Master thesis

AUDIO OVER ETHERNET USING EMBEDDED SYSTEMS

BY EMIL OHLSSON AND ERIK LUNDH

www.svep.se

Abstract

The last couple of years there has emerged many products for streaming music via Ethernet, where the home computer can often be considered to be a hub in the home entertainment system. Products like *Spotify* stream the music of choice down to the computer while products like *Sonos* and *AirPlay* make it possible to stream music from the computer out to Ethernet connected playback nodes. However, these systems are often expensive or limited to specific computer programs. This thesis studies the different methods of streaming audio and describes the design process for creating a platform for receiving audio over Ethernet. The implemented platform is fully capable of streaming sound, with the exception of minor playback issues. Different solutions to the playback issues are discussed and described.

Table of Contents

1 Introduction

1.1 Background

Recently several home entertainment systems for streaming audio have emerged on the market. The audio is streamed from databases existing in *the cloud* to the consumers home computers and then from the computers out to the speakers. The possibility to stream audio to several rooms at the same time has become a big selling point. However, many of the systems for streaming audio throughout the home are associated with problems concerning high cost and flexibility.

1.2 Aim of this thesis

This thesis is a comparative study of some of the existing systems for streaming audio throughout the home, such as PulseAudio, DLNA, Sonos and AirPlay, as well as a case study about implementing a listening node to such a system. The implemented node should be able to act as an output device to be used with PulseAudio, and it will be referred to as Svep Networking Audio Platform, or **SNAP**[1]. The SNAP should be capable of receiving a digital audio signal via Ethernet and render it to an electrical audio signal.

PulseAudio is the audio mixer that is used in most modern GNU/Linux distributions; it acts as a virtual layer between the sound card and the application playing the audio stream. This virtual layer can stretch over several computers via Ethernet which makes it possible for a computer to share its sound card with other computers.

The SNAP should be simple and cheap, use standard components and have Ethernet capabilities. It should run firmware that makes it act as a virtual sound card for PulseAudio. This way it is possible to use the existing functionality of PulseAudio to control the audio stream transmitted to the node. SNAP should also be energy efficient to make it possible to power the SNAP using Power over Ethernet, or **PoE**, which makes it easy to set up since both power and data communication can be delivered through an Ethernet cable.

1 As the reader may have noticed some words are highlighted, this is only used on the first use of the word and it indicates that the word is described in the glossary.

2 Method

This chapter is an overview of some of the existing products for streaming audio, as well as serve as an theoretical background to the thesis project. There are several areas that needs to be considered, but it is not necessary to go into detail in every field. Everything discussed have been used in some way during the thesis project.

Since computer networks are digital communication mediums, it is necessary to have a digital representation of the sound stream. The two main categories of digital audio are compressed and uncompressed audio. Normal computer networks have some inherit flaws in terms of delay and data loss, but there are several protocols that solve these problems. To be able to communicate audio via a computer network there must be some agreed upon way to send the audio, and in addition it is useful to use helper protocols that can describe or control the audio stream. Apart from the audio communication there must also be means for addressing the devices via the network, and this is in itself a complex task. One aim of the thesis is to study some of the existing products for audio streaming, and see what is useable in the case study. In order to design a hardware platform it must first be decided how the audio should be rendered, and what technique to use to. The different components must have means for communication and all the components must be powered. Finally, there must be some way to send the sound from the computer to the network.

2.1 Theory

2.1.1 Digital coding of audio

From an electrical point of view audio can be regarded as a continuous analog signal, and in order to make a digital representation of this signal it must be encoded. The standard way to do this in audio coherence is a technique called pulse code modulation, or **PCM**. The encoding is done by sampling the signal at constant sampling frequency. At each sample time the signal is quantified and stored as a digital value [1]. There are several techniques used to quantify the signal, but the focus here is on linear representation, or **LPCM**, as it is the standard variant of PCM. The fidelity of the encoding is determined by the sample rate and the resolution of the sample, e.g., the term CD-quality refers to a LPCM encoding with a sample rate of 44.1 kHz using 16-bit samples [2]. CD-quality with stereo sound gives a bit stream of 1 411 200 bit/s or roughly 1.4 Mbit/s.

2.1.2 Audio compression

There are mainly two categories of audio compression, *lossy* and *lossless*. The names of the categories refer to the fact that decoding of a lossless compressed stream results in a bit by bit copy of the original stream, whereas the lossy does not. Lossy formats include the **MP3** encoding, and lossless include the **FLAC** encoding. A compressed sound stream requires less bandwidth than an uncompressed stream. However, the downside of compressed streams is that they require more computing time than uncompressed streams, since they need to be decompressed before playback. This is not a problem on a normal home computer, but for an embedded system this could be a problem when performing other tasks such as controlling peripheral devices.

MP3 is an abbreviation of **MPEG**-1/2 layer 3 coding, which is a standard made by MPEG, and it is a highly effective compression algorithm. For example, an uncompressed CD audio stream of 1.4 Mbit/s can be encoded into a 192 kbit/s stream with minor or no audible loss, giving a compression ratio of 7:1 [3]. The compression is based on the effect of auditory masking, a limitation in the human hearing, e.g., sounds with higher frequencies can be masked by sounds with lower frequencies.

The specification of a "*high accuracy MPEG-1*" decoding process is very well defined, and done in a way that there can be no audible differences between compliant decoders. This, however, does not apply to the encoding process which merely defines that a model of the human hearing should be used. Decoding an MP3 stream is a process that requires a lot of computational power, e.g., one implementation using an ARM7 microprocessor took somewhere between 16ms and 21ms to decode a 26ms long MPEG frame [4] [5]. This doesn't give much time left to perform other tasks.

Free Lossless Audio Codec, FLAC, is an open source **codec** used to compress audio data without losing any information. The method is based on the fact that the source is a sound stream. The compression is performed by storing the difference between a mathematical estimate of a sample, which is based on the previous samples, and the real value using a coding technique called Golomb-Rice codes [6]. Golomb-Rice codes have the property that coding of small values results in shorter codewords [7], i.e., small values doesn't use as many bits as large values. This method of compression is more efficient than using a generic compression. A small comparison can be made using part of the song Hells Bells by AC/DC. When compressing using *zip*, a generic compression tool, the compression was 9% and when using FLAC the rate was 52.7%. The simplicity of the algorithm makes it easy to implement in electronic devices [8].

2.1.3 Problems with streaming audio

When sending data via **Ethernet** [9] the data is split into several discrete packets, often called datagrams, and on larger networks the packets may take different routes, resulting in an out-of-order arrival. There is a small probability that packets are discarded along the way, due to congestion or faulty transmissions. Streaming audio is a task with real time characteristics, which means that there are limitations on acceptable delays as well as vulnerability against dropped packets. Dropped packets in audio coherence will introduce small gaps in the audio stream, where each packet can be up to 8.5ms of audio when streaming in CD-quality. This is because a datagram may not be larger than a maximum transmission unit, which for Ethernet is 1542 bytes, including overhead data such as packet headers. A loss of 8.5ms audio will result in an audible artifact, like a click or a pop.

In a network there isn't infinite bandwidth, and the link is shared with other applications, for example; the typical theoretical bandwidth of wired Ethernet is either 100 Mbit/s or 1 Gbit/s. Compressing the stream lowers the needed bandwidth, but with the tradeoff that the stream must be decoded before it can be played. If the same stream is meant to be received by several nodes at the same time, then the data could be sent using **multicast**, this uses the bandwidth of one stream but the data can be received by all listening nodes [10].

If several sources, i.e., speakers, play the same sound but with a delay in between them, the listener will perceive the sound different compared to if they were synchronized. For example, if the delay is longer than ~30ms it will be perceived as an echo; for shorter delays, e.g., 1-15ms, the perception of the origin of the sound will be changed [11]. The delays in an Ethernet connection can vary depending on the load, type of router or switch, size of network etc.; which means it is not possible to estimate the delays. The delay between two nodes however, can very easily be measured using an application called *ping*. In a wired home network packet loss usually isn't a problem, but in a more unreliable network like wireless networks this is a problem. A solution to an unreliable link is transmission control, this means that all data must be acknowledged, and if it's not acknowledged it will need to be retransmitted. Using transmission control requires more overhead, as well as possible delays when waiting for lost packets.

2.1.4 Techniques related to streaming audio

There are several ways of dealing with the problems that a normal Ethernet introduces. In this section some of the relevant solutions to the problems listed above will be described.

Network Time Protocol, **NTP**, defined in **RFC** 5905 [12], is a protocol for synchronizing clocks on different computers on the Internet. NTP consists of primary servers, secondary servers and clients, where the primary server is synchronized to a reference clock, e.g., **GPS**. Clients and secondary servers synchronize to an upstream server, such as the primary server or another secondary server. A secondary server provides synchronization for downstream clients or servers. This is illustrated in Illustration 1. During synchronization several packets are sent between the hosts, where the current time stamp is sent, and the traversing time is calculated and corrected from the timestamps used during communication [13]. Accuracy down to 100μs is achievable in a local area network, and up to 10ms over the Internet [14].

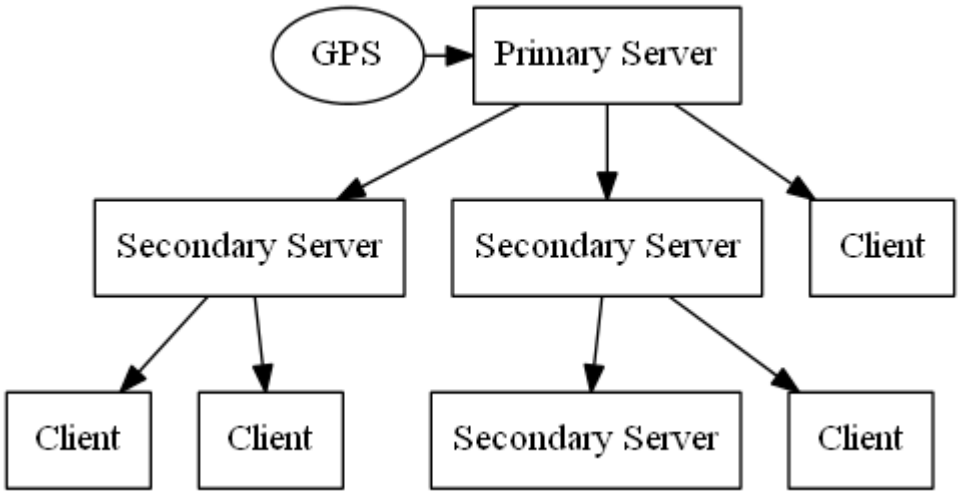

Illustration 1: An example of a NTP net

The Precision Time Protocol, **PTP**, or **IEEE** 1588 [15] is a high precision time protocol for synchronization in a local area network. PTP is much like NTP but scaled down for a smaller network making it able to acquire a higher accuracy, i.e., accuracy down to less than 1μs, at the cost of scalability [16].

The Global Positioning System, GPS, mainly used for positioning, was created by the U.S. Department of Defense and it is now maintained by the U.S. government. GPS uses a very precise clock for determining the position of the receiver, the same clock can also be used to determine time down to 1ns [17]. Using GPS as a time reference means adding extra hardware, which might not be an option in an embedded system.

The Real-time Transport Protocol, **RTP**, defined by RFC 3550 is a protocol used to deliver real time data over internet. By itself RTP is only a protocol which packages data and tags it with a time stamp and a synchronization source. Packets are not synchronized network wide, but it is possible to determine whether a packet is received out of order or if a packet is dropped [18]. There are several RFCs that describe different kind of profiles for RTP, such as RFC 3551 that describe how to use RTP in applications like audio and video conferencing. Another RFC of interest is RFC 5691 which describes how to stream multichannel MPEG sound. In order to create additional functionality there are several protocols that add sessions, synchronization etc. to multimedia RTP streams.

The Real Time Streaming Protocol, **RTSP**, is a protocol for controlling a multimedia server [19]. A multimedia server refers to a node that delivers a multimedia stream. RTSP does not carry the stream itself; it is rather a protocol for controlling the media playback. This is done by sending commands like *play*, *pause* or *mute*. The protocol is described in RFC 2326.

The Remote Audio Output Protocol, **RAOP**, is a protocol that combines RTP and RTSP in order to stream audio via internet [20]. It was originally developed by Apple Inc. to be used with their **AirPlay** system. This creates an encrypted audio stream between two network nodes that automatically discover each other

using the **ZeroConf** protocol suite. The encryption key is of course secret, so it cannot be used without the permission from Apple Inc. But it has been decoded, so there are open source implementations that can use RAOP. The use of these implementations can be considered to be a legal gray area.

The Session Description Protocol, **SDP**, and the Session Announcement Protocol, **SAP**, is a way for a server to announce and describe a multimedia session. The description is packed into SAP packets, and it is designed to include all the information needed to start playback. A session is announced by sending the packet via multicast to the same group as the multimedia stream is intended to be streamed to. This is further described later.

The techniques described here are useful when implementing a device capable of receiving an audio stream via Ethernet. Not all of them will be used in the implementation, but they serve as a foundation for making the necessary design decisions. This is further described in chapter three.

2.2 Existing systems for audio streaming

Digital Living Network Alliance, **DLNA**, is a collection of companies that collaborate with the goal to create compatible devices and thus enhance the user experience [21]. DLNA uses a normal home network, both wired and wireless, to connect the different devices. DLNA use a set of formats, i.e., Image, Audio, Video and Container. Most of the formats are proprietary, and the combinations of them are called profiles [22]. DLNA has three main device classes [23]; Home Network Devices, Mobile Handheld Devices and Home Infrastructure Devices, where Home Network Devices is composed of five device types.

- Digital Media Server, **DMS**, stores content and makes it available, e.g., **PC** and **NAS**.
- Digital Media Player, **DMP**, finds contents on a DMS and is able to render and playback it, e.g., TV and Stereo.
- Digital Media Renderer, **DMR**, plays content received from a DMC, e.g., TV and remote speakers.
- Digital Media Controller, **DMC**, finds content on a DMS and plays it on a DMR, e.g., remote, tablet and smart phone.
- Digital Media Printer, **DMPr**, prints content sent from DMCs or DMPs, e.g., Photo printer.

A device can be DLNA certified, this means that the device comply to the guidelines agreed on by DLNA and the device is capable of seamless integration with other DLNA certified devices.

Sonos is an proprietary system, from Sonos Inc. A Sonos system consists of nodes, which are either Zoneplayers or Zonebridges. These are connected to each other through a wireless mesh-network, Sonosnet, and control of the system is done by a remote service, which is either a Sonos remote or a smartphone application. The system needs an internet connection through the combined wired and wireless node, i.e., the Zonebridge, which can also be used for extending the system via a wired LAN [24].

AirPlay is a proprietary protocol from Apple Inc. that uses a regular home network for streaming multimedia. Airplay is a more generic variant of RAOP, since it allows video streaming as well [25]. There are two types of devices; senders, i.e., any devices running iTunes, and receivers, e.g., Apple TV or Airport. The receiver is only a passive unit which listens to and renders the multimedia stream [26].

Cobranet is a proprietary system from Cirrus Logic intended for large scale installations like stadiums and airports. It is intended to be used as a replacement for older analog audio systems. Cobranet uses a standard Ethernet network, but uses its own protocol to send up to 64 channels of uncompressed audio in one Ethernet cable, which must be at least Cat-5. Cobranet is capable of sending data both **unicast** and multicast and is able to synchronize playback [27].

PulseAudio is a software mixer for POSIX [28] compliant operating systems [29]. It acts as a proxy between media playing applications and the sound card. This virtual layer between application and hardware enables a lot of advanced features such as volume control for separate applications and the ability to control what output device to associate with different applications. This virtual layer can also stretch to other computers via Ethernet. The functionality of PulseAudio is split into several modules [30]; and there are modules for streaming sound using DLNA, RTP multicast, RAOP and so on [31]. Since PulseAudio is open source it is possible to implement new modules adding additional functionality. As well as modular support for a wide variety of different streaming formats there is also support for a PulseAudio native protocol which is based on **TCP**. PulseAudio has support to stream audio both using RTP multicast as well as stream via its native TCP based protocol.

It is useful to have an overview of the existing products capable of streaming audio, since it is possible to comply to an existing method and not have to reinvent the wheel. For example, by complying to the method PulseAudio uses for audio streaming it is possible to make use of the existing infrastructure of PulseAudio. This is further discussed in chapter three.

2.3 Amplifier topologies

A traditional analog amplifier can be seen as an adjustable voltage regulator, where the excess voltage is dissipated as heat in the active device, which for example can be a bipolar transistor or a **MOSFET.** For example, **Class AB,** a standard analog amplifier for audio use, has a maximum theoretical efficiency of approximately 78% during optimal conditions using ideal components and at full power [32]. This doesn't take into consideration the **dynamic range** of music, the reactive load of the speaker etc, which are both factors that reduce the efficiency of the amplifier. As the excess power will be dissipated as heat, a lower efficiency will generate more heat demanding larger cooling. To be able to generate the same amount of output power a larger power supply is needed, which adds up to an increased cost and size of the system [33]. In low power applications this might not be a problem, but for a higher power system this will be a factor that needs to be considered. An amplifier with higher efficiency is the **Class D** amplifier which, instead of linearly amplifying the signal, converts the signal to a Pulse Width Modulated signal, or **PWM** signal, which is then amplified and converted back to an analog signal using a low-pass filter. Generation of the PWM signal can be made by comparing the analog signal to a triangular wave using a comparator circuit, which will give the PWM as an output. Since the PWM signal have discrete levels, being either on or off, the only losses in the amplifying stage are the resistance of the active device, this gives the amplifier an efficiency of around 80-90% [35]. There are two types of Class D modulations, where the newer modulation technique, BD modulation, gives a smaller current through the low-pass filter. This reduces the emitted noise, giving the option of completely discarding the filter, since the speaker will act as a filter if the speaker wiring is short, i.e., less than 20 cm [36]. In Appendix A the different modulation techniques are graphically represented.

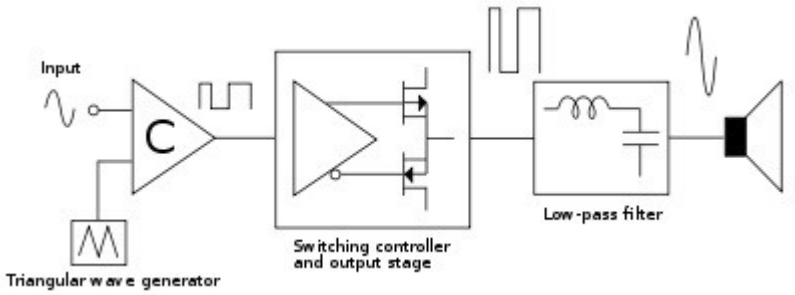

Illustration 2: Class D amplifier block diagram [34]

The knowledge about amplifiers is useful when designing the hardware that should render the digital audio stream to an audible audio stream. Since the amplifier circuitry have high impact on the power consumption of the SNAP, this is especially interesting. More about this in chapter three.

2.4 Communication Interfaces

For communication between peripheral devices, e.g., sending commands or audio data, some sort of interface is needed. The interface specifies how the electrical bus is constructed and defines the data protocol. A communicating device is often defined as either a Master or a Slave unit, where the master is controlling the data flow. How the devices are placed on the bus varies, from all being parallel with different addresses to being in series, also known as Daisy-chained**.**

Inter Integrated Circuit, **I2C** [37], is a bus originally created by Philips for communication between a micro controller and peripheral devices. The bus consists of only two wires, clock and data and it is 8-bit oriented and bidirectional. The devices are all connected **open drain** to the bus while the bus is pulled up by resistors. The use of open drain gives the capability for fast acknowledgments. Since each device connected to the bus adds some capacitance, it is possible to add as many devices as needed as long as the pull up resistors is able to drive a valid clock. Each device has its own address for identification during communication and every received byte is acknowledged. The base speed is 100kbit/s, while the standard has been extended to 400kbit/s, 1Mbit/s and 3.4Mbit/s speeds.

Integrated Interchip Sound, **I2S** [38], also created by Philips, is a protocol for sending digital audio data between **IC**s. The bus consists of three wires, clock, data and word select. Since sound streams generally has multichannel nature, it is useful to have support for this built into to the communication. I2S uses the word select signal to indicate channel. Another feature of I2S is that it is flexible in terms of clocking, making it useful for any sampling frequency available.

Serial Peripheral Interface, **SPI** [39], was created by Motorola for peripheral communication. SPI is capable of operating in full duplex and uses four wires, clock, master output/slave input, master input/slave output and slave select. Multiple devices can be connected to the bus, either in parallel where every slave select pin needs to be controlled on its own, or Daisy-chained where the data output on a slave is connected to the input on the next device. As the speed of SPI is not limited or predefined, the speed is only limited by the capabilities of the connected devices.

Joint Test Action Group, **JTAG**, or IEEE 1149.1-2001 is a protocol for performing boundary scans or observing and modifying behavior on assembled products. This enables the developer to test and debug the firmware in an embedded system. It can also be used to download the firmware to the embedded system [40].

The communication protocols described here will serve as a knowledge base when describing the hardware design in chapter three.

2.5 Power over Ethernet

Power over Ethernet, **PoE**, is used for powering devices via an Ethernet; e.g., **WLAN** Access Points, Network Cameras or **IP** Telephones. PoE is defined in the IEEE-standard 802.3at [41], and contains two different classes of devices; Power Sourcing Equipment, **PSE**, and Powered Device, **PD**. The PSE, which usually is a switch or a passive injector, supplies power by applying a DC voltage on two pairs of the data wires. The PD receives the power by filtering it out from the data pairs and rectifying it. According to the standard the PD should galvanically separate the PoE voltage from its internal voltages, and this can be done by having a switched DC/DC converter for the internal voltages. To ensure that the PSE doesn't supply power to a non-PoE device a start-up sequence is required to determine if the device is compatible and what power requirements it has. This is done by the PSE applying a small voltage on the data pairs and measuring the

drawn current, the PD identifies itself by drawing a typical current depending on its power requirement; if the device is not compatible it will not draw the required current and the PSE will not apply power. In the original standard, 802.af, the supplied power to a device is up to 15.4W, but it has been extended in the newer versions to provide up to 25.5W.

PoE is an interesting aspect to consider when designing low powered devices that are connected to an Ethernet, since makes it possible to power the device only using the Ethernet cable and removing the need for an external power supply unit. Since this is exactly the kind of device this thesis is about it will serve as a option in the hardware design, which is described in chapter three.

2.6 Automatic configuration of Ethernet devices, an overview

In order to use a network connected device there are basically four steps of configurations that should be done; these steps must be configured, or agreed upon, in the order below [42].

1. *Addressing*
   The device must by some mean acquire an IP address. This is needed because otherwise there is no way to direct data flow to the device.
2. *Name resolution* (Optional).
   The device should be given a name so the end user does not have to use the IP address when addressing the device. This is more of a convenience for the end user to be able to address the device using a descriptive name that can be resolved to an IP address.
3. *Device and service discovery*
   The rest of the network should know that the device is there and what kind of services it is offering, e.g., the device is a web server.
4. *Application*
   The rest of the network must know how to communicate with the device, e.g., if the device is a web server it uses **HTTP** to communicate.

This can be done by either manually configuring the device and the network, or the device has to comply with some kind of auto configuration standard. Unfortunately there is not one of those auto configuration standards, there are several. The two major actors on the market are Universal plug and play, **UPnP**, and zero configuration networking, ZeroConf. Both standards define a suite of protocols that each solves one of the steps listed above. Furthermore there are additional protocols that solve subsets of the tasks above.

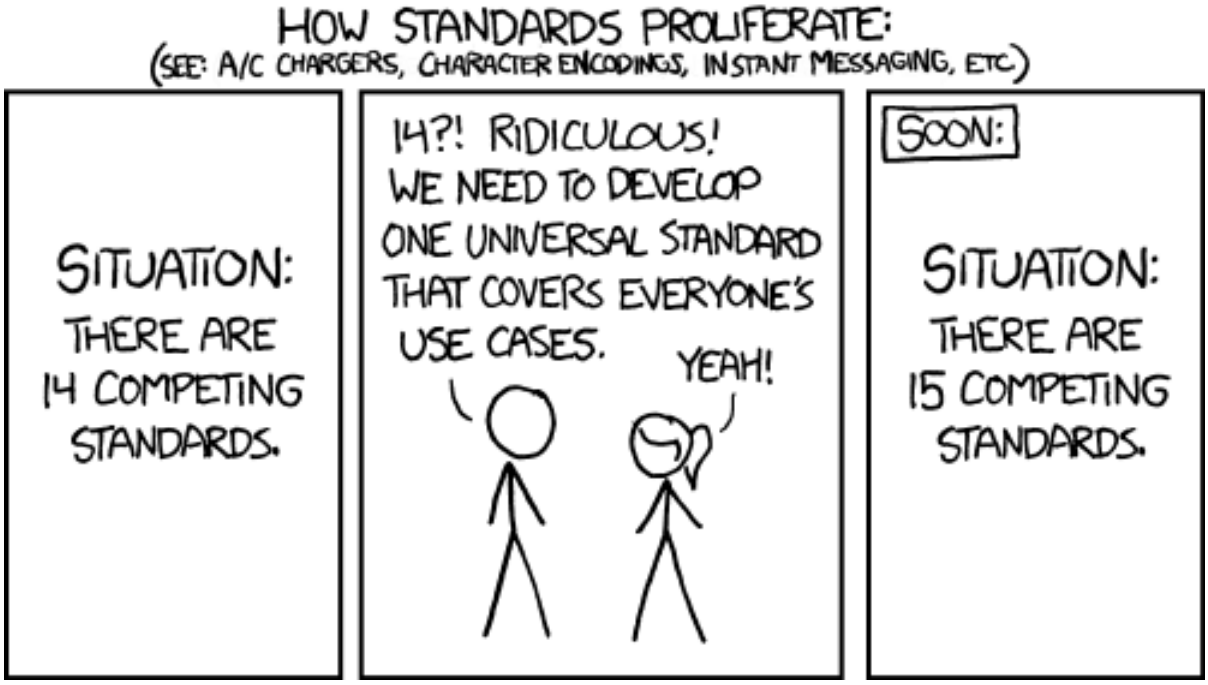

Illustration 3: How all the standards came to be [43]

It should be kept in mind that UPnP and ZeroConf have different aims, but as a subset of their solution some of the configuration steps listed above are solved. Both suites supply different solutions to all of the steps above except for addressing, which is solved the same way using **IPv4** Link-Local Addressing, **IPv4LL**[2], defined by RFC 3927 [44]. Also, it is possible to comply with both ZeroConf and UPnP; they don't exclude each other in any way. What follows below is first an overview of UPnP and ZeroConf, followed by a more in depth description of the protocols that constitute the two suits.

UPnP is managed by the UPnP forum and in order to make a device UPnP certified the developer need to be a member of the forum as well as complying with their application process. The forum membership cost US $5000 annually. The focus of UPnP is application communication protocols, this is to ensure that different devices easily can communicate with each other and make use of the services available in the network, e.g., printers and audio systems. As an effect the application communication specification is more mature than the auto configuration features [45][46]. UPnP does not define a protocol for name resolution, instead this is built into the service discovery protocol, since devices are identified by the services they supply. Service Discovery is done using **SSDP**, Simple Service Discovery Protocol [47]. The Application layer consist of several standardized device types with standardized communication protocols, called Device Control Protocol, **DCP**. As stated above this is the main focus of UPnP.

ZeroConf is focused on providing a generic foundation for automatic network configuration, and it has therefore omitted the application layer in its suite of solutions [48]. This is a very different approach than that of UPnP because this makes it possible to combine the ZeroConf with already existing applications, as well as future application protocols. ZeroConf is also more of an open standard with several open source implementations such as Bonjour and Avahi [49]. Name resolution is performed using multicast **DNS**, **mDNS**. This enables the devices to resolve names without the need for a name server. mDNS works much the same as DNS [50], but the name is resolved using a multicast query on the local link instead of querying a DNS server. Lastly service discovery is based on DNS Service Discovery, **DNS-SD**, which is also very much like DNS, only that services are resolved instead of names. Name resolving and service discovery are tightly

2 Also known as AutoIP.

bound together since both uses DNS records and should be sent together in the same datagram; it is therefore convenient to refer to both as **mDNS-SD**. As mentioned above the application layer is omitted in ZeroConf, this is because that there are already a lot of good application protocols, and there is no way of telling what kind of protocols will emerge in the future [51]. Apart from that there already exist best practices and standards on what protocol to use for different services, as well as other information needed to create a data flow [52]. This standardization is done by the Internet Assigned Numbers Authority, **IANA** [53].

2.6.1 Auto configuration protocols in detail

As mentioned there are a lot of auto configuration protocols addressing the same or similar problems and they are using different approaches and degrees of complexity. In order to understand and implement the different solutions they need to be further investigated.

Link-Local addressing is a method which enables devices to automatically obtain IP addresses without the need for a coordinating server, e.g., a **DHCP** server. The method, described by RFC 3927 [44], is as follows.

1. Select a random address from the range 169.254.1.0 to 169.254.254.255. Some addresses, the first and last 256 addresses of 169.254/16, are reserved for future use. By basing the random address on the MAC address then different devices should get different IP addresses.
2. When an address is chosen the device should probe the network to make sure that no other device uses that IP. This is done by sending **ARP** queries, but with the source fields empty. The number of probes, and the distance between them depend on the network.
3. Should a conflict be detected, i.e. any use of the IP, then the process is restarted with a new IP.

An equivalent procedure exist for **IPv6** [54] as well, but this is beyond the scope of this thesis.

Domain Name System, DNS, is a service for translating a textual representation of a host to a logical address, e.g., *localhost* is translated to 127.0.0.1 [55]. Normally the DNS service is run on one host, and if another host wants to translate a hostname it queries the DNS server. For a simpler network that doesn't have a DNS service running, the same result can be achieved locally using mDNS [56].

The functionality of DNS is implemented by sending out queries for different records, e.g., "*who have the name XXX*", and then there is a reply "*XXX have the address YYY*". This may sound simple enough, but there are a lot of rules describing the protocol and the protocol has been extended since it was originally created in the mid-eighties. Many of the rules can be omitted when doing a simpler implementation, at the cost of functionality.

mDNS works just like normal DNS, but instead of querying a DNS server the request is sent using multicast and the host with the queried name replies. To be able to distinguish between a global DNS query and a local mDNS query all local addresses, i.e., those ending with "*.local.*" should be resolved using mDNS.

DNS is a versatile system able to convey information about services, addresses etc. To be able to do this all information is stored in different kinds of records, and when resolving some query the DNS server responds by sending the matching records. Some of the relevant record types are listed below.

Record	Description
A	Address records, simply translates name to address.
AAAA	Same as A records, but for IPv6.
CNAME	Canonical name, or aliases.
PTR	Same as A records but translates address to name.
SRV	Services associated with domains [57].
TXT	Associates arbitrary text with domains.

Each DNS datagram contain a header describing the content of the datagram. Each record is sorted into one of four categories depending on the nature of the record, e.g., the record could be a query or an answer. The available categories are as follow:

Questions	Contains queries to the name server.
Answers	Answers to the question, from the name server.
Authorities	Contain information on authoritative name servers, i.e., other servers that can answer the query.
Additional	Contain information related to the query, but not answers.

As an example; a client need to resolve the name *XXX*, then a DNS datagram is composed with an A record, without any IP address, in the question section. The datagram is then sent to the DNS server. The server will respond with a DNS reply containing an A record with the corresponding IP address in the answer section.

The hostnames in DNS records consist of several labels separated by a dot, e.g., *svep.se* where the two labels *svep* and *se* are separated by the dot. The labels are encoded as a series of bytes, where the first byte describes the length of the label and the rest of the bytes are the label itself. The hostname is ended by a zero length label, i.e., just a zero, also referred to as the root domain. Given the example with *svep.se* it will be coded as 0x04*svep*0x02*se*0x00. Hostnames, or parts of a hostname, can also be compressed. The compression is based on the fact that the same hostname is likely to be used again several times in a DNS package, and then the same name can be reused by simply referring to where in the datagram the name can be found. Offsets are detected by checking if the two most significant bits in the length byte are 11 [55].

Multicast DNS adds several recommendations on how to handle different situations in order to avoid unnecessary load on the network. One such situation is when a host is powered on, it will want to know what kind of network it is connected to, and to avoid unnecessary traffic it is possible to ask that replies be sent as unicast. The same way, it is possible to send a query as a unicast, and then the reply to that query should be sent back as unicast. It is also suggested that the responses are cached to reduce the amount of queries [56].

When a multicast DNS responder start up there are two important steps to perform; probing, to make sure that the hostname is unique, and then announcing. Probing is done by sending an mDNS query asking for the desired host name, to make sure that the host name is not already used by another host. This also applies to other kind of resources that should be unique. When the mDNS probe packet is ready to be sent the host should wait for a random time, somewhere in the range 0-250ms, before sending it. The reason for waiting is if several devices are powered up at the same time, e.g., via a master switch, then all the probing will happen at the same time and this will cause unnecessary load on the network. After the initial delay three probe packets should be sent with 250ms delay between them, and if no conflict is detected, i.e., there is no response to the query, then the device can announce its existence. Should there be a conflict then the device will have to try to modify the requested resource to make it unique, or just disable that feature. When probing has been performed the next step is to announce which is done by sending a response package, with an indication that this is new data and the surrounding DNS caches should be updated [56].

DNS Service discovery is much like a DNS request, with the small difference that instead of querying for A records the query is for SRV records. Apart from that the functionality is the same as for mDNS, and the request are likely to be sent in the same DNS packet [58].

UPnP uses another protocol for service discovery; Simple Service Discovery Protocol, SSDP. This protocol also announces services via multicast, but instead of using DNS packets SSDP is based on HTML data formatting. Since HTML is not designed for this use there isn't the same kind of built in support for compressed names as in DNS; which ultimately leads to larger datagrams, with the upside that the content of the datagrams are more verbose since the content is text. The protocol bases the discovery on announcements, which means that each device announces its existence regularly. This leads to more traffic compared to DNS [47], but given the capacity of modern Ethernet networks this difference is almost negligible.

Another method of announcing services on a network is the Session Announcement Protocol, **SAP**, described by RFC 2974. SAP differs from both mDNS and SSDP in the sense that SAP announces multimedia sessions instead of services. Each session is identified by a unique number, called the session id, and the session is announced by sending a multicast datagram every five seconds [59]. As an example, PulseAudio uses SAP to announce RTP audio streams. The description of the announced session should be done using Session Description Protocol, **SDP**, described by RFC 4566 [60]. SDP session description consists of multiple lines of the format *type*=*value*. Where type is one character and the value is parsed based on the type. The fields are required to be arranged in a predetermined order and some fields are required to exist, which makes it possible to write a very simple parser.

Yet another method for service discovery is **SLP**, Service Location Protocol. Much like mDNS-SD services are resolved per request from the user, but with the addition that there may be service caches in the network to reduce traffic on larger networks [61].

2.7 Adding support for another audio output method

There are several ways to create new output streams for audio on a normal home computer. These basically boil down to four methods; creating a new media player, adding a plug-in to an existing media player, using proxy software between application and kernel or extending the kernel with drivers for a virtual device.

Implementing a new media player is not very difficult, and it is possible to design it in such a way which makes it easy to port to other systems. This solution requires the design of a user interface and the software will have to compete with all other media players on the market, which is perhaps the biggest downside. These factors add up and it can become a very massive project in terms of design and marketing etc.

Creating a plug-in to an existing media player is another way of adding audio output methods. However this is not always possible, some media players are built to be extended with plug-ins, while others are not. In order to add this functionality for a wide audience of media players a lot of plug-ins will have to be created since they often are media player specific. Plug-ins are also sometimes very noticeable, which might interfere with the user experience.

A virtual device created as an extension to the kernel is perhaps the most transparent method seen by the end user. The virtual device will be indistinguishable from real hardware and can act as the default sound device for the whole operating system. The downside of this solution is that writing kernel code is an extremely difficult process with no margins for error, and should there be a bug in the kernel code the entire operating system will crash. Kernel development is filled with *"black magic*" and obscure design consideration, and should be avoided if at all possible. However, some kernels have support for drivers running in the user space [62], i.e., running as an ordinary user program. Running a driver in user space makes the operating system less vulnerable to bugs in the driver, since a crash will only affect the driver itself. Another benefit of user space drivers is that they gain access to all resources available to ordinary programs such as standard libraries. Even though, it is still not an easy task to develop drivers. Furthermore

this method requires only one implementation for each operating system, or even each kind of operating system [63]. This method is valid for all the major operating systems, i.e., **GNU/Linux**, **Microsoft Windows** and **Mac OSX**, but for GNU/Linux and Mac OSX there are better methods explained below. In the case of windows there is the User-Mode Driver Framework, **UMDF**, which is a platform for creating user space device drivers [64].

Some operating systems also run a hardware abstraction layer, **HAL**, usable from the user space, i.e., proxy software. If this abstraction layer can be extended it is possible to create a virtual device without having to touch the kernel. This method can be used with Mac OS X [65], using the AudioHAL part of the Audio architecture. A similar method can be used with GNU/Linux using the software PulseAudio, which is already explained on page 13.

3 System Design

The platform to be implemented in this thesis, i.e., The SNAP, should be considered to be an evaluation platform and a proof of concept. Thus it should be designed with simplicity and speed of development in mind. This leads to a basic philosophy of a small number of ICs and wide use of existing software to improve development speed. It should also be easy to extend the platform in order to evaluate new peripherals as well as new networking protocols etc. What follows is an overview of the system.

Since most of the proprietary standards like UPnP and DLNA are expensive to use the implementation will make use of open and free standards. The SNAP should be easy to use, just plug it in to the network and it should all "*just work*". mDNS-SD offers a nice solution to autodiscovery as well as easy addressing of the device. This will make it easier to write a user-friendly control application running on a PC or smartphone, since it is possible to find SNAPs based on their description.

Many existing streaming applications such as **VLC** and PulseAudio stream audio using RTP streams and announce these streams using the SAP/SDP protocol. This is why the SNAP will have support for these protocols; since these are easy to use it will be possible to do functional tests early on in the development process. This will also make the SNAP versatile as it will be possible to adapt to a wide range of existing software without much extra work.

3.1 Hardware design

In order to make the hardware as easy as possible to realize the design should be kept simple. By keeping the design to a bare minimum in concepts and components the development process can be kept short, and without too much troubleshooting. To be able to receive an audio stream via Ethernet and to process it, the hardware will need an Ethernet controller as well as a processor. The audio stream must then be transmitted to an amplifier circuit to be able to play the sound stream on a speaker. There should also be some General Purpose Input and Output, **GPIO**, to be able to connect peripherals for evaluation purposes. The device will need power to operate and thus there must be some Power Supply Unit, **PSU**, which fulfills the power requirements of the components. This design is illustrated by Illustration 4.

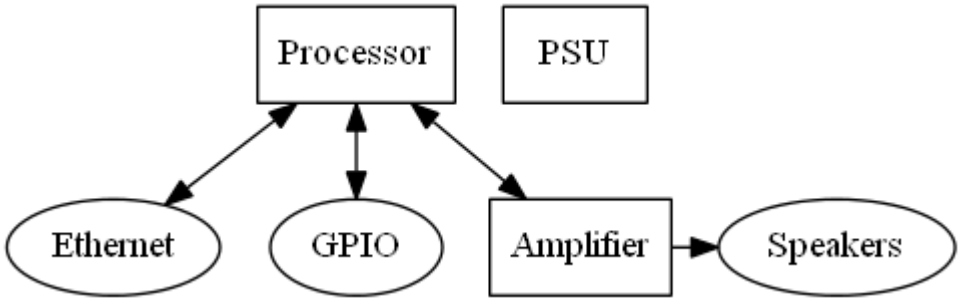

Illustration 4: Hardware layout

The choice of processor will dictate requirements on the rest of the components since the processor will be a central part of the design. Peripherals integrated with the processor will reduce the need for separate components, which is one of the aims for the design. On-chip peripherals for interfacing using I2S, I2C, SPI, **UART**, GPIO and Ethernet will enable communication with most of the peripherals needed. Another consideration is the computing capacity of the processor; in order to be able to evaluate compressed streams there need to be capacity enough to perform real time calculations on the data stream. Since it is hard to know how much capacity is enough it is better to have more than needed, and in a next revision of the hardware reduce the computing capacity if possible. Most processors used for embedded systems

have two or more memory areas, where the program memory and the runtime memory are the two areas of interest in this implementation. Larger program memory enables larger programs and larger runtime memory enables more data usage during runtime.

Many processors have Ethernet connection capabilities, but most make use of external circuitry to implement the physical layer [9]. To reduce the number of components a processor with internal physical layer was desired. Interfacing with the amplifier requires an I2S interface. With these requirements only one processor series was available; Luminary Micro Stellaris 9000-series based on the ARM Cortex M3 core. Some of the processors in this series have support for PTP, adding support for precise clock synchronization. Since the requirement for synchronization was unknown and the cost of adding PTP support was negligible this option was chosen. This resulted in four processors to choose from, they were all pin compatible only differing in the amount of program and runtime memory. The processor finally chosen was the one with the most memory, *LM3S9B96*, which gives the most room for development. Since all processors in the series are pin compatible a cheaper processor can be chosen in a later release when capacity requirements are better known.

When choosing the amplifier circuit, it is important to consider power usage, since high power consumption will create demands for cooling as well as affecting cost and weight. How the whole system is designed also inflicts in the decision of amplifier topology.For most cases an ordinary class D with AD modulation can be used. However, when the amplifier is close to the speaker, i.e., an active speaker, a filter less class D amplifier with BD modulation can be used, which simplifies the design. To allow a larger variety of speakers to be used, the amplifier should be able to drive at least the most common speaker impedances, i.e., four and eight ohms. Since the audio stream will need to be converted to an analog signal at some point, it is possible to reduce the amount of components by choosing an amplifier with built in DAC, or with equivalent functionality. This means that the amplifier should have a built in I2S interface for receiving sound data. To be able to reproduce a stereo stream the amplifier needs to have a stereo output. Of the qualified amplifiers the *TAS5713* was chosen due to its wider range of speaker impedances and the smaller capsule size. This amplifier also supports BD modulation.

For powering the system an initial thought was to use PoE, however this was omitted since PoE adds extra circuitry, and this can easily be added if needed later on. Instead an external regulated power supply of 10-24V was chosen, since this can also be supplied directly to the amplifier circuit. In order to supply 3.3V to both the processor and amplifier, the external power needs to be regulated; due to the inefficiency of a linear regulator a switching regulator was chosen. No major comparison was done when choosing a regulator, since any regulator capable of delivering current enough to power the circuitry as well as having at least the same range of input voltages as the amplifier, i.e., 10-24V, would suffice.

3.2 Firmware design

The purpose of the firmware is to realize the system design using the hardware described in the hardware design chapter. In order to do this, the firmware will need a scheduler and a TCP/IP stack; which there are several available to choose from. Since the *LM3S9B96* has a built in **ROM** with the **SafeRTOS** scheduler there is no point in using another scheduler. The *LM3S9B96* is also shipped with example code that uses the light weight IP stack, **lwIP**, written by Adam Dunkel. The lwIP is a mature and versatile IP stack, and since it is already ported to the processor it will save time. These two will act as a base for the firmware to be implemented.

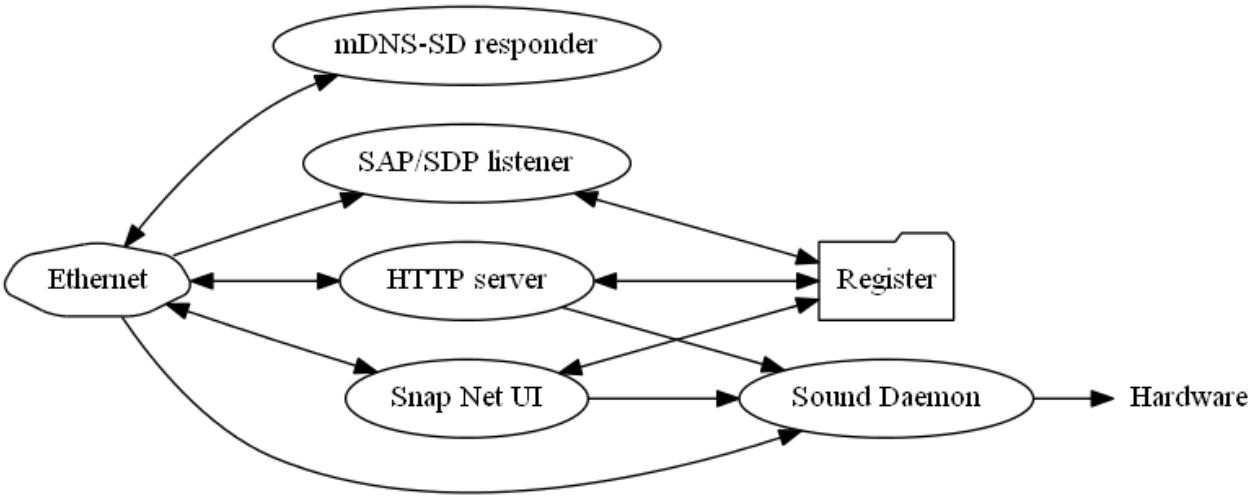

Illustration 5: Software modules. Threads are represented as circles and monitors as folders. Data flow is indicated by arrows.

By creating one thread for each task it is possible to focus on one problem at a time as well as making it possible to parallelize the development process. Apart from the processes described earlier there must be means to control the SNAP via a user interface. To solve this problem two services have been added; an HTTP server and a network User Interface, **UI**. Two interfaces are not really needed, but presenting a web page is almost trivial and it gives good visual result with minimum effort. A generic network user interface is handy during development since it is easy to add commands for testing purpose. Interthread communication is solved using a monitor design, where the data is guarded using semaphores. This makes it easier to later on extend the firmware with additional services without having to modify the existing threads. The main task of the device, to take an RTP sound stream and transmit it to an amplifier, is done in the Sound Daemon thread. An overview of the firmware is given by Illustration 5.

The different services should work as described earlier. The network UI should listen for commands and the following commands should be available:

Command	**Description**
get <identifier>	Fetches value of identifier.
set <identifier> <value>	Set the value of an identifier.
play	Start playback from the current session.
stop	Stop playback.
mute	Mute sound.
unmute	Unmute sound.

Identifier	**Description**
session_list	The list of all available sessions. Can not be set.
current_session <session id>	The currently selected session. Sessions are identified by their session id.
volume <value in dB, e.g. -0.5>	The volume.
name <name of the unit, e.g. "Kitchen snap">	Name of the unit.

The response of a command is either OK or ERR:<cause> followed by a newline. For commands that return several values, such as “get session_list” the response is OK followed by a newline, and then several lines. Each session is started with - - - -. And the current session is indicated with a * before the session id line.

Example:

```
get session_list
OK
- - - -
*s_id=1234
port=46014
…
- - - -
s_id=3241
port=45003
…
```

This design is first and foremost designed for development and demonstration since it allows simple implementation of a user application for controlling the SNAP.

The sound daemon is controlled by either the web server or the network user interface. Its task is to listen for RTP packets, decode them and place the sound samples in a buffer. It should also tell the hardware that there is sound to be played in a buffer, and then samples are fetched from the buffer via an interrupt routine. Anything related to playing sound, like synchronization is also handled by the sound daemon.

4 Results

The implementation went well as the simplistic design paid off and made the development process short and efficient. Early during the thesis work a lot of time was spent on doing preliminary studies, which was of great use later on in the thesis. The modular design also made it possible to start with the firmware implementation early in the project using a development kit, instead of having to wait for the hardware construction to be completed. This also made it possible to focus on the platform specific hardware when the finished board was available. Since the firmware already had been tested on the development kit, many of the bugs were already resolved when the development was moved to the SNAP. The existing software libraries proved very useful and built in debugging features was of great help.

The hardware platform was designed using the **PCB** design tool PADS from Mentor Graphics. The first stage was to draw the schematic which was fairly simple, as many of the components used already existed in the Svep library. However, some of the components, such as the processor and amplifier, needed to be added by hand. Before continuing with the layout, the schematic was reviewed and possible errors were corrected. To simplify the layout a four layer Printed Circuit Board, PCB**,** was chosen, giving separate ground and 3.3V layers, as well as two routing layers. For a more product looking system an initial thought was to place the board in some kind of casing and a suitable case was chosen and the board designed to fit inside it, however, the casing was never needed. Following the routing there was another review, after which the footprints of some components were corrected, finally the board was sent for manufacturing. Even though the board was checked several times during development some errors were found afterward; the I2C-bus pull up resistors were missing, the crystal pin out was rotated 90 degrees and the output filter for the amplifier was of AD-design instead of the intended BD-design. They all gave erroneous behavior but could be corrected; resistors were added, the crystals soldered in rotated and the mode of the amplifier changed to AD. A picture of the finished board can be seen in Illustration 6.

When the hardware design was completed, there was a problem when changing platform from development kit to the actual board. The development environment used was license bound to the development kit, and it was not possible to continue to use Code Composer Studio, **CCS,** on the new board. This was, however, not really a setback because it created incentive to migrate the firmware development to **Eclipse**. The migration proved to be a good move, since Eclipse is free, much faster and much richer in terms of useful features, such as better support for auto completion as well as better syntax highlighting. The reason for not using Eclipse from the start was that CCS was shipped with the development kit and the setup process was very well documented giving a fast startup. More details about the development can be read in appendix B.

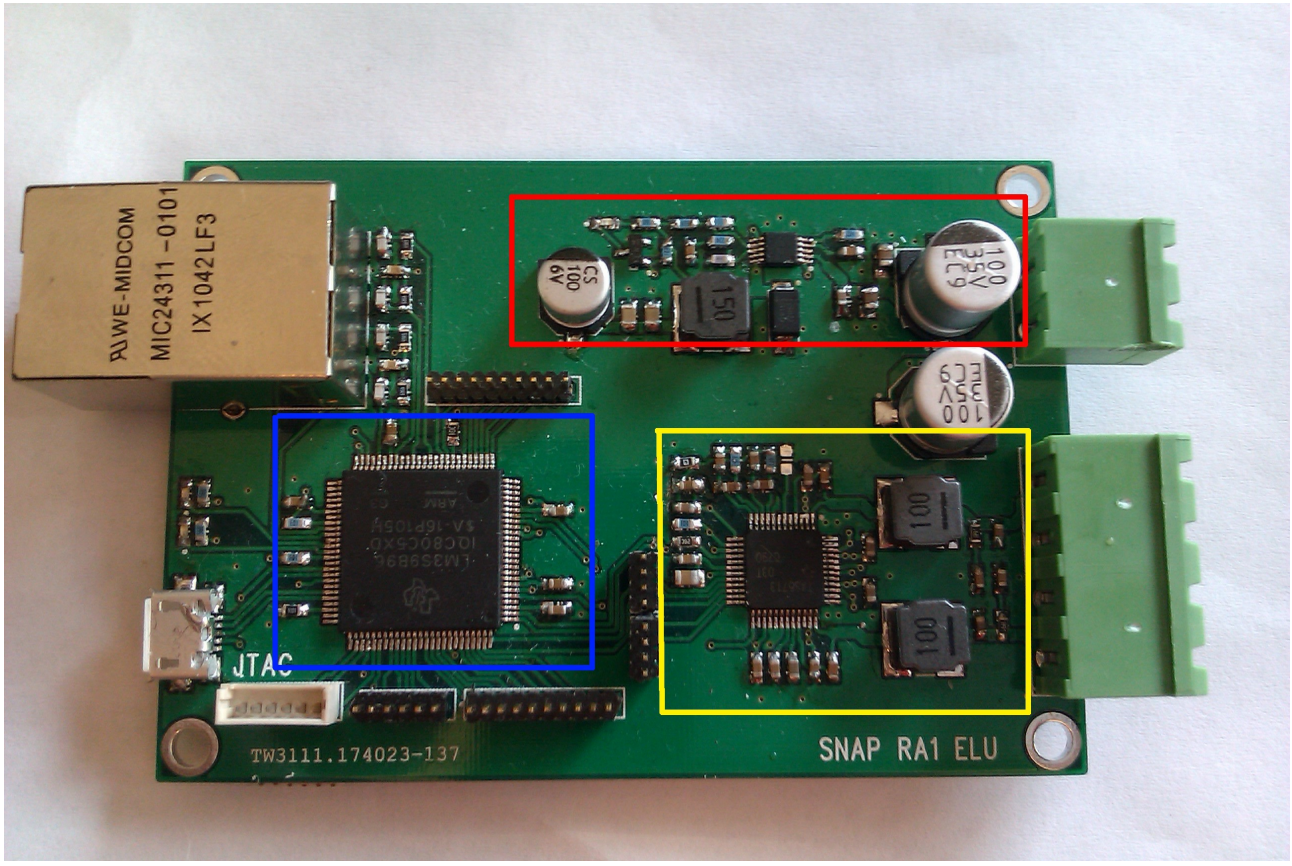

Illustration 6: The finished board with the PSU on top-right, the amplifier bottom-right and the processor on the bottom-right. Board dimensions are 55 by 80 mm.

The constructed platform is capable of streaming sound via PulseAudio, with some limitations. One problem is that SNAP suffers from packet loss. The packet loss happens when several packets arrive too close to each other which cause the incoming packet buffer in the Ethernet controller to overflow and discard packets. The ratio of the packet loss is approximately 1 per 1000 packets, where one packet contains 7ms of sound data, as by default in PulseAudio. The amount of packet loss also depends on the application playing the sound, which suggests that the amount of packet bursts depend on the size of the output buffer in the media player. A larger buffer will create longer bursts, but not as often, and the problem of the SNAP is the size of the bursts. PulseAudio buffers the sound data, and when the sound buffer is full all the sound data is transmitted in a burst. It is possible to tweak the sound output to lower the packet loss, but it is still a design flaw. Measurements have been made to make sure that there is no loss as long as the packets are not bursted, i.e., one packet is sent every 7ms.

The network used during testing is a very simple network; it consists of a direct connection between a computer and the SNAP. This network is almost free of delays and congestion, since there is no switch or router involved. It is not really representable for a real world network, but it has simplified the testing since the setup is really simple. In a real network there are larger delays, more bursts of data and packets may even arrive out of order.

The current design of the SNAP also suffers synchronization issues. The clock driving the amplifier circuit is not precise enough, this causes the playback to differ up to 3‰ from the input. This may cause to the playback to be 7ms, i.e., one datagram frame of sound, off after approximately 35 seconds. This may in the longer run cause buffer overflow or underrun in the SNAP, which ultimately causes skips in the playback. When the effects of this phenomenon first were observed, it caused the effect of an added noise in the

playback during a couple of seconds. This problem was solved by always monitoring the amount of data available in the buffer and make sure that playback was stopped when the buffer was empty, and incoming data was discarded when the buffer was full. This minimized the audible effect to a *click* or a short silence when this occurred, thus reducing the perceived error. When playing music this effect was almost negligible, since music often is very dynamic.

The initial thought of extending PulseAudio was discarded when it during discussions with the developers of PulseAudio it became apparent that the internal protocol for streaming sound was unsuitable to be used outside of PulseAudio, and it was advised to shift focus towards RTP streams [66]. This shift enabled a more generic platform capable of handling a wider range of audio sources, since there are a lot of applications capable of sending audio as RTP streams.

5 Discussion and conclusion

Just as mentioned in the Results chapter the implementation of the SNAP has gone very well, and the exceptions have already been mentioned. The implementation process have given some valuable insights into the world of streaming audio, and several experiences have been drawn regarding the components as well as the techniques used in this project. It has been realized that complying with different networking protocols is a difficult process, since there are so many factors to consider.

It is worth noting that the processor has a really long erratum, and while this hasn't caused many problems in this project it might just as well have caused severe problems . A bug in the hardware can be really hard to find and can set back a project several weeks. The amplifier circuit documentation has severe flaws which have led to some detective work in finding how to set up the amplifier for communication. Thanks to the simple design of the SNAP however, most work have been regarding the amplifier and the processor, and a more advanced design would likely have added new factors to consider as well as more documentation to study.

During development a simple network has been used, but in a real life situation a network might not be as friendly, adding delay, dropped packages and congestion. To be able to cope with these new problems, a larger buffer will likely be needed and the error detecting mechanisms will need more calibration; all this to allow for larger swings in the data stream.

The audio playback process is very sensitive to small delays, since almost any delay results in an audible effect. Since the process for receiving audio data via Ethernet and playing that data is a mixture of threads and interrupts it is sensitive to programming errors. There is a lot of data passed through this process, which makes it plausible that every bug will sooner or later expose itself but will at the same time be very hard to find. The bug causing noise during buffer overflow and underrun was very hard to find, since there is no way to monitor the execution for this special case. Instead the cause was detected when monitoring the audio signal using an oscilloscope, and the discontinuities could be viewed. This bug persisted even when the buffer fill level was monitored, which after some detective work was found to be due to interrupts happening while writing to a variable. This happens very seldom, but since the audio playback is a fast and repetitive process this occur a couple of times every minute . The experience drawn from this is that it is important to be thorough when writing this kind of code, since errors can result in obscure and hard to find bugs.

6 Future work

As mentioned earlier, the main purpose of the designed systems was to be as simple as possible to develop in order to study sound streaming via Ethernet. Should the SNAP be taken to a product stage there are a couple of redesigns available; e.g., the processor can be replaced with a cheaper model.

The packet loss mentioned in the Results chapter can likely be remedied using another method for fetching data from the Ethernet controller. Currently the Ethernet Controller signals new data with an interrupt, and then the data is fetched into memory by the processor. By using a technique called Direct Memory Access, **DMA**, the incoming data can be put directly into the RAM. DMA greatly reduces the overhead in handling incoming data, and word has that it is possible to receive data without packet loss even when there is only minimum inter-frame gap between the packets. Another way of reducing the packet loss is to change the processor, which will lead to the problem of porting the firmware to a new platform. Porting firmware is never an easy task, but the firmware is written in such a manner which makes it possible to just replace a couple of underlying functions.

The problem with the precision of the clock, which is driving the amplifier circuit, is a tricky problem to solve. The firmware can be written in a manner which detects buffer anomalies, which makes it possible to drop packets or delaying playback in order to synchronize it with the incoming stream. This kind of behavior will, however, cause skips in the audio playback, which is not very nice to listen to. A better solution is to change the clocking of the I2S bus and make it possible to adjust the playback speed, and thus adapt the playback to the incoming stream. The hardware should be able to do this clock adjustment, but it hasn't been implemented in the firmware yet. Another option is to add an external adjustable clock source as the processor has support for external clocking of the I2S module. A solution using a adjustable playback speed will also enable synchronization between several devices, which is not implemented in the current design.

Should the SNAP be taken to a product phase there are a couple of features that need to be added, not considering serial numbering and other administrative tasks. According to EU regulations devices introduced to the EU market after 2012 should not consume more than 0.5W in standby mode [67]. Currently the SNAP does not comply with the EU regulations as it uses ~1W when the amplifier is deactivated, but it is possible to reduce power consumption by putting the processor to sleep when it is idling. Currently the firmware is loaded into the processor via JTAG, which need a special connector on the PCB. Extra connectors are expensive and require PCB space. Instead firmware could be loaded into the processor using a Bootloader. With a bootloader the firmware can be loaded onto the processor via the **USB** port, which also enables firmware upgrades. Keeping the JTAG connector or at least the possibility to solder a connector could be, however, used as a sales pitch; by releasing the firmware as open source and releasing the documentation the end user can add features as needed. Depending on if the SNAP should be marketed as developer platform, to be built into home entertainment systems, or as an end user product, some kind of encapsulation should be designed. Should the marketing focus be on end users then a possible encapsulation could be a stereo speaker. Currently there is no control application for the SNAP; this is something that need to be developed should the SNAP be taken to a product phase. Developing a control application is not very hard; the current design is very simplistic and versatile. A control application could be implemented as a **GUI** coupled with a userspace driver or a HAL plugin. Something that could pose a problem with a control application is if there would raise a need for flow control in the audio stream, then this would have to be added both in the SNAP firmware as well as in the control application. Productifying the SNAP would also include the aspect of security as well as several users, with several streams etc. The current firmware has support for several streams on the network, but not for several users or for listening to more than one. With several users, or if the SNAP is connected to a shared or

vulnerable network, then the SNAP should be able to be protected with a password. Password protection will create a need for encryption, since otherwise it is easy to eavesdrop to the network for passwords. The SNAP could also be UPnP and DLNA certified to have the ability to be used in other existing home entertainment systems.

Currently there is only support for streaming sound via wired Ethernet, but a future design could enable support for streaming audio via WLAN. A design using WLAN would likely need some kind of flow control, since WLAN suffers from much more packet loss than wired network. Most WLANs to date offers a bandwidth fully capable of flow control, but since there might be much greater delays between received packets there must also be a long buffer to enable continuous playback.

Acknowledgements

We would like to thank Svep Design Center and all the wonderful people working there for giving us a place to do our thesis as well as a lot of support and without whom this thesis project would not have been possible. We would also like to aim special thanks to our supervisor at Svep, Mikael Hegardt, as well as Bertil Lindvall and Viktor Öwall, our supervisors at LTH. We would also like to extend our gratitude towards Ulf Tisting and Thomas Barregård for enduring our choice of music.

Both of us have had a wonderful time at LTH during the years we have studied, as well as the years we haven't. We have met many people during this time, and we would like to aim our gratitude towards all the people that have made the years at LTH as fun and exciting as they have been, there are not enough superlatives to describe how fun we have had. Both of us have participated more than our share in the student activities, and we hope that people continue to make LTH a wonderful place to stay, so that we one day may return and be able to make outrageous claims about how things used to be.

Last but not least we would like to thank our respective families, girlfriends, former girlfriends, Oscar, drinking pals, patient bus drivers, weird uncles and everyone else who are too many to remember, we will never forget you.

Glossary

AirPlay	Multimedia system created by Apple Inc.
API	Application Programming Interface.
ARP	Address Resolution Protocol, protocol for resolving logical addresses, i.e., IPv4 addresses, to physical addresses, i.e., MAC address.
Cat-5	Category 5 standardization of a cable, commonly used for Ethernet.
CCS	Code Composer Studio, an integrated development environment.
Class AB	An analog amplifier topology.
Class D	A digital amplifier topology.
CobraNet	Audio system by Cirrus Logic.
Codec	**Co**mpressor-**dec**ompressor.
DCP	Device Control Protocol.
DHCP	Dynamic Host Configuration Protocol, used in a network to distribute IP addresses.
DLNA	Digital Living Network Alliance.
DMA	Direct Memory Access.
DMC	Digital Media Controller, device class used by DLNA.
DMP	Digital Media Player, device class used by DLNA.
DMPr	Digital Media Printer, device class used by DLNA.
DMR	Digital Media Renderer, device class used by DLNA.
DMS	Digital Media Server, device class used by DLNA.
DNS	Domain Name Service.
DNS-SD	DNS Service Discovery.
Dynamic Range	A comparison between the different values in stochastic stream, such as sound.
Eclipse	An integrated development environment.
Ethernet	Normal wired home network, standardized by IEEE 802.
FLAC	Free Lossless Audio Codec.
GNU/Linux	Open Source operating system based on the Linux kernel.
GPIO	General Purpose Input Output.
GPS	Global Positioning System.
GUI	Graphical User Interface.
HAL	Hardware Abstraction Layer.
HTTP	Hypertext Transfer Protocol, the protocol used by to communicate with web servers.

I2C or I^2C	Inter Integrated Circuit.
I2S or I^2S	Integrated Interchip Sound.
IANA	Internet Assigned Numbers Authority.
IC	Integrated Circuits.
IEEE	Institute of Electrical and Electronics Engineers, see http://www.ieee.org
IETF	Internet Engineering Task Force, see http://www.ietf.org.
IP	Internet Protocol.
IPv4	Internet Protocol version 4.
IPv4LL	IPv4 Link-Local.
IPv6	Internet Protocol version 6.
JTAG	Joint Test Action Group.
KEXT	**K**ernel **Ext**ension.
KMDF	Kernel Mode Driver Framework.
LAN	Local Area Network.
LPCM	Linear Pulse Code Modulation.
lwIP	light weight IP.
MAC	Media Access Control.
Mac OS X	Operating system run on modern Mac computers.
mDNS	Multicast DNS.
mDNS-SD	Multicast DNS with Service Discovery.
Microsoft Windows	Popular operating system by Microsoft.
MOSFET	Metal Oxide Semiconductor Field Effect Transistor.
MP3	MPEG-1 or MPEG-2 Audio Layer III.
MPEG	Moving Picture Experts Group.
Multicast	Datagram sent to all hosts in a group, i.e., not necessary every host on the network.
NAS	Network Attached Storage.
NTP	Network Time Protocol.
Open Drain	If an output is open drain it can only sink current, often works in complement with some kind of pull up.
OS	Operating System.
PC	Personal Computer.
PCB	Printed Circuit Board, the board that the components are soldered onto.
PCM	Pulse Code Modulation.
PD	Powered Device.

PHY	Physical Layer.
PoE	Power over Ethernet.
PSE	Power Sourcing Equipment.
PSU	Power Supply Unit.
PTP	Precision Time Protocol.
PulseAudio	Audio mixer used in many modern GNU/Linux distributions.
PWM	Pulse Width Modulation.
RAM	Random Access Memory.
RAOP	Remote Audio Output Protocol.
RFC	Request For Comment, tool used by IETF to propose standards used on the Internet.
ROM	Read Only Memory.
RTOS	Real Time Operating System.
RTP	Real-time Transport Protocol.
RTSP	Real-Time Streaming Protocol.
SafeRTOS	A RTOS based on FreeRTOS, developed by Texas Instruments.
SAP	Session Announcement Protocol.
SDP	Session Description Protocol.
SLP	Service Location Protocol.
SNAP	Svep Networking Audio Platform.
Sonos	Multimedia system for streaming sound.
SPI	Serial Peripheral Interface.
SSDP	Simple Service Discovery Protocol.
TCP	Transmission Control Protocol.
UART	Universal Asynchronous Receiver Transmitter.
UDP	User Datagram Protocol, transport protocol without flow control.
UI	User Interface.
UMDF	User Mode Driver Framework.
Unicast	Datagram sent to only one host.
UPnP	Universal Plug and Play.
USB	Universal Serial Bus. Very common data bus for computers.
VLC	VideoLAN media player, a versatile open source media player.
WDF	Windows Driver Foundation.
WLAN	Wireless LAN.

ZeroConf	Zero Configuration, suite of protocols for automated network configuration.

References

[1] National Digital Information Infrastructure and Preservation Program. (2010). *PCM, Pulse Code Modulated Audio* [online]. Available: http://www.digitalpreservation.gov/formats/fdd/fdd000016.shtml [cited 2011-09-15]

[2] National Digital Information Infrastructure and Preservation Program. (2008). *Linear Pulse Code Modulated Audio (LPCM)* [online]. Available: http://www.digitalpreservation.gov/formats/fdd/fdd000011.shtml [cited 2011-09-15]

[3] K. Brandenburg. (2009). *MP3 and AAC explained* [online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.26.5956 [cited 2011-09-15]

[4] A. Schwarz. *ARM MP3/AAC player* [online]. Available: http://embdev.net/articles/ARM_MP3/AAC_Player [cited 2011-06-14]

[5] MP3 Converter. *Inside the MP3 codec* [online] Available: http://www.mp3-converter.com/mp3codec/frames.htm [cited 2011-06-14]

[6] J. Coalson. *FLAC project homepage. Format* [online]. Available: http://flac.sourceforge.net/format.html [cited 2011-09-15]

[7] S. Golomb. "Run Length Coding". *IEEE Transactions on Information Theory*. Vol. 12, pp 399-401, July 1966

[8] J. Coalson. *FLAC project homepage. Comparison* [online]. Available: http://flac.sourceforge.net/comparison.html [cited 2011-06-14]

[9] B. Forouzan. *Data Communications and Networking*, 4th ed. McGraw Hill. 2007. pp. 395-416

[10] B. Forouzan. *Data Communications and Networking*, 4th ed. McGraw Hill. 2007. pp. 678-693

[11] The Song Editor. *Delay Effects* [online]. Available: http://thesongeditor.com/ref/DelayEffects.jsp [cited 2011-06-14]

[12] D. Mills, U. Delaware, J. Martin, J. Burbank and W. Kasch. *Network Time Protocol Version 4: Protocol and Algorithms Specification*. IETF RFC 5905, June 2010. Available: http://tools.ietf.org/html/rfc5905 [cited 2011-06-14]

[13] NTP. *FAQ* [online]. Available: http://www.ntp.org/ntpfaq/NTP-s-algo.htm [cited 2011-06-14]

[14] D. Mills. *Network Time Synchronization Research Project* [online] Available: http://www.eecis.udel.edu/~mills/ntp.html [cited 2011-06-15]

[15] *IEEE Standard for Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*, IEEE Standards 1588, 2008

[16] H. Weibel. *Technology Update on IEEE 1588: The Second Edition of the High Precision Clock Synchronization Protocol* [online]. Available: http://www.ines.zhaw.ch/fileadmin/user_upload/engineering/_Institute_und_Zentren/INES/Downloads/Technology_Update_IEEE1588_v2.pdf [cited 2011-06-14]

[17] GPS.gov. *Timing* [online]. Available: http://www.gps.gov/applications/timing/ [cited 2011-06-14]

[18] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson. *RTP: A Transport Protocol for Real-Time Applications*. IETF RFC 3550, July 2003. Available: http://tools.ietf.org/html/rfc3550 [cited 2011-09-15]

[19] H. Schulzrinne, A. Rao, R. Lanphier. *Real Time Streaming Protocol (RTSP)*. IETF RFC 2326, April 1988. Available: http://tools.ietf.org/html/rfc2326 [cited 2011-06-14]

[20] CocoaDev. *RemoteAudioOutputProtocol* [online]. Available: http://www.cocoadev.com/index.pl?RemoteAudioOutputProtocol [cited 2011-06-14]

[21] Digital Living Network Alliance. *About DLNA* [online]. Available: http://www.dlna.org/about_us/about/ [cited 2011-06-13]

[22] E. Heredia. *An overview of the DLNA architecture*. Spring 2008. available from: http://download.microsoft.com/download/f/0/5/f05a42ce-575b-4c60-82d6-

208d3754b2d6/DLNA_Part_1_RS08.pptx [cited 2011-09-15]
[23] Digital Living Network Alliance. *DLNA Certified Device Classes* [online]. Available: http://www.dlna.org/digital_living/devices/ [cited 2011-06-13]
[24] Sonos Inc. *Sonos* [online]. Available: http://www.sonos.com/ [cited 2011-06-13]
[25] ZX2C4. *AirTunes 2 Protocol* [online]. Available: http://git.zx2c4.com/Airtunes2/about/ [cited 2011-06-13]
[26] Apple Inc. *iTunes remote* [online]. Available: http://www.apple.com/itunes/remote/ [cited 2011-06-13]
[27] CobraNet. *CobraNet FAQ* [online]. Available: http://www.cobranet.info/support/faq [cited 2011-06-13]
[28] *Standard for Information Technology – Portable Operating System Interface (POSIX). Base Specification, Issue 7*, IEEE Std 1003.1-2008, December 2008 Available: http://standards.ieee.org/findstds/standard/1003.1-2008.html [cited 2011-09-15]
[29] PulseAudio. *PulseAudio homepage* [online]. Available: http://www.pulseaudio.org [cited 2011-06-14]
[30] PulseAudio. *Modules* [online]. Available: http://www.pulseaudio.org/wiki/Modules [cited 2011-06-14]
[31] PulseAudio. *How to listen to the RTP stream* [online]. Available: http://www.pulseaudio.org/wiki/HowToListenToTheRtpStream [cited 2011-06-14]
[32] R. Elliott. *Efficiency* [online]. Available: http://sound.westhost.com/efficiency.htm [cited 2011-09-26]
[33] Ecler SA. *Digital Trilevel DT 4800/6800* [online]. Available: http://www.ecler.com/images/download/reportDTenglish.pdf [cited 2011-06-20]
[34] Rohitdb. Wikimedia Commons [online]. Available: http://en.wikipedia.org/wiki/File:Pwm_amp.svg [cited 2011-10-03]
[35] Bang Olufsen. *Advantages of ICEpower analogue Class D technologies* [online]. Available: http://www.icepower.bang-olufsen.com/en/technology/power/ [cited 2011-09-26]
[36] Texas Instruments. *Class-D LC Filter Design*. TI SLOA119A, April 2006. Available: http://www.ti.com/lit/an/sloa119a/sloa119a.pdf [cited 2011-06-20]
[37] NXP Semiconductors. *The I2C-Bus Specification. Version 2.1.* January 2000. Available: http://www.nxp.com/documents/other/39340011.pdf [cited 2011-09-27]
[38] NXP Semiconductors. *I2S bus specification.* February 1986. Available: http://www.nxp.com/acrobat_download2/various/**I2S**BUS.pdf [cited 2011-09-27]
[39] Microchip. *SPI – Overview and Use of the PICmicro Serial Peripheral Interface.* Available: http://ww1.microchip.com/downloads/en/DeviceDoc/spi.pdf [cited 2011-09-27]
[40] *IEEE Standard Test Acces Port and Boundry-Scan Architecture*. IEEE Std. 1149.1-2001, 2001. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=938734 [cited 2011-09-21]
[41] *IEEE Standard for Information technology.* IEEE Std. 802.3at-2009, 2009. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5306743 [cited 2011-10-03]
[42] H. Johns (2002, Dec). "Understanding Zeroconf and Multicast DNS", *O'Reilly Wireless* [online]. Available: http://www.oreillynet.com/pub/a/wireless/2002/12/20/zeroconf.html [cited 2011-06-23]
[43] R. Munroe. *XKCD* "Standards", Available: http://xkcd.com/927/ [2011-09-15]
[44] S. Chesire, B. Aboba and E.Guttman. *Dynamic Configuration of IPv4 Link-Local Addresses*. IETF RFC 3927, May 2005. Available: http://tools.ietf.org/html/rfc3927 [cited 2011-09-06]
[45] UPnP Forum. *UPnP website* [online]. Available: http://www.upnp.org [cited 2011-09-06]
[46] UPnP Forum. *How To Certify A Device* [online]. Available: http://upnp.org/sdcps-and-certification/certification/how-to-certify-a-device/ [cited 2011-06-22]
[47] UPnP Forum. *UPnP Device Architecture 1.1*. December 2008. Available: http://upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v1.1.pdf [cited 2011-09-21]

[48] S. Chesire. *Zero Configuration Networking (Zeroconf)* [online]. Internet: http://www.zeroconf.org/ [cited 2011-06-23]
[49] AirWiki. *Zeroconf Technologies* [online]. Available: http://www.science.uva.nl/research/air/wiki/ZeroconfTechnologies [cited 2011-06-22]
[50] B. Forouzan. *Data Communications and* Networking, 4th ed. McGraw Hill. 2007. pp. 797-812
[51] S. Chesire. *How does Zeroconf compare with Viiv/DLNA/DHWG/UPnP?* [online]. Available: http://www.zeroconf.org/ZeroconfAndUPnP.html [cited 2011-06-23]
[52] IANA. *Service Name and Transport Protocol Port Number Registry* [online]. Available: http://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xml [cited 2011-06-22]
[53] IANA. *Introducing IANA* [online]. Available: http://www.iana.org/about/ [cited 2011-06-22]
[54] S. Thomson, T. Narten and T. Jinmei. *IPv6 Stateless Address Autoconfiguration*. IETF RFC 4862, September 2007. Available: http://tools.ietf.org/html/rfc4862 [cited 2011-09-22]
[55] P. Mockapetris. *Domain Names – Implementation and Specification*. IETF RFC 1035, November 1987. Available: http://tools.ietf.org/html/rfc1035 [cited 2011-07-20]
[56] S. Chesire and M. Krochmal. *Multicast DNS* [online]. Available: http://files.multicastdns.org/draft-cheshire-dnsext-multicastdns.txt [cited 2011-07-20]
[57] A. Gulbrandsen, P. Vixie and L. Esibov. *A DNS RR for specifying the location of services (DNS SRV)*. IETF RFC 2782, February 2000. Available from: http://tools.ietf.org/html/rfc2782 [cited 2011-09-16]
[58] S. Chesire and M. Krochmal. *DNS-Based Service Discovery* [online]. Available: http://files.dns-sd.org/draft-cheshire-dnsext-dns-sd.txt [cited 2011-09-22]
[59] M. Handley, C. Perkins and E. Whelan. *Session Announcement Protocol*. IETF RFC 2974, October 2000. Available: http://tools.ietf.org/html/rfc2974 [cited 2011-07-18]
[60] M. Handley, V. Jacobson and C. Perkins. *SDP: Session Description Protocol*. IETF RFC 4566. Available: http://tools.ietf.org/html/rfc4566 [cited 2011-09-23]
[61] E. Guttman, C.Perkins, J. Veizades and M. Day. *Service Location Protocol, Version 2*. IETF RFC 2608. Available: http://tools.ietf.org/html/rfc2608 [cited 2011-09-23]
[62] X. Calbet. *Writing device drivers in Linux: A brief tutorial* [online]. Available: http://www.freesoftwaremagazine.com/articles/drivers_linux [cited 2011-09-26]
[63] Apple Inc. *Kernel Programming Guide* [online]. Available: http://developer.apple.com/library/mac/documentation/Darwin/Conceptual/KernelProgramming/About/About.html#//apple_ref/doc/uid/TP30000905 [cited 2011-09-23]
[64] Microsoft. *User-Mode Driver Framework (UMDF)* [online]. Available: http://msdn.microsoft.com/en-us/windows/hardware/gg463294 [cited 2011-09-26]
[65] Apple Inc. *Audio Device Programming Guide* [online]. Available: http://developer.apple.com/library/mac/#documentation/DeviceDrivers/Conceptual/WritingAudioDrivers/AudioOnMacOSX/AudioOnMacOSX.html [cited 2011-09-26]
[66] PulseAudio mailing archive. *Discussions about the native protocol of pulseaudio* [online]. Available: http://lists.freedesktop.org/archives/pulseaudio-discuss/2011-June/010494.html [cited 2011-09-27]
[67] European Commision. "*Commission's proposals to reduce standby elecric power consumption*" [online].
Available: http://europa.eu/rapid/pressReleasesAction.do?reference=MEMO/08/488, August 2008 [cited 2011-06-28]

Appendix A: Class D modulation

AD-modulation, **ADM**, or two-level modulation, is the classical Class D modulation where a 50/50 duty-cycle equals zero output, and changing the duty cycle creates the signal. ADM can be used in both differential and single ended mode, where in differential mode one of the signals is inverted.

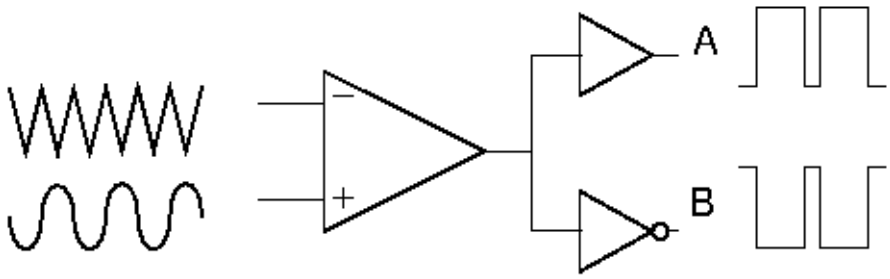

Illustration 7: AD-modulation

BD-modulation, **BDM**, or three-level modulation, is a further development of the traditional AD-modulation running only in differential mode. This gives the possibility to modulate both signals different from each other. Compared to ADM this gives less current through the low-pass filter, and with short speaker wiring the filter might not be needed at all.

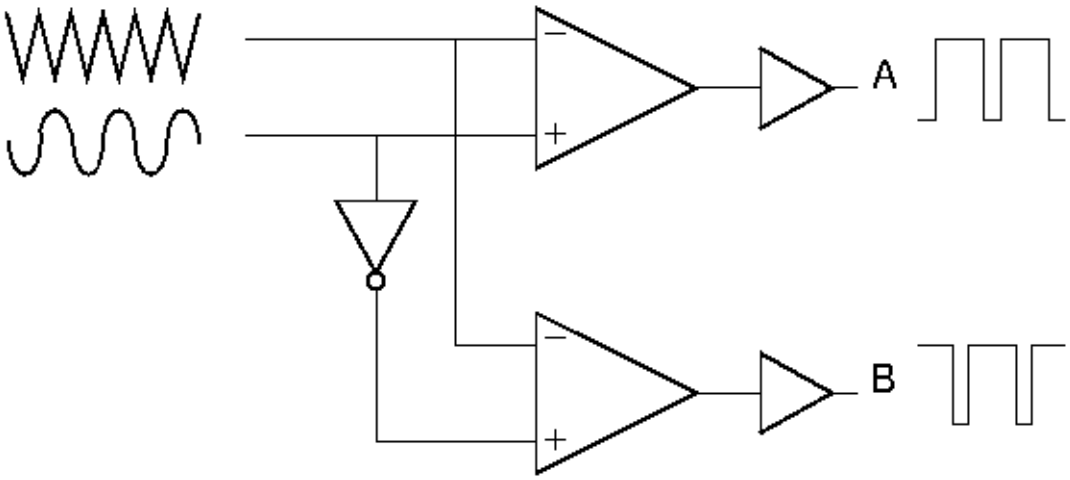

Illustration 8: BD-modulation

Appendix B: Build process

The development of the SNAP firmware was split into two main parts. The first part of the development took place on the *LM3S9B96* development board using Code Composer Studio, CCS, version 4. The second part of the development took place on the SNAP hardware using Eclipse. CCS is based on an legacy version of Eclipse and adapted to be used with Texas Instruments' products. The advantage with CCS is that its designed to be used in this kind of development, which means that there is support for automated build of the source code, support for downloading code to hardware and support for in-circuit debugging. The downside is that the trial license is limited to development board usage, and buying a full license is a $795 cost which was not an alternative within this thesis. Eclipse is a free, state of the art, modern development environment, with the main focus on Java development. Eclipse is also adaptable to be used with other tasks, such as development of embedded firmware. This requires the build to be set up manually, and that is not a trivial problem.

First of all a compiler toolchain is needed, and in the case of embedded system this is often a very complex task to set up from scratch. For the sake of this text it will be assumed that there is a toolchain, e.g., the CodeSourcery toolchain. The toolchain contains the tools used for creating executable files from source files, e.g., the compiler, the linker etc. Building an entire project takes longer time if the there are many source files to compile and if the files are large. As the project grows the compile time will as well, and for larger projects this can take hours. Needless to say, this slows down the development if the whole project needs to be compiled in order to verify small changes. To avoid this problem, instead of re-compiling the entire project only the files that have been modified should be compiled, and this will save a lot of time. One program for managing the build process, and keep track on which files have been modified is *GNU/Make*. Basically this checks if the target of a compilation is older than the source, and if it is it will recompile that file. GNU/Make need to know what files to check, and how to compile the files, this is described by a Makefile. The Makefile contains rules for creating targets, e.g., executable files, from dependencies, i.e., source files, using some method, e.g., the compilation. It is possible to configure Eclipse to use a given Makefile, which enables a custom build process.

When the firmware have been compiled into an executable file the firmware need to be tested. Testing is done on the target platform, which means that the firmware must be loaded onto the target. There are many ways to do this, but the focus here is via JTAG since it enables debugging as well. One way of doing this is via a program called *OpenOCD*, which is an open source software for JTAG communication. OpenOCD act as a server on the developer's computer, and by sending commands to this server it is possible to control the behavior of the connected device. It is very useful to be able to debug the firmware as well, and monitor the program run. To do this a program called *GDB* is used. GDB can establish a connection to OpenOCD, and via this link the firmware can be debugged. Using GDB itself to debug is however a somewhat tedious process. By connecting Eclipse to GDB it is possible to do all the development from Eclipse, which for this thesis makes it equivalent to CCS, only that it is free.