirements for the microphones changes. To avoid creating these artefacts ensure the digital side tone, DST is disabled while changing the source. For example, to change from left microphone to right first set DST_EN to 0, change the DST to Right then set DST_EN to 1.

8.2.2 Changing Sample Rate

It is not possible to change the sample rate of the codec whilst record or playback are enabled and active; however it is possible to change the sample rate while the digital side tone is enabled. If the sample rate is changed while the side tone is active then unwanted audible artefacts may be observed, this is because the CMX655D clock should be inactive when the sample rate is modified. To avoid unwanted effects disable the side tone while the sample rate is changed.

Additionally it is possible to minimise the artefacts by enabling the volume smoothing (Smooth Playback Switch). Some applications, such as Audacity, set each of the possible sample rates in turn when they start. If the side tone is enabled and volume smoothing is disabled this selecting of rate can amplify the artefacts heard.

8.3 DKMS (Dynamic Kernel Module Support)

As the CMX655 kernel module is not part of the official Raspberry Pi kernel image, manual installation is required. As part of the firmware provided by CML, support for DKMS is provided. DKMS makes the use of additional kernel modules easier as it defines a framework within which each module can be built for each kernel on a system. If a module is registered with DKMS, the DKMS program will copy the required source code into the system. This allows DKMS to rebuild the module when the kernel is updated.

8.4 Status Monitoring

The CMX655 codec driver uses the IRQN line to monitor some status events. This section describes the events monitored and the actions taken if the event occurs.

8.4.1 Class-D Over-current

The CMX655D differential power outputs of the Class-D amplifier are protected against overload current conditions with an automatic shut-off protection circuit. The CMX655D codec driver monitors the over-current status of the amplifier. If an over-current event is detected a kernel warning is produced. See section 9.4.1 for details on viewing kernel messages. The CMX655D will automatically disable the power amplifier. The CMX655 codec driver will attempt to re-enable the power amplifier a set number of times (default 5) after this a kernel error message is produced and the power amplifier is left disabled. The CMX655 codec driver's counter will be reset when the codec is setup, triggered by sound card open.

The number of restarts attempted can be configured via the device tree. The value of the CMX655 sub-node property 'cmx655D,classd-oc-reset-attempts' defines the number of restarts. See section 6 for information on how to use the device tree. If this value is 0 no restart attempts will be made. If this value is >10000 the restart limit will be disabled, the CMX655 codec driver will restart the power amplifier every time an over-current is detected.

8.4.2 Class-D Over-temperature

The CMX655 codec driver monitors the over-temperature status of the CMX655D. If an over-temperature event is detected a kernel error message is produced. See section 9.4.1 for details on viewing kernel messages. The CMX655D will automatically disable the power amplifier. The CMX655 codec driver does not re-enable the power amplifier.

9 Troubleshooting

9.1 Class-D Amplifier Over-current

The CMX655D codec driver monitors the over-current status of the CMX655D Amplifier. If an over-current event is detected a kernel error warning is produced. See section 9.4.1 for details on viewing kernel messages. The CMX655D will automatically disable the power amplifier. The CMX655D codec driver will attempt to re-enable the power amplifier a preconfigured number of times (default 5) after this period a kernel error message is produced and the power amplifier left disabled.

The amplifier can be re-enabled by toggling the SPKR_EN control. The CMX655 codec driver's counter will get reset when the codec is setup. To trigger this, restart any application that has the sound card open.

9.2 Class-D Amplifier Over-temperature

The CMX655D includes an integrated thermal detection circuit which automatically powers-down the Class-D amplifier differential power outputs when the device temperature reaches a critical level. The CMX655 codec driver monitors the over-temperature status of the CMX655D. If an over-temperature event is detected a kernel error message is produced. See section 9.4.1 for details on viewing kernel messages. The CMX655D will automatically disable the power amplifier. The CMX655 codec driver does not re-enable the power amplifier. The power amplifier can be re-enabled by toggling the SPKR_EN control.

9.3 Device Tree

The currently active device tree is represented by a folder structure located at `‘/proc/device-tree’`. All paths in this section are relative to this folder.

For the EV6550DHAT to work it requires the HAT hardware and sound card to be configured.

Hardware:

Ensure the hat folder exists and `‘cat hat/product’` produces the output: EV6550DHAT. If this this is not the case then it is likely that the problem is related to the hardware including connection problems.

Sound card:

Ensure the folder `‘soc/i2c@7e804000/cmx655’` exists. The number after the `i2c@` may differ on different versions of the Raspberry Pi.

It is also possible to view the current device tree as a source format by using the `dtc` command to convert the filesystem representation to device tree source. i.e. `‘dtc -l fs -O dts /proc/device-tree | less’`

9.4 Software Tools

This section contains information on tools that can be used to aid debugging each of the module blocks of the EV6550DHAT sound card driver. It is split into sections based on the tool.

9.4.1 DMESG

DMESG is a useful tool for kernel debugging. It allows access to kernel messages. For example the CMX655 codec driver monitors the over-current and over-temperature status events, if either of these events are detected the driver will produce a kernel message.

To use `dmesg` simply type `dmesg` into any terminal and press enter.

If the EV6550DHAT sound card is installed the output should contain a line similar to `'asoc-simple-card soc:sound: cmx655.1-0054 <-> 3f203000.i2s mapping ok'`.

If required the CMX655 codec driver can be built to produce debug messages. To do this add `'#define DEBUG'` to the top of the `cmx655.c` file, then re-install (run `./install.sh` script again). The kernel parameter `debug` is also required, to set this on a Raspberry Pi add `'debug'` to the line found in `/boot/cmdline.txt`. Reboot the pi and run `dmesg` again.

If attempting to debug a problem triggered by an action running `'dmesg --follow'` may prove useful.

DKMS

`dkms` can be used to identify the status of the kernel module that are not part of the kernel code. To ensure that the `cmx655` kernel driver is installed correctly run `'dkms status'` check for a line similar to `'cmx655-alsa, 1.0.0, 4.19.97-v7+, armv7l: installed'`.

10 Glossary

ALC	– Automatic Level Control
ALSA	– Advance Linux Sound Architecture
ASoC	– ALSA System on Chip
DAI	– Digital Audio Interface
DAPM	– Dynamic Audio Power Management
DKMS	– Dynamic Kernel Module Support
GPIO	– General Purpose Input Output
IC	– Integrated Circuits
I ² C	– Inter-Integrated Circuit
I ² S	– Inter-IC Sound
SAI	– Serial Audio Interface

CML does not assume any responsibility for the use of any algorithms, methods or circuitry described. No IPR or circuit patent licenses are implied. CML reserves the right at any time without notice to change the said algorithms, methods and circuitry and this product specification. CML has a policy of testing every product shipped using calibrated test equipment to ensure compliance with this product specification. Specific testing of all circuit parameters is not necessarily performed.

 CML Microcircuits	United Kingdom	p: +44 (0) 1621 875500	e: sales@cmlmicro.com techsupport@cmlmicro.com
	Singapore	p: +65 62888129	e: sg.sales@cmlmicro.com sg.techsupport@cmlmicro.com
	United States	p: +1 336 744 5050 800 638 5577	e: us.sales@cmlmicro.com us.techsupport@cmlmicro.com

www.cmlmicro.com