THE UNIVERSITY OF QUEENSLAND

AUSTRALIA

Low Cost Embedded Passive Bistatic Radar Detection Testbed

by

Flynn Kelly
4741858
flynn.kelly@uqconnect.edu.au

School of Electrical Engineering and Computer Science,
University of Queensland

Submitted for the degree of Bachelor of Science (Honours)
in the division of Electrical Engineering

4 November 2024

Flynn Kelly
flynn.kelly@uqconnect.edu.au

October 17, 2024

Professor Michael Bruenig
Head of School
School of Electrical Engineering and Computer Science
The University of Queensland
St Lucia, Queensland 4072

Dear Professor Bruenig,

In accordance with the requirements of the degree of Bachelor of Science (Honours) in the School of Electrical Engineering and Computer Science, I present the following thesis entitled

'Low Cost Passive Bistatic Radar Detection Testbed'

This thesis was performed under the supervision of Professor Bialkowski(EECS). I declare that the work submitted in the thesis is my own, except as acknowledged in the text and footnotes, and that it has not previously been submitted for a degree at the University of Queensland or any other institution.

Yours sincerely,

Flynn Kelly

Acknowledgements

Acknowledgements: recognise those who have been instrumental in the completion of the project. Acknowledgements should include any professional editorial advice received including the name of the editor and a brief description of the service rendered.

SUPERVISOR, FRIENDS, ETC....

Abstract

Abstract: MINIMAL VIABLE PRODUCT OVERVIEW
broad need, specific need, response aim, response methods, key outcomes and implications.

Affordable and effecient passive radar based detection, tracking and situational awareness technology is increasing in demand in a range of sectors and academia. As software-defined radio (SDR) technology becomes more widespread and single-board computers (SBCs) continue to gain processing power, the possibility of deploying passive radar systems in an embedded, cost-effective manner has emerged as a critical area of research and commercial development. The growing need for such systems is particularly evident in scenarios where traditional, expensive PC hardware setups at remote receiver (RX) locations are impractical or cost-prohibitive. This has created a specific need for streamlined, scalable systems that can function independently and effectively in various environments.

In response to these demands, this project aims to design and implement a low-cost, embedded passive radar detection testbed prototype system that leverages a digital broadcast signal as an illuminator of opportunity. The chosen approach involves utilizing existing embedded IoT hardware in combination with established digital signal processing (DSP) techniques and radar filtering algorithms. This integrated solution is intended to form a single embedded Linux testbed capable of handling both signal sampling and processing, along with sampling networking thereby eliminating the reliance on costly and complex physical hardware at each RX location.

Key outcomes of this research include the successful detection of aerial vehicles within controlled environments, an analysis of the system's latency, and a comprehensive evaluation of the overall cost-effectiveness of the design. The implications of this work include offering valuable insights into the scalability and cost-quality balance that can be achieved in future passive radar system implementations. Moreover, the project aims to deliver a re-usable, scaleable testbed which can be used to further develop and test passive radar based detection and situational awareness systems.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

To Do:

This section will contain a clear definition of the thesis topic, goals, project scope, and relevance of the project.

1.1 Topic and Relevance

Passive radar detection technology is a class of radar detection whereby the radar system does not emit any radiation. Instead, it uses existing electromagnetic signals in the environment, such as television or radio broadcasts, to detect and track objects. Passive radar can be bistatic, whereby the transmitter and receiver are separate, or multistatic, where there are multiple receivers. The technology has been around since the early 20th century, but has only recently become feasible due to advances in digital signal processing and computing [17]. The technology has a number of advantages over traditional radar systems. It is covert, as it does not emit any radiation, and is therefore difficult to detect and directly jam, leading to a concentrated interest from defence cirlces [20]. It is also relatively cheap, as it does not require a dedicated transmitter and hence has less energy consumption. Conversely, it has a number of disadvantages, such as a lower signal-to-noise ratio, and a requirement for a relatively large amount of computational power to process the received signals [17].

Bistatic passive radar detection has a wide range of applications centered around situational awareness, including air traffic control, border security, and environmental monitoring. Embedding the passive radar technology is a relatively new field buoyed by recent and increasing developments in computational power on Internet of Things (IoT) devices [26]. Key components and processes of a passive radar RX system include the antenna, the receiver hardware (software defined radio), signal sampling hardware and memory, along with the compute to complete the digital signal processing [18]. Advantages of implementing some or even all of these components in a low cost manner include increased scaleability, portability, and lower reliance on external high cost hardware such as a PC.

1.2 Aims and Objectives

To explore the feasibility of a low cost embedded passive bistatic radar detection, and provide a scaleable proof of concept, this thesis aims to provide a user friendly testbed for digital signal based passive radar detection. The work conducted in this thesis and the eventual prototype hopes to decrease the barriers to entry for passive radar detection technology, and provide

a platform for further research and development in the field. At a high level, the primary objectives of this thesis project are;

1. Develop an understanding and proof of concept of passive radar detection algorithms on low cost single board computers (SBC) connected to software defined radio (SDR) hardware.

2. Design and implement a small-sized modular passive radar detection system on single board computer hardware with an emphasis on user friendliness and scalability.

3. Verify the functionality of the low cost embedded passive radar detection system in a controlled environment against higher power computing results, and investigate the potential for scaling up to a multistatic system through a network.

also compare SDR modules?

1.3 Scope

This thesis focuses on the development of a hardware testbed based on software defined radio (SDR) and single board computer (SBC) technology. The scope of this thesis project is limited to the following:

- The development of a passive radar detection system using existing digital broadcast signals as illuminators of opportunity,
- The design of a small scale, modular, low cost system using single board computer hardware, specically with simple user buttons to commence and cease data collection and processing, and
- The verification of the passive radar detection system performance and latency in a controlled environment agains higher cost PC hardware.
- Exploring the potential for scaling up the system to a multistatic, remote RX system through a network.

Given that passive radar based detection is a broad topic with much academic progress and commercial technological development, this thesis will not aim to cover the broad subject area. Consequently, the following topics are out of scope for this project:

- The development of a complete bistatic passive radar detection system optimized for low latency, and object classification,
- The development of a full multistatic passive radar detection system, whereby angle of approach and precise location of a target can be determined, and
- The development of a passive radar detection system using non-digital broadcast signals as illuminators of opportunity.
- The modification and development of digital signal processing / detection algorithms optimized for latency and customised hardware applications.

Chapter 2

Background

Background:NEED TO RESEARCH AND DISCUSS THE EFFECTS OF NOISE AND TYPICAL ATTNENUATION - TEXTBOOK

This chapter collates the necessary background information for the project, including the fundamentals of passive radar, the use of illuminators of opportunity, range doppler mapping, radio hardware, digital signal processing, IoT architecture, and networking with embedded hardware.

2.1 Passive Radar Fundamentals

The key and unique feature of passive radar is its utilisation of existing illuminators of opportunity, such as television or radio broadcasts, to detect and track objects. The technology has been around since the early 20th century, with modern interest accelerated due to the use of passive radar systems on UHF TV signals and VHF FM radio tranmission systems in the 1980's [18]. Equivalent terms used to describe passive radar include passive coherent location (PCL), and passive covert radar (PCR), parasitic radar, piggyback radar. Specifically, *bistatic* radar refers to the distributed design of the transmitter and receiver, as opposed to classic *monostatic* radar. As reflectd by Figure 2.1 below, the turning parabolic of monostatic radar is able to receive both range and bearing of the signal echo, whereas passive bistatic radar measures time delay of the echos from the target, allowing doppler shift from the relative speed of the target to be measured.

The geometry of passive bistatic radar can be further explored and equations can be mapped accordingly, with the distance between the transmitter and receiver R being determined by known quantities such as the baseline as reflected below in Figure 2.2.

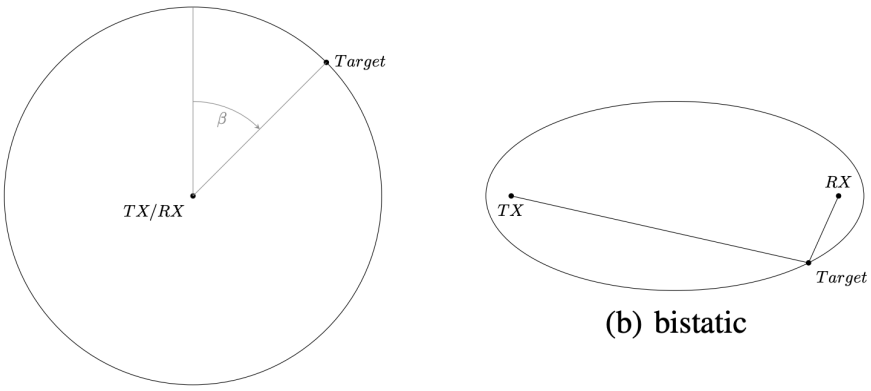
The bistatic range R_R is given by:

$$R_R = \frac{(R_T + R_R)^2 - L^2}{2(R_T + R_R + L\sin\theta_R)} \tag{2.1}$$

The Doppler shift f_D is given by the rate of change of the bistatic range sum:

$$f_D = \frac{1}{\lambda}\frac{d}{dt}(R_T + R_R) \rightarrow f_D = \frac{2v}{\lambda}\cos\delta\cos(\frac{\beta}{2}) \tag{2.2}$$

In the case of this project, both the TX (illuminator of opportunity) and the RX (embedded passive detection system) will be static, and the target will be moving, simplifying the mathematical calculations as much as possible, resulting in the cos version of equation 2 above.



(a) monostatic

(b) bistatic

Figure 2.1: Monostatic (a) and bistatic (b) radar topologies [26]

The Doppler shift will be used to determine the speed of the target as well as its relative directional motion, and the range will be used to determine the distance of the target from the receiver. Another important feature of bistatic passive radar systems is its performance which can be equated through the bistatic radar equation, which is equivalently derived as the monostatic radar equation [18].

$$\frac{P_r}{P_n} = \frac{P_t G_t}{4\pi R_T^2} \cdot \sigma_B \cdot \frac{1}{4\pi R_R^2} \cdot \frac{G_r \lambda^2}{4\pi} \cdot \frac{1}{kT_0 BF} \tag{2.3}$$

Where:

- P_r is the received target echo power.
- P_n is the receiver noise power.
- P_t is the transmit power.
- G_t is the transmit antenna gain.
- R_T is the transmitter-to-target range.
- σ_B is the target bistatic radar cross section.
- R_R is the target-to-receiver range.
- G_r is the receive antenna gain.
- λ is the signal wavelength.
- k is Boltzmann's constant (1.38×10^{-23} JK^{-1}).
- T_0 is the noise reference temperature.
- B is the receiver effective bandwidth.
- F is the receiver effective noise figure.

The denominator of the bistatic radar equation includes the term $\frac{1}{R_T^2 R_R^2}$.This term implies that with omnidirectional antenna patterns, the contours of constant signal-to-noise ratio (SNR) are described by the equation $R_T R_R$ = constant, which represents Ovals of Cassini. These ovals represent the locations in a given PBR co-ordinate sytem where the distances from the target to the transmitter and receiver remain the same. In the case of directional antennas, these contours are altered. Moreover, the signal-to-noise ratio is minimized when the target is equidistant from the transmitter and receiver ($R_T = R_R$), and maximized when the target is closer to either the transmitter or receiver [18].

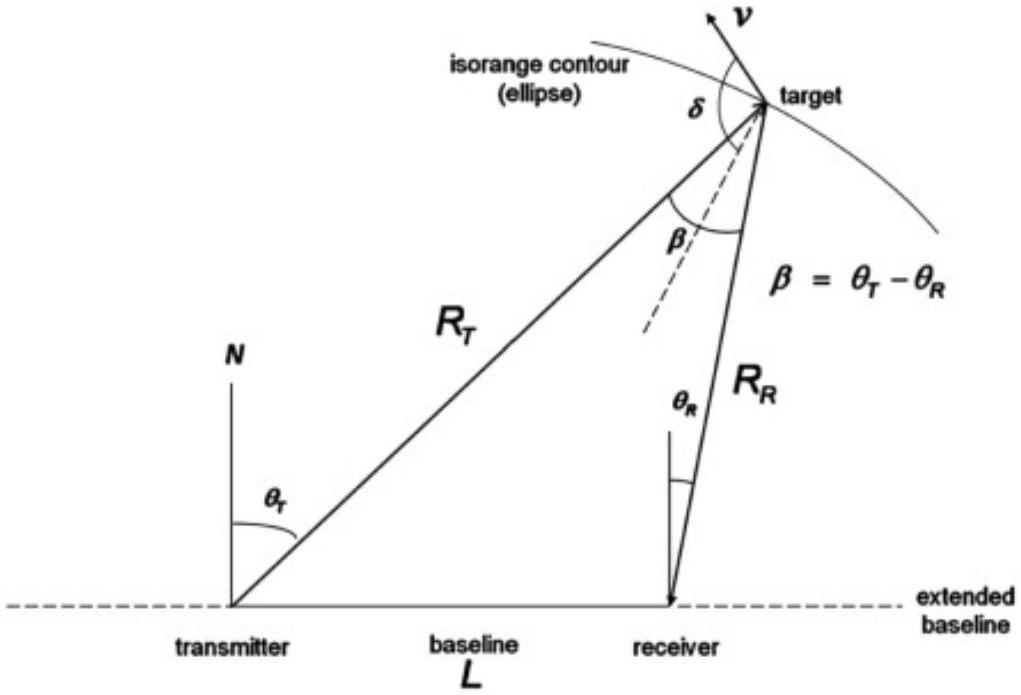


Figure 2.2: Bistatic radar geometry [18]

NEED TO MENTION / RESEARCH THE EFFECT OF NOISE AND DECIBELS EXPECTED HERE.

2.2 Illuminators of Opportunity

The illuminator of opportunity is the signal that is used to illuminate the target, and is the primary source of the signal that is received by the passive radar system. The illuminator of opportunity can be any signal that is transmitted through the air, such as television radio or even satellite broadcasts, and can be tailored to the specific requirements of the passive radar system. Griffiths and Baker outline the three key paramaters when selecting an illuminator [17]:

1. The **Power Density** at the target: It refers to the strength of the signal (in Watts per square meter) that reaches the target area from the illuminator. Higher power density can improve detection performance due to a stronger return signal.

2. The **Nature of the Waveform**: This includes the waveform's properties, such as bandwidth and modulation, which can affect the radar's resolution and ability to distinguish between targets and clutter.

3. The **Coverage**: The spatial area over which the illuminator's signal is spread. Adequate coverage is essential to ensure the target is within the illuminator's effective range.

Illuminator signals are not limited to terrestrial signals, and can also include signals from satellites, and can be tailored to the specific requirements of the passive radar system. The illuminator of opportunity primarily explored for this project is the DAB+ signal, and the target signal will be aerial vehicles - most likely in the form of civillian passenger jets. The DAB+ signal is a good option due to its high power density, and its relatively high bandwidth,

which can be used to improve the radar's resolution and ability to distinguish between targets and clutter. Moreover, the geographical proximity of a DAB+ transmitter at Mt Cootha to the University of Queensland, St Lucia campus, making it a potentially ideal choice for the project. Another prospective digital illuminator signal is DVB-T (digital video broadcast - terrestrial), which is similar in its digital modulation to DAB, but provides increased bandwidth and signal power [12]. Furthemore, as shown by Yin et. al [40], due to the relative complexity of DVB visual signals, more signal processing steps can be required, which could exceed prospective hardware limitations of this project.

A potential problem associated with the use of DAB as an illuminator is direct signal interference (DSI), with the effects being amplified in urban environments. Coleman et. al [7] explain that the sheer signal size of direct illuminator size relative to surveillance signal size results in a high level of DSI. They outline that the cross polarisation of the transmitted DAB signal can be utilised along with illuminator cancellation filtering to attain higher level suppression. The leakage of the illuminator signal into the target signal can be due to a range of factors including buildings, trees and other reflective items as highlighted by Palmer et. al [20].

Typical characteristics of Australian DAB+ signals include frequency of just over 200MHz, bandwidth of approximately 1.5MHz, and a minimal output power of 10kW effective radiated power (ERP), consequently covering a large area [7]. These digital signals employ a modulation scheme called COFDM (coded orthogonal frequency division multiplexing), which is a form of multi-carrier modulation that is robust against multipath interference [17]. COFDM works by dividing the signal into multiple, simultaneous streams which are orthogonal to each other, modulated at a different frequency, maximising robust signal propogation. This is particularly useful in the context of passive radar, as it allows for the target and reference signal to be received by the passive radar system even if it has been reflected off multiple surfaces, such as buildings or trees. More specifically, the Australian implementation of DAB+ is dictated by the ETSI standard EN 300 401, which specifies the modulation, coding and multiplexing of the signal [13]. DAB+ is encoded via the AAC+, with proper licensing required, the most commonly utilised forward error correction(FEC) is 3A [11]. At a high level, these protocols of DAB+ reflect a channel model which has encoding, modulation and noise handling (FEC) built in, which can be advantageous for passive radar systems.

All of the above features result in DAB signals being condusive for ambiguity function performance (analyzed in further detail below). This can mainly be attributed to the relatively wide bandwidth of DAB enabling good resolution, constant DAB envelope stemming from COFDM protocol, and the multipath resistance [19]. The above being said, every reason that DAB is appropriate for passive radar also extends to other digital illuminators of opportunity, such as DVB-T, which is also a COFDM signal, and has a higher bandwidth and power than DAB.

2.3 Range Doppler Mapping

DISCUSS RANGE AND FMAP scope

Range doppler mapping is a technique used to determine the distance and relative velocity of targets by analyzing the frequency shift (Doppler shift) and time delay of the received signals after they are reflected off the targets. The signal response of a target at a particular

range and velocity can be predicted by the ambiguity function seen in equation 2.4 [17].

$$\chi(\tau, f) = \int s_{\text{reference}}(t) s^*_{\text{received}}(t-\tau) e^{j2\pi ft}\, dt \tag{2.4}$$

Where:

- $\chi(\tau, f)$ is the ambiguity function.
- τ is the time delay.
- f is the Doppler frequency.
- $s_1(t)$ is the transmitted signal.
- $s_2(t)$ is the received signal.
- $s^*_2(t-\tau)$ is the complex conjugate of the received signal, time-shifted by τ.
- $e^{j2\pi ft}$ is the complex exponential representing the Doppler shift.
- The integral is taken over all time t.

The ambiguity function can be plotted, thereby visualising resolution, sidelobe patterns and any discrepancies in range and doppler. This is especially important for passive bistatic radar, whereby waveforms are not explicitly designed for radar and the geometry also has an impact [18]. Hughes visualises the geometry considerations and potential flaws of generic passive bistatic configuration below in Figure 2.3.

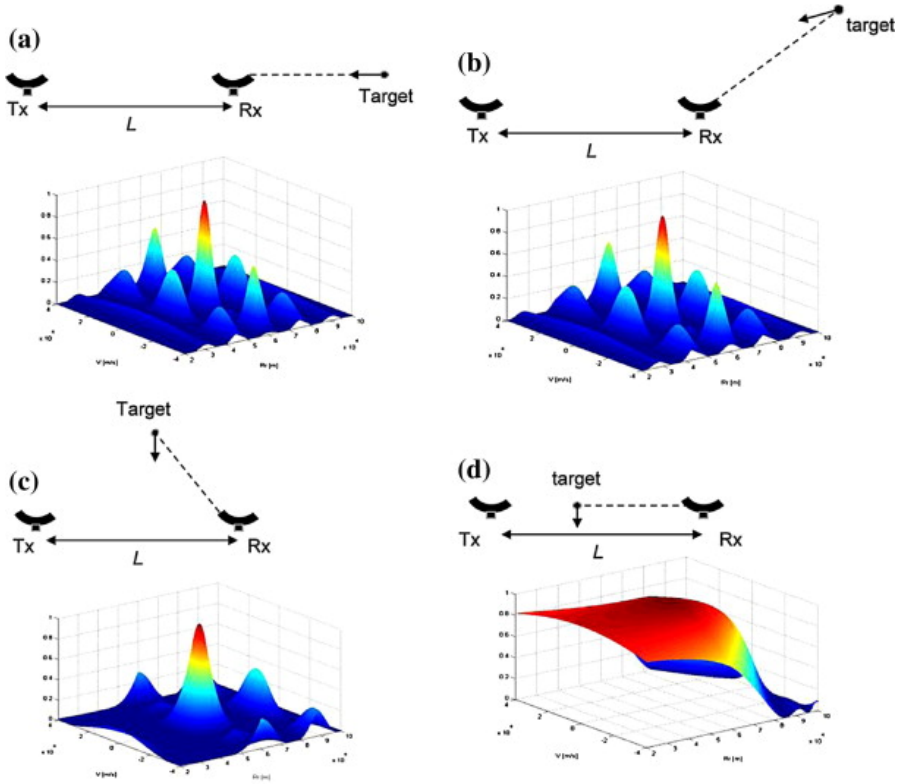


Figure 2.3: Geometry and ambiguity function [18]

Understanding the link between geometric configuration and theoretical signal properties, the practical manifestation of the ambiguity function is range doppler mapping. As shown in Figure 2.4, the map is a heat map and is derived with filters for noise minimsation, allowing for the visualisation of the target signal with minimal clutter.

Should i use a better RDM??

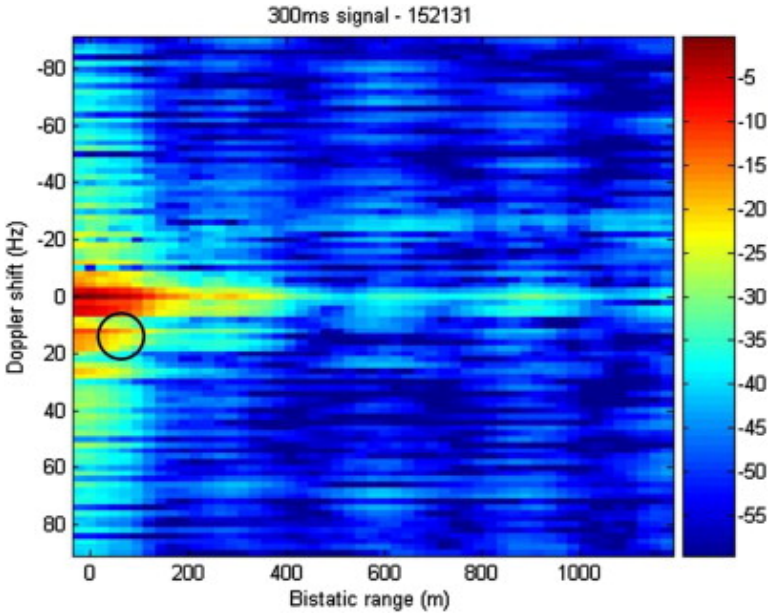


Figure 2.4: Example of range doppler map for WiFi PBR [18]

In the case of this project, the time delay will be utilised to calculate the bistatic range (x axis) and the Doppler shift will be used to calculate the relative velocity of the target (y axis). In summary, ambiguity function and subequent range doppler mapping will be vital for mapping the DAB+ signal characteristics for given geometry and motion of the surveilled target. MORE DETAIL

2.4 Radio Hardware

Fundamentally, the relevant radio hardware for a passive radar project can be broken into the antenna and the SDR module. Given the low cost aims of this testbed, hardware cost and performance is a major consideration.

2.4.1 Software Defined Radio

Luckily, the proliferation of general purpose SDR hardware modules has enabled easy access to radio frequency (RF) signals [33]. In general, a SDR system can exist purely as a receiver (lower cost modules) or as a transceiver, whereby connected software (e.g. PC) are utilised for the modulation and demodulation of signals [15]. The SDR system can be used to sample the RF signal, and then process it digitally, enabling the use of a wide range of software tools for further signal processing. Typically, as seen in the block diagram below in figure 2.5, SDR's comprise of an analogue front end, an analogue to digital converter (ADC) / digital to analagoue convertor (DAC), and some sort of band processing [15].

The most popular and lowest cost SDR module is the RTL-SDR, which is a USB dongle that can be used to receive (note the RTL-SDR is RX only) and decode a wide range of RF signal bands. This includes FM radio, DAB, and DVB-T signals, which are all potential illuminators of opportunity for passive radar systems. Typical bandwidth of the RTL-SDR is 2.4MHz, and the frequency range is 24MHz to 1.7GHz [32].

A plethora of other SDR modules exist, with varying costs and capabilities. A range of common SDR modules are listed in the table below 2.1, along with their key specifications.

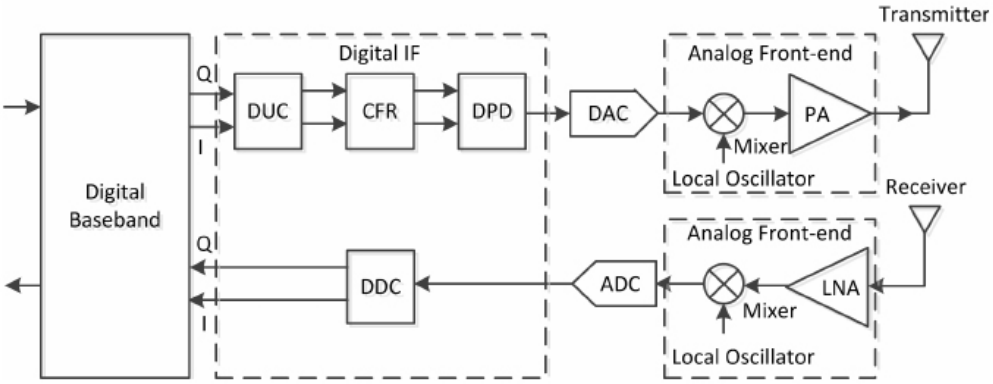


Figure 2.5: Block diagram of SDR transceiver [15]

Table 2.1: Comparison of SDR's [34]

SDR	Frequency Range	Bandwidth	RX ADC	Sampling Rate
USRP	0-6 GHz	Up to 160 MHz	12-14 bits	Up to 200 MS/s
RTL-SDR	24 MHz - 1.75 GHz	Up to 2.4 MHz	8 bits	Up to 3.2 MS/s
bladeRF	47 MHz - 6 GHz	Up to 56 MHz	12 bits	Up to 61.44 MS/s
HackRF	1 MHz - 6 GHz	20 MHz	8 bits	Up to 20 MS/s
LimeSDR	0.1 MHz - 3.8 GHz	61.44 MHz	12 bits	Up to 61.44 MS/s
SDRplay	1 kHz - 2 GHz	Up to 10 MHz	12 bits	Up to 10.66 MS/s
KrakenSDR	24 MHz - 1.76 GHz	Up to 2.5 MHz	8 bits	Up to 2.56 MS/s

As performance increases for the SDR's above so does the unit cost, which is an important consideration for the testbed design explored in scection 4.1. As seen in table 2.1, other than the RTL-SDR all of the SDR's have a large bandwidth making them suitable for digital illuminators, furthemore, the ADC sampling bits are often larger, resulting in more more accurate signals. However, the increased resolution of an ADC also results in larger storage requirements and potentially bottlenecks in the processing chain [34]. Another interesting specification to note which is not seen on 2.1 is the the number of RX channels. The RTL-SDR has one RX channel, whereas the USRP has two RX channels, which can be used to receive two signals simultaneously. This could be useful for the testbed, as it would allow for the simultaneous reception of the illuminator signal and the target signal, which could be used to improve the accuracy of the passive radar system. Specifically, the KrakenSDR features up to five channels, based on clustered RTL R820T2 tuners along with hardware to ensure the synchronisation of the channels [6]. A key advantage of the KrakenSDR in passive radar receiver systems is their ability to achieve angle of direction measurements, which can be used to determine the location of the target as shown by Cass [6].

2.4.2 Antenna

Quite simply defined by the IEEE as *That part of a transmitting or receiving system that is designed to radiate or to receive electromagnetic waves* [1], an antenna is crucial for any radio system, and in the case of passive bistatic radar, receiving all relevant signals. This section will provide a brief background of the key concepts and types of antennas relevant to passive bistatic radar. The ITEE standard definition of terms for antennas [1] also succinctly defines the following terms which are relevant;

1. **Radiation Pattern**: This describes the angular variation in radiation at constant distances from the antenna. The radiation pattern depends on the signal strength and is categorized into three main types: Isotropic, Directional, and Omnidirectional.

2. **Directivity**: Defined as the ratio of the power density in the direction of the pattern's maximum to the average power density at the same distance from the antenna.

3. **Gain**: Closely related to directivity but includes losses. Gain is defined as the directivity minus any losses experienced by the antenna.

4. **Polarisation**: This refers to the nature of the electric field radiated by the antenna. If the polarisation direction is not specified, it is assumed to be in the direction of maximum gain.

5. **Bandwidth**: The range of frequencies over which the antenna operates effectively.

6. **Beamwidth**: The angle between two points on the radiation pattern where the power is at its maximum. The two main types of beamwidth are Half Power Beamwidth (HPBW) and First Null Beamwidth (FNBW).

7. **Impedance**: The ratio of voltage to current at the antenna terminals.

As with considering the SDR module, the cost versus performance trade-off is also a key factor when selecting an antenna. The most common types of antennas used in passive radar systems are the dipole antenna and the Yagi-Uda antenna. The dipole antenna is a simple, low-cost antenna that is easy to construct and has a wide bandwidth. The Yagi-Uda antenna is a more complex, directional antenna that has a higher gain and a narrower bandwidth. Given the utilisation of digital broadcast signals with relatively high power, a simple dipole antenna may be sufficient for the testbed, however, the Yagi-Uda antenna may be required for more complex applications. High gain directional antennas can be chained together to receive signals from all directions, and in a passive radar context, determine the direction of the target [6].

2.5 Digital Signal Processing

Bandwidth RDM and clutter suppression

Broadly, the goal of the signal processing for general passive bistatic radar is to extract a range doppler map from a received signal. Obtaining a range doppler map involves a few steps, with specifics depending on the IOO chosen along with number of sampling channels. Furthermore, there is the option of autocorrelating these signals via correlation integrals which is the most time and compute intense, or through Batches algorithm in the frequency domain . Prior to the advent of of digital broadcasts, analague broadcast processing involved comprehensive filtering and synchronisation of at least two input channels[25]. However, more recently, given the nature of DAB / DVB-T and its COFDM modulation, filtering and synchronisation can be replaced with reconstruction of the surveillance signal [4], replacing the need for dual channel configuration. Fundamentally, the single channel version of this correlation is the direct signal (given its strength) compared with a time lagged version of itself given the refelection from the target, as reflected in 2.4. Mahfoudia, et al. demonstrate the methodology behind single channel correlation and reconstruction with a DVB-T illuminator, combined with clutter suppression techniques [23].

batches citation needed

The simplest, yet probably most computationally intense method of processing the doppler map is via traditional FFT-based correlation. It works by correlating large portions of the signal in time and frequency domains, with doppler shifts applied, creating frequency bins which are used to correlate with the received signal in the frequency domain, thereby obtaining any doppler shift experienced by the target. As for the time domain, the range estimates of a target can be obtained by taking the inverse fast fourier transform (IFFT) of the correlated signal in the frequency domain, providing the relevant time delay (and therefore range). The process can be repeated over multiple windows of a given signal and then combined to create a range doppler map [21].

The computationally intense FFT correlation method can be refined and optimised by taking slow time 'batches', reducing the size of required FFT calculations. Once the signal has been received and windowed, according to Moser et. al, a FFT the size of 2048 for the range domain and 512 for the doppler domain can be computed [26], see Figure 2.6.

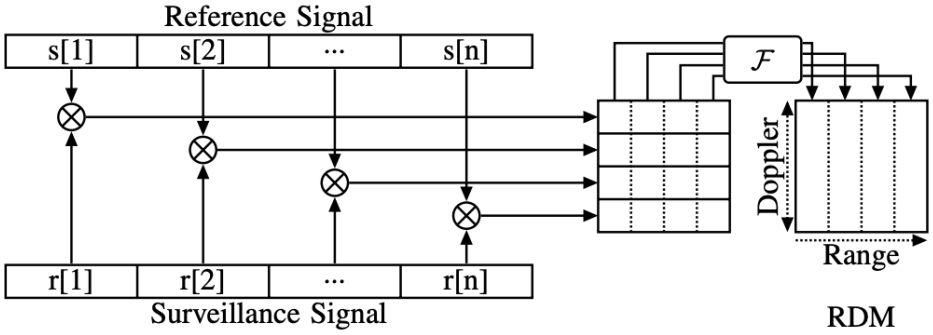


Figure 2.6: Correlation and FFT for Range Doppler Mapping [26]

Given that single channel configurations are vulnerable to noise, interference and multi-path, the outputs of pure correlation can result in clutter on the range doppler map. In the case of single channel configuration for PBR, the signl from the target (already quite weak) can be obstructed by unwanted reflections from buildings, trees, and other objects [17]. The above algorithms can be combined with a decluttering chain, which can implement something along the lines of a weiner filter, or a matched filter, to remove unwanted signals from the range doppler map [18]. The implementation of clutter suppression algorithms, specifically background mean subtrafction (BMS) and guard mean substraction (GMS) has been shown by Zhang et al, working to clear up range doppler maps [41]. This clutter suppression is especially neccessary around low doppler shift targets, as the clutter can be mistaken for the target signal. The presence of clutter given lack of suppression can be seen in Figure 2.7 below (yellow higher intensity pixels).
Logically, the process above relfects the most computationally intensive step of the project ... hence, considerations with regard to signal processing algorithms will need to be taken in order to keep the overall detection system low cost, despite it not being a major factor of the testbed creation itself.

2.6 IoT Architecture

The IoT architecture in the scope of this project refers to the computational platforms utilised to undertake the digital signal processing which then maps to tracking and detection of the

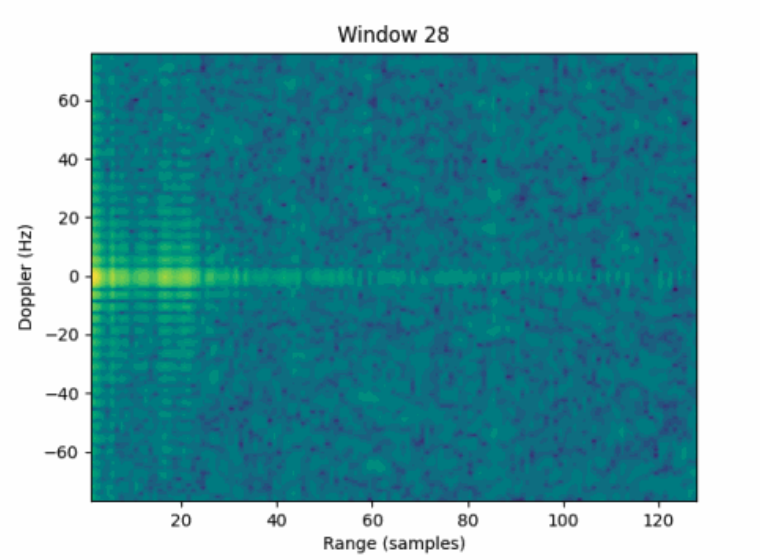


Figure 2.7: Clutter suppression (lack of) in range doppler map, taken from testing area with no target

target. At a high level, ITEE defines the Internet of Things (IoT) as *"as a world of interconnected things that are capable of sensing, actuating, and communicating among themselves and with the environment ... provides the ability to share information and autonomously respond to real/physical world events by triggering processes and creating services with or without direct human intervention"* [2]. In terms of PBR, existing studies have demonstrated the ability of off the shelf laptops [36], and there is a few studies that use IoT platforms for FM signal processing [26]. The vital consideration when exploring hardware are the DSP and sampling requirements for a given illuminator signal and antenna configuration. More broadly, IoT hardware has been used in a range of general tracking and detection applications. A prime example of this is the Flightradar24 service which utilises a network of ADS-B receivers to track aircraft in real time, where they provide a hardware kit of a Raspberry Pi and a DVB-T dongle (RTL-SDR) [14], channelling the data to a central server for processing and mapping.

According to Schupbach et. al [35], the broad scope capability requirements of the computational platform and associated processing can be grouped into signal acquisition, signal reconstruction and then the correlation function calculation. With a single board computer, this can be largely achieved through a combination of software libraries along with python script, however the hardware facilitating this processing is a key consideration. Largely, the IoT hardware is limited to single board computers, as microcontroller units (MCU) are not capable of the sampling and detection processing requirements, along with the fact that the project requires a full operating system for relevant SDR libraries. Exploring other potential architecture such as FPGA's, again are not as well suited, given their high cost and complexity, although their is precedent for the signal processing and recording [24]. A range of single board computers are available, and can be compared primarily based on their processing power, memory / memory access, and cost [39], as viewed in the table below.

There is precedent of previous use for the Raspberry Pi platform as demonstrated by Moser et. al [26] An alternate, higher powered choice in which Sednall demonstrated is the higher powered Nvidia Jetson, which includes faster processing times due to its quad core architecture [36]. The price for these options is $60 compared to $250 respectively. Ultimately, the choice

SBC	CPU	RAM	Storage Access	OS Support	Size (mm)	Price (USD)
Raspberry Pi Zero 2 W	Quad-core Cortex-A53, 1 GHz	1GB	microSD	Raspberry Pi OS, Linux	65 x 30	15
Raspberry Pi 4	Quad-core Cortex-A72, 1.5 GHz	2/4/8GB	microSD, USB	Raspberry Pi OS, Linux	85.6 x 56.5	35-75
Raspberry Pi 5	Quad-core Cortex-A76, 2.4 GHz	8GB	microSD, PCIe (through HAT), USB	Raspberry Pi OS, Linux	85.6 x 56.5	from 60
Nvidia Jetson Nano	Quad-core Cortex-A57, 1.43 GHz	4GB	microSD, PCIe (through adapter)	Ubuntu, JetPack	100 x 80	100
Orange Pi 5	Octa-core Cortex-A76/A55, 2.0 GHz	8GB	eMMC, microSD, PCIe	Android, Linux	90 x 60	80

Table 2.2: Comparison of Single Board Computers [39]

of embedded IoT hardware for the project can be reduced to the trade off between low cost and processing power, and will be a key consideration in the project plan. Furthermore, there is possibility to design custom hardware or utilise the GPIO functionality of the single board computer, as reflected by the high level definition of IoT architecture by ITEE [2].

2.7 Networking with Embedded Hardware

Given that the overall goal of this thesis project is to create a low cost, embedded passive radar detection system, valuable to consider how this would be visualised, be it via wired or wireless protocol. Essentially, depending on whether the RDM is computed on the 'edge' (ie embedded device) or on a PC, the networking requirements will differ. For example, if it is only IQ sampling completed on the embedded testbed platform, only the IQ data in the form of a .bin file needs to be transferred, however, if the RDM is computed on the embedded device, the entire RDM image / gif will need to be transferred. The following options vary in complexity and context, but all facilitate the relavant transfer of data from the embedded device to a PC for further processing or visualisation.

2.7.1 USB Drive Transfer

The most basic method of transferring data from an embedded device to a PC is via a USB drive. This method is simple and reliable, and is ideal for applications where the data transfer rate is not critical. The data is saved to a USB drive on the embedded device, and then transferred to a PC for further processing. However, it is not ideal for real-time applications, as the data transfer is limited by proximity and a manual process is required (despite large data storage capabilities).

2.7.2 SSH

Secure Shell (SSH) is a widely-used method for remotely interfacing with embedded devices and SBC's. It allows users to securely log in to a remote device over a network and issue commands as though they were operating directly on the device. This is particularly useful for headless systems, where the embedded device does not have a monitor or keyboard connected. SSH encrypts all communication, ensuring data security. Once an SSH connection is established between the Raspberry Pi and a PC, tasks such as starting data capture, configuring parameters, or transferring files can be done directly from the PC terminal [16]. Many applications use SSH for remote monitoring and control of embedded systems, as well as for viewing the UI of SBC's, for example VNC viewer.

2.7.3 Ethernet

Ethernet is one of the most common and reliable methods for data transfer between embedded systems and PCs, offering high data transfer speeds and low latency. Ethernet is a wired networking technology that forms the backbone of local area networks (LANs). It supports high-speed data transmission, typically at 100 Mbps (Fast Ethernet) or 1 Gbps (Gigabit Ethernet) [28], making it a suitable for applications that require real-time data transfer or when working with large datasets, such as IQ samples or RDM files. Ethernet uses a network cable (e.g., CAT5, CAT6) to connect devices in a LAN. It transmits data packets using the Internet Protocol (IP), allowing connected devices to communicate with one another. The data packets are routed through switches and routers, which direct the flow of traffic within the network. Ethernet is widely used in industrial automation, IoT, and other applications that require reliable and high-speed data transfer.

2.7.4 TCP Socket

A TCP (Transmission Control Protocol) socket is a software-based method for creating a reliable connection between two devices over a network. TCP ensures that all data packets are transmitted in the correct order and without loss, making it ideal for real-time data streaming applications such as radar data. In the case of this project the testbed would act as the client, initiating data transfer to the server (PC). This can be implemented over a wireless network [9].

2.7.5 Cloud Service

Another modern solution for data transfer between embedded devices and PCs is the use of cloud services. The embedded testbed can upload data to cloud storage, which can then be accessed from the MacBook Air, or vice versa. An obvious downside of this method is the reliance on an internet connection, and the potential for data security issues.

Chapter 3

Literature Review

This chapter will explore and summarise previous work on the topic of passive bistatic radar detection and associated hardware, ranging from academic literature to currently available commercial technology. The literature review will be divided into the following sections: low cost IoT hardware receiver, digital signal based illuminator of opportunity and existing commercial technology.

3.1 Low Cost IoT Hardware PBR Reciver Design

The Swiss army have conduncted a very informative pilot study utilising both FM radio and DAB+ signals as illuminators of opportunity for an IoT based receiver design. Moser et. al explore the performance of a Raspberry PI 3 GPU and CPU against a quad-core intel i7 equipped PC [26] . This paper also provided a good starting point for information regarding the difference in signal processing between digital (DAB+) and analogue (FM) illuminators of opportunity, including key characteristics as seen in Figure 3.1 .

Include their results too

	FM	DAB
Frequency	88 - 108 MHz	174 - 230 MHz
Modulation	analogue (FM)	digital (OFDM/DQPSK)
Bandwidth	0.05 MHz	1.536 MHz
Availability	Global	Local
Network	multi-frequency	single-frequency
Content dependency	yes	no

Figure 3.1: Comparison Table - Analog vs Digital Signals [26]

Moser et. al also provide a summary of the range doppler map generation process which itself is derived from Batches algorithm [25], including valuable performance measurements which indicate the efficacy of GPU processing over CPU processing. The specific hardware utilised in Moser's paper was a Raspberry Pi v3; 1.2GHz Cortex A53 CPU along with a 400MHz VideoCore IV GPU. As of writing, the estimated cost to recreate this exact hardware (including SDR-RTL hardware) would be approximately $100 AUD, representing a very low cost setup. Noting that hardware has also made significant advancements since the time of the paper (2019), with the Raspberry Pi v5 now available. The paper provides good hardware reference values for real-time processing limits of DAB frame computations, a key consideration area when optimizing for low cost hardware. Overall, this conference paper provides a good starting point for understanding the hardware requirements and limitations of a low cost IoT based passive bistatic radar receiver. Overall, this conference paper proides a valuable

benchmark and proof of concept for the capabilities of low cost IoT hardware (circa 2019) in PBR applications, highlighting that the Raspberry Pi 3 is capable of real-time processing of DAB+ signals. Notably this paper is difference from the thesis proposed given that it focuses on implementing real time processing and optimisation of detection DSP algorithms on low cost hardware, rather than the design and construction of a user friendly, scaleable testbed.

3.2 Illuminators of Opportunity

Throughout the literature review, a range of papers exploring illuminator signals were explored, the DTSO report Written by Palmer, Palumbo, Van Cao, and Howard provides a good theoretical basis. Whilst the scope of this proposal is confined to terrestrial illuminators (specifically digital audio), Palmer et. al highlight the wide range of use cases made available by other illuminators, including satellite signals and mobile phone signals. The report also provides a good overview of range doppler mapping, and target classification properties (despite the scope of this proposal not including target classification).

more comprehensive overview

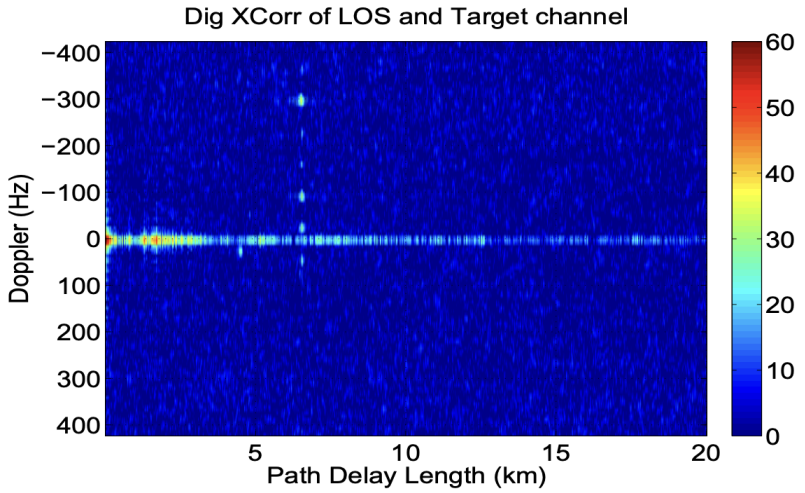


Figure 3.2: RDM of target moving away from receiver [20]

As seen above in Figure 3.2, in the .gif version of the image, the dot can be seen moving upwards (indicating motion away). This example also reflects effective de-cluttering of the RDM, a key step in passive tracking. An important consideration raised in this report is the edge case whereby the target moves along the bistatic ellipse as viewable in Figure 2.1, resulting in the doppler shift to be minimal and making tracking difficult.

3.3 Existing Commercial Technology

Commercial industry has been developing passive radar for a number of years now, with the majority of such development occuring in the military space. Specifically,these innovations are in the realm of air sureveillance and can be either ground based or air based [18]. Structurally, passive radar provides numerous advantages for defence applications including covert tracking due to the lack of a transmitter.

Lockheed Martin - SilentSentry: One of the earliest commercial products utilising passive radar, the SilentSentry utilises analog illuminator signals with a dual horizontal linear phased array antenna to passively detect and track airborne targets [20]. It claims a detection range of up to 200km with an azimuth coverage of 60-360 degrees. Given the analog nature of the signals (FM and other terrestrial signals), it has a receiver for the direct line of sight signal and the echoed target signal before applying a signal processing algorithm.

Chapter 4

Methodology and Design

This chapter contains the design decision and steps taken to complete the testbed creation and testing. The project can broken into the three main areas of hardware, software and digital signal processing.

4.1 Hardware

4.1.1 Software Defined Radio

The fundamental hardware aspect for this project is the Software Defined Radio (SDR), specifically, the SDR receiver module. As mentioned in section 2,4, software defined radio technology has been recently experienced decreasing costs and proliferation [33]. Broadly, the testbed was designed to accomodate a potential range of USB-A capable SDR modules, with the capability initially tested on a low cost and qualirt RTL-SDR to use as a benchmark, before progressing to later prototyping on higher end LimeSDR.

ADD SECTION LABELS

(i) RTL-SDR The low cost RTL-SDR was chosen as the initial SDR module for testbed prototyping due to its low cost and wide availability. The specifics of the RTL-SDR are stated in section 2.4 , it was connected to the RPi5 and higher level M1 Mac along with a simple SMI antenna as seen in the figure below 4.1.

REF

Figure 4.1: RTL-SDR with Simple SMA Antenna

The RTL-SDR is based on the Realtek RTL2832U chipset, and has a frequency range of 24MHz to 1.7GHz, and a bandwidth of 3.2MHz. The RTL-SDR is also relatively cheap, with a price of around $40 AUD, coming with compatibility to a wide range of software, including

MATLAB, and GNU radio [32]. More specifically, the RTL-SDR was originally designed as a DVB-T receiver for digital TV, but due to its versatility, it has been repurposed by the hobbyist community for a wide range of RF signal reception applications. Fundamentally, the RTL-SDR is compised of two IC chips as seen in the block diagram figure below 4.2; the RTL2832U, which is the digital TV demodulator, and the R820T, which is the tuner. The R820T is the chip that allows the RTL-SDR to tune to a wide range of frequencies, converting those signal frequencies to an intermediate baseband frequency which is processed by the RTL2832U. Inside the RTL2832U, the signal is then digitized and demodulated and sent to the host computer via USB.

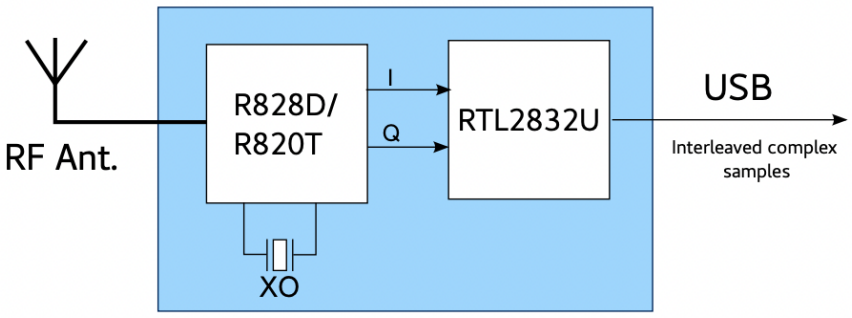


Figure 4.2: RTL-SDR Block Diagram [5]

In order to test the functionality and performance of the RTL-SDR, it was necessary to perform the following steps in isolation before integrating into the testbeed, which are reflected in the results section:

- **Software Installation:** Install the RTL-SDR drivers and software on the relevant host computers (RPi5 and M1 Mac).
- **Signal Reception:** Tune the RTL-SDR to the appropriate frequency range (approx 200MHz) and receive a digital broadcast signal (with yagi-uda antenna or simple SMA monopole).
- **Signal Processing:** Process the received signal values using python to ensure the values are not junk.
- **Noise Floor Measurement:** Measure the noise floor of the received signal to account for the signal-to-noise ratio (SNR) of the system (given test location).

(ii) LimeSDR The LimeSDR was chosen as the higher end SDR module for the implementation and testing of the testbed. As seen in Table 2.1, the LimeSDR has a frequency range of 100kHz to 3.8GHz, and a bandwidth of 61.44MHz. The LimeSDR is also more expensive than the RTL-SDR, with a price of around $600 AUD. Specifically, the LimeSDR used was the LimeSDR-USB Version 1.4, which can be seen in the figure below 4.3.

The actual SDR module was encased as seen in XXX , with the only relevant conenctions being a singular RX SMA port, power supply (6V DC) and a USB type B connection. The block diagram of the LimeSDR can be seen in the figure below 4.4, and evidently it is a more complex system than the RTL-SDR, most notably with the CycloneIV FPGA chip, which is used for signal processing and control.

add photo in results

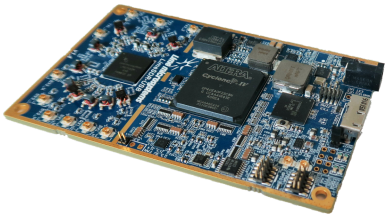


Figure 4.3: LimeSDR-USB Version 1.4 PCB [27]

In order to validate the standalone functionality of the LimeSDR, the following steps were taken, and discussed in the results section:

- **Software Installation:** Install the LimeSDR drivers and software, the software required is in section 4.2.1.
- **Signal Reception:** Tune the LimeSDR to the appropriate frequency range (approx 200MHz) and receive a digital broadcast signal (with yagi-uda antenna).
- **Signal Processing:** Process the received signal values using python to ensure the values are not junk.
- **Noise Floor Measurement:** Measure the noise floor for given location.

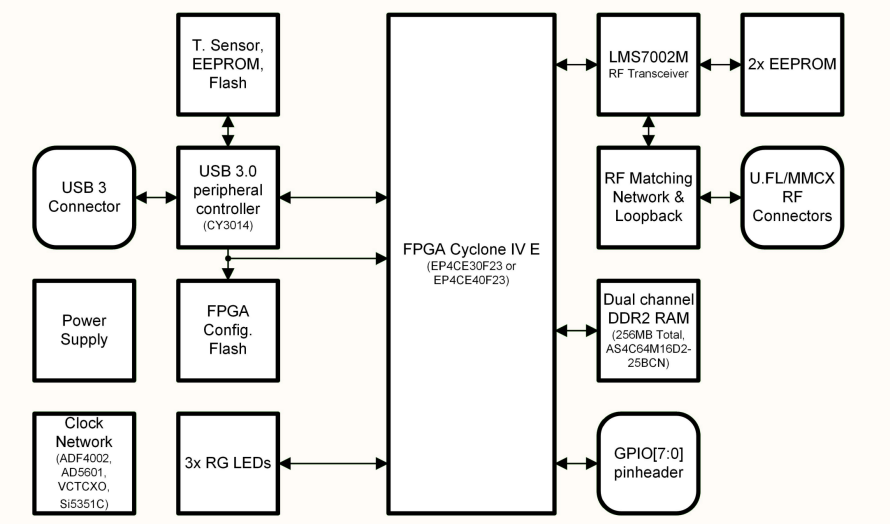


Figure 4.4: LimeSDR Block Diagram [27]

Another notable hardware feature of the LimeSDR is the LMS7002M transceiver chip which can facilitate both RX and TX capabilities, along with built in ADC/DAC (12 bit) capability [27]. The bandwidth advantage of the LimeSDR was a key factor in its selection, as the wider bandwidth allows for more data to be captured and processed, specifically in the context of the digital broadcast signal used as the illuminator of opportunity. This bandwidth advantage directly results in increased range resolution on the range doppler maps, which is a key metric for the detection of aerial vehicles.

Discuss USB 3.0 bottleneck, until X bandwidth samples/sec

4.1.2 Embedded Computing Platform

The embedded computing platform chosen for the testbed was the Raspberry Pi 5 (RPi5), specifically the version with 8GB of RAM [8].

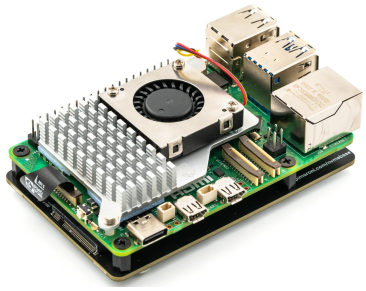


Figure 4.5: Raspberry Pi 5 with NVME SSD [29]

As seen in Table 2.2, the RPi5 has a quad-core ARM Cortex-A72 CPU, 8GB of LPDDR4-3200 SDRAM, and a 1Gbit Ethernet port. The RPi5 was chosen for its relatively low cost, small form factor, and wide range of software support. Other SBC's from Table 2.2 were considered, but the Rpi5 represented the best balance of performance and cost for the project, for example, the higher cost of the Nvidia Jetson was not justified for the project requirements. The viability of the Raspberry Pi5 in the context of testbed design and functionality was analysed based on the following points:

Advantages

- **Storage:** The RPi5 supports external storage options via microSD, along with configurable HATs (hardware attached on top) for NVME SSDs, see section 4.1.3 [29].
- **Size:** Its small form factor (85.6 x 56.5 mm) - roughly the size of a credit card, makes it easy to integrate into a testbed setup. Easily obtainable cooling fan.
- **GPIO:** 40 Pin GPIO header, can be used for interfacing with a prospective testbed user. Driven by the RP1 (custom I/O controller chip) [8], the I/O is programmable with python.
- **Networking Capability:** Includes a 1Gbit Ethernet port and Wi-Fi 802.11ac, facilitating high-speed data transfer and network communication. Capable of 217.6 MB/s over Wi-Fi [28].
- **Processing Capability:** Equipped with a quad-core ARM Cortex-A72 CPU and 8GB of LPDDR4-3200 RAM, enabling 2.4 GHz clock [8].
- **Cost-Effectiveness:** Price of $150 AUD, making it an affordable option for the project, approximately 1.5 the cost of the RPi4 (however almost double the performance) [8]
- **Software Support:** Extensive community support and availability of a wide range of software tools and libraries, especially for Linux-based systems, e.g. RTL-SDR, GNU Radio, etc. see section 4.2.1.

Disadvantages

Add to this as project progresses

- **Limited Built-In Storage:** The built-in storage is minimal, so additional external storage is necessary for handling large volumes of data.
- **Performance Constraints:** While powerful, it may not handle very high-frequency or complex radar signal processing as efficiently as more specialized or high-performance computing platforms, custom algorithms potentially required [26].

- **Heat Management:** Under continuous heavy load, the RPi5 may require additional cooling solutions to prevent overheating.
- **Limited I/O Bandwidth:** The available I/O bandwidth might be a bottleneck for applications requiring very high-speed data acquisition and processing.
- **Power Supply:** Requires a stable power supply of 5.1V / 15W; not ideal for potentially portable or battery-powered applications [28].

4.1.3 NVME Based Storage

Following on from the selection and testing of the Raspberry Pi 5 as the testbed computing platform, it became evident that the default 32GB microSD card used to boot and run the operating system was insufficient for the storage requirements of the project. The chosen solution for the bottleneck was to utilise a NVME SSD drive, connected via PCIe to the Raspberry Pi 5, specifically the Pimoroni Base [29]. It was necessary to test the efficacy of the NVME SSD in the context of the testbed, with the following steps take and discussed in the results section:

- **Hardware Installation:** Screw the NVME SSD drive into the Pimoroni Base, and connect the Base to the Raspberry Pi 5 via the PCIe connector.
- **Software Configuration:** Format the NVME SSD drive with operating system.
- **Performance Testing:** Test the read and write performance of the NVME SSD drive using the `hdparm` utility.

Further NVME SSD configuration details include formatting with the ext4 file system, which is the default for most Linux distributions. It was mounted to the Raspberry Pi 5 (RPi5) at the location /mnt/nvme0n1. The SSD connects via a PCIe x4 interface Gen 2.0, a high-speed standard commonly used for storage devices. It supports M.2 NVMe drives of the 2280 size. The specific model used is the Patriot P300 NVMe M.2 SSD with a 128GB capacity, serving both as storage and for booting the RPi5.

4.1.4 Antenna Configuration

Intially, a simple SMA whip antenna was used for the RTL-SDR, as seen in Figure 4.1. These low cost antennas are typically used for general purpose reception, and are not optimised for any specific frequency range. The whip antenna was used for initial testing and prototyping, and was capable of receiving a range of broadcast signals with a NOISE FLOOR X. Eventually a more specialised antenna was used, the Yagi-Uda antenna, which is a directional antenna that is commonly used for point-to-point communication. The Yagi-Uda antenna was used for the final testing and validation of the testbed. The Yagi-Uda was utilised with a CLF200 50 Ohm coaxial cable, which was connected to the RTL-SDR.

The physical setup of these antennas for testing comprised of an "Over the Shoulder" geometry whereby the antenna was placed behind a building in the line of sight of the TX signal (in this case Mt Cootha). For a single channel / antenna PBR setup, this geometry works to attenuate the direct path signal, and allow for the reflection to be better received by the antenna.

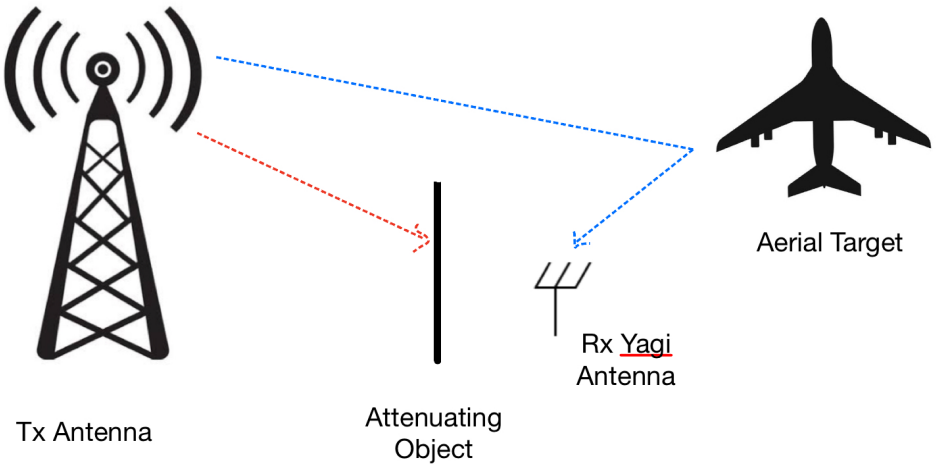


Figure 4.6: Geometry of Over the Shoulder Antenna Placement

Antenna Verification and Testing Requirements

It was necessary to have a range of hardware verification tests to ensure the antenna was functioning correctly and to quantify the signal strength of the illuminator signal. The following steps were taken to test the signal strength of the illuminator signal, and are discussed in the results section:

- **Signal Strength Measurement:** Measure the signal strength of the illuminator signal using the RTL-SDR and the SMA whip antenna.
- **Signal Strength Comparison:** Compare the signal strength of the illuminator signal using the RTL-SDR and the Yagi-Uda antenna.
- **Compare a Range of Test Locations:** Given the geography of Mt Cootha and the Brisbane air traffic fligh path (INSERT RELEVANT GEOGRAPHY MAP), it was necessary to test the signal strength at a range of locations to ensure the illuminator signal was being received, and compare any contributable noise factors.

4.1.5 Testbed Design

Building off the aforementioned hardware components, it was then necessary to develop a physical testbed setup that would allow for the SDR receiver module to be connected to the embedded computing platform, and for the entire system to be powered and networked. Also, given the scope of the project, it was necessary to have a protective encasing for the Rpi5 which also implemented user functionality (ie. pushbuttons and LED's). The testbed was designed to be portable and easily deployable, with the following key components and features:

ADD IMAGE OF TESTBED

- **Enclosure:** A custom 3D printed enclosure was designed to house the RPi5, the RTL-SDR, and the NVME SSD HAT seen in 4.5. The enclosure was designed to be compact and portable, with a small form factor that could be easily transported and deployed in various environments. The enclosure was designed to be easily opened and closed, with a removable top panel that allowed for easy access to the internal components. The

printed acrylic was also designed for cooling purposes, with ventilation holes on the top and sides of the enclosure to allow for airflow, bolstered by the Rpi5 active cooling fan and heatsink. The model from Rasmussen was identified as a base model, with potential for appropriate modification [31]. The base model can be viewed in the image below .

SHOW BASE MODEL

- **User Interface:** It was envisaged that the testbed have some sort of capability that would enable physical triggering of PBR sampling and processing. Arcade style push-buttons with LED's were selected to facilitate this interface [3]. It was envisaged that one pushbutton be responsible for signal sampling and the other a trigger for embedded edge processing.

- **Power Supply:** A 5V / 15W power supply was used to power the RPi5, with a USB-C connector that allowed for easy connection to the RPi5.

- **Networking:** The ethernet port on the Rpi5 was left unobstructed by the case for potential wired networking. With the main focus of being geared towards Wi-Fi based networking which was all facilitated by the Rpi5 system on chip.

- **Radio Front End:** It was intended for the SDR module to be connected via the USB 3.0 port with the enclosure not designed to house the SDR unit (for versatility). Per SDR functionality, the antenna was connected via coaxial cable, also not housed in the enclosure.

In summary, the testbed was designed to be a portable, easily deployable system that could be used to test and validate the passive radar detection system. The enclosure was designed to house the RPi5, the NVME HAT, with the remaining system hardware components not specific to the testbed itself (possible to change out). This part of the project design was important for achieving the primary objectives, notably objective 2 as stated in 1.2. Details of the actual build process and testing of the enclosure are discussed in 5.1.6.

4.2 Software

4.2.1 SDR Software

When utilising SDR modules, it is necessary to have capabilities to perform necessary processing on the sampeld data. This is typically done using SDR 'driver' software, which is a type of software that allows for the reception and baseband processing of radio signals using SDR hardware (ie. setting the tuner to a given frequency). Both the SDR modules utilised communicated with the host computer via USB 3.0, and utilise the libusb library for this. In the case of the selected hardware for this project, it was necessary to utilise relevant driver API's to interface with the RTL-SDR and LimeSDR, both from the testbed and the higher level M1 Mac, as shown below.

(i) RTL-SDR Given the widespread use of RTL-SDR dongles, there is a range of useable software stacks and drivers available. The interface software utilised for testing and visualising the RTL-SDR efficacy was GQRX, which is a software defined radio receiver powered by GNU Radio and the Qt GUI toolkit. GQRX is easy to use and provides a range of features for signal processing and analysis [37]. The software was installed on the RPi5 and the M1 Mac, and was used to receive and process the digital broadcast signal, also helping test the antenna. Whilst GQRX is used as a front end visualisation tool, it essentially wraps the **librtlsdr** library

[22], which is the actual API driver for the RTL-SDR (specifically the RTL2832U chip [38]). The librtlsdr library handles hardware communication, tuning, and data acquisition (gain, sampling, etc) [22], and consequently outputs 8-bit I/Q samples to the host computer. This library was tested on both the RPi5 and the M1 Mac, as seen in the results section.

(ii) LimeSDR The LimeSDR requires a more complex software stack than the RTL-SDR, given its higher performance and comparitively better hardware. As mentioned in section 4.1.1 and specifically seen in the LimeSDR block diagram 4.4, the LimeSDR has a custom LMS7002M transceiver chip, along with other programmable components. However, given the relatively smaller user numbers of the LimeSDR it was more difficult to interface (given lack of documentation). The following software was used to interface with the LimeSDR, the connection between these APIs can be viewed in the software block diagram below 4.7:

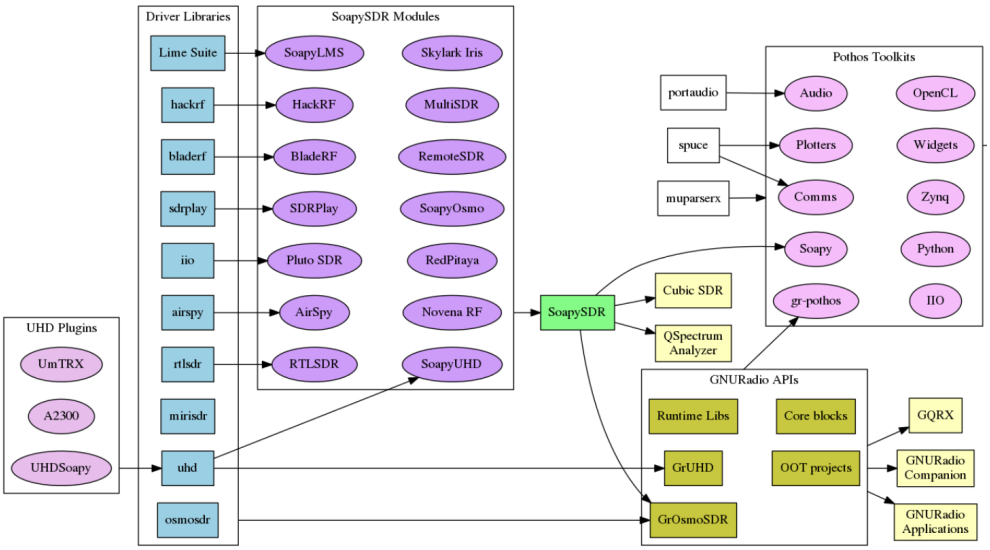


Figure 4.7: LimeSDR Software API Block Diagram

- **LimeSuite:** LimeSuite is the software that is used to interface with the LimeSDR hardware, and is used to configure the LimeSDR, set the frequency, gain, and bandwidth, and perform other functions [27].
- **SoapySDR:** SoapySDR [30] is a software abstraction layer that provides a common API for interfacing with a wide range of SDR hardware, including the LimeSDR. SoapySDR provides a simple and consistent API for interfacing with SDR hardware, and is used to communicate with the LimeSDR hardware from the host computer [**?**]. It essentially acts to wrap and unify the LimeSuite and LimeSDR hardware, providing a common interface for the host computer (ideal for porting from testbed to high level mac)
- **Python:** Python was used as an abstract interface, essentially calling the SoapySDR API, again, easily transferable between host computers.

In summary, the above software was selected to ensure portability between both the testbed and the higher level M1 Mac, whilst also accounting for potential SDR hardware changes. The individual component testing comprised of sampling DAB 9A, ensuring correct file parsing and basic noise floor readings.

4.2.2 Networking Requirements

Given the abundance of options explored in the background section, initially, the Raspberry Pi was connected to via SSH and VNC viewer over WiFi for configuration and testing. Eventually TCP baswed client-server communication was used to facilitate the transfer of data between the RPi5 and the M1 Mac. The TCP server was implemented in Python on the M1 Mac, and the TCP client was implemented in Python on the M1 Mac. The TCP client was responsible for sending the sampled data to the TCP client, which was responsible for receiving the data and processing it. This was implemented via the `socket` library in Python, and was tested on the RPi5 and the M1 Mac, with the results discussed in the results section.

4.3 Relevant Digital Signal Processing

Once the first phase of the testbed verification was completed, the next step was to implement the digital signal processing (DSP) algorithms that would be used to process the received signals and detect the presence of aerial vehicles, and then compare the results between the RTL-SDR and the LimeSDR, along with edge compute times (on the Rpi5) and higher level mac processing. Whilst both SDR implementations used the same fundamental DSP algorithms (FFT and autocorrelate), there were some system factors that were impacted by the comparison.

4.3.1 Sampling Rate and Bandwidth

Fundamentally, the LimeSDR supports much higher sampling rates and bandwidths than the RTL-SDR, with the LimeSDR having a maximum sampling rate of 61.44MHz and a bandwidth of 61.44MHz, compared to the RTL-SDR which has a maximum sampling rate of 3.2MHz and a bandwidth of 3.2MHz. This difference in sampling rate and bandwidth has a direct impact on the range resolution of the radar system, with the LimeSDR being able to achieve much higher range resolution than the RTL-SDR. The sampling rate and bandwidth of the SDR modules were tested and compared, with the results discussed in the results section.

4.3.2 Method for Testing and Comparing LimeSDR and RTL-SDR

Both SDRs were connected to the RPi5 testbed and were used to receive the same digital broadcast signal. The received signals were then processed using the same DSP algorithms, and the results were compared. The following steps were taken to test and compare the LimeSDR and RTL-SDR:

Chapter 5

Results and Discussion

5.1 Testbed Verification

Before the testbed was used to facilitate comparison between different SDR modules detection performance, its design and components were tested according to the steps in the methodology section. Whilst this was relatively straightforward, it was necessary to ensure the project could meet its aims of a simple, user friendly, potentially scaleable design.

5.1.1 SDR Module Verification

The RTL-SDR and LimeSDR Mini were tested to ensure they could be used in the testbed. This comprised of testing both the software driver functionality as mentioned in Section **??** and the hardware / communication functionality.

RTL-SDR The RTL-SDR was tested using a combination of librtlsdr command line tools and GQRX, both on the mac and the Rpi5. Starting with the below command to ensure the hardware was detectable:

```
rtl_test -t
```

In conjunction with the antenna testing below 5.1.3, the RTL-SDR waterfall plot was obtained via GQRX, as seen in Figure 5.1.

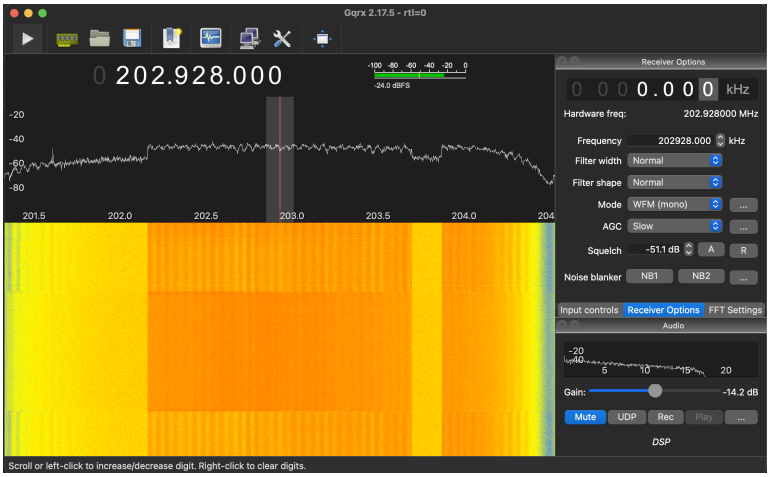


Figure 5.1: RTL-SDR Waterfall Plot (DAB Multiplex 9A)

Given that the relevant data processing was to be conducted with a python script, it was also neccessary to test the ability of the RTL-SDR to sample and output data to a .bin file. This was done using the following command:

```
rtl_sdr -s 2048000 -f 202.928e6 -n 2048000 testData.bin
```

The above test was successful in saving raw IQ data to a .bin file, in the executed directory. Where the saple rate was 2.048 MS/s, the centre frequency was 202.928 MHz (DAB Multiplex 9A) and the number of samples was 2048000 (1 second of data). The *testData.bin* file size for the above example was 4.1MB, highlighting the importance of the storage medium as the sample time is scaled up, as discussed in Section 5.1.2.

LimeSDR As with the RTL-SDR, the LimeSDR hardware and software was verified, however it required more effort to test given the relative lack of driver API's. Firstly, the hardware was tested using the Limesuite Hardware API:

```
LimeUtil --find
LimeQuickTest
```

Which worked to first detect the LimeSDR and then run a series of tests to ensure the hardware was functioning correctly. Specifically the quick test verfied functionality of the clock network, FPGA EEPROM, LMS7002M tuner chip, and the RF loopback. Once the physical hardware was verified it was necessary to ensure that the SoapySDR unifying API was working correctly and could detect the LimeSDR. This was done using the SoapySDRUtil command line tool and output:

```
SoapySDRUtil --find
######################################################
##     Soapy SDR -- the SDR abstraction library     ##
######################################################
Found device 0
addr = 1d50:6108
driver = lime
label = LimeSDR-USB [USB 3.0] 90706024F3821
media = USB 3.0
module = FX3
name = LimeSDR-USB
serial = 00090706024F3821
```

Further information about the LimeSDR and its channels was obtained using the following command:

```
SoapySDRUtil --probe="driver=lime"
```

The output of the above command can be viewed in the appendix A.1 and its information was used to configure the LimeSDR for sampling in the python script.
A python script was then created which utilised the SoapySDR API to sample and save data to a .bin file. The script is viewable in the Appendix A.3, this was based on similar code sourced from DeepWave Digital for an Air-T SDR [10]. Noteably, as seen in the SoapySDR probe information **??**, there was no automatic gain control for the LimeSDR RX1, therefore the low noise amplifier (LNA) gain was set to 30dB. The script was verified by saving a .bin

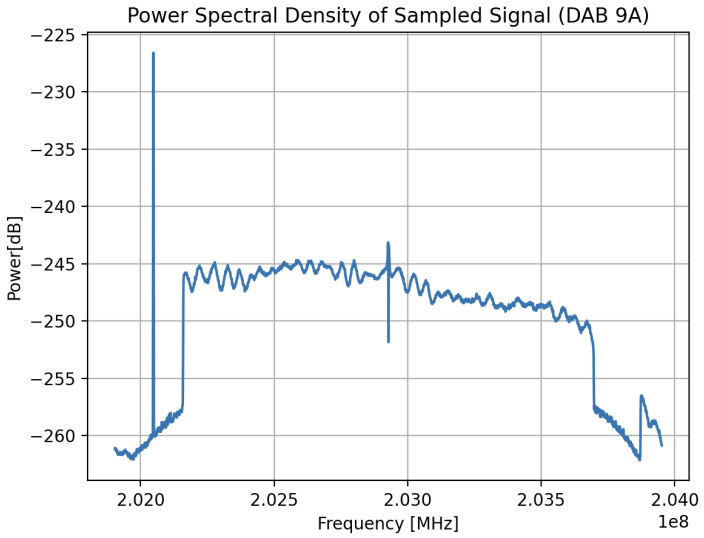


Figure 5.2: LimeSDR PSD Plot (DAB Multiplex 9A)

file and then plotting the PSD of the sample, showing the DAB multiplex centered around 202.928 MHz, as seen below.

Notably, given the LimeSDR has a 12 bit ADC, the output complex IQ data is 32 bits total, with a 16 bit real and 16 bit imaginary component (12 bits zero padded at LSB). The output settings of the LimeSDR are configured via the *SOAPYSDRCS16* format. This resulted in the saved .bin file being 8.2MB for 1 second of data, again showing the high storage requirements for the testbed.

5.1.2 RPi5 and NVME Testing

In order to compare and quantify the differences in the storage performance between the microSD card and the NVME SSD, a series of tests were conducted, utilising the following linux commands via the terminal of the RPi5.

```
lsblk
sudo hdparm -t --direct /dev/nvme0n1
sudo hdparm -t --direct /dev/mmcblk0
```

Resulting in the following output seen below in Table 5.1.

Table 5.1: Disk Read Performance: NVMe vs MicroSD Card

Device	**Read Performance**
NVMe SSD (/dev/nvme0n1)	751.22 MB/sec
MicroSD Card (/dev/mmcblk0)	84.83 MB/sec

The results in Table 5.1 clearly show the obtained significant performance increase when using the NVME SSD compared to the microSD card. Given the large amount of data that generated and processed during the SDR sampling.

5.1.3 Antenna Testing

Before calculations or detection was performed, the signal strength of the received signal was measured. This was done by first using the RTL-SDR and the monopole SMA antenna and compared to the Yagi-Uda antenna. The received signal strength, representing the illuminator DAB signal, was measured approximately 6km away from the transmitter tower. The results of the signal strength measurements are shown in Table 5.2.

Table 5.2: Noise Floor and SNR Comparison with -14.2 dB Gain (RTL-SDR)

Antenna Type	Noise Floor (dB)	Signal Strength (dB)	SNR (dB)
SMA Monopole	-60	-40.9	19.1
Yagi-Uda	-60	-17	43

The above table reflects the high gain of the Yagi-Uda antenna, which is a directional antenna, compared to the monopole SMA antenna. Overall, this testing was valuable to benchmark a noise floor and appreciate the neccessity for a high gain antenna, especially in the context of the low power target signal.

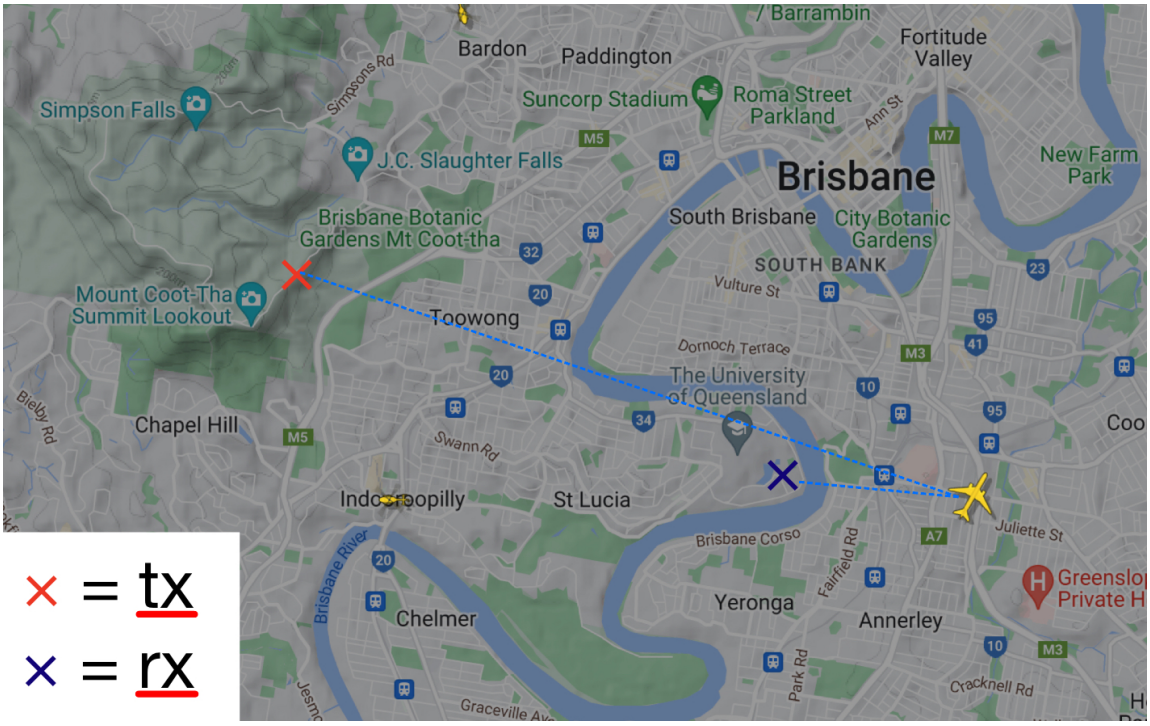


Figure 5.3: Antenna and Testbed Testing Geometry

5.1.4 GPIO and Software Excecutable Testing

The GPIO buttons and LED's were tested using a simple python script, which was executed on the RPi5. The script was able to detect the button press and light the LED, as expected. This was a simple test, but was important to ensure the user interface was functioning correctly. This was implemented via the gpiod library, which was installed on the RPi5. Specifically, the script utilised the gpiochip4 which is the GPIO chip on the RPi5.

Once basic arcade button functionality had been established, using the *subprocess* library, the python script was tested executing the RTL-SDR sampling script as below:

```
rtl_sdr_process = subprocess.Popen(['rtl_sdr', '-s', '2048000', '-f',
'202.928e6', '-n', '2048000', 'testData.bin'])
```

This was tested with the lime SDR

TALK ABOUT USE OF CRONTA FOR TASK AU-TOMA-TION ON BOOT

5.1.5 Detection Edge Processing Comparison

5.1.6 Enclosure Creation

5.2 RTL-SDR vs LimeSDR Detection Comparison

5.3 Overall Design Cost

5.4 Comparison to Existing Work

Chapter 6

Conclusion

Conclusion: what conclusions can be drawn from the results of your research?

6.1 Summary & Conclusions

6.2 Limitations

6.3 Possible Future Work

Chapter 7

Bibliography

[1] Ieee standard definitions of terms for antennas. *IEEE Std 145-1993*, pages 1–32, 1993.

[2] *Introduction to the Internet of Things*, pages 1–50. 2018.

[3] Adafruit. 16mm illuminated pushbutton, 2024. Accessed: 2024-09-01.

[4] Christian R. Berger, Bruno Demissie, JÖrg Heckenbach, Peter Willett, and Shengli Zhou. Signal processing for passive radar using ofdm waveforms. *IEEE Journal of Selected Topics in Signal Processing*, 4(1):226–238, 2010.

[5] Konstanty Bialkowksi. Coms4105 prac 1. University of Queensland, 2024. Image retrieved from the assignment sheet.

[6] Stephen Cass. Passive radar with the krakensdr > spot stuff with tv antennas and a software-defined radio. *IEEE Spectrum*, 59(11):16–18, 2022.

[7] C.J. Coleman, R.A. Watson, and H. Yardley. A practical bistatic passive radar system for use with dab and drm illuminators. In *2008 IEEE Radar Conference*, pages 1–6, 2008.

[8] Core Electronics. Raspberry pi 5 model b (8gb), 2024. Accessed: 2024-09-01.

[9] Ishan Dhar. Tcp connections and sockets: Deep dive into networking fundamentals with linux and python. `https://medium.com/@dhar.ishan04/tcp-connections-and-sockets-deep-dive-into-networking-fundamentals-with-linux-and-p` 2020. Accessed: 2024-10-08.

[10] Deepwave Digital. Recording signals tutorial. `https://docs.deepwavedigital.com/Tutorials/2_recording_signals/`, 2023. Accessed: 2024-09-20.

[11] Anthony Eden. Digital radio in australia: Dab+ technical overview, 2023. Accessed: 2024-09-02.

[12] Michael Edrich, Alexander Schroeder, and Fabienne Meyer. Design and performance evaluation of a mature fm/dab/dvb-t multi-illuminator passive radar system. *IET Radar, Sonar & Navigation*, 8(2):114–122, 2014.

[13] ETSI. En 300 401 - radio broadcasting systems; digital audio broadcasting (dab) to mobile, portable, and fixed receivers. `https://www.etsi.org/deliver/etsi_en/300400_300499/300401/02.01.01_60/en_300401v020101p.pdf`, 2016. Accessed: 2024-09-02.

[14] Flightradar24. Build your own ads-b ground station. `https://www.flightradar24.com/build-your-own`, 2024. Accessed: 2024-09-01.

[15] Vijay K. Garg. Software-defined radio. In *Wireless Communications & Networking*. Elsevier, 2007. Accessed: 2024-08-20.

[16] Alexander Gillis. Secure shell (ssh), 2023. Accessed: 2024-10-08.

[17] H. Griffiths, Christopher J. Baker, and Ieee Xplore. *An introduction to passive radar*. Artech House radar series. Artech House IEEE Xplore, Boston Piscataqay, New Jersey, first edition edition, 2017.

[18] Hugh Griffiths. Chapter 16 - passive bistatic radar. In Nicholas D. Sidiropoulos, Fulvio Gini, Rama Chellappa, and Sergios Theodoridis, editors, *Academic Press Library in Signal Processing: Volume 2*, volume 2 of *Academic Press Library in Signal Processing*, pages 813–855. Elsevier, 2014.

[19] Abdulkadir Guner. Ambiguity function analysis and direct-path signal filtering of the digital audio broadcast (dab) waveform for passive coherent location (pcl). 2002.

[20] Tri-Tan Van Cao James Palmer, Simon Palumbo, , and Stephen Howard. A new illuminator of opportunity bistatic radar research project at dsto. Report, Defence Science and Technology Organisation, May 2009 2009.

[21] Don Koks. How to create and manipulate radar range–doppler plots. Technical Report DSTO–TN–1386, Defence Science and Technology Organisation, Cyber & Electronic Warfare Division, Australian Government, 2012.

[22] Steve M. librtlsdr, 2024. Accessed: 2024-09-17.

[23] Osama Mahfoudia, François Horlin, and Xavier Neyt. On the feasibility of DVB-T based passive radar with a single receiver channel. In *Proceedings of the 15th International Radar Symposium (IRS)*, pages 1–6. IEEE, 2014.

[24] Filip Michalak, Wojciech Zabołotny, Łukasz Podkalicki, Mateusz Malanowski, Marcin Piasecki, and Krzysztof Kulpa. Universal rfsoc-based signal recorder for radar applications. In *2022 23rd International Radar Symposium (IRS)*, pages 136–140, 2022.

[25] C. Moscardini, D. Petri, A. Capria, M. Conti, M. Martorella, and F. Berizzi. Batches algorithm for passive radar: a theoretical analysis. *IEEE Transactions on Aerospace and Electronic Systems*, 51(2):1475–1487, 2015.

[26] Daniel Moser, Giorgio Tresoldi, Christof Schüpbach, and Vincent Lenders. Design and evaluation of a low-cost passive radar receiver based on iot hardware. In *2019 IEEE Radar Conference (RadarConf)*, pages 1–6, April 2019.

[27] myriadrf. Limesdr-usb, 2024. Accessed: 2024-09-10.

[28] Avram Piltch. Raspberry pi 5's wi-fi gets faster. 2024. Accessed: 2024-09-01.

[29] Pimoroni. Nvme base, 2024. Accessed: 2024-08-20.

[30] Pothosware. Soapysdr, 2024. Accessed: 2024-09-17.

[31] Rasmussen and Printables. Raspberry pi 5 case for pimoroni nvme ssd base option, 2024. Accessed: 2024-10-04.

[32] RTL-SDR.com. Buy rtl-sdr dongles (rtl2832u). `https://www.rtl-sdr.com/buy-rtl-sdr-dvb-t-dongles/`, 2024. Accessed: 2024-03-11.

[33] Mathew NO Sadiku and Cajetan M Akujuobi. Software-defined radio: a brief overview. *Ieee Potentials*, 23(4):14–15, 2004.

[34] Dilusha Samarasekara. Popular sdrs, 2023. Accessed: 2024-08-20.

[35] Christof Schüpbach and Samuel Welschen. Direct signal interference mitigation by slow-time frequency correction for ofdm-based passive radar. In *2018 19th International Radar Symposium (IRS)*, pages 1–10, 2018.

[36] Joshua Leigh Sendall. Implementation of a low-cost bistatic radar. Master's thesis, 2016.

[37] Superkuh. Rtl-sdr, n.d. Accessed: 2024-08-27.

[38] Maria V. Vildyaeva, Elizaveta A. Egorova, and Aleksandr B. Vavrenyuk. Using a neural network to convert a radio signal from an rtl-sdr receiver to text. In *2020 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus)*, pages 1449–1451, 2020.

[39] XDA Developers. Best single board computers, 2024. Accessed: 2024-09-01.

[40] Ying Yin, Shunsheng Zhang, Fuxia Wu, Zhulin Zong, and Wei Zhang. Passive radar detection with dvb-t signals. In *2016 CIE International Conference on Radar (RADAR)*, pages 1–5, 2016.

[41] Lu Zhang and Konstanty S. Bialkowski. Passive radar for low doppler targets detection using background estimation and subtraction. In *2020 IEEE International Conference on Radar (RADAR)*, pages 1–6, Brisbane, QLD, Australia, 2020. IEEE.

Appendices

Appendix A

System Intialization and Testing

The below sections detail the initial setup and testing of the Raspberry Pi 5 and different SDR hardware modules.

A.1 SoapySDRUtil LimeSDR Probe Output

```
[INFO] Make connection: 'LimeSDR-USB [USB 3.0] 90706024F3821'
libusb: warning [darwin_transfer_status] transfer error: timed out
[INFO] Reference clock 30.72 MHz
[INFO] Device name: LimeSDR-USB
[INFO] Reference: 30.72 MHz
[INFO] LMS7002M register cache: Disabled

-----------------------------------------------------
-- Device identification
-----------------------------------------------------
  driver=FX3
  hardware=LimeSDR-USB
  boardSerialNumber=0x90706024f3821
  firmwareVersion=4
  gatewareVersion=2.23
  hardwareVersion=4
  protocolVersion=1

-----------------------------------------------------
-- Peripheral summary
-----------------------------------------------------
  Channels: 2 Rx, 2 Tx
  Timestamps: YES
  Clock sources: internal, external
  Sensors: clock_locked, lms7_temp
     * clock_locked (Clock Locked): true
        CGEN clock is locked, good VCO selection.
     * lms7_temp (LMS7 Temperature): 41.495247 C
        The temperature of the LMS7002M in degrees C.
  Registers: BBIC, RFIC0
  Other Settings:
     * SAVE_CONFIG - Save LMS settings to file
       [key=SAVE_CONFIG, type=string]
     * LOAD_CONFIG - Load LMS settings from file
```

```
      [key=LOAD_CONFIG, type=string]
    * OVERSAMPLING - oversampling ratio (0 - auto)
      [key=OVERSAMPLING, type=int, options=(0, 1, 2, 4, 8, 16, 32)]
  GPIOs: MAIN

-----------------------------------------------------
-- RX Channel 0
-----------------------------------------------------
  Full-duplex: YES
  Supports AGC: NO
  Stream formats: CF32, CS12, CS16
  Native format: CS16 [full-scale=32767]
  Stream args:
    * Buffer Length - The buffer transfer size over the link.
      [key=bufferLength, units=samples, default=0, type=int]
    * Latency - Latency vs. performance
      [key=latency, default=0.5, type=float]
    * Link Format - The format of the samples over the link.
      [key=linkFormat, default=CS16, type=string, options=(CS16, CS12)]
    * Skip Calibration - Skip automatic activation calibration.
      [key=skipCal, default=false, type=bool]
    * align phase - Attempt to align phase of Rx channels.
      [key=alignPhase, default=false, type=bool]
  Antennas: NONE, LNAH, LNAL, LNAW, LB1, LB2
  Corrections: DC removal, DC offset, IQ balance
  Full gain range: [-12, 61] dB
    TIA gain range: [0, 12] dB
    LNA gain range: [0, 30] dB
    PGA gain range: [-12, 19] dB
  Full freq range: [0, 3800] MHz
    RF freq range: [30, 3800] MHz
    BB freq range: [-10, 10] MHz
  Tune args:
    * LO Offset - Tune the LO with an offset and compensate with the baseband CORDIC.
      [key=OFFSET, units=Hz, default=0.0, type=float, range=[-1e+07, 1e+07]]
    * BB - Specify a specific value for this component or IGNORE to skip tuning it.
      [key=BB, units=Hz, default=DEFAULT, type=float, range=[-1e+07, 1e+07],
          options=(DEFAULT, IGNORE)]
  Sample rates: [0.1, 61.44] MSps
  Filter bandwidths: [1.4001, 130] MHz
  Sensors: lo_locked
    * lo_locked (LO Locked): false
       LO synthesizer is locked, good VCO selection.
  Other Settings:
    * TSP_CONST - Digital DC test signal level in LMS7002M TSP chain.
      [key=TSP_CONST, default=16383, type=int, range=[0, 32767]]
    * CALIBRATE - DC/IQ calibration bandwidth
      [key=CALIBRATE, type=float, range=[2.5e+06, 1.2e+08]]
    * ENABLE_GFIR_LPF - LPF bandwidth (must be set after sample rate)
      [key=ENABLE_GFIR_LPF, type=float]
    * TSG_NCO - Enable NCO test signal
      [key=TSG_NCO, default=4, type=int, options=(-1, 4, 8)]

-----------------------------------------------------
-- RX Channel 1
-----------------------------------------------------
  Full-duplex: YES
  Supports AGC: NO
  Stream formats: CF32, CS12, CS16
```

```
  Native format: CS16 [full-scale=32767]
  Stream args:
     * Buffer Length - The buffer transfer size over the link.
       [key=bufferLength, units=samples, default=0, type=int]
     * Latency - Latency vs. performance
       [key=latency, default=0.5, type=float]
     * Link Format - The format of the samples over the link.
       [key=linkFormat, default=CS16, type=string, options=(CS16, CS12)]
     * Skip Calibration - Skip automatic activation calibration.
       [key=skipCal, default=false, type=bool]
     * align phase - Attempt to align phase of Rx channels.
       [key=alignPhase, default=false, type=bool]
  Antennas: NONE, LNAH, LNAL, LNAW, LB1, LB2
  Corrections: DC removal, DC offset, IQ balance
  Full gain range: [-12, 61] dB
    TIA gain range: [0, 12] dB
    LNA gain range: [0, 30] dB
    PGA gain range: [-12, 19] dB
  Full freq range: [0, 3800] MHz
    RF freq range: [30, 3800] MHz
    BB freq range: [-10, 10] MHz
  Tune args:
     * LO Offset - Tune the LO with an offset and compensate with the baseband CORDIC.
       [key=OFFSET, units=Hz, default=0.0, type=float, range=[-1e+07, 1e+07]]
     * BB - Specify a specific value for this component or IGNORE to skip tuning it.
       [key=BB, units=Hz, default=DEFAULT, type=float, range=[-1e+07, 1e+07],
           options=(DEFAULT, IGNORE)]
  Sample rates: [0.1, 61.44] MSps
  Filter bandwidths: [1.4001, 130] MHz
  Sensors: lo_locked
     * lo_locked (LO Locked): false
        LO synthesizer is locked, good VCO selection.
  Other Settings:
     * TSP_CONST - Digital DC test signal level in LMS7002M TSP chain.
       [key=TSP_CONST, default=16383, type=int, range=[0, 32767]]
     * CALIBRATE - DC/IQ calibration bandwidth
       [key=CALIBRATE, type=float, range=[2.5e+06, 1.2e+08]]
     * ENABLE_GFIR_LPF - LPF bandwidth (must be set after sample rate)
       [key=ENABLE_GFIR_LPF, type=float]
     * TSG_NCO - Enable NCO test signal
       [key=TSG_NCO, default=4, type=int, options=(-1, 4, 8)]

-----------------------------------------------------
-- TX Channel 0
-----------------------------------------------------
  Full-duplex: YES
  Supports AGC: NO
  Stream formats: CF32, CS12, CS16
  Native format: CS16 [full-scale=32767]
  Stream args:
     * Buffer Length - The buffer transfer size over the link.
       [key=bufferLength, units=samples, default=0, type=int]
     * Latency - Latency vs. performance
       [key=latency, default=0.5, type=float]
     * Link Format - The format of the samples over the link.
       [key=linkFormat, default=CS16, type=string, options=(CS16, CS12)]
     * Skip Calibration - Skip automatic activation calibration.
       [key=skipCal, default=false, type=bool]
     * align phase - Attempt to align phase of Rx channels.
```

```
      [key=alignPhase, default=false, type=bool]
  Antennas: NONE, BAND1, BAND2
  Corrections: DC offset, IQ balance
  Full gain range: [-12, 64] dB
    PAD gain range: [0, 52] dB
    IAMP gain range: [-12, 12] dB
  Full freq range: [0, 3800] MHz
    RF freq range: [30, 3800] MHz
    BB freq range: [-10, 10] MHz
  Tune args:
     * LO Offset - Tune the LO with an offset and compensate with the baseband CORDIC.
       [key=OFFSET, units=Hz, default=0.0, type=float, range=[-1e+07, 1e+07]]
     * BB - Specify a specific value for this component or IGNORE to skip tuning it.
       [key=BB, units=Hz, default=DEFAULT, type=float, range=[-1e+07, 1e+07],
           options=(DEFAULT, IGNORE)]
  Sample rates: [0.1, 61.44] MSps
  Filter bandwidths: [5, 40], [50, 130] MHz
  Sensors: lo_locked
     * lo_locked (LO Locked): true
        LO synthesizer is locked, good VCO selection.
  Other Settings:
     * TSP_CONST - Digital DC test signal level in LMS7002M TSP chain.
       [key=TSP_CONST, default=16383, type=int, range=[0, 32767]]
     * CALIBRATE - DC/IQ calibration bandwidth
       [key=CALIBRATE, type=float, range=[2.5e+06, 1.2e+08]]
     * ENABLE_GFIR_LPF - LPF bandwidth (must be set after sample rate)
       [key=ENABLE_GFIR_LPF, type=float]
     * TSG_NCO - Enable NCO test signal
       [key=TSG_NCO, default=4, type=int, options=(-1, 4, 8)]

-----------------------------------------------------
-- TX Channel 1
-----------------------------------------------------
  Full-duplex: YES
  Supports AGC: NO
  Stream formats: CF32, CS12, CS16
  Native format: CS16 [full-scale=32767]
  Stream args:
     * Buffer Length - The buffer transfer size over the link.
       [key=bufferLength, units=samples, default=0, type=int]
     * Latency - Latency vs. performance
       [key=latency, default=0.5, type=float]
     * Link Format - The format of the samples over the link.
       [key=linkFormat, default=CS16, type=string, options=(CS16, CS12)]
     * Skip Calibration - Skip automatic activation calibration.
       [key=skipCal, default=false, type=bool]
     * align phase - Attempt to align phase of Rx channels.
       [key=alignPhase, default=false, type=bool]
  Antennas: NONE, BAND1, BAND2
  Corrections: DC offset, IQ balance
  Full gain range: [-12, 64] dB
    PAD gain range: [0, 52] dB
    IAMP gain range: [-12, 12] dB
  Full freq range: [0, 3800] MHz
    RF freq range: [30, 3800] MHz
    BB freq range: [-10, 10] MHz
  Tune args:
     * LO Offset - Tune the LO with an offset and compensate with the baseband CORDIC.
       [key=OFFSET, units=Hz, default=0.0, type=float, range=[-1e+07, 1e+07]]
```

```
    * BB - Specify a specific value for this component or IGNORE to skip tuning it.
      [key=BB, units=Hz, default=DEFAULT, type=float, range=[-1e+07, 1e+07],
          options=(DEFAULT, IGNORE)]
Sample rates: [0.1, 61.44] MSps
Filter bandwidths: [5, 40], [50, 130] MHz
Sensors: lo_locked
   * lo_locked (LO Locked): true
      LO synthesizer is locked, good VCO selection.
Other Settings:
   * TSP_CONST - Digital DC test signal level in LMS7002M TSP chain.
     [key=TSP_CONST, default=16383, type=int, range=[0, 32767]]
   * CALIBRATE - DC/IQ calibration bandwidth
     [key=CALIBRATE, type=float, range=[2.5e+06, 1.2e+08]]
   * ENABLE_GFIR_LPF - LPF bandwidth (must be set after sample rate)
     [key=ENABLE_GFIR_LPF, type=float]
   * TSG_NCO - Enable NCO test signal
     [key=TSG_NCO, default=4, type=int, options=(-1, 4, 8)]
```

Listing A.1: SoapySDRUtil Probe Output for LimeSDR

A.2 SoapySDRUtil RTL-SDR Probe Output

```
Found Rafael Micro R828D tuner
[INFO] Opening Generic RTL2832U OEM :: 00000001...
Found Rafael Micro R828D tuner

-----------------------------------------------------
-- Device identification
-----------------------------------------------------
  driver=RTLSDR
  hardware=R828D
  index=0
  origin=https://github.com/pothosware/SoapyRTLSDR

-----------------------------------------------------
-- Peripheral summary
-----------------------------------------------------
  Channels: 1 Rx, 0 Tx
  Timestamps: YES
  Time sources: sw_ticks
  Other Settings:
     * Direct Sampling - RTL-SDR Direct Sampling Mode
       [key=direct_samp, default=0, type=string, options=(0, 1, 2)]
     * Offset Tune - RTL-SDR Offset Tuning Mode
       [key=offset_tune, default=false, type=bool]
     * I/Q Swap - RTL-SDR I/Q Swap Mode
       [key=iq_swap, default=false, type=bool]
     * Digital AGC - RTL-SDR digital AGC Mode
       [key=digital_agc, default=false, type=bool]
     * Bias Tee - RTL-SDR Blog V.3 Bias-Tee Mode
       [key=biastee, default=false, type=bool]

-----------------------------------------------------
-- RX Channel 0
```

```
    ----------------------------------------------------
      Full-duplex: NO
      Supports AGC: YES
      Stream formats: CS8, CS16, CF32
      Native format: CS8 [full-scale=128]
      Stream args:
         * Buffer Size - Number of bytes per buffer, multiples of 512 only.
           [key=bufflen, units=bytes, default=262144, type=int]
         * Ring buffers - Number of buffers in the ring.
           [key=buffers, units=buffers, default=15, type=int]
         * Async buffers - Number of async usb buffers (advanced).
           [key=asyncBuffs, units=buffers, default=0, type=int]
      Antennas: RX
      Full gain range: [0, 49.6] dB
        TUNER gain range: [0, 49.6] dB
      Full freq range: [23.999, 1764] MHz
        RF freq range: [24, 1764] MHz
        CORR freq range: [-0.001, 0.001] MHz
      Sample rates: [0.225001, 0.3], [0.900001, 3.2] MSps
      Filter bandwidths: [0, 8] MHz
```

Listing A.2: SoapySDRUtil Probe Output for RTL-SDR

A.3 Python Code for LimeSDR Signal Capture

```
import numpy as np
import os
import scipy
from matplotlib import pyplot as plt
import SoapySDR
from SoapySDR import SOAPY_SDR_RX, SOAPY_SDR_CS16

#######################################################################################
# Settings
#######################################################################################
# Data transfer settings
rx_chan = 0 # RX1 = 0, RX2 = 1 - Using the RX1 Wideband
N = 2048000 # Number of complex samples per transfer
fs = 2.048e6 # Radio sample Rate
freq = 202.928e6 # LO tuning frequency in Hz - DAB 9A
use_agc = True # Use or don't use the AGC
timeout_us = int(5e6)

# Recording Settings
cplx_samples_per_file = 2048000 # Complex samples per file
nfiles = 1 # Number of files to record
rec_dir = '/Users/flynnmkelly/Desktop/Thesis/PassiveRadarThesis/LimeSDR' # Location of
    drive for recording
file_prefix = 'TestFile' # File prefix for each file

#######################################################################################
# Receive Signal
#######################################################################################
# File calculations and checks
cplx_samples_per_file = N # Use entire buffer size for one file
```

```
real_samples_per_file = 2 * cplx_samples_per_file

# Initialize the LimeSDR receiver
sdr = SoapySDR.Device(dict(driver="lime"))
print(sdr.listSensors()) # Print sensor information

sdr.setSampleRate(SOAPY_SDR_RX, 0, fs) # Set sample rate
# Set gain mode to manual since AGC is not supported
sdr.setGainMode(SOAPY_SDR_RX, 0, use_agc) # Set to False to use manual gain
# SET THE GAIN HERE
lna_gain = 30 # Adjust this value based on your signal strength
sdr.setGain(SOAPY_SDR_RX, 0, "LNA", lna_gain) # Set the gain for the selected channel


sdr.setFrequency(SOAPY_SDR_RX, 0, freq) # Tune to DAB frequency

print('Configuration complete')

# Create data buffer and start streaming
rx_buff = np.empty(2 * N, np.int16) # Buffer for data
rx_stream = sdr.setupStream(SOAPY_SDR_RX, SOAPY_SDR_CS16, [rx_chan]) # Setup data stream
sdr.activateStream(rx_stream) # Start streaming

# Record the data
sr = sdr.readStream(rx_stream, [rx_buff], N, timeoutUs=timeout_us)
rc = sr.ret
assert rc == N, 'Error Reading Samples from Device (error code = %d)!' % rc

# Save data to file
file_name = os.path.join(rec_dir, '{}.bin'.format(file_prefix))
rx_buff.tofile(file_name)

# Stop streaming and close connection
sdr.deactivateStream(rx_stream)
sdr.closeStream(rx_stream)
```

Listing A.3: Python code for capturing DAB signal using LimeSDR

A.4 Code to Plot PSD for Testing LimeSDR

```
# Main Code to Read Data and Plot PSD
file_name = 'TestFile.bin' # Specify your file name here
fs = 2.048e6 # Sampling frequency
center_freq = 202.928e6 # Center frequency

# Read complex data from the file
complex_data = read_complex_int16(file_name)

# Compute the power spectral density
f, Pxx = welch(complex_data, fs, nperseg=2048)

# Shift the frequency axis
f_shifted = f + center_freq

# Plot PSD
```

```
plt.figure(figsize=(12, 6))
plt.semilogy(f_shifted, Pxx)
plt.title('Power Spectral Density of the DAB Signal')
plt.xlabel('Frequency (Hz)')
plt.ylabel('Power/Frequency (dB/Hz)')
plt.grid(True)

# Set x-axis limits to show the full spectrum around the center frequency
plt.xlim(center_freq - fs/2, center_freq + fs/2)

# Format x-axis ticks to show MHz instead of Hz
plt.gca().xaxis.set_major_formatter(plt.FuncFormatter(lambda x, p: f"{x/1e6:.3f}"))
plt.xlabel('Frequency (MHz)')

plt.show()
```

Listing A.4: Main Code to Read Data and Plot PSD