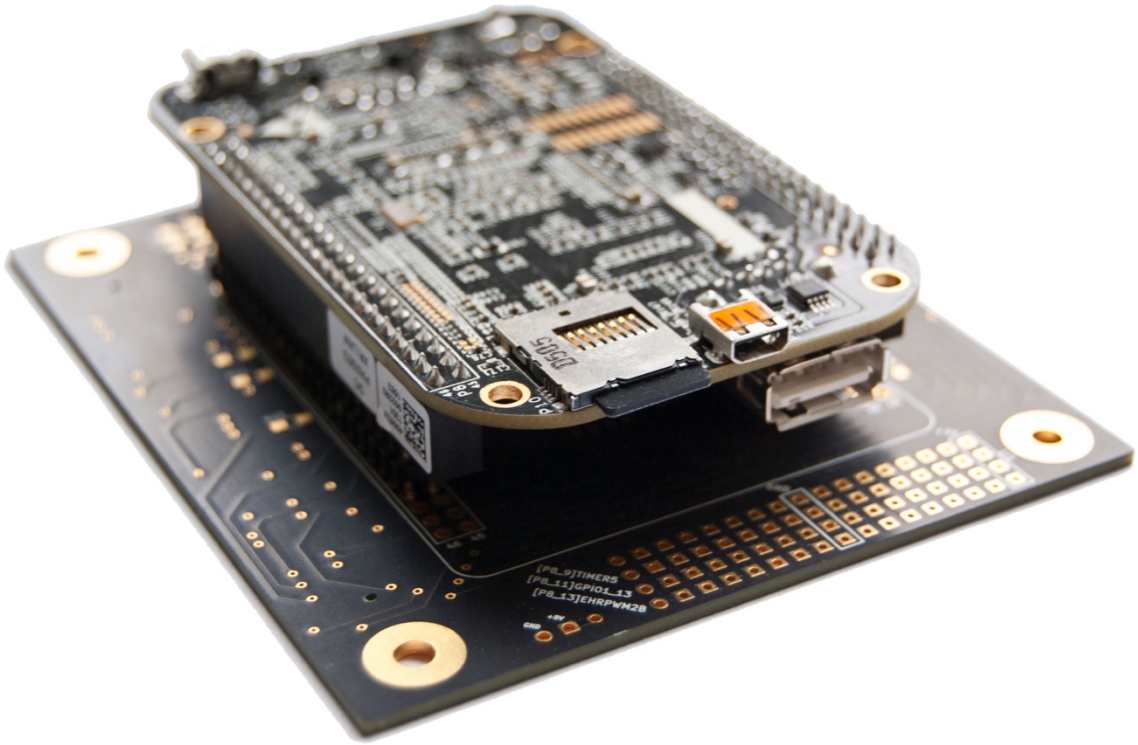
CHALMERS



Embedding Prefetching of Audio Content in ARM Processors

Bachelor's thesis in Computer Science and Engineering

Anders Spetz Rasmussen

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
GOTHENBURG UNIVERSITY
Gothenburg, Sweden 2017

DEGREE PROJECT REPORT

Embedding Prefetching of Audio Content in ARM Processors

ANDERS SPETZ RASMUSSEN

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY

Gothenburg, Sweden 2017

Embedding Prefetching of Audio Content in ARM Processors

ANDERS SPETZ RASMUSSEN

Examiner: Peter Lundin

Department of Computer Science and Engineering
Chalmers University of Technology / Gothenburg University
SE-412 96 Göteborg
Sweden
Telephone: +46 (0)31-772 1000

Cover: BeagleBone Black fitted on top of the host module PCB.

Department of Computer Science and Engineering
Gothenburg 2017

Abstract

This paper describes the design and implementation an audio amplifier host module. The hardware developed served as a physical interface between the amplifier and an ARM equipped evaluation board capable of running the Linux kernel. Both hardware and software development was carried out at amplifier manufacturer Lab.gruppen facilities according to specified technical requirements. The project resulted in a functioning embedded audio player prototype. More specifically, the project resulted in the necessary underlying hardware and firmware that is intended to serve as a platform for further development of the amplifier host module.

Keywords: BeagleBone Black, PCB, eLinux, ASoC, ALSA, I^2S, Device Tree

Sammanfattning

Den här rapporten beskriver design och implementering av en host-modul för en audioförstärkare. Den framtagna hårdvaran fungerade som ett fysiskt gränssnitt mellan förstärkaren och en ARM-baserat utvecklingskort med stöd för Linux. Både hårdvaru- och mjukvaruutveckling utfördes på plats i Lab.gruppen lokaler och enligt deras kravspecifikationer. Projektet resulterade i en fungerande prototyp avsedd för digital ljuduppspelning och mer specifikt i en hård- och mjukvaruplatform avsedd för vidareutveckling av host-modulen.

Nyckelord: BeagleBone Black, PCB, eLinux, ASoC, ALSA, I^2S, Device Tree

Acknowledgments

I would like to thank Lennart Hansson for sharing his expertise
and advice during the process of conceiving and writing this paper.

I would also like to thank all the people at Lab.gruppen
and in particular Marco, Karl-Johan and Lars for their personal and technical
support throughout the development process.

Contents

Nomenclature

ALSA – *Advanced Linux Sound Architecture*

API – *Application Programming Interface*

ARM – *Advanced RISC Machine*

BBB – *BeagleBone Black*

CDIO – *Conceiving* | *Developing* | *Implementing* | *Operating*

CPU – *Central Processing Unit*

DSP – *Digital Signal Processor*

eLinux – *Embedded Linux*

GPIO – *General Purpose In/Out*

I^2S – *Inter-IC Sound*

IC – *Integrated Circuit*

McASP – *Multichannel Audio Serial Port*

OSI – *Open System Interconnections*

PCB – *Printed Circuit Board*

PCM – *Pulse Code Modulation*

PDU – *Protocol Data Unit*

RISC – *Reduced Instruction Set Computing*

SFTP – *Secure Shell File Transfer Protocol*

SPI – *Serial Peripheral Interface*

SSH – *Secure Shell*

TTL – *Transistor-transistor Logic*

USB – *Universal Serial Bus*

1

Introduction

1.1 Background and Purpose

Ever since the transmission of audio content over networks was introduced on a commercial level an increasing number of devices with the ability to receive and play back an audio stream have surfaced. Lab.gruppen deals in the development of professional audio power amplifiers for both tour and installation. It is in the latter case where an investigation into the requirements and possibilities of implementing an embedded audio player in an existing amplifier is to be carried out.

Embedded audio players for streaming audio are relatively simple to implement in PC and mobile phone devices where the application specific requirements, such as connectivity and processing power, are already present. Thus, development budget sizes are motivated by the multifunctionality of those devices. In the case of the amplifier embedded audio player, however, the business prospects are based entirely on the audio player itself.

The audio player is to be embedded on a host module based around an ARM CPU in a medium sized audio amplifier. This host module complements the existing audio amplifier with the required connectivity and additionally expands the potentials of implementing several other interesting functions.

As such, the following questions are raised:

- What are the challenges faced when developing an embedded system?
- What are the implications of latency in such a system? How do you handle them?

1.2 Scope

While the final intention is to implement audio streaming over an IP-network, this lies beyond the focal point of the project. The focal point is rather to carry out proof-of-concept work, i.e. use an ARM based CPU and integrate this technology with an existing product. Choosing to work with an ARM application processor based evaluation board permits easier integration with second- and third-party software frameworks and tests thereof.

1.3 Method

The project followed the CDIO framework as first specified by Massachusetts Institute of Technology and later adopted and further developed in conjunction with, among other institutions, Chalmers University of Technology. As such, the work first focused on visualizing and conceiving the functionality of the planned prototype through relevant literature studies. This served to streamline the development of the required hardware and firmware and hence simplify implementation and user operation of the technology.

2

General System Description

This paper describes the implementation of a network host in an audio amplifier. The embedded system construction involved both hardware and firmware/software design. To generalize and thus make the system more understandable, it was modelled on the Open System Interconnections standard (OSI).

2.1 The Amplifier Host

Networked amplification is nothing innovative. In fact, Lab.gruppen was among the first manufacturers to license and implement Audinate`s Dante. [1] Dante, a digital audio networking protocol, was brought into the amplifiers to enable networked digital audio transport and device parameter control. [2] On the amplifier side, the protocol required a hardware/software interface to handle incoming digital audio and control signals. Consequently, a host Printed Circuit Board (PCB) was integrated in the amplifiers to serve this purpose.

The amplifier host forms an embedded Linux (eLinux) system connected to a network of other devices that facilitates communication between other devices on that network. Since the amplifier eLinux host is intended to receive and stream audio data it requires some sort of connection to the internet. Thus, when the technology has been implemented, the amplifier host gains the status of a network host. [3]

Network hosts are often depicted with the help of an OSI model. [4] While the complete OSI model is fully applicable in this system, doing so would go beyond the scope of this paper. However, due to the amplifier being fitted with an embedded system able to connect to the internet we can, as illustrated in figure 2.1, construct a model of the amplifier based on the OSI standard.

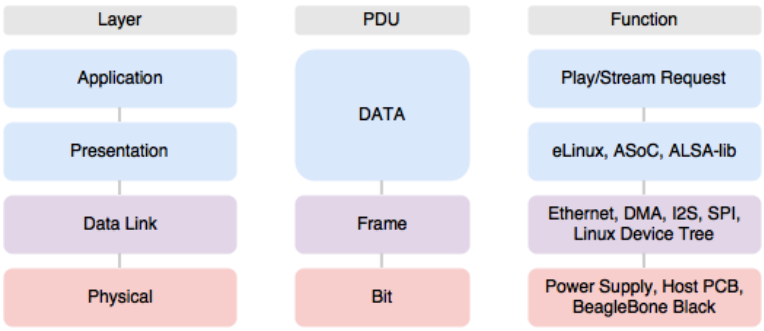


Figure 2.1 - OSI model of the amplifier network host. ***Application Layer****: Audio play/stream request.* ***Presentation Layer****: Audio data encoding/conversion.* ***Data Link Layer****: Audio data reception/transfer through Ethernet connectivity to DMA buffer.* ***Physical Layer****: Audio data physical interfaces (CPU/Host PCB/Amplifier) and the Linux Device Tree*

2.1.1 The Physical Layer

At the bottom of the OSI model stack dwells the physical layer. This layer is concerned with the reception of a raw unstructured audio bit stream and subsequently the transmission of the encoded bit stream over a physical medium. Application specific features such as power supply and BeagleBone Black (BBB) host PCB integration is described here.

2.1.2 The Data Link Layer

The next layer in the stack is the data link layer. This layer and the technology within is intended to provide error free transmission between two nodes on a network. This requires a well-defined network communications protocol such as Ethernet. The data link layer, we assume, is terminated when the data is stored in memory by the on chip integrated Direct Memory Access (DMA) controller. Furthermore, and as we shall see, to boot the Linux kernel in specific configuration the device tree is used and thus described here.

2.1.3 The Presentation Layer

The presentation layer correctly formats the data received according to well defined serial data bus protocols. In more detail, the presentation layer is entailed with data conversion; sample rate, bit resolution, bit order etc.

2.1.4 The Application Layer

At the top of the stack resides the application layer. This is where communication between two hosts on a network is established. Furthermore, the application layer can be visualized as the complete system application involving hardware, firmware and software and the correct design and configuration thereof, the complete application is generalized and described here.

2.2 The Streaming Audio Application

The amplifier host is prototyped around a Texas Instruments ARM application processor. This processor, in conjunction with other necessary host connectivity features, performs several tasks. Among them and most importantly the fetching, decoding and execution of digital audio. The system application audio signal flow can be simplified and illustrated by the below presented block diagram based on the digital-to-analogue audio chain.

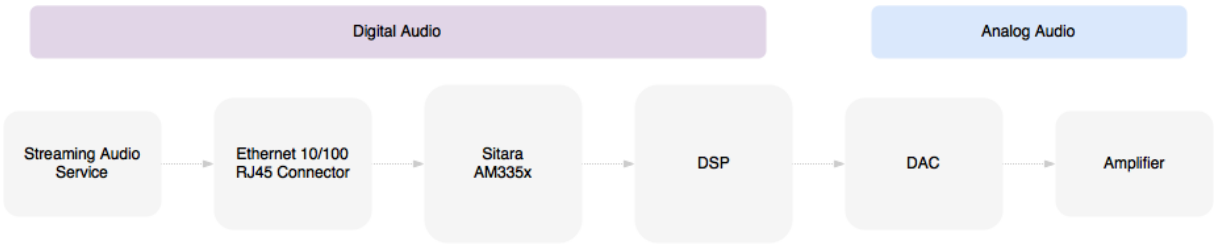


Figure 1.2 - The system digital-to-analogue audio flow

3

Design and Implementation of Physical Layer

Residing in the physical layer are all hardware related implementations. The following layer description is concerned with the design of hardware that ensures error free transmission over the physical mediums used, i.e. the BBB and Host PCB.

3.1 The Core of the Embedded System

At the core of the prototyped embedded system is an ARM Cortex-A8 processor. In conjunction with networking ability, memory and external clocking capabilities the embedded system processor communicates with the amplifier Digital Signal Processor (DSP) through a control and audio interface. This is illustrated in figure 3.1 below.

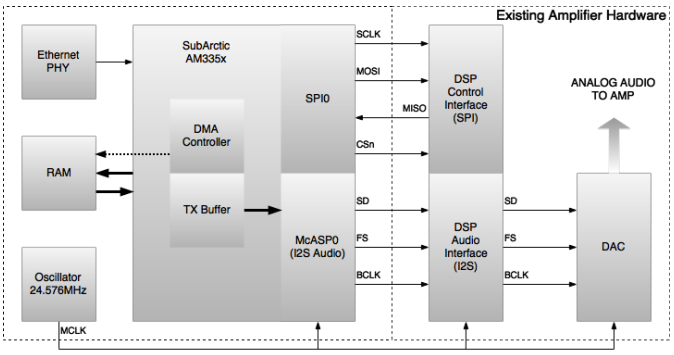


Figure 2.1 - The core of the embedded system and how it communicates with the amplifier DSP.

3.2 The Development Platform

There were a number of platforms that at first glance would seem appropriate for the project task of developing the eLinux host module prototype. These included but were not limited to, the Raspberry Pi and the BBB. Both platforms, however, have strong elaborative communities behind them and this in turn creates an accessible space in terms of development support. [5]

When considering audio capabilities and clock generation in particular, however, specific platform differences emerged. Since the audio amplifier host is responsible for generating the master clock for the System on Chip (SoC) digital audio peripheral as well as the amplifier DSP and digital/analog converter (DAC), a platform that can supply the correct clock frequencies was required. Both platforms support the 48kHz sample rate family but only the BBB achieves this through the implementation of an external crystal oscillator at 24.576Mhz. Furthermore, in addition to being open-source software, these boards are also open-source hardware, meaning the leap from prototype to finished hardware is minimized. [6]

3.2.1 The BeagleBone Black

The BBB is a single card, Linux capable open-hardware computer. The board centres a Sitara AM335x microprocessor which is based on the ARM Cortex-A8 architecture. This family of application processors were developed to provide support for a range of operating systems and thus making them the appropriate choice for embedded systems running the Linux kernel. [7] For prototyping projects involving eLinux centred around the AM335x microprocessor it forms an ideal platform. Since the BBB platform is open-hardware, the full PCB schematic is downloadable and free to use when taking the next step towards a finished product design. Furthermore, The BBB compatibility with Linux (Debian, Android, Ubuntu) combined with direct CPU input/output pin and bus access via expansion headers, makes it an ideal choice when wanting to interface the Linux kernel to external devices. [8]

The BBB and its pinout description etc. is presented in more detail in appendix **A.1**-**A.2**.

3.3 Printed Circuit Boards

The PCB is prevalent in modern electronics. It provides convenient means to assemble analogue and digital electronics to be implemented in confined spaces. These mixed signal systems, however, are prone to electrically harmful effects including but not limited to; stray capacitance and signal reflections on digital transmission traces. Therefore, the design of the PCB is often equally challenging as the circuit design itself. [9]

As illustrated in figure 3.2, modern PCBs are often multi-layered laminates comprised of two or more copper layers separated by an insulating substrate. It is this geometry, however and as we shall see, that provokes some of the difficulties when it comes to layout and board design. Circuit partitioning, to avoid cross-talk from parasitic components and between digital signal copper traces, needs careful attention to construct a functioning system. [10]

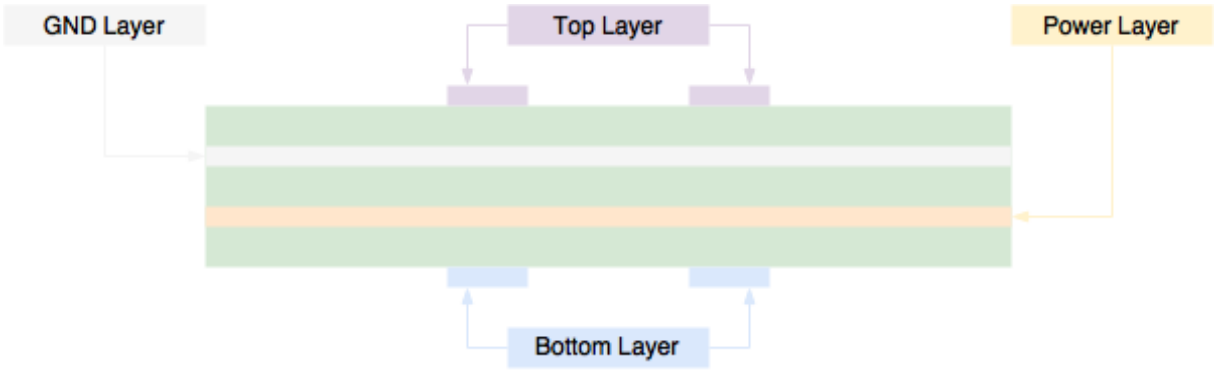


Figure 3.2 - Multi-layered PCB

3.3.1 Parasitic Components and Electromagnetic Interference

The cross section of a multi-layered PCB shows a geometry reminiscent of multiple capacitors. Similarly, two parallel signal traces can be regarded as a transformer. [11] The creation of these passive components is inevitable in multilayer PCB designs and can cause problems if not addressed properly. The stray capacitance and mutual inductance give rise to electric/magnetic fields, and hence the transfer of energy in the form of current and voltage respectively between signal traces on the PCB. [12]

As stated above, this transfer of energy is inevitable due to the nature of the PCB design and can only be minimized. Some of the complications that can arise include the corruption of precise logic signals in the form of reflections and/or ripple, and inter-IC interference resulting in degrading system performance. To minimize these complications there are several methods at hand, these include but are not limited to: [13]

- Circuit partitioning – properly separating analogue and digital signal circuits.
- Trace partitioning – addressing the space and area between circuit traces and loops respectively.
- Grounding structures – appropriate ground planning to separate critical components.
- Termination resistors – placement of termination resistors close to the signal source.

3.3.1 The Transmission Line and Reflection-less Matching

Any signal trace on a PCB can be regarded as a transmission line with a characteristic impedance Z_0. Furthermore, the trace is terminated at some load impedance Z_L. When a pulse signal travels down the transmission line with the impedance Z_0 and subsequently approaches the load impedance Z_L it will experience and react to this impedance. [14] To further describe and visualize the impedances Z_0 and Z_L, a suitable analogy is to refer to the impedances as two different mediums the pulse wave propagates through. Wave propagation theory then tells us that both absorption and reflection will occur at Z_L. The equation below describes the reflected wave both in terms of amplitude and phase. [15]

$$V_{reflected} = V_s \frac{Z_L - Z_0}{Z_L + Z_0}$$

Equation 3.1

This phenomenon can be problematic, and especially when dealing with precision sensitive digital circuits. At the extremes, open- and short-circuit, the whole wave is reflected with none and 180 degrees phase inversion respectively. However, when $Z_L = Z_0$ there is no reflection since the pulse wave does not perceive any transition from one medium to another. [16] Thus, reflection less matching or maximum power matching is achieved when source, transmission line and load impedances are all equal. Refer to appendix **B.1** for calculation of the characteristic impedance Z_0.

3.4 The Host PCB

To provide access for the BeagleBone Black to the amplifier a PCB adaptor card that would function as a physical interface between the BBB and the amplifier was designed. The PCB was designed based on the following requirement specifications:

- Self-sufficiency in terms of being able to power the evaluation card used in development. This would require the integration of a suitable voltage regulator on the PCB.
- Careful consideration to layout and trace routing to avoid crosstalk.
- The integration of pin-headers to provide direct access to the evaluation cards header connectors.
- The integration of host-to-amplifier connector.
- The integration of Power/Reset push-buttons. This due to the evaluation card being placed up-side-down on the adaptor PCB and therefore hindering access to the power and reset buttons on the evaluation card.
- The integration of a pin-header to provide access to the evaluation cards serial port/console.
- The integration of LEDs for easy verification of adaptor card power state etc.
- The integration of a stripboard to provide hardware modification ease.

(For complete PCB layout description refer to appendix **B.2**.)

3.4 Power Supply

The BBB requires a power source able to supply 5V DC at a maximum current draw of 2A. This can be done by means of connecting an AC/DC wall-wart converter to the integrated barrel jack on the card or also providing power over USB. However, to provide further prototype development ease and avoid tampering with the chassis of the amplifier, the decision to make the adapter card self-sufficient in terms of power seemed like a reasonable way forward.

As mentioned above the BBB operates at 5V DC and at a maximum load current of 2A. This scenario, however, is at the very extreme. It would be plausible to assume several, if not all, of the possible peripheral connections available were being used under conditions as such. Instead, investigating current draw during an idling state seemed like a more apt starting point. According to idle and nBench benchmark tests on the BBB and the test results thereof presented by Tony DiCola of Adafruit Industries, the BBB draws a current of approximately 300mA during an idling state. [17]

3.4.1 Linear vs. Switched Regulators

Linear regulators are cheap and due to the requirement of few external parts besides the actual IC involve less space when designing a board. Moreover, linear regulators do not exhibit any form of high frequency switching noise and require few parts besides the actual IC to function. These characteristics make them very simple to integrate in any PCB design. However, when it comes to efficiency there is no comparison to switched regulators as most linear regulators have typical efficiency of around 40% compared to the efficiency of switched regulators which typically reach 85%. When determining which of the two types of regulators better suit the given application, the load current I_{LOAD} and voltage difference between V_{IN} and V_{OUT} is of key significance. For linear regulators, the greater the voltage drop between V_{IN} and V_{OUT} the greater the power loss. [18] Given that the BBB is in an idling state drawing a current of 300mA and the regulated voltage drop is 11V the power loss dissipated in heat would be:

$$(V_{IN} - V_{OUT}) \cdot I_{LOAD} \Longrightarrow (16 - 5) \cdot 0.3 = 3.3W$$

Equation 3.2

This theoretically high dissipation of heat already at an idling state partly spoke for a switched regulator to be the better choice.

3.4.2 The LT1076CQ-5 Buck Converter and Power Stage

For the application, a step-down switching regulator (LT1076CQ-5) was chosen. The integrated circuit (IC) LT1076CQ-5 is a 5-lead TO-220 package with a fixed output voltage of 5V/2A and with an input voltage range of 8V-40V, and requires few external power stage components to function normally. [19] The power stage of a Buck Converter circuit is built up of several components besides the actual switched IC as illustrated in figure 3.3. These components need to be properly dimensioned to withstand the maximum switch current the IC outputs. [20] To calculate the power stage of a Buck Converter, four parameters of the intended circuit need to be known:

- The input voltage range: $V_{IN(min)} - V_{IN(max)}$
- The nominal output voltage: V_{OUT}
- The maximum output current needed by the application: $I_{OUT(max)}$
- The IC chosen for the buck converter design.

A detailed description of power stage calculations is presented in appendix **B.3**.

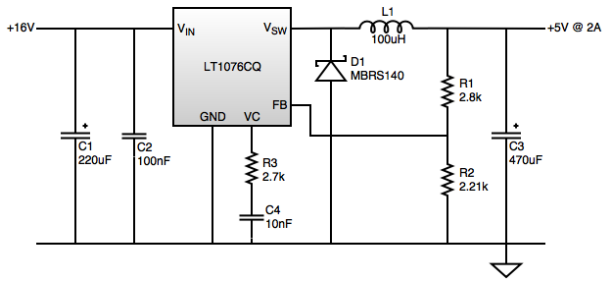


Figure 3.3 - The LT1076CQ-5 Power Stage

When designing a voltage converter circuit to be integrated on a PCB, considerations as to how the design is laid out on the PCB is of great importance. In the next section of this paper, some guidelines as to how to go about integrating a Buck converter circuit on a PCB are presented.

3.4.3 Buck Converter PCB Layout

Switched regulators display superior characteristics compared to linear regulators in terms of efficiency. However, in terms of electrical noise, they are inferior much due to the inherent high frequency switching noise of the actual IC as well as noise induced by current loops in surrounding components (power stage). [21] This implies that the noise induced by the power stage can be minimized by means of careful attention to layout of the components when designing the PCB.

To reduce undesirable side effects such as increase in voltage noise and lack of stability of the regulated voltage, certain attention to component layout must be considered. ROHM Semiconductor specifies the following general points to be studied:

1. Input capacitor and fly-back diode are to be placed on the same layer and as close as possible to the regulating IC.
2. Consider the placement of thermal vias to improve dissipation of heat.
3. To minimize noise radiating from V_{SW}, the inductor should be placed as close as possible to the IC. Do not make the copper area connected to V_{SW} larger than necessary.
4. The output capacitor needs to be placed close to the inductor.
5. The return path copper traces need to be kept away from electrically noisy areas. i.e. inductor and diode. [22]

In addition to the design suggestions above, a decision as to on which side of the PCB the circuit was to be placed had to be made. Due to the evaluation card being placed up-side-down on the adaptor card with the CPU and other critical components then also facing the same direction, the Buck converter circuit had to be placed on the back-copper layer of the adaptor PCB facing down towards the amplifier. Figure 3.4 below, show the top and bottom copper layer traces and areas in black.

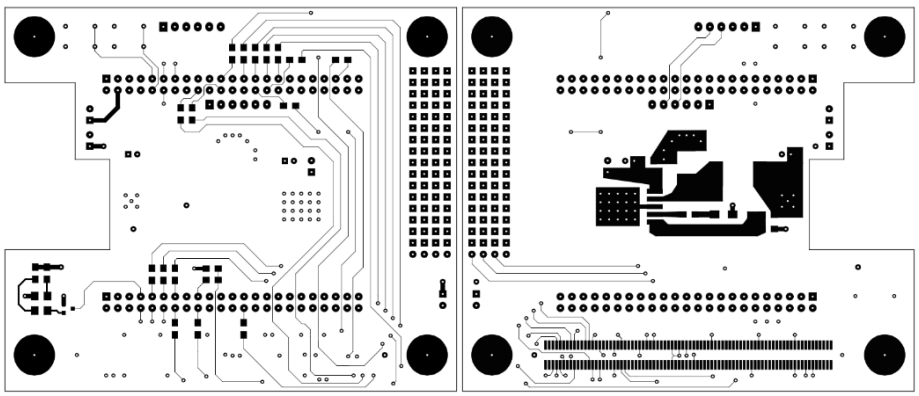


Figure 3.4 - PCB copper traces and areas.

4

Configuration of Data Link Layer

The data link layer description presents methods of establishing communication and network connection to the BeagleBone Black. These connections are not only relevant for the system as it is meant to function when fully implemented, but also serves the purpose of providing a functional software development setup. Moreover, the description is also concerned with how incoming audio data is temporarily stored in data buffers and the resulting latency thereof. The device tree description is also included here as it provides an efficient way of configuring the processor peripherals used to transfer data to the amplifier.

4.1 Connecting to the BeagleBone Black

To establish communication and network connection to the BBB there are three main ways; Internet-over-USB, regular Ethernet and thirdly with the use of an Ethernet crossover cable. The Ethernet crossover cable allows Ethernet devices to be interconnected without the necessity of an Ethernet switch. All three ways allows for the connection of the BBB to the internet and desktop software development environment of choice. [23]

Once a network connection has been established the network protocol Secure Shell (SSH) can be utilized to connect to the BBB SSH server through an SSH terminal on the development environment host. This allows the user to remotely execute commands and transfer files to and from the BBB using the SSH File Transfer Protocol (SFTP). [24]

The BBB also features a 6 pin Serial Debug port to fall back on should problems with the network connection arise. However, this port is also useful to monitor the Linux kernel boot process. Connection to the Serial Debug port is done with an USB-to-TTL cable. [25]

During software development, the setup illustrated in figure 4.1 below was used.

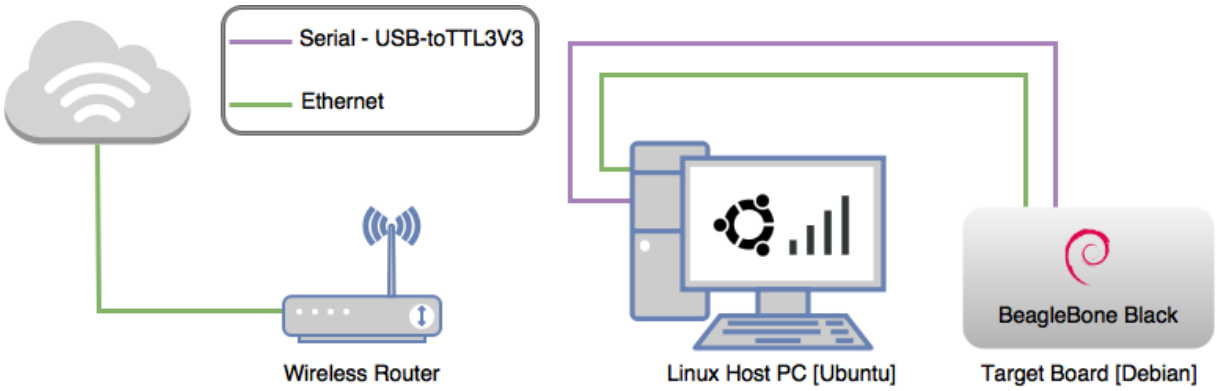


Figure 4.1 - Linux host PC sharing network connection with target board (BBB).

4.2 Direct Memory Access

For data heavy applications, Direct Memory Access (DMA) provides a way to lessen the burden on the CPU. The DMA controller gains control over the processor memory bus and thereby permits greater data throughput. [26]

As illustrated in figure 4.2 below, incoming data is placed in memory by the DMA controller. This process is initiated after the DMA controller and the CPU have exchanged DMA request and DMA acknowledge signals respectively. Assuming the CPU acknowledges the request, the DMA controller then takes complete control driving address and data buses as well as control signals. After the DMA data transfer has been completed, control is handed back to the CPU until next interrupt request is asserted. [27]

The Sitara AM335x processor features an internal DMA controller [28], which in addition to eliminating the necessity of syncing to external buffers, also allows for the transfer of data directly to on chip peripherals, in this case the McASP peripheral. [29] For clarity of data flow, however, below illustration uses external random access memory (RAM) to exemplify the described technology.

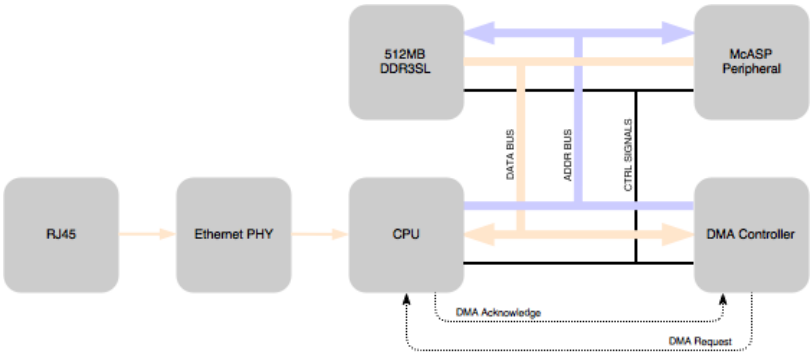


Figure 4.2 – Data flow from Ethernet connector and, subsequently, how it is handled by the DMA controller

4.3 The Linux Application Buffer

An audio playback application requires a buffer to store incoming data. Also, to avoid overworking the CPU, the buffer needs to be sufficiently large. Likewise, if the buffer size is too large it delays data to the point unnecessary for the system to still function properly. [30]

At its core, the description is concerned with the Advanced Linux Sound Architecture (ALSA, section 5.2) application buffer. Consequently, the calculated latency is based only on what the application buffer induces, not including other buffers in the specific digital audio chain. [31] Furthermore, and as we shall see, buffer induced latency is hardly problematic when discussing a stand-alone digital playback application.

ALSA application buffers are ring buffers divided into four layers; buffer, period, frame and sample. This is due to the size of the buffer often being too large to transfer at once. To allow faster transfer to the Multichannel Audio Serial Port (McASP) serializers the buffer is fragmented into periods holding stereo interleaved L-R sample frames. [32] The interrupt frequency generated by ALSA is controlled by setting the period size. Thus, latency is somewhat controlled by the predetermined period size. [33] Somewhat, because buffer size needs to be at least two times the period size. The ALSA application buffer configuration is illustrated in figure 4.3 below.

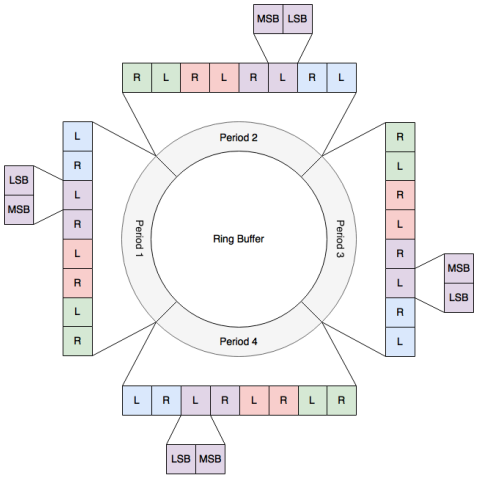


Figure 4.3 – Application ring buffer.

So, when does latency become a problem? Consider a recording studio where some musicians are engaged in live recording. The musicians each have their own room to avoid leakage between microphones set up for the various instruments. To be able to hear each other they are fed a monitor mix from the main tracking console in the control room. In this case, it is essential that latency is kept at a minimum to avoid timing difficulties for the musicians. Buffer induced latency then becomes a problem.

4.3.1 Buffer Induced Latency Calculations

This section presents an example of how to calculate ALSA application buffer induced latency. The described system is intended to operate at a 24Bit/96kHz audio resolution. The interrupt frequency generated by ALSA is controlled by setting the period size. To demonstrate the data-rate the system must be capable of we need to decide the following parameters:

- Sample size: 3Byte (24bit)
- Frame size: 6Byte
- Period size: 1365 frames [8190 Bytes]
- Buffer size: 5460 frames [32760 Bytes]
- Sampling rate: 96kHz
- Bps rate: 576000 Bytes/sec

With the above parameters, the ALSA interrupt frequency ($ALSA_{if}$) would be approximately 70Hz or every ~14ms.

$$ALSA_{if} = \frac{Period\ size\ (Bytes)}{Bps\ rate\ (Bytes\ per\ second)} = \frac{8190}{576000} \approx 14ms$$

Equation 4.1

Consequently, it is not the buffer size setting that affects overall buffer latency but instead the number of periods and the size of the periods. The buffer induced latency ($ALSA_{bl}$), assuming the above specified parameter settings, would total to around 56ms. [34]

$$ALSA_{bl} = \frac{Buffer\ size\ (Bytes)}{Bps\ rate\ (Bytes\ per\ second)} = \frac{32760}{576000} \approx 56ms$$

Equation 4.2

The above calculated latency is hardly problematic when discussing a stand-alone audio playback application. However, in a recording situation this kind of latency is unacceptable as it would lead to audible delay and hence coloration of sound. As the buffer induced latency is related to the number of periods in the buffer and if it is a problem, a way to solve this is to reduce buffer size by reducing the number of periods. This can be done by using the ALSA plugin interface described in section 5.2.1.

4.4 The Linux Device Tree

As opposed to PC systems, eLinux does not have a BIOS which primary purpose is to initialize necessary peripheral units on the motherboard. Instead, eLinux systems use files stored on a solid-state storage device to describe and initialize the peripherals specific to the system in question. Therefore, to describe the hardware peripherals used by any eLinux system, a unique set of files is needed. [35]

Before Kernel 3.8, as required by the specific eLinux system in question, a hardware descriptive file containing the necessary modifications to the Linux kernel was integrated directly in the Linux source code. However, due to increasing popularity of ARM-based evaluation boards and the resulting customization of the Linux kernel to describe the various and differing features of every ARM device, mainline Linux saw a significant expansion in amount of code. [36] For this reason, new ARM boards running the latest kernels use the flattened device tree (FDT) instead.

4.4.1 The Flattened Device Tree

The FDT is a data structure based on and like a binary tree structure where each node in the structure contain device hardware properties along with connections to parent and child nodes. [37] Basic device tree syntax is illustrated in figure 4.4 below.

Figure 4.4 – Representation of the device tree syntax with hardware node, node properties and values.

Consequently, since no processor and/or board architecture is alike, each data structure representative of a device has its unique set of device tree files to describe that device. These files can be categorized into two categories; .dts – board hardware definitions, .dtsi – SoC hardware definitions. For ARM-based CPUs these device tree source files are in the Linux source code directory; *arch/arm/boot/dts*.

Apart from describing the hardware present and used in the embedded system, the device tree also provides pinmuxing functions. That way, when moving towards a functional prototype, the system can be appropriately booted in the preferred configuration. For the interested, the Linux boot process and the system specific .dts file is described in more detail in appendix **C.1** and **C.2** respectively.

4.5 Serial Communication

Serial communication is a process where data is transmitted or received sequentially, one bit at a time. Integrated circuits (IC) employed on the same PCB or device often use serial communication to talk to each other. To reduce the number of pins and thus size and production cost, these ICs use serial buses such as SPI and I^2S which are described in this paper. [38]

4.5.1 Inter-IC Sound (I^2S)

Inter-IC Sound (I^2S) is a 3-line serial data protocol developed by Philips Semiconductors in 1986 as a result of an increasing number of digital audio devices being introduced on the consumer market. The I^2S bus was, as the name implies, developed for digital audio communication between ICs.

The bus, as mentioned above, uses three wires to transfer data:

- The serial data wire *SD* which is two time-multiplexed. One data wire is therefore capable of transmitting two channels (stereo) of audio data.
- The frame sync *FS* or *LRCLK* wire indicates which channel (Stereo Left or Right) is being transmitted.
- The serial clock wire *SCK* or BCLK which synchronizes transmitter and receiver.

In addition to the three wires described above there is a master clock (MCLK). This clock is commonly included and used by the audio codec to time and synchronize its internal operations. [39]

The frame sync *FS* clock signal is a 50% duty cycle signal operating at audio data sample frequency. It is this clock that enables I^2S to send two channels of audio data on the same wire. The BCLK frequency is the product of the audio sample rate, number of bits per channel and number of channels and must be set to match the number of bits transmitted per *FS* period, this clock will then pulse once for each bit on the *SD* wire. [40] In this case, due to the physical data bus being 32 bits wide, the BCLK will operate at a frequency of:

$$BCLK_{freq} = 96000 \cdot 32 \cdot 2 = 6.144MHz$$

Equation 4.3

Since I^2S was developed by Philips Semiconductors, several similar formats have sprung out of that original format. In this project, as specified and required by the existing hardware to function properly, the Left-Justified mode is used. Left-Justified mode is characterized by, as can be seen in figure 4.5 below, the immediate transmission of data bits on the transition of *FS* as opposed to I^2S mode where there is a delay of one *BCLK* clock cycle after *FS* transition.

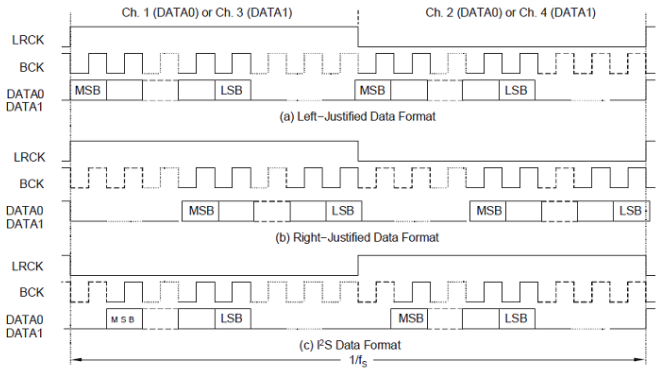


Figure 4.5 - Timing diagram for I2S formats [41]

4.5.2 Serial Peripheral Interface

The Serial Peripheral Interface (SPI) standard was developed by Motorola. It is a synchronous communication protocol used extensively in embedded systems. SPI operates in full duplex mode, i.e. communication in both directions simultaneously, configured in a master-slave architecture. The SPI bus is a four-wire bus and contains the following signals:

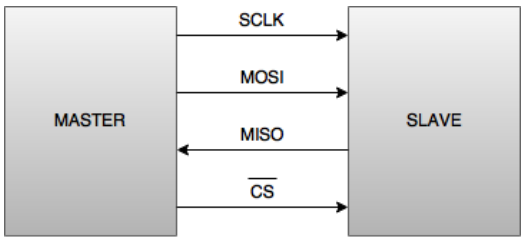


Figure 4.6 – SPI Master/Slave configuration

- SCLK – The master controlled Serial Clock
- MOSI – Master Out / Slave In
- MISO – Master In / Slave Out
- $\overline{CS}$ – The active low Chip Select (sometimes SS for Slave Select)

The chip select signal allows for multiple slave devices. The serial data is clocked by the SCLK and this happens simultaneously at master and slave sides. SPI is capable of data transmission rates of up to 1Mbps. [42]

4.5.3 Amplifier Master/Slave Configuration

The amplifier uses SPI communication between the host (Master) and DSP (Slave) to control various audio related parameters such as the activation of specific channels, gain control etc. When communication is initialized on the master (host) side a checksum derived from the word to be transmitted is calculated. The word including the checksum is then transmitted and received by the slave device (DSP) which stores the checksum calculated by the host in memory. Subsequently, the slave device then calculates a checksum derived from the same word and compares this to the one stored in slave memory. If the checksum matches up with the slave checksum, parameter adjustment on the slave device (DSP) is carried out.

5

Configuration of Presentation Layer

The presentation layer description is concerned with the configuration and building of the Linux kernel source code. Furthermore, the description also specifies and depicts the various software frameworks and their underlying components, such as codec, machine and platform drivers, used to realize an eLinux audio playback system.

5.1 Embedded Linux

The term embedded Linux is used to suggest an embedded system running mainline Linux. Thus, there is no unique Linux kernel intended for embedded systems, although configurations of mainline Linux built to fit a specific application are abundant. [43] In the case of the embedded audio prefetching application described in this paper, and due to the BBB being used as development platform, a set of scripts authored by Robert C. Nelson were used to configure the Linux kernel appropriately. [44] These scripts were used in a desktop Ubuntu host environment to rebuild, by means of cross-compilation, the mainline Linux kernel for the ARM target device. More on the actual process of how the application specific configurations were carried out is presented further down in this paper.

The basic requirements, as stated by Robert C. Nelson, to build a working kernel for ARM devices are an ARM Cross Compiler, Bootloader, Linux kernel source and an ARM based root file system. For clarity, the requirements with their respective sources are listed below. [45]

- ARM Cross-Compiler
    - Source: Linaro toolchain binaries [S1]
- Bootloader
    - Source: Das U-boot [S2]
- Linux kernel source
    - Source: Mainline tree [S3]
- ARM based Linux root file system (rootfs)
    - Source: Debian [S4]

5.1.1 The Linux Source Code

The source code of any software or firmware is the code as it was written by a programmer to model an application or operating system. In the case of the Linux kernel source code, it exists in numerous versions. [46] For the project, the mainline Linux kernel 4.8 was used. The source together with a set build scripts and instructions authored by Robert C. Nelson forms a working Linux kernel that can be installed on a microSD card to be booted on the BBB. However, to further configure the kernel for application specific requirements, edits as well as additions might be necessary.

The workflow during development thus involves the following steps:

1. Building bootloader, kernel and root file system from source.
2. Install on microSD card.
3. Verify the kernel is bootable.
4. Make edits to kernel source and rebuild.
5. Install rebuilt kernel on microSD card and debug.

In the case of the development of an eLinux audio application involving all the software components necessary, the *Sitara Linux Audio DAC Example* serves as an appropriate starting point. The *Sitara Linux Audio DAC Example* [47] is an introductory tutorial to audio codec development and kernel configuration. The tutorial uses the AM335x EZSDK 7.0 kernel source but is applicable on other kernel sources as well. The guide specifies the following source directories to be accessed and edited:

- *sound/soc/codecs/lpcm.c*
- *sound/soc/codecs/Kconfig*
- *sound/soc/codecs/Makefile*
- *sound/soc/davinci/davinci-evm.c*
- *sound/soc/davinci/Kconfig*
- *sound/soc/davinci/Makefile*

The above listed files and directories form a part of the ALSA System on Chip (ASoC) layer described in section 5.3 of this paper.

5.2 The Advanced Linux Sound Architecture

The Advanced Linux Sound Architecture (ALSA) is a software framework that forms a part of the Linux kernel. The intention of ALSA is to provide an API for the development of soundcard device drivers capable of automatic audio hardware configuration and fluid management of various audio devices in a system. Some of the noteworthy features of ALSA specified by *alsa-project.org* include: [48]

- Support for both consumer and professional multichannel audio interfaces
- Modularized audio drivers
- The library *alsa-lib* to simplify programming

The ALSA API (Application Programming Interface) can be divided into categories of supported interfaces. These include, but are not limited to; *control interface* – management and probing of sound card registers, *PCM interface* – management of digital audio playback/capture.

Since the release of kernel 2.6, ALSA forms a part of the stable Linux kernel and it offers an array of functionality. Among these functions is the use of device plugins for audio routing, sample rate conversion etc. [49]

5.2.1 ALSA Plugins

Music streaming services use a variety of compression formats and bitrates to transmit audio to the end user. For example, Spotify uses three different bitrates depending on user configurable preferences and/or whether the user is a premium subscriber or not. [50] To handle the various formats and bitrates on the application side, ALSA provides several ways to perform sample rate conversion on audio data. Usually this conversion is handled by the player and ALSA. It is then implemented in the player through the use of the ALSA API. However, when the setup of the soundcard driver is defective and/or complications are experienced, ALSA provides system-wide and user specific configuration files to handle these problems.

Another way to perform sample rate conversions is with the use of ALSA plugins. The plugins and their parameters are defined in configuration files that are parsed when the soundcard driver is called upon by an ALSA application. There are two configuration files that can be used depending on preference; the user specific *~/.asoundrc* and the system-wide configuration file */etc/asound.conf*. [51]

In the first code block in the code example in figure 5.1 below, *sl1* is defined as a slave to the PCM device *plughw*. The second code block further defines a PCM device named *complex_convert* of type plug and slave under the slave *sl1*.

```
pcm_slave.sl1 {
        pcm "plughw:CARD=LPCM,DEV=0"
        format S24_LE
        channels 2
        rate 96000
}

pcm.complex_convert {
        type plug
        slave sl1
}
```

Figure 5.1 - ALSA sample rate plugin definition in /etc/asound.conf

When playing audio through the command-line audio utility *aplay*, the plugin is invoked by:

debian@arm:~$ aplay –D complex_convert some-audio-file-name.wav

aplay now plays audio with the parameters defined *pcm_slave.sl1*.

5.3 ALSA System on Chip

The ALSA System on Chip (ASoC) layer was developed to provide better ALSA support for embedded system on chip processors and audio codecs. The layer is modular to provide ease of use when working on systems where hardware components, such as audio codecs, change. Thus ASoC, by means of its modular form, offers a convenient way to add or remove audio hardware to the system.

Codec drivers describe the capabilities of a soundcard or audio interface. These capabilities are not altered when used on other platform and machines. Therefore, it is important for the codec drivers to be platform independent, i.e. the codec driver should not contain any platform specific code. Moreover, the ASoC layer intends to provide easy SoC/codec and hence PCM/I^2S interface configuration by registering capabilities that are subsequently matched with application hardware parameters. [52]

Embedded system power management must be taken into consideration when working towards a finished system. This can be handled by the ASoC layer Dynamic Audio Power Management (DAPM), which is developed with portable devices in mind. [53] The DAPM function of the ASoC layer intends to automatically keep the codec at a minimum power consumption at all times depending on current configuration in terms of audio routing and active/inactive streams.

In addition to the above stated issues addressed by the ASoC layer also provides pop and click reduction, this is achieved by implementing correct codec power up/down sequences. To realize all the functions described above, ASoC is split into the three components described below. [54]

- **Codec driver**: platform independent and handles any codec specific audio controls, IO functions and interface capabilities.
- **Platform driver**: holds the audio interface drivers, in this case the I^2S interface driver, for the platform in question.
- **Machine driver**: links the codec and platform drivers.

While the codec driver does not fill any function as to configure the amplifier DSP/Codec IC in this case, it was nonetheless necessary in terms of creating a basic functioning ASoC system to allow for further development at a later stage. Due to the driver code not targeting any specific codec/DSP (currently implemented separately through the SPI interface) control interface it can be thought of as a dummy codec. However, to function as such, two structures are of particular interest; *struct snd_soc_dai_driver*, which holds the hardware parameters defined previously, and *struct of_device_id* which makes the driver device tree compatible.

As mentioned above, the ASoC machine driver joins the codec and platform drivers. More specifically, it is the machine digital audio interface (DAI) configuration within the machine driver that acts as this link and similar to the codec driver, the machine driver source code contains two structures of particular interest; *struct snd_soc_dai_link* and *struct of_device_id.*

The DAI link structure *snd_soc_dai_link* sets the McASP interface in the correct configuration with regards to format and sample rate and the structure *of_device_id* allows for this configuration to be called from the device tree. [55] [56] The object diagram below illustrates the ASoC layer's architecture and how it is called from the device tree.

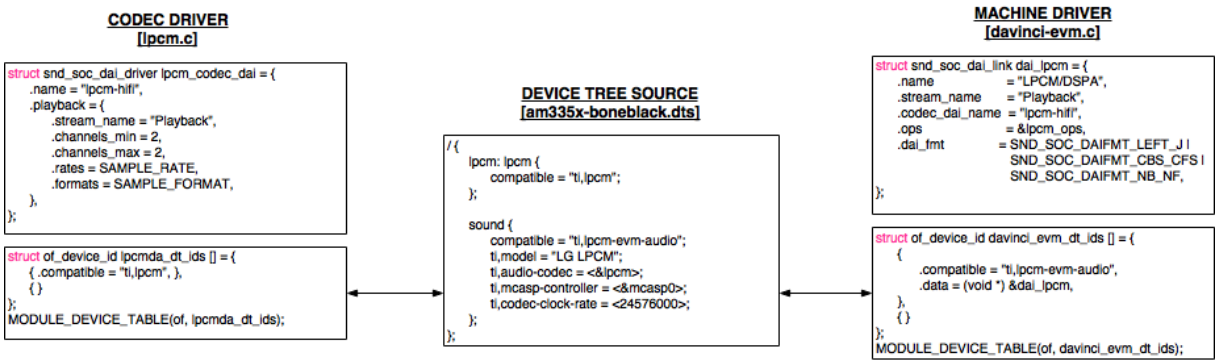


Figure 5.2 - Description of how the ASoC layer configuration is called from the device tree.

5.3.1 Kconfig and Makefile Entries

To allow the codec driver to be built into the kernel, entries to the files *Kconfig* and *Makefile sound/soc/codecs* directory are required. [57] Furthermore, at build time the driver is made selectable in the Linux graphic configuration interface by an entry in the Kconfig file in the *sound/soc/davinci* directory of the source code. Both code entries are presented in figure 5.3 and 5.4 below.

sound/soc/codecs/Kconfig	sound/soc/codecs/Makefile
select SND_SOC_LPCM config SND_SOC_LPCM tristate	snd-soc-lpcm-objs := lpcm.o obj-$(CONFIG_SND_SOC_LPCM) += snd-soc-lpcm.o

Figure 5.3 - Kconfig and Makefile entries sound/soc/codecs

sound/soc/davinci/Kconfig
config SND_SOC_LPCM tristate "SoC Audio support for LPCM on Beaglebone Black" depends on SND_DAVINCI_SOC_MCASP && SOC_AM33XX select SND_SOC_LPCM help Say Y or M if you want to add support for SoC audio on AM33XX BeagleBone board using McASP and LPCM.

Figure 5.4 - Kconfig entry: sound/soc/davinci

6

The Application Layer

The application layer is where the play/stream request functions are implemented. However, since the developed application involved both hardware and software, the application layer description can be looked upon as a detailed overall system description. The description is based around a three-stage pipeline structure that the digital audio data passes through.

6.1 The Embedded Linux Audio Application

Any Linux audio application such as *aplay*, the command line audio player, commonly follow a well-defined sequence of operation when initiated. [58] For streaming audio playback, this sequence can be generalized and represented in the form of the pseudo-code presented below:

open and connect socket(port, addr etc.)
initiate playback interface
set hardware parameters (format, rate etc.)
while not end of data to be processed:
audio playback
terminate playback interface
disconnect and close socket

The above pseudo code can be further generalized according to a Fetch | Decode | Execute model. Furthermore, since there is both hardware, firmware and software involved in the Fetch | Decode | Execute audio data processing model, it can be applied on the developed embedded system prototype as a whole. A visual representation of the model is presented in figure 6.1 below.

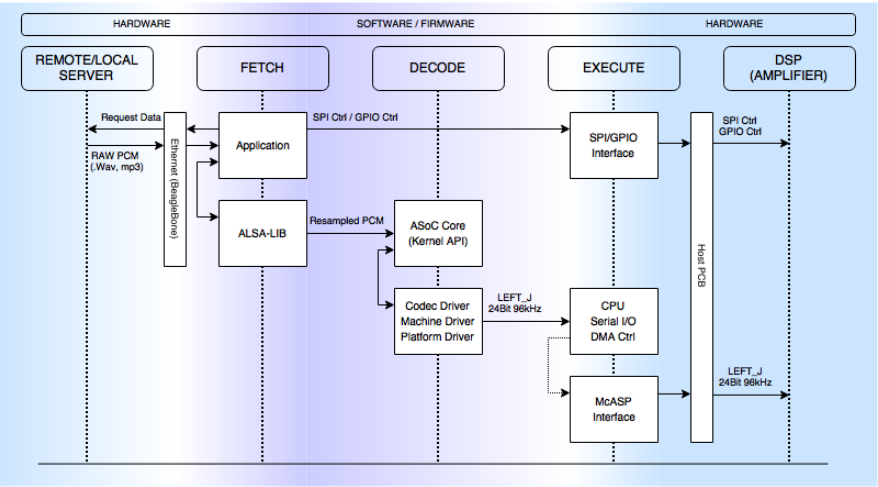


Figure 6.1 - Fetch | Decode |Execute model of the realized embedded system prototype

This serves to conclude the system technical description of this paper, we can develop on the stages of the generalized model above and follow the signal through the various hardware and software components designed and implemented.

7

Result

Proof-of-Concept project types aim, as implied, to prove a hypothesized concept. Therefore, the result might not be considered as important as the proof that the hypothesized concept worked. That said, if the project aim is reached and the concept has been proven to work, it is important to have developed a stable and well-defined platform that can serve well in further development.

The project entailed design and implementation of both hardware and firmware/software modules. For any of the above three specified main project modules to function properly as a complete application unit, careful attention to correct design had to be taken both to hardware and software. On the one hand, particular concern had to be taken to hardware design. This to ensure corruption free transmission of digital audio data. On the other hand, the delve into the world of interfacing electronics to embedded Linux proved to be an equally challenging task.

Both tasks of construction and implementation, however, were proven successful and the developed hardware and firmware functioned well when connected to the amplifier through the hardware interface, i.e. amplifier parameters were controllable and bit perfect audio could be played back and heard over speakers.

8

Discussion

As mentioned in results, the project resulted in a platform that can serve as a base for further development. While the prototype showed and proved the proposed concept it is still in a very basic form, i.e. it is able to playback/stream local and remote hosted audio respectively.

When developing an embedded system based around the Linux kernel, more often than not the scope is aimed at one or a few specific tasks this embedded system is supposed to accomplish. The kernel then, as it was built to accommodate the functions of the development board (BBB), is configured to accommodate for evaluation purposes that stretch far beyond the necessities of the specific function the eLinux system is intended for in the first place. Thus, for the intended system, the kernel in question contains unnecessary peripheral drivers and code.

Out of all the serial communication peripherals available on the Sitara AM335x processor only two were used. Then, building on the arguments mentioned above, it is clear that the kernel is scalable in terms of physical size and hence efficiency. Conversely, one first has to consider the amplifier and its capabilities and hence the required capabilities of the host hardware as well. As mentioned in the main sections of this paper, for example, the SPI master only communicates with one of several slaves available on the amplifier side. Therefore, the host PCB hardware has to be expanded to accommodate for this communication.

It is also necessary to consider the discussion concerning latency. At a bare minimum, the induced buffer latency is not a problem; bare minimum being the amplifier embedded audio player used as a standalone unit feeding only one stereo channel. However, considering the case of “install amplifiers” that often form part of bigger building audio systems, for example in shopping centres etc., latency and system response times is of key importance. First and foremost, this is due to the amplifiers forming a part of the security infrastructure of the building. In the case of forced building evacuations, for example, the system has to be able to respond to such requests rapidly and with minimum of delay. Secondly, when networking amplifiers to play audio in different rooms of the building, latency can cause problems in terms of being able to correctly sync the devices in that network. With regards to handling these circumstances, the implementation of Real Time Linux (RTLinux) would serve as a possible way forward.

References

[1] Lab.gruppen, “Networking”, http://labgruppen.com/technology/networking, [Accessed 28 May 2017]

[2] Audinate, “Dante Overview”, https://www.audinate.com/solutions/dante-overview, [Accessed 28 May 2017]

[3] The Linux Information Project, “Host Definition”, http://www.linfo.org/host.html, [Accessed 28 May 2017]

[4] Ciscopress.com, “The TCP/IP and OSI Networking Models”, http://www.ciscopress.com/articles/article.asp?p=1757634&seqNum=2, [Accessed 28 May 2017]

[5] Linux.com, “Top ten open-source…”, https://www.linux.com/news/top-10-open-source-linux-boards-under-200, [Accessed 31 May 2017]

[6] Hifiduino, “Raspberry Pi B+ Digital Audio”, https://hifiduino.wordpress.com/tag/raspberry-pi-digital-audio/, [Accessed 31 May 2017]

[7] Texas Instruments, “Sitara™ AM335x Processors”, http://www.ti.com/lsds/ti/processors/sitara/arm_cortex-a8/am335x/overview.page, [Accessed 28 May 2017]

[8] D. Molloy, “Exploring BeagleBone: Tools and Techniques for Building with Embedded Linux”, John Wiley and Sons Inc., pp.3

[9] Analog Devices, “Printed Circuit Board Design Issues”, http://www.analog.com/media/en/training-seminars/design-handbooks/Basic-Linear-Design/Chapter12.pdf, pp.12-1 | [Accessed on: 28/6-2017]

[10] Analog Devices, “Printed Circuit Board Design Issues”, http://www.analog.com/media/en/training-seminars/design-handbooks/Basic-Linear-Design/Chapter12.pdf, pp.12-1 | [Accessed on: 28/6-2017]

[11] Incompliancemag.com, “Crosstalk reduction between PCB traces”, http://incompliancemag.com/article/crosstalk-reduction-between-pcb-traces/, [Accessed 28 May 2017]

[12] Analog Devices, “Printed Circuit Board Design Issues”, http://www.analog.com/media/en/training-seminars/design-handbooks/Basic-Linear-Design/Chapter12.pdf, pp.12-1, [Accessed on: 28/6-2017]

[13] Analog Devices, “Printed Circuit Board Design Issues”, http://www.analog.com/media/en/training-seminars/design-handbooks/Basic-Linear-Design/Chapter12.pdf, p.12-1, [Accessed on: 28/6-2017]

[14] L. Bengtsson, “Elektriska Mätsystem och Mätmetoder”, Studentlitteratur 2012, p. 243

[15] L. Bengtsson, “Elektriska Mätsystem och Mätmetoder”, Studentlitteratur 2012, p. 243

[16] L. Bengtsson, “Elektriska Mätsystem och Mätmetoder”, Studentlitteratur 2012, p. 243

[17] T. DiCola, “Power Usage”, https://learn.adafruit.com/embedded-linux-board-comparison/power-usage, [Accessed 28 May 2017]

[18] Dimension Engineering, ”A beginner’s guide to switching regulators”, https://www.dimensionengineering.com/info/switching-regulators, [Accessed 28 May 2017]

[19] Linear Technology, LT1074/LT1076 Step-down Switching Regulator, http://cds.linear.com/docs/en/datasheet/1074fds.pdf, [Accessed 28 May 2017]

[20] B. Hauke, Application Report SLVA477B “Basic Calculation of a Buck Converter’s Power Stage”, http://www.ti.com/lit/an/slva477b/slva477b.pdf, p.1, [Accessed 28 May 2017]

[21] Henry J. Zhang, Application Note 140 ”Basic Concepts of Linear Regulator and Switching Mode Power Supplies”, http://cds.linear.com/docs/en/application-note/AN140fa.pdf, p.4, [Accessed 28 May 2017]

[22] ROHM Semiconductor, “Switching Regulator Series PCB Layout Techniques of Buck Converter”, http://rohmfs.rohm.com/en/products/databook/applinote/ic/power/switching_regulator/converter_pcb_layout_appli-e.pdf, p.1, [Accessed 28 May 2017]

[23] D. Molloy, “Exploring BeagleBone: Tools and Techniques for Building with Embedded Linux”, John Wiley and Sons Inc. 2015, p.26-27

[24] D. Molloy, “Exploring BeagleBone: Tools and Techniques for Building with Embedded Linux”, John Wiley and Sons Inc. 2015, p.33

[25] D. Molloy, “Exploring BeagleBone: Tools and Techniques for Building with Embedded Linux”, John Wiley and Sons Inc. 2015, p.33

[26] S. Ball, “Introduction to direct memory access”, http://www.embedded.com/electronics-blogs/beginner-s-corner/4024879/Introduction-to-direct-memory-access, [Accessed 28 May 2017]

[27] S. Ball, “Introduction to direct memory access”, http://www.embedded.com/electronics-blogs/beginner-s-corner/4024879/Introduction-to-direct-memory-access, [Accessed 28 May 2017]

[28] Texas Instruments, SPRS717J – “AM335x Sitara™ Processors”, http://www.ti.com/lit/ds/symlink/am3357.pdf, p.3, [Accessed 28 May 2017]

[29] S. Ball, “Introduction to direct memory access”, http://www.embedded.com/electronics-blogs/beginner-s-corner/4024879/Introduction-to-direct-memory-access, [Accessed 28 May 2017]

[30] Presonus, “The Truth About Digital Audio Latency”, http://www.presonus.com/learn/technical-articles/The-Truth-About-Digital-Audio-Latency, [Accessed 28 May 2017]

[31] J. Tranter, “Introduction to Sound Programming with ALSA”, Linux Journal, October 2004 [Online], Available: http://www.linuxjournal.com/article/6735, [Accessed 28 May 2017]

[32] J. Tranter, “Introduction to Sound Programming with ALSA”, Linux Journal, October 2004 [Online], Available: http://www.linuxjournal.com/article/6735, [Accessed 28 May 2017]

[33] alsa-project.org, “FramesPeriods”, https://www.alsa-project.org/main/index.php/FramesPeriods, [Accessed 28 May 2017]

[34] alsa-project.org, “FramesPeriods”, https://www.alsa-project.org/main/index.php/FramesPeriods, [Accessed 28 May 2017]

[35] D. Molloy, “Exploring BeagleBone: Tools and Techniques for Building with Embedded Linux”, John Wiley and Sons Inc. 2015, p.219

[36] D. Molloy, “Exploring BeagleBone: Tools and Techniques for Building with Embedded Linux”, John Wiley and Sons Inc. 2015, p.219

[37] devicetree.org, “Device Tree Specification Release 0.1”, https://github.com/devicetree-org/devicetree-specification-released/blob/master/devicetree-specification-v0.1-20160524.pdf, p.6, [Accessed 28 may 2017]

[38] J. Patrick, “Serial Protocols Compared”, http://www.embedded.com/design/connectivity/4023975/Serial-Protocols-Compared, [Accessed 28 May 2017]

[39] Philips Semiconductors, “I^2S bus specification”, https://www.sparkfun.com/datasheets/BreakoutBoards/I2SBUS.pdf, [Accessed 28 May 2017]

[40] J. Lewis, “Common Inter-IC Digital Interfaces for Audio Data Transfer, http://www.analog.com/media/en/technical-documentation/technical-articles/MS-2275.pdf, Analog Devices, [Accessed 28 May 2017]

[41] Picture, “AudioFormat.PNG”, http://processors.wiki.ti.com/index.php/File:AudioFormat.PNG

[42] J. Patrick, “Serial Protocols Compared”, http://www.embedded.com/design/connectivity/4023975/Serial-Protocols-Compared, [Accessed 28 May 2017]

[43] D. Molloy, “Exploring BeagleBone: Tools and Techniques for Building with Embedded Linux”, John Wiley and Sons Inc. 2015, p.55-56

[44] Robert C. Nelson, “ti-linux-kernel-dev”, https://github.com/RobertCNelson/ti-linux-kernel-dev, [Accessed 29 May 2017]

[45] Robert C. Nelson, “Linux on ARM, BeagleBone Black”, https://eewiki.net/display/linuxonarm/BeagleBone+Black#BeagleBoneBlack-BasicRequirements, [Accessed 29 May 2017]

[46] The Linux Information Project, “Source Code Definition”, http://www.linfo.org/source_code.html, [Accessed 29 May 2017]

[47] Wiki – Texas Instruments, “Sitar Linux Audio DAC Example”, http://processors.wiki.ti.com/index.php/Sitara_Linux_Audio_DAC_Example, [Accessed 29 May 2017]

[48] alsa-project.org, “Introduction”, http://www.alsa-project.org/main/index.php/Introduction [Accessed 29 May 2017]

[49] J. Tranter, “Introduction to Sound Programming with ALSA”, Linux Journal, October 2004 [Online], Available: http://www.linuxjournal.com/article/6735, [Accessed 29 May 2017]

[50] Spotify Support, “What bitrate does Spotify use for streaming”, https://support.spotify.com/us/using_spotify/search_play/what-bitrate-does-spotify-use-for-streaming/, [Accessed 29 May 2017]

[51] alsa-project.org, “Asoundrc”, https://www.alsa-project.org/main/index.php/Asoundrc, [Accessed 29 May 2017]

[52] alsa-project.org, “ASoC”, https://www.alsa-project.org/main/index.php/ASoC, [Accessed 29 May 2017]

[53] alsa-project.org, “DAPM”, https://www.alsa-project.org/main/index.php/DAPM, [Accessed 29 May 2017]

[54] alsa-project.org, “ASoC”, https://www.alsa-project.org/main/index.php/ASoC, [Accessed 29 May 2017]

[55] Wiki – Texas Instruments, “Sitar Linux Audio DAC Example”, http://processors.wiki.ti.com/index.php/Sitara_Linux_Audio_DAC_Example, [Accessed 29 May 2017]

[56] ASoC Machine Driver, “Machine DAI Configuration”, https://dri.freedesktop.org/docs/drm/sound/soc/ machine.html, [Accessed 29 May 2017]

[57] Wiki – Texas Instruments, “Sitar Linux Audio DAC Example”, http://processors.wiki.ti.com/index.php/Sitara_Linux_Audio_DAC_Example, [Accessed 29 May 2017]

[58] J. Tranter, “Introduction to Sound Programming with ALSA”, Linux Journal, October 2004 [Online], Available: http://www.linuxjournal.com/article/6735, [Accessed 29 May 2017]

[59] technick.net, “PCB Impedance and Capacitance of Microstrip”, https://technick.net/tools/impedance-calculator/microstrip/, [Accessed 31 May 2017]

[60] B. Molin, “Analog elektronik”, Studentlitteratur 2013, p.495-497

[61] B. Hauke, Application Report SLVA477B “Basic Calculation of a Buck Converter’s Power Stage”, http://www.ti.com/lit/an/slva477b/slva477b.pdf, p.1, [Accessed 28 May 2017]

Bootloader and Kernel Sources:

[S1] Linaro Compiler, “Toolchain Binaries”, http://www.linaro.org/downloads/

[S2] Bootloader, “Das U-boot”, https://github.com/u-boot/u-boot

[S3] Kernel, https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git

[S4] Filesystem, “Debian”, https://www.debian.org

Appendix

A.1 Hardware peripherals and connectors on BBB used in the project include:

- Expansion headers – I^2S, SPI, General Purpose In/Out, 5V DC rail, Digital Ground, Power and Reset
- Network – 10/100 RJ45 Ethernet connector
- Serial Debug Interface – a 6 pin connector to enable serial console connection with the BBB
- SD Card Slot – used to boot the configured Linux kernel
- External 24.576MHz crystal oscillator – clock input to the McASP (I^2S) peripheral

Hardware peripherals and connectors placement on BBB:

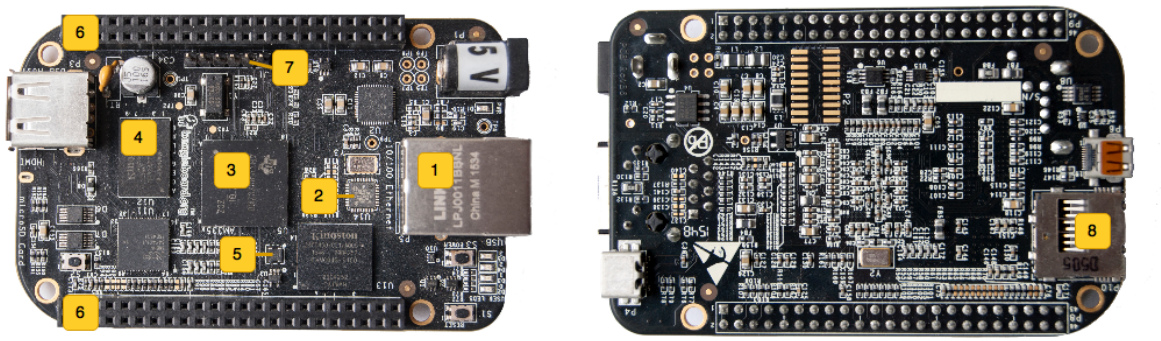


Figure A.1 - BeagleBone Black hardware

	Function	On-Board Physical
1	Network	RJ45 Ethernet Connector
2	Ethernet Processor	LAN8710a PHY
3	Microprocessor	Sitara AM335x Cortex-A8
4	Memory	512MB DDR3
5	Oscillator	24.576 Oscillator
6	Expansion Headers	Two 2x23 Female Headers
7	Serial Debug Connector	1x6 Male TTL3V3 Cable Connector
8	SD-Card	Micro-SD Slot

Table A.1 - Figure A.1 legend

A.2 BeagleBone Black P9 header pinout table.

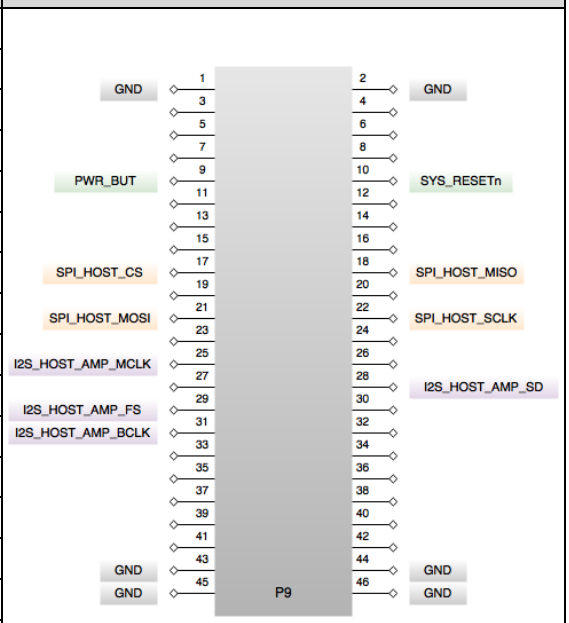
Pin	Assigned Function	
P9_1	GND	
P9_2	GND	
P9_9	PWR_BUT	
P9_10	SYS_RESETn	
P9_17	SPI_HOST_CS	
P9_18	SPI_HOST_MISO	
P9_21	SPI_HOST_MOSI	
P9_22	SPI_HOST_SCLK	
P9_25	I2S_HOST_AMP_MCLK	
P9_28	I2S_HOST_AMP_SD	
P9_29	I2S_HOST_AMP_FS	
P9_31	I2S_HOST_AMP_BCLK	
P9_43	GND	
P9_44	GND	
P9_45	GND	
P9_46	GND	

Table A.2 – BBB P9 expansion header and legend

B.1 PCB Microstrip Characteristic Impedance [59] [60]

As mentioned in section 3. of this paper, any PCB microstrip have a characteristic impedance Z_0. This characteristic impedance is defined as the square root of the series inductance L_0 over the parallel capacitance C_0:

$$Z_0 = \sqrt{\frac{L_0}{C_0}}$$

Equation B.1

In the case of the PCB microstrip the characteristic impedance Z_0 can be expressed as:

$$Z_0 = \frac{87}{\sqrt{\varepsilon_r + 1.41}} ln\left[\frac{5.98 \cdot H}{(0.8 \cdot W) + T}\right]$$

Equation B.2

This expression is based on, as expected, the geometry of the microstrip conductor and the permittivity constant of the dielectric that is used to manufacture the PCB. Moreover, the electric field surrounding the microstrip will exist outside of the dielectric material and hence experience an effectively lower than normal permittivity due to the design of the PCB. Figure B.1 illustrates the cross section of a PCB.

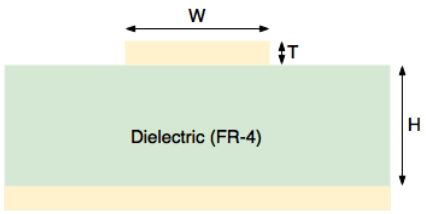


Figure B.1 - Cross Section of PCB

B.2 Host PCB layout.

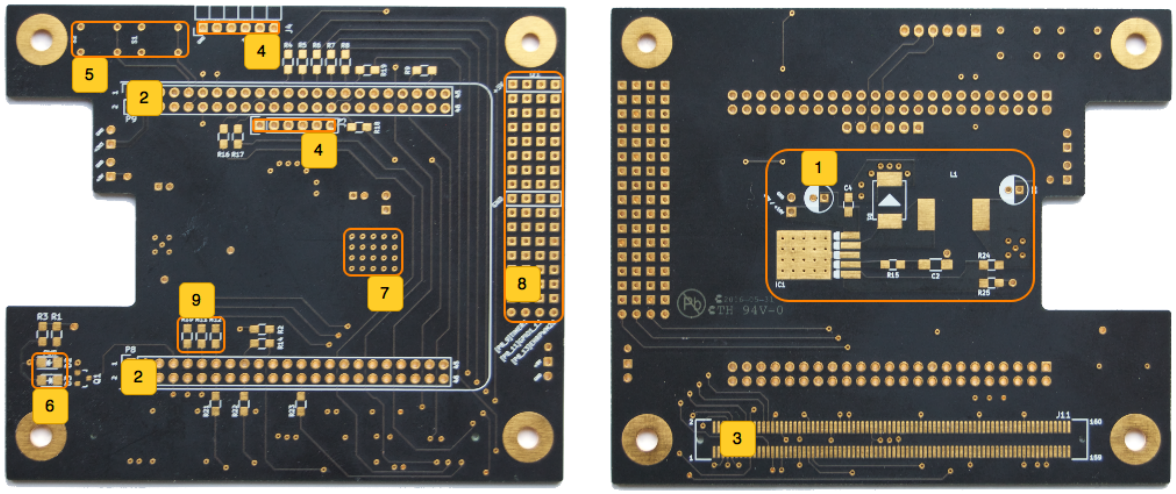


Figure B.2 - PCB Layout

	Function	On-Board Physical
1	Power Supply	LT1076CQ5 Power Stage
2	Expansion Headers	Two 2x23 Male Headers
3	Host-Amp Connector	2x80 Pin Connector
4	Serial Debug Headers	One 1x6 Female Header / One 1x6 Male Header
5	On/Off | Reset	Two Tactile Switches
6	System/Power Active	Two LEDs
7	Regulator Heat Sink	Thermal Vias
8	Expansion/Experiment Surface	2.54mm Through Hole Matrix

Table B.1 - Figure B.2 legend

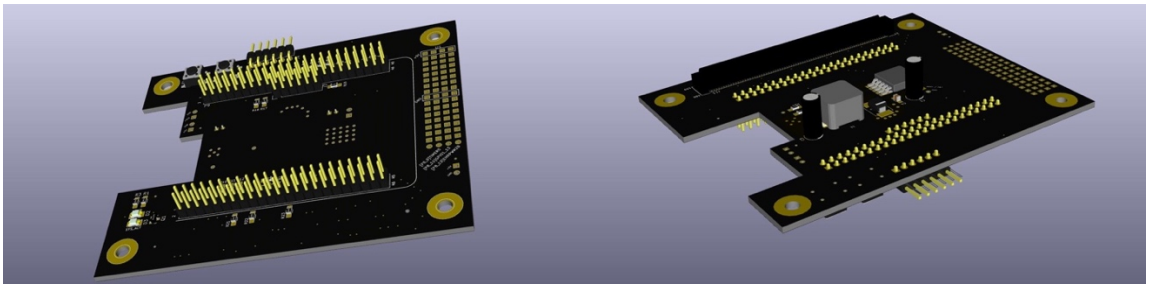


Figure B.3 - 3D rendition of the designed PCB

B.3 Power Stage Calculations [61]

Maximum Duty Cycle
The maximum duty cycle *D* is derived from the following formula:

$$D = \frac{V_{OUT}}{V_{IN(\max)} \cdot \eta} \cdot 100\% \Longrightarrow \frac{5}{16 \cdot 0.9} \cdot 100\% \approx 35\%$$

Equation B.3

Inductor Ripple Current
When the maximum duty cycle has been decided the inductor ripple current can be calculated. To determine the inductor ripple current, the inductor value is necessary. This value, or range of values appropriate for the specific IC, is often given in the component data sheet. For the IC LT1076CQ-5 the recommended inductor value is 100uH, this results in a ripple current of:

$$\Delta I_L = \frac{(V_{IN(\max)} - V_{OUT}) \cdot D}{f_S \cdot L} \Longrightarrow \frac{(16-5) \cdot 0.35}{90 \cdot 10^3 \cdot 100 \cdot 10^{-6}} \approx 43mA$$

Equation B.4

(Where f_s is the minimum switching frequency of the converter IC.)

IC Maximum Output Current
At this stage, the maximum output current of the IC can be determined.

$$I_{MAXOUT} = I_{LIM(\min)} - \frac{\Delta I_L}{2} \Longrightarrow 2.6 - \frac{0.043}{2} \approx 2.57A$$

Equation B.5

The maximum theoretic output current of the IC is 2.57A, this current value is well above the specified requirements. The results above prove to be sufficient and the maximum switch current surrounding components have to withstand can be approximated:

$$I_{SW(max)} = \frac{\Delta I_L}{2} + I_{OUT(max)} \Longrightarrow \frac{0.043}{2} + 2 \approx 2.2A$$

Equation B.6

Inductor Value
In this case, the inductor value was chosen according to recommendations in the IC data sheet. However, when there are no recommendations to inductor value, appropriate inductor selection can be made using the following equations:

$$L = \frac{V_{OUT} \cdot (V_{IN} - V_{OUT})}{\Delta I_L \cdot f_S \cdot V_{IN}}$$

$$\Delta I_L = (0.2\ to\ 0.4) \cdot I_{OUT(max)}$$

Equation B.7 & B.8

The inductor estimation equation above requires knowledge of the ripple current output by the switched regulator into the inductor and the ripple current cannot be calculated when the inductor value is not known. As a rule of thumb, inductor ripple current falls in the range of 20% - 40% of the output current.

Rectifier Diode Value

The rectifier diode needs to be chosen based on its forward current rating. In general, Schottky diodes are preferred as they reduce losses. The diode forward current I_F rating necessary was:

$$I_F = I_{OUT(max)} \cdot (1 - D) \Longrightarrow 2 \cdot (1 - 0.35) = 1.3A$$

Equation B.9

Setting the Output Voltage

Generally, voltage converters set output voltage using a resistive divider circuit, and so also in this case. The LT1076CQ datasheet specifies a feedback bias current I_{FB} of 0.5uA flowing into the Feedback/Sense pin and according to the application report `Basic Calculation of a Buck Converter's Power Stage`[61], the current to ground through the voltage divider circuit must exceed the feedback bias current I_{FB} flowing to the Feedback/Sense pin on the converter IC by a minimum factor of 100. Assuming a voltage divider current $I_{R1/2}$ of 1mA satisfies the above stated condition. Furthermore, the feedback voltage V_{FB} on the Feedback/Sense pin must be set at 2.21V as specified in the datasheet. Based on these conditions the resistors R_1 and R_1 part of the voltage divider network can be calculated:

$$R_2 = \frac{V_{FB}}{I_{R1/2}} \Longrightarrow \frac{2.21}{0.001} = 2.21k\Omega$$

$$R_1 = R_2 \cdot \left(\frac{V_{OUT}}{V_{FB}} - 1\right) \Longrightarrow 2.21 \cdot 10^3 \cdot \left(\frac{5}{2.21} - 1\right) \approx 2.8k\Omega$$

Equation B.10 & B.11

Input and Output Capacitor Selection

The values for input and output capacitors are in general given in the datasheet. However, with regards to external compensation of output capacitance the equation below can be used. The calculations are based on knowledge of previous calculated parameters such as the estimated inductor ripple current ΔI_L, the minimum switching frequency f_S (datasheet) and the desired output voltage ripple ΔV_{OUT}. Assuming a desired output voltage ripple of 0.1mV, the minimum output capacitance $C_{OUT(\min)}$ then is

$$C_{OUT(\min)} = \frac{\Delta I_L}{8 \cdot f_S \cdot \Delta V_{OUT}} \Longrightarrow \frac{0.043}{8 \cdot 90 \cdot 10^3 \cdot 0.1 \cdot 10^{-3}} \approx 600\mu F$$

Equation B.12

C.1

The BeagleBone Black relies on several device tree (DT) files to boot in a specific configuration. The device tree files can be divided into two main categories;

*.dtsi – DT platform code
*.dts – DT board code

these two categories of device tree files are later compiled with the help of a Device Tree Compiler (DTC) to form a Device Tree Blob (DTB) and it is this DTB that forms a part of the boot sequence/files to set the eLinux system in the specified configuration. Illustrated in figure C.1 below.

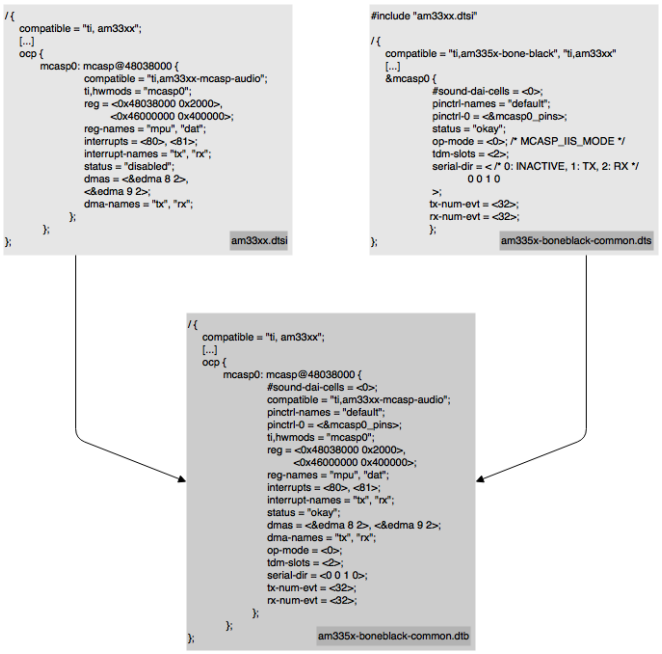


Figure C.1 - .dtsi + .dts = .dtb

C.2 am335x-boneblack.dts

Include, *root node* (/), *model* and *compatibility* declarations are stated first. In the case of the BBB, the McASP peripheral requires an external clock input to produce the correct clock frequencies for the 48kHz sample rate family. This is described in figure C.2 below.

```
#include "am33xx.dtsi”
#include "am335x-bone-common.dtsi"

/ {
        model = "TI AM335x BeagleBone";
        compatible = "ti,am335x-bone-black", "ti,am335x-bone", "ti,am33xx";

        clk_mcasp0_fixed: clk_mcasp0_fixed {
                #clock-cells = <0>;
                compatible = "fixed-clock";
                clock-frequency = <24576000>;
        };

                clk_mcasp0: clk_mcasp0 {
                #clock-cells = <0>;
                compatible = "gpio-gate-clock";
                clocks = <&clk_mcasp0_fixed>;
                enable-gpios = <&gpio1 27 GPIO_ACTIVE_HIGH>;
        };
};
```

Figure C.2 - Root node, compatibility statement and clock initialization

Following the root node, compatibility statement and clock initialization the am335x_pinmux node declared in *am335x.dtsi* is referenced. This allows for project specific pinmuxing and in this case the initialization of the McASP and SPI pins as shown in figure C.3.

```
&am33xx_pinmux {
        lpcm_audio_pins: lpcm_audio_pins {
                pinctrl-single,pins = <

                        0x1ac (PIN_INPUT_PULLDOWN | MUX_MODE0)
                        0x190 (PIN_OUTPUT_PULLDOWN | MUX_MODE0)
                        0x194 (PIN_OUTPUT_PULLDOWN | MUX_MODE0)
                        0x198 (PIN_OUTPUT_PULLDOWN | MUX_MODE0)
                        0x1a8 (PIN_OUTPUT_PULLDOWN | MUX_MODE0)
                        0x19c (PIN_OUTPUT_PULLDOWN | MUX_MODE2)
                        0x1a4 (PIN_OUTPUT_PULLDOWN | MUX_MODE2)
                        0x06c (PIN_OUTPUT_PULLUP | MUX_MODE0)
                >;
        };
        lpcm_spi0_pins: lpcm_spi0_pins {
                pinctrl-single,pins = <
                        0x150 (PIN_INPUT_PULLUP | MUX_MODE0) /* */
                        0x154 (PIN_INPUT_PULLUP | MUX_MODE0) /* */
                        0x158 (PIN_OUTPUT_PULLUP | MUX_MODE0) /* */
                        0x15c (PIN_OUTPUT_PULLUP | MUX_MODE0) /* */
                >;
        };
};
```

Figure C.3 - am33xx_pinmux node

Likewise, to initialize and apply the pinmux declarations above on the McASP and SPI peripherals, the *mcasp0* and *spi0* nodes declared in *am335x.dtsi* are referenced to configure and link the SoC peripherals.

```
&mcasp0 {
                                        pinctrl-names = "default";
                                        pinctrl-0 = <&lpcm_audio_pins>;
                                        status = "okay";
                                        op-mode = <0>;                          /* McASP_I2S_Mode */
                                        tdm-slots = <2>;                /* Was 2 */
                                        num-serializer = <4>;
                                        serial-dir = <                          /* 0: INACTIVE, 1: TX, 2: RX */
                                                        1 1 1 1
                                        >;
                                        tx-num-evt = <4>; /* Was 32 */
                                        rx-num-evt = <1>; /* Was 32 */
};

&spi0 {
                                        #address-cells = <1>;
                                        #size-cells = <0>;
                                        pinctrl-names = "default";
                                        pinctrl-0 = <&lpcm_spi0_pins>;
                                        status = "okay";
                                        spidev@0 {
                                                spi-max-frequency = <24000000>;
                                                reg = <0>;
                                                compatible = "linux,spidev";
                                        };
};
```

Figure C.4 - mcasp0 and spi0 referenced nodes

Finally, nodes to allow for the device tree content in the codec and machine driver to be called are declared. This is illustrated in figure C.5 below.

```
/ {

        lpcm: lpcm {
                compatible = "ti,lpcm";
        };

        sound {
                        compatible = "ti,lpcm-evm-audio";
                        ti,model = "LG LPCM";
                        ti,audio-codec = <&lpcm>;
                        ti,mcasp-controller = <&mcasp0>;
                        ti,codec-clock-rate = <24576000>;
        };

};
```

Figure C.5 - Device tree content in codec and machine driver