

# Software Defined Radio for Cognitive Wireless Sensor Systems

Final Report

George Smart

Supervisor: Dr John Mitchell

Second Assessor: Prof Paul Brennan

March 2010

## 1 Declaration of Originality

# DECLARATION

I have read and understood the College and Department's statements and guidelines concerning plagiarism.

I declare that all material described in this report is substantially my own work except where explicitly and individually indicated in the text. This includes ideas described in the text, figures, and computer programs.

Name: George Christopher SMART

Signature: \_\_\_\_\_

Date: Friday, 26<sup>th</sup> March 2010

## **2 Abstract**

The Software Defined Radio for Cognitive Wireless Sensor Systems project aims to design and build an activity monitoring device for ZigBee-based cognitive wireless sensor networks. This report explains the evolution of the activity monitor, detailing both the hardware and software aspects of the design.

The complete hardware design uses an FPGA to execute the algorithm in real time. Due to time constraints, the project has not progressed to this stage. At the current stage of development, the user is able to import a comma separated variable (CSV) file containing a sampled waveform at an IF into MathWork's MATLAB.

The MATLAB functions, written by myself, are able to detect if a given channel is busy or clear, and to output this information in a user friendly manner. Functions are able to plot calibrated frequency spectral density graphs to visually show the working. The user is able to specify the sample rate of the waveform and the busy threshold level as arguments to the function, as well as set more in-depth parameters for filter order and bandwidth inside the function. The functions also include some simple error checking to ensure that the user operates them correctly, and that results are reliable.

## Contents

1	Declaration of Originality.....	2
2	Abstract.....	3
3	List of Terms.....	6
4	List of Illustrations and Graphs.....	7
5	Introduction.....	8
5.1	Aim.....	8
5.2	Objectives.....	8
5.3	Report Layout.....	8
6	Background.....	9
6.1	ZigBee.....	9
6.1.1	Devices.....	9
6.1.2	Channel Access and Device Addressing.....	9
6.1.3	Power Consumption and Beaconing Modes.....	10
6.1.4	Security.....	10
6.1.5	Network Layer.....	11
6.1.6	Physical Layer.....	12
6.2	Software Defined Radio.....	13
6.2.1	Cognitive Networks.....	13
6.2.2	Concurrency.....	13
7	Approach.....	14
7.1	Complete System.....	14
7.2	Development System.....	15
8	Down-Conversion.....	16
8.1	Theory.....	16
8.2	Application.....	16
8.3	Testing.....	16
8.3.1	Mixer Frequency Response.....	16
8.3.2	Mixer Third-Order Intercept Point (IIP3).....	17
8.4	Anti-Alias Filter.....	19
9	Sample & Capture.....	20
9.1	Theory.....	20
9.2	Application.....	20
9.3	Test Captures.....	21
10	Decoding.....	22
10.1	Theory.....	22
10.1.1	Fast Fourier Transform (FFT) .....	22
10.1.2	Digital Filtering.....	23
10.2	Application.....	23
10.3	Testing.....	25
10.3.1	Time to Frequency Domain.....	25
10.3.2	Digital Filter.....	25
10.3.3	Filtering Channels.....	26
10.3.4	Scanning Functionality.....	27
11	Conclusion.....	28
12	Future Work.....	29
13	References.....	30
	Appendix A Graphs & Tables.....	31
	Appendix B MATLAB Source Code.....	33
	fft_data.....	33

zigbee..... 34

scan..... 37

### 3 List of Terms

The table below shows a list of terms used in this document.

3IIP	3rd order input intercept point
3OIP	3rd order output intercept point
Ad-Hoc	Point to point network
ADC	Analogue to digital converter
AES	Advanced encryption standard
ALOHA	Network protocol
AODV	Ad-hoc on-demand distance vector (routing protocol)
BB	Base band
CDMA	Code division multiple access
CSMA/CA	Carrier-sense media-access with collision avoidance
CSV	Comma separated variable
dB	Decibels
dBm	Decibels relative to 1milliwatt
DSP	Digital signal processing
DSSS	Direct sequence spread spectrum
ESA	Electronic spectrum analyser
ESG	Electronic signal generator
FDMA	Frequency division multiple access
FFD	Full functionality device
FFT	Fast Fourier transform
FIR	Finite impulse response
FPGA	Field programmable gate array
GHz	GigaHertz
GUI	Graphical user interface
HP	Hewlett-Packard
HZ	Hertz – Unit of frequency
IEEE	Institute of electronic and electrical engineers
IF	Intermediate frequency
IIR	Infinite impulse response
ISM	Industrial, scientific and medical
KHz	KiloHertz
LO	Local oscillator
LPF	Low pass filter
LQI	Link quality indicator
MAC	Media access control
MATLAB	Matrix Laboratory – A numerical computing environment written by The MathWork's Inc.
MHz	MegaHertz
MIC	Message integrity code
MSA	Mega-sample(s)
OQPSK	Offset-quadrature phase shift keying
PC	Personal computer
PHY	Physical layer
PLL	Phase locked loop
RF	Radio frequency
RFD	Reduced functionality device
RSSI	Received signal strength indicator
SAW	Surface acoustic wave
SDR	Software defined radio
TDMA	Time division multiple access
USB	Universal serial bus
VNA	Vector network analyser
Wi-Fi	Wireless fidelity – WLAN standard, based on IEEE 802.11.
WPAN	Wireless personal area network
WSN	Wireless sensor network
ZC	ZigBee coordinator
ZED	ZigBee end device
ZR	ZigBee router

## 4 List of Illustrations and Graphs

### Illustration Index

Illustration 1: Comparison of ZigBee Network Topologies.....	11
Illustration 2: Data Modulation Block Diagram.....	12
Illustration 3: IEEE 802.15.4 Channel Selection (2.45GHz PHY).....	12
Illustration 4: Complete System.....	14
Illustration 5: Development System.....	15
Illustration 6: RF Mixer.....	16
Illustration 7: Mixer Output.....	16
Illustration 8: Basic set up.....	16
Illustration 9: Two-Tone Test.....	17
Illustration 10: Input & Output of Filter.....	19
Illustration 11: Sampling of Analogue Signal.....	20
Illustration 12: Equipment Configuration for Test Captures.....	21
Illustration 13: Complete System with Functioning Parts.....	22
Illustration 14: Sinusoid in (a) Time- and (b) Frequency- Domain, related by a FFT.....	22
Illustration 15: Passband.....	23
Illustration 16: MATLAB Function Block Diagram.....	23
Illustration 17: Comparison between MATLAB FFT and Spectrum Analyser outputs.....	25
Illustration 18: Digital Filter Passband.....	25
Illustration 19: Unfiltered and Filtered Signals showing channel 13.....	26
Illustration 20: Unfiltered and Filtered Signals showing channel 11, 12, 14 & 15.....	26
Illustration 21: Scanning Function Output.....	27
Illustration 22: Proposed Interface.....	29

### Graph Index

Graph 1: Frequency Response of ZFM-2000.....	17
Graph 2: 3rd Order Intermodulation Products for 3IIP Calculation.....	18
Graph 3: ZFM-2000 Frequency Response, as measured by a VNA.....	31
Graph 4: Two-Tones & Intermodulation Products.....	31
Graph 5: ZFM-2000 Mixing & Intermodulation Products.....	32
Graph 6: HP 1GHz LPF Anti-Alias/Image Filter.....	32

## 5 Introduction

Wireless sensor networks (WSNs) are finding their way into many areas of modern life. Applications include home automation and industrial process monitoring, with use extending to battlefield surveillance in military applications<sup>[17]</sup>.

As WSNs become more common the need to have a self adapting network increases too. WSNs are often deployed into electrically noisy environments where conditions are regularly changing. A cognitive wireless sensor network is a solution.

In order to control a cognitive wireless sensor network efficiently, the network controller would need to be well informed of the environment. The ability to receive data from all channels at any time could greatly improve data throughput. This removes the channel choice from the network controller to the end node. Nodes can transmit on any clear channel and data will be routed to the network controller, changing channel if required.

This project allows a person to see how the WSN is using the frequency spectrum available to it.

### 5.1 Aim

The aim of this project is to design and build a device to concurrently monitor the spectral usage of a ZigBee system concurrently. This will incorporate RF front end hardware, digital sampling of the down-converted IF and demodulation of the ZigBee data. These topics are discussed in more detail in subsequent sections.

### 5.2 Objectives

The overall objective is to digitally capture and decode the ZigBee band. This is broken down into separate steps.

- Down-conversion is necessary to shift the ZigBee band from 2.4GHz to a frequency range usable by the sampling electronics.
- Sampling converts the down-converted analogue signal to a digital signal suitable for processing.
- Decoding the sampled data allows us to acquire information from the sampled ZigBee band. This allows us to deduce if a specific channel is in use. This could be further extended in order to decipher the frame headers and potentially the entire frame.

These steps are discussed in greater detail in the relevant sections further on in this document.

### 5.3 Report Layout

This report contains 10 chapters detailing work on the project. It also includes two appendices, which include extra graphs, data-tables, and raw results, as well as source code for functions written by myself and used in this report. References to functions are given in Courier font, for example `fft`, while references to documents or literature are made using the standard superscript square brackets<sup>[x]</sup>.

Each chapter has an introduction and several subsections. These are presented in the order that best aids understanding, with latter sections building on content of earlier sections.

A conclusion and future work chapters toward the end of the report detail how well the project went, and where improvements could be made.



## 6 Background

The background section explains the theory and technicalities of the overall system. It gives an overall idea of the field, including ZigBee, Software Defined Radio, Cognitive Radio and Concurrency.

### 6.1 ZigBee

ZigBee is a specification for wireless personal area networks (commonly referred to as WPANs). The standard is managed by the ZigBee Alliance, whom describe ZigBee to be “the global wireless language connecting dramatically different devices to work together and enhance everyday life”<sup>[1]</sup>. It is a software layer built on the IEEE 802.15.4 standard.

ZigBee style networks were originally devised when engineers realised that existing Wi-Fi and Bluetooth standards were unsuitable for some applications. There was a need for self managing digital networks with the ability to work without infrastructure (device to device, or ad-hoc). ZigBee can provide this kind network.

This work concerns itself primarily with the Physical Layer described in section 6.1.6. Other sections are provided as a background to the work, and for completeness.

#### 6.1.1 Devices

There are 3 types of ZigBee device:<sup>[2]</sup>

- A Network Coordinator is responsible for holding the entire network information including security and trust keys. It is the device that originally started the network and there can only be one coordinator device in a network. It is the most complex device on the network, requiring the most amount of memory and processing power.
- A Full Functionality Device (FFD) supports all the features detailed in the IEEE 802.15.4 standard, making it ideal for network coordinators (ZC) and routing (ZR). It can also be used to run an application layer for use in an end device (ZED).
- A Reduced Functionality Device (RFD) supports less features than a FFD. The reduced functionality is defined in the IEEE 802.15.4 standard. As a result, RFDs are cheaper and simpler to build and are typically used as end devices (ZEDs).

Typically, a network coordinator (ZC) will initiate a network. End devices, typically RFDs, run software to perform the required operations, for example sensing. They are designed to meet the required hardware and software specifications to perform the task reliably and nothing more. Designing them to a tight specification makes them cheap to manufacture, yet reliable. FFDs can also run software to perform specific tasks, as with the end devices. FFDs are also able to route packets around the network, functioning as routers (ZRs).

#### 6.1.2 Channel Access and Device Addressing

The IEEE 802.15.4 standard includes two methods for channel-accessing:

- In non-beaconing networks, ALOHA carrier-sense multiple-access with collision avoidance (CSMA/CA) is used, with an acknowledgement for received packets: This means that if a node has data to be sent, it will first listen on the channel for any traffic. If the channel is busy, the transmitting node must wait a arbitrarily random amount of time. This decreases the chances of collisions on the channel. Once the channel is clear, and the node has decided to transmit the data, the transmitting node must then wait for an acknowledgement

frame from the destination. If it does not receive one within a given time, it is assumed the data was lost and is resent 'later'.

- In beaconing networks, a super-frame structure is used to control channel access. This optional set up allocates nodes with time critical data a specific regular time-slot. The super-frames are set up by the network coordinator to transmit beacons at predetermined intervals (multiples of 15.38ms, up to 252 seconds)<sup>[3]</sup>. This provides 16 equal-length time-slots between beacons, used for channel access. This guarantees nodes a low latency and dedicated bandwidth channel.

For communications systems there are two resources available: time and frequency. Devices are either given a specific frequency for all of the time, called frequency division multiple access (FDMA); or, they are given a specific time-slot to transmit in, and they can use a wide range of frequencies, called time division multiple access (TDMA). ZigBee uses code division multiple access (CDMA) as a channel access method, allowing many nodes to use the same frequency and timeslots. Individual nodes are recoverable by from their code – a unique identifier for the node.

Device addresses are either 64-bit (extended) addresses, with the option for a 16-bit (short) address. The network makes use of the media access controller's (MAC's) ability to encode the source and destination addresses twice into the same packet, to reduce failure in the network.

### 6.1.3 Power Consumption and Beaconing Modes

Slave nodes are in a sleep mode for most of the duty cycle, waking up only to refresh their presence on the network. Waking from sleep to the point the node is ready to transmit data takes approximately 15ms, and enumeration of a new device onto the network takes approximately 30ms<sup>[3]</sup>.

ZigBee networks may be configured in beacon or non-beacon modes:

- Non-beacon mode operates like traditional multi-node networks. Each node operates independently, and decides when to transmit without outside control. From this, the possibility of nodes doubling (transmitting over each other) arises. This mode is typically used for systems where the slave node should sleep for as long as possible. The device wakes up with regularity to confirm its existence on the network, and then returns to sleep. It also wakes up to transmit any events, for example to transmit the sensor sample data. As the network coordinator is typically mains powered, it is able to have its receiver running continuously and so can receive the random transmissions of the nodes as events occur.
- Beacon mode provides a method for each client to know when to communicate with another. In this mode, the network coordinator organises communications. The main advantage of this is reduced power consumption. This is suitable for when the network coordinator also has power constraints. In this mode, the coordinator wakes up, and transmits a beacon. Each node listens for any data addressed to it. The network coordinator listens for any replies based on the super-frame structure, and then resumes sleeping itself. It is evident that this design is energy efficient, and is useful when devices are running from batteries.

Long sleep intervals between beacons mean that the timing systems in the devices have to either be very accurate or come on slightly earlier, to ensure that a beacon is not missed. Both of these options make the network more expensive in terms of energy or hardware quality.

### 6.1.4 Security

ZigBee boasts data security as a key benefit, over older networking techniques. The IEEE 802.15.4 MAC sub-layer offers four security services:<sup>[3]</sup>

- Access Control: The network coordinator holds a list of trusted devices on the network. Only pre-approved (enumerated) devices may talk in the network.
- Data Encryption: 128-bit AES (advanced encryption standard) symmetric keys. Data, commands, or beacons cannot be read by parties without the cryptographic key.
- Frame Integrity: MIC (message integrity code) protects data. Data cannot be modified by parties without the cryptographic key.
- Sequential Freshness: Freshness number determines if frames should be ignored. Prevents a party from recording an encrypted frame and replaying it, as the freshness number must be incremented for successive commands.

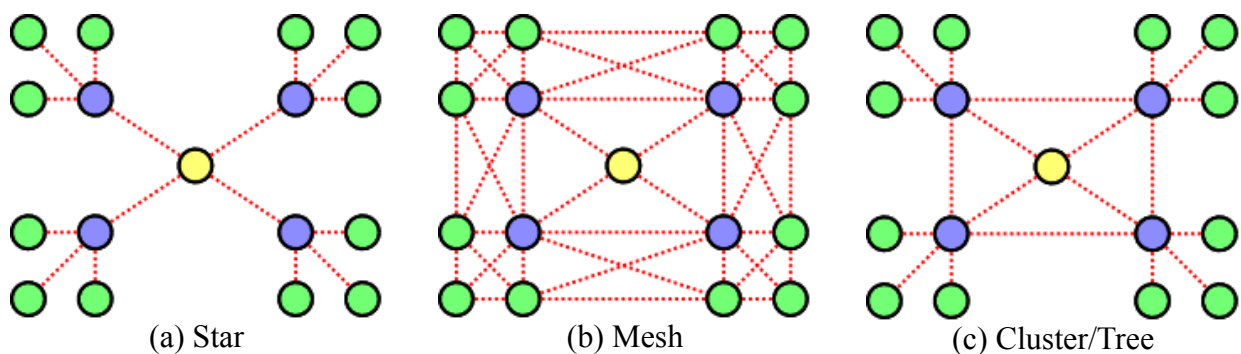
### 6.1.5 Network Layer

The network layer is an interface between the MAC layer of the IEEE 802.15.4 standard and the application layer, ensuring that applications adhere to the standard. The ZigBee network layer also provides services typical of a network layer, including frame routing (AODV), device association (including address assigning of new devices) and disassociation.

The network layer of the ZigBee stack supports 3 topologies:<sup>[4]</sup>

- Star: All nodes communicate with one central hub node (typically a router or coordinator).
- Mesh: Nodes can communicate with other nodes in the network, as well as the central node.
- Cluster/Tree: A Hybrid where end devices communicate with a local router in a star topology and routers communicate with each other (and the coordinator) in a mesh topology.

These topologies are shown graphically in Illustration 1.



*Illustration 1: Comparison of ZigBee Network Topologies*

The star (a) topology is a point-to-point set up. Each device communicates with the parent router/coordinator directly. This has the advantage of simplicity and potentially low power requirements.

In the mesh (b) topology, any device can communicate with any other device, providing they are within radio range of each other. This has the advantage of extending the physical area the network can be used over, and may be used to provide fault resistance if a certain node in the network goes down.

The cluster/tree (c) topology attempts to take the advantages of both star and mesh topologies. It maintains the fault tolerance of the mesh topology, and gains back the simplicity and reduced energy requirements of the star topology for devices on the edge of the network.

### 6.1.6 Physical Layer

The ZigBee physical layer is as specified in the IEEE 802.15.4 standard. For the 2.45GHz band, the operating frequency range is from 2400–2483.5MHz, with a fixed symbol rate of 62,500 symbols/second<sup>[3]</sup>. Four input bits are grouped together to make one symbol<sup>[3]</sup>. Combining these two values together, we obtain a value for data-rate, given by:

$$R = 4 \cdot 62.5 \times 10^3 = 250Kbit/s$$

Direct-sequence spread spectrum (DSSS) is used with the 2.45GHz ZigBee physical layer<sup>[3]</sup> to perform spectral spreading. There are several benefits to using this technique, most importantly:

- Resistance to signal jamming, that is, it is difficult for parties to block communications (either intentionally or unintentionally), and;
- The sharing of a single channel amongst several devices, so multiple nodes can transmit on the same channel simultaneously.

To achieve this spectral spreading, DSSS maps each symbol (4-bits) into a chipping sequence (32-bits) resulting in a spreading factor,  $SF$ , of 32 times<sup>[3]</sup>. With the fixed symbol rate of 62.5Ksymbols per second, we get a channel bandwidth as follows:

$$BW = SF \times SymbolRate = 32 \times 62.5 \times 10^3 = 2 \times 10^6 = 2MHz$$

This chip sequence is then modulated onto the carrier using offset quadrature phase shift keying (OQPSK) with half-sine pulse shaping and 2 chip bits per symbol<sup>[3]</sup>. The entire process is shown in Illustration 2.

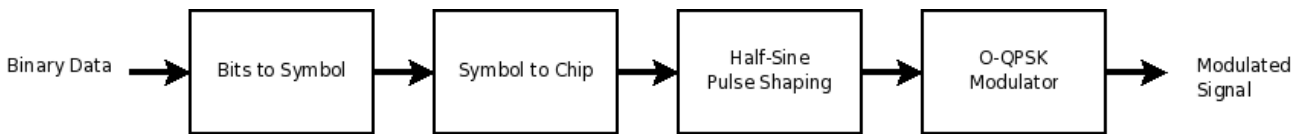


Illustration 2: Data Modulation Block Diagram

The IEEE 802.15.4 standard (section 6.1.2 *Channel Assignments & Numbering*) gives the following formula for the calculation of carrier frequency for a given channel number,  $k$ <sup>[3]</sup>.

$$F_C = 2405 + 5(k - 11) \text{ for } k = 11, 12, \dots, 26$$

Illustration 3 shows the 2.45GHz industrial, scientific and medical (ISM) band. It shows the frequencies of the 16 channels, their respective channel numbers, and their spectral position and spacing.

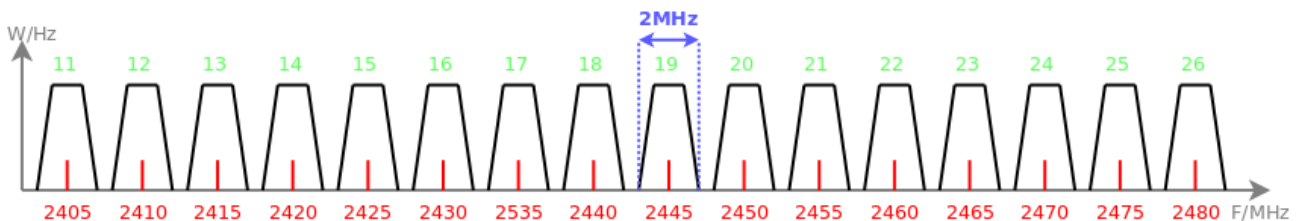


Illustration 3: IEEE 802.15.4 Channel Selection (2.45GHz PHY)

The 2.45GHz physical layer (PHY) supports channel numbers in the range 11...26. Channel 0 is provided by the 868MHz PHY, and channels in the range 1...10 by the 915MHz PHY. The 868/915MHz PHYs have different bit/symbol rates and different modulation techniques. This work does not concern itself with these PHYs, and so their specifications are not mentioned.

From the carrier frequency formula, we can deduce that ZigBee has a channel spacing of 5MHz on the 2.45GHz band. There is therefore no overlap of channels in the frequency spectrum.

## 6.2 Software Defined Radio

Software defined radio (SDR) is rapidly becoming the new method for radio signal processing. It offers several advantages over traditional radio processing systems. It allows for existing radio hardware to be replaced by digital signal processing (DSP) algorithms, offering a performance closer to ideal. In the receiver, the incoming signal is converted into the digital domain as soon as possible. This has the major advantage of reducing the amount of fixed hardware in a radio transceiver system.

By processing the signal in software, we eliminate the need to modify the hardware to reflect new modes or features, and so new radio protocols can be implemented by simply changing software. This eliminates the cost of designing and manufacturing.

Software defined radio is the obvious choice here, as we require the system to work for cognitive wireless sensor networks. As we also wish for the system to monitor all channels concurrently, a digital approach is more suitable.

### 6.2.1 Cognitive Networks

A cognitive wireless sensor network is one in which parameters relating to the physical wireless link are changed in real-time by the nodes in order to achieve the best communication. This is well defined by the quote:

*“Cognition, a continuous process involving sensing, reasoning, understanding and reacting, can be applied to wireless networks in order to adapt the system to the highly dynamic wireless ecosystem.”* <sup>[13]</sup>

This means that parameters of the radio transceiver are adjusted autonomously to achieve the best link. Software defined radio is paramount here, as it allows for easy control of the radio used for the link layer.

### 6.2.2 Concurrency

Existing hardware is able to monitor a single channel, but to monitor all of the channels on the 2.45GHz PHY simultaneously would require either 16 separate receivers or several receivers scanning through channels quickly.

Having 16 separate receivers would result in a bulky device which would be expensive to build. It would also be time consuming to calibrate each of the individual receivers.

The scanning approach requires less hardware and so would result in a cheaper and smaller device. However, scanning the receiver through 16 channels would be a slow process: after the channel is changed, the system must wait for the radio receiver to lock to the new channel, listen for a short time, and move on. As frame lengths are short, there is a large possibility that some data would be missed by the monitoring device.

SDR allows us to overcome this problem by implementing all 16 receivers inside one DSP chip. We are able to have the speed and reliability of 16 separate receivers with the size and cost of one DSP chip. In using a high specification general purpose field programmable gate array (FPGA), we are able to perform DSP for 16 separate radios and control a universal serial bus (USB) interface to transfer data to and from a host personal computer (PC).

## 7 Approach

The approach chapter details how the complete system solution is achieved, and how this system is broken down into smaller individually testable parts.

### 7.1 Complete System

As the aim of this project is relatively complex, it is split into 3 major steps. Each of these is explained in subsequent sections of this report.

Illustration 4 shows the complete block diagram for the system. The highlighted sections roughly match the chapters of this document.

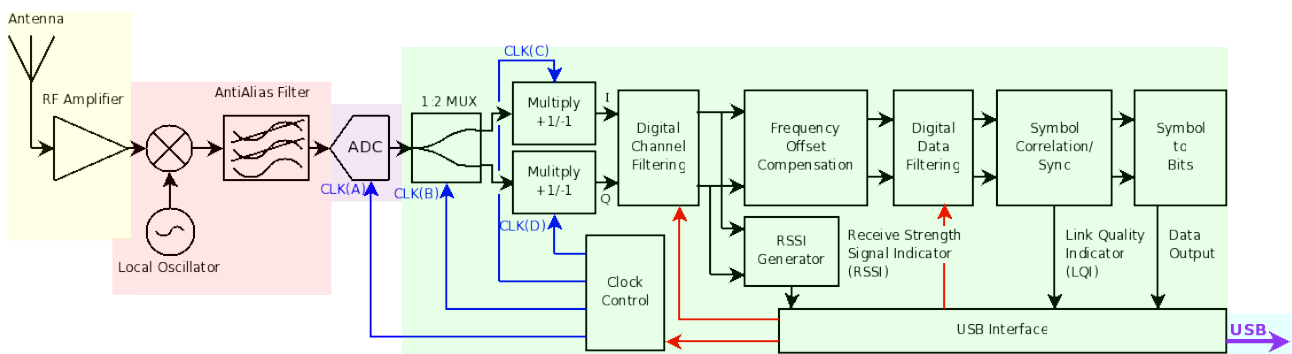


Illustration 4: Complete System

The yellow section corresponds to the RF input, and is not considered in this report. In later stages of the project's development, it will be a tuned RF amplifier and antenna. For testing purposes, this is replaced with a ChipCon CC2400 Evaluation Board is used, as explained in the testing sections of subsequent chapters.

The pink section is the down-converter (*Ch8*) and includes the anti-aliasing filter.

The purple section is the analogue to digital converter, and is responsible for the sampling (*Ch9*).

The green section is the decoding (*Ch10*) section. This is where the digital signal is processed.

The blue section is the interface section, supplying the user with a graphical user interface.

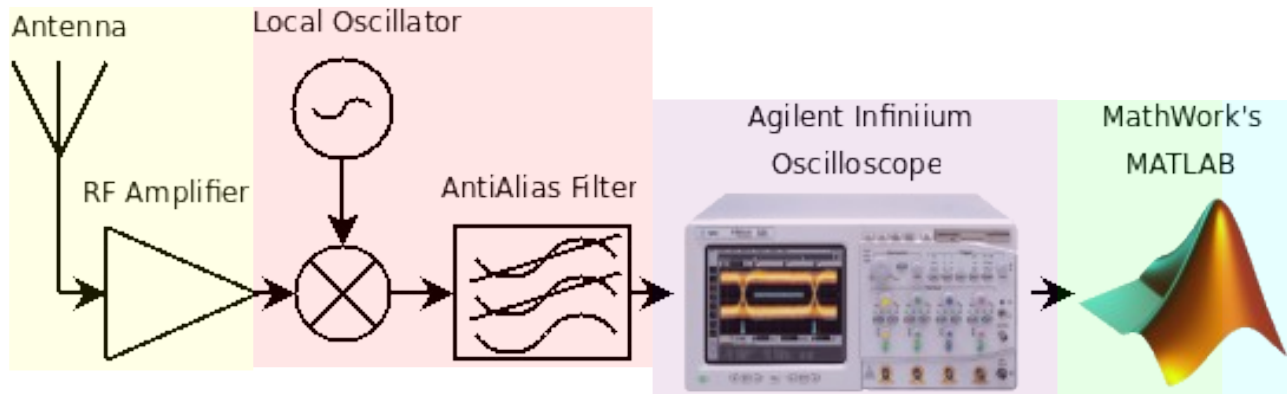
The system operates as follows:

- A transmitting node is received via the antenna, and input into the tuned RF front end amplifier. This amplifies the signal so it has enough power to operate the mixer in the linear region. It is important that this amplifier is a low-noise amplifier, as this plays a fundamental role in setting the maximum sensitivity of the receiver.
- This amplified signal is then fed to the down-converter, which takes the 2.45GHz signal from the RF front end and moves it down in frequency. This is done to overcome limitations in analogue to digital converters (ADCs). *See chapter 8 for more detail.*
- The ADC samples this down-converted signal, turning it from an analogue waveform into a digital discretised quantised data-stream. *See chapter 9.*
- This data-stream is then processed along the top path of the signal processing chain. Sampled radio-waves go into the DSP section from the ADC, and ultimately re-appear as binary data at the end of the processing chain. The USB interface controller collects the processed binary data from the end of the processing path, and packages it up to be sent off to the host PC. Aside from this, the USB interface also allows for settings and parameters from the host PC to be passed to individual blocks in the signal processing chain.

## 7.2 Development System

A simplified system was used to develop the device. This involved swapping some parts for commercially available parts to enable other system sections to be designed, calibrated and set up.

To do this, the block diagram in Illustration 4 from section 7.1 was changed to the one shown in Illustration 5.



*Illustration 5: Development System*

The analogue to digital conversion is done with an Agilent Infiniium oscilloscope. The MathWork's MATLAB provides a good environment to develop algorithms for signal processing. MATLAB also provides a graphical user interface (GUI) with graphing ability, allowing me to see the interim stages of the signal processing. The individual chapters in this document are based around the block diagram shown in Illustration 5.

## 8 Down-Conversion

The process of reducing the frequency of a signal is referred to as down-conversion. This is typically done when converting an radio frequency (RF) signal to intermediate frequency (IF); or, when converting from IF to base band (BB).

Here we use down-conversion to move from the 2.45GHz ISM band where ZigBee is used to an IF much lower in frequency. In doing this, we remove the need for very high specification ADCs allowing this system to use a much cheaper and widely available ADC.

### 8.1 Theory

An RF mixer is used to perform the down-conversion. Illustration 6 shows the RF mixer symbol. A double-balance mixer takes an RF input at frequency  $f_{RF}$  and a local oscillator (LO) input at frequency  $f_{LO}$  and produces an IF output consisting of the sum and difference frequencies,  $f_{IF} = f_{RF} \pm f_{LO}$ . The mixer output is typically fed through a filter to remove either the sum ( $f_{RF} + f_{LO}$ ) or the difference ( $f_{RF} - f_{LO}$ ) frequency. Graph 5 in Appendix A Graphs & Tables shows this.

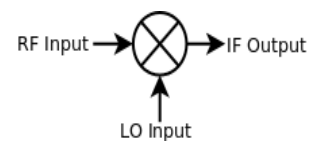


Illustration 6: RF Mixer

### 8.2 Application

An IF of 20MHz was picked. Using low-side injection ( $f_{LO} < f_{RF}$ ) combined with the principles of operation for an RF mixer, it is possible to calculate a suitable value for the local oscillator:

Assuming an ideal mixer,

$$f_{IF} = f_{RF} \pm f_{LO}$$

As we are concerned with the difference frequency only,

$$f_{IF} = f_{RF} - f_{LO}$$

From the IEEE 802.15.4 standard, the first channel supported by the 2.45GHz PHY has a carrier of 2405MHz. Using this, and our choice of IF frequency, 20MHz, we are able to calculate the IF:

$$f_{LO} = f_{RF} - f_{IF} = 2405 - 20 = 2385MHz$$

The output of the mixer will need to be followed by an image rejection filter and an antialiasing filter. Illustration 7 shows the output of the mixer. IF1 is the desired difference frequency. IF2, the image (sum) frequency. LO and RF are the two input frequencies. A double-balanced mixer should reduce the powers of LO and RF, but the signals still exist.

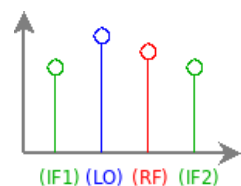


Illustration 7: Mixer Output

### 8.3 Testing

The Mini-Circuits ZFM-2000 mixers have an operating frequency range of 100-2000MHz. As I will be using them outside this specified operating range, I thought it important to verify the functionality of the devices and assess their suitability for use in the RF front end.

#### 8.3.1 Mixer Frequency Response

The first test carried out was to establish the frequency response of the system. The two electronic signal generators (ESGs) were used as RF and LO inputs – an electronic spectrum analyser (ESA) was connected at the IF

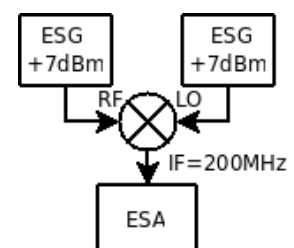


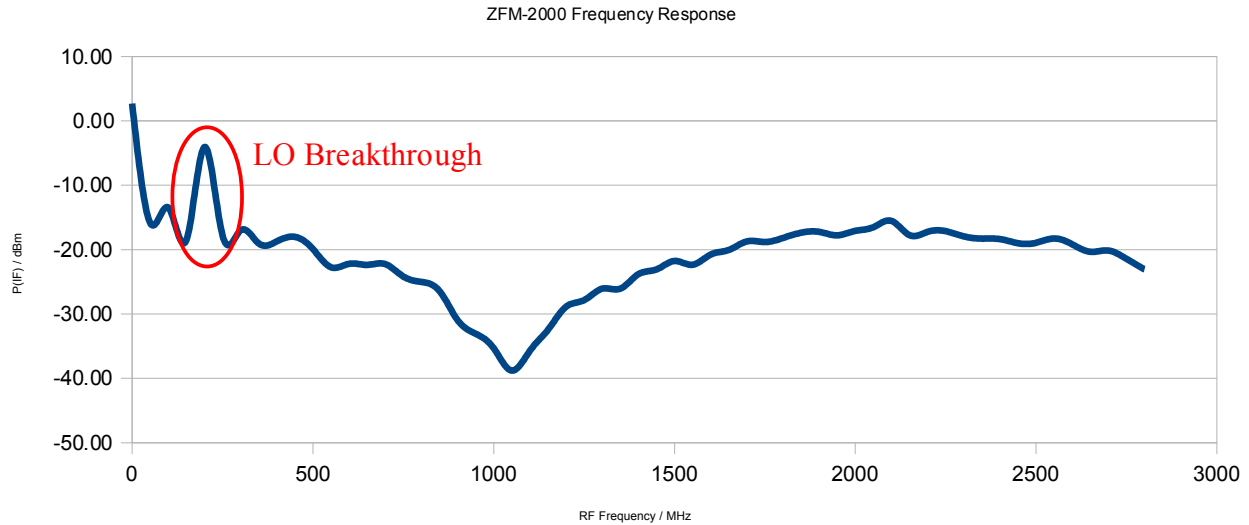
Illustration 8: Basic set up



output. Illustration 8 gives a graphical representation of the hardware configuration – numerical values can be found in Appendix A, table 2.

The RF and LO inputs were frequency-swept at constant power, such that:

$$F_{IF} = F_{RF} - F_{LO} = 200\text{MHz} \quad \text{and} \quad P_{IN(RF)} = P_{IN(LO)} = +7\text{dBm}$$



Graph 1: Frequency Response of ZFM-2000

Graph 1 shows the frequency response of the mixer under the test conditions given above. Graph 3 in Appendix A Graphs & Tables shows similar results from a RF vector network analyser (VNA).

### 8.3.2 Mixer Third-Order Intercept Point (IIP3)

The 3<sup>rd</sup> order input intercept point (3IIP) is a figure of merit for intermodulation product suppression – ideally, it should be as high as possible. As RF input power increases, so do the levels of the intermodulation products – these are unwanted signals. The levels of the intermodulation products increase at 3 times the speed of the RF input. The theoretical point at which the difference product is equal in level to the 3<sup>rd</sup> order harmonics is the known as the intercept point. The test is usually done with two-tone intermodulation products. If two RF signals  $f_{RF1}$  and  $f_{RF2}$  are mixed with LO frequency,  $f_{LO}$ , then the following output signals will exist at the IF output:

$$(2f_{RF1} \pm f_{RF2}) \pm f_{LO} \quad (2f_{RF2} \pm f_{RF1}) \pm f_{LO}$$

Of these outputs, the following two are likely to cause us problems, as they will be close to the desired IF signals:

$$(2f_{RF1} - f_{RF2}) - f_{LO} \quad (2f_{RF2} - f_{RF1}) - f_{LO}$$

With this in mind, we are able to devise a test to carry out a two-tone test on the mixer. Using three ESGs, a Mini-Circuits RF power combiner and an ESA, we are able to determine the characteristics of the Mini-Circuits ZFM-2000 mixer. Two of the ESGs are connected via the power combiner into the RF input of the mixer under test – these are the 'two-tones' used in the two-tone test. The third ESG is connected to the LO input of the mixer under test – this serves as our local oscillator. The IF output of the mixer is then connected to the ESA – this gives a graphical representation of the output of the mixer in the frequency domain. The block diagram in Illustration 9 shows the equipment configuration.

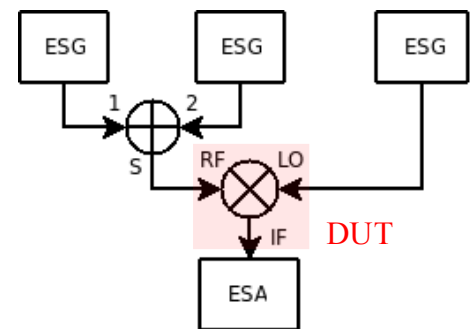
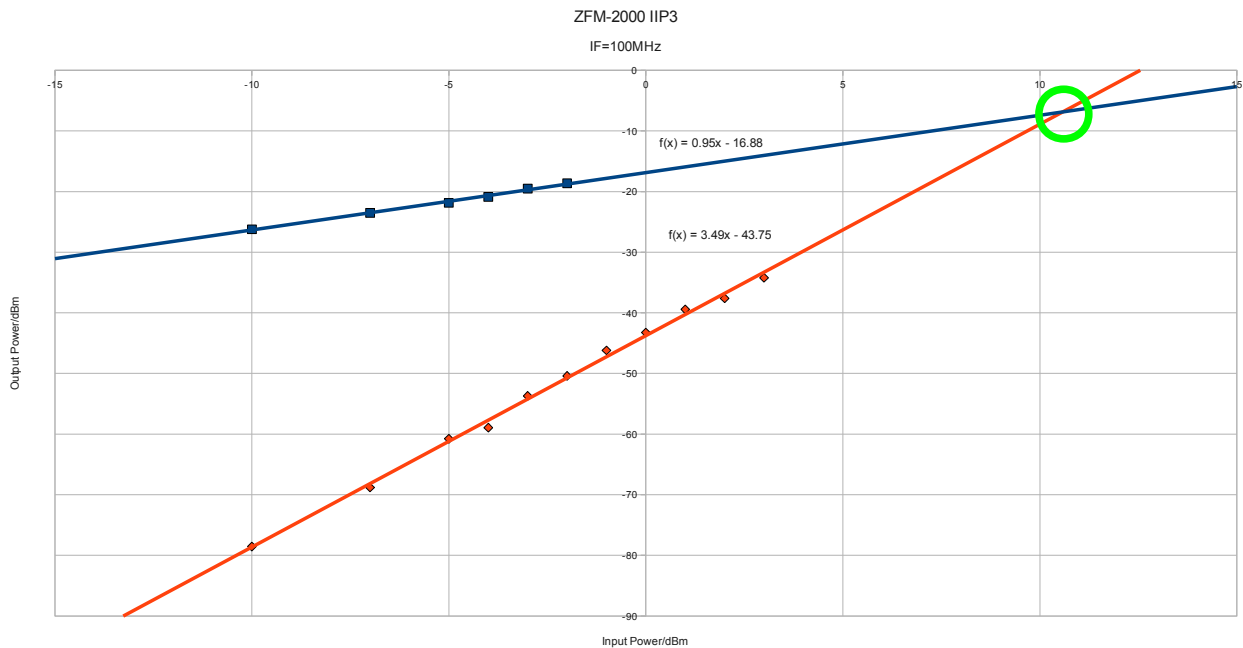


Illustration 9: Two-Tone Test

With a fixed LO input power, the RF input power was increased in 1dBm increments. The IF

fundamental difference frequencies  $f_{RF1} - f_{LO}$  and  $f_{RF2} - f_{LO}$  were recorded along with the 3<sup>rd</sup> order intermodulation products  $(2f_{RF1} - f_{RF2}) - f_{LO}$  and  $(2f_{RF2} - f_{RF1}) - f_{LO}$ . Graph 2 shows the results obtained from the two-tone test, and allows for the calculation of the 3<sup>rd</sup> order input intercept point (3IIP), circled green. Data for graph 2 can be found in Appendix A, table 1.



Graph 2: 3rd Order Intermodulation Products for 3IIP Calculation

Taking only the linear part of the data set, two linear trend lines were added to graph 2. The theoretical point at which the fundamental and the 3<sup>rd</sup> order intermodulation products intercept, is the 3IP. The input power required to achieve this is referred to as the 3IIP and similarly the output power at which this occurs, the 3OIP.

We can calculate the 3IIP by simply equating the two formulae to obtain their interception.

From graph 2 we get the formulas for both of the trend lines,

$$y_1(x) = 0.95x - 16.88$$

$$y_3(x) = 3.49x - 43.75$$

At the intercept point,

$$y_1(x) = y_3(x)$$

$$3.49x - 43.75 = 0.95x - 16.88$$

$$3.49x - 26.87 = 0.95x$$

$$2.54x - 26.87 = 0$$

$$2.54x = 26.87$$

$$x = \frac{26.87}{2.54} = 10.543307087$$

$$3IIP = 10.543dBm$$

This is a little lower than expected.

The 3IIP is a measure of how much distortion the mixer creates at a given input power, and how linear the mixer is. It shows how well the mixer handles power<sup>[10]</sup>. If this value is low, as ours is, it

indicates that the mixer may distort the output when high input levels are applied.

## 8.4 Anti-Alias Filter

As the mixer is not perfect, the two input signals, RF and LO will be present at the output, along with the sum and difference frequencies. As we are only interested in the difference frequency, the other three signals must be removed. Illustration 10 shows this pictorially: 10a shows the input to the filter, and 10b shows the filter output.



*Illustration 10: Input & Output of Filter*

As the device will be tested in a controlled environment, the antialiasing filter is used primarily to suppress the image frequencies. The source data is sent from a ChipCon CC2400EB evaluation board, and so the output is clean. Aside from harmonics, there is only ever one channel at a time being sent. There is no chance that there will be signals greater in frequency than half of the sample rate, and so I have chosen to use a standard part from the lab. The part used was a HP 1GHz low pass filter (LPF). Although this is not ideal, it serves the primary purpose of removing the LO, RF and image signals. Graph 6 in Appendix A Graphs & Tables shows the passband of the HP 1GHz LPF.

## 9 Sample & Capture

Sampling is the process of converting a continuous-time analogue signal into a discrete-time signal. The sampling provides the interface between the analogue domain and the digital domain. I have used an Agilent Infiniium oscilloscope as it provides an easy way of capturing waveforms for testing.

### 9.1 Theory

The continuous-time signal is discretised, so the signal only changes at specific intervals of time. It is then quantised, that is, rounded to a numerical value close to that of the analogue signal.

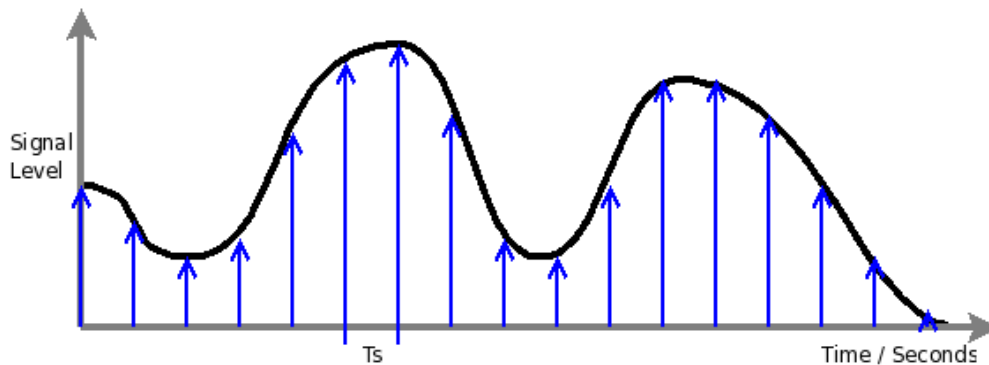


Illustration 11: Sampling of Analogue Signal

Illustration 11 shows a continuous-time analogue signal (black trace). This is sampled every  $T_s$  seconds, or, every  $T_s$  seconds, the current signal level (length of blue arrow) is read and output digitally.

A real ADC will have a fixed and finite resolution, and so the signal level is rounded to the closest digital level. The signal therefore suffers from quantisation distortion. Once accuracy is lost, it cannot be recovered.

The sampling time, shown in Illustration 11 as  $T_s$ , is very important. Usually referred to in terms of frequency,  $F_s = \frac{1}{T_s}$ , relates to the maximum frequency that can be sampled.

The Nyquist-Shannon sampling theorem guarantees that a bandwidth limited signal can be reconstructed from the sampled version, providing that the sample frequency  $F_s$  is more than twice the maximum frequency present in the bandwidth limited signal.

### 9.2 Application

The IEEE 802.15.4 standard specifies that there are 16 channels (11 to 26) ranging from 2405MHz to 2480MHz. The Agilent Infiniium oscilloscope has a maximum sample-rate of 100MSa/second, which was used. From the Nyquist-Shannon sampling theorem, we can calculate the maximum input bandwidth:

$$BW < \frac{F_s}{2}$$

$$BW < \frac{100 \times 10^6}{2}$$

$$BW < 50 \times 10^6 \text{ Hz}$$

So the maximum input frequency must be less than 50MHz.

By using principles of oversampling it is possible to get a better resolution, an improved signal to noise ratio, and to reduce cut-off steepness required on the antialias filter<sup>[9]</sup>.

The Infiniium scope is able to store a maximum of 65536 samples. From this, we can obtain the *observation period*, or the period of time of which will be captured:

$$T = \frac{65536}{100 \times 10^6} = 655.36 \times 10^{-6} = 655.36 \mu S$$

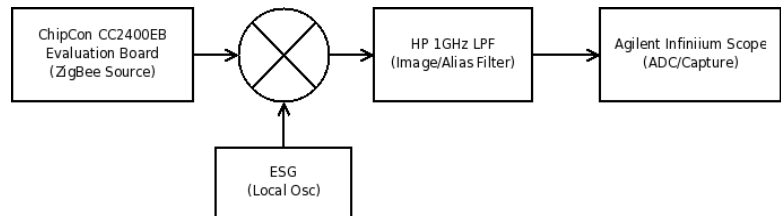
With a symbol rate of 62.5Ksymbols per second, we can calculate the number of symbols that will be captured:

$$N_{SYMBOLS} = (62.5 \times 10^3) \times (655.36 \times 10^{-6}) = 40.96 symbols$$

A sample of 40 symbols is more than enough to serve as a set of test signals for filter design. However, 40 symbols would not represent a complete frame and so it would be difficult to decode data from the capture.

### 9.3 Test Captures

To allow the development of the DSP algorithms in MATLAB, I needed to acquire some test captures. Illustration 12 shows how the equipment was set up to make such captures.



Channels 11 to 15 were captured using the Agilent Infiniium scope, and serve as test signals. Along side these test frames, sinusoidal signals of varied power were input into the system in place of the ChipCon CC2400EB. These enabled me to calibrate the graphs and to make a meaningful comparison with the user input threshold level.

Both the ZigBee and the power calibration signals were sampled at 100MSa/second with 65536 samples, as explained in section 9.2.

## 10 Decoding

Once the IF has been sampled and converted into the digital domain, it is then necessary to begin the process of recovering the original signal.

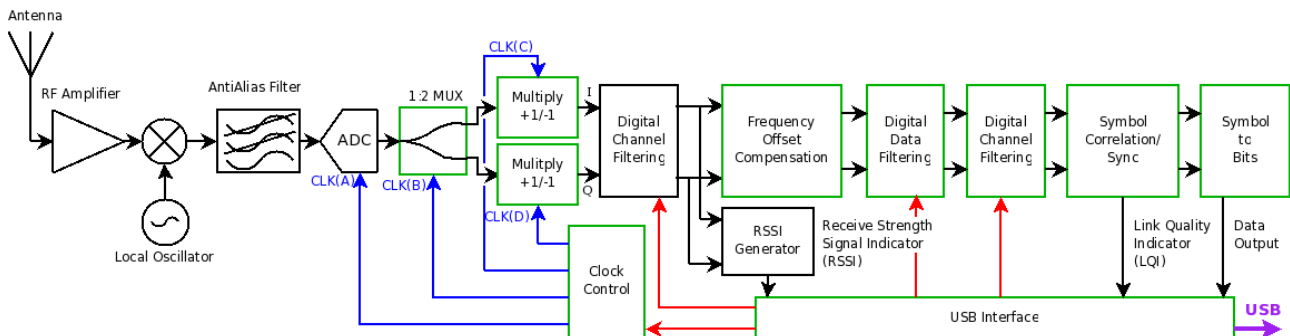


Illustration 13: Complete System with Functioning Parts

Illustration 13 shows the complete block diagram from section 7.1. The blocks with black outlines show the stages which have been implemented, those in green are reserved for future work. I started by getting the received signal strength indicator (RSSI) to work. I wanted this to be calibrated, so that meaningful values were displayed.

The first step of the project was to assess which channels are in use at any given time. The RSSI is an ideal way of doing this. Comparing this value to a threshold value will give a good indication if the specified channel is in use.

### 10.1 Theory

#### 10.1.1 Fast Fourier Transform (FFT)

The sampled data is first converted into the frequency domain by fast Fourier transform (FFT). The FFT is a complex algorithm to compute the discrete Fourier transform. There are many different approaches to computing an FFT. Mathwork's MATLAB `fft` function is based on the FFTW algorithm<sup>[14][15]</sup>.

The fast Fourier transform is used to convert the time domain waveform capture from the oscilloscope into the frequency domain. Illustration 14 shows how a sine wave in the time domain can be converted using a FFT into the frequency domain.

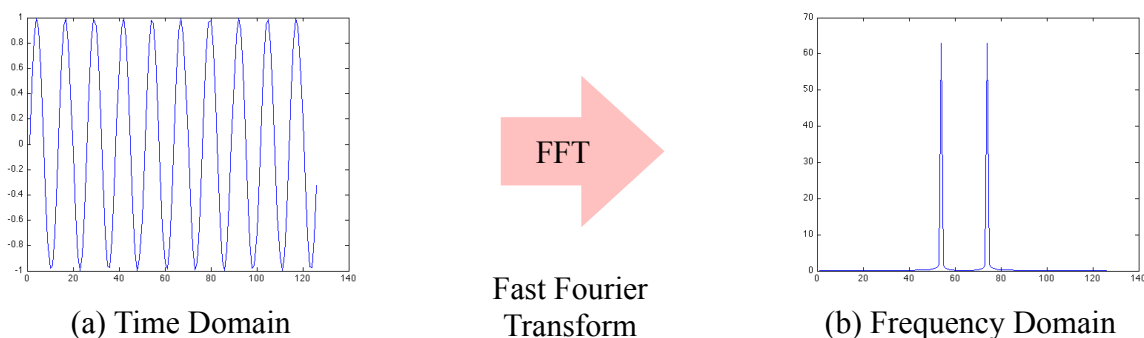


Illustration 14: Sinusoid in (a) Time- and (b) Frequency- Domain, related by a FFT.

### 10.1.2 Digital Filtering

Digital Filtering enables us to extract the frequency range of interest. I needed to filter out the frequency range for each given channel. Knowing the channel spacing and the carrier frequency, I was able to work out the stop and start frequency for each channel. From section 6.1.6, the channel bandwidth is 2MHz. Adding 500KHz to the filter bandwidth to allow for filter rolloff and to ensure that no part of the signal is lost gives a workable filter bandwidth of 2.5MHz.

I then derived a formula for the start and stop frequencies, centred around the carrier frequency. These two formulas are given:

$$f_{START} = f_C - \frac{f_{BW}}{2}$$

$$f_{STOP} = f_C + \frac{f_{BW}}{2}$$

Where  $f_{START}$  and  $f_{STOP}$  are the stop and start frequencies of the bandpass filter, respectively. Remembering that we have down-converted to an IF of 20MHz, we are able to combine the above formulae with the formula for the carrier frequency, to obtain the following two formulae:

$$f_{CARRIER} = 5(k - 11) + 20MHz \quad (\text{adapted for down-converted IF})$$

$$f_{START} = 5(k - 11) - \frac{f_{BW}}{2} + 20MHz$$

$$f_{STOP} = 5(k - 11) + \frac{f_{BW}}{2} + 20MHz$$

Where  $k$  is channel number in the range of 11...26. From these two formulas, it is possible to calculate the filter stop and start frequencies ( $f_{START}$  and  $f_{STOP}$ ) from channel number,  $k$ .

In MATLAB, a linear phase Finite Impulse Response (FIR) filter can be made using the MATLAB function `fir1`. `fir1` creates a normalised filter so that the passband response is 0dB. By default, a Hamming-window based linear phase filter is created<sup>[16]</sup>. This is explained in more detail in the testing of the digital filter, section 10.1.2.

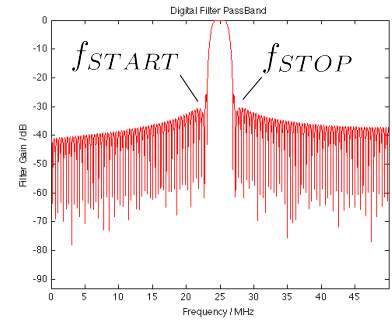


Illustration 15: Passband

## 10.2 Application

Illustration 16 shows a block diagram of the internal workings of my MATLAB functions.

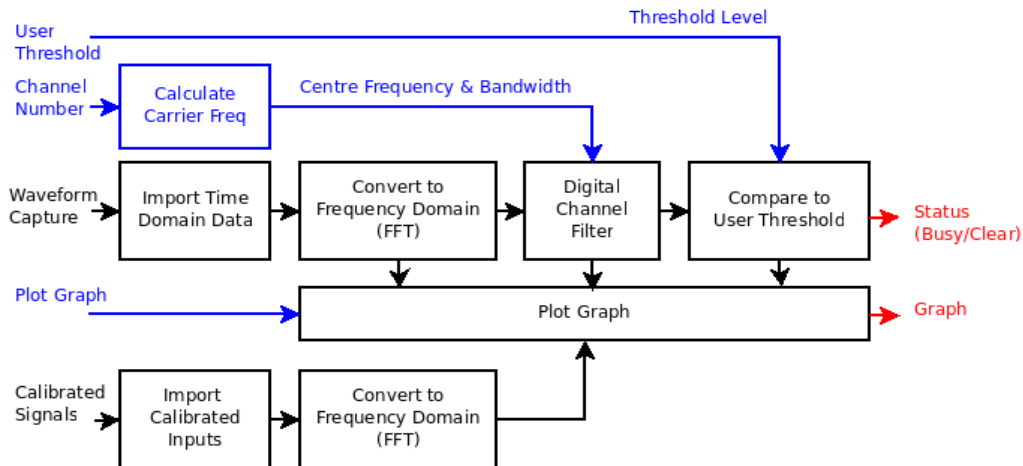


Illustration 16: MATLAB Function Block Diagram

User supplied inputs are marked with blue arrows, and outputs created by the function are marked with red arrows.

Appendix B shows the MATLAB source code for the functions written. These functions take the test captures from section 9.3 and are able to determine which channels are clear and which are busy, based on certain user input parameters. The bullet points below describe the signal flow through my `zigbee` function:

- The function is called with parameters specifying the user threshold (minimum signal level to be regarded as busy), channel number (for single channel analysis) and graph plotting (to enable a behind-the-scenes view). The user must also specify the input waveform (as an array of samples) and sample rate (in Hz).
- The channel number is checked for validity and converted to carrier frequency and filter start/stop frequencies.
- Input waveform is converted to frequency domain by means of FFT. This is then shifted to represent the continuous-time transform, using the `fftshift` function, and the magnitude calculated using the `abs` function.
- A digital filter is then created from the `fir1` function which takes normalised start and stop parameters. `freqz` and `abs` are used to get the frequency response of the normalised digital filter.
- The filter is then applied by means of multiplying the frequency domain passband of the filter with the frequency domain of the signal. This is convolution in the time domain.
- Data for the calibration signal of 0dBm is loaded from a `.mat` file. These powers are also transformed into the frequency domain, and then their peak value plotted for all frequencies. This provides a fixed line of known power.
- If the user requests a graph is plotted:
  - All of the remaining calibration power levels are imported, converted, and plotted at maximum for all frequencies. This gives us a calibrated scale. These scalings are used to calibrate the *y*-axis of the graph as power in dB.
  - A figure is initialised and split into two vertically stacked graphs using the `subplot` function.
  - The top graph shows the raw input data and the bottom graph shows the filtered data.
  - Graphs are scaled with the power calibration levels.
  - Axes titles and a graph title is applied to the figure.
- A comparison is made between the maximum power of the filtered signal and the user input threshold. If the channel power exceeds the user threshold, then the channel is regarded as busy. If not, then the channel is regarded as clear.
- Function ends.

To scan all channels, the `scan` function calls the `zigbee` function recursively in a `for` loop. Again, the function outline is given in bullet points below:

- A `for` loop is set up to run from channels 11 to 15 (test channels).
- The `zigbee` function (described above) is called repetitively with an incremented channel number.
- The return value is analysed and the result presented to the user in a tidy way.

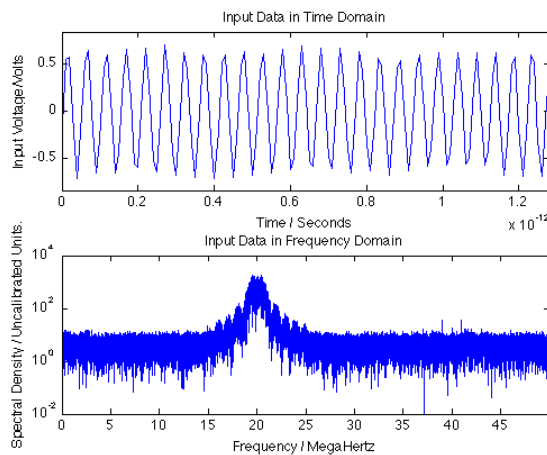


### 10.3 Testing

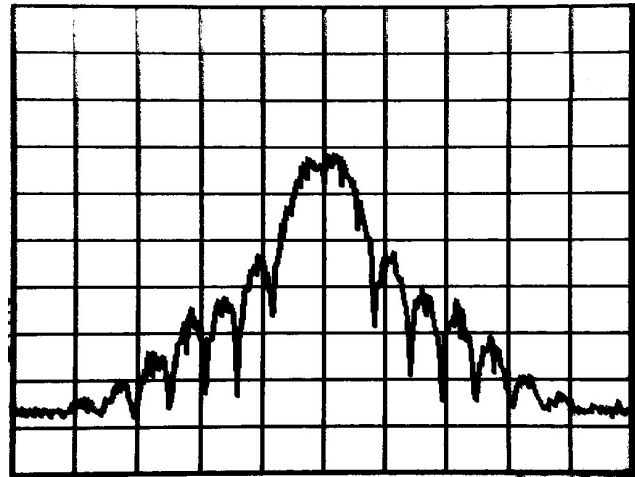
To ensure the functions operate as intended, tests had to be carried out. Functionality is broken into separate components each of which are tested separately. Each section below shows the testing.

#### 10.3.1 Time to Frequency Domain

Illustration 17 shows channel 11 in both the time and frequency domain. The time waveform has been magnified in so that the waveform is visible. This was created using an early revision of the code, and so the  $y$ -axis for the frequency domain is uncalibrated.



(a) MATLAB time & frequency domains after downconversion.



(b) Spectrum analyser view of ChipCon CC2400EB output.

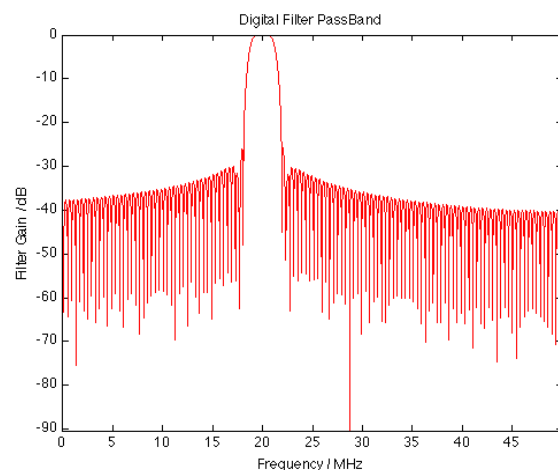
*Illustration 17: Comparison between MATLAB FFT and Spectrum Analyser outputs*

As can be seen from Illustration 17, the frequency domain output from the MATLAB function (a) replicates exactly that of the spectrum analyser (b). We can therefore expect that the MATLAB `fft` function provides reliable results for our needs with the default settings.

#### 10.3.2 Digital Filter

The normalised digital filter used was of order 500 and FIR type. This order was used as it was found to give the best functionality. A FIR filter was used because it is inherently stable, with the output returning to zero one sample after the last input. This is in contrast to infinite impulse response (IIR) filters, which require feedback to stabilise the filter.

Illustration 18 shows the transfer function of the digital filter. This filter was created by my function for use on channel 11, and so we can see that it has the parameters set out by the formulae in section 10.1.2. Due to the Hamming windowing technique used, there is ripple outside of the passband – these ripples are ringing artefacts. These can be eliminated using a rectangular window, but this causes greater distortion to the signal passing through the filter because of Gibbs phenomenon. Gibbs phenomenon causes considerable ripple in the passband as a result of discontinuity brought about by the sudden step in the filter. The Hamming function is one of a variety of

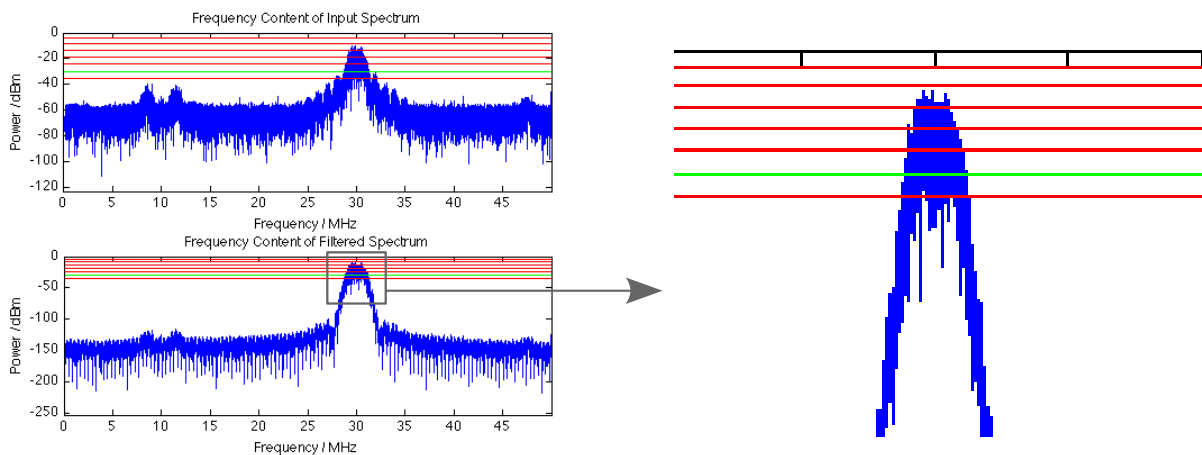


*Illustration 18: Digital Filter Passband*

different windowing techniques used. It reaches a good compromise between roll-off and ringing artefacts.

### 10.3.3 Filtering Channels

Illustration 19 shows an input of channel 13 (top plot), and the same waveform after it has been filtered using the digital filter (bottom plot). It can be seen that much of the noise has been removed from the signal. Notice the  $y$ -axis scale changes between the top and bottom plots. The red lines are the calibrated power levels, starting at 0dBm at -5dBm intervals. The green line is the user input threshold of -30dBm which covers the red line for the same power. We can see, the signal (in blue) exceeds the threshold level (green line) by more than 15dBm, and so the channel is regarded as busy.

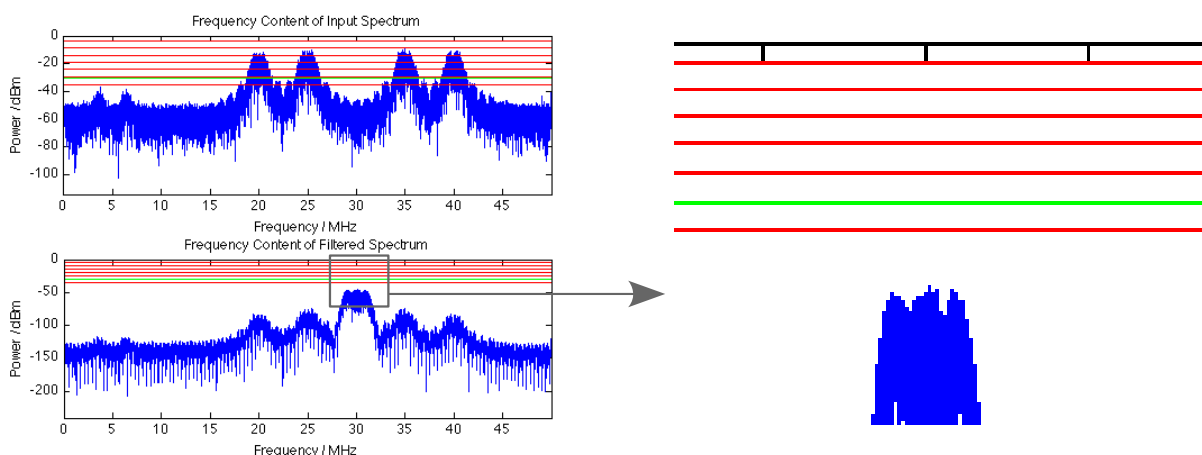


(a) Unfiltered and filtered data from channel 13

(b) Enlargement of filtered data.

*Illustration 19: Unfiltered and Filtered Signals showing channel 13*

Illustration 20 shows similar to Illustration 19 above. However, this time channels 11, 12, 14 and 15 are present while the filter is tuned to channel 13, which is not present. Here we see that, although there is some signal from the sidebands of channel 12 and 14, there is not enough signal present to exceed the threshold level and so the channel is regarded as clear.



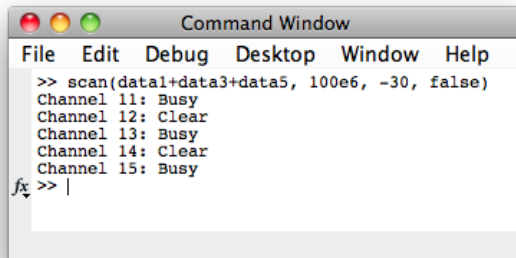
(a) Unfiltered and filtered data from channel 13

(b) Enlargement of filtered data.

*Illustration 20: Unfiltered and Filtered Signals showing channel 11, 12, 14 & 15*

### 10.3.4 Scanning Functionality

A scanning function, `scan`, is provided to allow a user to see what channels are in use for a given input waveform. Although this is not executed concurrency, the FPGA implementation will be able to run the same filter function 16 times (for each channel) in parallel, so this will not be a problem. Therefore this scanning function is provided for the non real-time solution. This method will be changed when the system is implemented in hardware and therefore run in real-time.

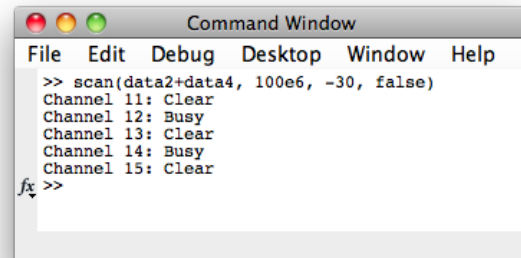


```

Command Window
File Edit Debug Desktop Window Help
>> scan(data1+data3+data5, 100e6, -30, false)
Channel 11: Busy
Channel 12: Clear
Channel 13: Busy
Channel 14: Clear
Channel 15: Busy
fx >> |

```

(a) Channels 11, 13 & 15 Busy



```

Command Window
File Edit Debug Desktop Window Help
>> scan(data2+data4, 100e6, -30, false)
Channel 11: Clear
Channel 12: Busy
Channel 13: Clear
Channel 14: Busy
Channel 15: Clear
fx >>

```

(b) Channels 12 & 14 Busy

#### *Illustration 21: Scanning Function Output*

From Illustration 21 we can see two examples of the `scan` function. (a) shows the scan function with input of channels 11, 13 and 15. MATLAB's command window shows those channels as being busy, while the other channels are regarded as clear. (b) shows the opposite, with channels 12 and 14 being reported as busy and the remaining channels clear.

## **11 Conclusion**

The solution provided in this document meets the aim of the project. The functions written by myself are able to identify which channels are in use for a given time domain capture. The functions provide a base from which to build the FPGA implementation.

Evolving the project from these algorithms in MATLAB to a concurrent hardware based solution would start the move to real-time processing.

The solution here uses only the first five channels of the ZigBee spectrum. This is for convenience and there is no hard-coded limiting factor. Extending the code to support all 16 channels on the 2.45GHz PHY would require a few simple changes.

The functions are not capable of distinguishing between ZigBee data and any other signal in the ISM band – they are, for example, unable to tell the difference between Bluetooth, Wireless LAN and ZigBee.

The project did not develop as quickly as I had hoped or liked, due to my lack of knowledge in the area. I found myself spending long periods of time familiarising myself with software such as MATLAB and Simulink. These are widely used in industry and so a good knowledge of them will prove to be very useful as I progress in my career.

I also learned a lot regarding the use of test equipment such as signal generators, spectrum analysers, and vector network analysers. Again, these are commonly used in RF engineering and so a good knowledge of their usage is important.

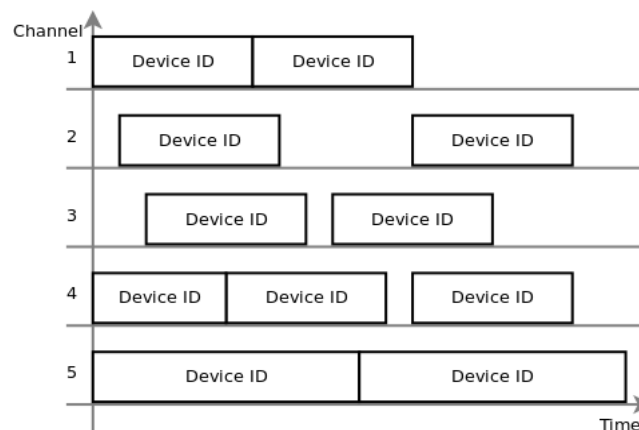
Extensions to this project are discussed in the Future Work chapter, following this.

## 12 Future Work

This project has great scope for extension. It is a complex project with many different parts, all of which must be fine tuned before the overall system will work as desired.

Persons wishing to further this work will need to have a good understanding of digital signal processing. Competency in MathWork's MATLAB and Simulink would also be of great benefit. For the real-time evolution to FPGA, a knowledge of FPGAs and familiarity with Xilinx development tools would greatly benefit the project.

Extending the project to decode frame headers would be the next obvious step, as it would add a lot of information to the system. It was suggested that a real-time user interface be developed such that data frames appear as blocks on the relevant channel showing the source device's ID. Illustration 22 shows this.



*Illustration 22: Proposed Interface*

Extending the device to decode the header would require a phase locked loop locked (PLL) to the carrier of the incoming signal. This would provide a carrier of the exact same frequency but without modulation which could be used to detect phase changes in the modulated signal. Considerations of phase noise and jitter must be taken into account if this approach is used.

An RF front end would need to be developed before the system could be put to proper use. This would involve the design of a low noise amplifier and tuned circuits to filter out the ISM band and amplify it. Commercially available filters, such as surface acoustic wave (SAW) filters may be suitable as they are cheap to purchase and are already tuned.

## 13 References

- [1] ZigBee Alliance, “ZigBee Alliance” Homepage, retrieved 24/11/2009  
(<http://www.zigbee.org>)
- [2] Mikhail Galeev, “Home networking with Zigbee,” Embedded.com Technical Insights, retrieved 24/11/2009  
(<http://www.embedded.com/columns/technicalinsights/18902431>)
- [3] “IEEE Standard 802.15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs)”, 2003.  
(<http://standards.ieee.org/getieee802/download/802.15.4-2003.pdf>)
- [4] ZigBee Alliance, “ZigBee Specification”, Document 053474r17, Jan 2008.  
(<http://www.zigbee.org/ZigBeeSpecificationDownloadRequest/tabid/311/Default.aspx>)
- [5] Rosemount Wireless Instrumentation, “Self-Organizing Networks: Wireless Topologies for In-Plant Applications”, Technical Note, Document 00840-0200-4180/Rev AA, 2007.  
(<http://www.emersonprocess.com/Rosemount/document/notes/00840-0200-4180.pdf>)
- [6] UMTSWorld.com, “CDMA Overview”  
(<http://www.umtsworld.com/technology/cdmabasics.htm>)
- [7] Liam Devlin, “Mixers”  
(<http://www.plextek.com/papers/mixers2.pdf>)
- [8] Mini Circuits, “Coaxial Frequency Mixer”, Mini Circuits ZFM-2000+ Datasheet  
(<http://www.minicircuits.com/pdfs/ZFM-2000+.pdf>)
- [9] John Watkinson, “The Art of Digital Audio”, ISBN: 0240513207
- [10] Janine Sullivan Love, “RF Front-end: World Class Design”, Newnes, ISBN: 9781856176224
- [11] Khaled Shuaib, Maryam Alnuaimi, Mohamed Boulmalf, Imad Jawhar, Farag Sallabi, Abderrahmane Lakas, “Performance Evaluation of IEEE 802.15.4: Experimental and Simulation Results”, JOURNAL OF COMMUNICATIONS, VOL. 2, NO. 4, JUNE 2007
- [12] *Unknown Author*, “Nallatech FPGA Board and System Generator Tutorial”, supplied by Dr John Mitchell, Dept of Electronic & Electrical Engineering, University College London
- [13] Frank H. P. Fitzek, Marcos D. Katz, “Cognitive Wireless Networks”, ISBN: 9781402059780
- [14] The MathWorks<sup>Inc</sup>, “Discrete Fourier Transform – MATLAB”, retrieved 21/02/2010,  
(<http://www.mathworks.com/access/helpdesk/help/techdoc/ref/fft.html>)
- [15] Matteo Frigo, Steven G. Johnson, “FFTW Home Page”, retrieved 21/02/2010.  
(<http://www.fftw.org/>)
- [16] The MathWorks<sup>Inc</sup>, “Window-based finite impulse response filter design – MATLAB”, retrieved 22/03/2010  
(<http://www.mathworks.com/access/helpdesk/help/toolbox/signal/fir1.html>)
- [17] T Bokareva, W Hu, S Kanhere, B Ristic, N Gordon, T Bessell M Rutten, S Jha, “Wireless Sensor Networks for Battlefield Surveillance”, Proceedings of The Land Warfare Conference, 2006. (<http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.88.8223>)

## Appendix A Graphs & Tables

This appendix contains test results, graphs, and other information not directly required in the main report, but are included for completeness.

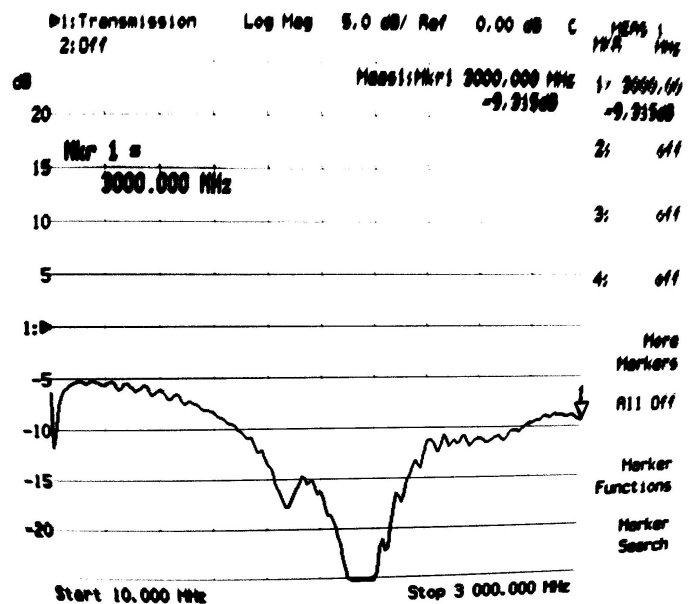
Local Osc		RF Input		IF Output	
Freq/	Level	Freq/	Level	Freq/	Level
M Hz	dBm	M Hz	dBm	M Hz	dBm
0	7	200	7	200	2.70
50	7	250	7	200	-15.89
100	7	300	7	200	-13.48
150	7	350	7	200	-18.50
200	7	400	7	200	-4.06
250	7	450	7	200	-17.80
300	7	500	7	200	-17.00
350	7	550	7	200	-19.04
400	7	600	7	200	-18.72
450	7	650	7	200	-18.02
500	7	700	7	200	-19.91
550	7	750	7	200	-22.74
600	7	800	7	200	-22.20
650	7	850	7	200	-22.37
700	7	900	7	200	-22.25
750	7	950	7	200	-24.18
800	7	1000	7	200	-25.02
850	7	1050	7	200	-26.43
900	7	1100	7	200	-30.95
950	7	1150	7	200	-33.07
1000	7	1200	7	200	-35.32
1050	7	1250	7	200	-38.80
1100	7	1300	7	200	-35.72
1150	7	1350	7	200	-32.46
1200	7	1400	7	200	-28.88
1250	7	1450	7	200	-27.85
1300	7	1500	7	200	-26.07
1350	7	1550	7	200	-26.07
1400	7	1600	7	200	-23.82
1450	7	1650	7	200	-23.08
1500	7	1700	7	200	-21.77
1550	7	1750	7	200	-22.35
1600	7	1800	7	200	-20.77
1650	7	1850	7	200	-20.06
1700	7	1900	7	200	-18.73
1750	7	1950	7	200	-18.82
1800	7	2000	7	200	-18.19
1850	7	2050	7	200	-17.38
1900	7	2100	7	200	-17.24
1950	7	2150	7	200	-17.77
2000	7	2200	7	200	-17.09
2050	7	2250	7	200	-16.52
2100	7	2300	7	200	-15.53
2150	7	2350	7	200	-17.75
2200	7	2400	7	200	-17.24
2250	7	2450	7	200	-17.13
2300	7	2500	7	200	-17.94

Table 2: ZFM2000 Frequency Response Test results

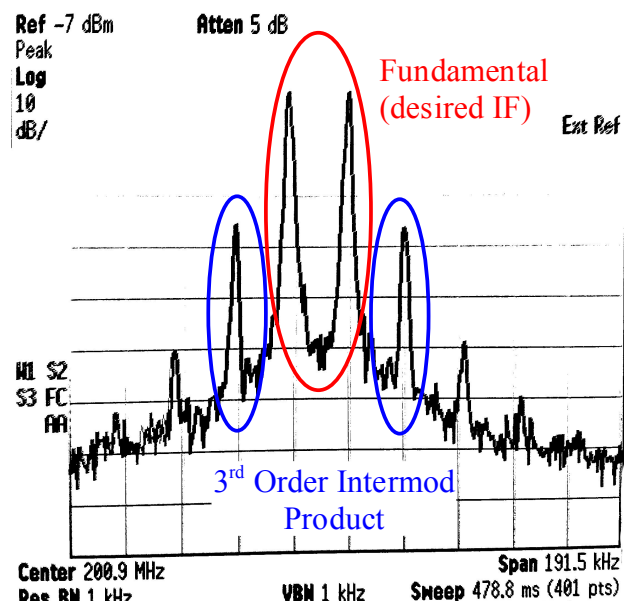
Input	1st Order	3rd Order
7	-14.36	-29.15
6	-14.68	-29.28
5	-14.83	-30.79
4	-15.25	-32.14
3	-15.69	-34.22
2	-15.94	-37.62
1	-16.57	-39.43
0	-17.07	-43.25
-1	-17.91	-46.19
-2	-18.64	-50.42
-3	-19.51	-53.7
-4	-20.89	-58.94
-5	-21.87	-60.78
-7	-23.51	-68.81
-10	-26.22	-78.55

— LO Breakthrough

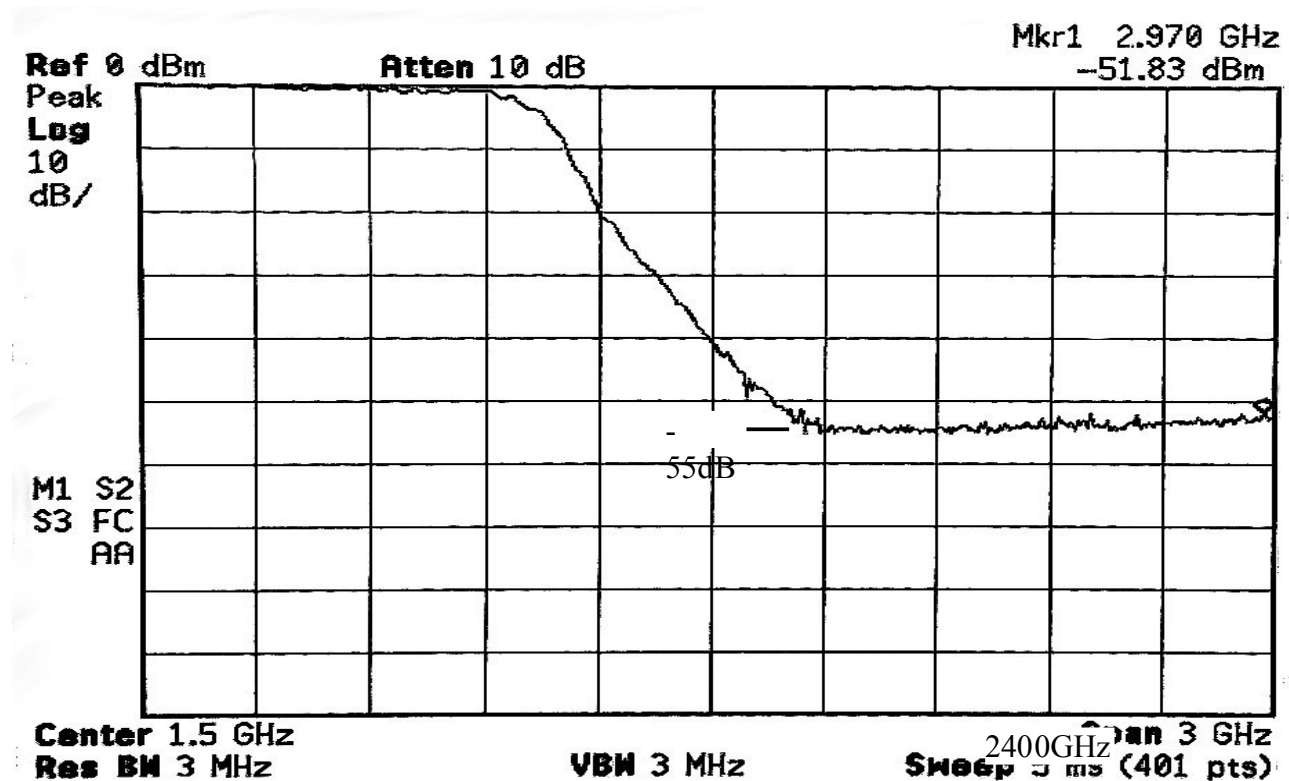
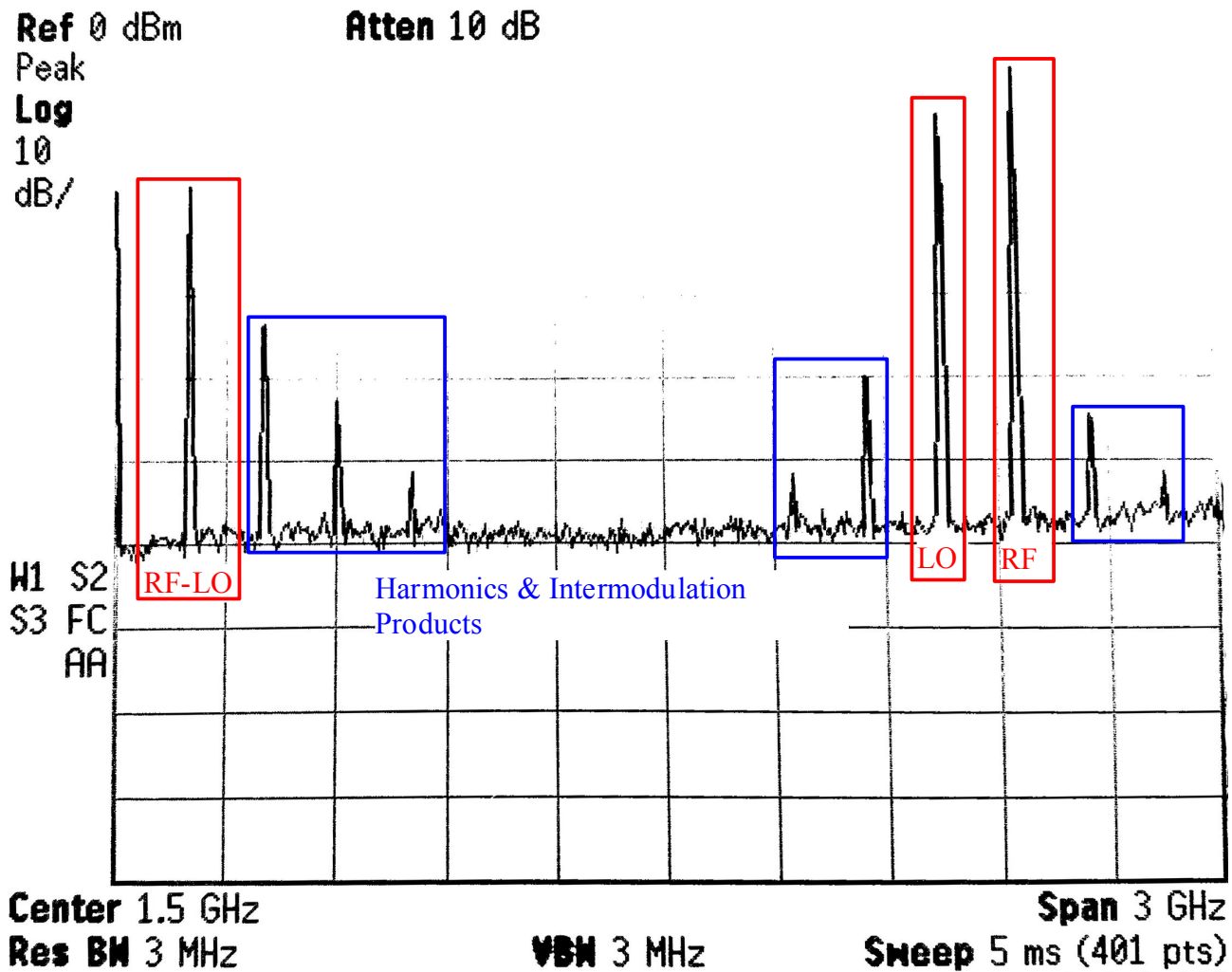
Table 1: 3rd Order Intermod. Test Data



Graph 3: ZFM-2000 Frequency Response, as measured by a VNA.



Graph 4: Two-Tones & Intermodulation Products



Graph 6: HP 1GHz LPF Anti-Alias/Image Filter



## Appendix B MATLAB Source Code

### *fft\_data*

The `fft_data` function takes one argument, `samples_in`, which is an array of time-domain samples. There is no way to calibrate the axes, as no information about time is input.

From this, the frequency domain is calculated and plotted. Illustration 17a from section 10.3.1 was created using this function.

```
function [] = fft_data(samples_in)
% Show FFT of Input
%
% George Smart
% Department of Electronic & Electrical Engineering
% University College London

% Setup Parameters Below
close all;
data = samples_in;
fSampling = 100e6; % in samples/second
Samples = 65536; % total number of samples

% Script Below %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Compute Time & Frequency Vectors
tIncrement = 1/fSampling; % sample time is reciprocal of sampling freq
tVector=[0:tIncrement:(tIncrement*(Samples-1))];
fVector=[(-fSampling/2):(fSampling/Samples):((fSampling-1)/2)];

% Compute Fast-Fourier Transform
dataFFT = fft(data(:,2)); % do the fft
dataFFT = fftshift(dataFFT); % shift the result, to resemble continuous
mag_dataFFT = abs(dataFFT); % magnitude of FFT, to remove complex parts

% Render Graphs
subplot(2, 1, 1)
plot(tVector/1e6, (data(:,2))); % plot fft of unfiltered data
axis([0 0.65536e-9 (min(data(:,2))+0.1*min(data(:,2))) (max(data(:,2))
+0.1*max(data(:,2))))]; % scale the graph nicely :)
title('Input Data in Time Domain'); % "Always title graphs"
xlabel('Time / Seconds'); % "... and label axis"
ylabel('Input Voltage/Volts');

subplot(2, 1, 2)
semilogy(fVector/1e6, (mag_dataFFT)); % plot fft of unfiltered data
axis([0 max(fVector)/1e6 10e-3 10e3]); % scale the graph nicely :)
title('Input Data in Frequency Domain'); % "Always title graphs"
xlabel('Frequency / MegaHertz'); % "... and label axis"
ylabel('Spectral Density / Uncalibrated Units.');
```

% End of Script (EOF)  
end

## zigbee

This zigbee function is the main function of those written by myself. It takes 5 arguments, listed below along with their definitions:

- waveform            input array of time data from the sampled waveform.
- fSample            sample rate frequency, in Hz.
- channel            channel number to tune filter to, in range 11 ... 15.
- squelch            user input threshold level, in dB.
- plotgraph           instruct function to plot graph, true or false.

All of these arguments must be included when the function is called.

```
function [busy] = zigbee(waveform, fSample, channel, squelch, plotgraph)
%
% George Smart
% Department of Electronic & Electrical Engineering
% University College London
%
% What it does:
% This function takes the following variables, and decided if within the
% specified parameters, the channel given is busy.
%
% Inputs:
%   waveform      (an array of sample values)
%   fSample       (Sampling frequency in Hz)
%   channel       (ZigBee channel, 11-26)
%   squelch       (Signal Threshold for Busy)
%   plotgraph     (true if the user would like a graph plotted showing details,
%                  false if not)
%
% Outputs:
%   busy          (1=busy, 0=clear)
%
%% Preliminary Tests
% Check for sensible channel number
% ZigBee channels are in the range 11 to 26 - This Script works only for
% channels 11-15, for proof of concept. If sampling was adjusted it would
% cope with all channels easily.
if ((channel > 15) || (channel < 11))
    error('zigbee:InvalidChannel','Valid Channels Range from 11 to 15 for this example
function')
end

% Discard Sample Time data (throw away column 1/keep column 2)
waveform=waveform(:,2);

%% Compute Frequency Ranges for Filtering
% Calculate the IF Frequency for given channel (in Hz)
fCarrier = 20e6 + 5e6*(channel - 11);
% Set Filter Bandwidth in Hz
FilterBandwidth = 2.5e6;
% Calculate the number of samples
Samples=length(waveform);
% Calculate Frequency Vector - Defines the frequency each sample in the FFT
% relates to.
fVector=(-fSample/2):(fSample/Samples):((fSample-1)/2)];
% Calculate Start and Stop Frequencies for the filter, in both Hz and
% Normalised (from 0 to 1).
BandStart = fCarrier-(FilterBandwidth/2);      % Hz
BandStop = fCarrier+(FilterBandwidth/2);
DigitalStart = ((BandStart/max(fVector-1))/2)+0.5; % normalised
DigitalStop = ((BandStop/max(fVector-1))/2)+0.5;

%% Compute Fast-Fourier Transform
% Execute a Fast Fourier Transform on input waveform
dataFFT = fft(waveform);
% Shift the resulting FFT from 0-2Fs to -Fs/2 to Fs/2, to resemble
% continuous transform results.
dataFFT = fftshift(dataFFT);
% Find the magnitude of the FFT
dataFFT = abs(dataFFT);

%% Create and Filter Signal Based on Calculations
% Define a normalised 500 order finite impulse response filter
PassBand = firl(500, [DigitalStart DigitalStop]);
% Determine the filters transfer function, H(e^jw)
PassBand = freqz(PassBand,1,Samples);
% Find the magnitude of the transfer function
PassBand = abs(PassBand);
% Multiply input data by transfer function, element-wise, to gain filtered
% data.
```

```

dataFIR = dataFFT .* PassBand;

%% Scale data for plotting.
Scale.Ref = 0;
Scale.Div = 0.49;
Datum.Ref = 12;
Datum.Div = 1;

% Load data from file
CalData = load('CalibrationData.mat'); % Load cal data into structure

%sine.a level
sine.a = ones(size(dataFFT)) .* (10 .*
log10(max(abs(fftfshift(fft(CalData.sine_00dBm(:,2)))))/Scale.Div)-Scale.Ref;
YmaxFFT = max(20*log10((dataFFT))); % find maximum y values
YmaxFIR = max(20*log10((dataFIR)));
YminFFT = min(20*log10((dataFFT))); % find minimum y values
YminFIR = min(20*log10((dataFIR)));
squ = ones(size(dataFFT)) .* squelch;
if (sine.a > YmaxFFT), YmaxFFT = sine.a; end % keep 0dBm as Reference
if (sine.a > YmaxFIR), YmaxFIR = sine.a; end

%% Plot Results if Required (plot=true)
if (plotgraph == true)

    % Create a dataset of the right size and level, from each of the calibration values.
    sine.a = ones(size(dataFFT)) .* (10 .*
log10(max(abs(fftfshift(fft(CalData.sine_00dBm(:,2)))))/Scale.Div)-Scale.Ref;
    sine.b = ones(size(dataFFT)) .* (10 .*
log10(max(abs(fftfshift(fft(CalData.sine_05dBm(:,2)))))/Scale.Div)-Scale.Ref;
    sine.c = ones(size(dataFFT)) .* (10 .*
log10(max(abs(fftfshift(fft(CalData.sine_10dBm(:,2)))))/Scale.Div)-Scale.Ref;
    sine.d = ones(size(dataFFT)) .* (10 .*
log10(max(abs(fftfshift(fft(CalData.sine_15dBm(:,2)))))/Scale.Div)-Scale.Ref;
    sine.e = ones(size(dataFFT)) .* (10 .*
log10(max(abs(fftfshift(fft(CalData.sine_20dBm(:,2)))))/Scale.Div)-Scale.Ref;
    sine.f = ones(size(dataFFT)) .* (10 .*
log10(max(abs(fftfshift(fft(CalData.sine_25dBm(:,2)))))/Scale.Div)-Scale.Ref;
    sine.g = ones(size(dataFFT)) .* (10 .*
log10(max(abs(fftfshift(fft(CalData.sine_30dBm(:,2)))))/Scale.Div)-Scale.Ref;
    sine.h = ones(size(dataFFT)) .* (10 .*
log10(max(abs(fftfshift(fft(CalData.sine_35dBm(:,2)))))/Scale.Div)-Scale.Ref;
    % Render Graphs
    % Plot Filter PassBand
    figure
    LogPassBand = 10*log10(PassBand);
    plot(fVector/1e6, LogPassBand, 'r')
    axis([0 max(fVector)/1e6 (min(LogPassBand)-max(LogPassBand)) 0]); % scale the graph
nicely :)
    title('Digital Filter PassBand'); % "Always title graphs"
    xlabel('Frequency / MHz'); % "... and label axis"
    ylabel('Filter Gain / dB');

    % Plot Spectrum, etc.
    figure % Load a new figure
    subplot(2, 1, 1); % First of two vertically stacked graphs
    plot(fVector/1e6, 20*log10(dataFFT)-YmaxFFT+Datum.Ref); % plot fft of unfiltered
data
    hold on; % Plot calibration lines over the FFT
    plot(fVector/1e6, sine.a-YmaxFFT, 'r');
    plot(fVector/1e6, sine.b-YmaxFFT, 'r');
    plot(fVector/1e6, sine.c-YmaxFFT, 'r');
    plot(fVector/1e6, sine.d-YmaxFFT, 'r');
    plot(fVector/1e6, sine.e-YmaxFFT, 'r');
    plot(fVector/1e6, sine.f-YmaxFFT, 'r');
    plot(fVector/1e6, sine.g-YmaxFFT, 'r');
    plot(fVector/1e6, sine.h-YmaxFFT, 'r');
    plot(fVector/1e6, squ, 'g');
    axis([0 max(fVector)/1e6 (YminFFT(1)-YmaxFFT(1)) 0]); % scale the graph nicely :)
    title('Frequency Content of Input Spectrum'); % "Always title graphs"
    xlabel('Frequency / MHz'); % "... and label axis"
    ylabel('Power / dBm');

    subplot(2, 1, 2); % Second of two vertically stacked graphs
data
    plot(fVector/1e6, 20*log10(dataFIR)-YmaxFIR+Datum.Ref); % plot fft of unfiltered
    hold on; % Plot calibration lines over the FFT
    plot(fVector/1e6, sine.a-YmaxFIR, 'r');
    plot(fVector/1e6, sine.b-YmaxFIR, 'r');
    plot(fVector/1e6, sine.c-YmaxFIR, 'r');
    plot(fVector/1e6, sine.d-YmaxFIR, 'r');
    plot(fVector/1e6, sine.e-YmaxFIR, 'r');
    plot(fVector/1e6, sine.f-YmaxFIR, 'r');
    plot(fVector/1e6, sine.g-YmaxFIR, 'r');
    plot(fVector/1e6, sine.h-YmaxFIR, 'r');
    plot(fVector/1e6, squ, 'g');
    axis([0 max(fVector)/1e6 (YminFIR(1)-YmaxFIR(1)) 0]); % scale the graph nicely :)
    title('Frequency Content of Filtered Spectrum'); % "Always title graphs"
    xlabel('Frequency / MHz'); % "... and label axis"
    ylabel('Power / dBm');

```

```
end

%% Analyse Filtered Data
% Find the peak value of the filtered data plot.
maxValue = max(20*log10(dataFIR)-YmaxFIR+Datum.Ref);
% If we exceed the signalThreshold, channel is busy, else, it isnt!
if (maxValue > squelch)
    busy = 1;
else
    busy = 0;
end
end
```

## scan

The scan function takes 4 arguments; waveform, fSample, squelch, and plotgraph. The channel argument is swept automatically by the function.

```
function scan(waveform, fSample, squelch, plotgraph)
%
% George Smart
% Department of Electronic & Electrical Engineering
% University College London
%
% What it does:
% This function takes the following variables, and prints out if each
% channel is either clear or busy.
%
% Inputs:
%   waveform (an array of sample values)
%   fSample  (Sampling frequency in Hz)
%   squelch  (Signal Threshold for Busy)
%% For Loop
for channel = 1:15
    if zigbee(waveform, fSample, channel, squelch, plotgraph)
        fprintf('Channel %i: Busy\n', channel)
    else
        fprintf('Channel %i: Clear\n', channel)
    end
end
```