Design of a Selective Calling Decoder for Radio Transmission

Internship Submission

Caitlin Barnes
Academic Supervisor: Gareth Lee
Industry Supervisor: Simon Chan
ABSTRACT

On large remote mine sites there is always a possibility of accidents occurring. These accidents need to be well recorded and understood to help prevent a similar accident occurring. In order to do this it is important to know who was recently in communication with anyone involved in the accident.

One of the ways to achieve this is by identifying which radios are being used for communication purposes around the time of the accident. This can be achieved as the majority of radios use Selective Calling (SelCall) tones and Automatic Number Identification (ANI) codes for identity purposes (Bailey 2003), (Weik 2012). This can be completed by using a Selective Calling Decoder.

This Selective Calling Decoder will need to have the ability to decode the SelCall values into their respective ANI codes and time stamp the date and time that they were used. These values will then need to be uploaded to a server and stored for future use. These should be stored in an easily accessible format and the server should include an ability to look up this information by employee ID, radio ANI code, date or time of transmission. Some extra features that were requested but not required included the ability to run the device on a multichannel radio system and the design of a website link to the server allowing remote access. This SelCall Decoder was to be a standalone device that could be easily connected into an existing system. RadLink Communications provided a controller named the BVR5000 to be used for this purpose.

The project has been completed to a simulation level of testing and has not yet been implemented into a standalone device. The testing in simulation involved reading a sound file stored on the same device as the SelCall Decoder, running the data from this file through a tone identification algorithm and outputting the determined ANI codes and a corresponding time stamp to a display on the screen of the device.

Unfortunately there were some aspects that were not completed in the period of this project. These included the ability to read in from an audio line in real time and to output the data in the correct format to the client’s server. Also the completed program was unable to be uploaded onto the standalone device for further testing and implementation. This was caused by a delay on design of the BVR5000. The multichannel reading ability was considered but not checked in this phase and will also need further work completed on it. A final section that was not considered during this period was the creation of a web page to link to the server to allow for remote access. These section should all be completed before the project is made live.

Whilst the project has not been completed it has been considered a partial success. The section that has been completed is fully functional in simulation and can be seen as a successful proof of concept. The completion of the project was unsuccessful mostly due to a delay in the completion of the BVR5000 and as such was outside of the author’s control.
# CONTENTS

Abstract ................................................................................................................................. i

Contents .............................................................................................................................. ii

List of Figures ....................................................................................................................... iv

List of Tables ......................................................................................................................... iv

Glossary of Abbreviations .................................................................................................... iv

Acknowledgments ................................................................................................................ v

1. Introduction ...................................................................................................................... 1

   1.1 Internship ..................................................................................................................... 1

   1.2 RadLink Communications .......................................................................................... 1

      1.2.1 Beyond Voice Radio ............................................................................................... 2

   1.3 Project Brief .................................................................................................................. 2

   1.4 Commercially Available Alternatives ......................................................................... 3

2. Literature Review ............................................................................................................. 4

   2.1 Radio Communication ................................................................................................. 4

      2.1.1 Terrestrial Trunked Radio ..................................................................................... 4

      2.1.2 Noise ...................................................................................................................... 4

   2.2 Selective Calling ........................................................................................................... 4

   2.3 Automatic Number Identification ............................................................................... 5

   2.4 Tone Identification Algorithms ................................................................................... 6

      2.4.1 Fast Fourier Transform ......................................................................................... 6

      2.4.2 Goertzel Algorithm ............................................................................................... 6

   2.5 C# and the .NET Platform ........................................................................................ 6

      2.5.1 Object Oriented Programming ............................................................................... 6

      2.5.2 Type-Safe Programming ....................................................................................... 7

      2.5.3 Memory Management ............................................................................................ 7

      2.5.4 Platform Support ................................................................................................... 8

3. Project Design .................................................................................................................. 9

   3.1 Reading from Input ...................................................................................................... 9

      3.1.1 Continuous Data Acquisition ............................................................................... 9

      3.1.2 Package Data Acquisition .................................................................................... 10

   3.2 Data Formatting .......................................................................................................... 10

      3.2.1 Data Set Up .......................................................................................................... 11
LIST OF FIGURES

Figure 1: WAV file Header ........................................................................................................... 10
Figure 2: Goertzel Layout ........................................................................................................... 11
Figure 3: Coefficient Set Up ........................................................................................................ 15
Figure 4: Overall Goertzel Algorithm .......................................................................................... 15
Figure 5: Setting Base Magnitude ............................................................................................... 16
Figure 6: First Level Filter .......................................................................................................... 17
Figure 7: Second Level Filter ....................................................................................................... 17
Figure 8: Final level Filter ........................................................................................................... 18
Figure 9: Example Sound Received .............................................................................................. 20
Figure 10: Test Graph of Sound Section ...................................................................................... 20
Figure 11: Sound Wave When Opened in Audacity ................................................................. 21
Figure 12: Testing Data Offset .................................................................................................... 21
Figure 13: Tone Identification Test ............................................................................................. 22
Figure 14: Data after First Filter .................................................................................................. 23
Figure 15: Data after Filter .......................................................................................................... 23
Figure 16: Data after Structuring ................................................................................................. 24
Figure 17: Sound File for ANI Code 12345 ................................................................................ 24
Figure 18: Output for ANI Code 12345 ..................................................................................... 25
Figure 19: Final Output ............................................................................................................... 25
Figure 20: Input Sound ................................................................................................................ 26

LIST OF TABLES

Table 1: SelCall Tones adapted from HF Link ........................................................................... 5
Table 2: Coefficients for N = 50 .................................................................................................. 12
Table 3: Coefficients for N = 500 ............................................................................................... 13
Table 4: Coefficients for N = 320 ............................................................................................... 14

GLOSSARY OF ABBREVIATIONS

SelCall  Selective Calling
ANI  Automatic Number Identification
TETRA  Terrestrial Trunked Radio
AWGN  Additive White Gaussian Noise
SDA  Short Data Applications
FFT  Fast Fourier Transform
DFT  Discrete Fourier Transform
SCADA  Supervisory control and data acquisition
OOP  Object Oriented Programming
WAV  Waveform Audio Format
CSV  Comma Separated Values
ACKNOWLEDGEMENTS

I would like to acknowledge the assistance of Simon Chan for offering me an internship and for his guidance provided during the course of this project. I would also like to acknowledge Gareth Lee for his academic support offered throughout the duration of this internship as well as the other staff within the School of Engineering and Information Technology at Murdoch University that have had the pleasure of assisting with my learning over the duration of my undergraduate degree.

It is also important to have support of friends during times such as these and I would like to acknowledge the support of my work colleagues Ben, Bryant, Earnest, Nick, Peter, Mark, Cleo and David for helping make RadLink an enjoyable place to work and providing advice and guidance on the work aspect of my internship when necessary. This support was also provided by my fellow thesis students Ash, Kieran, Ali, Liz and Tim who helped provide advice and corrections on the written and verbal presentations in the unit.

Finally I would like to acknowledge my indoor beach volleyball teams and my family and friends for providing me with an outlet during this time.

- Caitlin Barnes
  June 2015
1. INTRODUCTION

1.1 Internship
The requirements of bachelor of Engineering degree at Murdoch University includes a need to undertake either a research thesis or a relevant industry placement internship. The completion of an internship helps to provide students with knowledge and experience in an industry setting which can be invaluable at the completion of the degree. This internship was completed at RadLink Communications in the Research and Development department over the period of October 2014 to February 2015. This internship was involved in completing smaller sections of a large project that has been running for two years prior to the inclusion of the internship. This project involves testing and implementing small projects using the recently developed BVR5000 controller. The project was expected to run over a period of 16 weeks and involved the completion of supplementary submissions to show if the project was viable, well thought out, on track and any changes or problems that could slow the project down before completing a formal report and presentation on the work carried out throughout the period at RadLink. Overall this project helped with the application of skills learnt during the completion of the Bachelor of Engineering at Murdoch University and allowed an exposure to the differences that occur in a professional environment such as interconnected projects, budgeting, management and operations.

This report details the main project that was completed by the student during the internship at RadLink Communications.

1.2 RadLink Communications
RadLink Communications is a West Australian communications company that was founded in 2007. This company works towards implementing communications networks for a vast range of sectors including the mining, oil and gas industries. One of the advantages of RadLink over most communications companies is the ability for them to use a diverse range of radio suppliers as they are not separately affiliated with any one company (for example Motorola and SIMICO) allowing a larger range of flexibility for the company to provide what is required by the client. As a local company RadLink also has the ability to have a quick response to any problem that occurs and allows for quick and easy communication with the client. RadLink provides the clients with a Radio solution from start to finish or sections as required. This includes network design, implementation and future maintenance and upgrading of all systems. Normally this is done for larger systems but occasionally small systems are required and occasionally just short term leases of equipment is required for events (e.g. Concerts, large meetings). Some of the clients for RadLink include those in the oil, gas and, mining industries, shopping centres, hospitals, universities, local governments, utility companies and emergency service (RadLink Communications 2015).

As well as working with basic radio supply RadLink is aiming to improve upon their services by providing SDA capabilities along with the voice radio. This allows them to expand the uses for radio to enable them to be used for data collection and written instructions allowing for better communication (RadLink Communications 2015). In order to fully take advantage of the SDA capabilities of radio RadLink has created a separate research division to investigate further this capability and create external hardware that is able to fully harness SDA’s. This division is called Beyond Voice Radio (Beyond Voice Radio 2015).
1.2.1 Beyond Voice Radio

Beyond Voice Radio is a subsidiary of RadLink Communications that has been operating for approximately two years but has only recently started separating from RadLink. The main purpose of Beyond Voice Radio is to investigate the abilities presented by SDA’s and expand upon them. In order to achieve this they have been working on the creation of a controller named the BVR5000 that works mainly to allow the radios to communicate with equipment directly as well as supplying storage capabilities for data. This controller allows for the TETRA network to be expanded supporting both voice communication and data communication. Some of the advantages this presents are the ability to use this data to send messages (such as for machine break downs), record data into a SCADA system, and allowing one device to possess a larger range of communication capabilities. Whilst primarily working on producing the controller, Beyond Voice Radio is also working on discovering more capabilities for all sorts of radio networks big and small and each individual radio (Beyond Voice Radio 2015).

1.3 Project Brief

The goal of this project is twofold. Firstly is to demonstrate some of the abilities of the recently developed BVR5000 and secondly to produce a product that is of use to a client company and can be tailored to their specific needs.

The project being completed involves writing an application that was indicated as being of interest to one of RadLink’s clients. This project involves designing a SelCall decoder with the ability to decode multiple tone sets, lengths and number of tones. This is a viable project as the client is unable to find an off the shelf decoder with the flexibility needed for the system they use. This decoder should be able to record the tones used, determine the set ANI code for the set of tones, timestamp the data and upload all the information to a server in an easily accessible format. It was also suggested by the client that the device should use RJ45 connectors, operate on a 600Ω balanced audio or unbalanced audio, pass all pins through so that the device can be inserted into an existing system, have receive audio lines selectable to allow flexibility, combine both RX and TX audio lines to record both radio traffic directions, Pass decoded ANI and timestamps to a separate system in an easily accessible format for logging on a server, operate as single channel units or be cascaded to form a multichannel device, decode numerous variable SelCall formats and contain a phone book/lookup table to match aliases to ANI decodes. This design needs to be quite robust. On top of this, a website is to be designed with the ability to display the received codes in real time, as well as having a search capability via time or ANI code.

This decoder is to mainly be used for incident investigation and as such needs to be quite robust and reliable. For consideration in this is the ability to self-detect when an error has occurred and possibly be able to repair it or to at the least mark the data as untrustworthy or corrupt. Also of some use would be the ability to determine which tones have been corrupted. If not all so as to be able to print the tones that were recorded correctly for use by the investigation to at least narrow down the search that is being conducted.

In order to be able to complete this project some things need to be taken into consideration. This includes noise, error handling and flexibility. With the transmissions that are being decoded there is expected to be a reasonable amount of noise from surrounding sources such as other people attempting to use the channel to communicate, background noise (trucks, drills etc.) and any static that could have been picked up in the system. Due to the high level of noise the decoder has to first be able to distinguish the specific frequencies of the tones before determining if there are more than one tone set layered within a transmission and separating them out to be able to decode both of the
sets separately. This can be quite tricky and leads to our next consideration, error handling. As the design has to be robust any errors that occur would need to be found and documented quite specifically as to be able to determine what went wrong and where so that it can either be fixed or at the very least understood.

1.4 Commercially Available Alternatives
The idea of decoding SelCall codes is not a new one and previously companies have created similar devices to complete this. An example of one of these devices is the TrelInnovatorer AB SelCall CCIR Decoder Cl-100 (Trelnovatrer AB n.d.). Whilst these provide a similar service to that which is required in this project there are quite a few problems with them. In general most decoders are designed for aircraft transmission which uses a different style of SelCall (sending multiple tones at the same time, less tones, shorter time periods) and the decoders are all quite specific with their requirements. For those that are designed for the style of transmission presented in this case the decoders are designed usually for either transmission lengths of five or seven tones not both and will only be able to determine one set of SelCall tones per transmission. This is not viable for this project as the transmissions will be in both five and seven tones and may occasionally have more than one set of tones per transmission. The advantages of designing this also involve an ability to design it specifically to the system that is already in place and being able to cater to any individual quirks of that system.

From the perspective of RadLink, completing this project enables for the BVR5000 to be displayed to a client company that may be interested in using it more in the future for other usage as it will show the reliability of the controller and some of its basic functions.
2. LITERATURE REVIEW

In order to get a good understanding of this project there are sections that require to have a thorough level of research. These sections include the basics of radio communication, SelCall and ANI codes, tone identification algorithms and the C# language and .NET platform that are to be used in this project. Each of these sections can provide some answers as to how this project can be best approached to deliver the outcome that the client wants.

2.1 Radio Communication

Radio communication uses the wireless network to transmit sound from one location to another. This can be done in either one way or two way transmissions. In two way radio each radio is equipped with both a receiver and a transmitter. For this project the radios in use were half-duplex. This means they can be used to transmit or receive but not both concurrently. In the wireless spectrum each frequency can only be used by one radio at one time within a certain distance or risk corrupt transmissions. In order to account for this, radios normally come with a range of frequencies. If one frequency is in use the radios can be tuned to other frequencies. These frequencies are known as channels. However if a radio wants to be able to listen to a group of other radios these frequencies have to be known to the specific radio. Radio now uses both digital and analogue styles to communicate. This project was created for analogue radio transmission (RadLink Communications 2014).

2.1.1 Terrestrial Trunked Radio

Whilst having multiple channels allows for a larger amount of radios to be able to communicate concurrently in some cases there are not enough channels available. This is likely to occur in places such as large mine sites. In order to account for this large demand Terrestrial Trunked Radio (TETRA) was implemented. TETRA radio works by wrapping a control system around a set of radio channels. This allows for the channels to be assigned on demand and freed when a transmission is finished. By using this a few channels can cover a large area (RadLink Communications 2014).

2.1.2 Noise

One of the major problems that can be found in analogue radio is the transmission of noise. The majority of noise that occurs in radio transmission is channel noise. This noise has a flat power spectrum density and is known as Additive White Gaussian Noise (AWGN) (Chandrasekhar 2007). AWGN refers to noise that is added to intrinsic noise to a system, has a uniform power across the frequency band and has a normal distribution in the time domain with an average of zero (Linnartz 1995). This noise was a major consideration in this project.

2.2 Selective Calling

In a radio network where a large group of people are using the same channel it is sometimes needed for people to limit what is sent to whom. In this situation SelCall is used to address certain radios or stations to send the data communications. SelCall works by transmitting a set of predetermined tones to all stations and radios on the network. These stations have a single or set of tones that it has been programmed to accept or decline. By sending this across first it enables the radio to accept or decline the tone set before the operator knows there is a transmission on the station. There are multiple groups of tones that have been chosen for this with some examples that can be seen in Table 1. This style of communication makes it simpler to limit the communications on a radio channel to only those that need it. The standard tone set sizes are five or seven tones long and come in lengths that range from 20 milliseconds to 1 second (HF Link 2000) (Weik 2012).
Using SelCall on a radio channel can lead to some problems occurring if it is not used well. This includes the possibility that a user has declined the SelCall tones attempting to use the channel for a communication and thus cutting off the original communication. Most SelCall systems are designed to account for this but it is important to consider when implementing the system.

Whilst the original use of SelCall does not involve the tones being heard by the users a further use of end of transmission SelCall has been developed on site. This uses a SelCall address for each radio on the system as a set of squelches to end the transmission and can be heard by the receiver. This is the style of SelCall that is used in this project. Each of these tone sets is different for each radio which allows for outside programs to listen in and determine which radio is communicating. This ability is very important for this project (RadLink Communications 2014).

2.3 Automatic Number Identification

Automatic Number Identification (ANI) is used in conjunction with SelCall tones in radio communications. ANI codes contain a set of hexadecimal numbers from zero to F. These numbers are seen in Table 1. Each of these numbers have a corresponding SelCall tone (may be different depending on the SelCall protocol used). These ANI codes are usually correlated to a vehicle or location where a radio is stored. The original purpose for this was to be able to identify when a problem occurs on the network and allows radios that are not communicating to be identified (Bailey 2003).

Table 1: SelCall Tones adapted from HF Link

<table>
<thead>
<tr>
<th>Digit</th>
<th>CCIR (Hz)</th>
<th>EEA (Hz)</th>
<th>EIA (Hz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1981</td>
<td>1981</td>
<td>600</td>
</tr>
<tr>
<td>1</td>
<td>1124</td>
<td>1124</td>
<td>741</td>
</tr>
<tr>
<td>2</td>
<td>1197</td>
<td>1197</td>
<td>882</td>
</tr>
<tr>
<td>3</td>
<td>1275</td>
<td>1275</td>
<td>1023</td>
</tr>
<tr>
<td>4</td>
<td>1358</td>
<td>1358</td>
<td>1164</td>
</tr>
<tr>
<td>5</td>
<td>1446</td>
<td>1446</td>
<td>1305</td>
</tr>
<tr>
<td>6</td>
<td>1540</td>
<td>1540</td>
<td>1446</td>
</tr>
<tr>
<td>7</td>
<td>1640</td>
<td>1640</td>
<td>1587</td>
</tr>
<tr>
<td>8</td>
<td>1747</td>
<td>1747</td>
<td>1728</td>
</tr>
<tr>
<td>9</td>
<td>1860</td>
<td>1860</td>
<td>1869</td>
</tr>
<tr>
<td>A</td>
<td>2400</td>
<td>1055</td>
<td>2151</td>
</tr>
<tr>
<td>B</td>
<td>930</td>
<td>930</td>
<td>2433</td>
</tr>
<tr>
<td>C</td>
<td>2247</td>
<td>2400</td>
<td>2010</td>
</tr>
<tr>
<td>D</td>
<td>991</td>
<td>991</td>
<td>2292</td>
</tr>
<tr>
<td>E</td>
<td>2110</td>
<td>2110</td>
<td>459</td>
</tr>
<tr>
<td>F</td>
<td>1055</td>
<td>2247</td>
<td>1091</td>
</tr>
</tbody>
</table>
2.4 Tone Identification Algorithms
One of the most important sections of this project is the ability to decode the incoming sound files into tones. There are a couple of different algorithms that can do this in different ways which can be considered for this project. Two of the most useful algorithms for this are the Fast Fourier Transform (FFT) and the Goertzel Algorithm. Both of these algorithms were investigated in depth to determine which one was most suitable for the project.

2.4.1 Fast Fourier Transform
FFT is an implementation of the Discrete Fourier Transform (DFT). It is one of the most widely used in digital signal processing. The FFT transforms works by mapping a sequence f(x) into a frequency spectrum. This can then be used to determine the frequencies in the signal. The most common form of FFT used is known as the Cooley-Tukey algorithm. To use this the transform is broken down into two halves at each step limiting the algorithm to a power of two. Whilst the algorithm is recursive by nature most implementations will rearrange it in order to avoid explicit recursion. One of the advantages of this algorithm is that it breaks a large DFT into smaller DFTs. This ability allows it to be combined with other DFT solving algorithms (K. R. Rao 2011). Whilst this algorithm cuts down on computational power needed compared to the traditional algorithm it is highly likely that it could be difficult for the BVR5000 to handle.

2.4.2 Goertzel Algorithm
The Goertzel Algorithm is also a DFT technique that is used in signal processing and embedded systems. This algorithm works be using a very simple recursive formula. This Algorithm has very strong computational advantages over a FFT, a highly regular structure and can be easily cascaded. In this project the computational cost and possibility of cascade sections were very important (Glisic 2004). This was seen as being very attractive for this project. This formula involves calculating individual coefficients for each individual tone and running this through the algorithm to search for the magnitude of a frequency. As many of the values are set they can be precomputed and hard coded into the controller (Lyons 2012). Overall The Goertzel algorithm was seen as being more suited to the use in this project.

2.5 C# and the .NET Platform
For this project it was indicated that C# should be used for the programming of this application. As the author had not been introduced to this language prior to this project it was important to gain a basic understanding of how to use it. C# is the flagship programming language from Microsoft and has recently been updated to its fifth iteration C# 5.0 (Joseph Albahari 2012).

C# was designed to increase programmer productivity. As such it has a balance of simplicity, expressiveness and performance. Whilst the language itself is platform neutral, it has been designed to work quite well with the Microsoft .NET platform (Joseph Albahari 2012). Some of the important aspects of the language that need to be investigated before it can be used to its full potential are how the object oriented style of the program (OOP), the type-safety, the memory management and the platform support.

2.5.1 Object Oriented Programming
C# uses classes as an underlying structure and as such is a true object oriented language. Any program components should be encapsulated in a class which create and define the state and behaviour of an object. These objects then use methods in order to communicate with other objects. These classes should be designed to incorporate the three basic pillars of OOP which are encapsulation, inheritance and polymorphism (Balagurusamy 2010).
Encapsulation allows the ability to hide details of an object from the users and may stop them from having their state directly altered. This is done by using the keywords public, private and protected which determine the different levels of access that is granted by the class. Whilst not able to directly alter the state of an object it is possible to alter it indirectly using accessor or mutator methods. These methods are not used in this project and thus will not be explained further. An advantage of this type of programming is that it allows for software to be tested independently of the rest of the program (Balagurusamy 2010).

Inheritance is when an object is created based off another object or inherits sections of that object. In this situation the top level class is known as the parent or base class and the modified one is known as the sub, child or derived class. As classes should only contain necessary information subclasses can be used to add information that is not always used in the base class but may be needed occasionally. It is possible to have multiple subclasses for one base class allowing for multiple classes that use the same base data to be created easily. Overall this system helps to simplify the program as only extra components need to be stored in subclasses instead of needing to create a brand new class for a slightly altered object (Balagurusamy 2010) (Yaiser 2012).

Polymorphism is the concept of when multiple objects can be used to complete the same task. A real life example of this is when both a car and a bike can be used as transportation. Interfaces are used in this instance to allow for both objects to be queried. This helps in simplifying the program when using a large amount of methods with varying data types. An example of how this helps is when an integer is used as a floating point number in a mathematical function. In this case the basic value of these can be the same but an extra line may be first needed to convert an integer to a floating point if polymorphism is not in place. These extra lines would add up in a large program and could lead to more errors. However the opposite (using a floating point as an integer) should not occur and is investigated in type-safety (Yaiser 2012).

2.5.2 Type-Safe Programming
A type-safe language is one that only allows types to interact through protocols they define. This helps to keep each types internal consistency. This means that it is impossible to interact with one type as though it was another, for example treating an integer as a double or string. C# itself is a strongly typed language due to its rules regarding types being very strict. An example of this is that in order to use a floating point number in a function that only accepts integers it must first be explicitly converted by the user. This strictness of type-safety enables the host to have control over every aspect of security. This also ties in with the object orientation by not allowing the object type to be corrupted arbitrarily by bypassing its type rules (Joseph Albahari 2012).

Another important aspect of the C# type-safety is that it is enforced in both compile time and runtime. Enforcing the type-safety at compile time is known as static typing and allows for errors to be determined before the program is run. By doing this C# can work with other programs to make these errors easier to find and fix as well as allowing the program to be easier to manage, more robust and more predictable (Joseph Albahari 2012).

2.5.3 Memory Management
Memory management is very important in the design of programs. C# makes it quite simple by using automatic memory management. This has a garbage collector that executes as part of the program which collects and reallocates memory that is no longer referenced. This helps by making most memory deallocation hidden from the user and removes the need for pointers in most programs. Whilst they are not needed for most programming tasks the language still allows for pointers to be
used in tasks that are explicitly marked as unsafe by the user. This allows for them to be used in performance critical hotspots (Joseph Albahari 2012).

2.5.4 Platform Support
C# was primarily designed to run on a Windows platform and does so extremely well. Outside of the Windows platform however the amount of resources and support is relatively small even with the language standardized through Ecma International. Due to this fact a language such as Java would be more useful in a program that is expected to run on multiple platforms. There are three main ways that C# can made cross-platform compatible. The first of these is to run C# code on the server which can then output HTML (which is able to run on any platform), an example of this is found in ASP.NET. A second option is that C# has the ability to run on other runtimes outside of the Microsoft Common Language Runtime, an example of this is the Mono Project which has its own C# runtime. Finally it is possible for C# to run on any host that supports Microsoft Silverlight (which is supported on Windows and Mac OS X) which is quite similar to Adobes Flash player (Joseph Albahari 2012).
3. PROJECT DESIGN

This section aims to outline how the project was designed and built in order for the device to be created well and on time. Secondly any pertinent information for the project that has not already been outlined in the literature review will be included in this section.

For this project the design was separated into five sections. These were to read the input data, manipulate the data into a format that can be tested for tones, filtering and structuring of the tones outputted and finally time stamping and outputting to the server. Each of these sections involved different types of work and challenges.

3.1 Reading from Input

In order to be able to read from the input it is important to first be aware of the format of the input data. This was not supplied in the project outline and attempts to determine this have been unsuccessful. In order to account for this level of flexibility a couple of general designs have been implemented with different input reading but identical outputs to the second section of the project.

It was assumed that the data would be inputted in either a continuous form, directly from the radio in real time or in a discrete package form, uploaded from the radio at the end of each transmission. As such these are the two forms of data input that have been investigated and the program designed to accommodate. Another form that is possible though unlikely is the arrival of a large chunk of data; for example at the end of the working day. This form has not been investigated to a large level but has been accounted for in later areas of the project.

3.1.1 Continuous Data Acquisition

The idea for continuous data acquisition stems from the desired use of the product. Whilst not specified it is likely that the SelCall decoder will be connected to a radio and will this ‘listen’ to the radio communications in real time. This would lead to a more difficult level of data acquisition than assumed in this project and would need a further level of investigation and the program to be adapted. This is a preliminary look as to how this could be completed.

The first important area is to determine how the radio transmissions will be recorded. As there was little data supplied on this it will be assumed that it will be received in one of two ways. This could be via a serial wired connection to a radio with a reasonably constant upload rate or via a recording device placed next to the radio transmitter. Whilst both of these would be reasonably similar from a programmatic point of view it should be noted that the former would most likely provide a clearer signal due to the fewer steps between the transmissions. However this is dependent on the type and complexity of the type of radios that are used by the client. The latter would provide an easier design as it would enable the programmer to know all of the hardware used and to design around it. For either case it can be assumed that the data would be arriving as groups of magnitudes representing the sound wave.

In order to read the continuous data in a way that would make it simple to format would involve setting a section size of data to read and record to the program before storing other incoming data into the buffer to be run through after each section has been completed. This would enable the data to be stored in a way that would also be usable for package data acquisition if this is used instead. A problem that could occur using this style of data packaging is the package ending part way through a SelCall tone set. This could lead to the tone set not being picked up. A way that this could be tackled is by adding a section that does a pre-check on the data and looks for a collection of magnitudes that hint towards silence or low noise levels on the transmission line as a cut-off point.
This was not investigated further due to the large variables and inability to test using the received data. However it may need to be revisited as part of future works for the project when it nears completion.

### 3.1.2 Package Data Acquisition

The design for the package data arrival was based off some example radio samples that were supplied by the client. These samples were provided in the form of a WAV file for each set of transmissions that were included.

In order to be able to read in from a WAV file it is important to understand the format of the file. The file for a WAV file starts with a header that helps to provide the important information such as data size, sampling rate, byte rate and if Pulse Code Modulation (PCM) is being used. The header is followed by the data which is provided as consecutive magnitudes that when combined into a continuous graph provides a waveform. In most cases there are two separated waveforms designed for a left and right speaker. This is not applicable in this situation as there is only one feed in these radios. The data is followed by a stop bit which signals the end of the file (Sen M. Kuo 2006).

Reading of the file was done by adapting an existing template that was found online (yomakkk 2012). This allows for the separation of the header into the correct arrays whilst storing the data as it is useful in future sections. This storage area can be seen in Figure 1. The data is then read into a list format. This is then converted into an array to allow for the format to be consistent with that of the continuous data acquisition style. This allows for a simplification of future sections.

![WAV file Header](image)

**Figure 1:** WAV file Header

### 3.2 Data Formatting

After receiving and writing the data into an easy to read format the next step is to set up the data into a format capable of being passed through the Goertzel algorithm and to set up the Goertzel algorithm so as to be able to run the data through the algorithm correctly.
3.2.1 Data Set Up

The data is read in via an array that was set up in the previous section. This array is a collection of magnitudes at the specified sample rate. It was determined that the best way to sort through the data was to separate it into smaller sections to pass through the Goertzel Algorithm. The size of the sections being passed through the Goertzel would need to be equal or less than the length of the tones to be effective. With a tone length of 40ms the section size was arbitrarily set at 20ms. This turned out to be problematic when combined with the Goertzel set up as the section size had a relation to both the sampling rate and the number of samples in the Goertzel. The sampling rate was recorded in the WAV header and has previously been found as 8000. The number of samples that were needed for the Goertzel to work efficiently was found as 320 or above. This will be explained in the Goertzel Set Up section. With these constraints the section size needed to be 40ms or greater. Combined with the need for the section size to be 40ms or less a section size of 40ms was chosen.

Whilst within the reasonable range this value could lead to tones being missed if they sit perfectly between two sections. In order to account for this it was decided that the sections should overlap by 20ms. This led to the data set being doubled in size which may cause memory problems when placed on the controller. This was determined to be a reasonable loss for the increased level of detail but may need to be reconsidered if the program grows too large or appears to run quite slow in future testing.

3.2.2 Goertzel Set Up

Using the Goertzel Algorithm in order to determine the tones has advantages and disadvantages. An advantage is the ability to search for specific tones and programmatically ignore all others including those that are very close which allows for a simpler program with a true or false style at a later point. However by having to search for each tone individually it leads to the program needing to run the data through multiple times and may cause a large processing time for large amount of tones. As in this case there are only 16 tones that are being searched for it was determined that the future simplification was more advantageous than other options.

Setting up the Goertzel includes designing the program to be able to test for all tones and determining the values that are used later in the running of the Goertzel algorithm. The design of the program was based off the object orientated style of the C# environment which enabled each tone to be set up as its own individual object that were each identical in layout. This layout can be seen in Figure 2.

```csharp
class Goertzel
{
    public int Sampling_Rate = 8000;
    public int N = 320;
    public int tone;
    public double coeff;
    public double Q1;
    public double Q2;
    public double Q0;
    public double sine;
    public double cosine;
    public double Real;
    public double Imagin;
    public double magnitude;
}
```

Figure 2: Goertzel Layout
Figure 2 shows all of the important coefficients and values that need to be determined in order to run the algorithm. The majority of these are determined by using the tone value, the sampling rate and the number of samples (N). Of these values the sampling rate is predetermined by the input data and the tones are known. This leaves the only value to be set as the number of samples. This was chosen arbitrarily based off a similar program found online (techn0mad 2013) with N set to 50. Unfortunately this value was not well designed for the project as can be seen in Table 2.

\[
k = \text{int}(0.5 + \frac{N \times \text{Tone}}{\text{Sampling Rate}})
\]

(1)

\[
cosine = \cos\left(\frac{\pi}{N} k\right)
\]

(2)

\[
coeff = 2 \times \cosine
\]

(3)

(Lyons 2012)

Equations 1 through 3 are used to determine the coefficients for each tone using the known values of Sampling Rate = 8000 and tone frequencies.

Table 2: Coefficients for N = 50

<table>
<thead>
<tr>
<th>Tone</th>
<th>K</th>
<th>Cos</th>
<th>Coeff</th>
</tr>
</thead>
<tbody>
<tr>
<td>930</td>
<td>6</td>
<td>0.7290</td>
<td>1.4579</td>
</tr>
<tr>
<td>991</td>
<td>6</td>
<td>0.7290</td>
<td>1.4579</td>
</tr>
<tr>
<td>1055</td>
<td>7</td>
<td>0.6374</td>
<td>1.2748</td>
</tr>
<tr>
<td>1124</td>
<td>7</td>
<td>0.6374</td>
<td>1.2748</td>
</tr>
<tr>
<td>1197</td>
<td>7</td>
<td>0.6374</td>
<td>1.2748</td>
</tr>
<tr>
<td>1275</td>
<td>8</td>
<td>0.5358</td>
<td>1.0717</td>
</tr>
<tr>
<td>1358</td>
<td>8</td>
<td>0.5358</td>
<td>1.0717</td>
</tr>
<tr>
<td>1446</td>
<td>9</td>
<td>0.4258</td>
<td>0.8516</td>
</tr>
<tr>
<td>1540</td>
<td>10</td>
<td>0.3090</td>
<td>0.6180</td>
</tr>
<tr>
<td>1640</td>
<td>10</td>
<td>0.3090</td>
<td>0.6180</td>
</tr>
<tr>
<td>1747</td>
<td>11</td>
<td>0.1874</td>
<td>0.3748</td>
</tr>
<tr>
<td>1860</td>
<td>12</td>
<td>0.0628</td>
<td>0.1256</td>
</tr>
<tr>
<td>1981</td>
<td>12</td>
<td>0.0628</td>
<td>0.1256</td>
</tr>
<tr>
<td>2110</td>
<td>13</td>
<td>-0.0628</td>
<td>-0.1256</td>
</tr>
<tr>
<td>2247</td>
<td>14</td>
<td>-0.1874</td>
<td>-0.3748</td>
</tr>
<tr>
<td>2400</td>
<td>15</td>
<td>-0.3090</td>
<td>-0.6180</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>N</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sampling Rate</td>
<td>8000</td>
</tr>
</tbody>
</table>

The highlighted sections of Table 2 shows where the problems were occurring with both red and green highlights indicating where there were identical coefficients. With such a low value for N and with tones that are reasonably close together the coefficient for a large amount of the tones ended up being identical. This causes the algorithm to not be able to distinguish between the different tones. In order to fix this problem the sample number was increased to 500 as can be seen in Table 3.
By increasing the number of samples the problems with the coefficients was fixed enabling the tones to be differentiated from each other and the Goertzel algorithm to run as it should. However by using this value the number of sample was larger than the number that had been designed to be read in from the data set up section. This caused further trouble in the program. In order to fix this the samples that were read in were compared with the length of the time period that the number of samples was equivalent to. It was determined that for the most practical running of the goertzel algorithm that the number of samples should be as large as possible whilst constraining the time period to a value of 40ms or less.

\[ N = \text{Sampling Rate} \times \text{Tone Period} \]  
\[ N = 8000 \times 0.040 = 320 \]

By using Equation 4 the number of samples was compromised to 320 which provided the coefficients that can be seen in Table 4. This provided values that were far enough apart to be able to distinguish every tone but keeps the tone length within the size of the maximum sample size.

**Table 3: Coefficients for \( N = 500 \)**

<table>
<thead>
<tr>
<th>Tone</th>
<th>k</th>
<th>( \cos )</th>
<th>Coeff</th>
</tr>
</thead>
<tbody>
<tr>
<td>930</td>
<td>58</td>
<td>0.745941</td>
<td>1.491882</td>
</tr>
<tr>
<td>991</td>
<td>62</td>
<td>0.711536</td>
<td>1.423071</td>
</tr>
<tr>
<td>1055</td>
<td>66</td>
<td>0.575333</td>
<td>1.350666</td>
</tr>
<tr>
<td>1124</td>
<td>70</td>
<td>0.637424</td>
<td>1.274848</td>
</tr>
<tr>
<td>1197</td>
<td>75</td>
<td>0.587785</td>
<td>1.175571</td>
</tr>
<tr>
<td>1275</td>
<td>80</td>
<td>0.535827</td>
<td>1.071654</td>
</tr>
<tr>
<td>1358</td>
<td>85</td>
<td>0.481754</td>
<td>0.963507</td>
</tr>
<tr>
<td>1446</td>
<td>90</td>
<td>0.425779</td>
<td>0.851559</td>
</tr>
<tr>
<td>1540</td>
<td>96</td>
<td>0.356412</td>
<td>0.712824</td>
</tr>
<tr>
<td>1640</td>
<td>103</td>
<td>0.272952</td>
<td>0.545904</td>
</tr>
<tr>
<td>1747</td>
<td>109</td>
<td>0.19971</td>
<td>0.39942</td>
</tr>
<tr>
<td>1860</td>
<td>116</td>
<td>0.112856</td>
<td>0.225713</td>
</tr>
<tr>
<td>1981</td>
<td>124</td>
<td>0.012566</td>
<td>0.025132</td>
</tr>
<tr>
<td>2110</td>
<td>132</td>
<td>-0.08785</td>
<td>-0.1757</td>
</tr>
<tr>
<td>2247</td>
<td>140</td>
<td>-0.18738</td>
<td>-0.37476</td>
</tr>
<tr>
<td>2400</td>
<td>150</td>
<td>-0.30902</td>
<td>-0.61803</td>
</tr>
</tbody>
</table>

\[ N = 500 \]

\[ \text{Sampling Rate} = 8000 \]
Table 4: Coefficients for $N = 320$

<table>
<thead>
<tr>
<th>Tone</th>
<th>k</th>
<th>Cos</th>
<th>Coeff</th>
</tr>
</thead>
<tbody>
<tr>
<td>930</td>
<td>37</td>
<td>0.7475</td>
<td>1.4950</td>
</tr>
<tr>
<td>991</td>
<td>40</td>
<td>0.7071</td>
<td>1.4142</td>
</tr>
<tr>
<td>1055</td>
<td>42</td>
<td>0.6788</td>
<td>1.3576</td>
</tr>
<tr>
<td>1124</td>
<td>45</td>
<td>0.6344</td>
<td>1.2688</td>
</tr>
<tr>
<td>1197</td>
<td>48</td>
<td>0.5878</td>
<td>1.1756</td>
</tr>
<tr>
<td>1275</td>
<td>51</td>
<td>0.5391</td>
<td>1.0783</td>
</tr>
<tr>
<td>1358</td>
<td>54</td>
<td>0.4886</td>
<td>0.9772</td>
</tr>
<tr>
<td>1446</td>
<td>58</td>
<td>0.4187</td>
<td>0.8373</td>
</tr>
<tr>
<td>1540</td>
<td>62</td>
<td>0.3461</td>
<td>0.6922</td>
</tr>
<tr>
<td>1640</td>
<td>66</td>
<td>0.2714</td>
<td>0.5429</td>
</tr>
<tr>
<td>1747</td>
<td>70</td>
<td>0.1951</td>
<td>0.3902</td>
</tr>
<tr>
<td>1860</td>
<td>74</td>
<td>0.1175</td>
<td>0.2351</td>
</tr>
<tr>
<td>1981</td>
<td>79</td>
<td>0.0196</td>
<td>0.0393</td>
</tr>
<tr>
<td>2110</td>
<td>84</td>
<td>-0.0785</td>
<td>-0.1569</td>
</tr>
<tr>
<td>2247</td>
<td>90</td>
<td>-0.1951</td>
<td>-0.3902</td>
</tr>
<tr>
<td>2400</td>
<td>96</td>
<td>-0.3090</td>
<td>-0.6180</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>N</th>
<th>Sampling Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>320</td>
<td>8000</td>
</tr>
</tbody>
</table>

After determining the most appropriate $N$ value the next step was to declare each object individually and set the tones in the variable seen in Figure 2. The tones were set in order of the SelCall value they represented and the objects placed into an array that enabled the tones to each be tested consecutively from zero through to $F$ (930-2400) by running the data section through the Goertzel Algorithm. This array was looped through and the output values for each object stored under magnitude. Once all the tones had been tested for a section of data they were combined into a list and the loop moves through to test the next section of data. Each list of magnitudes is then placed into a larger list until the end of the data is reached. This list is then used to determine the presence of each tone in each section of data. The position of the magnitudes in this list is used to determine which tone each magnitude is for. A second list of the same proportions was also created with the tones in the correct positions for use in the filtering and structure section.

3.3 Tone Identification Algorithm

The third section that needed to be designed was the Goertzel Algorithm itself. The Goertzel algorithm was designed based off a similar layout that was found online (Lyons 2012) and altered to fit with the different use of it in this project. This project has the Goertzel step through multiple sections of data and search for multiple tones. As the Goertzel algorithm is designed to search for only one tone it was important to make sure that the important values were reset at the start of each new run. This led to there being three main sections of the Goertzel code; the reset function, the initialisation function and the calculation section. Whilst quite small each of these sections are equally important.
Making sure that the Goertzel Algorithm section resets the correct sections is quite simple but must be done in such a way that it only resets the correct values. These values are the Q1 and the Q2 values. It was determined that the best way to set this up was to create a separate function that was called at the start of each time the Goertzel Algorithm was run through and ignored the rest of the time. The section sets these values to zero. This allows for the section to be reliable and quite simple.

The second section of the Goertzel Algorithm was to set up the Goertzel to search the data for the correct tone. This needs to be run 16 times for each set of data and must be reset between each time to the correct tone. The section can be seen in Figure 3. This shows what needs to be done for each individual tone in order to determine the coefficient. The coefficient is then used further when running the magnitudes through the Goertzel Algorithm to determine the existence of a tone.

```csharp
// Sets all the constants needed
public void InitializeGoertzel()
{
    int K = Convert.ToInt32(0.5 + (N * tone / Sampling_Rate));
    double w = (2 * Math.PI / N) * K;
    cosine = Math.Cos(w);
    sine = Math.Sin(w);
    coeff = 2 * cosine;
    ResetGoertzel();
}
```

**Figure 3: Coefficient Set Up**

The final section of the Goertzel Algorithm was the calculations section. This involved all of the areas that calculated and placed values in the Q1 and Q2 positions. This function had to run for the length of the data and calculate all values. Overall this had three sections as can be seen in Figure 4.

```csharp
// Run through the process code (Want to loop through N for this)
public void ProcessSample(int sample)
{
    Q0 = coeff * Q1 - Q2 + sample; // determines new value based off old values and current sample
    Q2 = Q1;
    Q1 = Q0;
}

// Gets real and imaginary parts of sample
public void GetRealImag()
{
    Real = Q1 - Q2 * cosine; // used the dotreal magnitude in a different way.
    Imag = Q2 * sine; // not really used but could be useful in future
}

// Gets the magnitude² of the Goertzel
public double getMagneSqured()
{
    double result;
    result = (Q0 * Q0 + Q1 * Q1 - Q1 * Q1 * coeff); // works out the final magnitude of the tone being tested
    return result;
}
```

**Figure 4: Overall Goertzel Algorithm**

In Figure 4 the first method shows the movement of all values that are used. This is the section that uses the value of the current sample as well as two previous runs of the same section. This is the section that is run through until all of the data has been used. The second method in this is not truly
used in the code but helps to test for the correct magnitude. This was very helpful in the original testing of the Goertzel. The final method in this section is used to calculate the magnitude. This uses the values that were calculated in the first method of this section.

3.4 Filtering and Structure

After the magnitude list has been created by passing the data sets through the Goertzel algorithm it then needs to be filtered to determine which magnitudes are of a reasonable value. The matching tones for these values then need to be determined and structured to be outputted as a set of ANI codes for each section that was found. These sections also need to be time stamped to correspond with the correct time that these were found in the code.

3.4.1 Filter

The first step in determining what tones were detected is to find out what magnitude corresponded with a tone. The Goertzel Algorithm produces a magnitude for a tone being present much higher than that of no tone however a magnitude is still produced in both cases. Originally a magnitude was picked based on the values found in the sample program. This proved to be ineffective as each data sound file that was passed through the Algorithm had different levels of magnitudes depending on the quality of the sound. It was determined that the best way to pick a cut off magnitude was to base the value off the magnitudes found in each individual sound file. The code segment found in Figure 5 finds the largest magnitude in each 40ms segment that was passed through the Goertzel Algorithm before finding the overall average value. This value was then increased slightly to account for the small section of the sound file that should have these values in them. However it was still left quite low to account for the large level of noise found in the signal.

```csharp
double max = 0;
double reasonable_max = 0;
List<double> ListOfMax = new List<double>();

foreach (var item in listofmagnitudes)
{
    max = item.Max();
    ListofMax.Add(max);
}
reasonable_max = ListOfMax.Take(ListofMax.Count).Sum() / (ListofMax.Count - 50);
```

Figure 5: Setting Base Magnitude

After determining the base level, all magnitudes were checked against this value to determine if they could be present in the sound. This was done in a couple of different sections. The first section looped through the list of magnitudes that were found and compared them to the base level (reasonable_max). If these values were greater they were stored in a corresponding list along with their corresponding tone. The values that were not valid were discarded at this point. In order to keep the positions of the tones detected a zero was used as a place keeper that marked the 40ms sections that had no tones detected. This section only worked as a basic filter to bring the values down to a reasonable number leaving 40ms sections with up to three magnitudes that were detected. This section can be seen in Figure 6.
The next section investigated the number of tones that were found in each section. If there was only one tone discovered it progressed through to the next stage and the magnitude of the tone was discarded in order to save space. However the sections that had more than one tone were investigated. This was completed in two steps. The first step involved determining if any of the tones that were detected in the section were present in the previous section that had already been run through. This was an important step as the 40ms sections overlapped and quite often would contain two different tones in them. However unless the tone was exactly half in each section one would have a higher magnitude. This was further addressed with the overlapping of the 40ms sections. The section of code that compares the tones presence can be seen in Figure 7.

```csharp
#region find_tones
foreach (var item in listofmagnitudes)
{
    List<int> sublisttones = new List<int>();
    List<double> sublistmag = new List<double>();
    int x = 0;
    foreach (var i in item)
    {
        if (i >= reasonable_max)
        {
            tonefound = Toneset[x];
            magfound = i;
            sublisttones.Add(tonefound);
            sublistmag.Add(magfound);
        }
        x++;
    }
    Onlytonesdetected.Add(sublisttones);
    Magtonesdetected.Add(sublistmag);
    if (!sublisttones.Any())
    {
        sublisttones.Add(0);
        sublistmag.Add(0);
        Onlytonesdetected.Add(sublisttones);
        Magtonesdetected.Add(sublistmag);
    }
}
#endregion
```

Figure 6: First Level Filter

```csharp
int M = 0;
foreach (var inte in data)
{
    if (L >= 1)
    {
        if (Onlytonesdetected[L][M] == Onlytonesdetected[L - 1][0])               // if element is in previous tone delete it
            Onlytonesdetected[L][M] = Onlytonesdetected[L - 1][0];
        if (Onlytonesdetected[L].Count == 1)                                // L = position of inte
            deleteas.Add(Onlytonesdetected[L][0]);
    }
    M++;
}
```

Figure 7: Second Level Filter
The section was then checked again to determine if there were still multiple values. If not the remaining value was stored for further use and the next section was checked. However if there were still more than one value present further filtering was needed.

The next step in the filtering was to determine if any of the tones found had a value that was substantially higher than the others that were detected. The code segment for this can be found in Figure 8.

```
if (Onlytonesdetected[L].Count > 1)
{
    N = 0;
    double maximum = data.Max();
    foreach (var inte in data)
    {
        if (inte >= maximum)
        {
            delete0s.Add(Onlytonesdetected[L][M]);
        }
        M++;
    }
}
```

Figure 8: Final level Filter

In order to determine if a tone was substantially higher the largest magnitude in the section was found and all tones were compared to a value two thirds of the highest value. If there was only one value this was determined to be the tone that was detected with the rest being most likely the next tone in the sequence. If however there were still multiple tones detected the tones were all removed. It was determined that due to the nature of the project (identification) it was better to have a missing tone then an incorrect tone. This 40ms section was then replaced with a zero to represent the missing tone. At the point the list of tones consists of a single tone list with the tones themselves instead of the magnitudes. The magnitudes are removed to save space on the controller.

3.4.2 Structure and Time Stamp
After filtering the tones down to only one tone per segment the next step is to take these tones out of the list and separate them into groups of tones that correspond with a correct ANI size. The first step of this section involved removing repeated values (which would only be found in sections that were stored after the first step of filtering). After removing these repeated values the next step involved checking for large groups of non-zero values in areas of the list. These values were considered as ANI codes if there were more than three values within the section.

These sections also needed to account for any tones that might be missing. In the case of missing tones there would be a gap of up to 60ms or two zero placeholders. If a group of tones were found with this size gap it was considered highly likely that the two sections were one ANI code with a missing value. The position of the missing value was recorded along with the rest of the tones. Once the entire section had been searched for ANI code groups the data was due to be time stamped.

3.5 Time Stamping
There were two options to time stamping the data which depended on the style of incoming data. The easiest way to timestamp would be with a continuous stream of data. This form of time stamping would be completed by using the internal clock to record the time of each packet of data arriving and then connected to any ANI code outputted from that data. Whilst not perfectly accurate
this would be accurate to a reasonable time period. If no ANI code was outputted from the data this time stamp could be discarded.

However if the data is arriving in packet form which may contain a full days’ worth of data this would not be accurate enough. The style of time stamping that has been implemented in this project at this point uses the 40ms data packet size to count the time period between the point that the data was found and the beginning of the data packet. This is done by dividing the number of preceding packets by two (to account for the overlap of 20ms) and rounding the result up. This value is then multiplied by 40ms to output a value to represent seconds past since the start of the transmission.

This was completed this way for testing purposes so as to be able to be compared easier to the sound on Audacity sound editing software.

A section of this that was not completed was the final form that would need to be implemented in implementation. If the data is received as large packets a combination of the two time stamping techniques would need to be used. The plan for this would involve using the internal clock to take a preliminary time stamp of the time the data arrives and storing it. The second style of time stamping would then need to be implemented to count from the end of the data period instead of the start as was done for the testing situation. This value would then be deducted from the original time stamp in order to determine the exact time of the arrival of data. This has the advantage of being able to work for both streaming data and packet data. It would however be an unnecessary level of complexity if all data is streamed.

3.6 Output to Server

The final section of the program for the controller involved outputting the data. The only definition that was require in output of data is that it arrived in an easily accessible format. As such the data was formatted to be sent as Comma Separated Values (CSV) which is one of the simplest styles of output.

Past this point was not investigated in this period of the internship due to outside factors. One of the considerations that will need to be accounted for include the output style, as the BVR5000 has multiple options as can be seen in Appendix B. Also important is to include some form of database implementation on the client’s server which cannot be completed until just prior to the project going live or may need to be performed by the clients IT department.
4. RESULTS

Whilst this project was almost completed there were some setbacks that led to some areas not being completed within the specified time. One of the major causes of this was the BVR5000 not being completed on time. This delay led to the mostly completed program not being able to be tested in a practical sense. As such this section will include only the simulation results that were completed and some comments regarding the differences that may be expected from the completed product once it is uploaded into the BVR5000.

4.1 Reading from Input

The section that reads the incoming data is one of the least complete sections due to the lack of information provided regarding the format of the data. As the incoming data will arrive as sound data it is possible to make some assumptions regarding the data. The sound data is most commonly sent through as integer values that represent magnitudes sampled of an incoming soundwave. This set of data is used for the program as is and as such does not need much manipulation. However it is important to check that the incoming stream is an accurate representation of the sound wave being sent through. Figure 9 displays an example of sound that was sent through the program.

![Figure 9: Example Sound Received](image)

This is an example of one radio transmission that was sent through by the client. This transmission was opened into Audacity software in order to check the transmission and manipulate the data to be used to check for inconsistencies in the code. This data was then run through the section of the program designed to read in data and the magnitudes recorded in Microsoft Excel. These magnitudes were then graphed as can be seen in Figure 10. After determining the amount of magnitudes this section had the frequency calculated and compared to the correct section as can be seen in Figure 11.

![Figure 10: Test Graph of Sound Section](image)
When comparing Figure 10 and Figure 11 it is possible to see that the shapes of the two graphs are quite similar. It was determined after checking multiple sets of data this way that the method was sound and tested to a reasonable level to move on to the next section.

One of the major problems that may be faced with this in the implementation stage of the project is the lack of an internal soundcard on the BVR5000 that may prevent the reading in of sound data directly. The BVR5000 does have the ability to work with an external soundcard which will help with this problem however it may need further work in order to connect with the external soundcard.

4.2 Data Formatting
There was not much testing completed on the data formatting section of the project as it consisted of data manipulation and grouping as opposed to the sections that changed the data. There was however a quick test conducted to determine that the overlap of the tones was consistent. This test was completed using excel to graph three concurrent data packets and can be seen in Figure 12. Figure 12 shows three consecutive data packs of a known wave. If these values were consecutive Data Pack 2 would start off from the same point Data Pack 1 ends. However if the offset is accounted for it can be seen that Data Pack 2 starts from a point approximately 160 samples in which corresponds with the 20ms delay. This was considered adequate testing for this section of the project.
4.3 Tone Identification

The next section that needed to be tested and analysed was the tone identification algorithm. This was done by running the known tones through the program. This gave an output as can be seen in Figure 13.

These magnitudes that were outputted were then compared to the known values that were used. These were found to be a match. A secondary testing was conducted by using excel for a manual calculation. The values calculated using excel also showed a match to those outputted from the Goertzel Algorithm. These tests were run multiple times with different input values until it was determined that it was unlikely for a mistake to be made.

4.4 Filtering and Structure

These values found in Figure 13 were then run through a filter. Again this section was tested using a known tone set input with a tone period of first 100ms and then 40ms. Running these enabled the author to determine that all three sections of the filtering process were working correctly. The first filter provided an output as can be seen in Figure 14. This filter provided a base level and as such was not expected to output precise values. As Figure 14 shows this was the case. It however did provide an image of what could happen with overlapping tones. After multiple tests of a similar kind this filter was determined as being functional and the second filter was checked.
The second filter was difficult to check using the larger tone period of 100ms as these segments would have a large amount of undetected repeats in them. This is due to the multiple single values of the same tone passing the first stage of filtering as can be seen in Figure 15.

As such this section was tested using tones with a period of 40ms. The output from one of these tests can be seen in in Figure 16. This has the same input values as Figure 15 with the only difference being the tone period. As can be seen the output removed any form of repeated tone. This was done multiple times before moving on to test the third filtering step.
The final filter involved comparing the magnitudes in one section. These are what is left after filtering figure 14. This filter was not tested as thoroughly as the other filters. It was however found that most of the sections that made it through to this final filter did not pass this stage. As such this section may need some further investigation or may be removed from the code if it is found to be redundant in further testing.

As the structuring did not involve removing of important data such as was completed in the filtering section, testing of this section was not found to be quite as important. It was visually inspected when stepping through data to determine if any errors in structuring could be found. No such errors were detected and as such this section was determined to be working to a reasonable level. If more time was permitted this could be looked over more thoroughly in the future.

4.5 Overall Results
The overall output of this project proved to be mostly a success if an incomplete one. The current incarnation of the project runs in a simulated environment reading in a WAV file input of radio transmissions and outputting the detected ANI codes and the time of transmission that they were detected. This was run through multiple tests to determine if the program was working as it should be. The first of these tests was an example with no external noise and a known ANI code. This can be seen in Figure 17 and the output in Figure 18.
Figure 18: Output for ANI Code 12345

The output was seen as successful and this test was run multiple times with different ANI codes being checked, all of which were determined perfectly. The second test involved altering the codes slightly to include noise, vary the strength of the tones or to preclude a tone at some point in the transmission.

The program was unable to pick up the tone that was distorted. However the program was able to determine that there was meant to be a tone there and included a space to show a placeholder for where a tone was missing. This was determined to be successful and after the completion of multiple tests it was determined that the program was ready to be tested against the noisy incoming radio sound with unknown ANI codes. This can be seen in Figure 19.

Figure 19: Final Output
Whist it is impossible to determine whether the ANI tones are correct based on former knowledge the number of tones, the number set and the timing of the tones seen in Figure 19 seems compatible with the sound data that was being placed through the program. The corresponding sound section can be seen in figure 20. This was tested with multiple sets of tones and continued to provide output data that was similar to what was expected. As such this was seen as a successful simulation.

Whilst the program is working well in simulation there is still a need to test the program in implementation that has not yet been completed and may need to be further investigated when the BVR5000 has reached a reasonable level of completion.
6. CONCLUSIONS

The Selective Calling Decoder developed over the course of this project was completed in a fully simulated environment and was tested as such. This project has managed to produce a program that can read in a WAV audio file stored on the same device as the program and output ANI codes and timestamps to the screen of this device. This program has yet to be uploaded to the BVR5000 in order for implementation, has not been altered to allow input from an external source or in any format of than a WAV audio format and is unable to output this data to an external server provided by the client.

Some areas that have caused setbacks in the completion of this project include the delay on arrival of the BVR5000, an extended period of time used to test the hardware components of the BVR5000 once the shipment did arrive, a lack of understanding of the language C# early on in the project and changes made to the kernel program of the BVR5000 during the period of this internship. This project will most likely be completed once the BVR5000 has been fully tested.

Whilst not fully implemented this project has been considered successful as in a simulated environment it is able to output the correct data. This shows that the outcome is possible and can be delivered with further work.

6. FUTURE WORKS

Whilst this project has been completed to an adequate level for the time period involved there is still a reasonable amount of work that is left to be done before the project is complete. Some of these were steps that were changed during the process of the project, some were not completed due to a delay on receiving correct information and some were purely due to the time constraints of the project.

The work that still needs to be completed in order to have the project running as specified is to port the program to the new language of the controller, upload the program to the controller and run testing to determine that the program runs as well in reality as in simulation situations. Some other work that should be done if possible is to improve upon the current input and output data so that it is more flexible and able to handle different types of data input. Whilst these final two steps are not necessary they will be quite helpful in setting the BVR5000 SelCall decoder apart from those that are already on the market and help showcase the abilities of this controller.

The process of porting the program to the correct language was not completed in this project mostly due to time constraints and the late time of the change of base language. In order to port this program to a different language the process should be reasonably simple but time consuming. Some problems that may occur with this are the possible need to rewrite large sections of the existing code, reformatting of sections to make them friendlier to the new system and being able to understand enough of both languages to know what the different advantages and disadvantages of both would have on the program design (Mooney 1997). Things that need to be considered when completing this are the compatibility between the two languages, the different functions that are available, the differences and similarity between the same types of functions and the overall layout of the different systems.

The program was not able to be uploaded to the controller in this period of the project due to the BVR5000 still being in testing phase and the kernel not being completed. As well as this the program had not yet been ported to the correct language. These would all need to be completed before the program can be uploaded to the controller and the project taken through to completion.
process of uploading the program should also be quite simple but some things that will need to be considered are the difference in computing power between the simulation and controller, the need to use a plug in sound card opposed to the inbuilt one used in the simulation and the different way of communicating between the input and outputs. When the program is uploaded onto the hardware there are always some discrepancies between the two platforms. This difference between the simulation and implementation platform can have a large effect on the outcome of the project and should be understood as early as possible.

After uploading the program to the controller it is important to test the decoder under all conditions to determine if there are any faults. Some of these faults could be due to the differences between the simulation environment and the controller such as those mentioned above and may need some reprogramming to fix but no major changes. Other faults could be due to the different computing powers of the two and could cause large time delays that may need a large level of altering and slimming down of the program to be able to properly fix it. On top of these predictable faults there could be some faults that are caused by a general lack of compatibility between the program and controller which could not be found in simulation. Overall there are many more problems that can be found in implementation that may not be able to be seen during the simulation of a program and it is always important to test the implementation separately and thoroughly if it is possible.

Once the existing program has been successfully uploaded and possibly edited to be able to work on the BVR5000 it will need to run off a different input than has been previously assumed. Throughout the simulation stage of this project the input has always been assumed to come through a WAV file as presented by the client and the program has been run separately for each incoming WAV file. Whilst this is useable in simulation, for the implementation the program will need to run when a new input is detected which may or may not be in the WAV format and may come in a continuous stream of data as opposed to a package form. In order to account for this in the simulation layout there is a separate function for reading of data which can be removed as long as the final output of the data is in an array of values similar to those found in the output of a WAV file. This section has been left like this in order to provide flexibility to any future programming that may need to be completed to account for all input styles that could be used.

The final step that would be needed to be completed before the project can go live is to be able to improve the output to meet the final specifications of the brief and the ability to read it into the server. During this time period the output of the program was designed to be simple and easy to view on the screen to determine if there were any errors in it and as such was placed in a CSV format. Whilst useful for testing purposes this is not what was outlined in the brief for this project and as such in order to complete the project more work needs to be done. For this to be accomplished the client needs to be contacted for more specific information on the needs as the brief does not provide reliable information on what is needed.

A final optional piece of work that can be completed though was not mentioned in the brief is to create a website that links directly to the data that has been recorded on the server. This would enable the client to have easier access to the data and could provide the ability to search through the data that is available by date, time or ANI code number. This also allows the ability to match the ANI code number to employee numbers that are included in the system. By including a website the project will provide a complete system with a SelCall decoder, data logging capabilities and the ability to access and analyse the data at any time. Overall this will provide the client with the data they need and help display some of the capabilities of the BVR5000 as a Selective Calling Decoder.
REFERENCES


You mentioned you were looking for other projects for possible student development. I can suggest one that we would be interested in.....

**Sellipse Decoder/recorder**: Here at RTIO we use an analogue radio system and utilise Sellipse encoding as an Auto Number Identification for each two-radio in our fleet. When incidents occur two-way radio traffic recordings often play a crucial role in the investigation process. In these circumstances the ANI of a two-way can be paramount to the success of the investigation. We are often asked to capture the ANI of these voice recordings. This can be done but is not easy to do with our current methods. What would be needed to simplify this process would be a device that could be installed on our system to constantly decode ANI’s from the two-way radio channel audio and store these decoded ANI’s along with a time stamp on a server somewhere for collection when needed or for live viewing as they occur on an operators console.

The Rolls Royce device could have the following features...
- ✓ Use RJ45 connectors
- ✓ Operate on 600ohm balanced audio or unbalanced audio
- ✓ Pass all pins through so that the device can be inserted in to an existing system
- ✓ Have Receive audio lines selectable to allow flexibility
- ✓ Combine both RX and TX audio lines to record both radio traffic directions
- ✓ Pass decoded ANI and timestamp out in a easily accessible formats for logging on a server
- ✓ Operate as single channel units or be cascadable to form a multichannel device
- ✓ Decode numerous variable Sellipse formats (Tone Groups, Tone Durations, Number of tones, etc.)
- ✓ Contain a phone book/look up table to match aliases to ANI decodes
Appendix B

BVR5000 SPECIFICATIONS

- Utilise your TETRA network’s exceptional coverage to control and monitor your plant and equipment
- Remotely diagnose and measure field equipment status
- Access maintenance data for heavy machinery (eg SMU usage hours, brake usage, etc)
- Collect operational statistics from field equipment over your radio network
- Remote building and equipment access control
- Control remotely located infrastructure over your radio network

<table>
<thead>
<tr>
<th>General Specifications</th>
<th>Isolated</th>
<th>LED</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating Temperature</td>
<td>-40°C to +85°C</td>
<td>✓</td>
</tr>
<tr>
<td>Storage Temperature</td>
<td>-55°C to +65°C</td>
<td>✓</td>
</tr>
<tr>
<td>Operating Humidity</td>
<td>5% to 95% RH @ 50°C</td>
<td>✓</td>
</tr>
<tr>
<td>Operating Altitude</td>
<td>-500 metres to 4500 metres</td>
<td>✓</td>
</tr>
<tr>
<td>Dimensions</td>
<td>180 x 150 x 30mm</td>
<td>✓</td>
</tr>
<tr>
<td>Weight</td>
<td>500 grams</td>
<td>✓</td>
</tr>
<tr>
<td>Construction</td>
<td>Diecast Aluminum</td>
<td>✓</td>
</tr>
<tr>
<td>Power Supply</td>
<td>Wide Range 10 - 30VDC</td>
<td>✓</td>
</tr>
<tr>
<td>Power Consumption</td>
<td>≤ 1W</td>
<td>✓</td>
</tr>
<tr>
<td>Certifications**</td>
<td>C-RAD, CE, FCC Class A</td>
<td>✓</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CPU</th>
<th>Isolated</th>
<th>LED</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main Processor</td>
<td>Ti Sitara CPU (ARM Cortex A8)</td>
<td>✓</td>
</tr>
<tr>
<td>Clock</td>
<td>300MHz (up to 600MHz)</td>
<td>✓</td>
</tr>
<tr>
<td>OS</td>
<td>Linux</td>
<td>✓</td>
</tr>
<tr>
<td>Co-processor</td>
<td>STM32 CPU (ARM Cortex M4)</td>
<td>✓</td>
</tr>
<tr>
<td>Clock</td>
<td>72MHz</td>
<td>✓</td>
</tr>
<tr>
<td>Memory: Flash</td>
<td>256MB</td>
<td>✓</td>
</tr>
<tr>
<td>Memory: RAM</td>
<td>256MB</td>
<td>✓</td>
</tr>
<tr>
<td>External Memory: SD Card</td>
<td>Up to 32GB</td>
<td>✓</td>
</tr>
<tr>
<td>Real Time Clock</td>
<td>Supported</td>
<td>✓</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Ports</th>
<th>Isolated</th>
<th>LED</th>
</tr>
</thead>
<tbody>
<tr>
<td>RS-232</td>
<td>2</td>
<td>1 with Modern Flow Control</td>
</tr>
<tr>
<td>RS-485</td>
<td>1</td>
<td>Dedicated Full Duplex</td>
</tr>
<tr>
<td>USB Host</td>
<td>4</td>
<td>Type A (supports cameras, audio, etc)</td>
</tr>
<tr>
<td>USB Device</td>
<td>1</td>
<td>Mini-B Socket</td>
</tr>
<tr>
<td>Ethernet</td>
<td>2</td>
<td>10M/100M/1Gbps</td>
</tr>
<tr>
<td>CAN Bus</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Inputs/Outputs*</th>
<th>Isolated</th>
<th>LED</th>
</tr>
</thead>
<tbody>
<tr>
<td>Differential Analogue Inputs</td>
<td>4</td>
<td>16 bit resolution: 0V to 5V range ± 120 VDC common-mode range</td>
</tr>
<tr>
<td>Relay Switches</td>
<td>4</td>
<td>Independent COM/NO/NC Contacts</td>
</tr>
<tr>
<td>Opto-coupled Digital Inputs</td>
<td>4</td>
<td>Logic 0 / 4.5 - 50VDC, Logic 1</td>
</tr>
<tr>
<td>Opto-coupled Digital Outputs</td>
<td>4</td>
<td>Open Drain 50VDC @ 200mA Max</td>
</tr>
<tr>
<td>Video</td>
<td>✓</td>
<td>HDMI Output</td>
</tr>
<tr>
<td>Touchscreen</td>
<td>✓</td>
<td>via USB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Communication Protocols</th>
<th>RS-232</th>
<th>RS-485</th>
<th>Ethernet</th>
<th>USB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modbus RTU</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Modbus TCP</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>DNP3 #</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>IEC61850 #</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

** Protocols under development.

---

Beyond Voice Radio
represented in Australia by Radlink Communications

1300 73 65 87 | info@radlink.com.au
WA 6 Power Circle, Malaga, WA 6090, Australia
QLD Unit 22, 388 Newman Rd, Greenvale, QLD 4034, Australia

BEYOND VOICE RADIO

32 | Page