Project Number: ECE-WRM-WOR

Worcester Emergency Band Data Acquisition Project

A Major Qualifying Project Report

submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the

Degree of Bachelor of Science

by

John E. DeMello

Date: 26 April 2006

Approved:

Professor William R. Michalson, Major Advisor

Table of Contents

Table of Figures

Abstract

Interoperability and overcrowding of the emergency band channels have led to mixed communication in crisis situations. Despite FCC regulations and policies, the problem has persisted and is being addressed more critically since September 11th and Hurricane Katrina. The goal of this MQP is to begin work on a data collection and processing tool that will allow for emergency bands to be monitored and channel bandwidth usage to be calculated. Data gathered and the code used can also be used in conjuction with further WPI projects in this area.

Introduction

With the events of September 11th and Hurricane Katrina, multiple communication and interoperability problems slowed operations and hampered rescue efforts. This resulted in lives and money lost from a preventable situation. These interoperability issues were brought to light through previous crises, however not enough was done to prevent such events from occurring again. The interoperability and channel crowding dilemmas were addressed by the FCC by increasing the amount of bandwidth for emergency operations as well as introducing regional planning and more regulations.

Since September 11th and Hurricane Katrina, more bandwidth has been freed to reduce interference from surrounding bands and the FCC has been working to increase interoperability between jurisdictions and departments. The root of the issue, however, has not been addressed. The current FCC regulation assigns a new frequency pair only when the channel has been loaded to 70 mobile stations for conventional systems and 100 mobile stations for trunked systems. The purpose of this project is to determine if the channels are overloaded by examining the bandwidth usage of emergency channels within the Worcester Area. If overcrowding is already occurring at non-crisis times, the FCC regulations should be modified; however the opposite may also occur confirming that FCC regulations are still appropriate.

Background

Emergency Band History and Problems

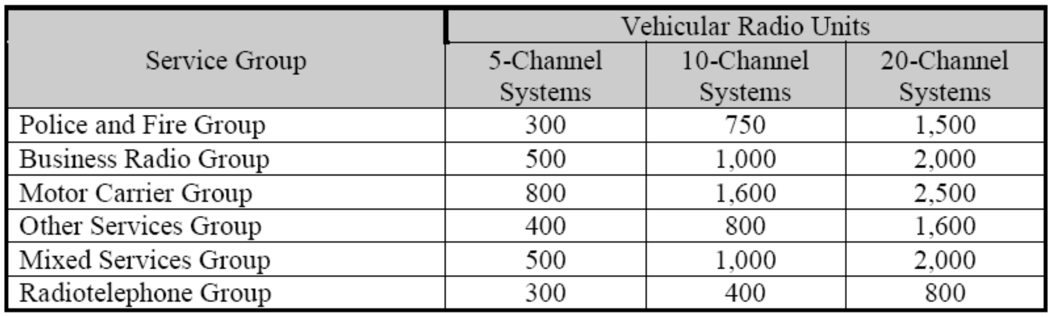
In the 1980's the Federal Communications Commission (FCC) released around 230 channels under the National Public Safety Planning Advisory Committee (NPSAC) in the 800 MHz band for use by public safety officials. This was in addition to the estimated 70 channels already established for this purpose in the 800 MHz band. The FCC remained very flexible when regulating these channels as each public entity had different needs associated with their locality. With such minute standards established, multiple systems were put into place from a variety of vendors that resulted in incompatible communications equipment between each municipality. Since such a variety of equipment was used, departments had multiple issues with interoperability such as when a high-speed chase crossed over town or state lines or if a large fire warranted several fire departments to respond.

The NPSPAC first addressed the issue of interoperability when they were assigned the 800 MHz channels, more specifically 821-824/866-869 MHz. During the Washington, D.C. Air Florida crash and D.C. Metrorail crisis, which both occurred during adverse weather conditions, multiple municipalities responded to the crisis and interoperability issues prevented more efficient handling of these scenarios. The National Plan was developed from issues such as overcrowding, interoperability and bandwidth issues from such diverse communications equipment.

The National Plan was the first time the FCC required states and localities to come together and plan the frequency assignments in their region so as to reduce overcrowding and bandwidth issues. Many municipalities implemented trunking systems to free up bandwidth, however many did not need such technology since their non-trunked systems were more than adequate. With some areas using trunking systems and with multiple trunking systems available while others did not use trunking, interoperability still remained an issue.

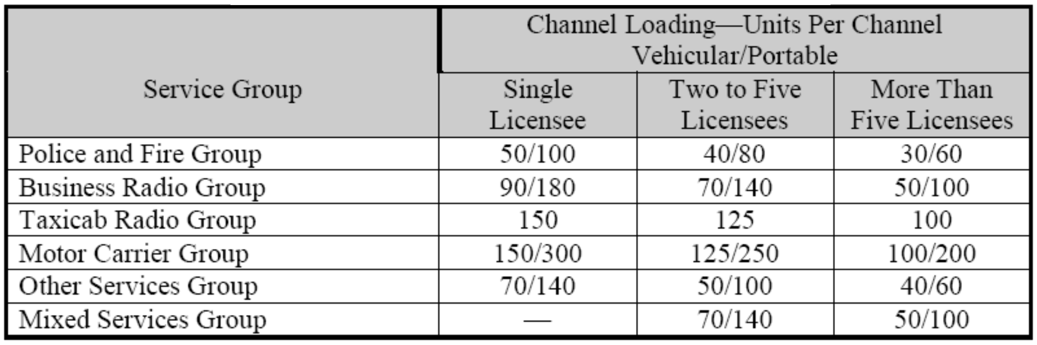
To prevent overcrowding on channels and bandwidth issues, the FCC established the standards listed in Figures 1 and 2 shown below.

> Trunked systems were authorized on the basis of the loading criteria presented in [Figure 1]. The *Second General Service R&O* further stated that any licensee using a trunked system occupied at 90 percent would be permitted to apply for additional channels. (Booz·Allen, 32)



Service Group	Vehicular Radio Units		
	5-Channel Systems	10-Channel Systems	20-Channel Systems
Police and Fire Group	300	750	1,500
Business Radio Group	500	1,000	2,000
Motor Carrier Group	800	1,600	2,500
Other Services Group	400	800	1,600
Mixed Services Group	500	1,000	2,000
Radiotelephone Group	300	400	800

Figure 1 - Loading Requirements for Trunked Systems



Service Group	Channel Loading—Units Per Channel Vehicular/Portable		
	Single Licensee	Two to Five Licensees	More Than Five Licensees
Police and Fire Group	50/100	40/80	30/60
Business Radio Group	90/180	70/140	50/100
Taxicab Radio Group	150	125	100
Motor Carrier Group	150/300	125/250	100/200
Other Services Group	70/140	50/100	40/60
Mixed Services Group	—	70/140	50/100

Figure 2 - Loading Requirements for Conventional Systems

Multiple municipalities use trunked technology to utilize multiple channels within a system simultaneously. In a trunked system, a data channel is used to send frequency information to each radio which designates which frequency to talk on. Unlike conventional systems which require each user to be on their own frequency, each radio will hop from different frequencies depending on which are already occupied thereby freeing up much more bandwidth. As much of the older, conventional systems begin wearing out, newer trunked systems are being instituted according to FCC regulation's section 90.623 which only allows a maximum of five frequency pairs to be assigned in a given area for conventional use. (Code, 2005)

These conventional systems are also regulated by Section 90.627 which forces each channel to be loaded to 70 mobile stations before any additional channels are assigned. For trunked systems this number is increased to 100 mobile stations per frequency pair with a maximum number of 20 pairs assigned at any one time. (Code, 2005)

Through more recent events such as September 11th, Hurricane Katrina, the Oklahoma City bombing and TWA Flight 800 crash, the FCC has determined that previous attempts to create more interoperability and lessen interference has proved less than optimal. During these disasters, interference with commercial wireless carriers and channel overcrowding and non-interoperability occurred highlighting the need for further action to be taken by the FCC and regional planners. At the FCC level, interference issues were addressed by reconfiguring the 800 MHz band and removing commercial wireless carriers such as Nextel out of the band. The two band configurations, before and after, can be seen in Appendix A.

This band reconfiguration did not address the issue of overcrowding, however, and the possibility that FCC regulations regarding the assigning of channels may be outdated. With population growths, the need for more first-responders, and rises in emergency band activity, 100 mobile stations per trunked pair and 70 mobile stations per channel for conventional systems may prove to border on overcrowding as it is.

The objective of this project is to collect data on emergency band transmissions within the City of Worcester and statistically analyze the bandwidth usage at different time intervals. The method must be replicable in order to apply to other geographical locations using different frequencies and also to allow for further data collection in Worcester.

Bandwidth Monitoring Equipment

Multiple methods can be used to log the data of radio bandwidth usage by the Worcester Emergency Services. To begin this process the signal must be obtained in order to monitor the channels being used. There are multiple paths that can be used to gather these signals and their corresponding channels used.

The Worcester Emergency Services are required by law to record their radio communications. These tapes, though not easily attainable, may be used and would be a low cost alternative to the next method. The Worcester Emergency Services tapes would provide sound files which could be downloaded to a computer and then analyzed.

The next method would be to purchase scanners to monitor the bands and record the output to a computer. The Worcester Emergency Frequency Band is split up as pictured in Appendix B.

The multiplicity of frequencies would be a deterrent to using this method of scanners. Scanners can only monitor one frequency continuously at a time if accurate data is to be collected. Scanners can *scan* different frequencies for activity, but if two frequencies are being transmitted over at the same time there will be no way to monitor both. Being limited to one scanner per frequency monitored, this will not give us an accurate depiction of the amount of bandwidth used at a given time.

Another method of logging frequency use data would be through the use of a spectrum analyzer. Spectrum analyzers take snapshots of a time-domain signal and perform a Fourier Transform which allows one to see the signal in the frequency domain. Depending on the spectrum analyzer, large bandwidths of frequencies can be monitored at once and their corresponding "usage" will show up as a peak.

With the onslaught of streaming radio stations, emergency radio enthusiasts have posted live audio streams of the emergency bands in a given area. These, however, under further investigation are unreliable since streaming requires a buffer and the connection may be lost at any given time.

The easiest method available for us to monitor the bands is to use multiple scanners. Below are some prices for the cheapest trunking scanners capable of monitoring the emergency bands in Worcester.

PRO-2051 1000-Channel Triple-Trunking Desktop Scanner	$149.97
PRO-2052 1000-Channel Dual-Trunking Desktop Scanner	$149.99
Uniden BCT-8 800 Mhz Mobile Scanner	$179.99

Figure 3 - Scanner Costs

In addition to scanners, a sound card is needed that is able to handle multiple sound inputs to a computer that will enable the recording of the signal directly to the computer. Two different sound cards were found which allowed multiple inputs. The first is a 10 input/output sound card that can be directly installed into a Windows or Mac capable computer. This sound card would enable multiple RCA channels of sound input to the computer that could be recorded and stored on the hard drive for further analysis. The Delta1010LT PCI Virtual Studio card is shown below.

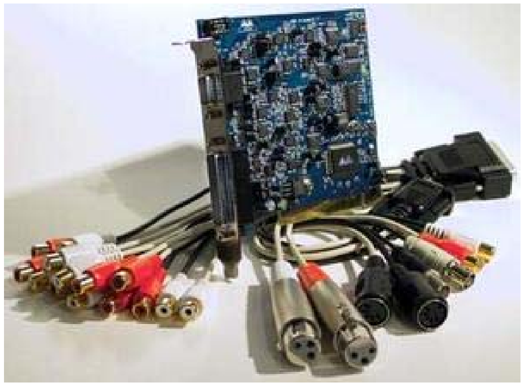


Figure 4 - PCI 10 Input/Output Sound Card (American)

Another alternative would be to purchase multiple USB sound inputs for the computer. This can tailor the need for multiple scanners better since an order would only have to be placed for the amount needed. The USB sound card is shown below.

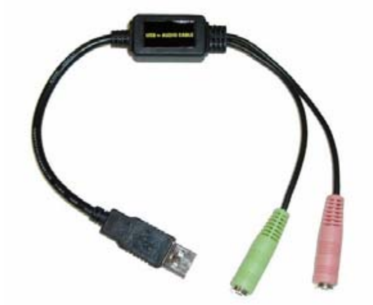
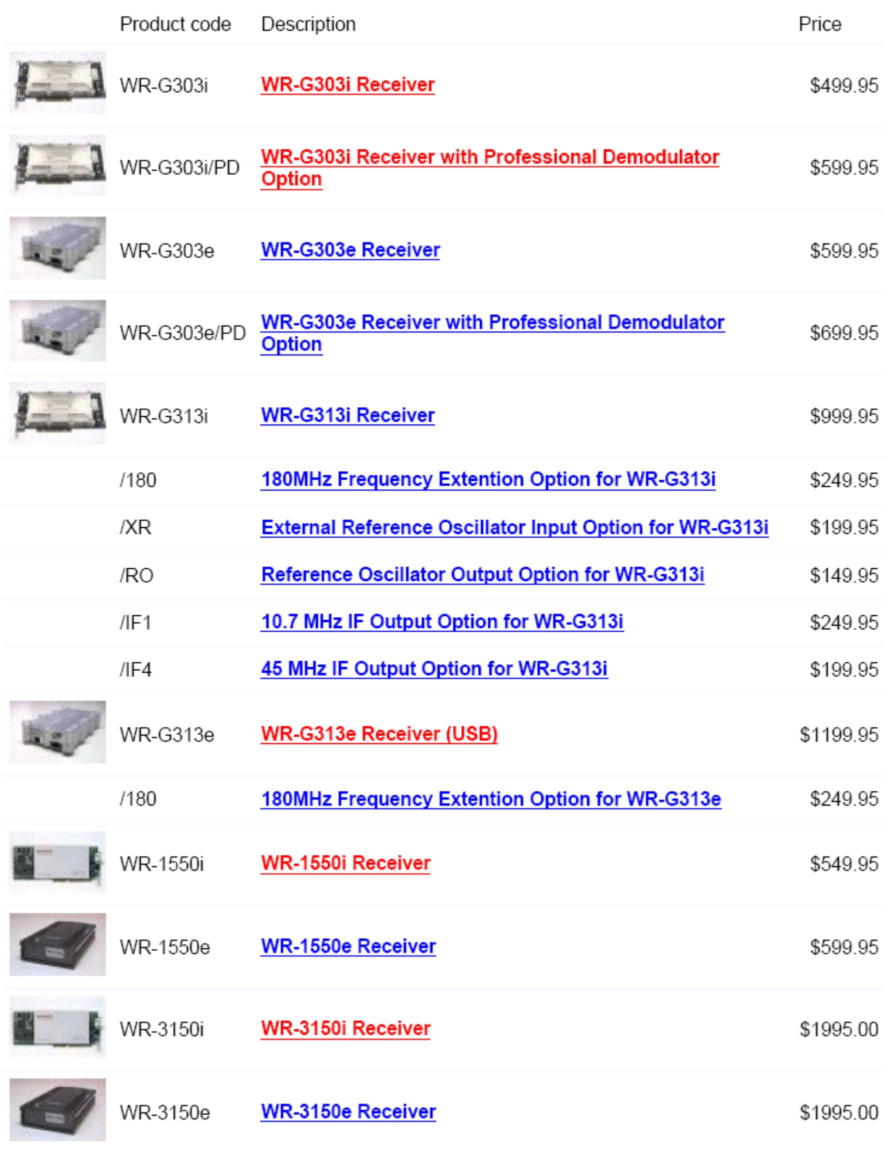


Figure 5 - USB Input/Output Sound Card (USB Gear)

A simple all-in-one alternative to purchasing a scanner and audio card necessary is provided by WinRadio but is much more expensive. The price chart for their products can be seen on the following page. They provide both internal and external receivers which are fully adjustable through the computer and can be logged through the software provided by WinRadio.

Even with logging by WinRadio or the recording of the scanner's output, there may be inherent static in the signal. Using Matlab's Data Acquisition software, these signals can be imported into Matlab and then filtered and analyzed. Matlab provides powerful tools to measure when and for how long a period of time a signal is present. More research must be done in this area to fully understand the capabilities of Matlab, but it remains the best and simplest tool for analyzing the signals and providing data.



Product code	Description	Price
WR-G303i	WR-G303i Receiver	$499.95
WR-G303i/PD	WR-G303i Receiver with Professional Demodulator Option	$599.95
WR-G303e	WR-G303e Receiver	$599.95
WR-G303e/PD	WR-G303e Receiver with Professional Demodulator Option	$699.95
WR-G313i	WR-G313i Receiver	$999.95
/180	180MHz Frequency Extention Option for WR-G313i	$249.95
/XR	External Reference Oscillator Input Option for WR-G313i	$199.95
/RO	Reference Oscillator Output Option for WR-G313i	$149.95
/IF1	10.7 MHz IF Output Option for WR-G313i	$249.95
/IF4	45 MHz IF Output Option for WR-G313i	$199.95
WR-G313e	WR-G313e Receiver (USB)	$1199.95
/180	180MHz Frequency Extention Option for WR-G313e	$249.95
WR-1550i	WR-1550i Receiver	$549.95
WR-1550e	WR-1550e Receiver	$599.95
WR-3150i	WR-3150i Receiver	$1995.00
WR-3150e	WR-3150e Receiver	$1995.00

Figure 6 - WinRadio Products (WinRadio)

Data Collection Methods

After examining the scanner and sound card methods, further investigation was conducted into the viability of using a spectrum analyzer. The following outlines a process by which data collection, acquisition and analysis can be conducted through both the sound card and spectrum analyzer and assesses the best method. It follows two major paths that can be taken and are outlined in the flowchart below.

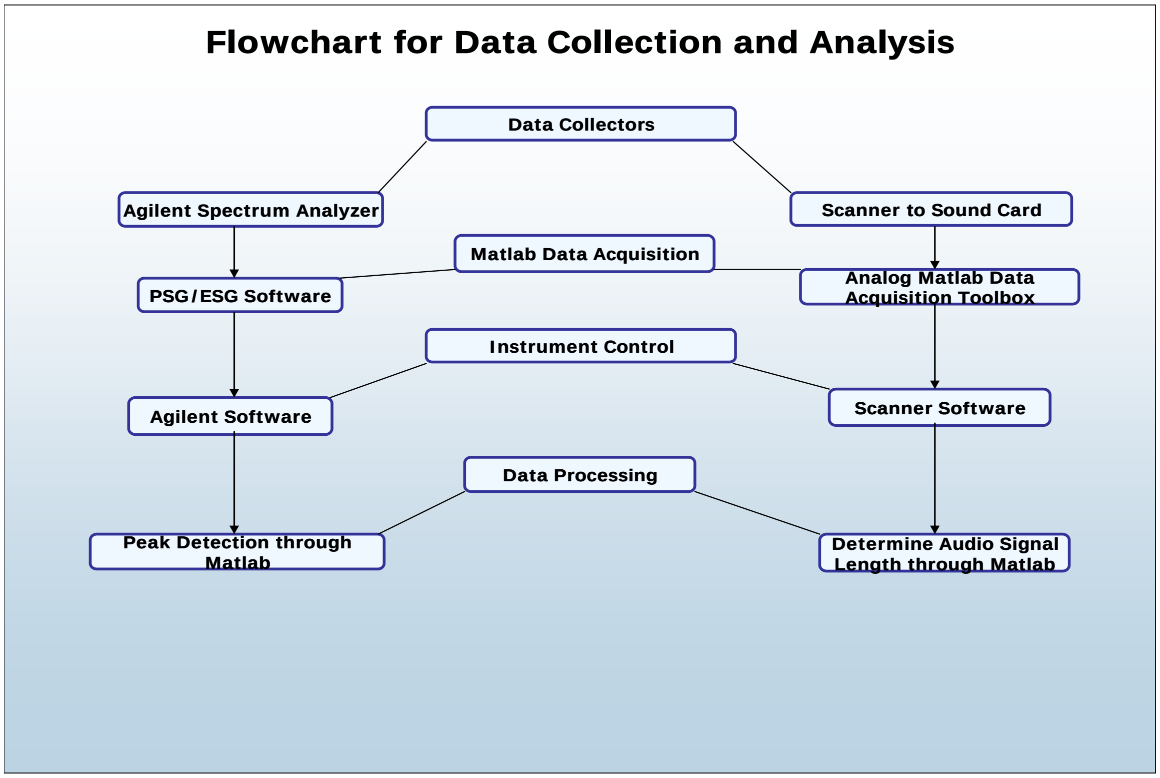

Flowchart for Data Collection and Analysis
Data Collectors
Agilent Spectrum Analyzer
Scanner to Sound Card
Matlab Data Acquisition
PSG/ESG Software
Analog Matlab Data Acquisition Toolbox
Instrument Control
Agilent Software
Scanner Software
Data Processing
Peak Detection through Matlab
Determine Audio Signal Length through Matlab

Figure 7 - Project Flowchart

From the flowchart, two distinct paths are shown within the four sections. The data collectors that were investigated in depth are the Agilent E4445A PSA Series Spectrum Analyzer and a simple trunking scanner to sound card set-up.

The Spectrum Analyzer presents many features which would make this project easier. With a very simple interface, the spectrum analyzer can quickly be set up to monitor the necessary bandwidth by setting both the start and end frequencies. The spectrum analyzer performs the necessary calculations to scan the frequencies in between and perform an FFT which is then displayed on the screen. Using the signal generator, a single signal was able to be seen on the spectrum analyzer. This signal could be seen as a peak at a given amplitude on the Y-axis with a certain frequency on the X-axis. Noise inherent in the lines and the equipment could be seen at lower amplitudes. Other features of the spectrum analyzer allows for a change in the scanning time as well as a determination of the positive peaks in the signal. In addition to setting these values, a high gain 800 MHz antenna is needed to monitor the emergency band. This would change the look of signal quite a bit, adding more noise and more peaks at different amplitudes.

Agilent, in addition to its simple data collection methods, provides a LAN connection on the back of its spectrum analyzer containing its own IP address. Using free software developed by Agilent (PSG/ESG), a computer can be used to log data from the spectrum analyzer as well as store and control the analyzer's settings. Using this data acquisition technique, the data from the spectrum analyzer can be passed through the LAN to a computer, through the PSG/ESG software and directly into Matlab 7.0.

Using Matlab, the data can be logged using the data acquisition toolbox, and then manipulated using its data processing techniques. Filtering can be performed as well as peak detection to determine which frequencies are in use. Using the scanning time of the Agilent Spectrum Analyzer, the time a channel is used can be determined by multiplying the amount of data points are peaked at a certain frequency by the scan time. This data can then be further used in plots and processed in the statistics toolbox to obtain further measurements.

The second data collection method analyzed was using a generic trunking scanner to sound card technique. The scanner would be limited by only being able to monitor one channel continuously at a time. Unlike the spectrum analyzer, the scanner's scanning capabilities allow it to search through the channels for one that is in use, but cannot monitor tell if two or more are in use. This would limit the amount of channels one would be able to listen to at any given moment.

The scanner to sound card configuration, though not practical, is very simple. The scanner could be set to a certain frequency and the sound from that channel could be connected to a computer's sound card using a simple male-to-male headphone jack. This configuration can be done with multiple scanners and sound cards. Up to three sounds cards can be installed in a computer at once, providing six different channels that could be utilized.

After the sound reaches the sound card, Matlab can be used to acquire the data. Using the data acquisition toolbox, the analog input from the sound card can be transferred into Matlab using the **ai = analoginput('winsound')** command. This only applies to one channel. However, the toolbox also allows for the addition of more inputs manually by the user. All sound card channels could be used and logged using Matlab.

Though not an all-in-one software feature such as Agilent provides, scanners sometimes come with software which can be used to control each scanner. This however, would require many different ports to be used for control of those scanners since an output would be necessary from the computer to each scanner. This may not be feasible with all the sound cards necessary.

After the data is logged into Matlab, the length of the audio signal could be determined using filtering techniques. By filtering out the noise in a signal, one could then measure the amount on time a signal is present (voices on the channel) and determine the length of time the band is occupied. This would have to be done multiple times for each data log collected from each scanner. The times would then have to be matched up to see at any given moment how much bandwidth is in use. This would not give as accurate a depiction as the spectrum analyzer since a limited number of channels would be monitored.

Both scanners and the spectrum analyzer can be used to monitor the emergency bands, however, the advantages of the spectrum analyzer far outweigh that of the scanner. The scanner would give a limited view of the amount of bandwidth used in the emergency bands, while the spectrum analyzer would be able to give us the most accurate analysis that we could obtain and automate.

Methodology

With the two instruments that can be used to measure the bandwidth usage in the emergency bands, the spectrum analyzer remains the best as it is the most accurate and most readily available. The Agilent E4445A PSA Series Spectrum Analyzer is connected to the Agilent Vector Signal Analyzer which converts the frequency domain signal into a digital signal with multiple points that the computer can read. This information is passed through the Agilent VXI card on the back of the computer and utilizes a firewire connection. Network connections to multiple units are also required including the Spectrum Analyzer, Signal Generator and the Windows based Agilent equipment This enables control of those machines by the computer's software.

After the connections are made the Agilent IO control must be run in order to prepare the computer to collect data from and send data to the Signal and Spectrum Analyzers. Within the Agilent Programs you will also find a program for the Spectrum Analyzer where you can control the resolution of the waveform and what frequencies you are looking at. This program shows the software functionality of the two analyzers. This however, is not useful for detecting the bandwidth usage because recording the signals would take up gigabytes worth of space every hour. To effectively measure the bandwidth usage and make other calculations on the emergency bands, Matlab is the best tool since it can directly interface with the two analyzers.

The Matlab code written was divided into three separate *.m files that together can measure the frequencies of interest from the spectrum analyzer and determine when a signal is present. This presence is then recorded and saved to the hard drive where it is later recalled and processed to determine how much bandwidth was used during certain periods of time. The three sections of code were made to run separately, these sections include the threshold section, collection and storage section, and data processing section. These three sections mesh together as the threshold section is used to find a dBm whereby a signal with a greater dBm would be deemed a "detection" and anything below would be ignored as noise. The threshold would be used by the collection and storage section which would record these "detections" and thee corresponding time they were detected. After being stored in separate files, the data processing section searches through these stored files and loads the points needed and processes the data for bandwidth usage.

Threshold Program

The threshold section was created to determine the threshold dBm required to give a false detection rate of .01% by taking 100,000 samples and finding the largest 10 values. The threshold is set to that tenth value so as out of 100,000 samples taken by the collection and storage section of the code, only 10 false detections of white noise would be made. This code using the current TestEquipment computer in the Navigation lab would take around three days to complete. Through the code the threshold voltage is obtained by setting the frequency of interest to one where the channel is not in use and only contains white noise. This noise is inherent in all channels and follows a Gaussian pattern which in theory means the white noise should follow a similar pattern with the noise signal's power. The commented code for the threshold section can be seen in Appendix C.

Collection and Storage Program

The collection and storage section of the code records “detections” at user defined frequencies until a user defined stop time. The threshold dBm from the previous section must be manually inserted into the code in order to provide for an accurate threshold voltage depending on your location and the equipment being used. After defining the threshold, the number of frequencies of interest must be entered as well as each frequency being examined. In addition, the user must ensure that the upper and lower bounds of the frequency data captured from the vector signal analyzer cover the frequencies of interest. To set these values ensure that the UserStartFreq is set to below your lowest value and the UserSpan covers the entire span of frequencies needed. Afterwards, a StopTime is also needed for input from the user.

As the code runs it will execute a batch once which start the VSAVector Application and set all the variables needed. It will then capture one moment’s worth of data and process it for the frequencies of interest which through code were turned into points of interest corresponding to the individual points taken in from the vector signal analyzer. At the same moment the capture time is recorded and placed into a matrix which will be attached to each point. From these points a comparison is made to the threshold voltage to determine if a signal is present or not and records these detections as “1”s or if there is no detection it is recorded as a “0”. A number “2” may also be recorded if the vector signal analyzer failed to complete the capture of the points and in that case will be regarded as an error.

The program then loops until the StopTime is met continually capturing more points and comparing them to the threshold. All points, however, are not placed and saved in one matrix. Matlab has a finite number of points that it can record in a matrix and is also limited by the amount of memory available for processing the data. To ensure the program does not crash because Matlab cannot handle the size of the matrix, the matrix is saved and then cleared according to what the MaxPoints variable is. This variable will be different for every machine but a good value to set it as is 35,000 points which is then divided by the number of frequencies of interest set by the user. The loop will save these files automatically and keeps track of a table with the times each saved file covers and its name which is simply a 0, 1, 2, 3, etc. When the StopTime has been exceeded, all data will save to the workspace and the code closes the VSAVector Application and clears the variables. The code can be seen in Appendix D.

Data Processing Program

The data processing portion of the code recalls the saved files from the Collection and Storage Program, loads the files corresponding to User entered times and assess the bandwidth usage for each frequency. The User is able to put in any period of time that matches the dataset currently in the workspace folder where the files from the Collection and Storage section was saved. From the user-defined start and stop times, the program opens the file detailing each of the saved files and scans for the first batch of vectors that exceeds the start time. It then loads the previous file and searches through the vector times to find the exact start time. After finding the start time within the file, the program begins to count the amount of time each frequency is occupied by assessing the ones and zeros and their corresponding times. The program cycles through and uses the **etime** function to assess the total amount of time the channel is occupied. Each file is loaded and processed in the same way until the stop time is reached.

After the stop time is reached, the program then uses the total time of channel “in-use” time and compares it to the elapsed time between the start and stop times. This give a percentage of bandwidth usage for each frequency of interest. The program also counts the amount of time multiple channels are occupied as well as all channels are occupied. These numbers are output to two graphs detailing times when multiple channels are used and the percentage of time each channel is occupied. This code can be seen in Appendix E.

Results and Conclusions

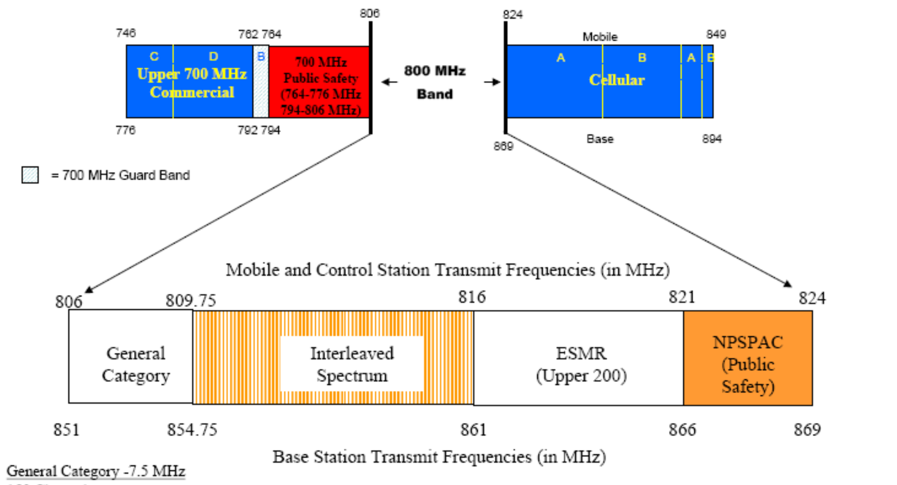
With the three sections of code written and described, each section can be run in Matlab to determine the bandwidth usage of the Worcester Emergency Bands. This portion of the project, however, was not completed due to technical problems arising from the Agilent software. The cryptkey which confirms the license on the software had corrupted itself and technical support was consulted on this. After some time another call was placed and I was able to receive a temporary key for the software. This solved the problem of running the software, but the software itself must have corrupted some files. As this occurred for the last couple weeks of the project, I was unable to reinstall the program and run the program to obtain the correct values.

With that said, much of the code was run through and debugged to ensure values were being recorded even though the values were wrong. Time did not allow for the debugging of the Data Processing section of the code and neither did the malfunctioning software. Should the software be reinstalled and the equipment works properly, these three sections of code will provide the necessary tools to assess the bandwidth usage of the Worcester Emergency bands and are flexible enough to measure the usage of other frequencies. With that information, conclusions can be drawn about the FCC’s allotment of mobile stations per channel. Different analyses of the information may also reveal other characteristics about bandwidth usage as well.

This model can be used for multiple different frequencies and the User Interface can be made much easier with extra coding and knowledge of the Matlab GUI. Another option is to export the code to C+ or Visual Basic and code through a GUI there which will allow users that are not familiar with Matlab to use the program. The code already written, however, provides a good basis for any future code to be written and for accurate conclusions to be made upon the data should collection take place.

Appendix A – 800 MHz Reconfiguration Plan

*Figures taken from <http://wireless.fcc.gov/publicsafety/800MHz/bandreconfiguration/downloads/CWS_presentation_ITA_10-5-04.pdf>



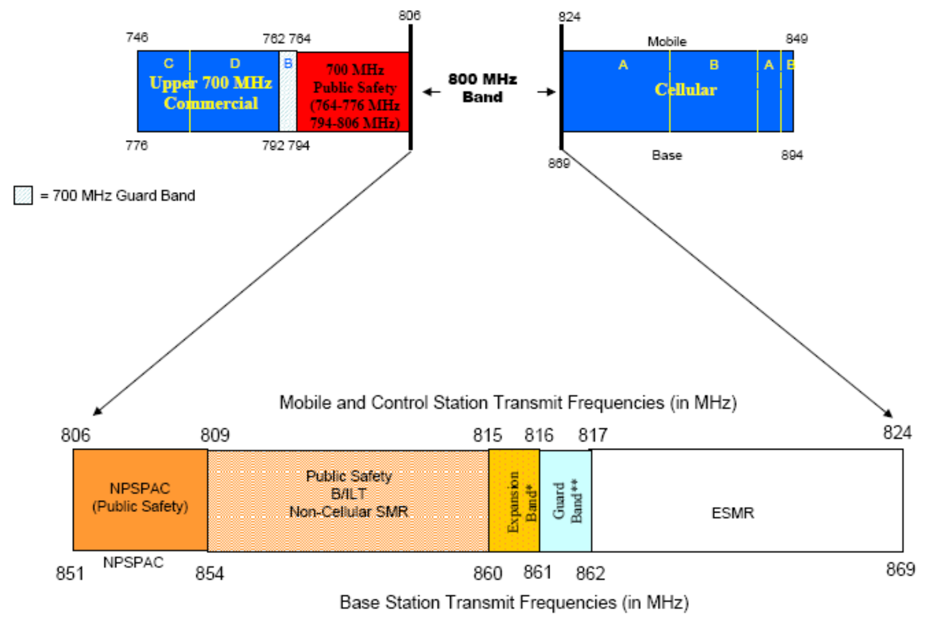
General Category -7.5 MHz
150 Channels
Licensed by EA Blocks of 25 channels (SMR)
Some Incumbent Operators Remain

ESMR/Upper 200 – 10 MHz
200 Channels
Licensed by EA
Non EA incumbents are currently undergoing mandatory relocation

NPSPAC - 6 MHz
225 Channels @ 12.5 kHz spacing
5 Channels @ 25 kHz spacing
5 Mutual Aid Channels

Interleaved Spectrum -12.5 MHz
250 Channels
80 SMR Channels
(Licensed by EA, Some Incumbent Operators Remain)
70 Public Safety Channels
50 Business Channels
50 Industrial Land Transportation Channels

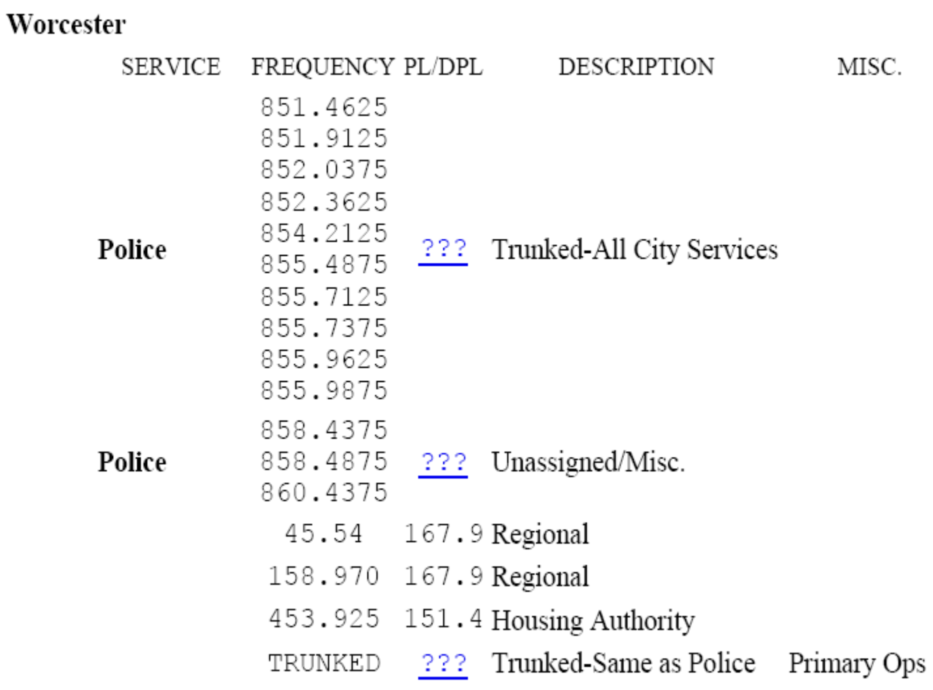
PRE-RECONFIGURATION BAND PLAN



*No public safety system will be required to remain in or relocate to the Expansion Band; although they may do so if they choose.
**No public safety or CII licensee may be involuntarily relocated to occupy the Guard Band.

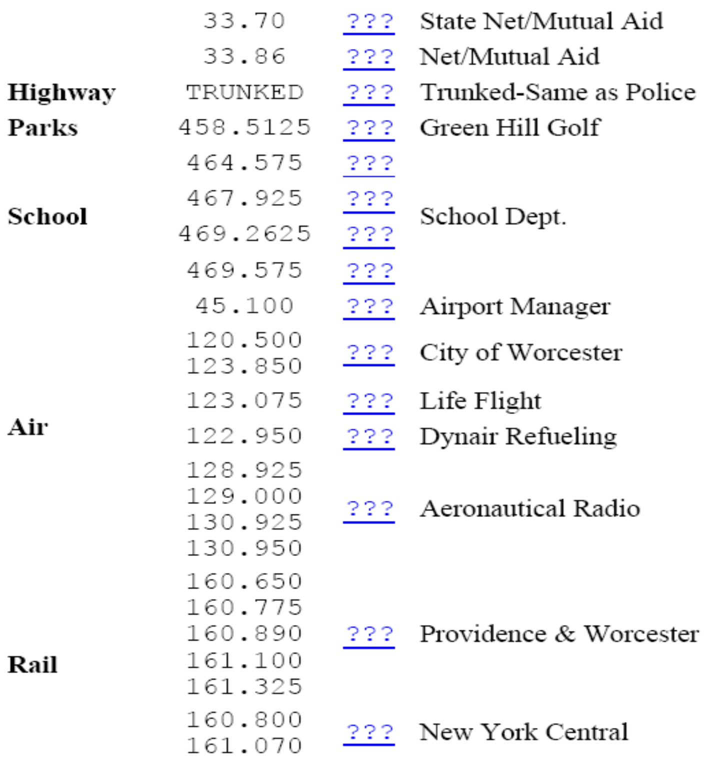
POST-RECONFIGURATION BAND PLAN

Appendix B – Worcester Emergency Frequencies



Worcester

SERVICE	FREQUENCY	PL/DPL	DESCRIPTION	MISC.
Police	851.4625 851.9125 852.0375 852.3625 854.2125 855.4875 855.7125 855.7375 855.9625 855.9875	???	Trunked-All City Services	
Police	858.4375 858.4875 860.4375	???	Unassigned/Misc.	
	45.54	167.9	Regional	
	158.970	167.9	Regional	
	453.925	151.4	Housing Authority	
	TRUNKED	???	Trunked-Same as Police	Primary Ops



	33.70	???	State Net/Mutual Aid
	33.86	???	Net/Mutual Aid
Highway	TRUNKED	???	Trunked-Same as Police
Parks	458.5125	???	Green Hill Golf
School	464.575	???	School Dept.
	467.925	???	
	469.2625	???	
	469.575	???	
Air	45.100	???	Airport Manager
	120.500 123.850	???	City of Worcester
	123.075	???	Life Flight
	122.950	???	Dynair Refueling
	128.925 129.000 130.925 130.950	???	Aeronautical Radio
Rail	160.650 160.775 160.890 161.100 161.325	???	Providence & Worcester
	160.800 161.070	???	New York Central

Appendix C – Threshold Matlab Code

```
%   THRESHOLD takes multiple measurements from the Agilent
%   Technologies 89600 Vector Signal Analyzer Application.
%
%   This routine creates, sets up, starts a measurement, reads
%   voltage data, processes the data for the corresponding point of
%   interest, determines the highest 10 voltage readings from 100,000
%   samples in order to attain a threshold with a .01% false detection
%   rate.

% Clear any variables and matrices
clear all;

% Initialize the Failed variable
Failed = 0;

% Start frequency is set.  Span is set to a small value since our interest
% is only in a single point.
UserStartFreq = 852000000;
UserSpan = 2000000;
CenterFreq = UserStartFreq + (UserSpan / 2);

% PointsResolution can be used to change the resolution of the Vector
% Signal Analyzer.  Set values are as follows : 51, 101, 201, 401, 801,
% 1601, 3201, 6401, 12801, 25601, 51201, 102401.  The spacing between each
% point is then calculated.
PointsResolution = 102401;
PointSpacing = UserSpan / PointsResolution;

% Frequency of interest is set to an 800Mhz band frequency not in use
% in the area.
UserFreqInterest = 853458000;

% Point of Interest is calculated
PtInterest = round((UserFreqInterest - UserStartFreq) / PointSpacing) + 1;

% Creates a new object which opens the AgtVsaVector application.
% Note: This may interfere with instances of this application which are
% already running.
hApp = actxserver('AgtVsaVector.Application');

% Get interfaces to major items in the instrument.
hMeas = get(hApp,'Measurement');
hDisp = get(hApp,'Display');

% Set all measurement defaults in the Spectrum Analyzer to their defaults
invoke(hDisp,'Default');
invoke(hMeas,'Default');
invoke(hMeas,'Reset');

% Set Center and Span frequencies and the Resolution Bandwidth (points)
hFreq = get(hMeas,'Frequency');
set(hFreq,'Center', CenterFreq);
set(hFreq,'Span', UserSpan);
set(hFreq, 'Points', PointsResolution);
```

```
% Set input range to 1 dBm on Channel 1
hInputs = get(hMeas,'Inputs');
hInputChans = get(hInputs,'InpChannels');
hInputChan = get(hInputChans,'Item',1);
set(hInputChan,'Range',1);

% The default trace 1 shows a spectrum.
% Set trace 1 to display dBm.
hTraces = get(hDisp,'Traces');
hTrace1 = get(hTraces,'Item',1);
enumVsaTrcFormat = 0;                           % vsaTrcFmtLogMag = 0
set(hTrace1,'Format',enumVsaTrcFormat);

% Set for single measurement
set(hMeas,'Continuous',0);    % False = 0

% Start measurement
invoke(hMeas,'Start');

% Wait until the measurement is done.  If measurement is done
% vsaStatusBitMeasDone = 1, if not it equals 0.
% Set timeout to 5 seconds
enumVsaStatusBits = uint32(1);            % vsaStatusBitMeasDone = 1
bMeasDone = 0;
t0=clock;
while(bMeasDone==0 & etime(clock,t0)<=5)
   pause(.5);
   hStatus = get(hMeas,'Status');
   Value = get(hStatus,'Value');
   bMeasDone = bitand(uint32(Value), enumVsaStatusBits);
   release(hStatus);
end

% Check if meas was stuck, if stuck sends a fail to complete and skips to
% take another measurement.  Places a '2' in the ChannelUse vector to
% signify a failed measurement.

if bMeasDone == 0
      Failed = 1;
else

    % Scale the displays properly
    hTrace2 = get(hTraces,'Item',2);
    invoke(hTrace1,'YScaleAuto');
    invoke(hTrace2,'YScaleAuto');

    % Get the Amplitude data
    enumVbVarType = 5;              % vbDouble = 5
    enumVsaTrcDataSelect = 2; % vsaTrcDataY = 2

    vArrY = get(hTrace1,'Data',enumVbVarType,enumVsaTrcDataSelect,0);

    Amplitude = vArrY(PtInterest)

end
```

```
% Counts the number of points checked until 100000
Points = 1;

% Loops the following according to the points collected desired to end.
% Currently it is set to 100000 points to give a .01% False Detection Rate
while(Points < 100000)

% Clear the Amplitude Data
    clear vArrY

% Repeats previous code that takes in a measurement.
    set(hMeas,'Continuous',0);      % False = 0
    invoke(hMeas,'Start');

    enumVsaStatusBits = uint32(1);        % vsaStatusBitMeasDone = 1
    bMeasDone = 0;
    t0=clock;
    while(bMeasDone==0 & etime(clock,t0)<=5)
       pause(.5);
       hStatus = get(hMeas,'Status');
       Value = get(hStatus,'Value');
       bMeasDone = bitand(uint32(Value), enumVsaStatusBits);
       release(hStatus);
    end

    if bMeasDone == 0
     Failed = Failed + 1;
    end

    hTrace2 = get(hTraces,'Item',2);
    invoke(hTrace1,'YScaleAuto');
    invoke(hTrace2,'YScaleAuto');

    enumVbVarType = 5;              % vbDouble = 5
    enumVsaTrcDataSelect = 2; % vsaTrcDataY = 2

    vArrY = get(hTrace1,'Data',enumVbVarType,enumVsaTrcDataSelect,0);

% Appends the new amplitude data onto the end of the vector already created
    Amplitude = [Amplitude; vArrY(PtInterest)];

% Increments Points for the Loop
    Points = Points + 1;

end

% Quit the VSAVector application
invoke(hApp,'Quit');

% Release objects
release(hTrace1);
release(hTrace2);
release(hTraces);
release(hInputChan);
release(hInputChans);
release(hInputs);
release(hFreq);
release(hDisp);
release(hMeas);

% Delete the application object
delete(hApp);

% Sort the Amplitude by inverting the vector and then using the sort
% command.  Picks out the tenth value and stores it in threshold.
AmplitudeInv = [Amplitude]';
SortedAmplitude = sort(AmplitudeInv, 'descend');
Threshold = SortedAmplitude(10);
```

Appendix D – Collection and Storage Matlab Code

```
%   COLLECTION AND STORAGE takes multiple measurements from the Agilent
%   Technologies 89600 Vector Signal Analyzer Application.
%
%   This routine creates, sets up, starts a measurement, reads
%   voltage data, processes the data for the corresponding points of
%   interest, stores the outcome of that processing, and quits the
%   application.
%
%   The application may loop several times depending on the time
%   constraints placed onto the program by the user.

% Clear any previous data and variables in the Matlab Workspace
clear all;

% User can set the Start Frequency as well as the Span as long as it is
% below 32 MHz.  Center Frequency is calculated from these values to be
% input into the Agilent 89600.
UserStartFreq = 851000000;
UserSpan = 5000000;
CenterFreq = UserStartFreq + (UserSpan / 2);

% User must set the Stop Time for Data collection.
Year = 2006;
Month = 2;
Day = 28;
Hours = 2;
Minutes = 22;
Seconds = 0;

% Puts User Stop Time in the proper form
UserStopTime = [Year Month Day Hours Minutes Seconds];


% PointsResolution can be used to change the resolution of the Vector
% Signal Analyzer.  Set values are as follows : 51, 101, 201, 401, 801,
% 1601, 3201, 6401, 12801, 25601, 51201, 102401.  The spacing between each
% point is then calculated.
PointsResolution = 102401;
PointSpacing = UserSpan / PointsResolution;


% Threshold dBm determined for a .01% False Detection Rate.
Threshold = -80.109;


% User may set multiple frequencies of interest.  After the frequencies are
% submitted, the corresponding points to those frequencies are calculated.
% Warning : Points may not be on the exact frequency due to the Point
%           Spacing.
UserSetFreqNum = 10;
UserFreqInterest = [851462500 851912500 852037500 852362500 854212500
855487500 855712500 855737500 855962500 855987500];

% Maximum points allowable to create a safe buffer between data collection
% variable and the maximum variable size due to memory allocation and index
% restrictions.
MaxPoints = 35000 / UserSetFreqNum;


% Adjusts UserSetFreqNum to be used in a loop to create variables.  65
% corresponds to char(65) = A
FreqAdjust = UserSetFreqNum + 65;
for i = 1 : UserSetFreqNum,
    PtInterest(i) = round((UserFreqInterest(i) - UserStartFreq) /
PointSpacing) + 1;
end



% Initialize variables
SaveNum = 0;
PointsMem = 0;

% Create a new 89600 object by opening the AgtVsaVector.exe application
% Warning: This may interfere with instances of this application which are
% already running.
hApp = actxserver('AgtVsaVector.Application');

% Gets the measurement and display interfaces to the E4445A PSA Series
% Spectrum Analyzer
hMeas = get(hApp,'Measurement');
hDisp = get(hApp,'Display');

% Set to defaults and Reset the measurement interface to ensure
% measurements have not begun and are not being exported to the 89600
% Vector Signal Analyzer
invoke(hDisp,'Default');
invoke(hMeas,'Default');
invoke(hMeas,'Reset');


% Set Center and Span frequencies and the Resolution Bandwidth (points)
% using User defined Values
hFreq = get(hMeas,'Frequency');
set(hFreq,'Center', CenterFreq);
set(hFreq,'Span', UserSpan);
set(hFreq, 'Points', PointsResolution);

% Set input range to 1 dBm on Channel 1
hInputs = get(hMeas,'Inputs');
hInputChans = get(hInputs,'InpChannels');
hInputChan = get(hInputChans,'Item',1);
set(hInputChan,'Range',1);

% The default trace 1 shows a logaritmic spectrum of the User defined
% frequency band.
hTraces = get(hDisp,'Traces');
hTrace1 = get(hTraces,'Item',1);

% Sets trace 1 to display dBm so the upcoming read will return dBm.
```

```
enumVsaTrcFormat = 0;                                      % vsaTrcFmtLogMag
= 0
set(hTrace1,'Format',enumVsaTrcFormat);

% Set for single measurement so a snapshot is taken of the spectrum that
% can be exported to the Vector Signal Analyzer and then to Matlab
set(hMeas,'Continuous',0);    % False = 0

% Start measurement
invoke(hMeas,'Start');

% Wait for measdone to ensure the full User spectrum is measured and the
% measurement did not get stuck.  If the measurement is stuck, bmeasdone =
% 0 and quits loop after 5 seconds.
% Set timeout to 5 seconds

enumVsaStatusBits = uint32(1);            % vsaStatusBitMeasDone = 1
bMeasDone = 0;
t0=clock;
while(bMeasDone==0 & etime(clock,t0)<=5)
   pause(.5);
   hStatus = get(hMeas,'Status');
   Value = get(hStatus,'Value');
   bMeasDone = bitand(uint32(Value), enumVsaStatusBits);
   release(hStatus);
end

% Record the time of measurement and save to SaveTable as well as put in
% the ChannelUse Matrix
ChannelUse.Time = clock;
SaveTable = [0 clock];

% Check if meas was stuck, if stuck sends a fail to complete and skips to
% take another measurement.  Places a '2' in the ChannelUse Matrix under
% all the Frequencies to signify a failed measurement.  If the measurement
% was not stuck, processes the points of interest to see if they are above
% the threshold dBm.
if bMeasDone == 0
      for j = 65 : (FreqAdjust - 1),
        ChannelUse.(char(j)) = 2;
    end
    j = 65;
else

    % pretty up the displays
    hTrace2 = get(hTraces,'Item',2);
    invoke(hTrace1,'YScaleAuto');
    invoke(hTrace2,'YScaleAuto');

    % get amplitude data
    enumVbVarType = 5;              % vbDouble = 5
    enumVsaTrcDataSelect = 2; % vsaTrcDataY = 2

    vArrY = get(hTrace1,'Data',enumVbVarType,enumVsaTrcDataSelect,0);

    % get phase data x axis (frequencies)
    enumVsaTrcDataSelect = 1; % vsaTrcDataX = 1
    vArrX = get(hTrace1,'Data',enumVbVarType,enumVsaTrcDataSelect,0);

    % Loop to determine when channel is above threshold (in use).  If in use
    % then ChannelUse will have a 1 appended onto the end of it.  If not, a 0
    % will be appended.

    for j = 65 : (FreqAdjust - 1),
        z = j - 64;
        if vArrY(PtInterest(z)) > Threshold
            ChannelUse.(char(j)) = 1;
        else
            ChannelUse.(char(j)) = 0;
        end
    end

    j = 65;
end


% Loops the following according to the time desired to end.
while(etime(UserStopTime, clock) > 0)

    clear vArrY
    invoke(hMeas,'Start');

    % wait for measdone, but don't bother it too often
    % Set timeout to 5 seconds
    enumVsaStatusBits = uint32(1);        % vsaStatusBitMeasDone = 1
    bMeasDone = 0;
    t0=clock;
    while(bMeasDone==0 & etime(clock,t0)<=5)
       pause(.1);
       hStatus = get(hMeas,'Status');
       Value = get(hStatus,'Value');
       bMeasDone = bitand(uint32(Value), enumVsaStatusBits);
       release(hStatus);
    end

    ChannelUse.Time = [ChannelUse.Time; clock];
    if PointsMem == MaxPoints
        TimeSave = clock;
    end



    % check if meas was stuck
    if bMeasDone == 0
      for j = 65 : (FreqAdjust - 1),
            ChannelUse.(char(j)) = 2;
        end
        j = 65;
    else

    % pretty up the displays
    hTrace2 = get(hTraces,'Item',2);
    invoke(hTrace1,'YScaleAuto');
    invoke(hTrace2,'YScaleAuto');
    % get amplitude data
    enumVbVarType = 5;              % vbDouble = 5
    enumVsaTrcDataSelect = 2; % vsaTrcDataY = 2

    vArrY = get(hTrace1,'Data',enumVbVarType,enumVsaTrcDataSelect,0);


    for j = 65 : (FreqAdjust - 1)
        z = j - 64;
        X = vArrY(PtInterest(z))
        if vArrY(PtInterest(z)) > Threshold
            ChannelUse.(char(j)) = [ChannelUse.(char(j)); 1];
        else
            ChannelUse.(char(j)) = [ChannelUse.(char(j)); 0];
        end
        j = 65;
    end

    end



    % Saves vectors to a file and clears the vectors so they do not exceed
the
    % maximum allowable size by Matlab and memory.
    if PointsMem == MaxPoints
        SaveNum = SaveNum + 1;
        save(['Trial', num2str(SaveNum)], 'ChannelUse');
        clear ChannelUse;
        PointsMem = 0;
        SaveTable = [SaveTable; SaveNum TimeSave];

        for j = 65 : (FreqAdjust - 1),
            ChannelUse.(char(j)) = 2;
        end
        ChannelUse.Time = clock;
        j = 65;

    else
        PointsMem = PointsMem + 1;
    end

end

if SaveNum == 0
    SaveNum = SaveNum + 1;
    save(['Trial', num2str(SaveNum)], 'ChannelUse');
    clear ChannelUse;
end

save('Table of Saved Files', 'SaveTable');

invoke(hApp,'Quit');

% Release objects
release(hTrace1);
release(hTrace2);
release(hTraces);
release(hInputChan);
release(hInputChans);
release(hInputs);
release(hFreq);
release(hDisp);
release(hMeas);

% Delete App
delete(hApp);
```

Appendix E – Data Process Matlab Code

```
% DATAPROCESS evaluates data from the COLLECTION mfile and the resulting
% files that are saved in the workspace.  Using user defined times, this
% program determines the bandwidth usage over the course of a given time.

% Attention: Make sure you are working in the proper workspace with all of
% the saved files from DATAPROCESS or else this program will not work.

% User defined Start and End Times taken within the constraints of the files
saved
Start.Year =
Start.Month =
Start.Day =
Start.Hour =
Start.Minute =
Start.Second =
StartTime = [Start.Year Start.Month Start.Day Start.Hour Start.Minute
Start.Second];

End.Year =
End.Month =
End.Day =
End.Hour =
End.Minute =
End.Second =
EndTime = [End.Year End.Month End.Day End.Hour End.Minute End.Second];

TotalChannels = 10;

% Load the table of times and corresponding saved files.
Load('Table of Saved Files', 'SaveTable')

% Initialize x to zero.  x will be used to count through the loop below
x = 0;

% Takes the time from the matrix holding start and stop times for each
% saved files in the workspace.
FileTime = SaveTable(x);

% Searches for the User defined start time by comparing that time to the
% times within the saved files in the workspace.  Stops when the start time
% is greater than the FileTime
while(etime(StartTime, FileTime) > 0)
    x = x + 1;
    FileTime = SaveTable(x);
end

% Set the reference number for the first file of interest back one so a
search
% can be conducted on the individual file to find the exact start time match
% within.
x = x - 1;

% Loads the current file of interest for searching
Load ('Trial', [num2str(x)], 'ChannelUse');
```

```
% Initialize the variable
timestampused = 0;

% Creates a loop which sorts through each file and loads additional files
% until the User defined stop time.  The sorting counts the amount of time
% the channel is occupied.
for y = 1 : TotalChannels
    TotalTime(y) = 0
end

z = 1;
a = 1;

% Loads the first line of the matrix including the time and whether each
% channel was occupied or not.
FileTimeRecall = ChannelUse.Time(z, :)

% Loops to count how many times each channel is in use and totals it up.
% Flag determines which channels are in use and the sum tells how many
% channels were in use at any one time.
while(etime(FileTimeRecall, EndTime) > 0)

    for j = 65 : (FreqAdjust - 1),
        if (ChannelUse.(char(j))(z) = 1) & (Flag.(a) == 0)
            TimeStart(a) = ChannelUse.Time(z, :);
            Flag(a) = 1;
            a = a + 1;
        elseif (ChannelUse.(char(j))(z) == 0) & (Flag.(y) == 1)
            Flag(a) = 0;
            TotalTime(a) = TotalTime(a) + etime(ChannelUse.Time(z),
TimeStart(a));
        end
    end

    ChsUsed = sum(Flag);

    % Loops to determine how many channels were used at one time.  Places
into the
    % variable FlagB which keeps track of times when multiple channels are
used.
    % TotalTimeMulti keeps track of the amount of time multiple channels are
used.
    for w = (1 : (TotalChannels - 1))
        if (ChsUsed > w) & (FlagB.(w) == 0)
            FlagB.(w) = 1;
            TimeStartMulti(w) = ChannelUse.Time(z);
        elseif (ChsUsed <= w) & (FlagB.(w) == 1)
            FlagB.(w) = 0;
            TotalTimeMulti(w) = TotalTimeMulti(w) + etime(ChannelUse.Time(z),
TimeStartMulti(w));
        end
    end

    z = z + 1;
    FileTimeRecall = ChannelUse.Time(z);
    % Determines if the loop is at the end of the loaded file.  If it is the
code
    % loads a new file.
    if FileTimeRecall = 0
        x = x + 1;
        z = 1;
        Load ('Trial', [num2str(x)], 'ChannelUse');
        FileTimeRecall = ChannelUse.Time(z, :);
    end

    if ChannelUse.Time(z, 4) < ChannelUse.Time((z+1), 4)
        if timestampused = 0
            TimeStamp = [ChannelUse.Time((z+1), 4) ChannelUse.Time((z+1), 3)
TotalTime TotalTimeMulti];
            timestampused = 1;
        else
            TimeStamp = [TimeStamp; ChannelUse.Time((z+1), 3)
ChannelUse.Time((z+1), 4) TotalTime TotalTimeMulti];
        end

        TotalTime = 0;
        TotalTimeMulti = 0;
    end
end

% Collects to total time elapsed as well as the TotalTime for each channel.
% Time is also calculated for time multiple channels were occupied.
TotalTime = TimeStamp(:, 3: (TotalChannels + 3)) / 3600;
TotalTimeMulti = TimeStamp(:, (3 + TotalChannels): (3 + 2 * TotalChannels)) /
3600;
TotalElapsedTime = etime(StartTime, EndTime);

% Creates two bar graphs for the total amount of time used for each channel
% as well as the amount of time multiples channels are used
bar(TimeStamp(:, 2), TotalTime);
bar(TimeStamp(:, 2), TotalTimeMulti);

% Calculates the bandwidth used in each channel and the total bandwidth
% of all the channels
for g = (3: TotalChannels + 3)
    TotalTime = TimeStamp
    for h = (TotalChannels + 3 : (2 * TotalChannels + 3))
        for k = (65: 65 + TotalChannels)
            TotalFreq.(char(k)) = sum(Timestamp(:, g)) / 3600;
            MultiFreq.(char(k)) = sum(Timestamp(:, h)) / 3600;
            TotalBand.(char(k)) = sum(Timestamp(:, g))/ TotalElapsedTime *
100;
            TotalMultiBand.(char(k)) = sum(Timestamp(:, h))/ TotalElapsedTime
* 100;
        end
    end
end
```

Works Cited

American Home Outlet. 8 February 2006. American Home Outlet. 13 March 2006. <http://www.americanhomeoutlet.com/index.php?main_page=product_info&products_id=34135&zenid=82deb51f62ef6d85eacf6f6e8e2323d2>

Booz·Allen & Hamilton, Inc. – 800 Mhz Study. 23 March 1998. SAFECOM. 27 Jan 2006. <http://www.safecomprogram.gov/NR/rdonlyres/1EC47ABD-89BA-40F2-B55A-F83C4B4189A8/0/800MHz_study.pdf>

Code of Federal Regulation Title 47, Volume 5. 1 October 2005. APCO International. 27 Jan. 2006. <http://www.apcointl.org/frequency/90.txt>

USB Gear. 2005. USB Gear. 13 March 2006. <http://www.usbgear.com/USBG-X3S.html>