Southampton Solent University

School of Media, Arts and Technology

BSc (Hons) Media Technology

2016

Adam Weeden

Intelligent Silence Detection

Supervisor : Steven Moulster
Date of presentation : May 2016

Southampton Solent University
School of Media, Arts and Technology

BSc (Hons) Media Technology

Academic Year 2015-2016

ADAM WEEDEN

10118721

Media Technology Project

Intelligent Silence Detection

Supervised by Steven Moulster
MAY 2016

This report is submitted in partial fulfilment of the requirements of Southampton Solent University for the degree of BSc (Hons) Media Technology.

Acknowledgements

I am grateful for the assistance of my project supervisor Steve Moulster for their guidance and encouragement to complete this project.

I am also extremely thankful for the advice given to me by my faculty's tutors Andrew Horsburgh, Chris Barlow and Sean Lancastle towards completing this assignment.

I extend my gratitude to Southampton Solent University for giving me this opportunity, and to the student union's radio station Radio Sonar for being the inspiration for this project.

I also acknowledge my gratitude towards my parents who have supported me morally and economically over the past 4 years at university.

Abstract

This report covers the usage silence detection technology in today's radio broadcasting industry and investigates whether it is possible to use consumer-level microcontrollers in order to perform the act of silence detection.

This report begins by looking in the necessities behind the technology and will go on to talk about the development of an alternative technology for entry level radio stations with smaller budgets using consumer-level microcontrollers as an alternative to traditional off-the-shelf commercial solutions.

During this project an audio signal was fed into an Arduino Uno, and the signals analysed, at which point it was determined that Arduino was not a suitable microcontroller for the application of Silence Detection

Contents

List of Figures

List of Tables

Introduction

In the radio broadcasting industry, every second on air counts, as music and adverts are scheduled for maximum impact. One of the worst things a broadcaster can do is to broadcast dead-air (Eastman, 2012) – particularly in radio. Dead-air can be caused by a number of reasons.

Dead-air can result in impatient listeners switching to competing stations to continue listening to scheduled content, as well as legal considerations with a station's broadcasting licence.

The presence of dead-air can lead to radio stations losing listenership, which can be harmful to a station's income, as commercial partners may lose financial interest within a station or network – a loss of which will be ultimately be economically bad for the broadcaster.

Some of the reasons dead-air can be caused, can be due to a number of different factors such as technological, administrative or human failures.

Some of the technological reasons that can result in dead-air can include the failure of playout systems, studios and routing equipment. Administrative errors can also cause problems, such as scheduling errors, as well as other human errors such as presenters not turning up on time.

Because of the chance of software or equipment failure, a separate system is required to be able to compensate for these failures. The purpose of this project was to determine if it would be possible to design and build a broadcast-ready silence detection unit using readily available consumer-level microcontrollers such as Arduino or Raspberry Pi.

The main reason for producing a system using consumer-level equipment is being to be able to provide a low cost technical solution for not-for-profit broadcasters such as student, hospital and community radio stations with similar functionality to more expensive equipment available commercially. While there are software solutions available, they may still require investment in additional equipment, so may not be as cost-effective as they seem at first glance. This project will go on to investigate the processes behind silence detection, and the development of a system using consumer-level microcontrollers.

Aims and Objectives

The aim of the project is to create a device using consumer-level microcontrollers, which would be capable of monitoring one or more analogue audio inputs, to carry out analysis of the audio signal, to look out for audio transmission errors such as silence, distortion, clipping, and noise.

The device designed during the product would be able to interpret the results from analysing the audio signal, to carry out a defined action, which could include: communicating with a web monitoring service, switching audio sources, playing out localised audio, or triggering general purpose inputs/outputs.

This project has been designed at aiming to make silence detection more intelligent, stepping past the boundaries of traditional systems which could just determine silence, but also other audio transmission problems that could occur, such as clipping or distortion.

As part of this project it is also to be considered to work to integrate the system to be able to be controlled and or monitored from a web interface or the cloud, and follow 'policy' driven triggers, such as on detecting clipping, sending a push notification to the studio to remind the operator to double check their levels, or on detecting silence to send a SMS message to a contact.

The level of success of this project will be determined on whether a system can be set up that can accept multiple inputs, and detect when a signal drops below a defined threshold.

The device will be deemed functional if it will be able to detect the amplitude of an input audio signal drop below a configurable threshold, as opposed to just the sole presence of an audio signal – this will require, at minimum, the analysis of speech based frequency bands.

Background

What is Silence Detection?

Community and Commercial radio stations use silence detection as part of their everyday transmission chain. Silence detection systems are commonly used to monitor the state of the transmission feeds on transmission sites, acting as a last line of defence to aid in preventing a radio station from transmitting dead-air for an extended period of time. (Walters, 2006)

Why Do Radio Stations Use Silence Detection?

It is in the best interest of radio stations that they monitor their transmission feeds for number of different reasons. The main reason a broadcaster would want to do ensure they prevent silence, is from a commercial standpoint.

Commercial Protection from Silence Detection

If a radio station broadcasts silence, listeners are likely to quickly lose interest in the station, and tune in to a competitor. The result of losing listeners could mean that advertisers would be less willing to invest money into the station, which could have a devastating impact.

ofCom, the law and licences

Other than the loss of listenership and commercial revenue silence can cost a radio station, broadcasters could be seen to be in breach of their broadcasting licence, and as a result lose their licence through being revoked, or to a competitor when the licence is re-advertised at the end of its term.

Rules regarding the requirements for broadcasters are laid out in the Broadcasting Act (1996) and Wireless Telegraphy Act (2006). Under guidance given by ofCom the Site Engineering Code for Analogue Radio Broadcast Transmission Systems, stations are responsible for their distribution systems, and should be able to switch to a standby source, or switch off their transmitter in the event of failure. (ofCom, 2013)

What Silence Detection Systems are Available?

A number of systems do exist to monitor for silence, with solutions available in the form of both hardware and software.

Hardware

One of the current popular choices and low cost hardware solutions is the Sonifex Redbox SD-1, currently available on the market for around £400 (prices checked May 2016). The Sonifex Redbox SD-1 is capable of monitoring one stereo input, and switching to a standby source, which can triggered via GPIO. One of the downsides to the Sonifex Redbox SD-1 is that out of the box is unable to provide a way to remotely monitor the state of a transmission feed.

Software

Aside from hardware based options, there are solutions which exist in the form of computer software, an example of which is PiraCZ Silence Detection, which works by taking the input from a soundcard, and monitoring it.

PiraCZ is capable of performing a variety of defined actions in the event of silence being detected, such as triggering GPIO (similar to hardware solutions available), opening software, and sending HTTP requests. However, because PiraCZ is a software solution within an operating system (Microsoft Windows), there is a higher chance of failure of the device due to software on the device crashing.

In addition to the risks of software failure, is the requirement of additional hardware, such as a split from a distribution amplifier, or to switch the audio.

While audio could in theory be fed directly through the device itself through means of a loopback which could present new problems. Loopback is where the device decodes the audio input, before processing it, and re-encoding it.

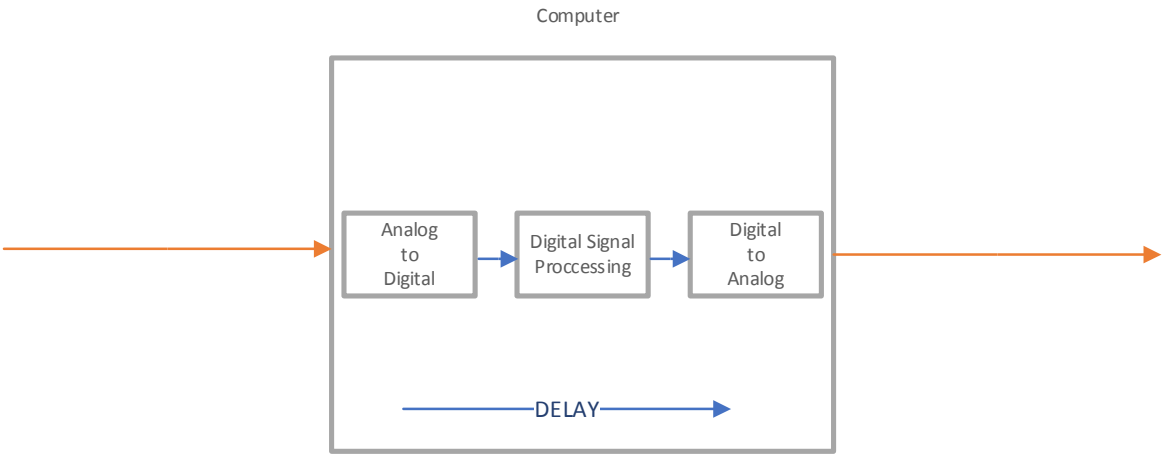


Figure 1 Delay caused by Loopback

The process of looping back the audio through the device could result in unnecessary delay to the audio which might be undesirable in a live environment.

In addition to delay, a failure of the computer running the software, would halt the loopback of sound, ultimately taking the station off air. Another consideration of working with loopback audio is the quality of the audio interface used with the device, as it could result in a transmission of poor quality without a broadcast quality audio interface.

One problem faced by many entry level silence detection units, is that they do not come with the ability to offer remote site monitoring out of the box.

Since most radio stations studios are not located on the same sites they transmit from, this can make difficulties when monitoring a remote transmitter site, or multiple sites which are often unattended.

Monitoring of remote sites is highly important to broadcasters, as they need to be made aware of any problems that are occurring in their transmission chains. Without the presence of the ability to monitor remote sites, this can add unnecessary delay in allowing station technical staff to respond appropriately to the situation.

What Causes Dead-Air?

Dead-air can be caused by a number of different reasons, and it is important that the station can be informed as soon as possible.

Case Study: Wireless Group PLC.

Examples of causes of dead-air have included a case in 2008, where a fire broke out on the programming floor of talkSPORT during their breakfast show on the 17th October. talkSPORT was taken off air for a period of 90 minutes while the show was moved to a backup location (Martin, 2008). In this instance where talkSPORT went off air, no back-up system had been triggered, as the studio hadn't been signed out (ibid.). In this instance, a silence detection system could have been in place which could have prevented an unscheduled silence from happening.

Case Study: British Broadcasting Cooperation

At the BBC in 2009, one of their radio broadcasting hubs suffered a catastrophic power failure on the 2nd May, which took BBC Radio 1, 1Xtra and BBC Asian Network off air for about 45 minutes. However, in the incident involving the BBC, their automatic silence detection system successfully activated approximately 30 seconds after the power failure. (Baddhan, 2009)

Case Study: Celador Radio

On 4 February 2015 Sam FM in Southampton suffered a technical failure with one of their studio to transmitter links, which resulted in their transmission stream cutting in and our every few seconds (Sam FM, 2015), which wasn't enough to trigger one of their silence detection systems.

Case Study: Bauer Radio

Aside from technical failures, human error has been shown to take part in contributing to unscheduled silence, as presenters failing to turn up. One claimed instance of this was from one of Chris Evans' radio show on Virgin Radio (now Absolute Radio)

Methodology

This chapter begins by introducing the development of the project as it progressed and then goes on to mention how the aims of the project changed throughout the project.

Outline

From the outline of the project, the main intention was to base the project of silence detection around consumer-level microcontrollers. A number of solutions were investigated for this project.

Amongst the choices found were the Raspberry Pi, Arduino Uno and Intel Galileo. Each of these microcontrollers were shortlisted, as they were readily available, and are an open source platform, minimising the need for proprietary software or development kits. From the shortlisted devices, the Arduino was found to be the most cost-effective solution, however concerns were raised as to whether it would be capable of maintaining a web interface, while sampling and processing the audio inputs.

Choosing the Microcontroller

From the initial research into the microcontrollers, the Intel Galileo provided a suitable compromise for the capabilities of the Arduino, as it had all the capabilities of running the same type of code as Arduino, as well as having built in networking capabilities and being able to run an operating system such as Windows on Devices or Linux to allow easier integration onto the internet of things.

Unfortunately for the Intel Galileo, its hardware had quickly become out-dated, and Microsoft had ended support for running their Windows on Devices platform on the Intel Galileo. From all of the devices, the Intel Galileo was the most expensive, and as a result also had the least amount of active users/support. A decision was made that would be more cost and time effective to choose an alternative platform.

There are a number of other microcontrollers available as an alternative to the Intel Galileo, such as the MinnowBoard Max, and DragonBoard410c (Microsoft, 2015) however the Minnowboard Max is more expensive than the Intel Galileo

costing over £100 (RS Components, 2015) which is less financially feasible for this project, and the DragonBoard410c is less readily available in the UK.

The third option investigated for this project was the Raspberry Pi, which would be capable of running the web software, and carry out analysis on the audio signals. The disadvantage however of the Raspberry Pi, was the inability to take analogue readings from its GPIO pins, however due to open nature of the device, there would be analogue-to-digital converters available to use on the platform.

From the research carried out, it was decided that the Raspberry Pi was chosen as the best option for a number of reasons, including cost, networking capabilities, and readily available software/development platforms available.

Choice of Audio Format

In broadcast when transmitting audio, there is usually the choice of using analogue audio, or digital audio. For the scope of this project, the primary focus will be on analogue audio, as this is the simplest form to monitor.

If there is further time available in the project, it could be worth developing support for digital audio. Digital audio is today commonly encoded in AES3 (Robjohns, 2007), also known as AES/EBU, an open standard set by the Audio Engineering Society (AES) and the European Broadcasting Union (EBU).

(European Broadcasting Union, 2004)

In order to be able to process AES3 audio, would require separate circuitry from the analogue signal path for a number of reasons.

This is due to the operational differences between the two technologies, and for device protection.

In order to make the device complaint with the AES/EBU specification set by the EBU, the sending or receiving circuit must use a coupling transformer (NTI Audio, 2012).

Another consideration for working with AES3 audio is the operational voltages of AES3 data.

According to the AES3 specifications, the typical voltage range for an AES3 is between 2 and 7 volts peak to peak (European Broadcasting Union, 2004). An operational voltage of 2 to 7 volts could prove problematic whilst integrating with a microcontroller like the Raspberry Pi, which has an operational GPIO voltage of 3.3v for a digital high (Raspberry Pi Foundation, 2015).

Though the Raspberry Pi would be capable of taking a digital high reading from a 2 volt AES3 signal as the minimum threshold for digital high is 1.8 volts (Raspberry Pi Foundation, 2015), a high end voltage from an AES3 signal at 7 volts could have the potential to cause damage to the microcontroller, and would need to be passed through a limiter to prevent damage to the device.

With time and form factor considerations, integrating support for AES3 audio into the project would not be feasible, and the project will only focus on monitoring analogue audio.

Audio Input Arrangement

In consideration with the design requirements of the project, decisions need to be made to the number of audio inputs the device will be able to support and monitor. The device should require a minimum of two audio inputs that it can switch between. In order to provide redundancy, additional inputs can be incorporated into the design. Examples of inputs can be seen in the table below:

Table 1 - Audio Input Arrangement Option 1

Input	Name	Description
A/B	Transmission Feed MAIN (Stereo)	Main transmission feed
C/D	Transmission Feed HOT Standby (Stereo)	Backup of main transmission
Y/Z	Output Loopback	Output Monitoring

Table 2 - Audio Input Arrangement Option 2

Input	Name	Description
A/B	Transmission Feed MAIN (Stereo)	Main transmission feed
C/D	Offline WARM SPARE (Stereo)	Local backup system, e.g. a CD Player
Y/Z	Output Loopback	Output Monitoring

In each of these systems, the output can be looped back and monitored internally to check for the audio continuity. This means that the system would be able to check itself to ensure that audio is still passing through the device to ensure corrective action has worked. In the event of a failure, the device could be programmed to play media internally, or as a final resort send a command to shut down the transmitter.

Choice of Programming Language

Once the Raspberry Pi had been chosen for the project, research needed to be carried out into which would be the best operating system to run the device with. The most flexible option was to run a Linux distribution, as this would allow flexibility over choice of programming language, allowing the software to be developed using a multitude of programming languages including C++, Java and Python compared to Windows on Devices, which is limited C++.

The language chosen for this project was Python, as it was a much simpler programming language to use, and would not need compiling, compared to C++ which would be able to save time during the project's development.

Once the programming language had been chosen, the next important step was to find a method of taking analogue inputs from audio sources into the Raspberry Pi. Following a small amount of research online into analogue-to-digital converters (also known as ADCs) one suitable candidate was found, the ADC Pi Plus from AB electronics. This analogue-to-digital converter board was capable of taking 8 analogue inputs at 17 bits, and came with Python code libraries to help integrate it into the project using data transferred to the Raspberry Pi over a protocol called I^2C (AB Electronics, 2016).

The ADC Pi Plus is based around the Microchip MCP3424 analogue-to-digital converters, from which further research found, only had a sample rate of up to 250Hz (Microchip, 2009) which seemed concerning.

Based on the Nyquist-Shannon sampling frequency theorem, to represent a signal, it needs to be sampled at twice the maximum frequency that is going to be sampled. In the world of radio, the highest frequency usually sampled is around 15 kHz (Lesurf [no date]), which is based around traditional FM processing. To represent the frequency range of FM Radio would have required a sample rate of at least 32 kHz. Therefore, it was found that the sample rate of the ADC Pi Plus was inadequate for this task. As a compromise in finding an analogue-to-digital converter, a decision was made to at least focus on being able to capture the human vocal frequency range. The main frequency components of human speech are between 250 and 2000 Hz (Wiley, 1993), so to capture speech properly, it

would be desirable to have a sampling frequency of at least 4 kHz, which is still outside the maximum sampling of the ADC Pi Plus.

Once device which was found to be able to sample at least 4 kHz turned out to be the Arduino, which is capable of reading analogue signals at a rate of 10 kHz, based on the analogue read time of 100 microseconds (Arduino, 2016).

The next challenge is sending data between the Arduino and the Raspberry Pi. One way in which this can be done is using serial. The first stage of doing this was to determine how much data needs to be sent across the serial connection.

To determine how much data needed to be sent across the serial connection required the knowledge of that information needed to be sent. By default, Arduino will sample analogue reads at 10-bits per channel recorded. For the purpose of this project, it would be of interest to be able to monitor two stereo inputs, and one stereo output, so in total 6 channels.

For the Raspberry Pi to be able to differentiate between these channels, the Arduino will need to send this information over Serial as well. The simplest way to do this is in binary form. To do this, this will require at least three bits (as shown below in the table), which can represent up to 8 different channels.

Table 3 Proposed Channel ID Bits

Data	Channel
000	Input 1 (L)
001	Input 1 (R)
010	Input 2 (L)
011	Input 1 (R)
100	Output 1 (L)
101	Output 2 (R)
110	[Spare]
111	[Spare]

From this, each channel will require at least 13 bits to identify the channel, and the analogue level. To make sure that the receiver (the Raspberry Pi) is able to understand the information from the Arduino, and check the data to see if it is

correct, it will be necessary to use start and stop bits, and cyclic redundancy checking. The way in which the data can easily be checked is to use a parity bit following the stop bit. The way in which the parity bit works, is it helps to check that all the data is there, by means of parity checking. The principal of parity checking is to make sure that the bit message received is either all odd or all even, allowing the receiver to be sure it has received a correct message. For the purpose of this project, an EVEN parity system is adopted as choice.

To ensure a more successful data transfer, the messages should be kept as short as possible, to prevent data loss. In the case of transferring the analogue values from the Arduino to the Raspberry Pi, this means each channel should be sent in its own message.

Table 4 - Proposed Message Structure

1	1	0	0	0	1	1	0	1	1	0	1	1	0	1	0	1	1	1	0
Start Bit			Channel Bit			Analog Read Data Bits										Stop Bit			CRC

However, there is still a problem, as the Arduino is only capable of measuring signals between 0 and 5v, and as such, it will not be possible to directly measure an audio input, as the signal being received will be cycling between positive and negative voltages, so therefore wouldn't be able to be measured by the Arduino without some means of signal compensation. One solution to this is to use a full wave rectifier to correct all voltages to be positive. However, using a full wave rectifier is not a suitable application, as this will mean that it would be almost impossible to measure peak to peak voltage changes, and the device would not be making efficient use of its dynamic range.

Instead, the correct way to rectify this is to lift the audio signal. Lifting the signal can be achieved using a network of resistors. Since the Raspberry Pi and Arduino have a 5v power output, this can be used as the power source. The Raspberry or Arduino power source combined with two identical resistors can form a potential divider, to lift the audio signal by 2.5v, as illustrated below.

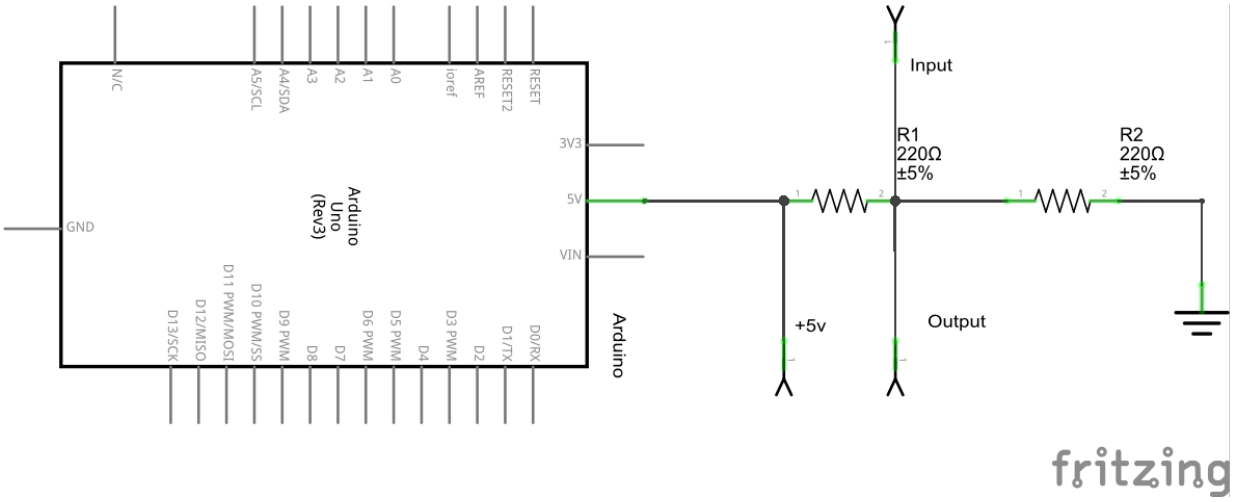


Figure 2 Lifting the Audio source

Since the new "zero level" is 2.5v, this will read out on the Arduino's analogue read as a value around 512 or 1000000000 in binary.

When coming to program the Arduino in reality, it became apparent that it would not be possible to sample inputs simultaneously. This can become an issue if each pin were to be sampled sequentially, as this would lower the effective sample rate for each channel by a factor of 6, equivalent to sampling at 1.6 kHz, which would mean an effective frequency range of 0 ~ 800 Hz. Though this could be doubled by only monitoring the signals in mono, there would not be sufficient sampling to adequately cover the vocal frequency spectrum.

Some reconsideration to this project was required, so a decision has been made to whether accuracy is required to determine the presence of audio compared to no audio at all. A short investigation was carried out into the effects of under-sampling audio, by simulating the frequency response available. Using a variety of Musical and vocal audio samples which had been normalised to broadcast levels.

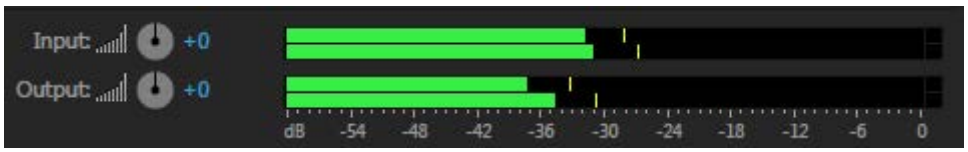


Figure 3 - Kate Bush – Wuthering Heights (Piano Intro Sample)

The first sample was taken from Kate Bush's song Wuthering Heights, where the song starts with a high pitched piano and voice – this has effectively attenuated by about 6dB.

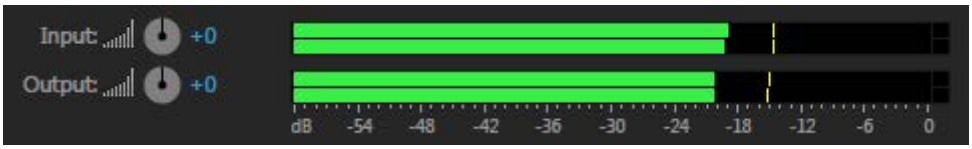


Figure 4 Prince - The Most Beautiful Girl in the World

The second sample was taken from Prince's song, The Most Beautiful Girl in The World. This song does not appear to be adversely effected by low pass filtering.

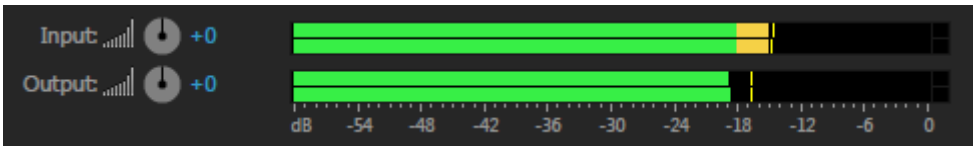


Figure 5 - Heart Breakfast (London) - Emma Bunton [GLOBAL]

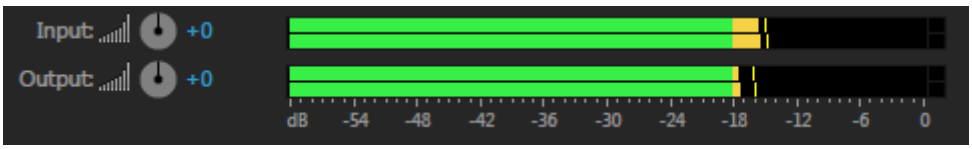


Figure 6 - Heart Breakfast (London) - Jamie Theakston [GLOBAL]

Figures 5 and 6 show levels taken from both female, and male radio presenters, though the female vocal has been attenuated marginally more than the male presenter, this does not seem to have significantly reduced the levels of the feed.

From the tests carried out on the audio samples, it can be concluded that under sampling the audio should not have a significant impact on determining whether to trigger a silence warning, provided a sensible trigger range is defined.

Based on the sample rate of 10 kHz, requiring 20 bits per sample to transfer the data over serial, a data rate of 200,000 bits per second is required, where the Arduino is only capable of sending 115200 bits per second (11.52 bits per sample). Therefore, the Arduino serial is not capable or sending enough information over the serial to satisfy the requirements, and therefore, the requirements of the project should be reassessed.

As the Arduino is only theoretically capable of sending 115200 bits per second, it would only be realistic to be able to sample one channel of audio as a compromise on the original proposals.

To prototype the sampling capabilities of the Arduino, a breadboard was setup with two 220Ω resistors as a potential divider, with a reading being taken between the power and top resistor, the setup shown below.

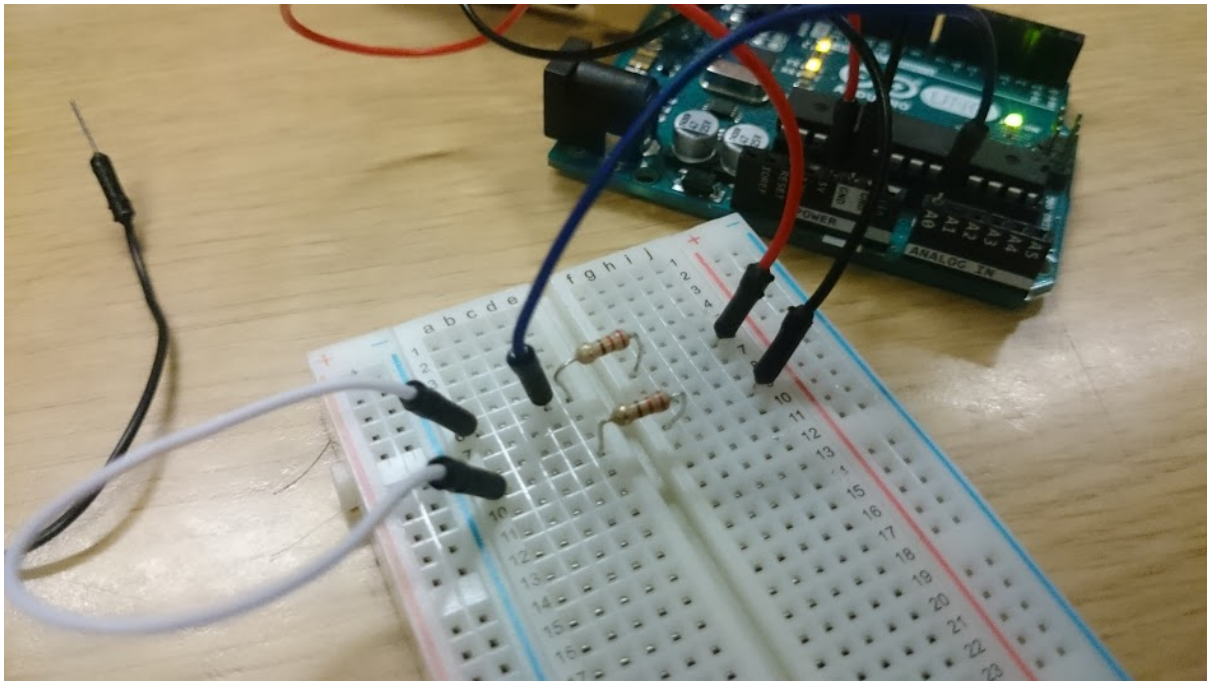


Figure 7 - Breadboard Power Supply

To test the device, an NTI Minerator MR1 would be used. The NTI Minerator MR1 will be used to generate a sine wave over a number of different frequencies. The Minerator MR1 has a balanced XLR input, and in order to input the signal into the breadboard, a specially terminated cable would be required. A test cable was assembled by removing a male adapter from an XLR cable, and replacing it with jumper cables, to produce a female XLR to jumper lead cable, as shown in the picture below.

Figure 8 Minerator test cable

The HOT and GROUND pins from the XLR testing cable were then inserted into the breadboard over the top resistor, in order to "lift" the voltage of the audio signal by 2.5 volts, in order to bring the full signal to within the sampling voltage range of the Arduino.

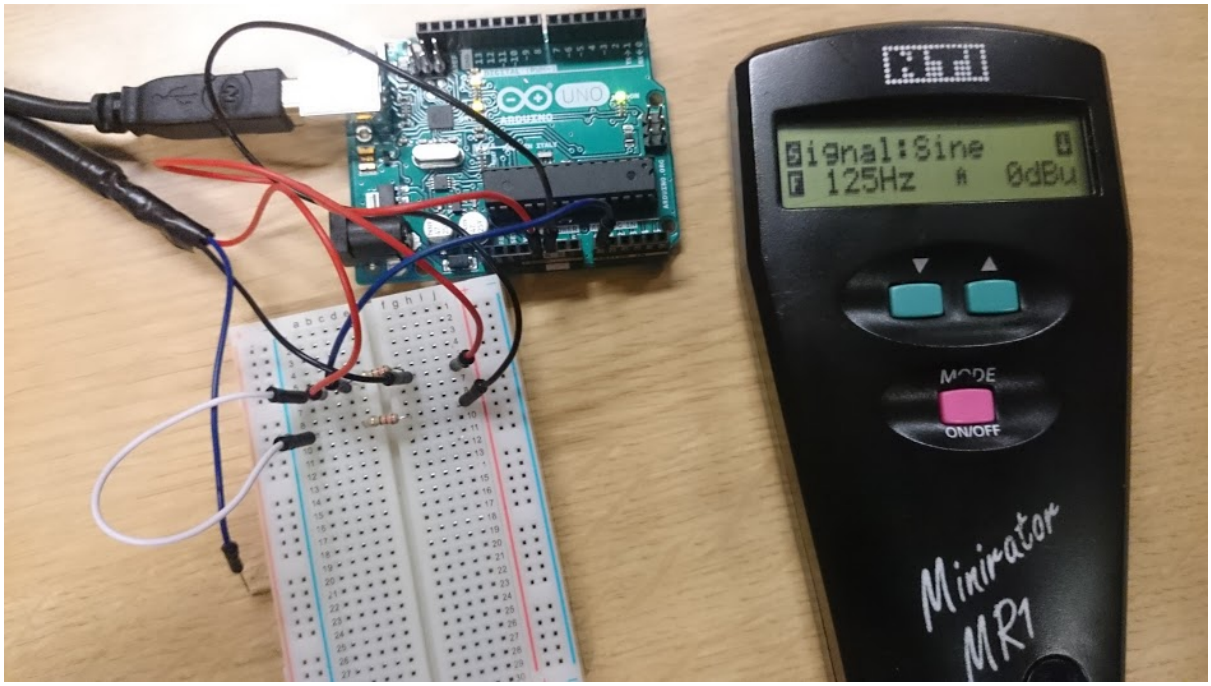


Figure 9 the bench test setup

Once the circuitry was complete, it would be necessary to test how well the setup would be able to sample the audio, and this would require code to be compiled to be written to the Arduino. To keep things simple, the initial software written would only need to read the analogue signal input, and feed it back to a computer which would be running Arduino serial monitor/plotter. Based on information from the Arduino documentation, the command required in order to read an analogue signal using Arduino is **`analogRead();`**.

```
1  int channel = 0; // Set Channel to Analog Pin 0
2  void setup()
3  {
4    Serial.begin(115200); //Setup serial connection
5  }
6
7  void loop()
8  {
9    Serial.println(analogRead(channel)); //Read CHANNEL and send via serial
   }
10
```

Results

The results when received from the Arduino initially began to look promising when first analysed in the Arduino Serial Plotter

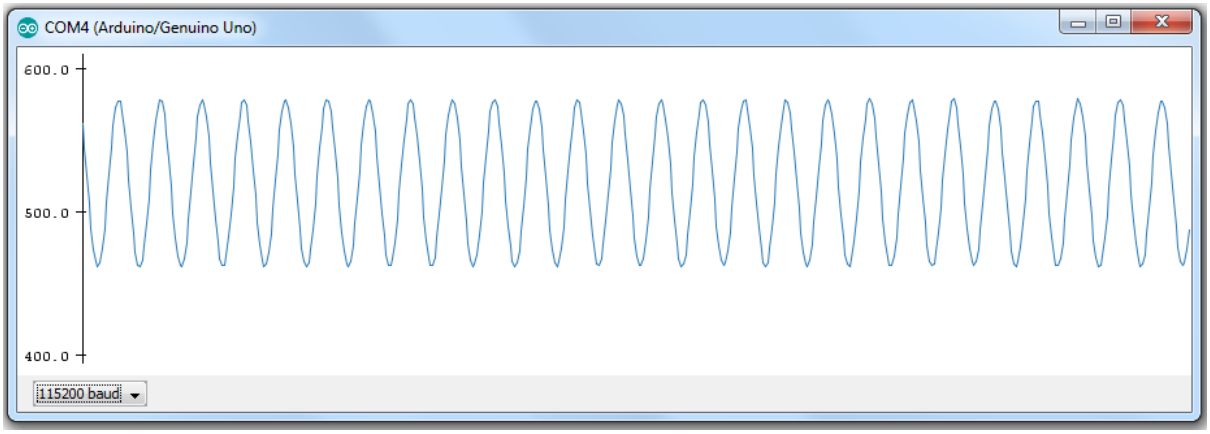


Figure 10 Arduino Serial Plotter (125 Hz)

The results from the plotter show that the zero level of the sine wave appears to fall around the 512 level, which is half way between 0 and 1023, where the level would be expecting to sit whilst lifted at 2.5v.

Based on the information previously available, the Arduino should be capable of sampling at 10 kHz. To prove that the device is capable of sampling at this rate, it would be realistic to be able to sample a sine wave up to the frequency of 5kHz.

At a test carried out at a frequency of 20 Hz a sine wave was reasonably reproduced, however did suffer from clock jitter, which would be perfectly acceptable to expect, as there is no sample clock for the Arduino, and the sample

rate is being limited by the speed at which the Arduino is able to read an analogue input pin.

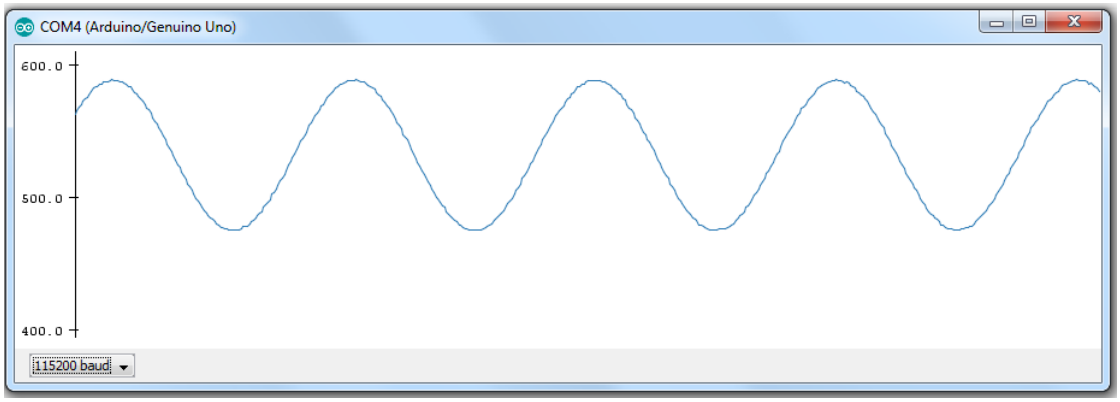


Figure 11 Arduino Sampling & Serial Test (20Hz)

Further tests were carried out at 25 Hz, 30 Hz, 40Hz, 50 Hz, 65 Hz, 80Hz and 100 Hz and the results seemed to be reasonably stable up to this point, with a sine wave being returned.

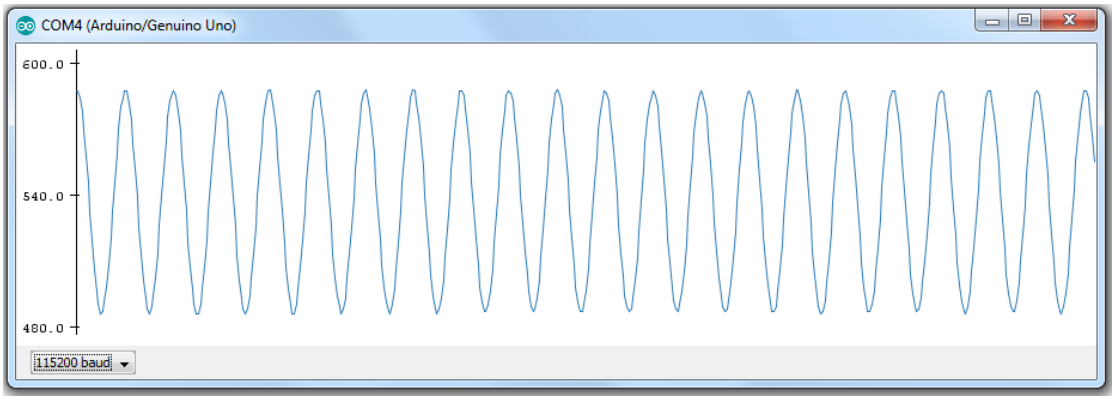


Figure 12 – Arduino Sampling and Serial Test (100 Hz)

When the frequencies tested starting getting higher, they began to start showing signs of quantisation errors, in the form of a more spikey and non-uniform waveform.

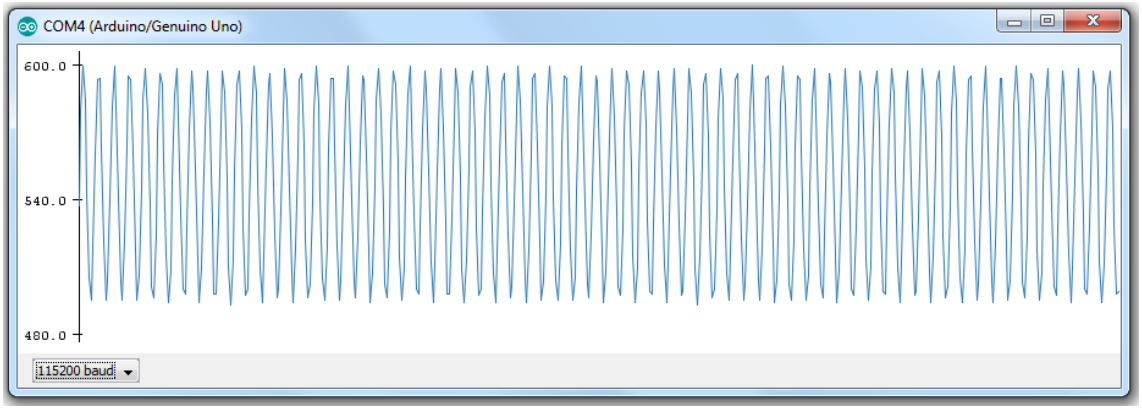


Figure 13 Arduino Sampling and Serial Test (315 Hz)

By the time that the frequencies reached 1.25 kHz, some signals had become extremely distorted, and were no longer looking like a sinusoidal wave. If this waveform were to be measured, it could return pulsating readings, which could cause problems when trying to identify samples below a trigger threshold.

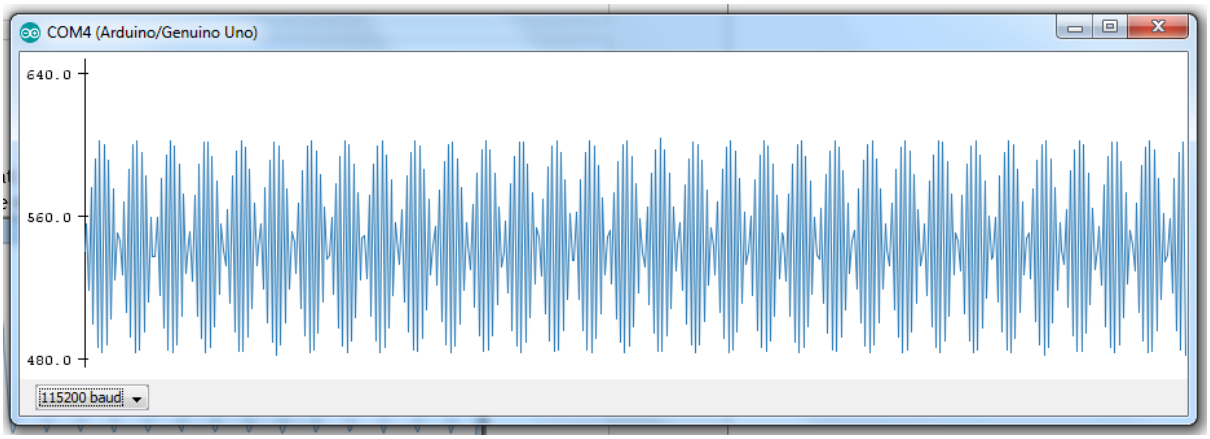
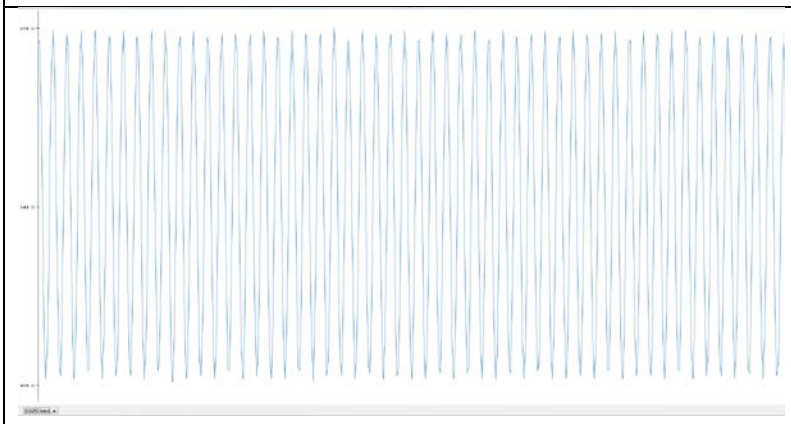
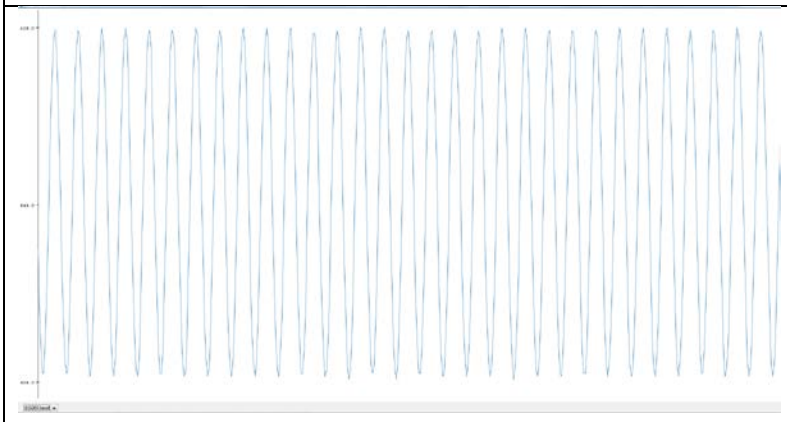


Figure 14 Arduino Sampling and Serial Test (1.25 kHz)

The reasons why the waves are distorted is due to the fact that the signals were being under-sampled. Taking a look at each of the waveforms revealed some interesting results, where the frequencies at 125 Hz, 250 Hz, and 2.5 kHz appeared to look like sine waves, particularly when neighbouring frequencies above 250 Hz were looking distorted. The information that the frequencies reveal, point to them being harmonics of each-other, linked to the rate at which the audio signals were being sampled at.

Table 5 - Harmonics helping identify sample rate

	*Figure 15 125 Hz Sampling Harmonic*
	*Figure 16 250 Hz Sampling Harmonic*
	*Figure 17 2500 Hz Sampling Harmonic*

The sample rate predicted from the information obtained by the tests (250 Hz) is significantly different to the rate of 10,000 Hz claimed by Arduino.

Due to the low sample rates from the Arduino, this project does not find the Arduino suitable for the task required.

Though an audio input is possible with frequencies under 250 Hz, and the problem with the high frequency distortion/aliasing errors could be rectified using a band pass filter, the frequency ranges for speech aren't adequately covered enough to satisfy the scope of the project.

Speech bands are located at a frequency of 250 Hz to 2000 kHz (Wiley, 1993), so only the lowest ends of speech could be picked, up, with most of the voice bands almost invisible to the Arduino.

The results of this project have concluded that it is not feasibly possible to design and build a broadcast-ready silence detection system using the consumer-level microcontrollers researched for the purpose of this project.

As it was not possible to satisfy the requirements of the base system, there is no way to further develop the other desirable features specified in the aims and objectives of this project.

Conclusion

This project looked in to whether a consumer-level microcontroller would be able to be used as a silence detection device in a radio broadcasting environment, and whether they could be used to improve on currently available silence detection units, without an excessive cost to the broadcaster.

From the development and testing of the Intelligent Silence Detection System project, it can be concluded that it is not possible to produce a broadcast-ready silence detection system using an Arduino and Raspberry, as they are not powerful enough to satisfy the minimum requirements of this project. Instead this project would recommend free software solutions and standard soundcards in order to implement a silence detection system on a low budget.

Recommendations

For smaller stations which require silence detection systems, the recommendation will be to use a software based system such as PiraCZ silence detection.

This can be achieved in a cost effective manner, as the hardware requirements are not too demanding, an old recycled PC should be sufficient to run the free software.

A basic line level audio card should be sufficient for the task, and for best results, should have its own feed from a distribution amplifier.

Instead of passing the audio through the computer, the computer could instead control a magnetically latching relay using a general purpose input/output or microcontroller over serial.

The device should be able to integrate or interact with the cloud, as it has the capability to send HTTP requests. The device will also be able to send text messages, as SMS messages can be sent by sending emails to a SMS Gateway company such as TextLocal.

References

4RFV.COM INTERNATIONAL BROADCAST NEWS, 2012. The BCD audio rack mount range.: BCD audio: International broadcast news [viewed 19 October 2015]. Available from: http://www.4rfv.com/0V6XBKVR75O0/the-bcd-audio-rack-mount-range.htm

AB ELECTRONICS, 2016. ADC pi plus [viewed 3 May 2016]. Available from: https://www.abelectronics.co.uk/p/56/ADC-Pi-Plus-Raspberry-Pi-Analogue-to-Digital-converter#

ARDUINO, 2015. IntelGalileo [viewed 9 November 2015]. Available from: https://www.arduino.cc/en/ArduinoCertified/IntelGalileo

ARDUINO, 2016a. AnalogRead [viewed 3 May 2016]. Available from: https://www.arduino.cc/en/Reference/AnalogRead

ARDUINO, 2016b. AnalogReadResolution [viewed 3 May 2016]. Available from: https://www.arduino.cc/en/Reference/AnalogReadResolution

ARDUINO, 2016c. Serial [viewed 4 May 2016]. Available from: https://www.arduino.cc/en/Reference/Serial

ATMEL CORPORATION, 2015. ATmega48A, ATmega48PA, ATmega88A, ATmega88PA, ATmega168A, ATmega1688PA, ATmega328, ATmega328P datasheet. [viewed 4 May 2016]. Available from: http://www.atmel.com/images/Atmel-8271-8-bit-AVR-Microcontroller-ATmega48A-48PA-88A-88PA-168A-168PA-328-328P_datasheet_Complete.pdf

AUDIO ENGINEERING SOCIETY, 2015. About the audio engineering society [viewed 10 November 2015]. Available from: http://www.aes.org/about/

BADDHAN, L., 2009. Power failure forces BBC radio off-air. bizAsia,

BBC radio 1 emergency tape kicks in on 2nd may 2009 with Vernon Kay, 2009 . YouTube

BUSH, K., 1978. Wuthering Heights. In: Wuthering Heights [CD]. EMI

COWAN, J.P., 1993. Handbook of environmental acoustics. John Wiley

EASTMAN, S.T., D.A. FERGUSON and R. KLEIN, 2012. Media promotion & marketing for broadcasting, cable & the Internet. CRC Press

EBU, 2004. EBU tech 3250-2004 specification of the digital audio interface (AES/EBU). [viewed 4 May 2016]. Available from: https://tech.ebu.ch/docs/tech/tech3250.pdf

Freberg, J., 2004. *Digital audio: Inside the AES3-2003 digital audio standard.* [Online]
Available at: http://www.tvtechnology.com/miscellaneous/0008/digital-audio-inside-the-aes32003digital-audio-standard/256297 [Accessed 10 11 2015].

Gruszka, M. C., 2008. *Unlocking the Mystery Of AES3 Digital Audio, Part 2.* [Online]

Available at: http://www.tvtechnology.com/audio/0014/unlocking-the-mystery-of-aes3-digitalaudio-part-2/200513
[Accessed 10 11 2015].

Heart Breakfast (London), 2016 [Radio]. Global Media. 3 May. 08:31

Intel, 2014. *Datasheet for Intel Galileo Gen 2.*
[Online] Available at:
http://download.intel.com/support/galileo/sb/intelgalileogen2prodbrief_330736_003.pdf [Accessed 9 11 2015].

Intel, 2015. *Where to buy Intel.* [Online]
Available at:
http://www.intel.co.uk/buy/uk/en/product/emergingtechnologies/intel-galileo-gen-2board-462144
[Accessed 9 11 2015].

JAMES, M., 2013. Ofcom site engineering code for analogue radio broadcast transmission systems broadcasting -radio. [viewed 30 April 2016]. Available from: http://stakeholders.ofcom.org.uk/binaries/broadcast/guidance/tech-guidance/code2013.pdf

LESURF, J., n.d. Stereo FM Radio [viewed 3 May 2016]. Available from: https://www.st-andrews.ac.uk/~www_pa/Scots_Guide/RadCom/part21/page1.html

Landau, H. J., 1967. Sampling, Data Transmission and the Nyquist Rate. *Proceedings of the IEEE,* 55(10), p. 1701.

MARSHALL, D., 2001. Nyquist's sampling theorem [viewed 3 May 2016]. Available from: https://www.cs.cf.ac.uk/Dave/Multimedia/node149.html

MARTIN, R., 2015. Fire leaves talkSPORT off-air [viewed 1 May 2016]. Available from: http://radiotoday.co.uk/2008/10/fire-leaves-talksport-off-air/

Microchip, 2015. *MCP3008.* s.l.:s.n. MICROCHIP TECHNOLOGY INC (2009). 18-Bit, Multi-Channel ΔΣ Analog-to-Digital Converter with I 2 CTM Interface and On-Board Reference

Microsoft, 2015. *FAQs.* [Online]
Available at: http://ms-iot.github.io/content/en-US/Faqs.htm#galileo [Accessed 9 11 2015].

Microsoft, 2015. *Getting Started.* [Online]

Available at: https://ms-iot.github.io/content/en-US/GetStarted.htm [Accessed 9 11 2015].

NELSON, P., 1994. The Most Beautiful Girl in the World. In: The Gold Experience [CD]. Warner Bros., NPG

NTI Audio, 2012. *AES3, AES/EBU Application Note.* Unknown: NTI Audio.

PLUNKETT, J., 2016. TalkSport back on air after studio fire. The Guardian, 22 January

RS Components, 2015. *Raspberry Pi 2 B.* [Online]
Available at: http://uk.rs-online.com/web/p/processor-microcontroller-development-kits/8326274/ [Accessed 9 11 2015].

SAM FM, 2015. We've got the Norman Collier's. Please bear with us whilst those squirrels from the Carling ad go up the mast. In: Twitter. 4 February 2015 [viewed 1 May 2016]. Available from: https://twitter.com/samradiouk/status/563049090192510979

Shah, A., 2014. *Microsoft's custom Windows OS now on Galileo Gen2 board.* [Online]
Available at: http://www.pcworld.com/article/2691252/microsofts-custom-windows-os-now-ongalileo-gen2-board.html [Accessed 09 11 2015].

WALTERS, D., 2006. How to build a radio station. Lulu Enterprises, UK