


GSoC Proposal for BeagleBoard.org

BeagleBoard.org
Jul 02, 2025

Table of contents

Chapter 1

Introduction

Bela is an open-source embedded computing platform for creating responsive, real-time interactive systems with audio and sensors. It features ultra-low latency, high-resolution seneor sampling, a convenient and powerful browser-based IDE, and a fully open-source toolchain that includes support for both low-level languages like C/C++ and popular computer music programming languages like Pure Data, SuperCollider and Csound.

The Bela Cape Rev C integrates the ES9080Q 8-channel DAC and TLV320AIC3104 stereo codec, offering high-quality audio I/O. However, the ES9080Q lacks Linux support, limiting the device's capabilities.

1.1 Summary links

- **Contributor:** Jian De
- **Mentors:** Giulio Moro
- **Repository:** TBD
- **Weekly Updates:** TBD

1.2 Status

This project is currently just a proposal. Preliminary codebase analysis has identified key areas for improvement, and initial discussions with the community and mentors are underway.

1.3 Proposal

- Created accounts across OpenBeagle, Discord and Beagle Forum
- The PR Request for Cross Compilation: #209
- Created a project proposal using the proposed template

1.4 About

- **Forum:** u/jiande
- **OpenBeagle:** jiande
- **Github:** jaydon2020
- **School:** Universiti Teknologi Malaysia

- **Country:** Malaysia
- **Primary language:** English, Mandarin Chinese, Bahasa Maslaysia
- **Typical work hours:** 9AM-5PM Malaysia Time (UTC+08)
- **Previous GSoC participation:** This would be my second time participating in GSOC

Chapter 2

About the Project

Project name: ALSA drivers for ES9080Q and TLV320AIC3104

2.1 Description

The Bela cape rev C comes with a ES9080Q 8-channel DAC and a TLV320AIC3104 stereo codec. The former has no support on Linux, while the latter is well supported. This project involves:

- writing a driver for the ES9080Q, which interacts with existing McASP and DMA drivers
- have the ES9080Q and TLV320AIC3104 show up as a single ALSA device with 2 inputs and 10 outputs. Explore whether this can happen via a simple wrapper in the device tree or requires a dedicated driver.
- If we select the longest duration for the project, then we could add a Xenomai port to the ALSA driver.

2.2 Goals and Objectives

The goal of this project is to upstream support of ES9080Q; simultaneous access to ES9080Q and TLV320AIC3104 via ALSA

This project aims to:

1. Write a Linux driver for ES9080Q Codec
2. Platform drivers interacts with existing McASP and DMA drivers
3. Machine driver ES9080Q and TLV320AIC3104 show up as a single ALSA device
4. Patch and upstream the ES9080Q codec driver
5. Port the ALSA driver to Xenomai for real-time performance.

Chapter 3

Methods

3.1 Linux ALSA Overview

This section is to introduce the Linux ALSA (Advance Linux Sound Architecture) framework.

The ALSA framework is a part of the Linux kernel that is supported and maintained by the Linux community. This makes it feasible to adapt the framework to the BeagleBoard device by designing a driver that utilizes TI's McASP support. ALSA includes a collection of sound card drivers, including actual codec drivers, and can support adding new codec drivers.

3.1.1 ALSA Architecture Overview

Within the kernel there are separate drivers for each component. For each board a sound-card instance is created, usually using the sound {} device tree node, that links together various components like different McASP instances to a codec or an HDMI bridge.

ASoC framework (ALSA System On Chip) (Kernel space):

The aim of the ALSA System on Chip (ASoC) layer is to improve ALSA support for embedded system-on-chip processors and audio codecs. The ASoC framework provides a DMA engine which interfaces with McASP and DMA framework to handle the transfer of audio samples. ASoC also supports the dynamic power management of audio path through the DAPM driver. ASoC acts as an ALSA driver, which splits an embedded audio system into three types of platform independent drivers: the CPU DAI, the codec and the machine drivers.

ALSA core: Provide the API device driver with access to the user application. PCM, CTL, MIDI, TIMER ...

3.2 ASoC drivers (Kernel space)

The major goal for this project is to create a ASoC drivers that have the ES9080Q and TLV320AIC3104 show up as a single ALSA device.

ASoC drivers allow the implementation of hardware dependent code for ASoC driver classes:

- Codec drivers
- Platform drivers
- Machine drivers

3.2.1 Codec Drivers Implementation

TLV320AIC3104 codec driver is well supported. The driver code is under linux kernel sound/soc/codecs/tlv320aic31xx.c

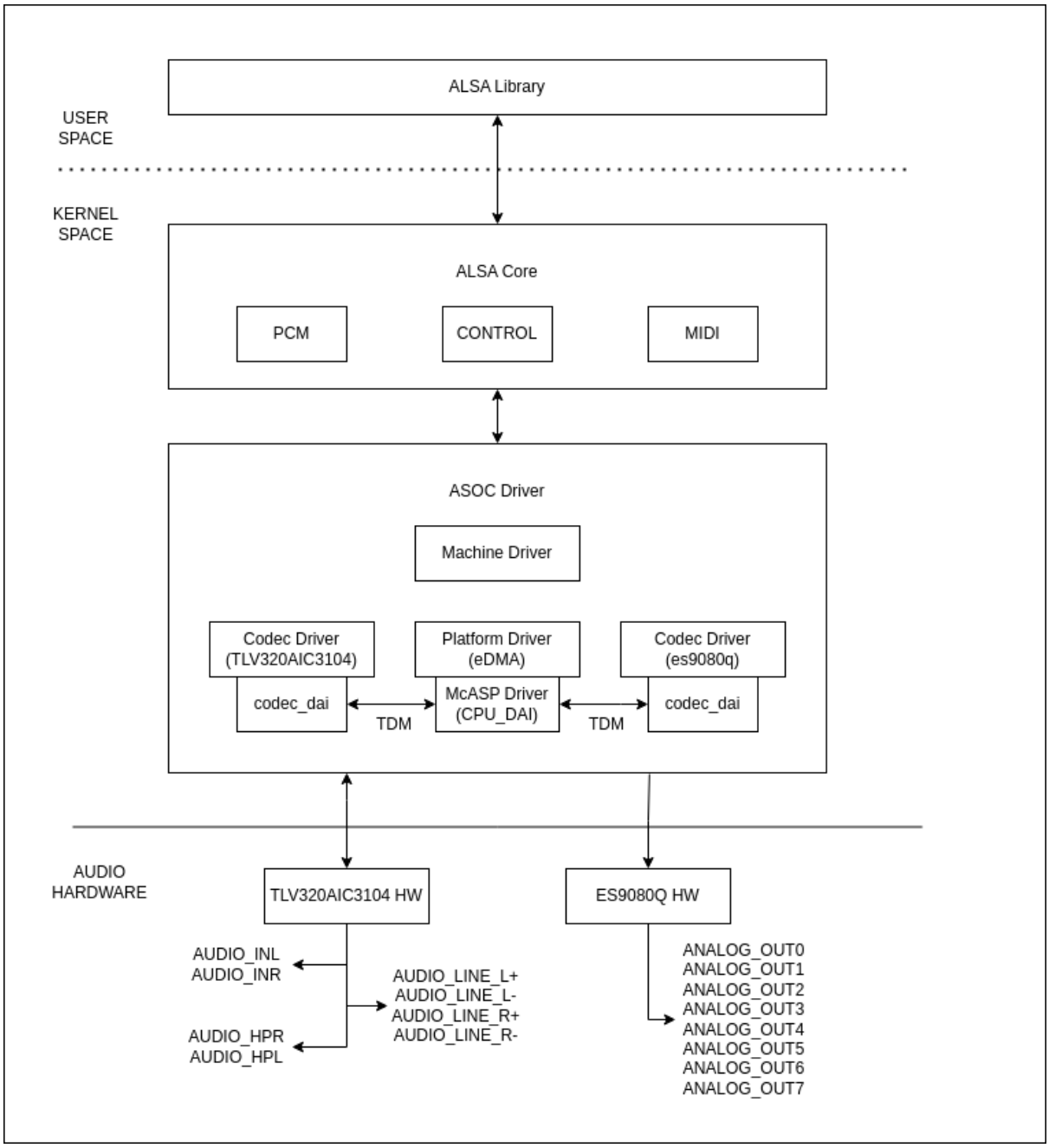


Fig. 1: Figure 1: ALSA Framework Architecture

ES9080Q codec driver has no support on Linux. The primary task for this project is to create a codec driver.

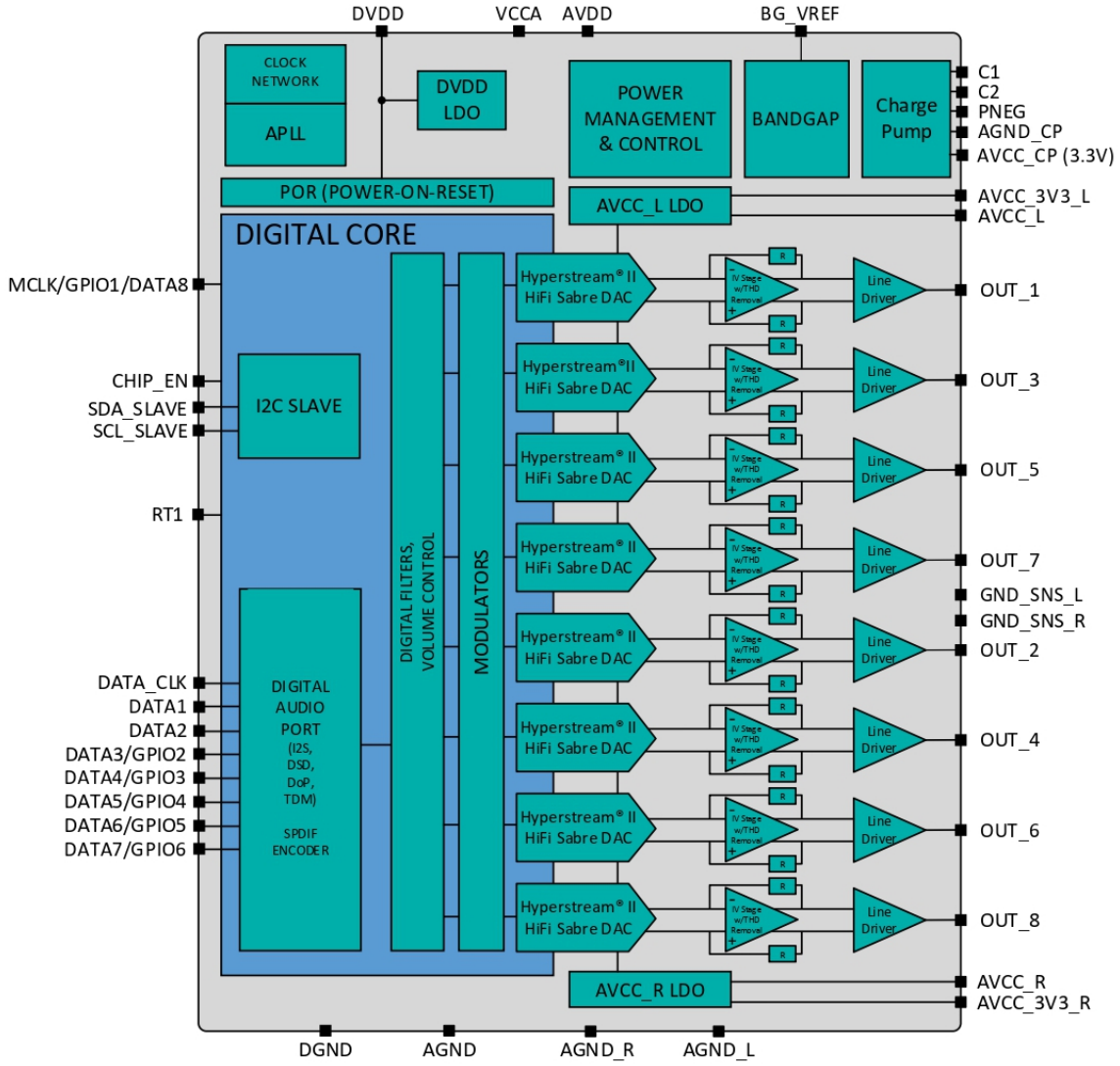


Fig. 2: Figure 2: Function Block ES9080q DAC

The following step is to write a codec ALSA driver:

1. Configuring I2C driver:

The extra thing we need is to define struct *of_device_id*. struct *of_device_id* is defined to match the corresponding node in the *.dts* file:

```
static const struct of_device_id es9080q_of_match[] = {
    { .compatible = "ess,es9080q" },
    {},
};

MODULE_DEVICE_TABLE(of, es9080q_of_match);
```

Defining and registering the I2C driver

```
static struct i2c_driver es9080q_i2c_driver = {
    .driver = {
```

(continues on next page)

(continued from previous page)

```
            .name    = "es9080q",
    },

    .probe          = es9080q_i2c_probe,
    .remove         = es9080q_i2c_remove,
    .id_table       = es9080q_i2c_id,
}
```

The *es9080q_i2c_probe()* function is the driver's entry point. The function first allocates memory for common data and sensor-private data. Under the probe function initialize register map. Example below shows the probe function for tlv320aic3x codec.

```
// Example from tlv320aic31xx.c codec driver
static int aic31xx_i2c_probe(struct i2c_client *i2c)
{
    /* ... other codes */

    aic31xx->regmap = devm_regmap_init_i2c(i2c, &aic31xx_i2c_regmap);

    /* ... other codes */
}
```

2. Configuring DAIs Component:

Analog codec capabilities are declared in the driver's snd_soc_dai_driver structure. Example below shows the DAI capabilities of the tlv320aic3x codec. The DAI link name will be used later in the machine driver's DAI link array to specify what codec DAI is being connected

```
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,
};
```

Important: That implementation is limited to as specific use case. I would use the ESS ES9080Q datasheet as a further source of information.

For es9080q codec driver parameters configuration. I will refer to the driver documentation and the implementation in Bela's code core/Es9080_Codec.cpp

3. Configuring ASoC Component:

ASoC uses a common structure, *snd_soc_component_driver*, which is actually the instance of the codec driver that contains pointers to the codec's routes, widgets, controls, and a set of codec-related function callbacks along with one or more struct snd_soc_dai_driver.

Example below shows the ASoC Component of the tlv320aic3x codec.

```
static const struct snd_soc_component_driver soc_codec_driver_aic31xx = {
    .probe                  = aic31xx_codec_probe,
    .set_jack               = aic31xx_set_jack,
    .set_bias_level         = aic31xx_set_bias_level,
    .controls               = common31xx_snd_controls,
    .num_controls           = ARRAY_SIZE(common31xx_snd_controls),
    .dapm_widgets           = common31xx_dapm_widgets,
    .num_dapm_widgets       = ARRAY_SIZE(common31xx_dapm_widgets),
    .dapm_routes            = common31xx_audio_map,
    .num_dapm_routes        = ARRAY_SIZE(common31xx_audio_map),
    .suspend_bias_off       = 1,
    .idle_bias_on           = 1,
    .use_pmdown_time        = 1,
    .endianness             = 1,
};
```

4. Configuring DAI Operations:

```
static int aic31xx_hw_params(struct snd_pcm_substream *substream,
                struct snd_pcm_hw_params *params,
                struct snd_soc_dai *dai)
{

}

static const struct snd_soc_dai_ops aic31xx_dai_ops = {
    .hw_params      = aic31xx_hw_params,
    .set_sysclk     = aic31xx_set_dai_sysclk,
    .set_fmt        = aic31xx_set_dai_fmt,
    .mute_stream    = aic31xx_dac_mute,
    .no_capture_mute = 1,
};
```

3.2.2 Platform Drivers Implementation

An ASoC platform driver class can be divided into audio DMA drivers, SoC DAI drivers and McASP drivers. The platform drivers only target the SoC CPU and must have no board specific code.

Important: That implementation will utilizes TI's existing drivers

---

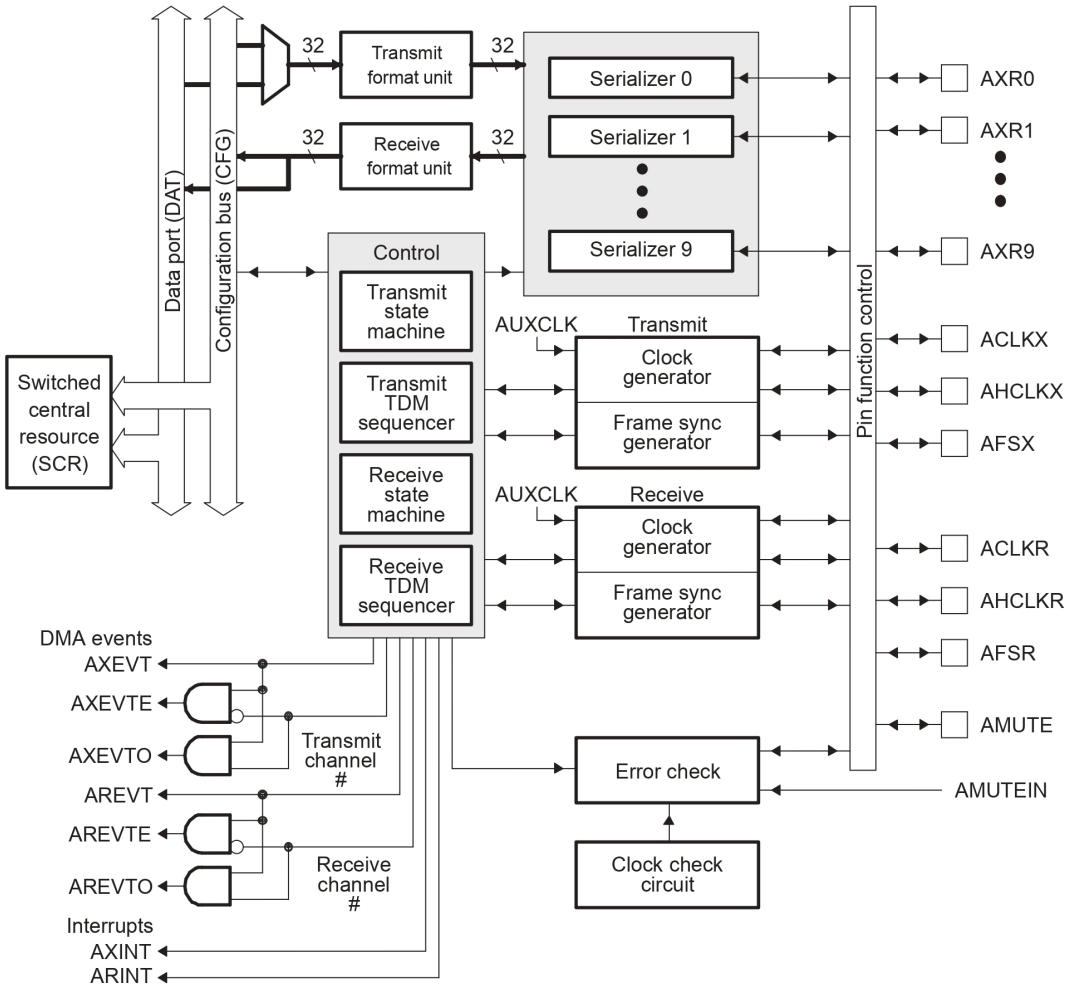
The audio subsystem present on various TI SoCs consists of two major components:

- The McASP driver can be found at sound/soc/davinci/davinci-mcasp.c
- eDMA driver which can be found at arch/arm/common/edma.c and drivers/dma/edma.

3.2.3 Machine Drivers Implementation

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.

Note: There's also a line output available on board which is driven by the same digital data as HPL/R. Also, when running alongside the ES9080Q, this is actually running as TDM (not I2S) because the TLV320AIC3104 is generating the bitclock for both codects at 256 clocks per frame. Similarly in the "Sound Card Definition", it should actually



A McAsp has 10 serial data pins.

B AMUTEIN is not a dedicated McASP pin, but typically comes from one of the DSP's external interrupt pins.

Fig. 3: Figure 3: Block Diagram McASP

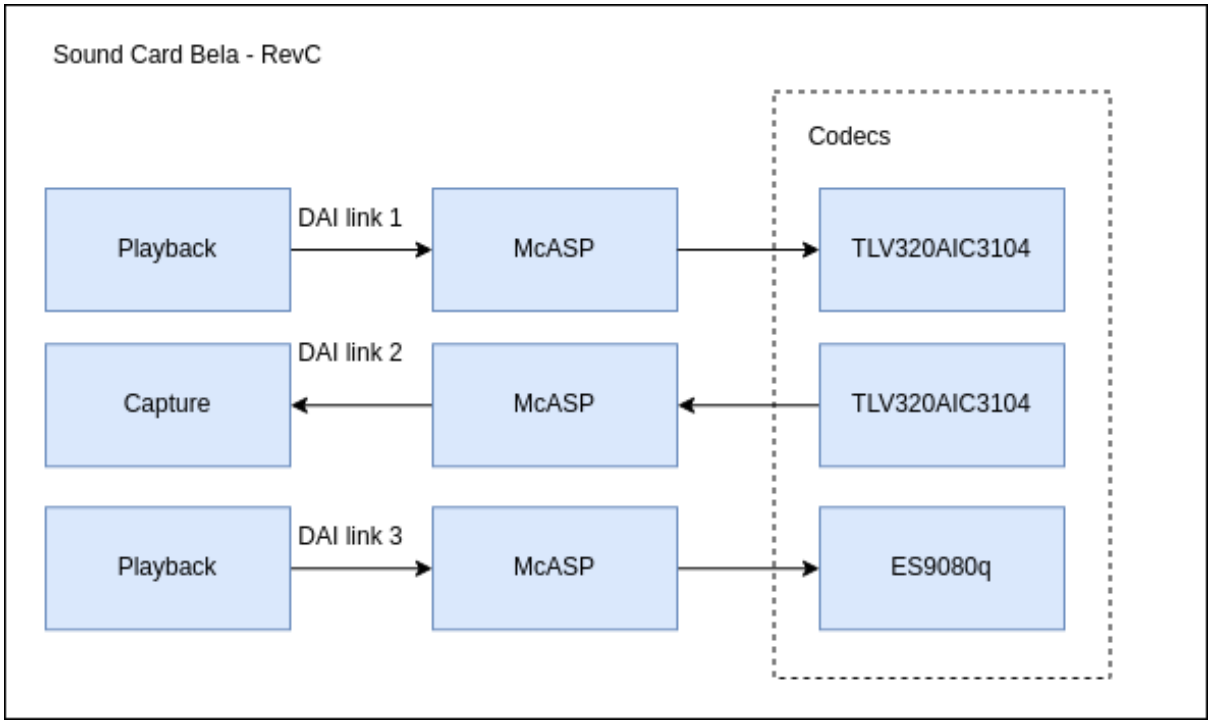


Fig. 4: Figure 4: Sound card schematic

be the tlv320aic310x the one generating the clocks. That's the currently used configuration. Other configurations may be possible with the current wiring, but we should probably start from the one that's already in use on Bela.

TDM Slot Mapping:

- ES9080Q: Uses 8 TDM slots (32-bit each) for 8-channel output.
- TLV320AIC3104: Uses TDM for 2-channel input/output (AUDIO_INL/R and HPL/R).

ALSA PCM Device:

The combined soundcard will appear as a single ALSA PCM device with:

- Playback: 10 channels (8 from ES9080Q + 2 from TLV320AIC3104).
- Capture: 2 channels (from TLV320AIC3104).

1. Soundcard device tree configuration

```
static const struct of_device_id mcasp_dt_ids[] = {
    {
        .compatible = "bela,cape_revC",
        .data = &dm646x_mcasp_pdata,
    },
    { /* sentinel */ }
};
MODULE_DEVICE_TABLE(of, mcasp_dt_ids);
```

Sound Card Definition

```
sound {
    compatible = "simple-audio-card";
    simple-audio-card,name = "Bela-Cape";
```

(continues on next page)

(continued from previous page)

```
    simple-audio-card,format = "i2s";
    simple-audio-card,bitclock-master = <&codec_dai>;
    simple-audio-card,frame-master = <&codec_dai>;

    simple-audio-card,cpu {
        sound-dai = <&mcasp0>;
        system-clock-frequency = <24576000>;
    };

    codec_dai: simple-audio-card,codec@0 {
        sound-dai = <&es9080q>;
        clocks = <&mcasp0_ahclkx_mux>;
        clock-names = "mclk";
        dai-tdm-slot-num = <8>;
        dai-tdm-slot-width = <32>;
    };

    simple-audio-card,codec@1 {
        sound-dai = <&tlv320aic3104>;
        clocks = <&mcasp0_ahclkx_mux>;
        clock-names = "mclk";
        dai-format = "i2s";
    };
};
```

2. Initialize platform driver

The ASoC machine driver connects the codec drivers to a PCM driver by linking the DAIs exposed by each module. It instantiates the sound card (a software component in ASoC architecture).

The structure snd_soc_dai_link, in ASoC core, defines a link that connects two DAIs from different modules. Such a link is called a DAI link. A given machine driver can have one or more of DAI links, which are connected at runtime to form an audio path.

```
static struct platform_driver davinci_mcasp_driver = {
    .probe          = davinci_mcasp_probe,
    .remove         = davinci_mcasp_remove,
    .driver         = {
        .name       = "davinci-mcasp",
        .pm     = &davinci_mcasp_pm_ops,
        .of_match_table = mcasp_dt_ids,
    },
};

module_platform_driver(davinci_mcasp_driver);
```

3.3 Send ALSA driver patch to Linux kernel

https://kernelnewbies.org/FirstKernelPatch

3.4 ALSA Debug or troubleshooting

If one or more audio interfaces are missing, the sound card cannot probe:

- Look for missing audio interfaces in /sys/kernel/debug/asoc/dais.
- Check kernel config, modules and error trace for missing interfaces.

- Check devices not probed in /sys/kernel/debug/devices_deferred.

3.5 Co-Kernel Concept of Xenomai

Xenomai is one of many inux real-time solutions that improve the real-time performance of Linux by adding an RTOS kernel cobalt on top of linux. The combination of real-time kernel cobalt and non-real-time kernel linux can not only provide the hard real-time performance of industrial-grade RTOS, but also take advantage of the excellent network and graphical interface services of the Linux operating system, which has huge advantages in terms of product development cycle and cost control, and the structure is as follows:

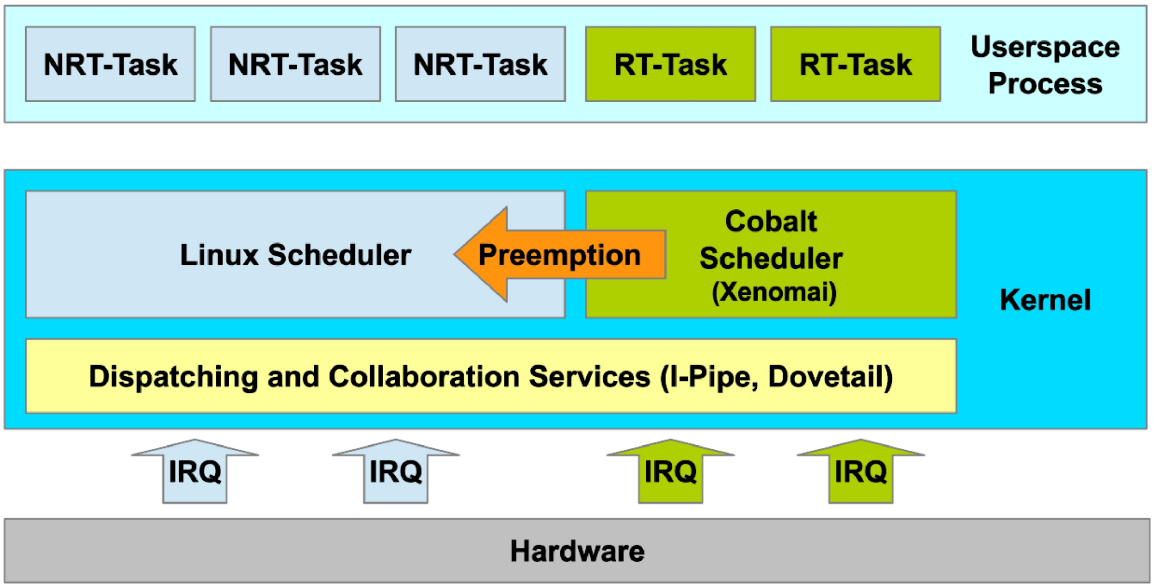


Fig. 5: Figure 5: Xenomai Structure

There are three main parts, and the porting of Xenomai is also built around these three parts:

1. libcobalt, a real-time application library in user space;
2. real-time kernel Cobalt in kernel space;
3. The hardware architecture is related to the Linux kernel version: ipipe-core or dovetail.

3.6 Plan Implementation Alsa driver in Xenomai

Another part for the project is experimanting the ALSA driver integrate with Xenomai's real-time frameworks to achieve ultra-low-latency audio performance for real-time application.

Note: Stretch Goal!

For Xenomai 3, the ALSA SoC driver will be ported as an RTDM-based device driver, prioritizing audio interrupts via Xenomai's RTDM-IRQ API and optimizing DMA transfers using real-time memory pools to minimize jitter. This approach isolates audio processing within Xenomai's real-time domain, shielding it from non-real-time Linux tasks.

For Xenomai 4's EVL core will be tested with minimal driver modifications, leveraging its ability to grant real-time capabilities to unmodified Linux drivers, thereby simplifying integration.

Performance will be benchmarked across three configurations—vanilla Linux, Xenomai 3 (RTDM), and Xenomai 4 (EVL)—using tools like cyclictest and jack_delay to measure worst-case latency, jitter, and throughput. More info can refer Running benchmarks

3.7 Software

- VSCode IDE
- Alsa-mixer
- linux-kernel-v6.12
- xenomai4

3.8 Hardware

- Beaglebone Black
- Beaglebone-ai-64
- Bela Cape Rev C
- Logic Analyzer
- Jtag

Chapter 4

Timeline

4.1 Timeline summary

Date	Activity
Feb 27 - Mar 24	Connect with possible mentors and discuss project ideas
Mar 25 - Apr 8	Proposal review and Submit application
Apr 9 - May 7	*Proposal accepted or rejected (April 9 - May 7)*
May 8 - Jun 1	*Community Bonding Period (May 8 - June 1)*
Jun 2 - Jun 8	*Week #1, (June 2) Configuring Codec Driver I*
Jun 9 - Jun 15	*Week #2, (June 9) Configuring Codec Driver II*
Jun 16 - Jun 22	*Week #3, (June 16) Configuring McASP Driver*
Jun 23 - Jun 29	*Week #4, (June 23) DMA & Machine Driver*
Jun 30 - July 6	*Week #5, (June 30) Testing & Upstream Prep*
July 7 - July 13	*Week #6, (July 7) Upstream Prep*
July 14 - July 20	*Week #7, (July 14) Xenomai 4 Setup*
July 18	*Submit midterm evaluations (July 18)*
July 21 - July 27	*Week #8, (July 21) Xenomai 4/EVL Integration*
July 28 - Aug 3	*Week #9, (July 28) Optimization & Benchmarks*
Aug 4 - Aug 10	*Week #10, (August 4) Optimization & Benchmarks*
Aug 11 - Aug 17	*Week #11, (August 11) Documentation*
Aug 19	*Week #12, (August 19) final project video*

4.2 Timeline detailed

4.2.1 Proposal accepted or rejected (April 9 - May 7)

- Familiarize Linux Kernel Sound Subsystem Documentation [6]
- Familiarize ES9080Q Documentation[2] and understand Bela's code core/Es9080_Codec.cpp[4]
- Familiarize TDM, McASP, eDMA and understand Bela's code core/Tlv320_Es9080_Codec.cpp[4]
- Familiarize Xenomai IRQs and DMA[7][8] and understand Bela's code core/RTAudio.cpp[4]

4.2.2 Community Bonding Period (May 8 - June 1)

- Introduce myself to the organization
- Making an Introductory Video.
- Downloading and Cross-Compiling the Kernel
- Setting up the debugging enviroment with JTag Emulator
- Installing Xenomai Kernel and Libraries

4.2.3 Week #1, (June 2) Configuring Codec Driver I

- Making an Introductory Video.
- Configure device tree for ES9080Q (I2C node, clocks, reset GPIO).
- dd Kconfig/Makefile entries for the driver.
- Implement regmap I2C communication and probe basic functionality.
- Debug with logic analyzer (verify I2C transactions).

4.2.4 Week #2, (June 9) Configuring Codec Driver II

- Define ALSA component (snd_soc_component_driver) with DAPM widgets/routes for 8 outputs.
- Add mixer controls (snd_kcontrol_new) for volume/mute.
- Declare DAI driver (snd_soc_dai_driver) with TDM slot configuration (8 × 32-bit).

4.2.5 Week #3, (June 16) Configuring McASP Driver

- Configure McASP0 in device tree (TDM mode, 8 slots, 32-bit).
- Link McASP to ES9080Q via snd_soc_dai_link.
- Validate clock synchronization (BCLK/WCLK) with oscilloscope.

4.2.6 Week #4, (June 23) DMA & Machine Driver

- Configure eDMA for McASP TX/RX with cyclic buffers.
- Implement machine driver to merge ES9080Q (8ch) and TLV320AIC3104 (2ch) into a single ALSA device.
- Validate 10-channel playback (speaker-test -c 10) and capture (arecord -f S32_LE).

4.2.7 Week #5, (June 30) Testing & Upstream Prep

- Document driver usage, DTS snippets, and debug steps.
- Submit RFC patch to ALSA mailing list for early feedback.

Important: I think it's more 3 weeks doing the ES9080Q and 1 week configuring the three subsystems. Anyhow, on balance it seems reasonable to reach a working ALSA driver by midterm. That would allow enough time for back and forth with the maintainers to make sure by the end of the project the driver is ready to marge.

4.2.8 Week #6, (July 7) Upstream Prep

- Prepare user guide for ALSA driver setup

4.2.9 Week #7, (July 14) Xenomai 4 Setup

- Install Xenomai 4 with EVL core
- Test unmodified ALSA driver under EVL real-time scheduling.

4.2.10 Submit midterm evaluations (July 18)

- Document the progress made during the first phase of the project.

Important: **July 18 - 18:00 UTC:** Midterm evaluation deadline (standard coding period)

4.2.11 Week #8, (July 21) Xenomai 4/EVL Integration

- Tweak EVL thread priorities for audio IRQs and DMA.
- Profile latency with cyclictest and latency_histogram.

4.2.12 Week #9, (July 28) Optimization & Benchmarks

- Tweak EVL thread priorities for audio IRQs and DMA.
- Compare latency/jitter between vanilla Linux and Xenomai 4.

4.2.13 Week #10, (August 4) Optimization & Benchmarks

- Prepare performance graphs and analysis.

4.2.14 Week #11, (August 11) Documentation

- Finalize user guide for Bela Cape audio setup. Both Linux and Xenomai
- Create demo video showcasing 10-channel playback and latency benchmarks.

4.2.15 Week #12, (August 19) final project video

- Submit final project video, submit final work to GSoC site and complete final mentor evaluation.

Important: **August 25 - September 1 - 18:00 UTC:** Final week: GSoC contributors submit their final work product and their final mentor evaluation (standard coding period)

September 1 - September 8 - 18:00 UTC: Mentors submit final GSoC contributor evaluations (standard coding period)

4.2.16 Initial results (September 1)

Important: **September 1 - November 9:** GSoC contributors with extended timelines continue coding

November 4 - 18:00 UTC: Final date for all GSoC contributors to submit their final work product and final evaluation

November 11 - 18:00 UTC: Final date for mentors to submit evaluations for GSoC contributor projects with extended deadline

---

Chapter 5

Experience and approach

1. I've worked on projects that combine hardware and software, I built a line following robot with dsPIC30f4011 microcontroller. Solder and design my own hardware with the PIC microcontroller. Understand about GPIO, PWM, UART, ADC/DAC. I have experience in C programming and know how circuit connection with breadboard.
2. I've build a custom PCB to control robotic pneumatic valve with Altium designer.
3. During Google Summer of Code 2022, I worked on improving beagle-config a tui config tools for BeagleBoard devices, learning how Linux systems are configured and how to modify device settings.
4. Rewrite the beagle-config using rust with Ratatui - github
5. During my internship with StarFive, I ported temperature sensor driver (1-wire) to a JH7100's VisionFive on U-Boot - report
6. I'm familiar with embedded linux, low-level driver development and device-tree.

5.1 Contingency

- **If I get stuck on my project and my mentor isn't around, I will use the following resources:-**
    1. **Hardware & Documentation Resources:**
        - BeagleBoard Guides: Reference the BeagleBoard Getting Started Guide for hardware setup, pin configurations, and kernel compilation.
        - Bela Resources: Study Bela's schematics and existing driver code (core/Es9080_Codec.cpp) to understand hardware interactions.
        - TI Documentation: Consult McASP Technical Reference Manual and eDMA guides for DMA/clock configurations.
        - ES9080Q Datasheet: Use the ESS ES9080Q datasheet for register maps and timing diagrams.
        - ALSA Project Wiki: Reference guides for ALSA architecture, PCM interfaces, and driver development.
        - Linux Sound Subsystem Documentation: Official Linux kernel documentation for ALSA driver components.
        - Xenomai Guides: Reference the Xenomai Homepage
    2. **Community & Forums**
        - BeagleBoard Forum: Post questions about hardware issues (e.g., McASP pin conflicts, device tree errors).
        - TI E2E Forum: Seek help for McASP/eDMA configuration challenges.

  - Xenomai Mailing List: Ask for guidance on real-time IRQ/DMA optimizations.

5.2 Benefit

This project will enhance open-source audio hardware support:

- Upstream Support for ES9080Q
- **Merges ES9080Q (8 outputs) and TLV320AIC3104 (2 inputs/2 outputs) into a single ALSA interface with 10 outputs and 2 inputs.**
  - Simplified User Experience
  - Multi-Channel Audio

Enable low-latency applications for audio/music:

- Xenomai support ensures deterministic timing for audio processing

5.3 Misc

- The PR Request for Cross Compilation: #209

Chapter 6

References

1. Texas Instruments, “Interfacing DRA7xx Audio to Analog Codecs.” Application Report, Dec. 2015 [Revised Jan. 2017]. Link
2. Everest Semi, “ES9080 32-bit High-Performance 8-Channel DAC”. Product Datasheet, Dec. 2020 [Revised Apr. 2022], Link
3. Bela.io, “What is Bela? - The Bela Knowledge Base,” Bela.io, 2016. https://learn.bela.io/using-bela/about-bela/what-is-bela/ (accessed Apr. 01, 2025).
4. Bela.io, “BelaPlatform/Bela: Bela: core code, IDE and lots of fun!,” GitHub, https://github.com/BelaPlatform/bela (accessed Apr. 01, 2025).
5. Bela.io, “Bela cape Rev C” GitHub, https://github.com/BelaPlatform/bela-hardware/tree/master/cape/bela_cape_rev_C3 (accessed Apr. 01, 2025).
6. Kernal.org, “Sound Subsystem Documentation — The Linux Kernel documentation,” Kernel.org, 2025. https://www.kernel.org/doc/html/latest/sound/index.html (accessed Apr. 01, 2025).
7. Philippe Gerum , “Xenomai 3 :: Xenomai 3,” Xenomai.org, 2025. https://v3.xenomai.org/ (accessed Apr. 01, 2025).
8. Philippe Gerum , “Xenomai 4 :: Xenomai 4,” Xenomai.org, 2025. https://v4.xenomai.org/ (accessed Apr. 01, 2025).
9. “3.2.2.2. Audio — Processor SDK Linux for AM67A Documentation,” Ti.com, 2024. https://software-dl.ti.com/jacinto7/esd/processor-sdk-linux-am67a/latest/exports/docs/linux/Foundational_Components/Kernel/Kernel_Drivers/Audio.html (accessed Apr. 03, 2025).
10. John Madieu, Mastering Linux Device Driver Development: Write custom device drivers to support computer peripherals in Linux operating systems , Packt Publishing, 2021.
11. “Linux Audio (1): alsa architecture and RK3588 PCM instance - ArnoldLu - □□□,” Cnblogs.com, 2024. https://www.cnblogs.com/arnoldlu/p/18057519 (accessed Apr. 03, 2025).
12. “RTDM Real-Time Drive Interrupt Registration - Muduo - □□□,” Cnblogs.com, 1027. https://www.cnblogs.com/wsg1100/p/18653521#6rtdm%E9%A9%B1%E5%8A%A8%E7%9A%84%E4%B8%AD%E6%96%AD%E6%B3%A8%E5%86%8C%E6%B5%81%E7%A8%8B (accessed Apr. 03, 2025).