THE UNIVERSITY OF MELBOURNE

Capstone Project

Department of Electrical and Electronic Engineering

Final Report

MSP Telecommunications Team

Project Title:	MSP Telecommunications Team
Identifier:	SS5
Date:	June 15, 2016
Student Workers:	Joseph McKee, 541124
	Fan Ren, 668268
	Callum Maltby, 539190
	Kainan Ma, 672453
Academic Supervisor:	Professor Stan Skafidas
	Professor Jonathan Manton
	Associate Professor Margreta Kuijper
Academic Examiner:	Associate Professor Margreta Kuijper
Version:	v1.0 Final, June 15, 2016

Executive Summary

This Capstone project begins the design of a novel telecommunications system for the CubeSat under construction by the Melbourne Space Program (MSP). The project produced a flexible and adaptive system that is capable of implementing different modulation formats, operating at different centre frequencies, with adaptive bandwidths for variable deployment and orbital altitude. These qualities ensure that the system achieves its stated purpose of reliably transmitting as much data between the satellite and Earth as possible. With such a broad scope and relatively few precise specifications, it was left to the authors to make tradeoffs and optimise the system as they saw fit.

A new type of inflatable antenna was designed to provide 13dBi of antenna gain and an increase in data rate of up to 150 times over comparable technologies. Much of the difficulty inherent to this project resulted from that fact: while the power constraints are similar to comparable systems, the computational load is significantly higher. This report conducts a power budget and link budget feasibility study and describes the design methods and findings of the project. The selection of OFDM and its associated advantages is explored, plus the method for selecting the baseband processor (BBP) and the analog front end (AFE). Central to this project was the design and implementation of the baseband operations on a TS201S TigerSHARC DSP in addition to the analog carrier-band operations conducted on the AFE. Additional work on antenna design, legal considerations and project integration is also included. Future work is also described to provide a rough scope for subsequent Capstone or FYP members.

This report concludes that meeting the desired system specification is feasible and describes an evaluation board representation of the system for demonstration.

Acknowledgements

The authors would like to acknowledge the contributions of Professor Stan Skafidas, Professor Jonathan Manton and Associate Professor Margreta Kuijper to this project. Thanks must be extended to our industry advisor, Mr Les Davey, who has provided enormous guidance on working with the ACMA. Special thanks must also go to Analog Devices and Xilinx for their donation of all our evaluation boards.

In addition, the authors would like to thank the students from the 2015 and 2016 University of Melbourne Electrical & Electronic Engineering Capstone Projects, and members of the Melbourne Space Program (MSP). Without the support from such a committed group of people, or its exciting application, this project would not have been possible.

Abbreviations

ACMA	-	Australian Communications and Media Authority
ACS	-	Add, Compare, Select
ADC	-	Analog to Digital Converter
AGC	-	Automatic Gain Control
AFE	-	Analog Front End
ALU	-	Arithmetic Logic Unit
BBP	-	Baseband Processor
BER	-	Bit Error Rate
BPSK	-	Binary Phase Shift Keying
CFOE	-	Coarse Frequency Offset Estimation
CLU	-	Communications Logic Unit
COTS	-	Commercial, Off The Shelf
CP	-	Cyclic Prefix
DSP	-	Digital Signal Processor
FEQ	-	Frequency Equalisation
FPGA	-	Field Programmable Gate Array
FMC	-	FPGA Mezzanine Card
FFT	-	Fast Fourier Transform
HPA	-	High Power Amplifier
IALU	-	Integer Arithmetic Logic Unit
ICI	-	Inter Subcarrier Interference
ISS	-	International Space Station
ITU	-	International Telecommunication Union
ITU-RR	-	ITU Radio Regulations
IQ	-	In-phase and Quadrature
LNA	-	Low Noise Amplifier
LVDS	-	Low Voltage Differential Signalling
PSAM	-	Pilot Symbol Assisted Modulation
OFDM	-	Orthogonal Frequency Division Multiplexing
SDR	-	Software Defined radio
SMA	-	SubMiniature version A
SNR	-	Signal-to-noise Ratio
THR	-	Trellis History Register
TT&C	-	Telemetry, Tracking and Command
VDSP++	-	Analog Device's DSP Simulation Environment

Contents

List of Figures

List of Tables

1 Project Overview

1.1 System Purpose

In late 2014, a group of engineering students came together to form the University of Melbourne Space Program. Today, this group has expanded to over 100 undergraduate and masters students dedicated to a single purpose: to launch a satellite into space 600km above the surface of the earth. In order to accelerate development time, the satellite will be built to a CubeSat form-factor, where the satellite fits within a 100mm cube and weighs approximately 1kg. The students are segmented into teams, each overseeing an element of the design, where the four major categories are flight systems, flight vehicle, operating systems and telecommunications. All members within the this Capstone group are considered senior members of the telecommunications team. Some 2015 Capstone projects have already been conducted on the CubeSat project, the most notable of which designed a control system for the radio-telescope atop Redmond Barry. With issues of spectrum allocation and future satellite goals initially undeveloped, it was imperative that this Capstone designs a flexible telecommunications system, capable of implementing different system modes and modulation schemes, operating at different centre frequencies over different bandwidths while remaining compatible at different orbital altitudes. This is to ensure the reliable transmission of as much data between the satellite and Earth as possible.

Projects in this space are becoming increasingly common in graduate engineering courses around the world. Many nano-satellite missions serve as educational tools; a way for students to apply the theory that they have learnt at university. Many CubeSats communicate using simple high beamwidth, low gain half-wave dipole [15] antenna systems, which often contribute antenna gains of ≈ 1.5dBi. This in turn necessitates slow data transmission speeds, since each data bit must be asserted for a relatively long period in order for the bit energy to rise above the noise floor. Because of this, many CubeSats only achieve maximum data rates of 10kbps, resulting in little over 1MB transmitted per day [16].

The telecommunication system outlined in this report addresses this slow data rate, and radically differs from any system on a comparable satellite. This is due to the inclusion of an inflatable antenna, as shown in Figure 1. Conceptually, sublimating benzoic acid is used to expand a metalized balloon which forms a parabolic reflector to focus the RF power to within a 3dB full-beamwidth of 10° at 5.83GHz. This increases the received power and contributes a satellite antenna gain of 13dBi. The symbol period can be considerably shorter than for existing satellites while still meeting the same signal-to-noise ratio (SNR), which allows the system to transmit at data rates of up to 1.45MBits/sec, dramatically increasing our average data throughput to 237MBytes per day.

This increased data rate places strains on the telecommunication system that comparable systems are not burdened with: the baseband processor (BBP) chip must perform its algorithms significantly faster than similar systems. In order to meet these requirements, a powerful BBP is required: the TigerSHARC ADSP-TS201S is clocked at 500MHz and has an instruction set architecture (ISA) that allows a clock cycle budget to be met. Meeting this

Figure 1: CubeSat with the parabolic reflector rendering [1]

clock cycle budget is critical for real-time system operation.

As such, the deliverable outcome of this project is an evaluation board representation of the final system. The system also has to be capable of operating high data rate modes for the primary and secondary communication systems, and a high reliability mode for telemetry, tracking, and command (TT&C). The chips used will be the same, and the algorithms written will be transferable; the evaluation stage serves as a not only a proof-of-concept, but a development platform for future work. Once this system is demonstrated to be functional, the MSP can take the design and build on it to produce flight-ready PCBs. Combined with the rest of the satellite, these will fill a presently untapped sector of the satellite telecommunications industry. Affordable and low data rate satellites are already available in commercial and educational contexts and expensive satellites with enormous throughput capabilities already exist, but the middle ground is under-resourced. There exist a range of applications which cannot justify purchasing large existing satellites but require a data rate greater than existing CubeSats can provide [16]. These can be as diverse as extending universal internet coverage, real-time imaging and performing scientific measurements for CSIRO.

1.2 Existing CubeSats

There are few student initiatives in Australia that involve ground-up design of a CubeSat, and of those none have the intention of launching the built satellite. However, there are many universities world-wide who have launched CubeSats both as proof-of-concepts and to host payloads. This section will briefly overview a few relevant satellite initiatives.

1.2.1 QB50 nano-satellite initiative

The QB50 mission is the European Union's initiative to launch a fleet of 50 CubeSats from countries world-wide. Australia currently has three universities partaking in the mission, with researchers from the University of South Australia, the University of New South Wales and the University of Sydney each contributing a unique payload that can be integrated into the satellite constellation. The QB50 initiative promotes payloads in atmospheric measurements, bush-fire detection and imaging of the earth's surface.

1.2.2 BLUESat

BLUESat are a University of New South Wales student group who are developing a 2U satellite for self-education purposes. BLUESat have been affiliated with the MSP team and, since MSP's conception, have been very helpful for sharing system insights. BLUESat's CubeSat will not be launched into space, however their current design endorses a low-power communication system with standard dipole antennas for GMSK modulation in the 400MHz amateur satellite band. If BLUESat were to persist with a design for launch, its 2U form factor would lend it to larger payloads that require a relatively low (10kbps) data-rates.

1.2.3 MIT CubeSats

MIT's publication Inflatable Antenna for CubeSat: Fabrication, Deployment and Results of Experimental Tests [17] demonstrates a university with the common pursuit of increasing LEO data-rates and pushing small satellites into higher orbits by developing a novel inflatable antenna. MIT's engineers demonstrate inflatable antenna gains of as high as 15dBi using a frontal feeder geometry for 2U and 3U CubeSats. As mentioned in the paper, this facilitates data-rates in the Mbps range, as opposed to the more standard kbps range. MIT have measured these antenna gains at as low as 2.4GHz. These results give ample motivation for MSP's mission of developing a Cassegrain topology inflatable antenna at 5.6GHz.

1.2.4 NASA Nodes

As a part of NASA's ongoing initiative to give small satellites access to space through facilities on the ISS, they have launched two satellites as a proof-of-concept for future small satellite mesh networking [18]. NASA communicates with one of two independent orbiting satellites J and K, who autonomously decide which is more optimal to relay telecommands and data with the ground station.

1.2.5 Commercial solutions to CubeSat telecommunication

There are many retailers that sell CubeSat specific telecommunication systems for off-the-shelf purchase. These systems often boast low-power requirements, but make trade-offs in achievable data-rates. GOMSpace's AX100 NanoCom [19] communications module offers data-rates from 0.1kbps to 112kbps depending on the SNR offered in the link budget. This module supports FSK/GMSK modulation, Reed-Solomon FEC and a very typical AX.25 (amateur radio standard) data-link layer. MSP's CubeSat is investigating the same data-link protocol, but this has not been the work of this capstone.

1.3 OFDM Overview

Orthogonal Frequency Division Muliplexing (OFDM) as a modulation scheme has experienced a rapid growth in popularity since being used in applications as varied as 4G mobile phones, WiFi and terrestrial digital television (DVB-T).

OFDM is a technique to transmit bits simultaneously over a large number of closely spaced frequencies, known as subcarriers. With OFDM, symbols can be transmitted at a low symbol rate while the total data rate is similar to a single-carrier transmission with the same sub-carrier modulation scheme and bandwidth, since each symbol consists of multiple data bits. Due to the low symbol rate, the interval gain (cyclic prefix) can be used to offer robustness against Inter-Symbol Interference (ISI). Data may be modulated onto each subcarrier using any standard modulation scheme: while Quadrature Amplitude Modulation (QAM) is popular, Phase-Shift Keying (PSK) will be implemented here.

Even though the sidebands from each subcarrier are overlapping, they can still be received without interference and demodulated independently. In order to achieve this, a high degree of orthogonality is required to ensure that, as shown in Figure 2, at the sprectral peak of one subcarrier, all others have zero amplitude. As such, the subcarriers can be closely spaced to provide OFDM's very high spectral efficiency.

However, OFDM systems are very sensitive and vulnerable to carrier offset and drift caused by Doppler shift and frequency offsets. These problems lead to the inter-carrier interference (ICI). Another deficiency of the OFDM system is that it constrains the choice of SNR to a discrete set. The symbol period $T = \frac{N_{SC}+N_{CP}}{B}$, where N_{SC} is

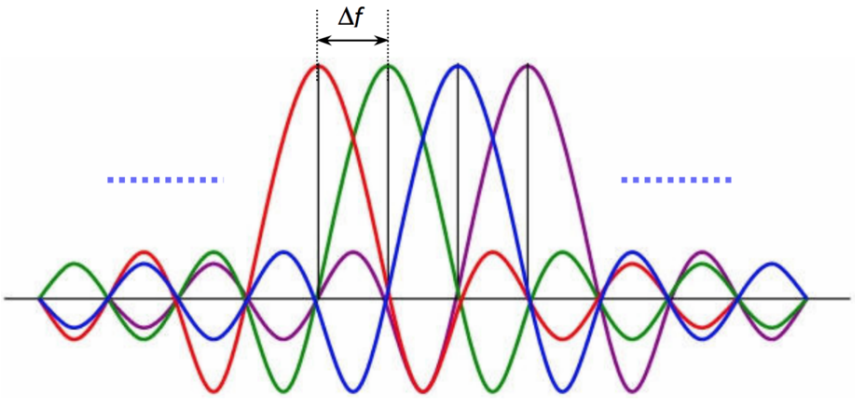


Figure 2: OFDM subcarriers [2]

the FFT size, N_{CP} is the length of cyclic prefix, and B is the bandwidth. Because the FFT size has to be a power of two for efficiencies in the BBP, the symbol period too is discrete. Thus, the range of $SNR = \frac{PT}{N_0}$ choices are discrete, ensuring that the choice of SNR is a step. In this project, the entire transmission window is conducted at a fixed bandwidth.

This OFDM system will address all of the above concerns, as well as provide a basis for a variable rate system for future teams. One possibility, as shown in Figure 3, demonstrates how the received power varies with the satellite's orbiting. The system could promote a variable-bandwidth protocol to achieve constant SNR over the satellite's orbit.

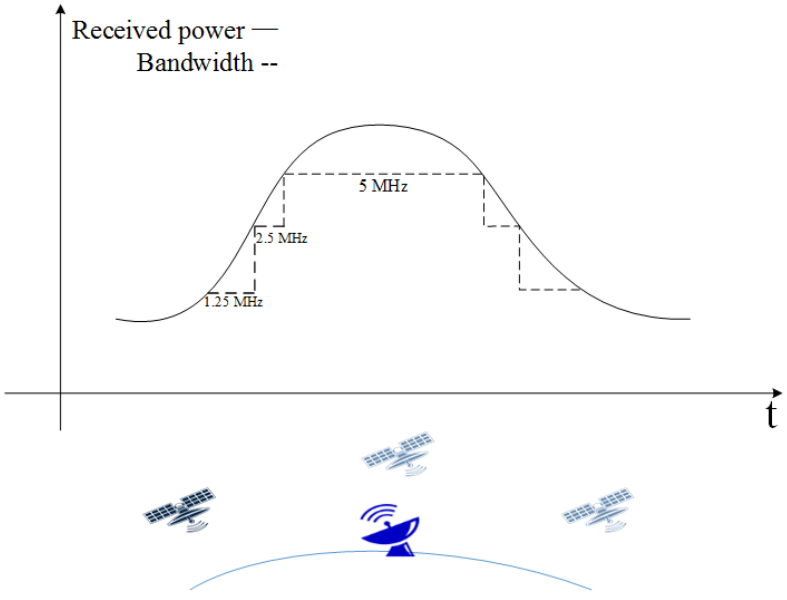


Figure 3: The stepped bandwidths for various received power

A key advantage of OFDM is that it can be efficiently implemented with the FFT algorithm, as shown in Figure 15[1].

[1]See Section 5.4.3

Term	**Value**
Bandwidth	5 MHz
Data rate	1.5 Mb/s
Symbol period (with CP)	16 μs
Symbol period (without CP)	12.8 μs
Guard interval period	3.2 μs
Subcarrier spacings	78.125 kHz
Coding rate	$\frac{1}{2}$
Number of pilot subcarriers	4
Number of data subcarrier	48
Number of subcarriers	52
Number of pad bits	12
FFT size	64
Interleaving depth	67 symbols
Cyclic prefix length	16
Modulation scheme	BPSK
Number of bits per subcarrier	64
Number of complex numbers in one sample	80
Length of a short preamble	10 short symbols
Short preamble period	8 μs
Length of a long preamble	2 long symbols
Long preamble period	8 μs

Table 1: OFDM specification

A OFDM symbol holds 48 information bits: 24 data bits are encoded and interleaved to generate this block size. After that, 4 subcarrier pilots and 12 padding bits are inserted to fill a 64-bit symbol. The 64 bit symbol is mapped to a set of 64-bit complex numbers using BPSK modulation. The main parameters of a OFDM symbol are provided in Table 1.

1.4 Physical Constraints

The telecommunications system has to perform more computations per seconds than most comparable systems, but only has access to a standard wattage. This is limited by the physical size of the cube: five of the six 100mm square sides are covered with PV cells. At the time of writing, these cells can generate 8W when in full sun, some of which goes into recharging the battery to be used while in the shadow of the earth. The specifics of this power system falls within the scope of another Capstone project. The relevant information is the allowable power draw of

the telecommunications system: during transmission it is allowed to draw enough power to enable 1W of RF output power [2]. In order to avoid overshadowing the solar panels, the parabolic reflector design had to be complicated to place a secondary reflector on the Earth side of the satellite.

In order to fully utilise the increased power density afforded by the parabolic reflector, the satellite must be oriented toward Melbourne. This actuation is performed by magnetorquers: loops of current-carrying wire that interact with the magnetic field of the Earth to exert a torque on the satellite. The pointing accuracy of this is dependent on many factors, including the maximum allowable current draw and the sensor's sampling rates. Furthermore, sensors must pass data into algorithms to calculate an estimate of where the satellite thinks it and Melbourne are, and based off these determine the correct amount to rotate the satellite. Since there will be inaccuracies in both of these processes, the telecommunications team applied a constraint that the angular offset between the desired and actual orientation of the satellite must be less than 5°, which is the parabolic reflector's half power beamwidth. This places an upper limit on the directivity of our antennas: if the designed antenna beamwidth was significantly less than 10°, it is likely that the receiver would pick up only negligible amounts of power since the transmitter was oriented elsewhere: to ensure reliable transmission the directivity is upper-bounded. As such, the maximum pointing loss term for the parabolic reflector in the link budget is -3dB.

[2] See Section 2.1

2 Feasibility Study

2.1 Power Budget

The Capstone group has worked closely with the Power team to ensure that sufficient power will be made available to the telecommunications system during transmission, which is the most power intensive period within the orbit. Power is harvested via PV cells, and onboard batteries and supercapacitors can be discharged at times of large power draw. The power draw from the communications board is described in Table 2. Each row refers to a particular voltage rail required by the telecommunications hardware: the AFE, DSP and the power amplifier, and the typical loads are derived from the datasheets for each of these [5, 20]. The Power team has selected buck and boost converters with minimum and maximum load currents, and efficiencies at the given loads.

No specific power amplifier has been selected yet, so the given parameters are an aggregate of possible choices [21, 22]. Each of the power amplifier choices output over 30dBm, satisfying the 1W output power requirement from the link budget[3]. The quoted $P_{out} = 8W$ is the power output from the voltage regulator at the input to the power amplifier. Such amplifiers exhibit poor power efficiency in order to provide the high linearity that OFDM requires [23].

[3] See Section 2.2.2

Power System Output	**Vout [V]**	**Min load [A]**	**Max load [A]**	**Typical load [Ω]**	**Typical efficiency**	**Typical** P_{out}	**Typical** P_{in}	**Typical** P_{loss}	$P_{out,min}$	$P_{out,max}$
AFE_3V3	3.3	0.01	3	0.3	85	0.99	1.165	0.175	0.033	9.9
AFE_1V8	1.8	0.02	3	0.0045	85	0.0081	0.010	0.001	0.036	5.4
AFE_1V3	1.3	0.001	1	0.6	61	0.78	1.279	0.499	0.0013	1.3
DSP_Core	1.05	0.1	6	2.06	85	2.163	2.545	0.382	0.105	6.3
DSP_DRAM	1.5	0.02	3	0.28	85	0.42	0.494	0.074	0.03	4.5
DSP_IO	2.5	0.02	3	0.15	87	0.375	0.431	0.056	0.05	7.5
DSP_ANALOG	1.05	0.001	1	0.02	50	0.021	0.042	0.021	0.00105	1.05
V_AMP	5	0.1	1.5	1.2	75	8	10.667	2.667	0.5	7.5
Communications TOTAL	-	-	-	-	-	12.7571	16.631	3.874	-	-

Table 2: Telecommunications board power budget during RF transmission

2.2 Link Budget

2.2.1 Overview

The link budget is a way of quantifying the power losses during transmission of an RF signal over the channel. In essence, it is a list of different sources of known signal power gain and attenuation, and the margins introduced to account for highly variable or difficult-to-model phenomena. This can be used to determine the expected signal power at the receiver. Work performed in channel characterisation can provide rough bounds for the thermal noise power and varying attenuations that can be expected during transmission. Using this information, a signal to noise ratio (SNR) value can be determined using the discrete OFDM bit period choices to estimate the data rate. Furthermore, a link budget is required for submission of an application to the ITU and ACMA for bandwidth allocation, according to ACMA Regulations [24].

2.2.2 Details

A centre frequency of $f_{centre} = 5.65$GHz and a conservative orbital radius of $R_{orbital} = 600$km are used for the following analysis. The considered sources of gain and loss are described below, and tabulated in Table 3.

Term	**Value for whip antenna (dB)**	**Value for parabolic reflective antenna (dB)**
Path Loss	-163.05	-163.05
Ground station antenna gain (dBi)	43	43
Satellite antenna gain (dBi)	1.5	13
Ground station and satellite pointing loss	-0.5	-3
PAPR reduction	-10	-10
Ground station feeder loss	-1	-1
Satellite feeder loss	0	0
Ionospheric and tropospheric polarization loss margin	-3	-3
Additional ionospheric loss	-1	-1
Tropospheric losses	-0.2	-0.2
Rain losses	-1.2	-1.2
	-135.44dB	-126.44dB

Table 3: Link budget gains and attenuation terms

The largest constituent term to this power difference is the free space loss (FSL). This reflects the fact that areal power density must decrease as the wave propagates outward [25]. The free-space loss is given by Equation 1:

$$FSL = 10log\left(\frac{4\pi r}{\lambda}\right)^2 \tag{1}$$

This is dependent upon the signal frequency since the amount of RF power captured is dependent on the effective aperture of the antenna, which is always less that the perpendicular area swept by the receiving antenna. As the wave frequency increases, diffraction becomes less pronounced, so the antenna will exhibit a decreased sensitivity to waves approaching the receiver circuit at angles further out from the main lobe [25].

It is well understood that the peak to average power ratio (PAPR) is one of the main deficiencies of OFDM. This is a manifestation of the beating phenomena: when multiple sinusoids of similar frequencies add up a time-varying envelope is observed. In order to prevent clipping, the average output power must lie below the maximum output power. There is a probabilistic relationship at play as demonstrated in Figure 4, as the considered PARP increases the likelihood of observing a sample with this power ratio decreases. For the purposes of this report, a 10dB margin can be introduced, which brings the clipping rate below an error rate that the encoding scheme can correct for.

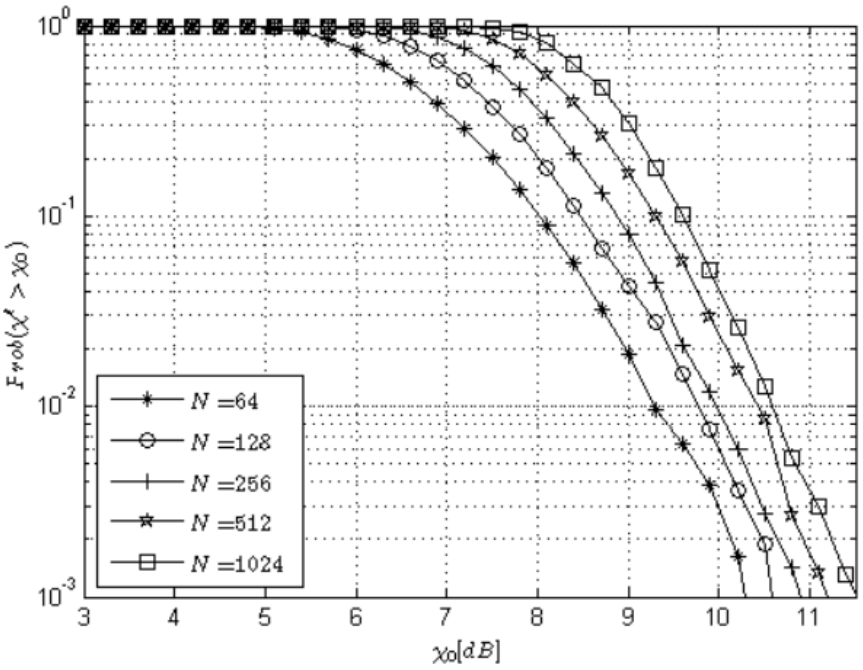


Figure 4: PARP likelihood curves [3]

However, this issue forms a key area of future work for a subsequent Capstone or MSP team. Multiple PARP reduction techniques exist [3], and will be further investigated to reduce the -10dB loss margin introduced in the link budget.

2.2.2.1 Antennas

RF simulation[4] has determined an inflatable antenna and feedhorn pairing at 5.65GHz has a gain of 13dBi with a beamwidth of 10°. If the attitude control system exhibits inaccuracies, or the models to determine where the satellite is located have deficiencies, the satellite will not be pointing directly at the ground station[5]. A maximum pointing loss of -3dB exists for the parabolic reflector and -0.5 for the half-wave dipole, since it has a far broader beamwidth.

Feeder losses are attenuations to power due to resistive and other losses in the wires and conduits connecting the antenna and the analog front end (AFE) chip. On the satellite, these are physically close together, so we can neglect the feeder losses. At the ground station on top of Redmond Barry, these are physically separated by roughly 5m. Since this conduit has not yet been designed, the loss was estimated from a literature review [26] at -1dB.

2.2.2.2 Atmospheric Effects

The selected AFE is capable of transmitting up to 6GHz, but if it proves impossible to access any frequencies below this due to cost constraints or legal difficulties, a secondary mixer can be implemented to bring the final transmission frequency up to microwave range: 24GHz. This brings advantages in directivity, but also significantly increases power losses due to absorption by H_2O and O_2 molecules in the troposphere. At 5.65GHz, these losses were very small, contributing only -0.2dB. However, water molecules exhibit resonance near the microwave range at 22.23GHz [27] as seen in Figure 5: this is why household microwaves output RF energy at this frequency. If transmission occurs at 24GHz, the total tropospheric loss rises to -6.2dB, assuming that the troposphere depth is 20km and the maximum zenith angle is 70°. These are conservative assumptions, since the troposphere generally has a depth of between 7km and 20km. Losses due to absorption by rain were also taken into account via the literature review [26], however this source of attenuation is time-varying. If a particularly dense rainstorm occurs, the transmission window can be skipped since the data at this stage isn't time-critical. Moreover, getting accurate data for the type of rain and clouds commonly observed above Melbourne is difficult, so conservative margins of -1dB at 5.65GHz and -3dB at 24GHz were introduced [26]. Power losses will also occur due to mismatches in circular polarization phase[6]. The worst case is when the received wave and the receiving antennas are offset by 45°, resulting in a power loss of -3dB.

2.2.2.3 Results

Adding all of these together gives the total theoretical difference between transmitted and received signal power, as described in Table 3. Following discussion with other teams in the program, the telecommunications team was

[4]See Section 7.5
[5]See Section 1.4
[6]See Section 7.2

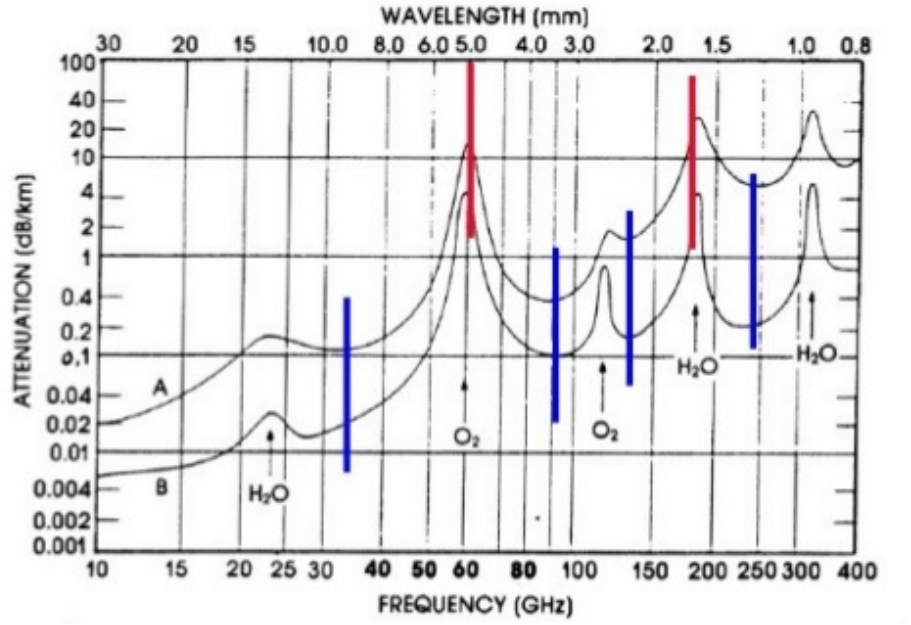


Figure 5: Tropospheric losses due to H_2O and O_2 absorption [4]

granted 1W of RF output power during the period of transmission. This does not include the power taken to run the hardware, or the efficiencies of the power amplifier. By substituting in the expected downlink transmission power of 1W, the following received signal powers are determined, as shown in Table 4. The downlink is chosen for analysis since it is the most power limited, and carries the bulk of the data.

Antenna Type	**Signal Power [W]**
Parabolic reflector	2.266×10^{-13} [W]
Whip	2.853×10^{-14} [W]

Table 4: Downlink received signal powers calculated by the link budget

Since BPSK is implemented on each individual non-null subcarrier, the power requirement of each tone is always constant. If QAM were employed, then some tones would demand a greater proportion of the available output power. However, in this case the transmit power for each tone is simply $\frac{1}{52}$ [W] since 11 bits carry no power to serve as guard-bands and an additional tone is nulled to ensure no DC offset in baseband. This bit power is the quantity of interest for deriving the SNR per bit. This can be converted to a bit energy by multiplying by the symbol period minus the cyclic prefix. In order to do this, the magnitude of noise power must be known and was conservatively chosen to be $150°K$ via a literature review of similar systems [26, 28]. This assumption will be formalised by a non-Capstone member for the MSP team.

2.2.3 Achievable Bit Rates

Using this system noise temperature, the noise power spectral density can be defined. Dividing the energy per tone by this quantity yields the SNR per bit, then a logarithm is taken of this value. A benchmark was set to achieve a BER of $\approx 10^{-5}$, which requires an SNR of between 9 and 10dB. Figure 6 was derived from MATLAB example code:

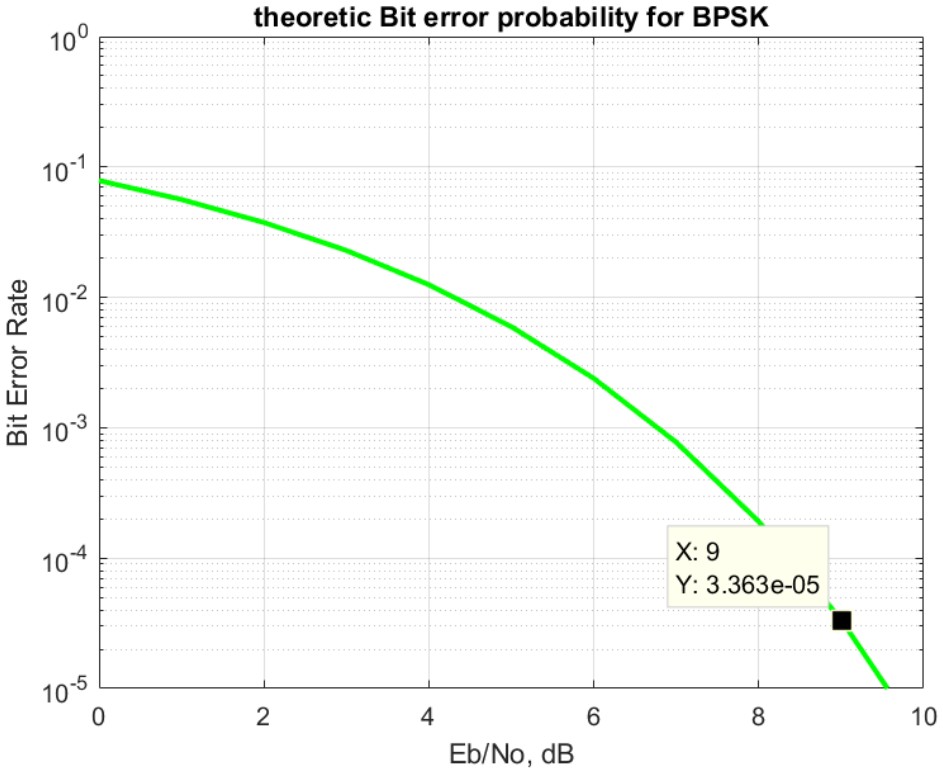


Figure 6: BER vs. SNR curve for BPSK-modulated OFDM, assuming no ICI.

Additionally, to account for any sources of power loss not accounted for in Table 3, an extra margin is required. Numerous literature review placed this additional margin at 20dB, however the attenuations have been well modelled in this case, so this margin has been reduced to around 6.8dB. The parabolic reflector meets these margins while utilising all subcarriers, and so maintains the 5MHz bandwidth. Due to its lower antenna gain, the whip antenna must decrease its bandwidth by a factor of eight to achieve a 9dB increase in signal power. This discrete division is mandated by the FFT algorithm.

Table 5 also includes a coding gain term. This was derived from a literature review of a similar system to be 1.5dB, but analysis on the scheme implemented in this project will be presented in Section 5.2.4, along with further analysis on the relationship between coding gain and BER.

Term	**Value for parabolic reflective antenna (dB)**	**Value for whip antenna (dB)**
Number of active tones per symbol	52	52
Number of data tones per symbol	48	48
Boltzmann's Constant $[m^2kgs^{-2}K^{-1}]$	1.38E-23	1.38E-23
Bandwidth [Hz]	5.00E+06	6.25E+05
Downlink received signal power [W]	2.27E-13	2.85E-14
GS antenna noise temp $[^\circ K]$	150.0	150.0
Downlink noise PSD [W/Hz]	2.07E-21	2.07E-21
Symbol period [secs]	1.60E-05	1.28E-04
Non-CP symbol period [secs]	1.28E-05	1.02E-04
Symbol frequency [Hz]	62500.00	7812.50
Energy per tone [J]	5.58E-20	5.62E-20
Eb/N0	27.0	27.1
Eb/No [dB]	14.31	14.34
Coding gain [dB]	1.5	1.5
Required Eb/N0 for BER = 10^{-5} [dB]	9.0	9.0
Downlink margin [dB]	6.8	6.8
Convolutional rate	0.5	0.5
Data rate [kbit/sec]	1459.1	182.4
Average transmission window [secs/day]	1329.0	1329.0
Decoded data amount [bits]	1.99E+09	2.49E+08
Decoded data amount [Mbytes]	237.6	29.7

Table 5: Data rates from link budget

2.2.4 Future Work

A known disadvantage of OFDM is its high peak-to-average power ratio. The given transmission power of 1W is the maximum that the power team can provide to the telecommunications team when the OFDM subcarriers superimpose. An amplifier with high linearity is desired to output a signal within the dynamic range [23], however on average when the subcarriers aren't superimposing the output power is in an ideal case 17dB less than the worst-case peak power. In order to meet the required SNR margin, further work will be conducted to weigh up reductions in the SNR margin, occupied bandwidth and the amount of allowable clipping. There exist numerous methods to remedy this issue [3, 29].

3 Specifications

3.1 OFDM Simulation

3.1.1 Bit Error Rate in BPSK OFDM

Since the Cyclic Prefix is appended to eliminate the the Inter-Symbol-Interference (ISI) by extended guard intervals, the bit energy spanning over the symbol period is extended to the cyclic prefix as well [30]. Assuming that the symbol period is T_d and the cyclic prefix period is T_{cp}, the symbol energy is given by Equation 2:

$$E_s = \frac{T_d}{T_d + T_{cp}} E_b \tag{2}$$

In addition, a number of tones at the edge of the are not used to transmit valid data due to the roll off the passband spectrum. This means all bit energy is concentrated in the valid tones and the noise are spread out in entire passband. Assume the number of data subcarriers is equal to nDSC, and the number of OFDM tones is equal to nFFT (FFT size). The symbol energy is given by Equation 3:

$$E_s = \frac{nDSC}{nFFT} E_b \tag{3}$$

Combining Equation 2 and Equation 3, the symbol SNR is given by Equation 5:

$$\frac{E_s}{N_0} = \frac{E_b}{N_0} \frac{nDSC}{nFFT} \frac{T_d}{T_d + T_{cp}} \tag{4}$$

Expressed in dB, this becomes:

$$\frac{E_s}{N_0} dB = \frac{E_b}{N_0} dB + 10 \log \frac{nDSC}{nFFT} + 10 \log \frac{T_d}{T_d + T_{cp}} \tag{5}$$

The MATLAB simulation shows that the performance of BPSK in OFDM is nearly the same as the performance for theoretical BPSK, as seen in Figure 7.

3.1.2 Simulink OFDM Model

There are two purpose of the simulation model in which the MATLAB Simulink is used.

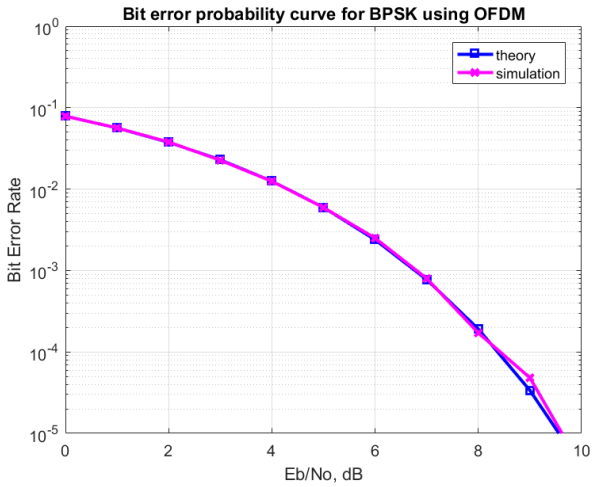


Figure 7: Simulation OFDM BER vs theoretical OFDM BER

- Verify the calculation in the Link Budget
- Provide the prototype for hardware design

The model of the digital part of the OFDM system in Simulink is shown in Figure 8. A random bit sequence is generated as a random integer, and sent through the interleaver, BPSK mapper, OFDM modulator and gain normalization blocks. Bounds for the sampling rate are derived within the link budget, and the channel is classified as AWGN without Rayleigh fading. A 6dB SNR margin is reserved for non-ideal synchronization, quantization, RF model and etc. The signal travels back to the receiver through OFDM demodulator, FEQ, BPSK demodulator and deinterleaver. The bit error rate is counted at the end of the receiver to validate the link budget.

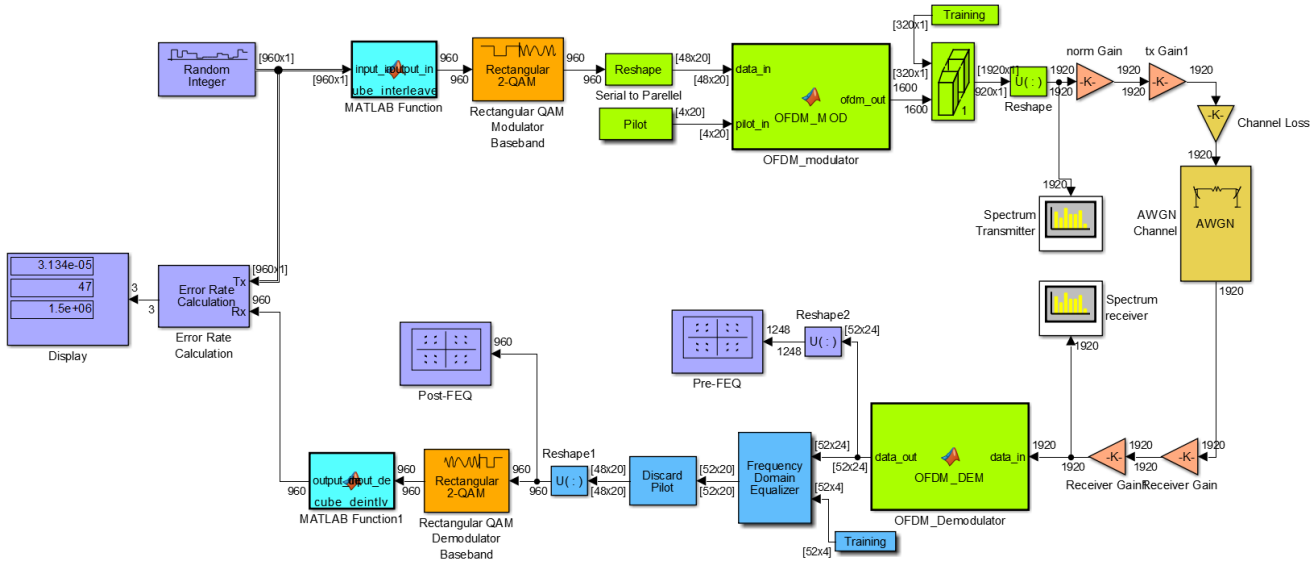
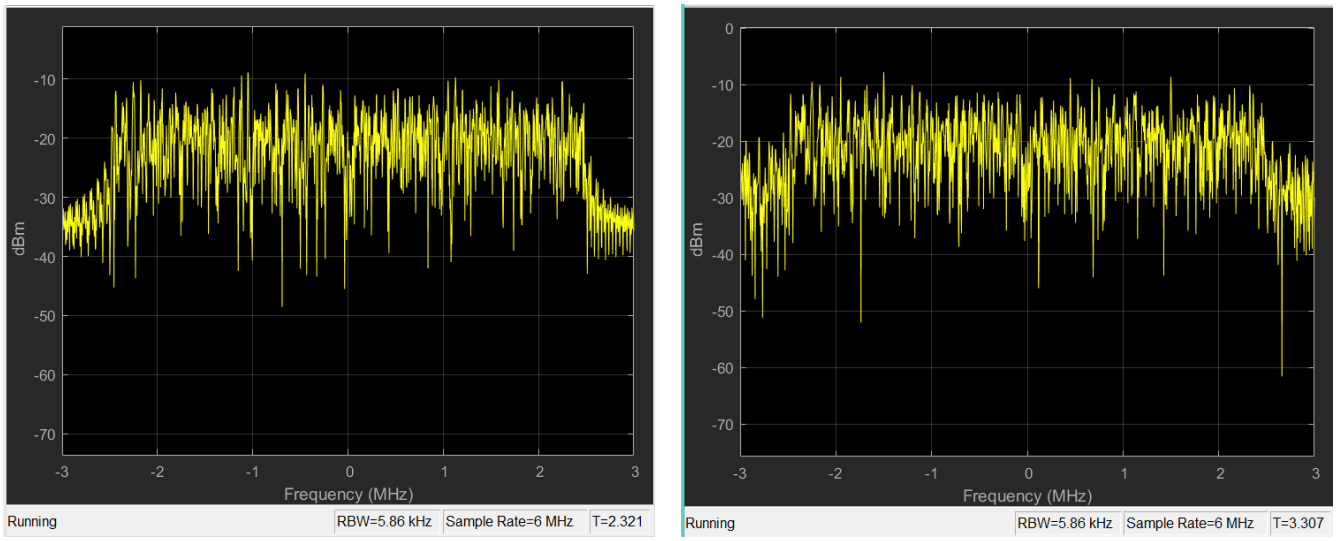


Figure 8: The top layer of MATLAB Simulink model

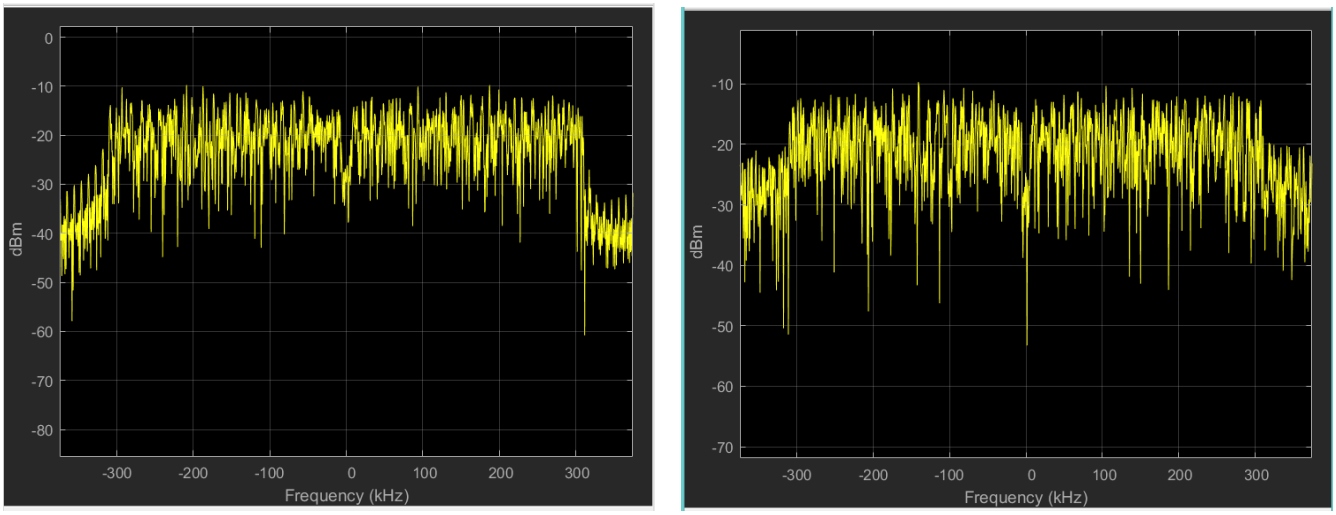
For the 5MHz profile, the transmit and receive spectra are shown in Figure 9. These spectra can be compared against those presented in Section 10.2 to show that the hardware implementation behaves as expected.



(a) Transmitter (b) Receiver

Figure 9: Spectrum of 5MHz OFDM profile

The bit error rate for 9dB bit SNR is approximately 3.1×10^{-5} which agrees with the theoretical result. For the 625kHz profile, the transmit and receive spectra are shown in Figure 10.



(a) Transmitter (b) Receiver

Figure 10: Spectrum of 625kHz OFDM profile

The bit error rate for 9dB bit SNR is approximately 3.03×10^{-5} which agrees with the theoretical result. To conclude, subtracting a 6.8dB SNR margin for 5MHz profile using parabolic antenna and 625KHz profile using whip antenna, the 3×10^{-5} bit error rate is achievable using the extremely conservative 1.5dB estimate for coding gain under AWGN channel, verifying the results from the link budget.

3.2 System Modes

The system has to be able to operate in three different modes: which necessitates the design of such a flexible system. There is a high data rate system employing 5MHz OFDM via the parabolic reflector, a moderate rate but high reliability system using 625kHz OFDM via the crossed half-wave dipoles, and finally a low rate, very high reliability BPSK system for TT&C transmitting via the dipole antennas. This will be used for beaconing during the detumbling phase as the satellite and ground station attempt to lock on to one another. It will also be used to turn off the satellite in the event of harmful interference being caused to surrounding satellites.

3.3 Offline system demonstration

The deliverable outcome for this project will be an offline evaluation board representation of the final flight-ready system. Once this has proven the concept to be valid, work can progress on designing the final PCB. The data path is shown in Figure 11.

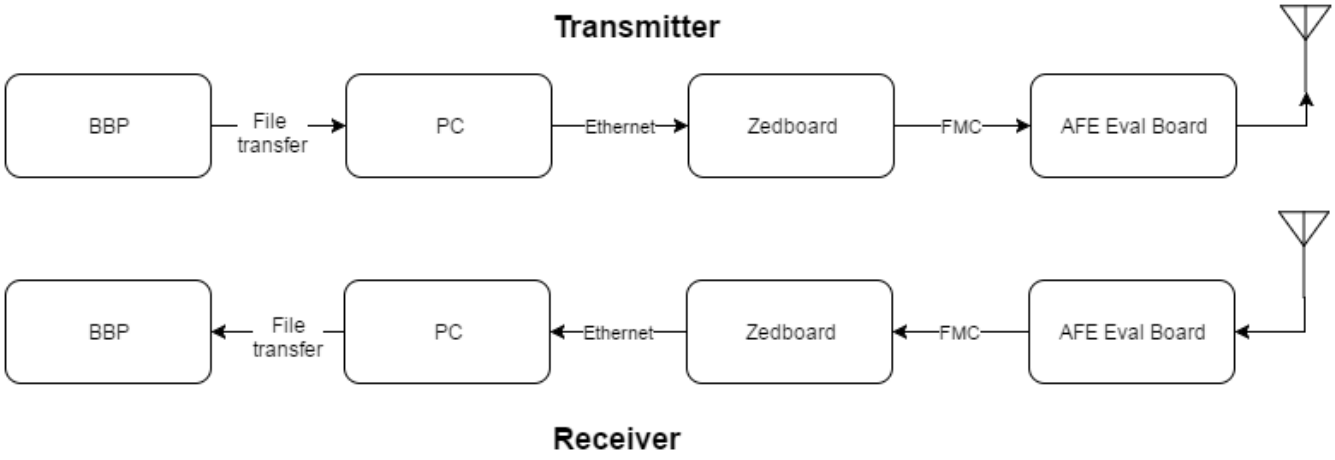


Figure 11: Offline data path

A test file is generated on a laptop, and all baseband operations are conducted on the BBP. This step outputs OFDM data frames into complex time domain samples, which are passed into a MATLAB OFDM transmission model, which drives the AFE to modulate these samples to the carrier frequency. In this demonstration, file transfers will take the place of data transmission over an LVDS interface.

Another PC runs a reception Simulink model which captures data when the model starts, an executes augmented MATLAB code to perform synchronisation and output FEQ gains to be applied on the DSP. It is neccessary to perform this on MATLAB due to the offline nature of the system. The synchronized complex time domain OFDM frame are written to a file for transfer to the BBP, where the baseband reception algorithms are implemented. Here, the received and original messages are compared to determine a BER for validation.

This data chain, while convoluted, is necessary to test the system. On the final board, the two chips will communication over an LVDS interface, however this standard is only designed for short transmission distances, and is hence not broken out on either evaluation board. This is because LVDS was originally desired as a communication standard between chips of the same computer monitor. In a future Capstone project, a system can be implemented where the DSP communicates in real-time with the AFE. Optimising the baseband code is important to this end, since all operations will need to have taken place within the cycle window.

4 Hardware Proposal

4.1 Original Requirements for Hardware

MSP's CubeSat mission goals include building a system that is high-speed, reliable and flexible to ensure the success of future satellite missions. Hardware must thus be chosen to reflect these requirements. Since there are very few commerical, off-the-shelf (COTS) high data-rate CubeSat systems that provide the necessary development flexibility, appropriate hardware was selected for our system. Similar CubeSat designs are only made possible by the additional gains provided by the antenna systems [31]. To meet requirements of system flexibility, The following factors should be considered in hardware evaluation.

1. Very low risk hardware system solutions.
   Since there is a lot of high speed hardware design involved in this project, designing with high frequency analog and digital components would be a major risk in the first design stage. Evaluation boards are preferable to simulate and verify the whole system in the first development stage, and they should be transferable to the customer board design.

2. Very high flexibility for AFE implementation.
   Since the legal issue is a major constraint for the RF carrier frequency and bandwidth selection, the centre frequency and bandwidth of AFE should be adjustable and the range should be as large as possible. The performance for the gain control and and data interface should also be particularly considered.

3. Very high capability for baseband processor (BBP) design.
   In order to implement an OFDM system with the data-rate adaptable in different link modes, the BBP should be capable to process all OFDM algorithms in the defined symbol time period. The efficiency of computation blocks, size of memories and bandwidth of data interface are the essential factors for evaluation.

4. Quick to implement for an engineering student.
   As a Master students program and one year capstone project, the design methodology should be accessible and understandable for the engineering students in a reasonable time frame.

5. Proven performance in communication systems.
   The hardware should have good records of communication system implementations with verified functionality in industry or research programs.

6. Space qualified.
   Since the final product will be launched into space, the components should be processed and manufactured to meet the space environment requirements.

The system could be generally divided into AFE, BBP and Antenna subsystems as shown in the Figure 12, each of which are discussed in subsequent sections.

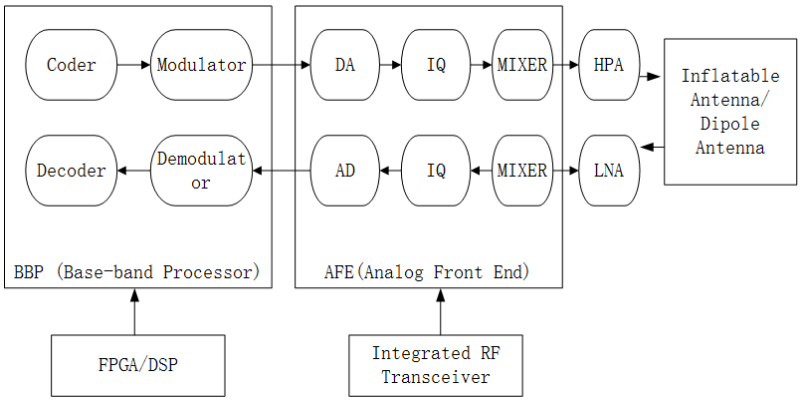


Figure 12: General hardware proposal, indicating the logical separation of BBP and AFE chips.

4.2 Analog Front End proposal

Because the team's negotiations with ACMA had only initiated, it was imperative to select an analog front end (AFE) that traversed as many legal bands as possible. This would ensure flexibility in design when it came to settling on bandwidth. This decision was propounded by the requirements of the novel antenna, which is of relatively small geometry[7]. Since it does not employ a dipole as its primary antenna, it was also important that this AFE offered lower-wavelength communications that the typical UHF/VHF bands where typical CubeSat systems reside [15, 16, 32]. The Analog Devices AD9364 AFE was selected since it meets these requirements.

The outstanding features of the AD9364 are shown below [5]:

- RF transceiver with integrated 12-bit DACs and ADCs
- 70 MHz to 6.0 GHz range for centre frequency
- 200 kHz to 56 MHz tunable channel bandwidth
- LVDS high speed data interface
- In-built AGC with alternative MGC modes
- SPI for control interface
- Mainly used for WIFI/LTE communications
- Evaluation board is available

[7] See Section 7

- Supported by MATLAB SDR

The AD9364 has integrated AD/DA, IQ modulation, flexible gain control and an RF mixer. The brief internal block diagram of AD9364 is shown in Figure 13.

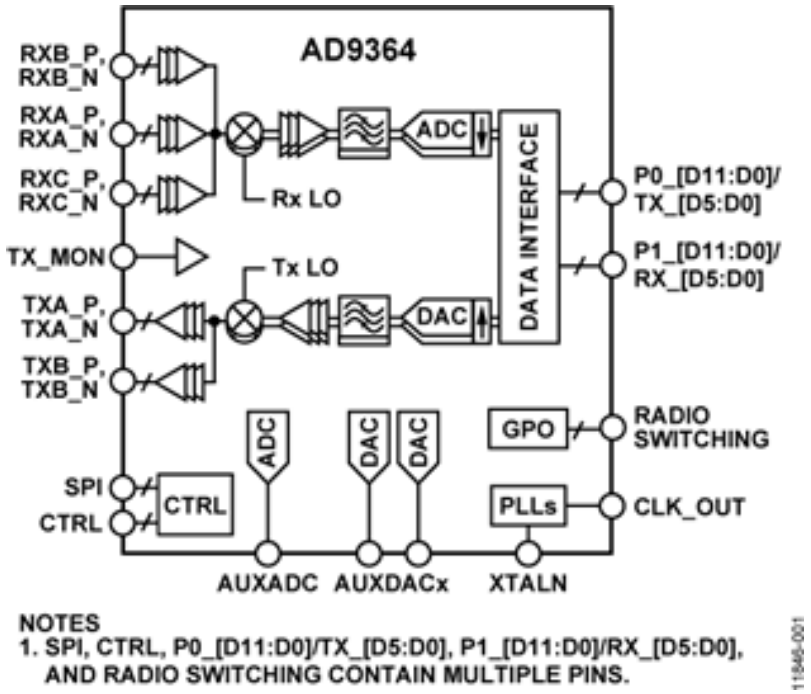


Figure 13: The block diagram of AD9364 [5]

4.3 Base-band Processor proposal

4.3.1 DSP and FPGA comparison

The DSP and FPGA are the two most popular processors for communication signal processing on the market. A DSP is a specialised micro-processor, and it is well suited for algorithm-intensive tasks. Normally the DSP could be programmed in C code or assembly code. Assembly code builds are more efficient than those with C code if the optimization is conducted properly, but can be time-consuming for an inexperienced programmer. For a DSP, the clock cycle consumption is the key criterion for the algorithm evaluation. An FPGA is a combination of large resources of logic gates and memory blocks for the user to design their digital circuits. A hardware description language (HDL) is generally used for FPGA programming, with Verilog and VHDL being the two most common HDL languages in the industry. The performance of FGPAs are usually limited by their logic resources and clock speed. The comparison of DSPs and FGPAs for this project are shown in Table 6.

FPGAs are more flexible than DSPs in terms of hardware adaptability and extensibility, but the time frame for development for FPGA cannot meet our requirement. Although a DSP has lower scalability than an FPGA, a

	DSP	FPGA
Power	Medium	Medium
Cost	Medium	Medium
Performance	Limited by MIPS	Limited by logic resource and clock speed
Scalability	Medium	High
Development Language	C or Assembly	HDL(Verilog or VHDL)
Estimated Schedule for development	6 Month	12 Month

Table 6: DSP and FPGA comparison

specialised DSP which includes an enhanced communication instruction set is capable enough to implement all of the algorithms in an OFDM system. Therefore, the Analog Devices TS201-S DSP was selected as the BBP, since published technical reports [9, 33] indicated that (with extrapolation) the most complex algorithms could be performed in sufficient clock-cycles to meet the OFDM topology proposed in Section 3.

4.3.2 DSP ADSP-TS201S

The ADSP-TS201S is one of the members of the Analog Devices TigerSHARC processor family. It is particularly optimized for telecommunication infrastructure and other large, demanding multiprocessor DSP applications. There exist accelerated hard-wired logic for complex multiplications, Viterbi decoding and interleaving which are essential for OFDM computation. It is also notable that this processor contains the LVDS interface which enables the direct connection between AFE and BBP. The hardware architecture of ADSP-TS201S is shown in the Figure 14.

To achieve the required cycle budget for OFDM processing, the DSP's processor contains numerous qualities that will benefit the system [34]:

- Wireless telecomm applications
- Up to 600MHz clock rate
- 16.8 microseconds for 1024-point complex FFT
- 200 kHz to 56 MHz tunable channel bandwidth
- 24 Mbit on-chip DRAM
- Enhanced instruction set for communication algorithm
- LVDS interface
- Evaluation board is available

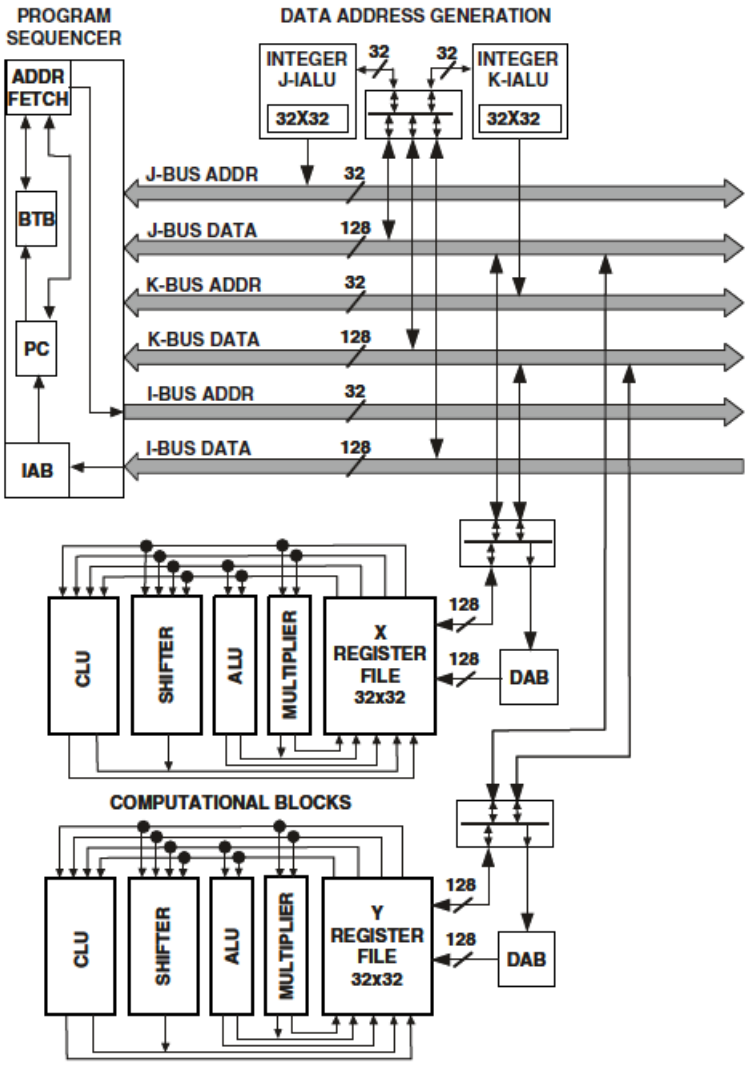


Figure 14: The hardware architecture of the AD TS201-S as illustrated in the programming reference manual [6].

- A dedicated CLU, that can perform trellis and interleave operations very efficiently.
- Pipe-lined architecture, so memory reads, communications arithmetic, arithmetic logic, integer logic and shifter logic can all operate in a paralleled fashion.
- An easily compilable and referable instruction set, including C-integrable-assembly code instructions.
- Extensive hardware, programming and assembler reference manuals.

Of particular importance is the presence of dual compute blocks (X, Y) and their associated paralleled CLU, SHIFTER, ALU and MULTIPLIER. These are harnessed by the compiler to produce fairly efficient algorithms, operating within the available cycle budgets[8]. As an example, if the team wishes to pursue a constraint length $k = 5$

[8]See Section 4.3.3

convolutional coder in the future, as opposed to the current $k = 4$ code, these compute blocks can be addressed independently from assembly to perform very efficient decoding, with almost no additional cycle costs compared to the current implementation.

4.3.3 Cycle budget

Since the clock rate for ADSP-TS201S is standardised at 500MHz, with the option to be clocked at 600MHz, an estimation for the BBP cycle budget is instantiated in Table 7-9. This cycle budget needs to be met to ensure real-time operability of the telecommunications architecture proposed in this project.

BBP	**AD TS201-S**	
Clock speed	500	MHz
Data period + CP	16	μs
Total clock cycles	8000	cycles
Clock cycle margin	4000	cycles
Total available clock cycles	4000	cycles

Table 7: Overall clock budget analysis

Major transmitter algorithm	**Clock cycle estimations**	**Clock (%) estimations**
Encoder	500	26.3
Interleaver	500	26.3
Mapper	300	15.8
IFFT	500	26.3
Add CP	100	5.3
Total	1900	100

Table 8: Clock budget estimations for major transmitter functions.

Major receiver algorithms	**Clock cycle estimations**	**Clock (%) estimations**
Viterbi decoder	1500	39.5
Deinterleaver	500	13.2
Demapper	400	10.5
FEQ	800	21.1
FFT	500	13.2
Total	3700	100

Table 9: Clock budget estimations for major receiver functions.

FFT and Viterbi estimates were obtained by extrapolating the cycle counts presented in technical reports [9] and

[33] respectively. Since performances of other modules could not be sourced, they were given estimates based on their comparative complexity to the Viterbi and FFT algorithms. In order to account for processor interruptions, stalls and data-link layer interactivity, the team asserted a 50% margin on the total clock cycles to achieve adequate system reliability. Clearly, the results of the estimates suggest that pursuing development on the TigerSHARC processor is feasible for the proposed high data-rate OFDM system.

4.4 Hardware sponsorship

At full commercial retail price, each TigerSHARC TS-201S DSP and AD9364 evaluation board set from Analog Devices cost roughly $5000AUD, and the ZedBoards from Xilinx cost $495USD. As these prices were clearly out of the budget of the group's FYP allowance and the MSP funding, sponsorship was sought from Analog Devices and Xilinx and these companies were kind enough to donate four TigerSHARC TS-201S evaluation boards, four AD9364 evaluation boards, four seats of VisualDSP++, two ZedBoards and two copies of the Zynq programming environment for a combined value in excess of $21,000 AUD.

5 Baseband Digital System Design

5.1 Physical Layer Base-band Architecture

The digital base-band design of the CubeSat system has been based on physical layer specifications of the WiFi standards. Primarily, this is because WiFi supports an adaptive rate protocol using OFDM modulation. Secondarily, WiFi is a highly documented standard and there are widespread third-party resources available for design purposes.

Data-link layer protocols[9] are also being developed by the telecommunications team, but have not been the focus of the Capstone team. Since the WiFi data-link and MAC layer is concerned with the networking of satellites, the data-link will be instead based on the point-to-point amateur-radio standards, AX.25, due to its reduced scope. The data-link protocols will be principally developed on the operating system's chip - the Texas Instruments MSP432.

5.1.1 Data Flow

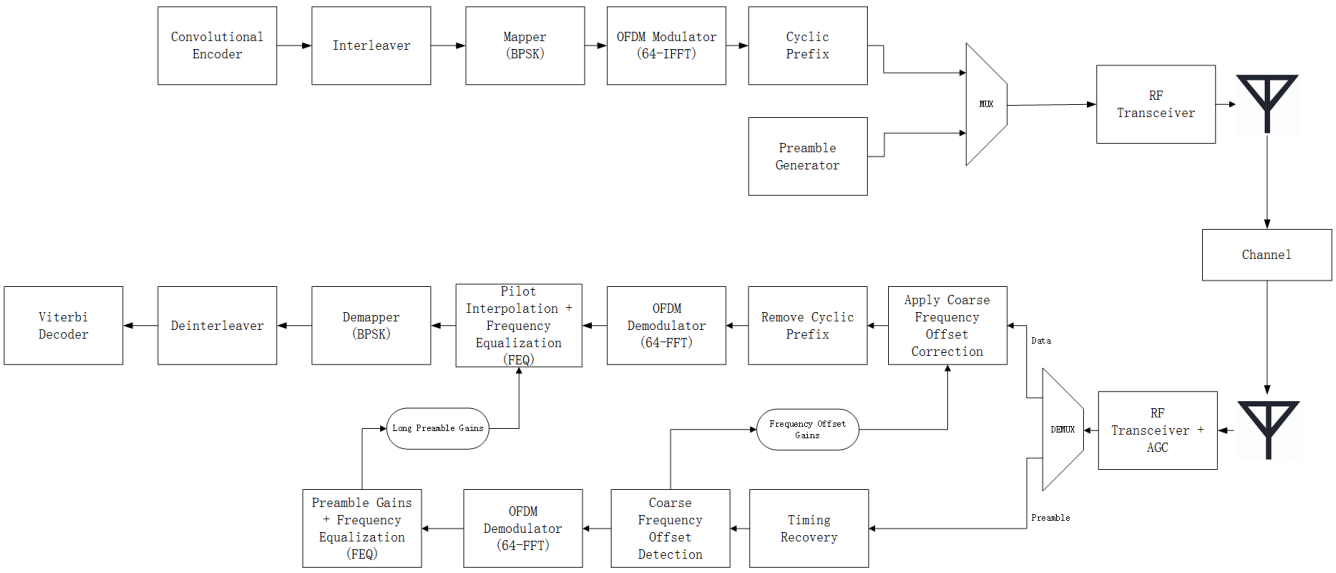


Figure 15: Interconnections of the high level base-band system architecture.

As illustrated in Figure 15, the base-band digital system is significantly complex and requires the interconnection of many functions to achieve OFDM modulation and provide robustness to channel effects. The WiFi standards outline an appropriate frame format for the data pass, which contains both synchronisation (preamble) symbols and data (payload) symbols.

A high-level system overview will be provided here, with further description (at the exception of the OS) provided in each module's relevant sections.

[9]See Figure 66 for details of telecommunication system abstraction in the OSI model.

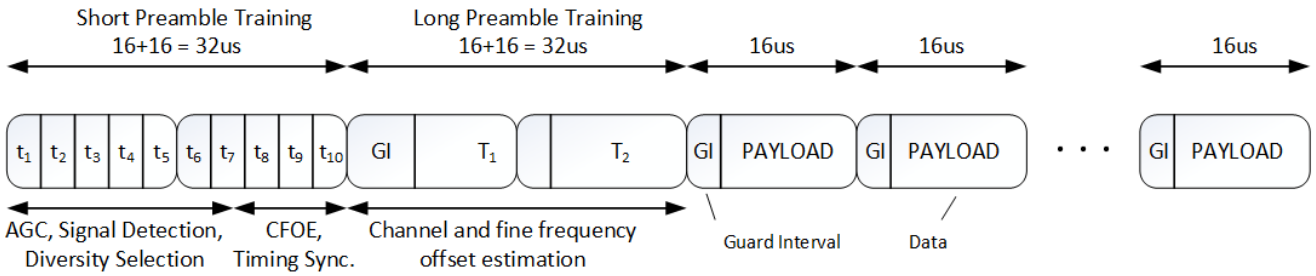


Figure 16: Frame format of the data passed through the link

In the transmitter, we have:

Operating System provides the transmission data to the communications system. The operating system is the first point in the system at which the amalgamated PHY data can be interpreted.

Convolutional Encoder performs forward error correction encoding for the system, reducing the overall system's BER.

Interleaver interlaces binary data over fixed intervals. Consequently, if channel bursts occur only part of a deinterleaved packet will have been destroyed.

Mapper maps the binary domain elements to a complex domain floating point number.

OFDM Modulator converts the complex frequency domain inputs to a complex time-domain output signal.

Cyclic Prefix copies a fixed length of samples from the end of a given time-domain symbol to the beginning of the symbol to mitigate inter-symbol interference.

Preamble Generator inserts a known training sequence into the transmitted stream for various synchronisation functions.

RF Transceiver maps the real parts of the time-domain sequence to in-phase sub-carriers, and the imaginary parts to quadrature sub-carriers. Performs mixing to 5.65 or 5.83 GHz. Gives the mixed signal to the antenna for transmission.

The receiver consists of two separate passes. One pass performs the synchronization and no tangible data is throughput into the system. The second is the data pass, which occurs after the system is synchronised and trained.

In the synchronizing pass, we have:

RF Transceiver + AGC performs the down-mixing and demapping of in-phase and quadrature sub-carriers to complex time-domain outputs. Automatically controls the gain of the receiver to ensure the input is within the

optimal range of the transceiver's ADC.

Timing Recovery uses the known short preamble sequence to determine the frame boundary.

Frequency Offset Detection implies the frequency offset by comparing the phase of samples in the one half of the long preamble sequence with identical samples in the second half of the long preamble sequence.

OFDM Demodulator demodulates the time-domain signal to a complex frequency domain signal. The constellation is likely to now contain imaginary components due to phase disturbances in the channel.

Preamble Gains + FEQ estimates equalizer gains for data equalisation using the long preamble training sequence.

In the data pass, we have:

RF Transceiver + AGC common to the synchronising path.

Coarse Frequency Offset Correction applies a phase gain to coarsely account for frequency mismatches in the mixing frequency and doppler[1] effects.

Remove Cyclic Prefix removes the portion of the signal likely to have been affected by inter-symbol interference.

OFDM Demodulator demodulates the time-domain signal to a complex frequency domain signal. The constellation is likely to now contain imaginary components.

Pilot Interpolation + FEQ performs an update on the frequency gains based on interpolation of the pilot tones. Applies preamble gains to data tones for coarse equalisation. Applies pilot gains to data tones for fine equalisation.

Demapper performs hard-decision decoding on the frequency sub-carrier constellation, and outputs a binary sequence.

Deinterleaver reconstructs the interleaved data to overcome channel burst errors in FEC.

Viterbi Decoder implements the Viterbi algorithm to decode the convolutionally encoded sequence.

5.1.2 Data Format

The data types were able to be reflected in the receiver modules, due to the use of hard-decision decoding[10].

Before the team started programming the data pass, the data types of Figure 17 were decided on to assist with design modularity. The benefits of this arrangement extends beyond modularity and also had enormous benefits

[10] See further discussion in Section 5.2.1.

in the ease of debugging the data pass. Because we were able to clearly define the boundaries of the OFDM symbol throughout the code, not only could we recognise faults more easily, but it resulted in no costly data format re-arrangements inside the mapping module.

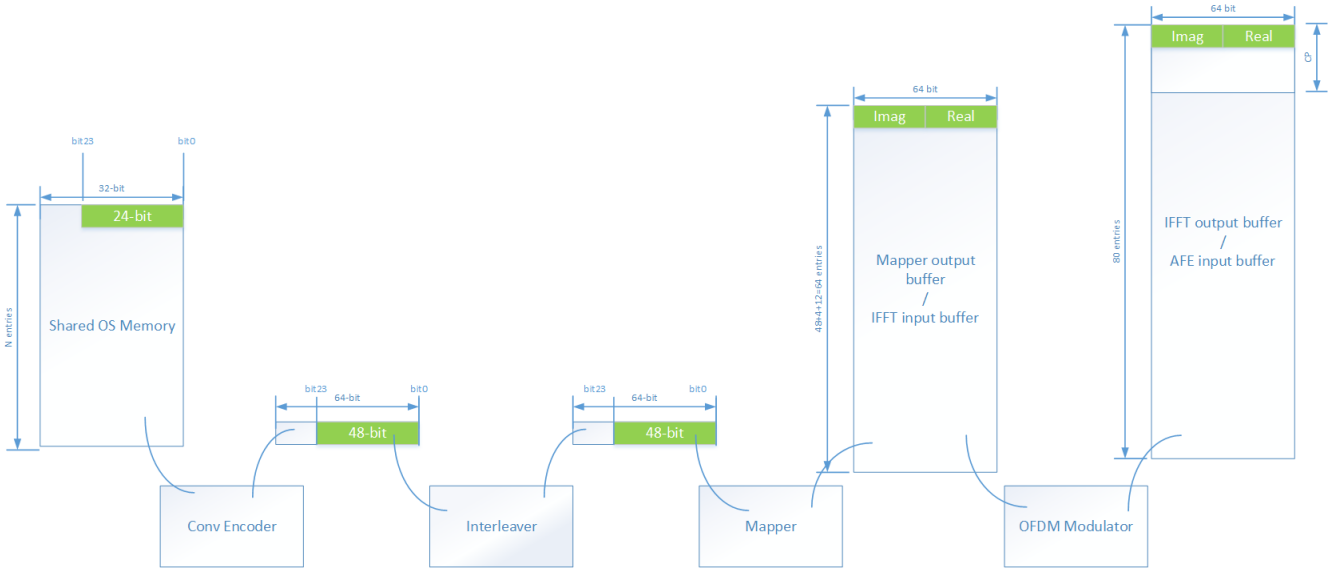


Figure 17: The established data-types of each algorithm, for the purposes of optimisation and clarity.

5.1.3 Inter-module rate mismatch and latency

Each module in the data pass has associated with it both latencies[11] and internal memory, sometimes a large array containing histories of past data. The two primary examples of these are the Viterbi decoder, which must store large amounts of trace-back data, and the interleaver module which must access one large block of data at a given time to perform its operations.

The processes external to these functions do not, however, want the additional complexity of slowing down and speeding up data throughput based on the state of the Viterbi decoder and interleaver. To counteract these effects, internal ping-pong buffers were employed in the relevant modules to ensure a constant data input stream and constant data output stream could be interfaced to.

Once the ping buffer is full and the pong buffer is depleted, the ping and pong buffers can switch places.

[11] See the software latency description diagram in Figure 67

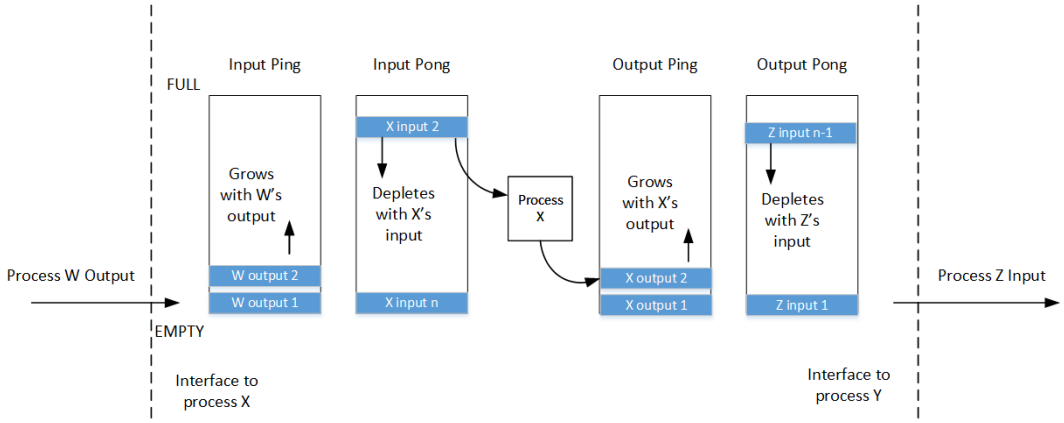


Figure 18: Use of ping pong buffer memory structures to decouple the inputs and outputs of various data-flow modules.

5.2 Encoding Scheme

5.2.1 Outline

As already mentioned, the digital base-band design of the CubeSat has been predominantly based on a WiFi PHY. This is largely due to its extensive documentation, but also since it is a high data rate protocol employing convolutional codes.

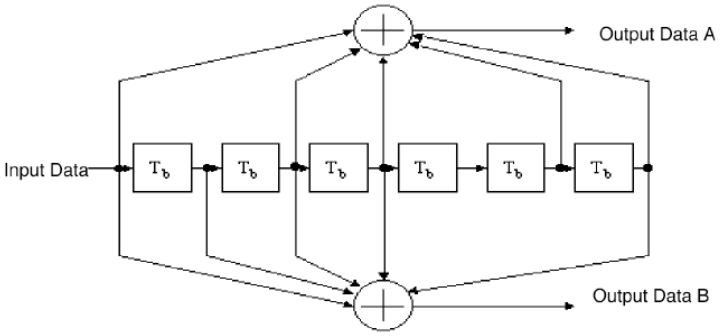


Figure 19: WiFi convolutional encoder topology, with constraint length $k = 7$ [7].

Convolutional codes are supported in the TigerSHARC hardware as efficient integer/fixed-point path metric calculations. In other words, it supports both soft and hard decision decoding system architectures.

5.2.1.1 Hard decision decoding

For the purposes of this design we have elected to use hard decision decoding, despite the additional $\approx$ 2dB offered if soft-decision decoding is pursued, illustrated in Section 5.2.4. This decision was not a result of increased Viterbi

algorithm complexity. In fact, employing soft decision decoding in the TigerSHARC's architecture would not contribute to programming or computational complexity inside the Viterbi trellis itself.[12]

Using soft-decision decoding requires the use of more complicated fixed-point data types. The add-compare-select structure used for the efficient Viterbi decoding can take fixed-point branch metric inputs of 8-bits in length, which would result in a soft-decoding quantisation precision of $\frac{2}{2^8-1}$ at the output of the mapper. Since the TigerSHARC can operate quite efficiently on 16-bit operations, the 8-bit soft-decision decoding would be the most practical form of soft-decision decoding for the system. Symbols using smaller bit-string representations would suffer in performance due to the difficulty in separating the data formats during deinterleaving.

Figure 20: The deinterleaver module also must be compatible with any changes to the data-type made by the mapper for the Viterbi decoder.

The inefficiencies and complexities that come with the soft-decoded data types inside the deinterleaving module was the major deterrent to electing not to support soft-decision decoding in the initial build. This would add significant memory requirements to the system, since what was once a single information bit is instead mapped into an 8-bit fixed point number. Since the system is trying to maximise the data pass' resistivity to channel burst errors, and TigerSHARC memory is limited, it would be counter-productive to reduce the interleaving memory capacity by a factor this large without further optimising the compile-time memory distrubution inside the TigerSHARC.

Furthermore, the additional computational requirements of the interleaving brought about by the increased number of memory accesses in the DSP would most likely push the number of clock cycles over the limit imposed by our cycle budget, if not performing any assembly level optimisations.

5.2.1.2 Flushing and state management

The convolutional coder has been designed to operate with a flushing efficiency of 1/32; after every 32 symbols the convolutional state is returned to zero. This is to prevent the propagation of burst errors through the convolutional trellis without reducing the coding gain of bits located at the tail of the input sequence. This is an effect the system would have suffered if we were to have simply truncated the tail bits, without flushing the system with 0's.

States were maintained inside the convolutional algorithms by employing static variables and static arrays where possible. These behave as global variables, only the variable's scope is secured exclusively to the called function. Hence, they do not have the unreliability associated with global variables. By avoiding arguments being passed in

[12]See Section 5.2.3.1 for trellis architecture details

and out of functions the code became clearer at the interfaces and resulted in fewer memory accesses required by the DSP.

5.2.2 Encoder implementation

The encoder is structured in its simplest FIR form.

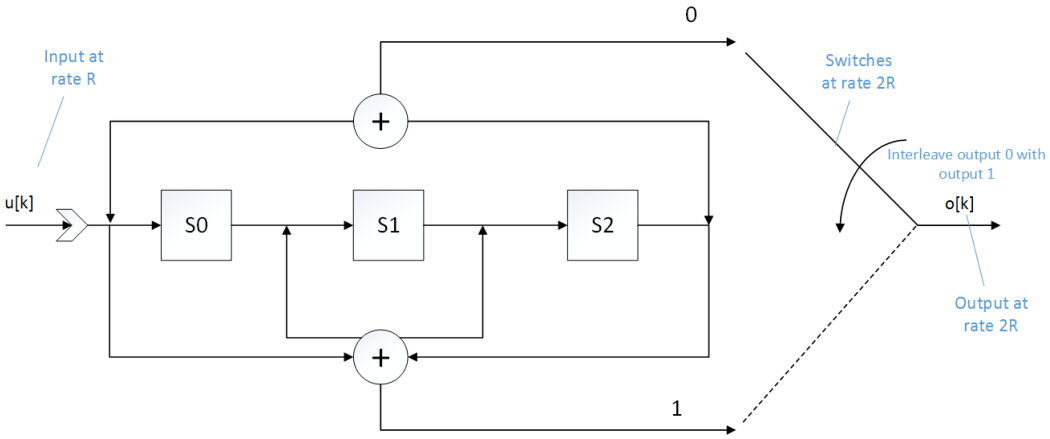


Figure 21: The implemented $k = 4$ convolutional encoder.

It differs from the WiFi standard in that it can only achieve a $k = 4$ add, without using more than one trellis in the decoder.

Optimisation lies in the subtlety of how the XOR's are executed inside the DSP. It is easiest to think of a convolutional coder as operating on a bit-by-bit basis. A rate $\frac{1}{2}$ convolutional coder will produce two output bits for a given input bit when each of the relevant states have been XOR-ed. Fortunately for the cycle budget, the DSP is not restricted to bit-wise XORs – 28 results can be easily computed in what otherwise would have produced only one result. See Figure 22 for illustration.

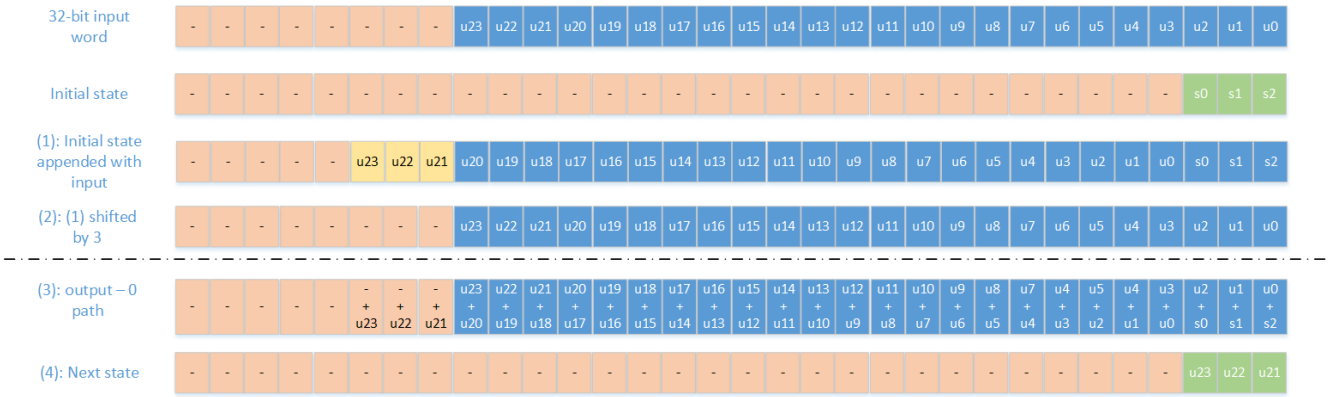


Figure 22: Illustration of an optimised 32-bit encoder algorithm.

The green squares of Figure 22 indicate the state bits, while the red bits indicate invalid results. The state can be updated by logic right-shifting (1) by 24, the number of data bits in a symbol. '+' indicates the addition of two elements over the binary field.

Clearly, this encoder implementation decreases in complexity by a factor of two if we were to use 64-bit additions, or 4 if they were 128-bit additions. The TigerSHARC would indeed be capable of these changes since it is a 128-bit processor. These changes were, however, not pursued in favour of a clear definition of the OFDM symbol boundary through the data-pass, see Figure 17 and associated reasoning. The encoder performs the exact same set of operations for the second output stream, only with additional right-shifts representing the two middle shift registers in Figure 21.

The question remains: how to perform the convolutional bit-interleaving to merge the two output sequences? The TigerSHARC architecture provides the solution with a 64-bit register load operation that takes the upper 32-bits and interleaves with the lower 32-bits when moving an operand into the CLU [6]. This particular command can only be executed via the assembly instruction `XYTHR1:0 = R3:2 (I) ;;` which cannot be performed using compiled C. This can be easily achieved through using the `asm()` command outlined in the TigerSHARC compiler manual [35].

5.2.3 Decoder implementation

One of the most significant challenges in designing the base-band algorithms was implementing the Viterbi algorithm efficiently in the TigerSHARC environment. One of the reasons the TigerSHARC was selected was due to the presence of the CLU, which can perform operations in parallel with ALU and IALU instructions[13]. To add to this challenge, Analog Devices do not supply any example code for implementing Viterbi in the TigerSHARC, and nor are there any sufficiently descriptive implementation reports online. This section will serve to demystify the implementation of Viterbi in the TigerSHARC for future MSP developments.

5.2.3.1 TigerSHARC trellis architecture

It is worth mentioning that the trellis structure employed by the TigerSHARC is to be used for more than just the Viterbi algorithm, and thus has been presented in a more general and non-descript manner by Analog Device's engineers.

This said, the diagram indicates that the trellis can perform path metric calculations on up to 8 states, using branch metrics defined by $[\gamma_1, \gamma_2, \gamma_3, \gamma_4]$. In other words, we will be completely describing the branch metrics of 16 out-going transitions with a set of only 4 numbers. Note also that each state's branch metric values $\pm\gamma$ are always mirrored by their negative values $\mp\gamma$ on the partnered out-going branch.

[13] See Section 4.3.2

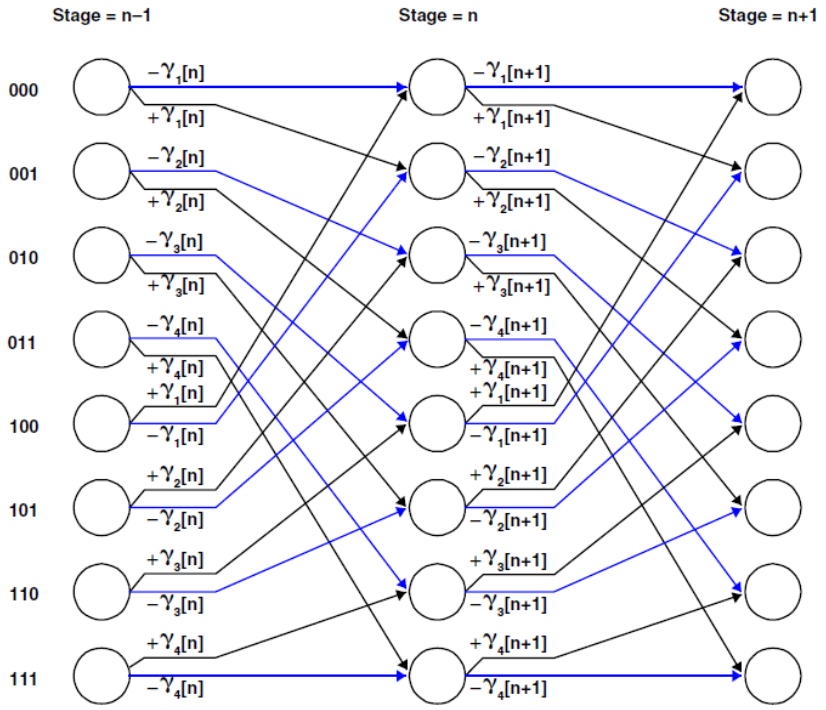


Figure 23: The available trellis structure as presented by Analog Devices in the Programming Reference [6].

To understand why this is possible, it is essential to redraw the trellis diagram and consider a mock input sequence.

Notice the separation of individual butterflies in Figure 24, each containing symmetry about their vertical centre. Consider that the trellis, at some stage n, receives an input of `01`. The topology indicates that out-going branch metrics need to be the negative of each other for implementation in the TigerSHARC. Let's define a transition yielding `01` as contributing a branch metric of $b_m[n] = +1$, a transition yielding `10` as contributing a branch metric of $b_m[n] = -1$, while a transition of `00` will contribute $b_m[n] = 0$ to the path metric. More formally:

$$b_m[n] = -d(r[n], \hat{r}) + 1 \tag{6}$$

where:

- $d(r[n], \hat{r[n]}) := w(r[n] - \hat{r}[n])$ is the traditional definition of the Hamming Distance of the received sequence r with some estimated transmit sequence $\hat{r}$ at stage n.
- $b_m[n]$ is the calculated branch metric at stage n.

The branch metric calculations related to other received sequences are produced using (6), and are found in Table 10.

Finally, to see why a b_m has been defined as a decreasing affine function of the Hamming Distance $d(r, \hat{r})$, we need to observe the operation of the add-compare-select (ACS) function employed in the TigerSHARC architecture, as shown in Figure 25.

		r			
	b_m	00	01	10	11
	00	1	0	0	-1
$\hat{r}$	01	0	1	-1	0
	10	0	-1	1	0
	11	-1	0	0	1

Table 10: A table of branch metrics for possible binary input pairs and trellis outputs.

Figure 25 shows that the 8 path metrics related to each state are stored inside registers TRn and TRm, which each utilise 16-bit fixed-point representation. The Rm register stores the four branch metric quantities $[\gamma_1, \gamma_2, \gamma_3, \gamma_4]$.

The function selects the maximum of the path metrics of state n after combination with the branch metrics for preservation in state n+1. Therein lies the logic that a predicted match adds 1 to the path metric, while a complete mismatch substracts 1. It follows that the path metric calculations need to be initialised at a reference of $p_0 = 0$, rather than a traditional reference of $p_0 = -\infty$.

The ACS function also outputs a single bit, 0 or 1, to identify if the previous state was obtained via the transition from TRn or from TRm respectively. This is then stored in the Trellis History Registers (THR) which become the basis of trace-back routine employed in C.

Using the 8-bit signed integer representations of 1 as `0x01` and -1 as `0xff`, we are able to construct Table 11; a description of the 32-bit branch metric words loaded into Rm under a particular received sequence r. This is achieved by tracing back through Figure 25 to determine under which conditions each branch metric is added and subtracted from the path metrics.

The ACS input/output structure in Figure 26 returns the correct convolutional outputs when simulated in VDSP++, where the branch metric register `Rm` is defined by Table 11.

r	Rm
00	`0xff0000ff`
01	`0x00010100`
10	`0x00ffff00`
11	`0x01000001`

Table 11: 32-bit words containing the four 8-bit values, γ_1 through γ_4, concatenated left-to-right.

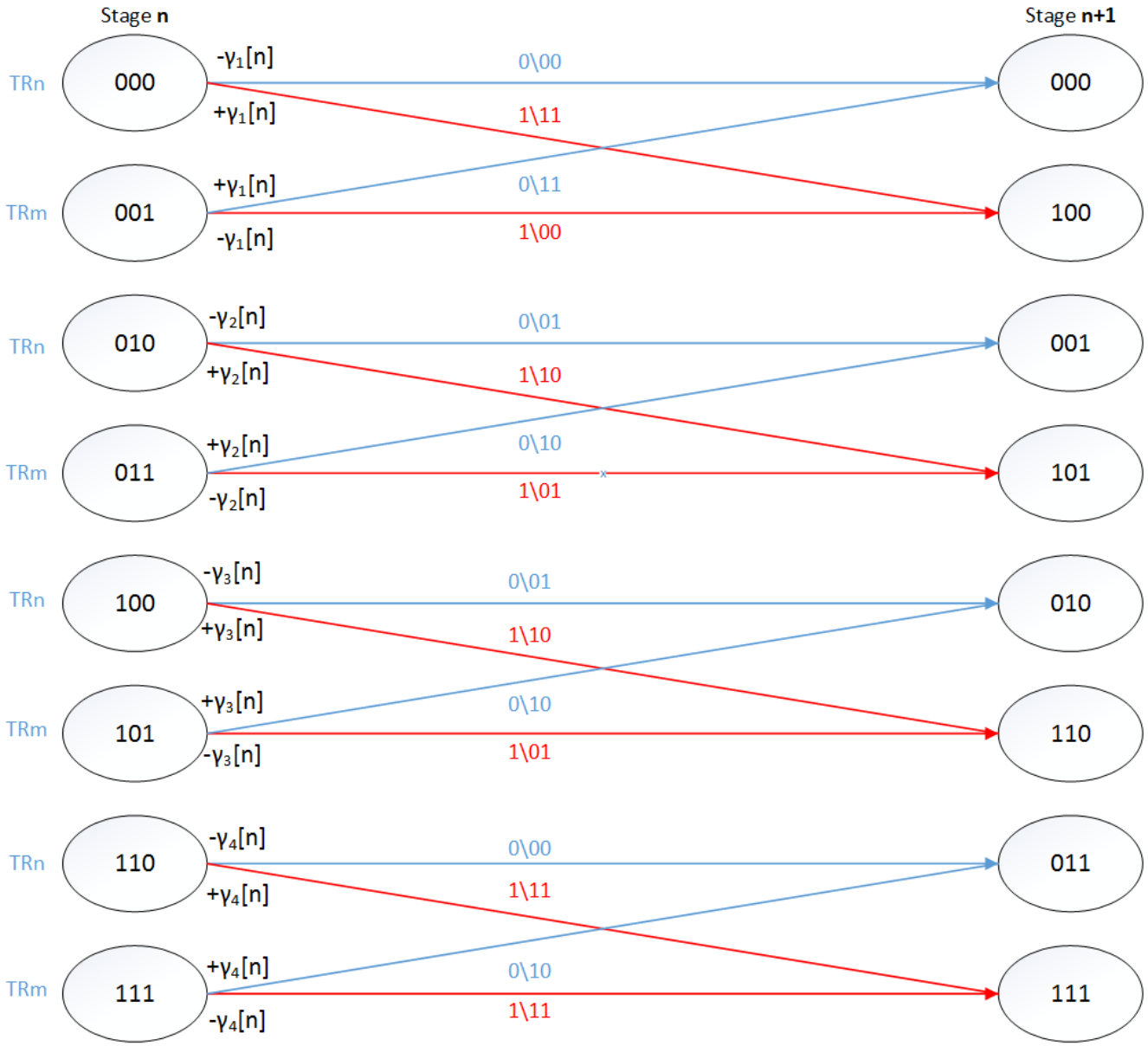


Figure 24: A re-ordered Viterbi trellis diagram.

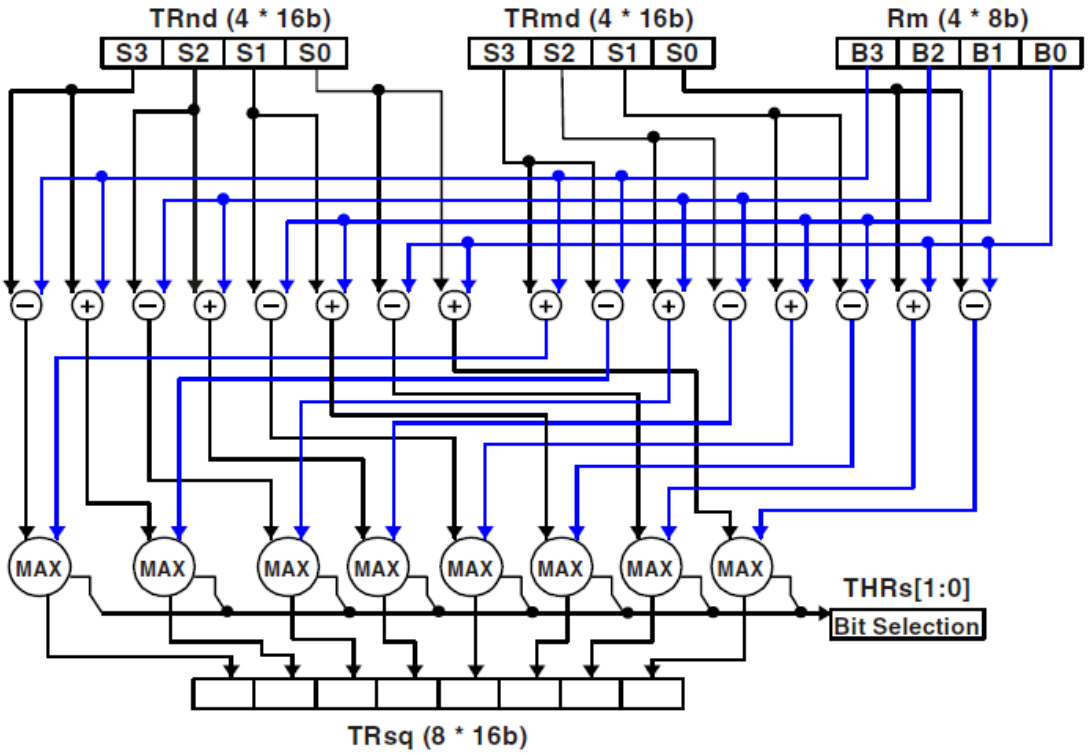


Figure 25: The field logic ACS function used by the TigerSHARC to update Viterbi path metric data.

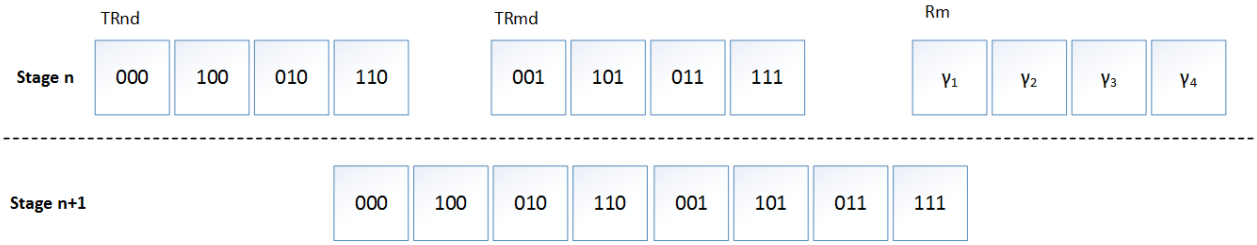


Figure 26: Illustration of the location of the state variables in the inputs of the ACS structure.

5.2.3.2 Trace-back

The trace-back routine is comparatively simple when compared to the effort required to bring up the forward-routine. This is largely because there are no field logic structures in the DSP's CLU to assist with trace-back, and instead we must implement the C code as efficiently as possible.

The trace-back routine employed a look-up array of 8 different states as follows in Figure 27.

```
typedef struct State {
        int prev[2];
        unsigned int recBit;
        int tbPos;
} State;
```

Figure 27: State structure employed in C for the trace-back routine.

where

- `prev[2]` contains the preceding states, addressable by the relevant THR bit.
- `recBit` contains the bit that was transmitted to push the encoder into this state, i.e. the decoded bit.
- `tbPos` indicates the correct position of the THR bit in the 8-bit output word. The order of bits in the THR output word is the same order as the states are positioned in the 128-bit output register of Figure 26.

Before trace-back begins, we must identify the state which has the highest path metric associated with it, representing the node with the largest likelihood of the correct sequence.

Finally, trace-back can occur over a simple iteration of the `State` structure of Figure 27. The benefit of using this structure is that the trace-back can occur with absolutely no comparisons, since the previous states are simply addressed via the `prev` vector. This trade-off of memory versus comparison operations is one that has been exploited many times in the code, and is made possible by the enormous 24-Mbit memory capacity of the TigerSHARC.

5.2.4 Coding Gain

The combination of the constraint length $k = 4$, rate $\frac{1}{2}$ convolutional encoder and its associated hard-decision viterbi decoder module resulted in a coding gain of $\approx$ 5dB for a target BER of $10^{-}5$, as seen in Figure 28.

The choice to move to soft-decoding would add an extra $\approx$ 2dB of channel gain. However, as mentioned in Section 5.2.1, the development of the soft-decision decoder was not performed for this project build, since the additional computational complexity and development time in the deinterleaver module was too restrictive for the delivery of

this project.

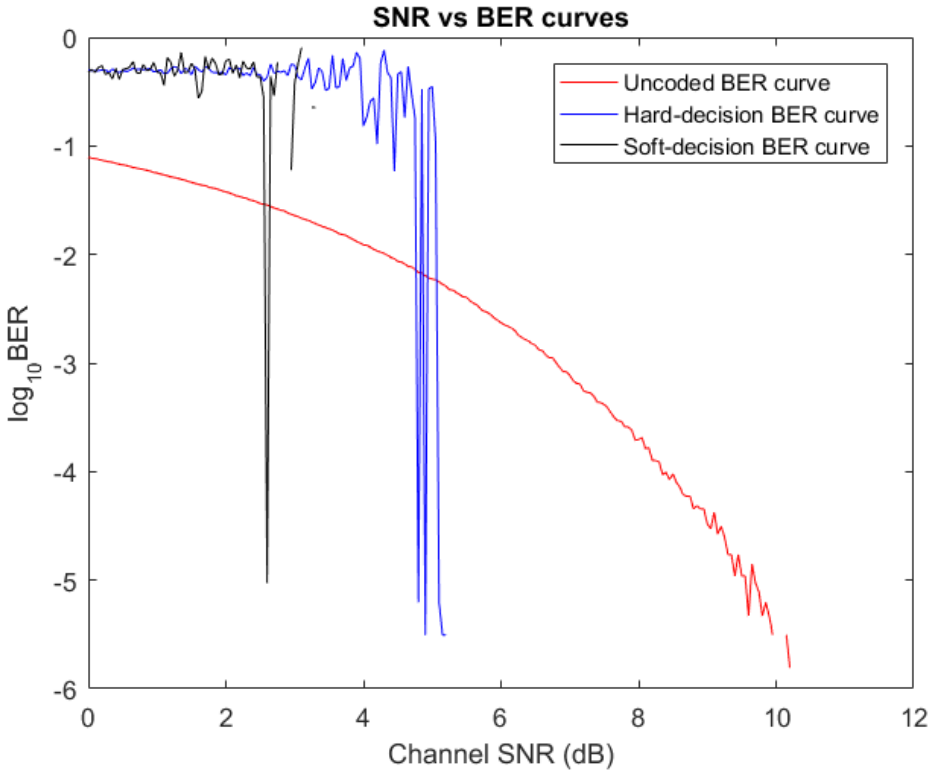


Figure 28: BER performance of the uncoded OFDM channel, and soft-decision and hard-decision coded channels. This model assumes no inter-subcarrier interference, and has thus been simulated as a BPSK system.

5.3 Interleaver and Deinterleaver

5.3.1 Outline

It is well-known that the BER performance of convolutional codes suffers drastically when the input bit string has burst errors [36]. Burst errors occur frequently in the space channel due to fluctuating sun activity or from co-channel interference. One solution to counter-act burst errors is to append a Reed-Solomon code and block-deinterleaver to the output of the convolutional decoder, to form a concatenated code, correcting for bursts at the output of the Viterbi decoder. This unfortunately can not be implemented without assembly level optimisations on the current system, due to the computational complexity required and the lack of finite-field logic in the TigerSHARC processor.

Figure 29 demonstrates a mock-interleaver of bit depth $d = 5$. Clearly, if a larger bit depth was selected, the bit error density at any given point in the output will be reduced.

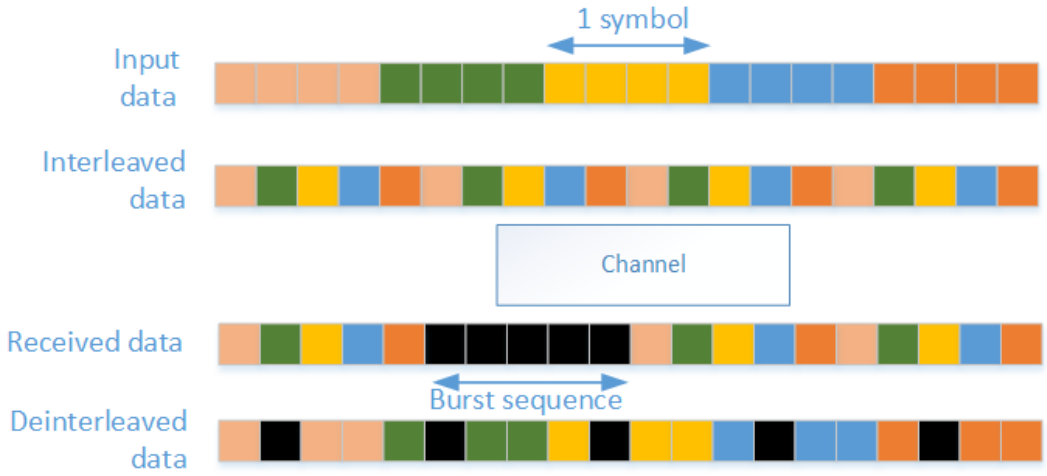


Figure 29: Illustration of an interleaver counteracting channel burst errors.

5.3.2 Interleaving solution

Bit-wise interleaving is a second method of burst correction, and can be instead implemented after the encoder and before the decoder. This has the effect of rearranging the output bit-stream so that channel burst errors can be separated in the channel with 67 symbol's worth of bits separating them. Thus, assuming that no more than one burst error occurs in a given interleave length, if a burst error with a duration of up to 1ms occurred, no two bits in a set of 67 symbols would be compromised. Unfortunately, the trade-off is burst errors generated inside the Viterbi decoder can no longer be corrected at the output.

The misfortune of the trade-off mentioned above is addressed however, by the target SNR and the simulated BER performance at the input output of the Viterbi decoder. The SNR of the system has been designed to achieve a codeless BER[14] of $\approx 10^{-5}$, and is thus much lower than the observed convolutional threshold input BER of 0.02.

For the purposes of this report, an interleaving bit-depth of 67 symbols was selected, primarily because depths exceeding this pushed the TigerSHARC memory into unbounded regions, resulting in run-time errors. Since solar flares have variable duration, it is difficult to pick a burst length to correct for, instead the system provide as much interleaving protection as the hardware allows.

5.3.3 Implementation

Two possible interleaving solutions were proposed, each with their merits and disadvantages.

[14] See Section 2.2

5.3.3.1 Cycle-optimised interleaver

The cycle optimised interleaver was consistently able to achieve average cycle counts per symbol of ≈ 400 cycles. This was achieved by using the same interleave command issued in the convolution encoder[15].

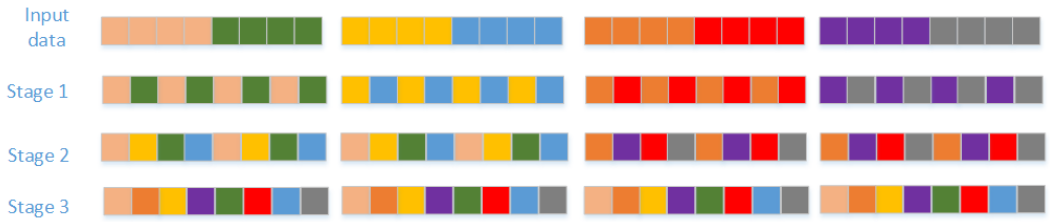


Figure 30: Cycle-efficient interleaving to an arbitrary bit-depth 2^N.

As demonstrated, this form of interleaver can interleave to a bit-depth of 2^N in only N data passes. Since the DSP stores information in integral blocks of 32- and 64-bits instead of one large bit stream, the program needed to keep track of where the updated boundary of an interleaved symbol while it was performing an interleave pass over a given stage.

To deinterleave, the inverse logical operation was formed by interleaving the n^{th} stage N times before passing the result to the $(n-1)^{th}$ stage. This added complexity resulted in a deinterleaver cycle count of ≈ 1600 cycles.

5.3.3.2 Alternative interleaver

The interleaver ISA optimisation outlined above came at a significant cost to the data-flow outlined in Figure 67.

- For the interleaver to be optimised, it required the input bit format remain in a 32-bit format, not the 48-bit format desired for clear definition of the symbol boundary.
- This block interleaver required 'bursts' of computational intensity; disrupting the constant symbol-by-symbol throughput of the data-pass.

In order to address both of these concerns, an alternative bit-wise interleaver was developed to maintain a 48-bit symbol boundary and to distribute the interleaver's operations uniformly over the transmitted data flow.

A simpler bit-mask-shift interleaver was implemented. A prime-number symbol depth of 67 was chosen so the binary data would be mapped to unique locations in the interleaved sequence when looping through the circular interleaver output buffer.

This interleaver satisfies the two design requirements, but still suffers a latency[16] equal to the interleaver block

[15] See section 5.2.2
[16] See section 5.1.3 for methods to combat latency

size and unfortunately is far less efficient than the cycle-optimised interleaver, with an average cycle count ≈ 1200 cycles.

While the performance was considerably diminished, both interleaver and deinterleaver modules were able to be fit inside their corresponding cycle budgets. Further assembly optimisation can be expected to reduce this count if future MSP teams wish to develop the physical layer.

5.4 BPSK Mapper and Demapper

The OFDM subcarriers shall be modulated using Binary Phase Shift Keying (BPSK). The mapping is to convert a binary bit 0 or 1, obtained from the interleaver, into a 64-bit complex number $-1+j0$ or $1+j0$, where the lowest 32 bits are for the real part and the highest 32 bits are for the imaginary part, so that the data can be then processed by the 64-floating-point IFFT function.

The mapper first inserts the 4 pilot subcarriers and 12 pad bits (zeros) into an OFDM symbol, which increases the 48-bit symbol into 64-bit. It then maps each bit in the symbol to the corresponding 64-bit complex numbers, 1 or -1, in BPSK.

5.4.1 Pilot Subcarriers

In OFDM, 4 pilot subcarriers are inserted in each symbol for coherent detection to remain robust to fine frequency and phase offsets. These pilots are located on subcarriers $-21, -7, 7, 21$, as shown in Figure 31, with values $1, 1, 1, -1$ respectively. This is achieved by establishing known frequency and phase references that can be detected at the receiver side, so the frequency equalizer (FEQ) in the receiver can estimate and compensate for disturbances. The subcarrier pilots shall also be modulated with BPSK. In order to avoid DC offset in the A/D and D/A converter, and to prevent carrier feed-through in the RF system, the subcarrier at 0 is also nulled, which is usually seen as a pad bit. Since our symbols are BPSK modulated, the pilot subcarriers and data subcarriers can be processed together.

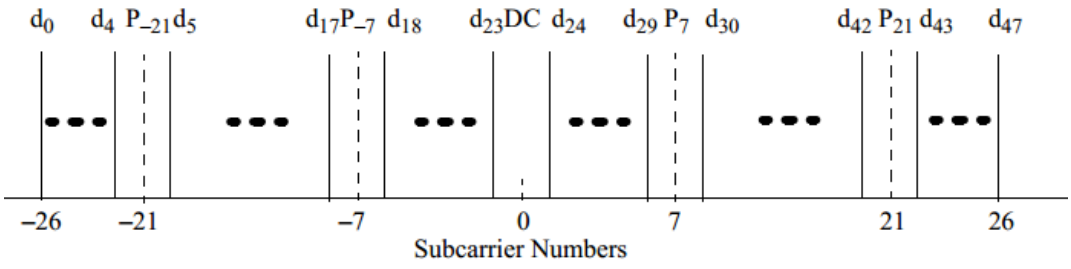


Figure 31: Subcarrier Frequency Allocation [7]

5.4.2 Pad Bits

The remaining 11 pad bits are allocated at both sides of the OFDM symbol's spectrum to serve as the signal's guard band. At the edges of the band, the transmit spectrum mask begins to fall, as shown in Figure 32. Thus, to avoid loss of the desired signals in the frequency domain and make sure the transmitted signals are within the spectrum mask, we place nulls at the edge sides.

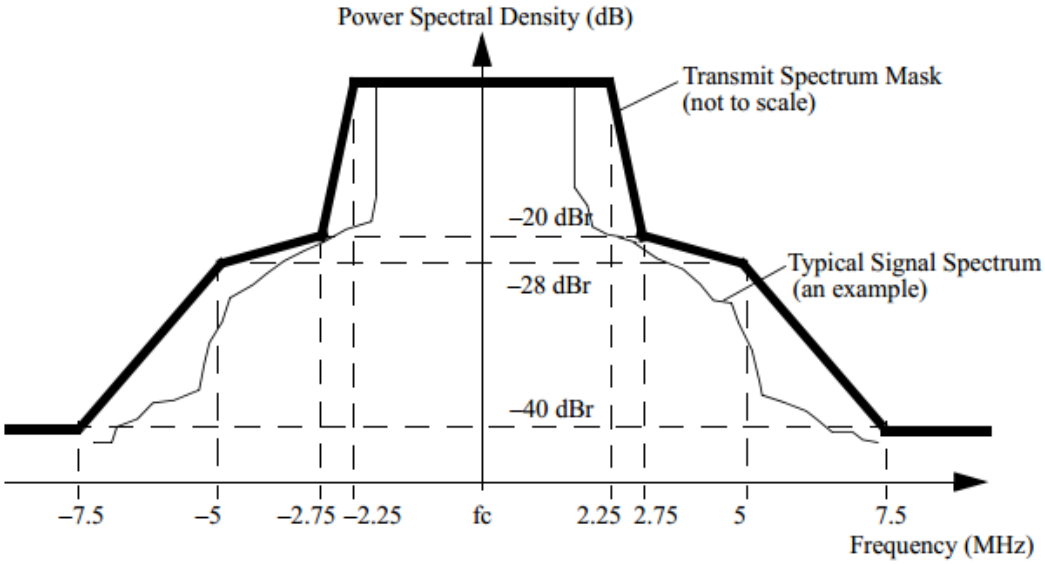


Figure 32: Transmit spectrum mask for 5 MHz transmission [7]

To make the OFDM symbols match the OFDM modulation equation, the data subcarriers number 0 through 47 need to be mapped to the frequency offset index -26 to 26, while the pilot subcarrier locations and the 0th (DC) subcarrier shall be skipped. The mapping relationship is shown in Figure 33.

-32 to -27	-26 to -22	-21	-20 to -8	-7	-6 to -1	0	1 to 6	7	8 to 20	21	22 to 26	27 to 31
pad bits 0s	data bits 24 to 28	pilot subcarrier	data bits 29 to 41	pilot subcarrier	data bits 42 to 47	DC 0	data bits 0 to 5	pilot subcarrier	data bits 6 to 18	pilot subcarrier	data bits 19 to 23	pad bits 0s

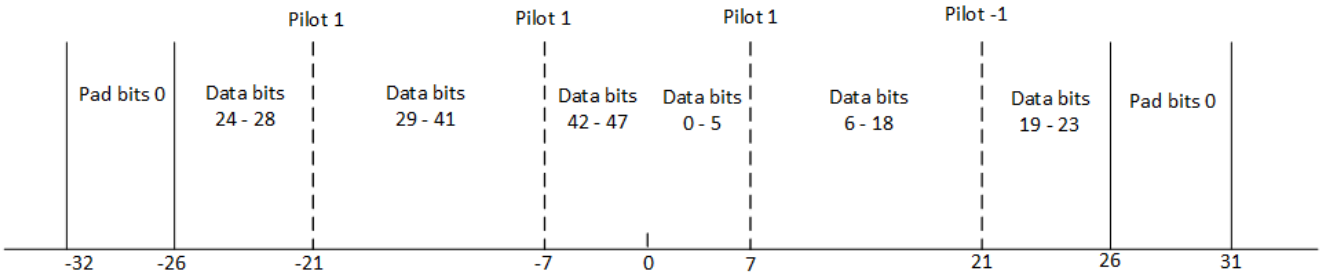


Figure 33: Subcarriers locations after frequency shift

5.4.3 BPSK Modulation

Subsequently, the 64-bit OFDM symbols are modulated into 64 complex numbers. The modulation scheme is BPSK, where the encoded and interleaved binary bits are mapped to the corresponding complex numbers bit-by-bit: in BPSK the data stream is divided into groups of 1 bit and then mapped into complex number, $I + jQ$ representing the IQ constellation, as shown in Figure 34. The output value d is normalized by multiplying a normalization factor K_{MOD}.

$$d = (I + jQ) \times K_{MOD} \tag{7}$$

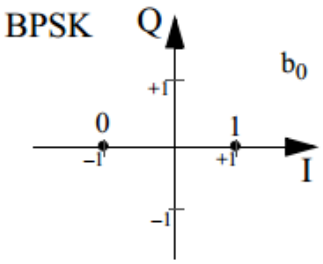


Figure 34: BPSK Constellation [7]

In BPSK, $K_{MOD} = 1$. Therefore, the output value is 1+j0 if the input of the mapper is bit 1 and -1+j0 if the input is bit 0, as shown in Table 13.

Input bits	I-out	Q-out
0	-1	0
1	1	0

Table 12: BPSK encoding table

Both the real part and the imaginary part of an output value are stored in IEEE754 single-precision floating-point format (Binary32), so that they can be interpreted by the 64-floating-point FFT function in the DSP.

5.4.4 Binary32

The Binary32 format consists of three parts as shown in Figure 35, where the decimal value is determined by Equation 8.

$$\text{value} = (-1)^{\text{sign}} \times (1 + (\text{mantissa})^{10}) \times 2^{\text{exponent}-127} \tag{8}$$

Figure 35: IEEE754 Single-precision floating-point format (Binary32) [8]

The Binary32 is also used by the float data type in C language, so when programming the mapper, the output can be simply stated as $float$ rather than explicitly converting them into a hexadecimal representation. The relationship of conversion is shown in Table 13.

Highest 32 bits	**Lowest 32 bits**
Imaginary: 0	Real: 1/-1
Binary32: 0x0000 0000	Binary32: 0x3F80 0000 / 0xBF80 0000
float: 0	float: 1/-1

Table 13: BPSK encoding table

5.4.5 Demapper

The demapper is just the inversion of the mapper. At the receiver side, it BPSK demodulates each received complex numbers $\pm a \pm j0$ via a hard-decision back to into digit bits 1 or 0 and then removes the pilot subcarriers and pad bits to get back the 48-bit OFDM symbols, which will later be deinterleaved and decoded.

5.5 OFDM modulator and demodulator

5.5.1 FFT optimization in DSP

IFFT and FFT are used to implement OFDM modulator and demodulator. The IFFT transforms the frequency domain data into time domain sampling points and FFT perform the opposite operation. A demonstration of a standard 16 point FFT structure is shown in Figure 36.

The C code for this structure implemented in Tigersharc DSP takes approximately 5000 cycles. The main reason for this high cycle count is due to inefficient access to the DSP's memory interface and poor organization in the compiled data structure.

As shown in Figure 14, TigerSHARC DSP supports dual quad memory accesses (J-BUS data and K-BUS data), SIMD operations via the dual computational blocks, and four simultaneous instructions in one cycle (VLIW). However, without reconstructing the assembly algorithms, it is impossible for compiler to fully utilize all of these

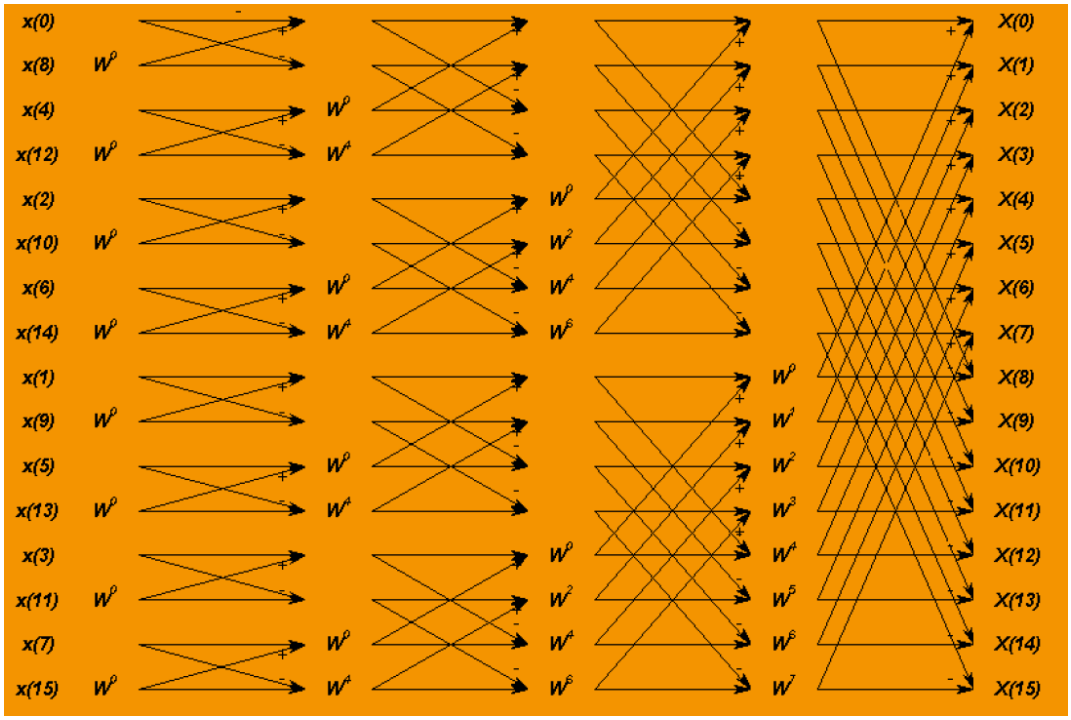


Figure 36: standard 16 point FFT structure [9]

resource in DSP and generate very efficient code. The Analog Devices engineers have constructed an assembler reference document and instructional on how to efficiently program N-length floating-point FFTs.

In order to take advantage of fast quad memory access, the butterfly structure has to be changed to accommodate this feature. The data input to DSP is 32-bit real and 32-bit imaginary. If the data reads for butterfly are sequential, two complex data values could be read out in one cycle instead of 4 cycles. Taking this into consideration, the FFT structure could be reorganized as in the Figure 37 [9]. Thus, the operations for the new structure needed in a single butterfly process is shown in Table 14.

Since there are two computational blocks in the TigerSHARC, two adjacent butterflies could be processed in a SIMD fashion. Combining this feature with quad memory access [9] demonstrates that the Fetch F1, F2, K2, F4 for butterfly1 can be performed in one cycle and computed in the X-Compute block, while butterfly2 could be arranged in Y-compute block in the same cycle. Because the two adjacent butterflies are processed at the same time, the S1 and S2 for the two butterflies can be stored into memory in one cycle quad memory fashion, and S3 and S4 are stored in another cycle. The A3 and A4 could be combined together due to the single add/subtract instruction in the DSP. Totally, two adjacent butterflies only consume two data fetches, four multiplications, four ALU, and two memory store. This only yields to four cycles pipeline execution in the DSP because of the VLIW structure. So far, the architecture provides two spare fetches that have not yet been occupied, so the twiddles can be fetched from memory in these two idle fetch cycles. The new operations table is shown in the Table 15.

In the 64-point FFT, there are 6 stages and 32 butterflies for each stage. Since 4 cycles for butterflies is achievable in

Mnenonic	**Operation**
F1	Fetch Real(Input1) of butterfly
F2	Fetch Imag(Input1) of butterfly
K2	Fetch Real(Input2) of butterfly
F4	Fetch Imag(Input2) of butterfly
T1	Fetch twiddle Real
T2	Fetch twiddle Imag
M1	K2*T1
M2	F4*T2
M3	K2*T2
M4	F4*T1
A1	M1-M2=real(input2*twiddle)
A2	M3+M4=imag(input2*twiddle)
A3	F1 + A1=real(Output1)
A4	F1-A1=real(Output2)
A5	F2+A2=imag(Output1)
A6	F21A2=imag(Output2)
S1	store(Real(Output1))
S2	store(imag(Output1))
S3	store(Real(Output2))
S4	store(imag(Output2))

Table 14: FFT operations in a single butterfly

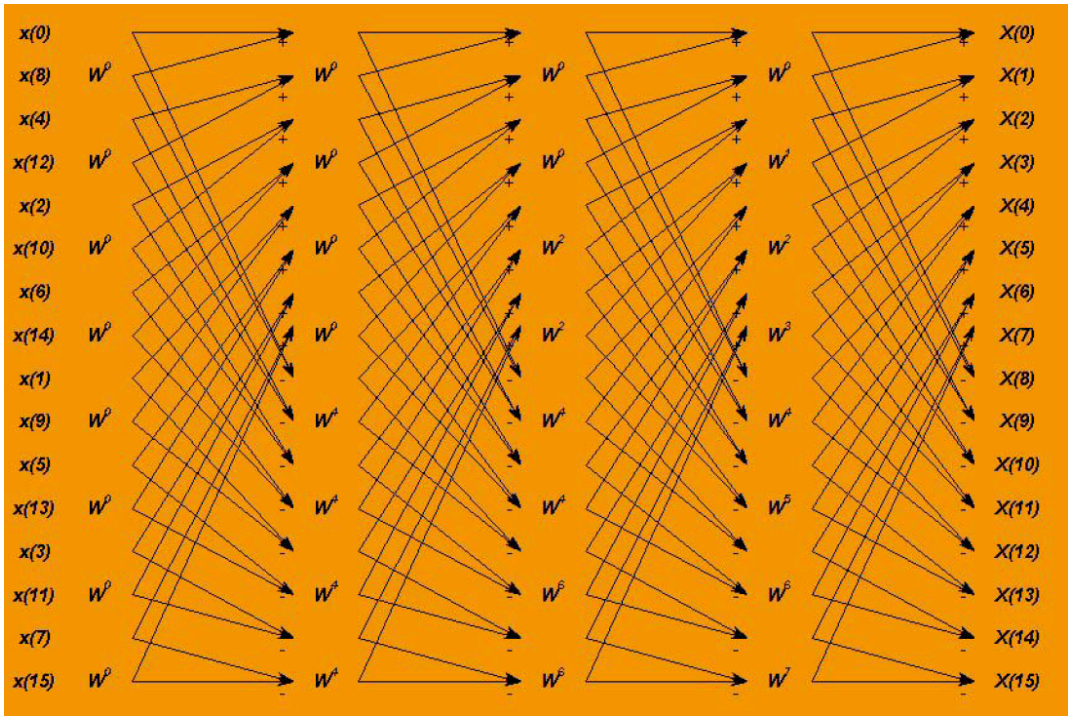


Figure 37: re-organized 16 point FFT structure [9]

this DSP structure, the total cycle count with overhead consideration will be $6(stages)\times32(butterflies)/2(butterflies)\times 4(cycles) = 384(cycles)$. Considering the overhead for software pipe-lining and context switch in the assembly function, the 550 cycles result in simulation is reasonable.

5.5.2 IFFT implementation using FFT structure

The inverse FFT can use the same FFT structure, only swapping the real and imaginary parts of the input and output with a scalar. The inverse DFT equation [37] is given by:

$$x(n) = \frac{1}{N}\sum_{k=0}^{N-1} X(k)e^{\left(\frac{j2\pi kn}{N}\right)} \tag{9}$$

$$= \frac{1}{N}\sum_{k=0}^{N-1} (X_{real}(k) + jX_{imag}(k))(cos\left(\frac{2\pi kn}{N}\right) + jsin\left(\frac{2\pi kn}{N}\right)) \tag{10}$$

After some arrangement:

$$x(n) = \frac{1}{N}\sum_{k=0}^{N-1} [X_{real}(k)cos\left(\frac{2\pi kn}{N}\right) - X_{imag}(k)sin\left(\frac{2\pi kn}{N}\right)] + j[X_{real}(k)sin\left(\frac{2\pi kn}{N}\right) + X_{imag}(k)cos\left(\frac{2\pi kn}{N}\right)] \tag{11}$$

Mnenonic	Operation
K1	Twiddle address mask
K2	Twiddle Fetch
K3	Twiddle address increment
F1	Fetch input data for butterfly1
F2	Fetch input data for butterfly2
M1	real(lower input data)*real(twiddle)
M2	imag(lower input data)*imag(twiddle)
M3	real(lower input data)*imag(twiddle)
M4	imag(lower input data)*real(twiddle)
A1	M1-M2=real(input2*twiddle)
A2	M3+M4=imag(input2*twiddle)
A3	real(input1)+/-A1=real(output1,2)
A4	imag(input1)+/-A1=imag(output1,2)
S1	Store(output1, two butterflies)
S2	Store(output2, two butterflies)

Table 15: Optimized FFT operations for two butterflies in parallel

Assume $X_{swap}(k) = X_{imag}(k) + jX_{real}(k)$, the DFT $X_{swap}(k)$ is equal to:

$$x(m) = \sum_{m=0}^{N-1} X_{swap}(n) e^{\left(\frac{j2\pi mn}{N}\right)} \tag{12}$$

$$= \sum_{n=0}^{N-1} [X_{imag}(m) cos\left(\frac{2\pi mn}{N}\right) + X_{real}(m) sin\left(\frac{2\pi mn}{N}\right)] + j[X_{real}(m) cos\left(\frac{2\pi mn}{N}\right) - X_{imag}(m) cos\left(\frac{2\pi mn}{N}\right)] \tag{13}$$

It is easy to find the that $x(m)$ is the swap of the real and imaginary parts of $x(n)$ with a scaling. This implies that the IFFT could be calculated by swapping input, performing the FFT, and swapping output. The block diagram for IFFT implementation by using FFT is given by Figure 38.

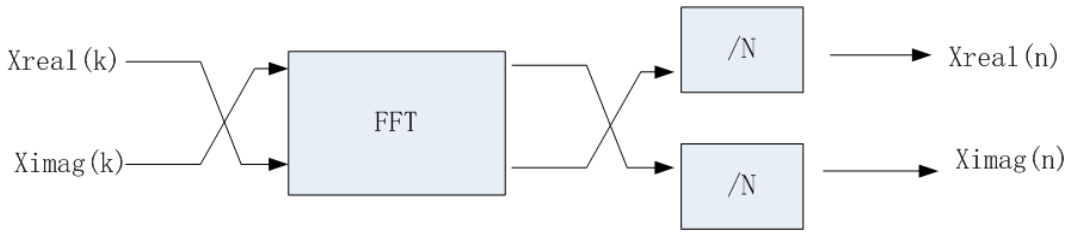


Figure 38: IFFT calculation by using FFT

5.6 Cyclic Prefix

The cyclic prefix (CP) refers to the prefixing of a symbol with repetition ends. In this project, the cyclic prefix is the last 16 complex numbers of a symbol copied and prefixed to the head. The symbol is the 64 complex numbers output from the IFFT.

The cyclic prefix serves as the guard interval during transmission, which eliminates the inter-symbol interference (ISI) from the previous symbol. In a practical transmission, it would happen if a symbol is delayed, so its end part would interfere the next symbol, as shown in Figure 39. A solution to this problem is to introduce some time gap between two symbols, and fill this with the cyclic prefix. The cyclic prefix is a repetition of the end part of the symbol, and is pasted to the head of the symbol, as the dash line shown in Figure 40. Therefore, the actual data in the symbol two could be safe even if the delay happens in the first symbol, and only the cyclic prefix is interfered. The length of cyclic prefix is equal to the guard interval. The guard length is 16 [38], so the last 16 complex numbers of the symbol shall be copied and pasted.

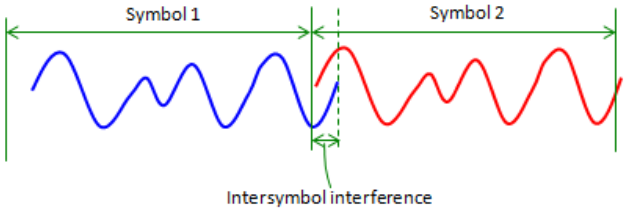


Figure 39: Inter-symbol interference caused by symbol 1's delay [10]

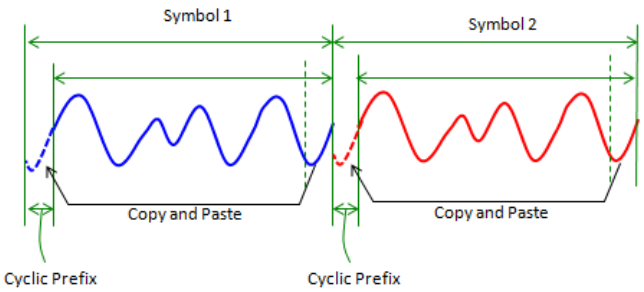


Figure 40: The cyclic prefix introduced into symbols [11]

Finally, the cyclic prefix allows the linear convolution of the channel to be treated as a cyclic convolution, which is the time-operation of the DFT. This allows for frequency-domain signal processing techniques in equalisation to ensure robustness to a multi-path channel.

5.7 Automatic Gain Control

Automatic Gain Control (AGC) exists in order to scale an input signal so that it occupies the full input scale of an ADC, in order to maximise the resolution of the digitized representation. The received signal will exhibit wide power swings over the course of a single transmission due to differing free space losses and atmospheric attenuation, so a single time-invariant scalar is not appropriate: AGC is required instead. Since the system doesn't operate in real-time, this Capstone report and the hardware demonstration conducts AGC as described in Section 6.4. The method outlined below is a proposed solution for AGC once a real-time system has been established.

In IEEE 802.11 the header of the OFDM frame displays Short Training Symbols (STS) and Long Training Symbols (LTS). The first few symbols are shown in Figure 16, and the STS is intended to let the receiver detect the OFDM signal and stabilize the AGC mechanism.

To ensure an incoming frame is not dropped, the AGC must quickly converge and also stabilize the AGC gain when the OFDM symbol is received. The AGC finite state machine shown in the Figure 41 is proposed for this purpose. When some signal is detected at the input, a fast convergence for AGC is required. In this solution, the AGC Fast Attack 1 is intended to detect a stable OFDM signal and get a stabilized AGC gain for OFDM preamble detection. At this time, the auto-correlation in synchronization is disabled because the AGC has not been settled [39].

When the AGC gain settle flag is asserted from AFE, the AGC Fast Attack 2 state is triggered and synchronization is enabled. Once the short preamble sequence is detected by the auto-correlation, the AGC should be locked or changed to slow attack mode. This will prevent signal corruption due to unexpected interference or gain fluctuation in the payload receiving stage. If there is a consecutive OFDM frame been detected after one OFDM frame, the synchronized AGC Fast Attack (AGC Fast Attack 2) is recalled by the state machine. The digital system is disabled after a time delay if no OFDM frame is received and return to Signal Detection state. The details of each state is shown in Figure 42.

5.8 Synchronisation (Schmidl's Method)

For the offline system build that is the deliverable of this Capstone project, all synchronisation techniques proposed here-in have been implemented in the MATLAB environment, as opposed to the DSP environment. The efficacy of DSP implementation has informed the selection of synchronisation algorithms.

5.8.1 Timing Recovery

In Schmidl's symbol timing method the training symbol contains two identical halves. Multiplying the conjugates from the first half of the long preamble samples with those of the second half, the phase of the products will be

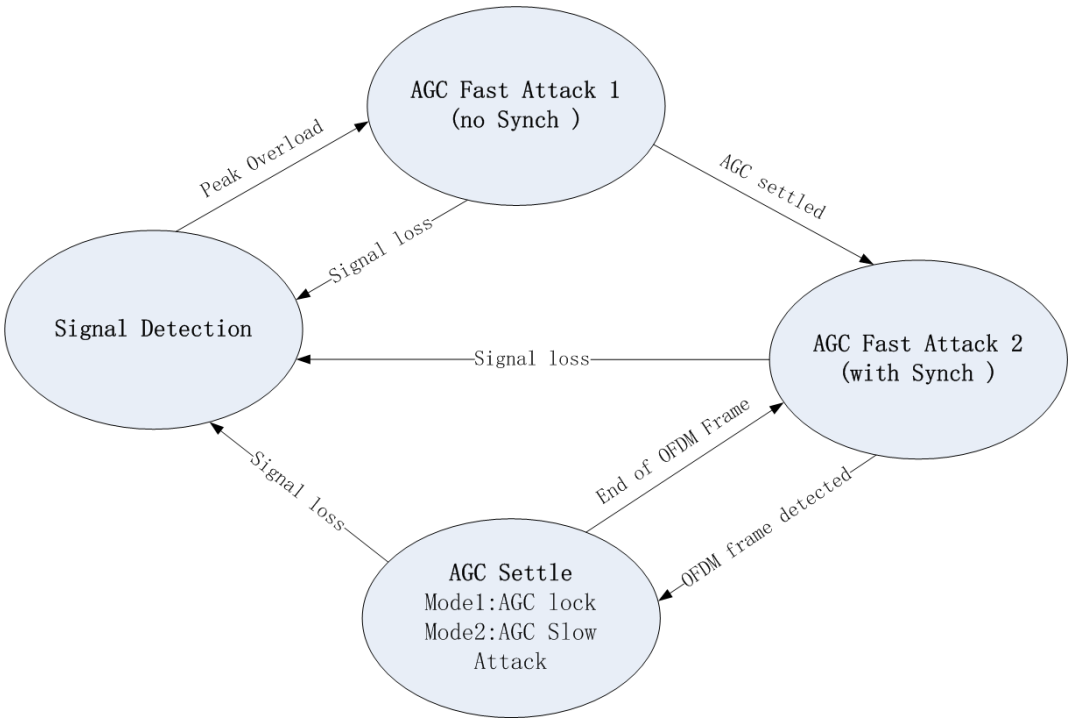


Figure 41: The AGC FSM

constant and the resultant magnitude of the sum will become a large value. This large value indicates the acquisition of data [40]. The transmitted baseband OFDM signal is given by:

$$S_k = \frac{1}{\sqrt{N}} \sum_{n=0}^{N_c-1} c_n \cdot e^{j2\pi kn/N} (-N_g \leq k \leq N-1) \tag{14}$$

in which N_g is the last samples copied to the CP, N_c is the number of tones, and N is number of IFFT points. The received samples at the receiver could be expressed as:

$$S_k = e^{j2\pi\nu n/N} \sum_{m=0}^{l-1} S_{k-te-m} \cdot h_m + n_0 \tag{15}$$

in which ν is the carrier frequency offset normalized with OFDM tone spacing $1/(NT_s)$, te is the delay of symbol, h_m is the channel reponse with channel length l, and n_0 is Gaussian random noise. If two identical sequence are transmitted with the pattern:

$$S_s = [A, A] \tag{16}$$

Assume the first symbol is $S_{1,n}$ and the second symbol is $S_{1,n+L}$. The identity of these two symbols will be maintained after the channel response, except a phase shift (ϕ) due to frequency offset(Δf). The received two

State	Description	Input	Next State	Output
Signal Detection	Detect a significant increase in the signal power level	Free air anolog signals	AGC Fast Attack 1	1. Enable AGC fast attack 2. Enable digital system to prepare receiving signals
AGC Fast Attack 1	AGC fast attack without synchronization	AFE AGC settled indicator	AGC Fast Attack 2	1. Enable digital system to start auto-correlation 2. Stay in the AGC fast attack mode
		Signal loss	Signal Detection	After a delay disable digital system
AGC Fast Attack 2	AGC fast attack with synchronization to detect short preambles	Synchronization pulse	AGC Settle	1. Enable OFDM timing recovery 2. Disable AGC fast attack mode
		Signal loss	Signal Detection	After a delay disable digital system
AGC Settle	The AGC has been completely settled, and start to receive the entire OFDM frame	Preamble symbols and data symbols	AGC Settle	Stay in the state and signal is normal
		End of OFDM frame	AGC Fast Attack 2	Prepare to receive preamble and adjust AGC
		Signal loss	Signal Detection	After a delay disable digital system

Figure 42: The AGC FSM

halves of the sequence are given by:

$$r_{1,n} = s_{1,n} \cdot e^{j2\pi\Delta f n T_s} + n_0 \tag{17}$$

$$r_{1,n+L} = s_{1,n+L} \cdot e^{j2\pi\Delta f (n+L) T_s} + n_0 \tag{18}$$

$$= r_{1,n} \cdot e^{j2\pi\Delta f L T_s} + n_0 \tag{19}$$

Regardless of the noise, the relation of two parts are given by:

$$r_{1,n+L} = r_{1,n} \cdot e^{j2\pi\Delta f L T_s} \tag{20}$$

Based on this relation, the timing location estimator is given by:

$$M(d) = \frac{|P(d)|^2}{(R(d))^2} \tag{21}$$

in which

$$P(d) = \sum_{n=0}^{L-1} (r^{\star}_{d+n} \cdot r_{d+n+L}) \tag{22}$$

$$R(d) = \sum_{n=0}^{L-1} |r_{d+n+L}|^2 \tag{23}$$

As shown in the Figure 45, in every multiplication of $r_{1,n}$ and $r_{1,n+L}$, there is vector been generated shown in the green line. Since all the these products have the same phase and same amplitude, the vectors could be accumulated together to estimate the peak power of the timing metric function.

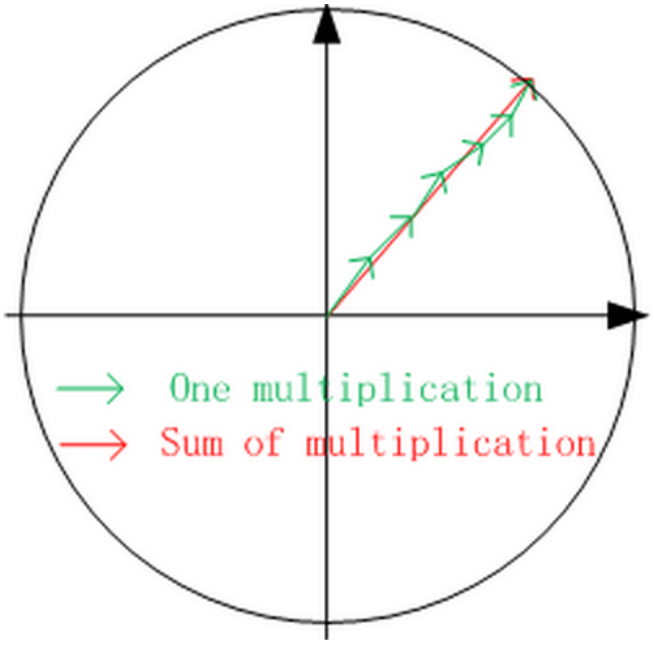


Figure 43: The accumulated multiplication of timing metric

The OFDM frame contains 10 short preambles, so there will be a number of consecutive peaks the the short preamble is detected. the number of consecutive peaks could detemine the start of the OFDM frame. The flow chart of the process is shown in Figure 44.

The simulation of timing metric in one OFDM frame is shown in Figure 45. As given in the graph there are nine peaks was detected in the short preamble and the symbol boundary could be determined based on the position of these peaks.

5.8.2 Frequency Offset

Once symbol timing is recovered, the long preamble could be used to estimate frequency offset. As shown in Equation 20, the phase of the correlation between $r_{1,n}$ and $r_{1,n+L}$:

$$\phi = 2\pi \Delta f L T_s \tag{24}$$

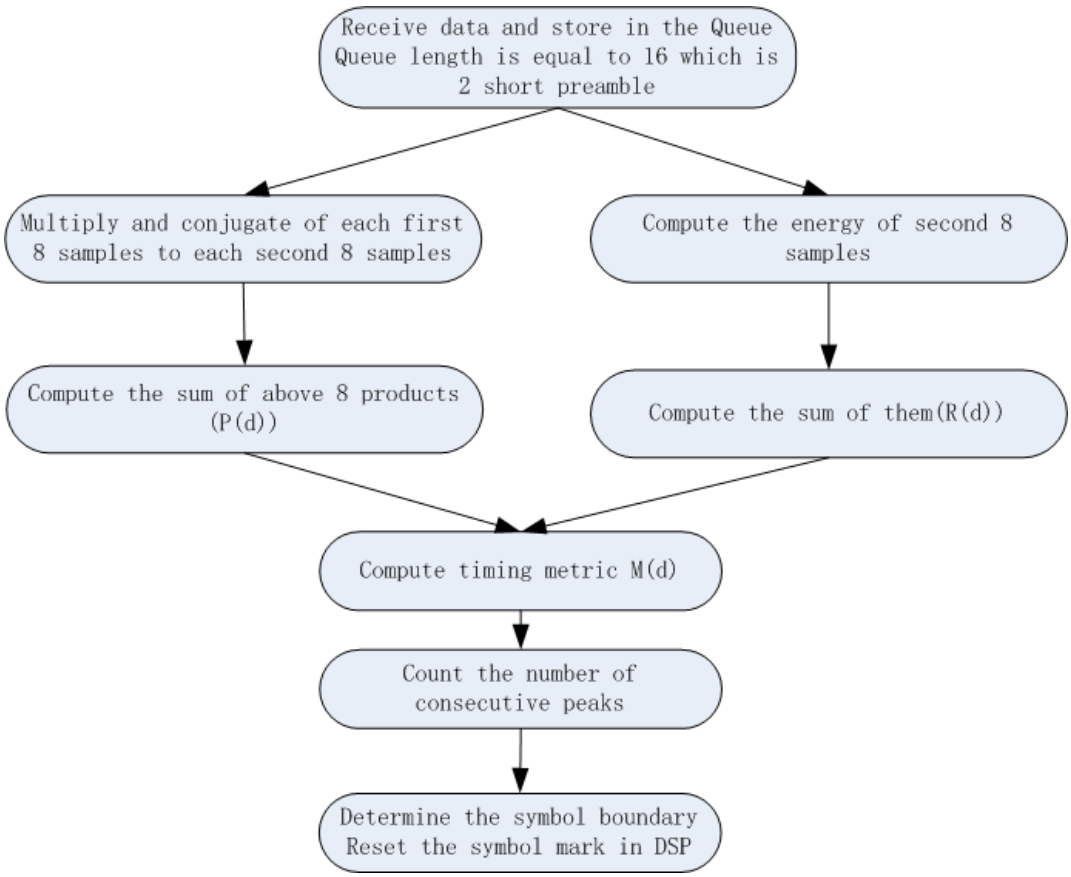


Figure 44: Timing recovery flow chart

can be estimated from Equation 22

$$\phi = angle(P(d)) \tag{25}$$

When ϕ is smaller than π, the frequency offset is given by:

$$\Delta f = \frac{\phi}{2\pi L T_s} \tag{26}$$

when ϕ is larger than π and z is an integer, the frequency offset is equal to:

$$\Delta f = \frac{\phi}{2\pi L T_s} + \frac{2\pi z}{2\pi L T_s} \tag{27}$$

In the long preamble, $L = 32$. The sampling rate is $f_s = \frac{1}{T_s} = 5$MHz. Therefore, if ϕ is larger than π, the Δf should be larger than 78.125Khz. In the satellite system, doppler is expected to contribute up to 150kHz of frequency offset. This being the case, the attitude control team will construct some on-board estimate of the velocity of the satellite. Since the operating system is able to control the AFE via a SPI interface, the majority of this offset can be compensated for by a tuned mixing frequency and the remainder by the synchronisation methods postulated above. Thus, the case for ϕ is larger than π is ignorable. If Flight Systems' measurements turn out to be unreliable, the synchronisation scheme can be adapted to have a larger acquisition range at the sacrifice of some computational complexity in the synchronisation pass.

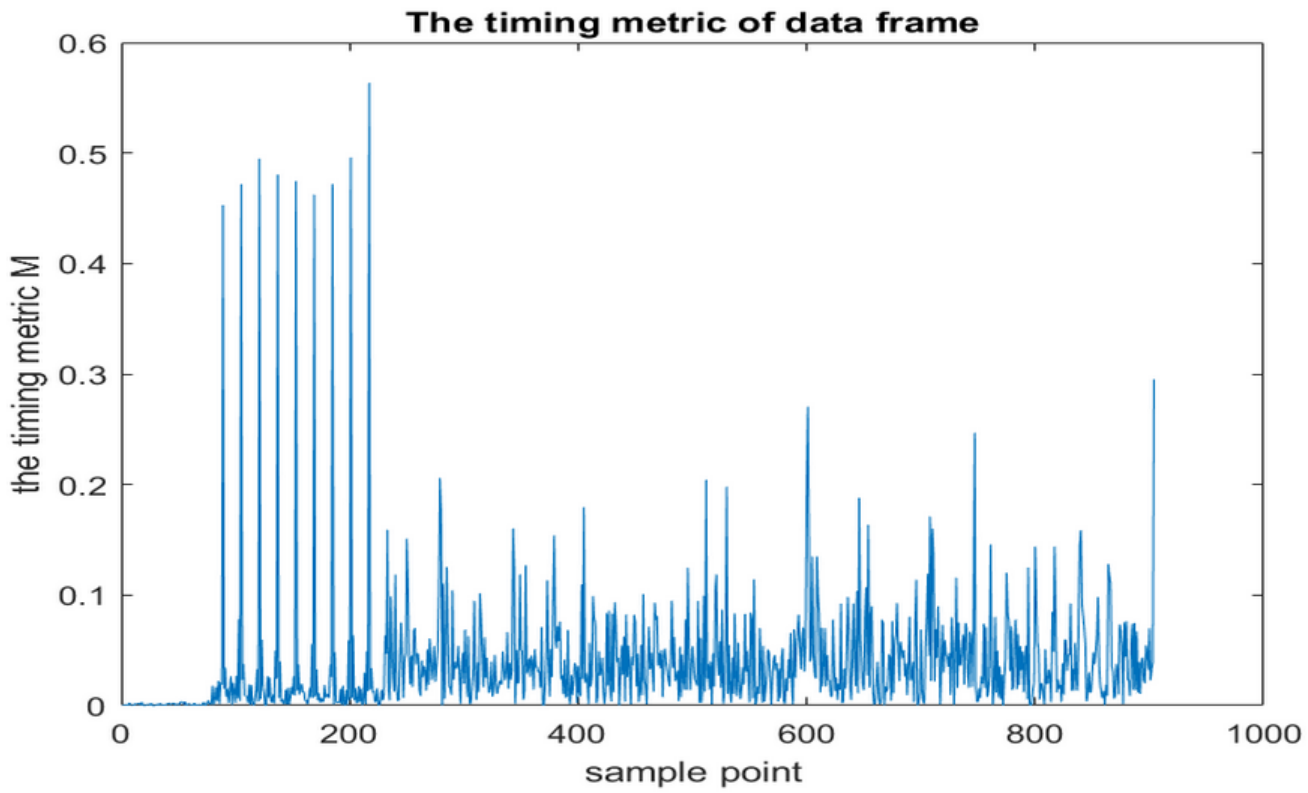


Figure 45: The accumulated multiplications of the timing metric

Since the symbol timing is acquired from the timing recovery, the location of the long preamble is available. The flow chart of the process is given in Figure 46[17]:

[17]In MATLAB OFDM synchronization simulation example "OFDMSynchronizationExample" there is bug for applying the frequency correction. The phase multiplied to signal should be $e^{-j2\pi\Delta f n T_s}$ rather than $e^{j\Delta f n T_s}$

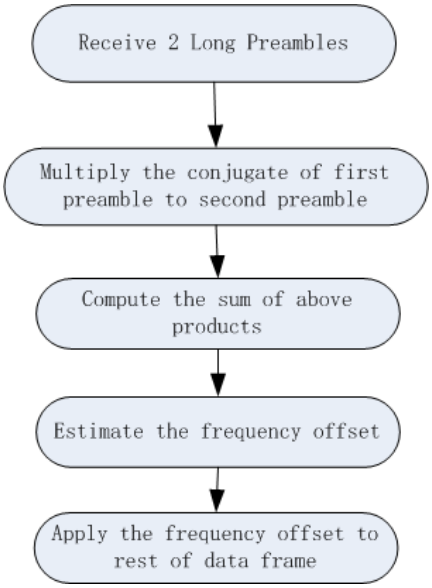


Figure 46: Frequency offset correction flow chart

5.9 Frequency Equalization

5.9.1 Outline

Frequency Equalization (FEQ) is a clever technique employed in the satellite's synchronization and data pass stages. There are three types of error that FEQ corrects for in the system:

- Sampling phase mismatch.
- Frequency selective magnitude compensation.
- Compensation in channel phase errors.

As such, FEQ is often referred to as a 'fine frequency compensation' algorithm since it is able to account for additional phase error after the course frequency compensation module has performed its rectification. FEQ operates by performing an estimate of the channel using the long preamble training sequence displayed in Figure 16. It then performs further correction of evolving phase error due to fine frequency offset via pilot tones inserted into the payload symbols. Since the OFDM framework demodulates into the frequency domain, it is natural to perform equalization techniques with the sub-carriers, not samples. Currently, for the offline system, the base-band performs FEQ with the help of MATLAB, with adequate cycles remaining in the DSP architecture to transfer functionality over on a step-by-step basis.

5.9.2 Implementation and proposal

The Least Minimum Mean Square Estimator (LMMSE) is a common method employed to compensate for magnitude and phase in both fast and slow fading channels [41]. The initial channel estimate in MATLAB is given by:

$$\hat{H}_{pre}[k] = \frac{E[\frac{Y^*_{pre}[k]}{X^*_{pre}[k]}]}{E[\frac{Y_{pre}[k]}{X_{pre}[k]}\frac{Y^*_{pre}[k]}{X^*_{pre}[k]}]} \tag{28}$$

$$\hat{H}_{pre}[k] = \frac{E[H_{pre}[k]]}{E[H_{pre}[k]H^*_{pre}[k]]} \tag{29}$$

where:

- $Y_{pre}[k]$ is the received preamble sequence.
- $X_{pre}[k]$ is the transmitted preamble sequence.
- $\hat{H}_{pre}[k]$ is the channel estimate based on the preamble training sequence.

Since pilot assisted modulation (PSAM) is used to correct for evolving frequency offset, channel phase and channel amplitude, it's necessary to update this estimate with the multiplication of the $H_{pil}[k]$ throughout the duration of the data frame.

Hsieh and Wei [41] propose a pilot tone interpolation method via the application of a second order polynomial, which can out-perform linear piece-wise interpolation methods while maintaining a low complexity. The interpolator is given by:

$$\hat{H}[k] = \hat{H}[mL + l] = C_1\hat{H}_p[m-1] + C_0\hat{H}_p[m] + C_{-1}\hat{H}_p[m+1] \tag{30}$$

$$C_1 = \frac{\alpha(\alpha+1)}{2} \tag{31}$$

$$C_0 = -(\alpha-1)(\alpha+1) \tag{32}$$

$$C_{-1} = \frac{\alpha(\alpha-1)}{2} \tag{33}$$

$$\alpha = \frac{l}{N} \tag{34}$$

when m is the pilot tone number, L is the pilot tone separation, N is the total number of subcarriers, l is the interval between a given data carrier k and pilot tone at position mL. This method has largely been selected since it will not contribute significant complexity to the DSP.

5.10 Optimising Code

Currently, the clock cycles consumed by each blocks are in Tables 16 and 17.

Transmitter	**Clock cycles consumption**
Memory read	35
Convolutional encoder	37
Interleaver	1284
Mapper	571
IFFT	703
Total clock cycles	**2630**

Table 16: Total clock cycles used in the transmitter data-pass.

Receiver	**Clock cycles consumption**	**Comments**
FFT	472	Well optimised.
FEQ	2492	Change to fixed-point multiplications.
Demapper	223	Quad memory access.
Deinterleaver	1207	Bit-wise implementation. Quad memory access.
Viterbi decoder	1057	Quad memory access.
Write to memory	335	Quad memory access.
Total clock cycles	**5786**	

Table 17: Total clock cycles used in the receiver data-pass.

The clock cycles for the transmitter and the receiver are required to be below 8000 per symbol, since the 500MHz processor has 16μs to generate the next symbol. Asserting a limit of 4000 leaves contingency for emergencies and functionality expansions.

It should be noted that the FEQ costs too many clock cycles. This issue can be optimized if we use the 32-Bit Fixed-Point Complex Multiplication Instructions [42], which will be executed in the next stage of the project. In addition, some blocks, even though use only a few clock cycles, still have the space of optimization on the assembly level.

6 Analog Front End

The final system will utilise an Analog Devices AD9364 analog front end chip. This supports central frequencies between 70MHz and 6.0GHz with bandwidths between 200kHz and 56MHz [5]. Importantly, being desired for 4G applications, it supports OFDM and can communicated with the TigerSHARC over low voltage differential signalling (LVDS). This constraint removed a significant number of chips from consideration. The team was lucky enough to receive four of these evaluation boards from Analog Devices as a university donation.

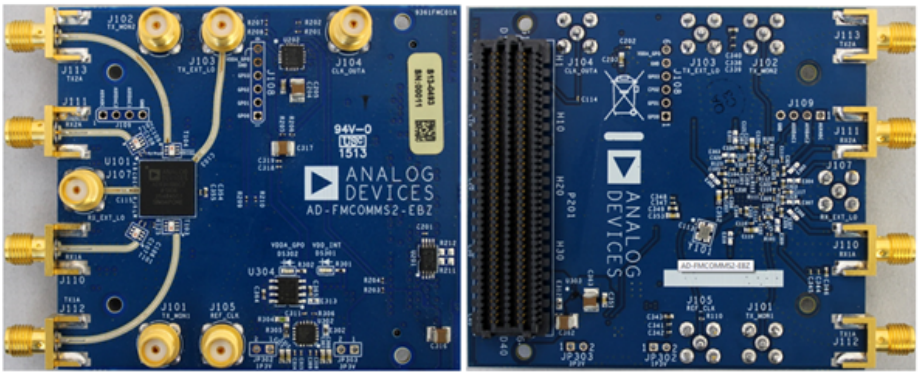


Figure 47: AD9364 evaluation board top and bottom [12]

For the purposes of testing, the centre frequency was set to 2.4GHz. This was done for a number of reasons: no legal restrictions on transmitting apply in this band, antennas were easy to acquire, and the baluns on the evaluation boards were optimised for operation below 2.4GHz. A balun is a device that converts a *bal*anced signal to an *un*balanced signal, which is where the name comes from. A simple example of a balun is the RF choke seen on the end of laptop power charging cables to prevent high frequency electromagnetic signals from being either radiated from the source or being received by the cable and passing into the system under protection. The main purpose of a balun is to reject common mode noise while passing desired differential mode signals. Furthermore, 2.4GHz was an appropriate centre frequency since WiFi transmissions could be picked up and considered in the received signal as noise. In the absence of other noise-producing circuits, this was acceptable.

6.1 Required Hardware and Software Tools

The DSP and AFE chips are both small ball grid array (BGA) packages that are difficult to rout PCBs for, and even harder to debug issues. An entire Capstone project could be devoted to routing a six or eight layer board for the telecommunications system, but the capabilities of the chips must first be tested, and the algorithms running on these chips must be tested with as little additional complexity introduced as well. As such, an evaluation board stage was required to test the capabilities of the hardware and the code to be implemented on it.

The AD9364 evaluation board includes the chip itself, SMA connectors for antennas and an FPGA Mezzanine Card

(FMC) connector. SMA pigtail cables and and SMA 2.4GHz antennas were purchased to allow the transmitter and receiver circuits to communicate. The discussed above, the final PCB will utilise LVDS to communicate, which was designed for communication within the same piece of hardware, such as between the screen and motherboard of a laptop. It was optimised for low power usage over short distance at the expense of long range, and so is not appropriate to implement a free LVDS cable. As such LVDS is not supported in the evaluation board packages. Instead, the pins are broken out into a male FMC connector, so a driver board with a female FMC connector was required. The board selected was the Xilinx Zedboard as seen in Figure 48, two of which were donated by Digilent/Xilinx for this project.

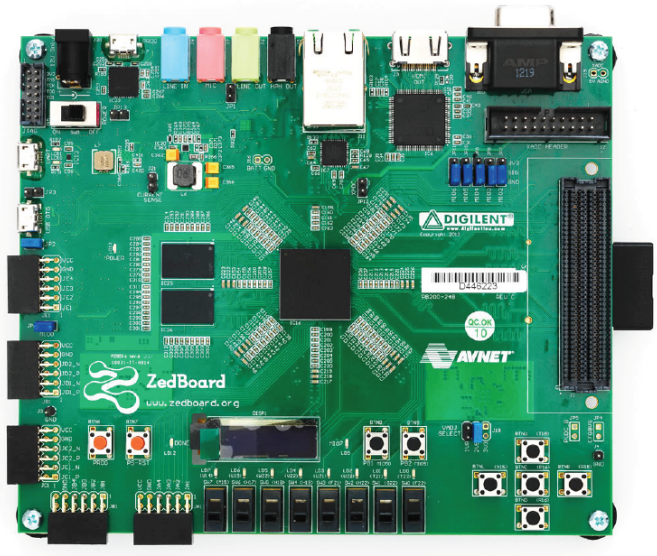


Figure 48: Zedboard image [13]

This was chosen partially because of its low $500USD cost. Although these boards were a donation, low cost was a factor if the team needed to replace broken boards or purchase more. Significant amounts of documentation existed, and MATLAB provided numerous example Simulink models and MATLAB code to drive the board. Looking further ahead, the board was also capable of executing the required instructions to allow realtime demonstration.

This board is a breakout for the Zynq-7000 chip, which contains a processor side and an FPGA component as seen in Figure 49. This meant that the first test to determine whether the hardware was functional consisted of booting from a bare Linux distribution and controlling some LEDs on the board. Some built-in C code also allowed a scope to be viewed by connecting a computer screen to the HDMI port of the Zedboard and running the scope program from the Linux command line.

6.2 MATLAB Implementation

There were a number of ways to drive the AFE. The processor and programmable logic could have been hard coded, but this approach was not taken for the Capstone demonstration due to time constraints, plus the fact that getting

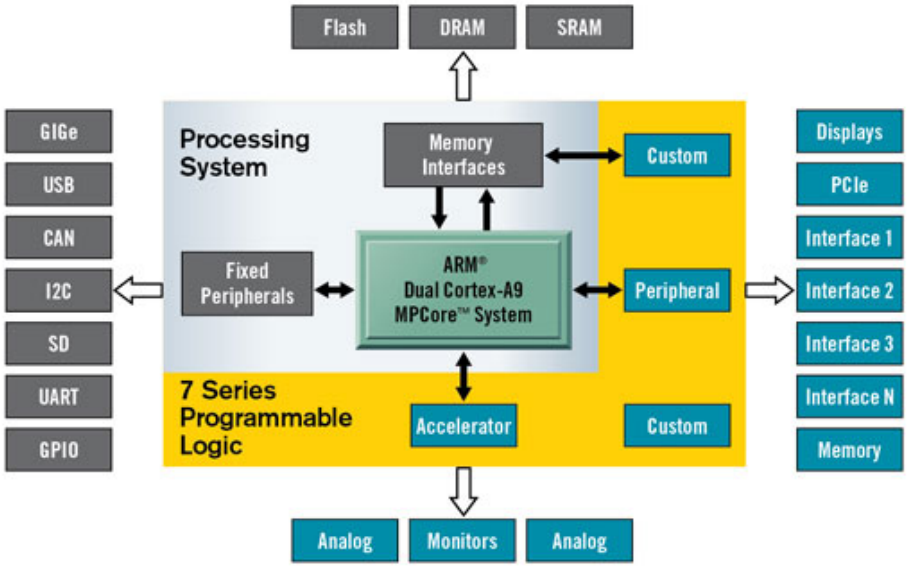


Figure 49: Zynq-7000 chip architecture [14]

the FPGA working would have advanced progress at the tangent to the final objective, since the final flight PCB will only have the DSP and AFE chips on it. Fortunately, MATLAB provided a satisfactory solution in a significantly less time: one of the major reasons for selecting the Zedboard was its interoperability with MATLAB. The host computer and the Zedboard were connected via a Gigabit ethernet connection, which prevented the authors from implementing an online demonstration. Without flashing the programmable logic of the Zynq-7000, a constant connection to the host computer was required, which meant that the Gigabit ethernet port of the DSP evaluation board couldn't connect to the ZedBoard, as shown in Figure 11 on page 19. Furthermore, learning how to access the ethernet port through code was likely to be time consuming, and difficult to achieve within the time frame.

Therefore, the idea of hard coding the AFE modules was abandoned in favour of executing Simulink models which could automatically identify, authenticate and connect with external hardware. sdrRx and sdrTx Simulink blocks were available as part of the Communications Systems Toolbox in MATLAB, which contained dialogues to vary the sampling frequency and mixing frequency among other parameters. On the receiver side, code would pull complex doubles from a data file then transmit them through the sdrTx block. On the receiver side, code was written to implement synchronisation as described in Section 5.8 and write the synchronised complex doubles to a data file to be handed back to the DSP. This MATLAB code can be found in Appendix C. Getting the toolchain working proved difficult due to the multitude of different add-in packages required, but the issue was eventually overcome.

During the first days of testing, egregious bit error rates of 4% were observed when transmitting at 2.4GHz at a sampling rate of 520.841kHz over an SMA antenna, and this error rate was not reduced when a wired connection was used. This rate was used first as it was the default, and getting the system functional with this sampling rate was the first step on building toward 5MHz. Furthermore, no signal power was received when the signal bandwidth was increased to 5MHz. By outputting the received bitstream, it was seen that the only bits to ever be in error were those at either end of the 48 bit symbol. This indicated that the internal AFE filter was causing errors: in fact the

default filter settings asserted a passband frequency of 173.61kHz and a -80dB stopband frequency of 217.02kHz, so data carrying tones 0-2 and 45-47 were being filtered out, and these were often decoded incorrectly. Every OFDM symbol takes up the full bandwidth, in order to provide subcarrier orthogonality at the given sampling rate, so the AFE filter was converted to an all-pass. Separately implementing guard-bands is not required, since the OFDM protocol explicitly includes these.

A fundamental hardware limitation prevents testing of this system at a 5MHz sampling rate: MATLAB receives complex doubles, so a 5MHz sampling rate would correspond to bare data transmission rates of 640MB/sec, just below the theoretical upper limit of Gigabit ethernet. This causes a buffer underflow to occur at the transmitter, since the AFE was trying to 'grab' samples before MATLAB could supply them. Therefore, for the demonstration and the Results section, the default 520.841kHz sampling rate was employed.

6.3 Determining Start of Transmission

In this deskign phase, the transmitter side is set to continue repeating its data for 100secs, and the receiver model is executed sometime within that. As a first step to present this report, this can determine whether the synchronisation algorithm in particular is functional. It is understood that this doesn't mimic the final implementation, but is an important stepping stone.

A future Capstone group may implement a more realistic scheme will be implemented: summing up the received power across some time window and comparing this to a threshold energy value. This comparison will determine whether the transmission has begun or not, and tell the reception hardware whether to begin the reception process.

6.4 Automatic Gain Control

For the preliminary and final reports, Simulink models will drive the AFE. Parameters such as filter tap coefficients are abstracted away by a filter design wizard, as are the settings for AGC. For testing, AGC Slow Attack was selected from a MATLAB dialogue. Information this detailed is held within the MATLAB environment, and not released to the end user. If the gain details need to be verified, GNU Radio can be run to break out all AFE parameters.

The Simulink model in Figure 50 includes an SDR which outputs complex time domain signals which are fed through a broken-out AGC block to amplify the samples up to an average power of 250mW over 100 samples. This is output to a file to be dumped into the DSP simulator's memory. The QPSK block is there as a testing relic: it performs timing recovery and hence was used to test the channel by transmitting a character string as the encoder output.

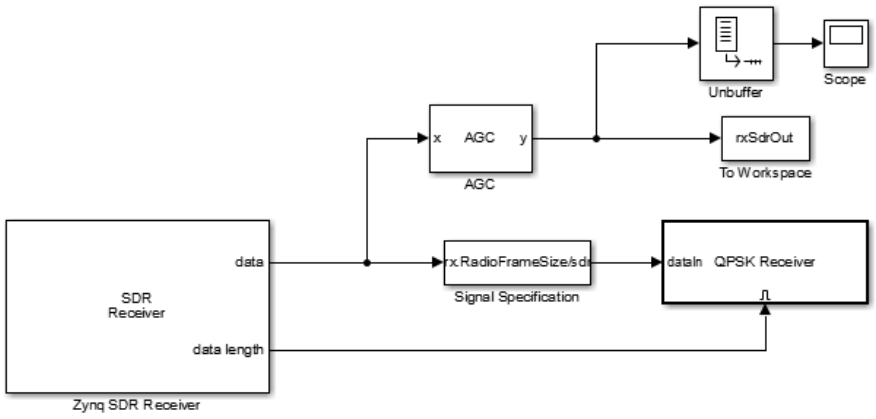


Figure 50: OFDM Reception Simulink model

6.5 Future Work

This evaluation system was selected with further expansion in mind: the Zedboard donation included software licenses for Vivado, which can be used in conjunction with MATLAB's HDL coder to flash HDL code to the Zynq-7000's programmable logic. This will enable a future evaluation stage with the Zedboard and DSP communicating in real time. Since the Zynq-7000 contains a processing system (PS) and programmable logic, C code can be writen on the device and executed just as C code would be on a Raspberry Pi. Generating the HDL code can be done by passing MATLAB code through the MATLAB HDL Coder Package, then porting this to Vivado for hardware-implementation. This will enable the DSP and AFE evaluation boards to communicate in real time.

7 RF Antenna Design

7.1 Motivating the Inflatable Antenna

Today, the low-cost of nano-satellite launches are overshadowed by their low bit-rate performance, and therefore cannot utilise payloads that require higher data rates. Such payloads might include high data-rate telemetry, high resolution LEO observation satellites, and other complex scientific payloads.

The low power, low mass and low volume nature of nano-satellites prompts industry to be creative with their data-link to achieve data rates meeting the needs of modern payloads. For the purposes of this proof-of-concept satellite, the team have elected to develop an inflatable antenna that will meet high SNR requirements to achieve OFDM modulation through space, without substantially reducing orbital life-time when in lower orbits. At lower orbits (310-400km), Flight Systems' orbital models report that orbital drag accounts for a substantial factor in the satellite life-time [43].

The design initiative was motivated by the development team at Massachusetts Institute of Technology, who are working towards developing their own inflatable antenna for 2U and 3U satellites [17].

The Capstone team have provided firm system specifications to the payload team, who are at this time verifying and iterating the proposed topology. The payload team are also engineering the inflation system, where they are considering the mass of benzoic acid required to achieve appropriate pressurisation at vacuum.

Since work of the Capstone group does not extend to analysis of the inflation process, the results of their work will not be presented in this section. There will instead be specific detail on the RF design process of the antenna.

7.2 Primary Specifications

Requirement	Source	Value	Units
Antenna Gain	Link Budget	15	dB
Antenna Volume	Volume Budget	200	cm^3
Reliability	-	95%	-

Table 18: The primary requirements of the antenna design.

In addition to these quantifiable specifications, the antenna should be applicable and flexible to the volume requirements of future cube satellite missions and should be a low cost design. Circular polarization can be achieved with appropriate antennas consisting of orthogonal dipoles or orthogonal patch elements.

7.3 Overview of high gain antenna topologies

7.3.1 Frontal feeder parabolic

The frontal feeder geometry is the most reliable parabolic antenna topology with the lowest number of moving parts. It is the topology currently being developed by MIT [17], consisting of a patch antenna, feed horn, and primary reflector to collate the EM waves to achieve gain.

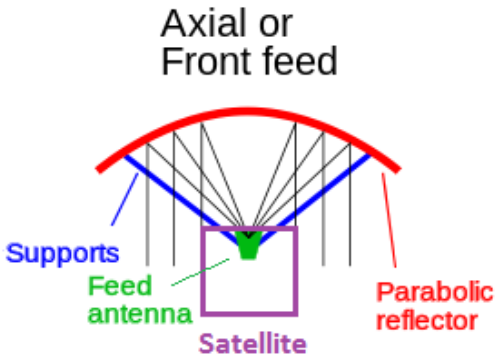


Figure 51: Illustration of a frontal feed antenna.

Merits of the frontal feeder include its high gain which is proportional to the size of the primary reflector, and also the expandability of the design. Disadvantages include that during communications the satellite is in the path of the high frequency radiation, potentially causing interference to the RF equipment in the transceiver or other precision measurement devices. This impeded path will also result in a reduced transmission power and consequently reduce the gain of the antenna at the satellite ground-station. A final disadvantage of the frontal feeder is that the inflatable will block the direct path of sunlight to the satellite's solar panels during the day-time. When the satellite is charging, it would require actuation of the magnetorquers to rotate the satellite so panels face the sun and are able to charge the battery. In other words, the satellite would never be able to harvest power without and transmit at high-speeds at the same time.

All parabolic inflatable designs require some mechanism to deploy. At the time-being this mechanism is proposed to be capsule, which doubles as the antenna feedhorn, coupled with a cap that is separated with the addition of nichrome burn wire. In the frontal feeder design, this cap serves no purpose other than to initiate this inflation function.

7.3.2 Cassegrain parabolic

A Cassegrain topology differs from the frontal feeder geometry in that it utilises the placement of a secondary reflector to change the direction of data-travel with respect to the feeder location.

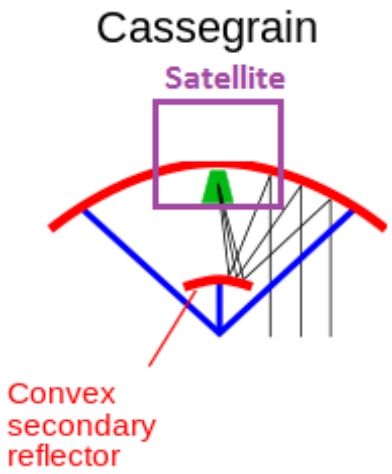


Figure 52: Illustration of a Cassegrain antenna.

The Cassegrain geometry solves the problem of satellite power harvesting by placing the antenna in the direction of data transmission, rather than in the direction of the sun. It also addresses the concern that satellites may perform badly when subject to high intensity self-interference. Disadvantages include that is inherently less reliable than the frontal feeder due to the addition of a secondary reflector that is needed to provide gain. Furthermore, the small-scale geometry of the secondary reflector needs to be appropriately considered when selecting frequencies to communicate on.

A final advantage which came about with some design ingenuity, is that the Cassegrain design can use the capsule's cap as the secondary reflector, effectively doubling the value of volume otherwise used in the frontal feeder topology.

7.3.3 Gregorian parabolic

The Gregorian topology is almost identical to the trade-offs of the Cassegrain design. It only differs in its deployment of an ellipsoid secondary reflector (concave) as opposed to the hyperbolic secondary reflector (convex) of the Cassegrain. This arises due to the parabola's focus point now being located at a lower height than the secondary reflector. This mere observation results in a larger volume requirement to host a Gregorian type geometry.

7.3.4 Helical and Yagi-Uda Antennas

Helical and Yagi antennas both are relatively large devices, which are commonly used in satellite and long distance radio communications. They can typically achieve a limited high gain (around 10-12dB) but would require large internal volume requirements and complicated deployment mechanisms due to their rigid structures.

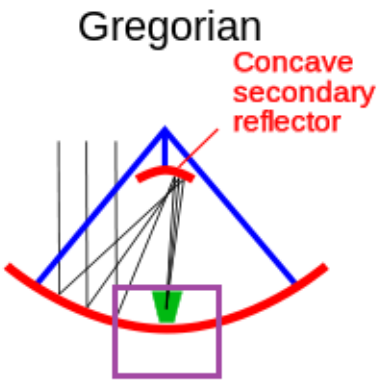


Figure 53: Illustration of a Gregorian antenna.

7.4 Design Methodology

7.4.1 Design Cost Analysis

In order to settle on an appropriate antenna geometry, the team conducted an appropriate cost analysis on various high gain antenna designs. This design cost function is summarised in Tables 19 through 21.

High gain design	Volume	Gain	Reliability	Cost	Flexibility
Frontal Feeder	10	8	8	10	5
Cassegrain	8	9	7	10	9
Greogorian	6	9	7	10	9
Helical	3	7	10	8	5
Yagi-Uda	2	7	10	8	4

Table 19: The design incentives of each topology for each of the primary requirements.

	Volume	Gain	Reliability	Cost	Flexibility
Weightings	0.25	0.25	0.25	0.05	0.2

Table 20: Weightings assigned to each primary design requirement.

Chosen weightings and specifications were decided on among the team members. As observed in Table 21, the total incentive to persist with Cassegrain antenna design was larger than any other antenna, so it was selected for preliminary design, simulation and prototyping.

High gain design	Volume	Gain	Reliability	Cost	Flexibility	Design Incentive
Frontal Feeder	2.5	2	2	0.5	1	8
Cassegrain	2	2.25	1.75	0.5	1.8	8.3
Greogorian	1.5	2.25	1.75	0.5	1.8	7.8
Helical	0.75	1.75	2.5	0.4	1	6.4
Yagi-Uda	0.5	1.75	2.5	0.4	0.8	5.95

Table 21: Total design incentive after the design weightings were applied to each of the antennas.

7.4.2 Cassegrain Design

7.4.2.1 Overview

The antenna will consist of a patch, coupled with a feed-horn, hyperbolic secondary reflector, and parabolic primary reflector. In order to optimise for volume, the team elected to connect the reflective paraboloid with a non-reflective ellipse rather than a sphere.

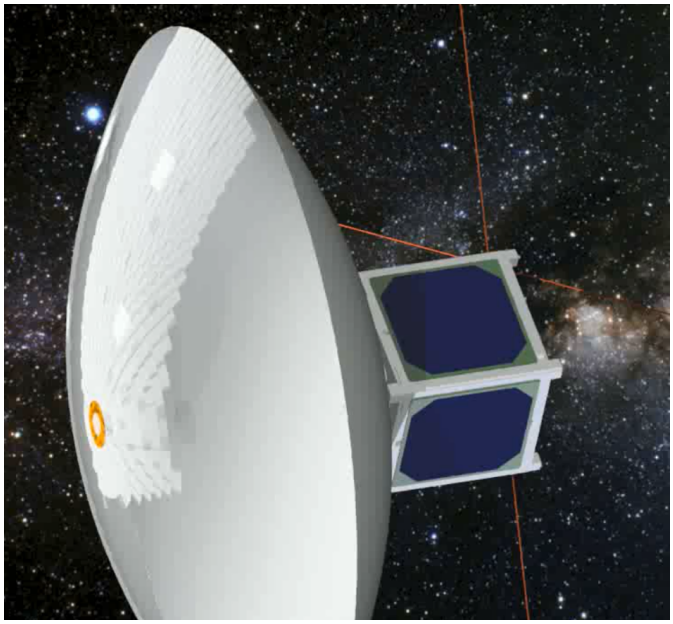


Figure 54: CAD model of inflatable antenna [1]

While this complicated the parametric analysis, it resulted in a more compact volume that ultimately translates to a higher orbital lifetime. The gain [44] of a parabolic antenna can be calculated as:

$$G = 10\log_{10} k \left(\frac{\pi d}{\lambda}\right)^2 \tag{35}$$

Where: G is the gain over an isotropic source in dBi, k is the efficiency of the antenna, assumed to be 50%, d is the

diameter of the parabolic aperture in meters and λ is the wavelength of the signal in meters. This is the primary equation used to calculate the diameter of the dish. However, there are more constraints that arise due to the fact that the antenna will be contained inside a balloon, including:

- The inflated volume must be as low as possible to reduce drag.
- The primary reflector paraboloid must continuously connect to the transparent material, holding the secondary for good shape retention.
- In the downlink, all RF reflected from the secondary must be also reflected from the primary.
- In the uplink, all RF reflected from the primary must be also reflected from the secondary.
- The secondary must be large enough to sufficiently cover the feed-horn's beamwidth.
- The secondary must be small enough to reasonably fit into the satellite's chassis.
- The flaring of the feedhorn must be large enough to create a sufficiently small beamwidth.
- The geometry must be large enough to operate efficiently at the selected communications frequency of 5.65 GHz and 5.83 GHz.

7.4.2.2 Secondary reflector design

In order to satisfy the listed constraints, a MATLAB package was designed to conveniently adjust and trade-off the system design parameters. The constraints outlined in Equations (38)-(42) are enough to capture a complete parametric description of the secondary reflector for a simple ray-tracing model when employing traditional descriptions for a hyperbola and parabola in 2-dimensions.

Offset Hyperbola

$$\frac{(y - h_{hyp})^2}{a_{hyp}^2} - \frac{x^2}{b_{hyp}^2} = 1 \tag{36}$$

Parabola

$$y = dx^2 \tag{37}$$

Figure 55 describes a set of variables required to define the various parameters in the above equations.

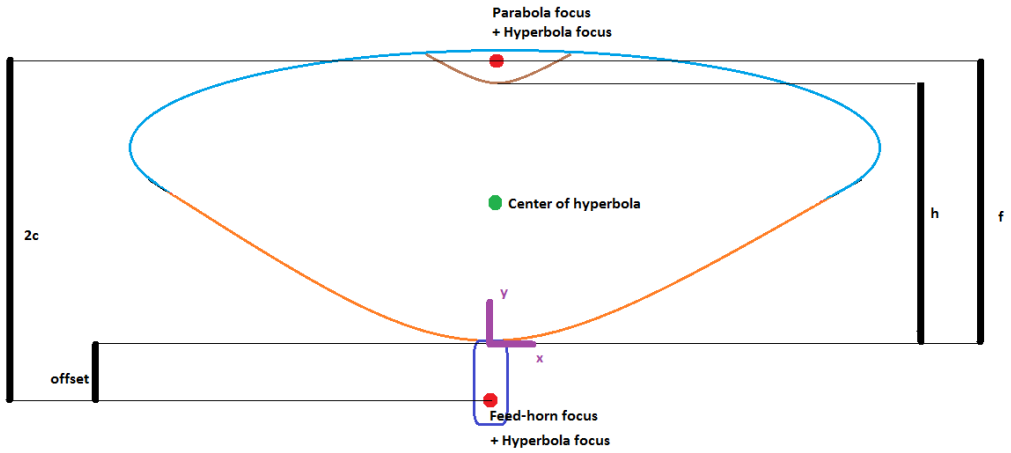


Figure 55: A diagram of the variables used to create a complete parametric analysis of the antenna geometry.

Geometric pairing of hyperbola and parabola [44]

$$\text{Hyperbola offset: } h_{hyp} = c - h_{offset} \tag{38}$$

$$\text{Hyperbola eccentricity: } e = \frac{c}{a_{hyp}} \tag{39}$$

$$\text{Hyperbola focal distance: } c = \frac{f_p + h_{offset}}{2} \tag{40}$$

$$\text{Hyperbola vertical axis: } a_{hyp} = h - (c - h_{offset}) \tag{41}$$

$$\text{Hyperbola horizontal axis: } b_{hyp} = \sqrt{c^2 - a_{hyp}^2} \tag{42}$$

where:

- h_{offset} is the offset of the focal point of the feedhorn
- f_p is the focal point of the parabola.

7.4.2.3 Ellipsoid envelope design

Similarly, the MATLAB package must construct a description of the antenna's transparent ellipsoid envelope.

Offset ellipse

$$(\frac{x}{a_{ell}})^2 + (\frac{y - h_{ell}}{b_{ell}})^2 = 1 \tag{43}$$

Condition of secondary reflector attachment

$$y_{hyp} = y_{ell}|_{x_{hyp}=rL} \tag{44}$$

where:

- The pair (x_{hyp}, y_{ell}) is a valid point on the ellipse.
- The pair (x_{hyp}, y_{hyp}) relates to highest point of the secondary reflector.
- L is the ratio of the secondary reflector radius to the primary reflector radius.

After some lengthy algebra, condition (44) yields

$$h_{ell} = \frac{2dy_{par}(x_{hyp}^2 - x_{par}^2) - (y_{hyp}^2 - y_{par}^2)}{2(d(x_{hyp}^2 - x_{par}^2) - (y_{hyp} - y_{par}))} \tag{45}$$

where:

- x_{hyp} is the radius of the hyperbola.
- y_{hyp} is the height of the hyperbola at $x = rL = x_{hyp}$.
- x_{par} is the radius of the parabola.
- y_{par} is the height of the parabola at $x = r = x_{par}$.
- d is the diameter of the parabolic reflector.

Condition of continuity The ellipsoid envelope must be smoothly connected to the paraboloid reflector.

$$\frac{dy_p}{dx}|_{x=x_{par}} = \frac{dy_h}{dx}|_{x=x_{par}} \tag{46}$$

This yields:

$$b_{ell} = \sqrt{1 - (y_{par} - h_{ell})^2 - 2d(y_{par} - h_{ell})x_{par}^2} \tag{47}$$

$$a_{ell} = \sqrt{x_{par}^2 - \frac{y_{par} - h_{ell}}{2d}} \tag{48}$$

7.4.2.4 Verification of analysis via an independent ray tracing routine

To verify the geometry forged by the aforementioned relationships, it is important to compound the MATLAB script with a display of how rays pass through the antenna. This will serve as verification that the geometry can be passed onto the payload team for more rigorous RF simulation, and give the flight systems and flight vehicle teams a very good indication of the final volume requirements of the antenna system.

It should be noted that the routine of Figure 57 will not be used to predict antenna gain, but can be used as a means to easily identify losses due to parabolic/hyperbolic beam mismatches. When implemented into MATLAB's GUI environment, the effectiveness of the equations can be easily verified by inspecting the plot generated inside the interface. The parameters relevant to the tuning of the design can be entered into the model via the toolbar

beneath the plot. See Figure 56. The current input parameters are the ones used to develop our 1^{st} official prototype geometry.

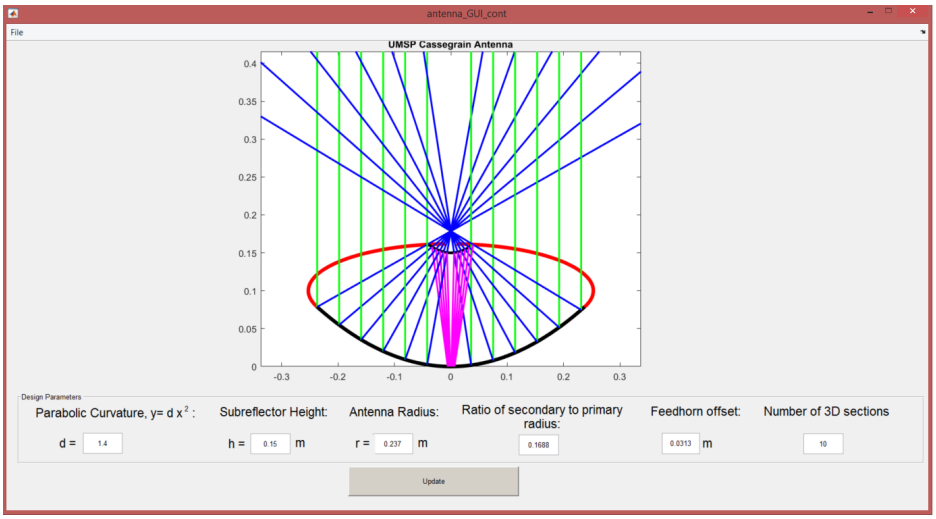


Figure 56: The MATLAB GUI environment, indicating the tunable design parameters and the generated antenna geometry.

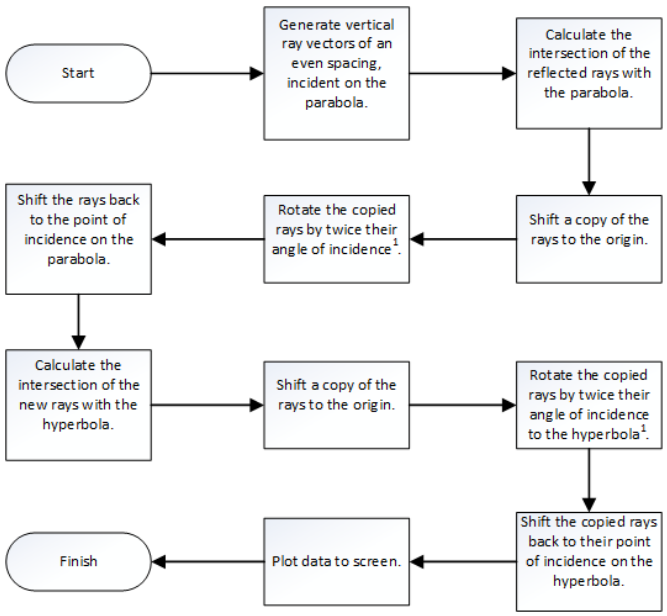


Figure 57: Routine for geometric verification by ray tracing.

7.4.2.5 Antenna design for 3-dimensional prototyping

Clearly, the methods provided so far are sufficient only to build up a 2-Dimensional illustration of the inflatable structure, and further effort is required to move this model towards a 3D prototype. Since the structure is not rigid, we cannot use the power of AutoCad to rotate our parabola by 360 degrees for the purposes of, for instance, 3D printing. Since we have proposed to construct this antenna from Mylar, we must create a pattern of 2-Dimensional

cuts required to achieve this 3D shape, once pieced together. Instead there is a need for a little further creativity with MATLAB. Since it is complicated to make a curved 3D structure from a single piece of material, the group has elected to piece N smaller pieces of Mylar together to achieve a sufficient approximation of the paraboloid and ellipsoid shapes. For the paraboloid this is relatively straight-forward. We can solve a line-integral along the length of the parabola, as a function of its height in the z-direction.

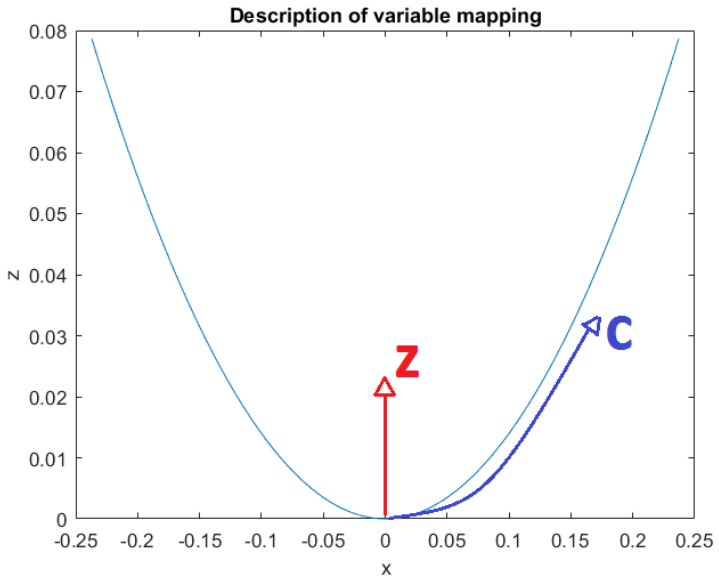


Figure 58: Definition of coordinate systems required to establish 2D cuts.

The mapping to the z plane is described by (50)

$$c(z) = \int_0^c \sqrt{(\frac{dz}{dz})^2 + (\frac{dx}{dz})^2} dz \tag{49}$$

$$c(z) = \int_0^c \sqrt{1 + \frac{1}{4dz}} dz \tag{50}$$

We can also obtain the circumference of a contour at a height, z, via the simple equation:

$$C(x) = 2\pi x$$

$$C(z) = 2\pi\sqrt{\frac{z}{d}}$$

where $z = dx^2$. Substituting in the inverse of the solution for $c(z)$, $z(c)$, we can theoretically obtain an expression for $C(z(c)) = C(c)$, required to detail the sections of inflatable material. Finally, we can divide the circumference of the parabola, C at some arc length, c, by N, the number of sections of our paraboloid. However, since the equivalent expression of (50) for the ellipse:

$$c(z) = \int_0^c \sqrt{\frac{1 + z^2 a^2}{b^2(b^2 - z^2)}} \tag{51}$$

is unsolvable, we have elected to use MATLAB to compute the numeric integral. The methods employed by the MATLAB script to calculate the ellipsoid cuts are outlined in Figure 59, and is identical to the method employed to calculate the paraboloid cuts.

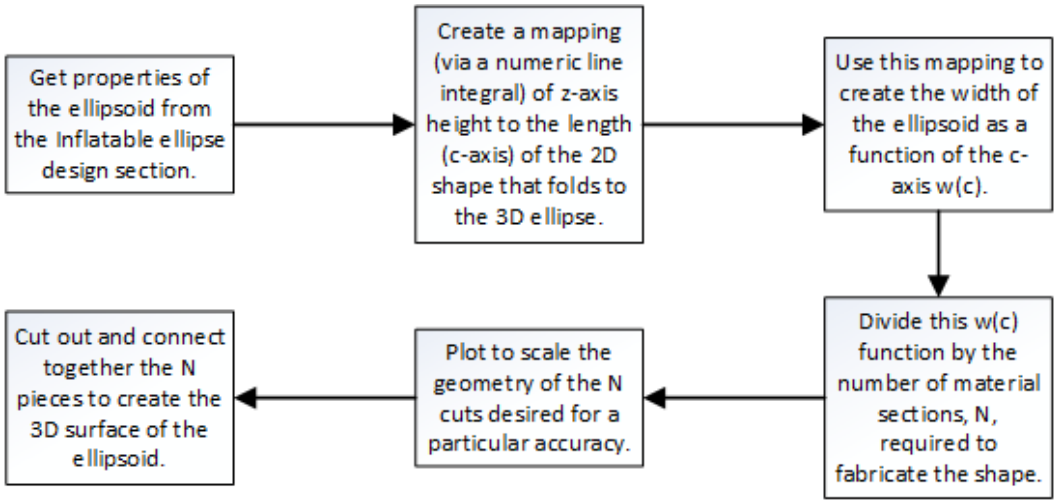
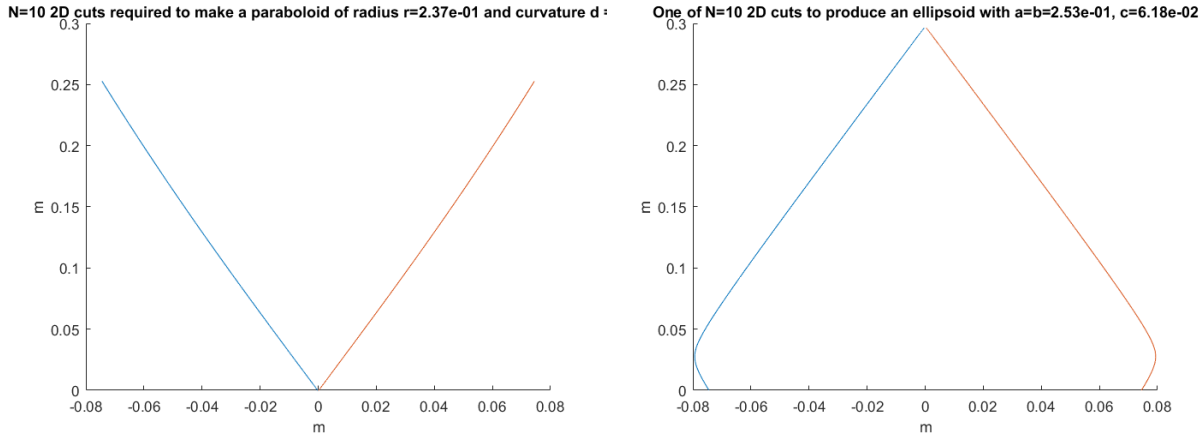


Figure 59: The MATLAB routine employed to generate the 3D cuts.

When outputted to a plot in MATLAB, the 3D geometry could be realised through welding sections of cuts detailed in Figure 60.



(a) (b)

Figure 60: One of N cuts required to achieve both the paraboloid geometry (left) and ellipsoid geometry (right).

Figure 61: The first inflatable antenna prototype of geometry.

7.4.2.6 Assembly

Due to the novel design of the proposed antenna, the cuts require manual assembly. Many efforts were conducted to obtain consistent welds using a PCB etcher located in the Electrical and Electronics Engineering building at Melbourne University. The etcher looked capable of welding two pieces of Mylar together, since the power was finely adjustable either side of the Mylar's melting point. Difficulties with consistency persisted, resulting in many inconsistent welds and rendering the process completely infeasible for assembly. Other group members have since tested other methods of combination by heat application and it was discovered that the selected Mylar was in fact not heat sealable.

Newer samples ordered by the group had a much better response to thermal clamping, warranting further investigation with the EEE departments PCB etcher. Alternative and/or additional sealing methods explored by the team include use of an Araldite epoxy adhesive, which currently is being used to seal the current inflatable prototype.

7.4.2.7 Trade-offs in RF design

As alluded to earlier, there are many trade-offs involving volume, carrier frequency, gain and bandwidth to ensure an optimal antenna design. The team has elected carrier frequencies of 5.65GHz and 5.83GHz[18] to meet these requirements, since Cassegrain antenna efficiency increases with wavelength. The 5.65GHz and 5.83GHz bands are the highest frequencies we can achieve without exceeding the 6GHz limit of our AFE[19].

Higher frequencies also result in a higher gain from the RF-emitting feedhorn. This higher gain results in a narrower beamwidth, ultimately projecting more signal into the effective area of the secondary reflector. Currently, simulation results provided by other team members indicate the biggest limit in the antenna simulation is the feedhorn properties, which currently assumes a conical form-factor and emits at a half power beamwidth of 52°. To be optimal, the beamwidth needs to be 23° to match the sub-reflector geometries. This can be achieved by increasing the flare angle of the conical waveguide, but it cannot exceed the allowable 200 cm^3 volume provided by Flight Vehicle for the inflatable antenna payload.

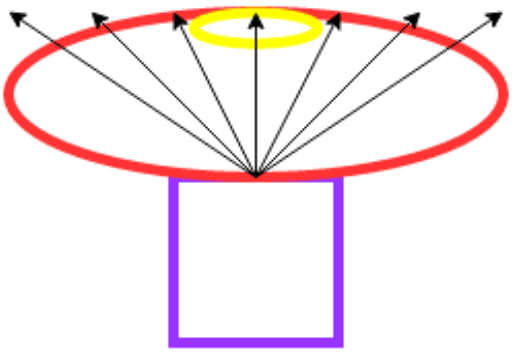


Figure 62: Parabolic reflector power densities at 5.65GHz

Finally, increasing the gain of the antenna too much will reduce the beamwidth sufficiently enough that the Flight Systems (attitude control) team can no longer point with any accuracy, rendering the system completely non-viable. Currently, Flight Systems are working to the pointing requirements of ±5 degrees, meaning the antenna can have a beamwidth no smaller than 10 degrees.

7.5 Results

Currently the antenna gain has been simulated to produce 13 dBi of gain. This was calculated by using an RF simulation of the conical feedhorn [45] by performing a surface integral in MATLAB over the data points that reside inside the effective secondary reflector's beamwidth[20]. Each data point contains the power intensity at a given angle (θ,ϕ), and thus requires a surface integral over the solid angle to obtain an effective input power. It is clear that

[18] See Section 8.
[19] See Section 6.
[20] See Section 7.4.2.5

the majority of losses occur a result of the feedhorn/secondary reflector beamwidth mismatch described in Section 7.4.2.7.

7.6 Future Work

Due to time restrictions and the need to invest dedicated development time into the satellite's data-pass, RF simulations and further design iterations on the inflatable have been passed to a dedicated payload team for further analysis. In order to achieve the extra 2dB of gain in the inflatable antenna, a number of methods can be explored further.

- A secondary IF mixer could be employed after the AFE to mix the carrier to a band of 24-24.05 GHz; the next closest amateur-satellite band. This would dramatically increase the efficiency of the inflatable, and would mean a lower volume antenna design could be achieved to produce the same antenna gain and beam-width. The primary concern of this method is the added power consumption of a secondary mixer to the system, and hence would be a simple trade-off against the additional link gain provided.

- The secondary reflector could be brought closer towards the feedhorn, and even slightly extended in radius to cover more surface area, reducing beamwidth mismatch.

- In the unlikely event that the above two works do not present solutions, the frontal feeder geometry could be reconsidered, since it performed only slightly worse than the Cassegrain in the initial cost analysis, and there is no secondary reflector that causes the SNR reduction.

8 Spectrum Allocation

8.1 Overview

Radio spectrum, known by many as the 'invisible resource', and its associated procedures is a source of major cost and procedural latency for the satellite design team. If left unaddressed until late in the design, this has the potential to create enormous hurdles that cannot be overcome unless the system is completely redesigned.

Applications for wireless networks are ultimately processed by the International Telecommunications Union (ITU), who administer strict policies on obtaining this spectrum through their Radio Regulations (ITU-RR) Volumes I-IV [46]. The policies most relevant to MSP's launch extend from requirements on power spectral density (PSD) to transmission and reception power intensity and are outlined in Article 21 [46].

The services of the ITU are administered by a relevant domestic organization. In the case of the MSP launch, preliminary arrangements have been sought by the Capstone team through the Australian Communications and Media Authority (ACMA). In addition to the expected processing time of 9 months required by the ITU, the ACMA demand an additional month to formalise an application and present it to the ITU.

8.2 Service classification

ITU-RR Vol. I [46] outlines many definitions of system types that must be addressed to determine an appropriate bandwidth assignment. Of those available, MSP's mission can be classified as either a 'Fixed-satellite' or 'Amateur-satellite' primary or secondary service. If allocated as a primary service on a particular band, MSP may police secondary services on the same band to silence other transmissions in the event of harmful interference. This right cannot be exercised when operating as a secondary service.

8.3 Summary of the table of frequency allocations

Researching the ITU-RR volumes in addition to email exchanges with ACMA - Melbourne, ACMA - Canberra, and our experienced industry advisor, Mr Les Davey, has led to the following summary of frequency allocations given in Table 22.

Problems with acquiring spectrum, as outlined in the summary table, are largely attributed to two different cases: coordination and the presence of applications.

Bandwidth (MHz)	Classification	Link Type	Notes
2500-2520	Fixed-satellite	Space to earth	Spectrum Embargo 26
2520-2535	Fixed-satellite	Space to earth	Broadcasting satellite present - ACMA suggest difficulties licensing
2655-2670	Fixed-satellite	Earth-to-space	Requires coordination
2670-2690	Fixed-satellite	Both	Requires coordination
3400-3500	Fixed-satellite	Space-to-earth	C-band commercial operations
3500-3600	Fixed-satellite	Space-to-earth	C-band commercial operations
3600-3700	Fixed-satellite	Space-to-earth	C-band commercial operations
3700-4200	Fixed-satellite	Space-to-earth	C-band commercial operations
4500-4800	Fixed-satellite	Space-to-earth	C-band commercial operations
5150-5250	Fixed-satellite	Earth-to-space	Specific for feeder link to the mobile-satellite service.
5650-5670	Amateur	Space-to-earth	Available for amateur-satellite
5830-5850	Amateur-satellite	Earth-to-space	Available
5850-5925	Fixed-satellite	Earth-to-space	Known spectrum embargo - enquired ACMA
5925-6700	Fixed-satellite	Earth-to-space	Available - less commercially occupied

Table 22: Summary of available bandwidths, classifications, link-type, and possible application complications.

8.3.1 Coordination

Coordination is a measure undertaken by the ITU in the situation that an application for frequency will potentially conflict with an existing application somewhere else in the world. For satellite operations, these conflicts will largely occur over C-band frequencies where telcos and other commercial companies have deployed their own systems. In the event coordination is required, the conflicting party (whomever has applied to the ITU at a later time) is required to pay administration costs to both the ITU administration and the conflicted party until the issue is resolved. This has been quoted to be of the order of thousands of dollars per month by ACMA, and typically is resolved for no less than $30,000. Obviously, given the low-cost nature of this CubeSat design, coordination is completely unfeasible for MSP.

8.3.2 Embargoes

ACMA define a spectrum embargo as 'an administrative tool to facilitate orderly spectrum planning' [47]. As such, if applying for an embargo it is essential that the application is able to outline that the purpose of the network is aligned with the rationale of the planned spectrum embargo. This process, however, has been advised against by ACMA since it adds further procedural complexity when an alternative frequency could be more easily acquired.

8.3.3 Selection of frequencies

In light of the above information, and the positive responses from ACMA, Melbourne and Mr Les Davey, an up-link and down-link frequency of 5650-5655 MHz and 5830-5835 MHz is to be currently being sought by the Capstone team and MSP.

8.4 Amateur radio compliance

The decision to pursue an amateur radio license is a conflicting one. While it is the only choice for a relatively inexpensive license, the radio design employed by the Capstone team must be made open-source and cannot be used to generate profit for MSP or is stake-holders.

This has ongoing implications for the satellite after launch. A requirement of amateur radio is that the link is made available to all HAM radio operators world-wide. This means that data transmitted to and from the satellite must remain unencrypted and available to any interested licensed party. In order for this to occur, all modules of the data pass in Section 5 must be published globally.

9 Project Management and Satellite Integration

This project is being conducted in conjunction with other Capstone groups, under the guidance of the Melbourne Space Program (MSP), which has the stated goal of launching a satellite at the end of 2017. Multiple teams are working to develop their own individual systems which must fit together, and there are students in the telecommunications team who are not part of the Capstone group. That means that some work crucial to designing the telecommunication system wasn't performed by ELEN90070 students, and their contributions are referenced appropriately. Because of this, significant effort must be expended by group members in co-ordination with other teams and students.

9.1 Team Responsibilities

The telecommunications team is made up of undergraduate electrical and chemical engineering students and Master of Electrical Engineering students, in addition to the Capstone members. These other members have been entrusted with important design tasks, such as determining the transmission windows, determining the required amount of benzoic acid required to keep the inflatable antenna at an appropriate pressure for the period of the mission, verifying literature reviews for antenna calculations and researching methods to seal the inflated balloon to prevent outgassing. While the results of their work cannot be considered part of this report, time was required to determine the tasks to assign to these members and confirm the correctness of their work.

In addition, team members will move on as their courses complete, so a handover process has been put in place to ensure that when senior members leave the program, junior members are ready to seamlessly continue progress. That has been a driving goal of the Capstone, to generate enough documentation that subsequent teams don't have to reinvent the wheel.

9.2 Inter-team communication

For the satellite to function, each subsystem must be able to communicate with other subsystems: the Power team must supply the correct voltage rails to the Communications team; the Communications board must respond to commands from the Operating System, and the communications board must fit within the envelope determined by the Flight Vehicle team. As a more tactile example, the connection pins between the boards must line up. This requires constant communication with other teams, otherwise the communications system exists in a vacuum as a standalone education example.

A few examples of this are: presenting the telecommunications system progress at a number of design reviews, answering questions about power requirements and requesting physical space within the satellite. There are weekly and ad-hoc meetings to talk about how the different subsystems need to work together.

10 Testing and Results

10.1 Simulation

In both the 625kHz and 5MHz bandwidth profiles, the BER for a 9dB bit SNR is approximately 3.1×10^{-5} after a 6dB SNR margin is reserved for non-ideal sychronization, quantization, RF model and ect. These results agree with the analysis in link budget. Furthermore, peak-to-average power ratio terms of 12dB were occasionally observed, validating the analysis in Section 2.2.2.

10.2 Hardware Verification

Once the system was fully functional, 18 test runs were conducted in the DnB lab following the process described in Figure 11 on page 19. Since this is not an RF shielded environment, the receiving antenna often picked up noise with a strong central peak centred around 2.4GHz. This was initial thought to be from WiFi routers, however this protocol transmits over a 20MHz spectrum. The magnitude of this noise was time variant, and resulting in different reception spectra as seen in Figures 63 and 64.

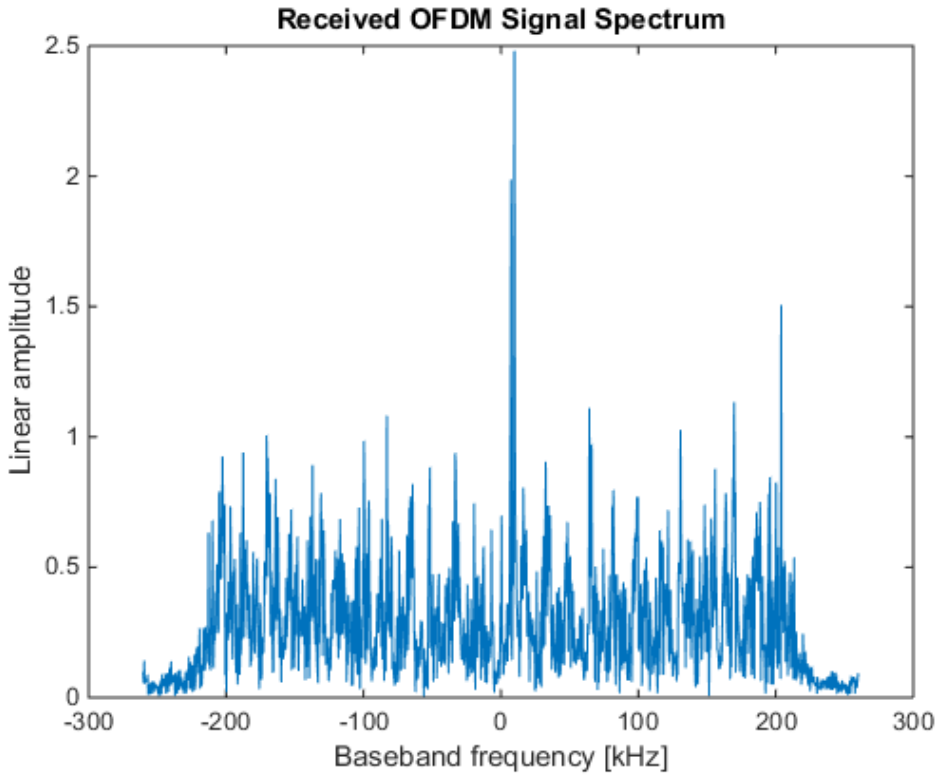


Figure 63: Received Spectrum with significant interference

For the ease of use, the DSP process was simulated on the VDSP++ platform. A test message was generated on a

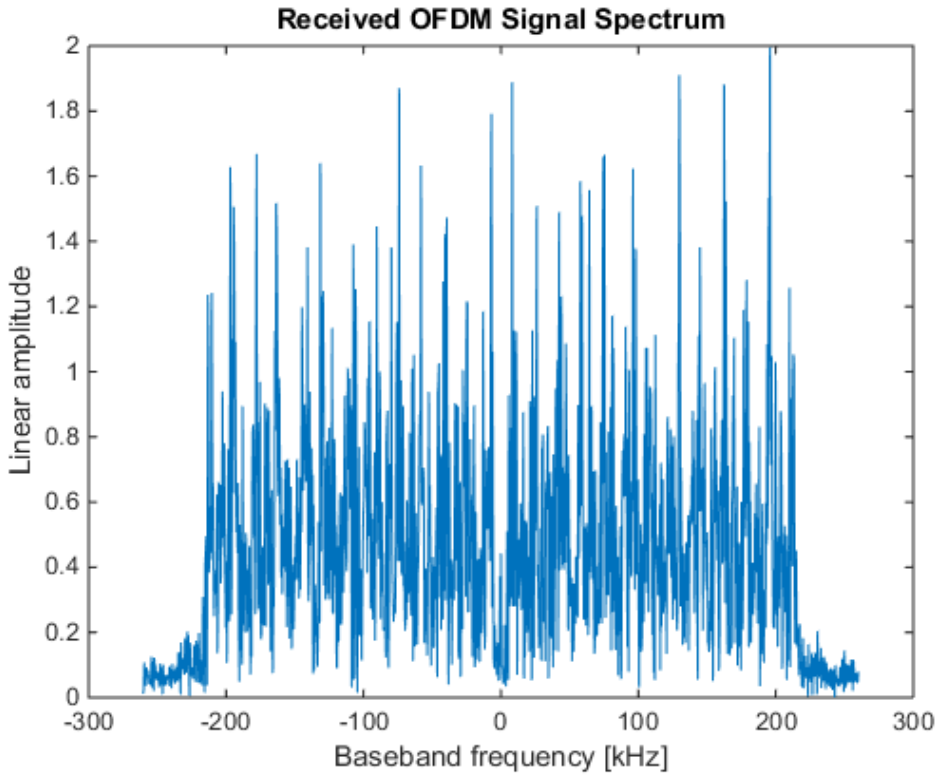


Figure 64: Received Spectrum without interference

laptop, which was then interleaved and encoded. This was dumped from the DSP's memory to a .dat file. This was mapped on the DSP simulator, and these complex time domain samples were transferred to a MATLAB environment to be transmitted via the SDR. A receiver script collected complex time domain samples, and ran MATLAB code to perform synchronisation and automatic gain control (AGC). These complex time domain samples were passed back to the VDSP++ environment where they were decoded, and a binary string was generated.

All the binary files generated during every trial of this process were transferred to a MATLAB environment which extracted the binary strings to print the received character strings, and determine input and output bit error rates (BER). Figure 65 was achieved via executing this script, shown in Listing 24 in Section C.2.

Determining the effect of the interference is more complicated than just assuming that the demapper will get its estimation of the affected tone wrong 50% of the time. A strong interference peak will increase the power of the received signal, so the intended tones peaks won't be magnified as much. Reducing the power of each tone as it passes into the demapper increases the probability of incorrect demapping, and hence increases the channel BER. Modelling this effect in a MATLAB simulation is difficult: so the simulated curve in Figure 65 just inverts particular bits in the string depending on whether a uniformly distributed pseudo-random number is less than a desired bit error rate.

As expected, when the decoder BER was low, the Viterbi decoder was able to correct for nearly all errors. However,

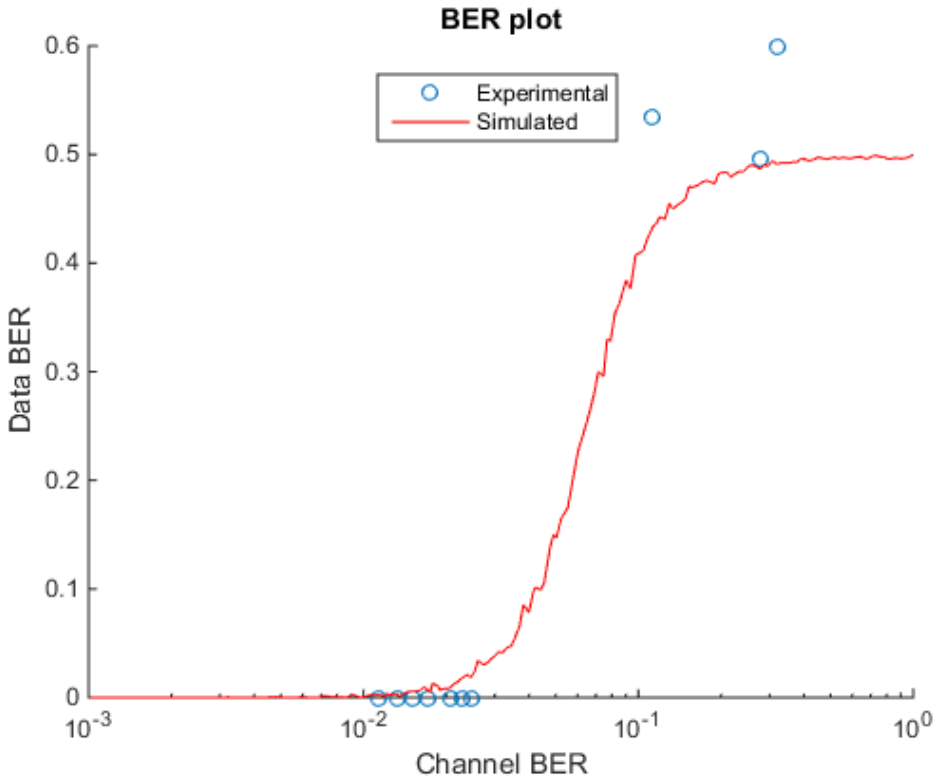


Figure 65: Experimental and Simulated BER curves

at higher input BERs, the bursty output nature of the Viterbi decoder came to the fore: once the errors came close enough to each other the decoder output became nonsense. A BER of 0.5 is the worst possible output, since there is no preference for the bits being correct or incorrect. A 'knee' was found where the input BER equalled the output BER, for the $k = 4$ decoder this lay at an input BER of $\approx$2%. Beyond this, the output deteriorated.

Trials were conducted with the transmitter and receiver antennas spatially separated by distances of 100mm to 3m in order to give different channel BERs. Given the small bit length of each transmitted frame (11 hexadecimal symbols), it is not surprising that trials superimpose upon one another. It would have been desirable to get some channel error rates between 0.03 and 0.1 to lie along in the transition region, however this proved difficult: interference was either too low and most channel errors were corrected, or the magnitude of the peak was too high, causing the desired spectrum to be attenuated, in addition to a central tone being knocked out, resulting in a data BER of around 0.5.

As can be seen, the experimental results loosely match with the simulated. Running each experimental trial took a significant amount of time, and time constraints prevented more tests from being run to average out outliers in the data. Of particular note is the fact that frames with channel BERs up to 0.0246 were decoded without error, whereas the simulation predicted error rates of 2%. This can be ascribed to chance: the transmitted frames only held 11 hexadecimal symbols totalling 264 bits, meaning that experimental channel bit error rates between zero and 0.00378 were not achievable. The simulator tested 1000 frames per error rate, and so the occasional bursty

error output would drag the simulated data BER up. In single frame tests, whether the bursty output case is seen is deterministic. With that in mind, it can be concluded that the constructed system operates as expected.

Section 5.2.4 describes the effect of including a hard decision decoder on the SNR. As seen in Figure 28 on page 41, a coding gain of approximately 5dB is attained, significantly larger than the conservative margin used to generate the link budget. The coding gain can be found by taking the horizontal distance between the SNR curves.

11 Conclusion

As a sub-team in with the MSP, we were required to design a flexible, adaptable and accessible communication system for the satellite. We successfully navigated major steps in communication system design including a feasibility study, requirement collection, power and link budget evaluation, specification, prototype simulation, hardware selection, high level architecture design, low level module design, verification, system integration and testing.

OFDM is an important modulation scheme in contemporary communication systems and it is widely adopted by 4G, WiFi and DVB standards. In order to deliver a flexible and standardized system, we decided to partially follow the IEEE 802.11a standard rather than create a new communication protocol. The benefits derived from this decision abound: the communication protocol has been solidified by world leading organizations and companies; enormous research in this area which could potentially help us to overcome unpredictable issue has already been conducted; very rich MATLAB resources are available in the prototype and verification stage; there are a range of hardware choices to implement the OFDM system and many of them have been optimized in this implementation.

Some of the hardest tasks have included: understanding the depths of the IEEE and ITU standards. The devil lay in the detail, and even small sections could cause our base-band processor to produce nonsensical output. This was also true with the MATLAB implementation of the AFE driving circuit. Also of great difficulty was optimising C code within the DSP environment, and ensuring correct data flow occurred between blocks. A particularly challenging aspect of this project has been deciding where to spend our time. Such a broad Capstone scope allowed us to devote time where the greatest improvements could be made. This freedom has allowed us to learn more by weighing up the trade-offs of a particular choice to a greater extent than other Capstone projects, which more closely adhered to a tight specifications set. This has been more satisfying, and more closely mimics commercial practise.

The analysis conducted to date indicates the feasibility of the system in meeting the program specification. Further work needs to be done by further FYP groups or MSP team members in the following areas:

- Select high power amplifier
- Investigate selection of a microwave transmission band
- Set up the online evaluation stage
- Design dipoles and the patch antenna to sit inside the parabolic reflector
- Include buffers to store data dumps from the OS team
- Implement Reed-Solomon encoding if the cycle budget is amenable

- Design and construct prototype PCB's for the telecommunications system
- Design the hardware to drive the ground station transmission, potentially based off the evaluation platforms used for this Capstone. This should interface with the RF dish orientation Capstone project from 2015 [1]

In general, we feel that this project has been a terrific learning experience, and look forward to continual work with MSP in fulfilling the system design.

References

[1] R. Mearns, "MSP CubeSat with Parabolic Antenna." May 2016.

[2] "Phy basics: How ofdm subcarriers work." http://www.revolutionwifi.net/revolutionwifi/2015/3/how-ofdm-subcarriers-work. Accessed: 17-5-2016.

[3] M. C. P. Paredes and M. García, "The problem of peak-to-average power ratio in ofdm systems," *arXiv preprint arXiv:1503.08271*, 2015.

[4] S. Hermelin, "Radio wave propagation over the earth." http://www.slideshare.net/solohermelin/4-radio-wave-propagation-over-the-earth, 2010. Accessed: 15-5-2016.

[5] Analog Devices, *AD9364 RF Agile Transceiver, Rev. C*, 2014.

[6] Analog Devices, *ADSP-TS201S TigerSHARC® Processor Programming Reference, Rev. 1.1*, 2005.

[7] IEEE Standard Association, *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, 2012.

[8] "A mind forever programming." http://amindforeverprogramming.blogspot.com.au/2013/07/random-floats-in-glsl-330.html. Accessed: 16-5-2016.

[9] B. Lerner, "Writing efficient floating-point ffts for adsp-ts201 tigersharc processor," *2004-05-06]. http://www. analog, com/dsp*, 2004.

[10] "Communication - ofdm." http://www.sharetechnote.com/html/Communication_OFDM.html. Accessed: 16-5-2016.

[11] "Communication - ofdm." http://www.sharetechnote.com/html/Communication_OFDM.html. Accessed: 16-5-2016.

[12] "AD9361 evaluation board image." https://www.arrow.com/en/research-and-events/articles/the-ad9361-sdr-for-today-and-tomorrow. Accessed: 16-5-2016.

[13] "Zedboard image." http://www.nutaq.com/blog/zeptosdr-architecture-and-api. Accessed: 16-5-2016.

[14] "Zynq-7000 system architecture." https://commons.wikimedia.org/wiki/File:Xilinx_Zynq-7000_AP_SoC.jpg. Accessed: 16-5-2016.

[15] M. Garg *et al.*, "Cubesat final report," 2003. Accessed: 18-5-2016.

[16] P. B. Henderson, *High Data Rate Radio Transmitter for Cube Satellites*. PhD thesis, Citeseer, 2009.

[17] A. Babuscia, M. Van de Loo, Q. J. Wei, S. Pan, S. Mohan, and S. Seager, "Inflatable antenna for cubesat: fabrication, deployment and results of experimental tests," in *Aerospace Conference, 2014 IEEE*, pp. 1–12, IEEE, 2014.

[18] "Nasa nodes." http://www.nasa.gov/feature/nasa-small-satellite-duo-deploys-from-space-station-into-earth-orbit. Accessed:14-6-2016.

[19] "Gomspace ax100 module." http://gomspace.com/index.php?p=products-ax100. Accessed:14-6-2016.

[20] Analog Devices, *Estimating Power for ADSP-TS201S TigerSHARC Processors*, 2006.

[21] MWT Microwave Technolofy, *MMA-445933H-02 2W High Efficiency Linear Power Amplifier*, 2012.

[22] TriQuint Semiconductor, *TGA2706-SM 2 Watt C-Band Packaged Power Amplifier, Rev. G*, 2015.

[23] A. Singh and H. Kaur, "Non linearity analysis of high power amplifier in ofdm system," *International Journal of Computer Applications*, vol. 37, no. 2, pp. 37–41, 2012.

[24] L. Davey, "Conversations with Les Davey." 2016.

[25] D. Roddy, *Satellite Communications*, ch. The Space Link. McGraw-Hill, 2006.

[26] C. J. R. Capela *et al.*, "Protocol of communications for vorsat satellite," Apr 2012.

[27] P. L. Rice, A. G. Longley, K. Norton, and A. Barsis, "Transmission loss predictions for tropospheric communication circuits, volume 1," tech. rep., DTIC Document, 1967.

[28] S. Cakaj, "Antenna noise temperature for low earth orbiting satellite ground stations at L and S band," 2011.

[29] M. I. Abdullah, "Comparative study of papr reduction techniques in ofdm," 2011.

[30] A. J. Coulson, "Bit error rate performance of bpsk modulated ofdm synchronized using a pilot symbol," in *Personal, Indoor and Mobile Radio Communications, 2001 12th IEEE International Symposium on*, vol. 2, pp. F–86, IEEE, 2001.

[31] "X-band transmitter can transmit up to 13.3 gb per pass with a 5 m station, designed for cubesat and nanosatellites (leo)." http://www.syrlinks.com/en/products/cubesats/hdr-x-band-transmitter.html. Accessed: 17-5-2016.

[32] B. team, "Cubesat design overview report," Accessed: 18-5-2016.

[33] P. Y. Cheung, G. A. Constantinides, and J. T. de Sousa, *Field-Programmable Logic and Applications: 13th International Conference, FPL 2003, Lisbon, Portugal, September 1-3, 2003, Proceedings*, vol. 2778. Springer Science & Business Media, 2003.

[34] A. Devices, "Adsp-ts201s tigersharc embedded processor datasheet," 2006.

[35] Analog Devices, *C/C++ Compiler and Library Manual for TigerSHARC® Processors, Rev. 4.1*, 2008.

[36] Y. Porat and A. Reichman, "Burst error characteristics of viterbi decoding of convolutional codes," in *Electrical and Electronics Engineers in Israel, 1991. Proceedings., 17th Convention of*, pp. 230–233, IEEE, 1991.

[37] R. G. Lyons, *Understanding digital signal processing*. Pearson Education, 2010.

[38] S. R. Mittal and V. Mittal, "A cyclic prefix ofdm system with bpsk," 2012.

[39] A. Fort and W. Eberle, "Synchronization and agc proposal for ieee 802.11 a burst ofdm systems," in *Global Telecommunications Conference, 2003. GLOBECOM'03. IEEE*, vol. 3, pp. 1335–1338, IEEE, 2003.

[40] T. M. Schmidl and D. C. Cox, "Robust frequency and timing synchronization for ofdm," *Communications, IEEE Transactions on*, vol. 45, no. 12, pp. 1613–1621, 1997.

[41] M.-H. Hsieh and C.-H. Wei, "Channel estimation for ofdm systems based on comb-type pilot arrangement in frequency selective fading channels," *Consumer Electronics, IEEE Transactions on*, vol. 44, no. 1, pp. 217–225, 1998.

[42] I. Analog Devices, "Adsp-ts201 tigersharc® processor hardware reference," 2004.

[43] A. A. Saad, "Orbital drag calculations for 400km orbit with inflatable antenna." 2016.

[44] T. A. Milligan, *Modern antenna design*. John Wiley & Sons, 2005.

[45] H. Mulyono, "RF Simulations." May 2016.

[46] "Radio regulations articles." http://www.itu.int/en/history/Pages/RadioRegulationsA.aspx?reg=41be=S0202000024, 2012. Accessed: 17-05-2016.

[47] "Spectrum embargoes." http://www.acma.gov.au/Industry/Spectrum/Spectrum-planning/Current-APs-info-and-resources/spectrum-embargoes-spectrum-planning-acma. Accessed: 17-5-2016.

A Supplementary Figures

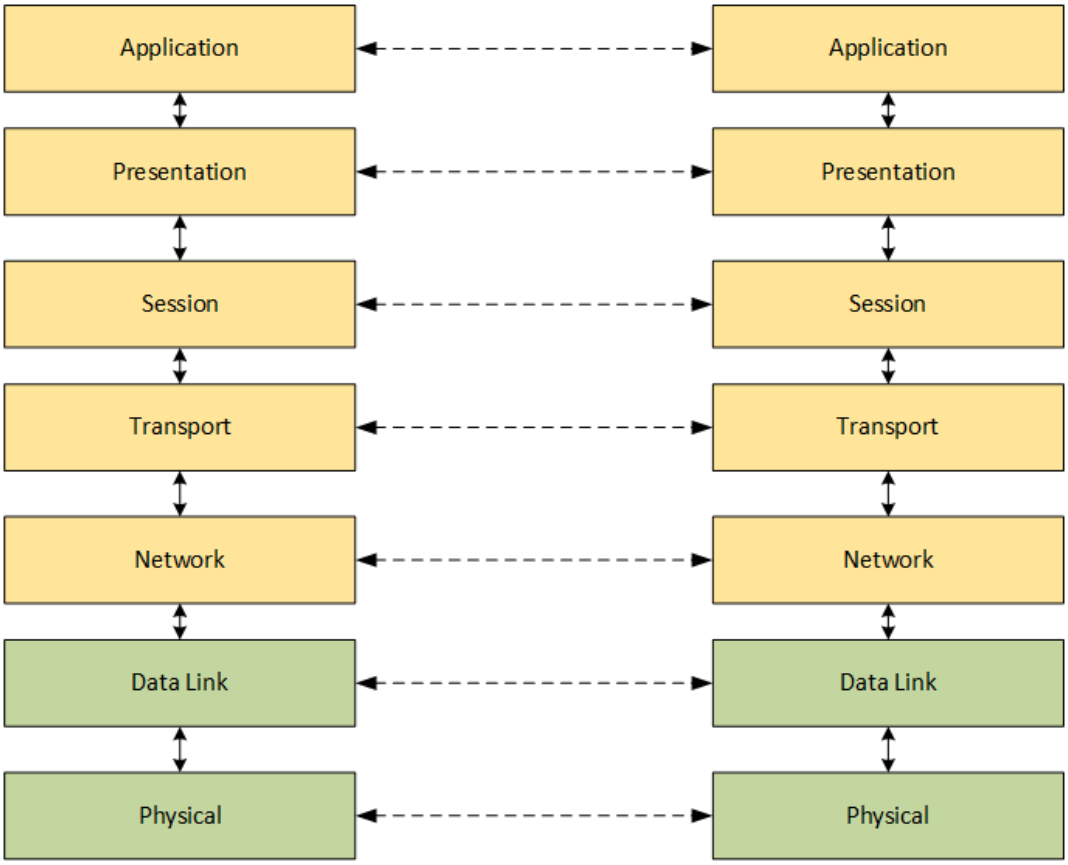


Figure 66: The OSI model for abstracting telecommunication systems. The green layers will be implemented on the CubeSat, but the orange layers are not required.

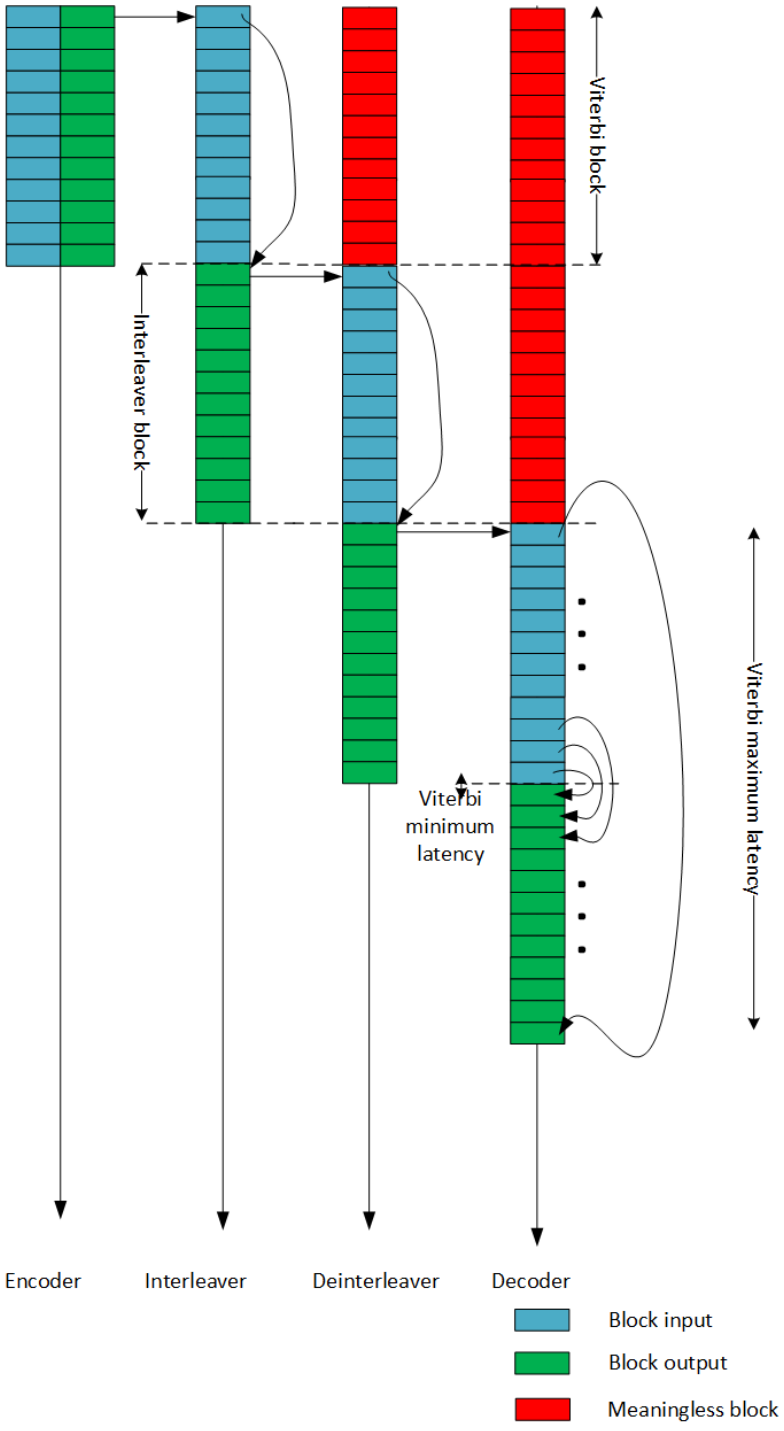


Figure 67: A latency description of the software data-pass.

B DSP Code

```
/*
The transmitter function includes
*/

/*
CHANNEL MODEL will be inserted between the TX and RX. The current channel model is a combination of AWGN
and multi-path noise.
*/

/*
The receiver functions include two seperate modes - Synchronization and Data pass - for which we have
a maximum of 8000 clock cycles to use for each pass.

  - Synchronisation (using the Frame preamble).
    - Receive from the LVDS interface (currently a dummy function, and instead is received from file)
    - Frame boundary detection (time domain), currently performed in MATLAB.
    - Frequency offset equalisation (frequency domain), currently performed in MATLAB
    - FFT
    - Frequency equalisation (frequency domain)
      - Phase and gain estimation, performed in MATLAB.
      - Phase compensation, performed by the DSP.
  - Data pass (using the Frame payload)
    - Receive from the LVDS interface (currently a dummy function, and instead is received from file)
    - FFT
    - Frequency equalisation
      - Stage 1 gain compensation and phase compensation using preamble estimates.
      - Gain and phase estimation using linear interpolation on the pilot subcarriers, performed in MATLAB.
      - Stage 2 gain and phase compensation using the pilot subcarrier estimates.
    - Demapping
    - Deinterleaving
    - Viterbi decoding
    - Giving the output to the OS.
*/
//************************ Includes ***********************************
#include <stdio.h>
#include <sysreg.h>
#include <builtins.h>
#include "FFTDef.h"
#include "mapper.h"
```

```
#include "convolutional.h"
#include "testbench.h"
#include "memory.h"
#include "interleaver_new.h"
#include "cyclic_prefix.h"
#include "lvds.h"
#include "freqEqualisation.h"
#include "input.h"


//************************ Defines ***********************************
#define LATENCY (2*INTV_WORDS+CONV_SIZE)
#define TOT_DATA_PASS (LATENCY + TOT_SYMBOL_NUM)
#define TOT_NUM_PASSES (TOT_DATA_PASS*4/3) // Includes preamble addition.

//************************ Externs ***********************************
extern IFFT32( float (*)[], float (*)[], float (*)[], float (*)[], int, int );
extern FFT32( float (*)[], float (*)[], float (*)[], float (*)[], int, int );

//************************ Global Variables ****************************
extern init();

#pragma align 4
section ("data1ab")
float ifft_output[outbuff_size];

#pragma align 4
section ("data1ab")
float ifft_output_cp[NUM_CP];

#pragma align 4
section ("data1ab")
float fft_input[NUM_CP];

#pragma align 4
section ("data1ab")
float fft_output[outbuff_size];

float input[2*N] = {
                    #include "inputs/ifft_input128.dat"
                    };

unsigned int demap_output[2];

#pragma align 4
float map_out[MAP_SIZEOUT];

#pragma align 4
```

```
section ("data2ab")
  float ping_pong_buffer1[outbuff_size];

#pragma align 4
section ("data3ab")
  float ping_pong_buffer2[outbuff_size];


volatile int
  tmp_i0,
  tmp_i1;

section ("data10a")
unsigned int INPUT[TOT_SYMBOL_NUM]=
  #include "4800words24bit.txt"

section ("data10a")
unsigned long int OUTPUT[TOT_SYMBOL_NUM];

//********************************* main ****************************************
void main( void ){

  float *RECEIVED;
  unsigned int conv_input;
  unsigned long long int enc_output, intv_output, deintv_output;

  float RECEIVED_SYM[IFFT_OUTPUT_SIZE + NUM_CP];
  float TRANSMITTED_SYM[IFFT_OUTPUT_SIZE + NUM_CP];
  float recLongPreamble1[SAMPLE_SIZE];
  float recLongPreamble2[SAMPLE_SIZE];
  //float gains[PAYLOAD_SYMBOLS][2*BITS_IN_SYMBOL] =
    //#include "feqGains.dat"

  float eqData[2*BITS_IN_SYMBOL];
  //float feqInput[TOT_SYMBOL_NUM][SAMPLE_SIZE] =
    //#include "feqInput.dat"

  unsigned long long int demap_output;
  unsigned int vit_output;
  unsigned long long int burst;
  float BER;
  int i,j;
  int data_numrx = 0, tXdata_num = 0;

  convolutionalInit();
  //printf("{");
  i = 0;
  while (i <(TOT_NUM_PASSES)){
```

```
    printf("SYMBOL %d\n", i);
137 // TRANSMITTER CODE
    switch (i%SYMBOLS_IN_FRAME){
139   case 0:
        // Short preamble part 1
141     preamble_transmit(TRANSMITTED_SYM,0);
        break;
143   case 1:
        // Short preamble part 2
145     preamble_transmit(TRANSMITTED_SYM,1);
        break;
147   case 2:
        // Long preamable part 1
149     preamble_transmit(TRANSMITTED_SYM,2);
        break;
151   case 3:
        // Long preamble part 2
153     preamble_transmit(TRANSMITTED_SYM,3);
        break;
155   default:
        //printf("%d\n",i);
157     // Read from memory
        tmp_i0 = __builtin_sysreg_read( __CCNT0 );          // read initial cycle count
159     conv_input = memRead(INPUT);
        tmp_i1 = __builtin_sysreg_read( __CCNT0 );          // read initial cycle count
161     printf("memRead count: %d\n",tmp_i1-tmp_i0);
        //printf("%d: Input = 0x%06x\n",i,conv_input);
163
        tmp_i0 = __builtin_sysreg_read( __CCNT0 );          // read initial cycle count
165     enc_output = convEncoder(conv_input);
        tmp_i1 = __builtin_sysreg_read( __CCNT0 );          // read initial cycle count
167     printf("convEncoder count: %d\n",tmp_i1-tmp_i0);
        //printf("Enc output = 0x%012llx\n",enc_output);
169
        // Interleaver - internal buffers.
171     tmp_i0 = __builtin_sysreg_read( __CCNT0 );
        intv_output = interleaver(enc_output);
173     tmp_i1 = __builtin_sysreg_read( __CCNT0 );          // read initial cycle count
        printf("interleaver count: %d\n",tmp_i1-tmp_i0);
175     //printf("interleaver output: 0x%012llx\n",intv_output);

177     tmp_i0 = __builtin_sysreg_read( __CCNT0 );          // read initial cycle count
        mapper(intv_output, map_out);
179     tmp_i1 = __builtin_sysreg_read( __CCNT0 );          // read initial cycle count
        printf("mapper count: %d\n",tmp_i1-tmp_i0);
181
        // OFDM IFFT
183     tmp_i0 = __builtin_sysreg_read( __CCNT0 );          // read initial cycle count
```

```
        IFFT32(&(map_out), &(ping_pong_buffer1), &(ping_pong_buffer2), &(ifft_output), N, COMPLEX)
    ; // empty the first 16 complex number for CP
185     tmp_i1 = __builtin_sysreg_read( __CCNT0 );          // read initial cycle count
        printf("FFT count: %d\n",tmp_i1-tmp_i0);
187
        // Insert the cyclic prefix
189     tmp_i0 = __builtin_sysreg_read( __CCNT0 );          // read initial cycle count
        cp_insert((ifft_output), (ifft_output_cp));
191     tmp_i1 = __builtin_sysreg_read( __CCNT0 );          // read initial cycle count
        printf("CP insert count: %d\n",tmp_i1-tmp_i0);
193
        // Transmit the data
195     data_transmit(ifft_output_cp,ifft_output,TRANSMITTED_SYM);
        tXdata_num++;
197 }


199
    // Print IFFT to file to be given to AFE set-up
201 /*for (j=0; j<SAMPLE_SIZE; j+=2){
      printf("RE: %f IM: %f\n", TRANSMITTED_SYM[j],TRANSMITTED_SYM[j+1]);
203 }
    printf("\n");*/
205
    // RECEIVER CODE
207 switch (i%SYMBOLS_IN_FRAME){
      case 0:
209     // Short preamble sequence part 1.

211     getLVDS(RECEIVED_SYM, TRANSMITTED_SYM);
        break;
213   case 1:
        // Short preamble sequence part 2.
215     getLVDS(RECEIVED_SYM, TRANSMITTED_SYM);
        break;
217   case 2:
        // Long preamble sequence part 1.
219     getLVDS(recLongPreamble1, TRANSMITTED_SYM);
        break;
221   case 3:
        // Long preamble sequence part 2.
223     getLVDS(recLongPreamble2, TRANSMITTED_SYM);

225     break;
      default:
227     // Payload (data) symbol
        getLVDS(RECEIVED_SYM, TRANSMITTED_SYM);
229
        tmp_i0 = __builtin_sysreg_read( __CCNT0 );          // read initial cycle count
```

```
231         cp_remove(TRANSMITTED_SYM, fft_input);
            tmp_i1 = __builtin_sysreg_read( __CCNT0 );          // read initial cycle count
233         printf("CP remove count: %d\n",tmp_i1-tmp_i0);


235
            // OFDM FFT
237         tmp_i0 = __builtin_sysreg_read( __CCNT0 );          // read initial cycle count
            FFT32(&(fft_input), &(ping_pong_buffer1), &(ping_pong_buffer2), &(fft_output), N, COMPLEX)
        ;
239         tmp_i1 = __builtin_sysreg_read( __CCNT0 );          // read initial cycle count
            printf("IFFT count: %d\n",tmp_i1-tmp_i0);
241
            // FEQ
243         //applyGains(gains[i-PREAMBLE_SYMBOLS],fft_output,eqData);

245         //for (j=0; j<SAMPLE_SIZE; j+=2){
            //   printf("RE: %f IM: %f\n", eqData[j],eqData[j+1]);
247         //}
            //printf("\n");
249
            // Demapper
251         tmp_i0 = __builtin_sysreg_read( __CCNT0 );
            demapper(fft_output,&demap_output);
253         tmp_i1 = __builtin_sysreg_read( __CCNT0 );          // read initial cycle count
            printf("demapper count: %d\n",tmp_i1-tmp_i0);
255         //printf("DEMAP OUTPUT: 0x%012llx\n",demap_output);
            // Deinterleaver
257         tmp_i0 = __builtin_sysreg_read( __CCNT0 );
            deintv_output = deinterleaver(demap_output);
259         tmp_i1 = __builtin_sysreg_read( __CCNT0 );          // read initial cycle count
            printf("deinterleaver count: %d\n",tmp_i1-tmp_i0);
261         //printf("deinterleaver output: 0x%012llx\n", deintv_output);

263         // Viterbi decoder
            tmp_i0 = __builtin_sysreg_read( __CCNT0 );
265         vit_output = convDecoder(deintv_output);
            tmp_i1 = __builtin_sysreg_read( __CCNT0 );          // read initial cycle count
267         printf("convDecoder count: %d\n",tmp_i1-tmp_i0);
            //printf("Viterbi output = 0x%06x\n",vit_output);
269
            // Write to memory
271         tmp_i0 = __builtin_sysreg_read( __CCNT0 );
            memWrite(vit_output,OUTPUT);
273         tmp_i1 = __builtin_sysreg_read( __CCNT0 );          // read initial cycle count
            printf("memWrite count: %d\n",tmp_i1-tmp_i0);
275         data_numrx++;
            //printf("data_numrx = %d\n",data_numrx);
277
```

```
        }
279     //printf("i = %d\n",i);
        i++;
281   }
      printf("data_num = %d\n",tXdata_num);
283   printf("i = %d\n",i);
      //printArrayInt(OUTPUT,TOT_SYMBOL_NUM);
285
      //BER = ber_calc(INPUT, OUTPUT);
287   //printf("BER = %f\n", BER);
  }
```

Listing 1: dspCode/main.c

```
 #define TOT_SYMBOL_NUM 4800
2#define NUM_FLUSH (TOT_SYMBOL_NUM/EFF_CONV_SIZE)
 #define BITS_IN_DOUBLE 64
4#define BITS_IN_SYMBOL 48
 #define N_SYMBITS 48
6#define BITS_IN_INT 32
 #define BITS_IN_SHORT 16
8#define BITS_IN_BYTE 8
 #define BIT_LEN 1
10#define OFDM_BITS 64
 #define NUM_CP 32 // each imag and real is float
12#define IFFT_OUTPUT_SIZE 2*OFDM_BITS
 #define SYMBOLS_IN_FRAME 16
14#define PREAMBLE_SYMBOLS 4
 #define PAYLOAD_SYMBOLS 12
16#define SAMPLE_SIZE (IFFT_OUTPUT_SIZE+NUM_CP) // the size of each ifft_out, consisting of 64 tones
      + 16 CP. each time we transmit 160 floats
```

Listing 2: dspCode/input.h

```
 // Included libraries
2#include "defts201.h"
 #include "builtins.h"
4
 // Definitions
6#define PRINT_LINE_LEN 20
 //#define CONV_SIZE 32
8#define CONV_SIZE 12
 #define EFF_CONV_SIZE (CONV_SIZE-1) // Don't count the inserted flush bits.
10#define BITS_IN_INPUT 24
 #define OUTPUT_SIZE (RATE*BITS_IN_INPUT)
12#define TOT_INPUT_BITS (CONV_SIZE*BITS_IN_INPUT)
```

```
#define RATE 2
#define TOT_OUTPUT_BITS (RATE*TOT_INPUT_BITS)
#define TRANS_LEN 138
#define SEED 34642
#define BITS_IN_DOUBLE 64
#define BITS_IN_INT 32
#define BITS_IN_SHORT 16
#define BITS_IN_BYTE 8
#define BIT_LEN 1
#define NUM_STATES 8
#define NUM_BUTTERFLIES 4
#define THR_BITS_PER_BUTT 8
#define RX_MASK 0x3;
#define LAST_STAGE (CONV_SIZE-1)
#define INIT_STAGE 0
#define INIT_STATE 0

// Type definitions
//typedef unsigned char byte;
typedef struct State {
    int prev[2];
    unsigned int recBit;
    int tbPos;
} State;

// Global variables
State stateVect[NUM_STATES];

long long int VIT_METRIC0;
long long int VIT_METRIC1;
int MAX_STATE;

// Function Prototypes

void convolutionalInit(void);
unsigned long long int convEncoder(unsigned int);
//int convDecoder(unsigned long long int,
//        unsigned int [],
//        unsigned int [],
//        int *state_index);
int convDecoder(unsigned long long int);
unsigned int viterbiTraceBack(unsigned int THR[TOT_INPUT_BITS], State stateVect[NUM_STATES], int *
    state_index);
int calcMaxState(long long int VIT_METRIC0,long long int VIT_METRIC1);
```

Listing 3: dspCode/convolutional.h

```
1
 #include "convolutional.h"
3#include <stdio.h>

5unsigned long long convEncoder(unsigned int input){
   // Code for generator polynomials 0 - {1,0,0,1} and 1 - {1,1,1,1} {s3 s2 s1 u}|
7  // Encoder outputs MSB to the RHS of the output word. This is the first Tx'ed bit
   // and is according to the Little Endian convention, to fit the TS201 architecture.
9  // Encoder will take 744+24 input bits to produce 1488+48 output bits
   // The excess 16 bits are non-data and serve to return the encoder to its
11 // initial conditions to prevent any enormous bit errors propagating through
   // the remaining trellis structure in the decoder.
13 //
   //  CONVOLUTIONAL CODER TOPOLOGY
15 //    --|+|------|+|------|+|------|+|------>> 1
   //    |   |    |      |      |
17 //    |   | ____ |  ____ |  ____ |
   //  u---->--| S3 |---| S2 |---| S1 |--
19 //    |   | ____ |  ____ |  ____ |
   //    |   |    |      |      |
21 //    -----|+|--------------------|+|------>> 0
   //

23
   int i;
25 unsigned long int del1, del2, del3;
   unsigned long int output[RATE];
27 unsigned long long int combined_output;
   static long int state=0;
29
   // Setup encoder state
31 state = state^(input << 3);
 ;
33 // Setup additive encoder terms
   del1 = state >> 1;
35 del2 = del1 >> 1;
   del3 = del2 >> 1;
37
   // See document explaining this shift-add
39 output[0] = state^del3;
   output[1] = output[0]^del2^del1;
41
   // Mask out the valid range of the addition.
43 output[0] = output[0]&(0xffffff);
   output[1] = output[1]&(0xffffff);
45
   asm(
47 "XR2 = %1;;"
```

```
    "XR3 = %2;;"
49  "THR1:0 = R3:2 (I);;"
    "%0 = THR1:0;;"
51  : "=xl" (combined_output)
    : "x" (output[0]), "x" (output[1])
53  : "XR2", "XR3");
    // Update the encoder state
55  state = state >> BITS_IN_INPUT;
    return combined_output;
57

  }
59

  int convDecoder(unsigned long long int input)
61{
    long branchMetricTable[NUM_BUTTERFLIES] = {0xff0000ff,0x00010100,0x00ffff00,0x01000001};//for
      00, 01, 10, 11
63  int i, j;
    int rx_bits;
65  __builtin_quad res;
    int vit_output;
67  unsigned long long int vit_input = input;
    unsigned long long int tb_hist;
69  static int tb_out_offset = 0;
    static int stage=0;
71  int decoded_bit;
    unsigned int *temp;
73  static unsigned int tb_mem_ping[TOT_INPUT_BITS];
    static unsigned int tb_mem_pong[TOT_INPUT_BITS];
75  static int *state;
    static unsigned int *TB_PING = tb_mem_ping, *TB_PONG = tb_mem_pong;
77  // Initialise the path metric.
    if(stage==INIT_STAGE){
79    VIT_METRIC0 = 0x0000000000000000;
      VIT_METRIC1 = 0x0000000000000000;
81    temp = TB_PING;
      TB_PING = TB_PONG;
83    TB_PONG = temp;

85    // Reset the state of viterbi to 0 since the data flush is known to have occured.
      (*state) = INIT_STATE;
87

    }
89  // Fill up the Trellis History Registers for calculation during the traceback
    for(i=0;i<BITS_IN_INPUT;i++){
91    rx_bits = vit_input & RX_MASK;
      __acs_max_8s(VIT_METRIC0,VIT_METRIC1,branchMetricTable[rx_bits],0x0,res,tb_hist);
93    TB_PONG[tb_out_offset] = (unsigned int)(tb_hist >> 56);
      VIT_METRIC1 = __builtin_high_64(res);
```

```
95      VIT_METRIC0 = __builtin_low_64(res);
        vit_input = vit_input >> 2;
97      tb_out_offset = (tb_out_offset+1)%TOT_INPUT_BITS;
    }
99


101 //TraceBack
    vit_output = viterbiTraceBack(TB_PING,stateVect,state);
103
    // Determine the state with the maximum path metric.
105 if(stage == LAST_STAGE){
      (*state) = calcMaxState(VIT_METRIC0,VIT_METRIC1);
107 }
    stage = (stage+1)%CONV_SIZE;
109 return vit_output;
  }
111
  int calcMaxState(long long int VIT_METRIC0, long long int VIT_METRIC1){
113 short max_state;
    int state_index=0;
115 max_state = (VIT_METRIC1 >> 48) & 0xffff;//state0

117 if(((VIT_METRIC1 >> 32)&0xffff) > max_state){//state4
      max_state = (VIT_METRIC1 >> 32)&0xffff;
119   state_index = 4;
    }
121 if(((VIT_METRIC1 >> 16)&0xffff) > max_state){//state2
      max_state = (VIT_METRIC1 >> 16)&0xffff;
123   state_index = 2;
    }
125 if((VIT_METRIC1&0xffff) > max_state){//state6
      max_state = (VIT_METRIC1&0xff)&0xffff;
127   state_index = 6;
    }
129 if(((VIT_METRIC1 >> 48)&0xffff) > max_state){//state1
      max_state = (VIT_METRIC1 >> 48)&0xffff;
131   state_index = 1;
    }
133 if(((VIT_METRIC1 >> 32)&0xffff) > max_state){//state5
      max_state = (VIT_METRIC1 >> 32)&0xffff;
135   state_index = 5;
    }
137 if(((VIT_METRIC1 >> 16)&0xffff) > max_state){//state3
      max_state = (VIT_METRIC1 >> 16)&0xffff;
139   state_index = 3;
    }
141 if(((VIT_METRIC1)&0xffff) > max_state){//state7
      max_state = (VIT_METRIC1&0xffff);
```

```
143     state_index = 7;
    }
145   return state_index;
  }
147
  void convolutionalInit(void){
149   int i;
    for (i=0;i<NUM_STATES/2;i++){
151     // Generate the dependencies to previous states
      stateVect[i].prev[0] = ((i<<1)^0x0)&0x7;
153     stateVect[i].prev[1] = ((i<<1)^0x1)&0x7;

155     // Generate the receive bits under traceback to either preceding state. This
      // is generated by traversing down the RHS of the Viterbi diagram and observing
157     // the output produced under branching. There is only one possible recBit for any
      // given state.
159     stateVect[i].recBit = 0;
    }
161
    for (i=NUM_STATES/2;i<NUM_STATES;i++){
163     // Generate the dependencies to previous states
      stateVect[i].prev[0] = ((i<<1)^0x0)&0x7;
165     stateVect[i].prev[1] = ((i<<1)^0x1)&0x7;

167     // Generate the receive bits under traceback to either preceding state. This
      // is generated by traversing down the RHS of the Viterbi diagram and observing
169     // the output produced under branching. There is only one possible recBit for any
      // given state.
171     stateVect[i].recBit = 1;
    }
173   stateVect[0].tbPos = 7;
    stateVect[4].tbPos = 6;
175   stateVect[2].tbPos = 5;
    stateVect[6].tbPos = 4;
177   stateVect[1].tbPos = 3;
    stateVect[5].tbPos = 2;
179   stateVect[3].tbPos = 1;
    stateVect[7].tbPos = 0;
181 }

183 unsigned int viterbiTraceBack(unsigned int THR[TOT_INPUT_BITS], State stateVect[NUM_STATES], int *
      state_index){
    static int tb_in_offset = 0;
185   unsigned int vit_output, tb_bit;
    int i;
187
    vit_output = 0;
189   for(i=0;i<BITS_IN_INPUT;i++){
```

```
    vit_output = vit_output << 1;
    vit_output = (vit_output)^(stateVect[(*state_index)].recBit);
    tb_bit = (THR[TOT_INPUT_BITS-1-tb_in_offset] >> (stateVect[(*state_index)].tbPos)) & 0x1;
    (*state_index) = stateVect[(*state_index)].prev[tb_bit];
    tb_in_offset = (tb_in_offset+1)%TOT_INPUT_BITS;
  }
  //printf("bye ",i);
  return vit_output;
}
```

Listing 4: dspCode/convolutional.c

```
#include "cyclic_prefix.h"
#include <stdio.h>

void cp_insert(float ifft_output[IFFT_OUTPUT_SIZE], float ifft_output_cp[NUM_CP]){
  int i;
  for(i = 0; i<NUM_CP; i++){
    ifft_output_cp[i] = ifft_output[i + (IFFT_OUTPUT_SIZE-NUM_CP)]; // copy the last 16 sample
  }
}

void cp_remove(float received[IFFT_OUTPUT_SIZE+NUM_CP],float output[IFFT_OUTPUT_SIZE]){
  int i,j=0;
  for (i=NUM_CP;i<(NUM_CP+IFFT_OUTPUT_SIZE);i++){
    output[j]=received[i];
    j++;
  }
}
```

Listing 5: dspCode/cyclic_prefix.c

```
#include "input.h"

void cp_insert(float (*), float (*));
void cp_remove(float received[IFFT_OUTPUT_SIZE+NUM_CP],float output[IFFT_OUTPUT_SIZE]);
```

Listing 6: dspCode/cyclic_prefix.h

```
//*****************************************************************
//      Defines for the FFT routines for TigerSHARC family of processors
//      FFTDef.h
//*****************************************************************
```

```
#if !defined(__FFTDEF_H_)
 #define __FFTDEF_H_

 //********************************** Macros ************************

 #define mPUSHQ(arg) \
   Q[k27 += -4] = arg;;

#define mPOPQ(arg) \
    k27 = k27 + 4;;  \
   arg = Q[k27 += 0];;

#define mENTER \
    j26 = j27 - 0x40; k26 = k27 - 0x40;; \
   [j27 += 0xFFFFFFF4] = cJMP; k27 = k27 - 0x04;;

#define mRETURN \
    cjmp = [j26 + 0x40];; \
   cjmp(ABS)(NP); j27:24 = Q[j26 + 0x44]; k27:24 = Q[k26 + 0x44];;


 //*********************************************************************
/*here N and the fft type must be defined.
 -N represents the number of input points of the fft, either real or complex
-The fft type may be real and then FFT_real must be defined or
 complex and then FFT_real definition must be commented
-In real FFT case, N may be 64, 128, ..., 32768 for TS201 and 64, 128,..., 8192 for TS101
 -In complex FFT case, N may be 32, 64, ..., 16384 for TS201 and 32, 64,..., 4096 for TS101
-for all other cases, an error message is shown on the screen after the project is built
 */

 #define N          64       // Complex FFT size
//#define FFT_Real
 //*********************************************************************

 #if N==32
     #ifdef FFT_Real
         #define initialization_error
         #define MAX_FFT_SIZE  16384
         #define outbuff_size 16384
     #else
         #define MAX_FFT_SIZE  32
         #define outbuff_size 64
     #endif
#elif N==64
     #define MAX_FFT_SIZE  64
     #ifdef FFT_Real
         #define outbuff_size 64
```

```
54      #else
            #define outbuff_size 128
56      #endif
    #elif N==128
58      #define MAX_FFT_SIZE  128
        #ifdef FFT_Real
60          #define outbuff_size 128
        #else
62          #define outbuff_size 256
        #endif
64  #elif N==256
        #define MAX_FFT_SIZE  256
66      #ifdef FFT_Real
            #define outbuff_size 256
68      #else
            #define outbuff_size 512
70      #endif
    #elif N==512
72      #define MAX_FFT_SIZE  512
        #ifdef FFT_Real
74          #define outbuff_size 512
        #else
76          #define outbuff_size 1024
        #endif
78  #elif N==1024
        #define MAX_FFT_SIZE  1024
80      #ifdef FFT_Real
            #define outbuff_size 1024
82      #else
            #define outbuff_size 2048
84      #endif
    #elif N==2048
86      #define MAX_FFT_SIZE  2048
        #ifdef FFT_Real
88          #define outbuff_size 2048
        #else
90          #define outbuff_size 4096
        #endif
92  #elif N==4096
        #define MAX_FFT_SIZE  4096
94      #ifdef FFT_Real
            #define outbuff_size 4096
96      #else
            #define outbuff_size 8192
98      #endif
    #elif N==8192
100         #ifdef FFT_Real
                #define MAX_FFT_SIZE  8192
```

```
102             #define outbuff_size 8192
            #else
104             #ifdef __ADSPTS201__
                    #define MAX_FFT_SIZE  8192
106                 #define outbuff_size 16384
                #else
108                 #define initialization_error
                    #define outbuff_size 8192
110                 #define MAX_FFT_SIZE  8192
                #endif
112         #endif
    #elif N==16384
114     #ifdef __ADSPTS201__
            #define MAX_FFT_SIZE  16384
116         #ifdef FFT_Real
                #define outbuff_size 16384
118         #else
                #define outbuff_size 32768
120         #endif
        #else
122         #define initialization_error
            #define outbuff_size 8192
124         #define MAX_FFT_SIZE  8192
        #endif
126 #elif N==32768
        #ifdef __ADSPTS201__
128         #ifdef FFT_Real
                #define MAX_FFT_SIZE  32768
130             #define outbuff_size 32768
            #else
132             #define initialization_error
                #define outbuff_size 8192
134             #define MAX_FFT_SIZE  8192
            #endif
136     #else
          #define initialization_error
138       #define outbuff_size 16384
          #define MAX_FFT_SIZE  16384
140     #endif
    #else
142     #define initialization_error
        #define MAX_FFT_SIZE  8192
144     #define outbuff_size 8192
    #endif
146
    #define REAL              0
148 #define COMPLEX           1
```

```
150#endif //__FFTDEF_H_
```

Listing 7: dspCode/FFTDef.h

```
/*  fft32.asm

Prelim rev. October 19, 2003 - BL
  Rev. 1.0 - added real inputs case - PM

This is assembly routine for the Complex radix-2 C-callable FFT on TigerSHARC
family of DSPs.

I. Description of Calling.

  1. Inputs:
    j4 -> input (ping-pong buffer 1)
    j5 -> ping-pong buffer 1
        j6 -> ping-pong buffer 2
    j7 -> output
    j27+0x18 -> N = Number of points
    j27+0x19 -> REAL or COMPLEX

  2. C-Calling Example:
    fft32(&(input), &(ping_pong_buffer1), &(ping_pong_buffer2), &(output), N, COMPLEX);

  3. Limitations:
    a. All buffers must be aligned on memory boundary which is a multiple of 4.
    b. N must be between 32 and MAX_FFT_SIZE.
    c. If memory space savings are required and input does not have to be
       preserved, ping_pong_buffer1 can be the same buffer as input.
    d. If memory space savings are required, output can be the same buffer
       as ping_pong_buffer2 if the number of FFT stages is even (i.e.
       Log2(N) is even) or the same as ping_pong_buffer1 if the number of
       FFT stages is odd (i.e. Log2(N) is odd).

  4. MAX_FFT_SIZE can be selected via #define. Larger values allow for more choices
     of N, but its twiddles will occupy more memory.
  5. This C - callable function can process up to 64K blocks of data on TS201
      (16K blocks on TS101) because C environment itself necessitates memory.
      Therefore, if more input points are necessary, assembly language development
      may become a must. On TS201, a block of memory is 128K words long, so
      maximum N is 128K real points or 64K complex points. TS101 contains
      only 2 blocks of data memory of 64K words and 4 buffers must be
      accommodated. Therefore, maximum N is 32K real words or 16K complex words.


II. Description of the FFT algorithm.
```

```
44
      1. The input data is treated as complex interleaved N-point.
46    2. Due to re-ordering, no stage can be done in-place.
      3. The bit reversal and the first two stages are combined into
48       a single loop. This loop takes data from input and stores it
         in the ping-pong buffer1.
50    4. Each subsequent stage ping-pongs the data between the two ping-pong
         buffers. The last stage uses FFT output buffer for its output.
52    5. Although the FFT is designed to be called with any point size
         N <= MAX_FFT_SIZE by subsampling the twiddle factors, for ADSP-TS20x
54       processors, the best cycle optimization is achieved when MAX_FFT_SIZE=N.
         For ADSP-TS101 all choices of MAX_FFT_SIZE are equally optimal.
56

58  III. Description of the REAL FFT algorithm.

60    1. The input data is treated as complex interleaved N/2-point. The N/2 point complex
         FFT will be computed first. Thus, N is halved, now number of points = N/2.
62
      2. Details and source code of the N/2 point complex FFT are in II above.
64
      3. Real re-combine:
66      Here the complex N/2-point FFT computed in the previous steps is recombined to
        produce the N-point real FFT. If G is the complex FFT and F is the real FFT,
68      the formula for F is given by:

70      F(n) = 0.5*(G(n)+conj(G(N/2-n)))-0.5*i*exp(-2*pi*i*n/N)*(G(n)-conj(G(N/2-n))).

72      From this the following can be derived:

74      conj(F(N/2-n)) = 0.5*(G(n)+conj(G(N/2-n)))+0.5*i*exp(-2*pi*i*n/N)*(G(n)-conj(G(N/2-n))).

76      Thus, this can be computed in (n,N/2-n) pairs, as follows (dropping factor of 2):

78      G(n) ------------------------------------>------------------------------>------> F(n)
                                                \ +/                          \ +/
80                                               \/                            \/
                                                 /\                            /\
82                        conj                  / -\  exp(-2*pi*i*n)*i        / -\   conj
        G(N/2-n) -----> conj(G(N/2-n))------>------------------------------>------> F(N/2-n)
84
        This is very efficient on the TigerSHARC architecture due to the add/subtract
86      instruction.


88
  IV. For all additional details regarding this algorithm and code, see EE-218
90      application note, available from the ADI web site.
```

```
92 */
  //********************************** Includes *********************************
94
  #include "FFTDef.h"
96#include "defts201.h"

98 //************************* Externs ***********************************

100 .extern _twiddles;

102 //****************************** FFT Routine ******************************
  .section program;
104 .global _FFT32;

106 _FFT32:

108 //******************************** Prologue *******************************

110   mENTER
    mPUSHQ(j19:16)
112     mPUSHQ(xR31:28)
    mPUSHQ(xR27:24)
114   mPUSHQ(yR31:28)
    mPUSHQ(yR27:24)
116
  //********************************** Setup ********************************
118   j17 = [j27 + 0x18];;                    //j17 = N
    j11 = [j27 + 0x19];;                    // j11=COMPLEX or REAL, off the stack
120
    comp(j11,COMPLEX);;                       // Complex or Real?
122   if jeq, jump _FFTStages1and2;;
    j17=ashiftr j17;;                         // if Real, half N
124 //*****************************************************************************
  _FFTStages1and2:
126
    j11 = j31 + j17;;                                   // j11=N
128
    xr3=j11; k7=k31+_twiddles;;
130   k1=j11; j8=lshiftr j11;;                           // k1=N, j8=N/2
    j9=lshiftr j8; xr0=MAX_FFT_SIZE; xr3=LD0 r3;;             // j9=N/4, compute the twiddle
      stride
132   k8=lshiftr k1; xr0=LD0 r0; xr1=j11;;
    k8=lshiftr k8; xr1=LD0 r1; xr2=(31-3);;                 // k8=N/4, Compute Stages-3
134   k0=j4; k10=lshiftr k8; xr1=r1-r0; xr0=lshift r0 by -32;;        // k0->input, xr1=bit
      difference between MAX and N
    k10=lshiftr k10; xr0=bset r0 by r1; xr30=r2-r3;;            // k10=N/16, xr30=Stages-3
136   k10=k10-1; xr0=lshift r0 by 2; LC1=xr30;;                 // k10=N/16-1, LC1=Stages-3
    k9=xr0; k4=k31+(MAX_FFT_SIZE/4-1);;
```

```
138   k4=not k4; j10=lshiftr j9;;                          // initial twiddles pointer mask, j10=N/8

140 //****************** Bit Reverse and Stages 1 & 2 *****************************

142   k5=lshiftr k1;;                                    // k5=N/2
      j0=j31+j6; k6=k6-k6;;                               // j0->ping_pong_buffer2
144   j1=j0+j9; LC0=k10;;                                 // j1->ping_pong_buffer2+N/4, LC0=N/16-1
      j2=j1+j9; k1=k0+k5;;                                // j2->ping_pong_buffer2+N/2, k1->input+N/2
146   j3=j2+j9; k2=k1+k5;;                                // j3->ping_pong_buffer2+3N/4, k2->input+N
      j12=j3+j9; k3=k2+k5;;                               // j12->ping_pong_buffer2+N, k3->input+3N/2
148   j13=j12+j9; k5=lshiftr k5;;                            // j13->ping_pong_buffer2+5N/4, k5=N/4
      j14=j13+j9; r1:0=q[k0+k6];;                            // j14->ping_pong_buffer2+3N/2
150   j15=j14+j9; r3:2=q[k2+k6];;                            // j15->ping_pong_buffer2+7N/4

152   r5:4=q[k1+k6];;
      r7:6=q[k3+k6];;
154
      k6=k6+k5 (br); fr0=r0+r2, fr20=r0-r2;;
156   r9:8=q[k0+k6]; fr2=r1+r3, fr29=r1-r3;;
      r11:10=q[k2+k6]; fr4=r4+r6, fr21=r4-r6;;
158   r13:12=q[k1+k6]; fr5=r5+r7, fr28=r5-r7;;

160   r15:14=q[k3+k6]; fr18=r8+r10, fr22=r8-r10;;
      k6=k6+k5 (br); fr19=r9+r11, fr31=r9-r11;;
162   fr26=r12+r14, fr23=r12-r14;;
      fr27=r13+r15, fr30=r13-r15;;
164
      fr20=r20+r28, fr28=r20-r28;;
166   fr29=r29+r21, fr21=r29-r21;;
      fr22=r22+r30, fr30=r22-r30;;
168   fr31=r31+r23, fr23=r31-r23;;

170 .align_code 4;
  _Stages1and2Loop:
172     r1:0=q[k0+k6]; q[j2+=4]=yr23:20; fr16=r0+r4, fr24=r0-r4;;
        r3:2=q[k2+k6]; q[j3+=4]=xr23:20; fr17=r2+r5, fr25=r2-r5;;
174     r5:4=q[k1+k6]; q[j14+=4]=yr31:28; fr18=r18+r26, fr26=r18-r26;;
        r7:6=q[k3+k6]; q[j15+=4]=xr31:28; fr19=r19+r27, fr27=r19-r27;;
176
        k6=k6+k5 (br); q[j0+=4]=yr19:16; fr0=r0+r2, fr20=r0-r2;;
178     r9:8=q[k0+k6]; q[j1+=4]=xr19:16; fr2=r1+r3, fr29=r1-r3;;
        r11:10=q[k2+k6]; q[j12+=4]=yr27:24; fr4=r4+r6, fr21=r4-r6;;
180     r13:12=q[k1+k6]; q[j13+=4]=xr27:24; fr5=r5+r7, fr28=r5-r7;;

182     r15:14=q[k3+k6]; fr18=r8+r10, fr22=r8-r10;;
        k6=k6+k5 (br); fr19=r9+r11, fr31=r9-r11;;
184     fr26=r12+r14, fr23=r12-r14;;
        fr27=r13+r15, fr30=r13-r15;;
```

```
186
      fr20=r20+r28, fr28=r20-r28;;
188   fr29=r29+r21, fr21=r29-r21;;
      fr22=r22+r30, fr30=r22-r30;;
190 .align_code 4;
      if NLC0E, jump _Stages1and2Loop;
192   fr31=r31+r23, fr23=r31-r23;;

194  q[j2+=4]=yr23:20; fr16=r0+r4, fr24=r0-r4;;
    q[j3+=4]=xr23:20; fr17=r2+r5, fr25=r2-r5;;
196  q[j14+=4]=yr31:28; fr18=r18+r26, fr26=r18-r26;;
    q[j15+=4]=xr31:28; fr19=r19+r27, fr27=r19-r27;;
198
    q[j0+=4]=yr19:16;;
200  q[j1+=4]=xr19:16;;
    q[j12+=4]=yr27:24;;
202  q[j13+=4]=xr27:24;;

204 //********************** Stages 3 to Log2(N)-1  ******************************

206  j0=j31+j6; k5=k31+0;;

208 .align_code 4;
  _StageLoop:
210   yr3:0=q[j0+=4];   k3=k5 and k4;;                                    // F1,    K1
      xr3:0=q[j0+=4];   r5:4=l[k7+k3];;                                   // F2,    K2
212   LC0=k10;          k5=k5+k9;;                                        //        K3,
        M1

214   yr11:8=q[j0+=4]; k3=k5 and k4;    fr6=r2*r4;;                         // F1+,
      K1+
      xr11:8=q[j0+=4]; r13:12=l[k7+k3]; fr7=r3*r5;;                       // F2+,   K2+,
        M2
216                                     fr14=r2*r5;;                      //
       M3
      j1=j31+j5;        k5=k5+k9;       fr6=r10*r12;  fr16=r6-r7;;        //        K3+,
        M1+,   A1
218
      yr23:20=q[j0+=4]; k3=k5 and k4;   fr15=r3*r4;;                      // F1++,  K1
      ++,  M4
220   xr23:20=q[j0+=4]; r5:4=l[k7+k3];  fr7=r11*r13;;                     // F2++,  K2
      ++,  M2+
                                        fr14=r10*r13; fr17=r14+r15;;      //
        M3+,   A2
222   j2=j1+j11;         k5=k5+k9;       fr6=r22*r4;   fr18=r6-r7;;       //        K3
      ++,  M1++,  A1+

224   yr31:28=q[j0+=4]; k3=k5 and k4;   fr15=r11*r12; fr24=r0+r16, fr26=r0-r16;;  // F1+++, K1
```

```
    +++, M4+,    A3
    xr31:28=q[j0+=4]; r13:12=l[k7+k3]; fr7=r23*r5;    fr25=r1+r17, fr27=r1-r17;;      // F2+++, K2
    +++, M2++,   A4
226         q[j1+=4]=r25:24;                    fr14=r22*r5;   fr19=r14+r15;;                    // S1,
              M3++,   A2+


228
  .align_code 4;
230 _BflyLoop:
        q[j2+=4]=r27:26;  k5=k5+k9;         fr6=r30*r12;  fr16=r6-r7;;                    // S2----,K3
    -,    M1-,    A1--
232
        yr3:0=q[j0+=4];   k3=k5 and k4;     fr15=r23*r4;  fr24=r8+r18,  fr26=r8-r18;;     // F1,     K1
    ,     M4--,   A3---
234     xr3:0=q[j0+=4];   r5:4=l[k7+k3];    fr7=r31*r13;  fr25=r9+r19,  fr27=r9-r19;;     // F2,     K2
    ,     M2-,    A4---
        q[j1+=4]=r25:24;                    fr14=r30*r13; fr17=r14+r15;;                  // S1---,
          M3-,    A2--
236     q[j2+=4]=r27:26;  k5=k5+k9;         fr6=r2*r4;    fr18=r6-r7;;                    // S2---, K3
    ,     M1,     A1-

238     yr11:8=q[j0+=4];  k3=k5 and k4;     fr15=r31*r12; fr24=r20+r16, fr26=r20-r16;;    // F1+,    K1
    +,    M4-,    A3--
        xr11:8=q[j0+=4];  r13:12=l[k7+k3]; fr7=r3*r5;     fr25=r21+r17, fr27=r21-r17;;    // F2+,    K2
    +,    M2,     A4--
240         q[j1+=4]=r25:24;                    fr14=r2*r5;   fr19=r14+r15;;                  // S1
    --,         M3,     A2-
        q[j2+=4]=r27:26;  k5=k5+k9;         fr6=r10*r12;  fr16=r6-r7;;                    // S2--,   K3
    +,    M1+,    A1
242
        yr23:20=q[j0+=4]; k3=k5 and k4;     fr15=r3*r4;   fr24=r28+r18, fr26=r28-r18;;    // F1++,   K1
    ++,   M4,     A3-
244     xr23:20=q[j0+=4]; r5:4=l[k7+k3];    fr7=r11*r13;  fr25=r29+r19, fr27=r29-r19;;    // F2++,   K2
    ++,   M2+,    A4-
        q[j1+=4]=r25:24;                    fr14=r10*r13; fr17=r14+r15;;                  // S1-,
          M3+,    A2
246     q[j2+=4]=r27:26;  k5=k5+k9;         fr6=r22*r4;   fr18=r6-r7;;                    // S2-,    K3
    ++,   M1++,   A1+

248     yr31:28=q[j0+=4]; k3=k5 and k4;     fr15=r11*r12; fr24=r0+r16,  fr26=r0-r16;;     // F1+++, K1
    +++, M4+,    A3
        xr31:28=q[j0+=4]; r13:12=l[k7+k3]; fr7=r23*r5;    fr25=r1+r17,  fr27=r1-r17;;   // F2+++, K2
    +++, M2++,   A4
250 .align_code 4;
          if NLC0E, jump _BflyLoop;
252       q[j1+=4]=r25:24; fr14=r22*r5;  fr19=r14+r15;;                                 // S1,
    M3++,   A2+
```

```
254     q[j2+=4]=r27:26;                    fr6=r30*r12;  fr16=r6-r7;;                    // S2----,
            M1-, A1--

256     j0=j31+j5;                          fr15=r23*r4;  fr24=r8+r18,  fr26=r8-r18;;     //
            M4--,  A3---, swap ping-pong pointers
        j5=j31+j6;                          fr7=r31*r13;  fr25=r9+r19,  fr27=r9-r19;;     //
            M2-,   A4---
258     q[j1+=4]=r25:24;                    fr14=r30*r13; fr17=r14+r15;;                  // S1---,
            M3-,   A2--
        q[j2+=4]=r27:26;                                  fr18=r6-r7;;                  // S2---,
                    A1-

260
        j6=j31+j0;                          fr15=r31*r12; fr24=r20+r16, fr26=r20-r16;;    //
            M4-,   A3--
262                                                       fr25=r21+r17, fr27=r21-r17;;    //
                    A4--
            q[j1+=4]=r25:24;                              fr19=r14+r15;;                  // S1--,
                    A2-
264     q[j2+=4]=r27:26;                    fr24=r28+r18, fr22=r28-r18;;          // S2--
            A3-

266     j0=j31+j6;                          fr25=r29+r19, fr23=r29-r19;;    //                    A4
        -
        q[j1+=4]=r25:24; k5=k31+0;;                                                       //
         S1-
268 .align_code 4;
        if NLC1E, jump _StageLoop;
270     q[j2+=4]=r23:22; k4=ashiftr k4;;                                                  // S2-,
        shift the mask

272 //***************************** Last stage *********************************
    k9 = ashiftr k9;;//in this manner any MAX_FFT_SIZE can be used
274
    yr3:0=q[j0+=4]; yr5:4 = l[k7+=k9];;                                   // F1,
276
    xr3:0=q[j0+=4];   xr5:4=l[k7+=k9];;                                               // F2,    K2
278 j1=j31+j7;          fr6=r2*r4; LC0=k10;;                              //              M1


280
    yr11:8=q[j0+=4];  yr13:12=l[k7+=k9];;
            // F1+
282 xr11:8=q[j0+=4];  xr13:12=l[k7+=k9]; fr7=r3*r5;;                              // F2+,   K2
      +,   M2
      j2=j1+j11;                    fr14=r2*r5;;                          //
      M3
284                                     fr6=r10*r12;  fr16=r6-r7;;                  //
        M1+,   A1
```

```
286   yr23:20=q[j0+=4]; yr5:4=l[k7+=k9];   fr15=r3*r4;;                                          // F1++,
          M4
      xr23:20=q[j0+=4]; xr5:4=l[k7+=k9];    fr7=r11*r13;;                                        // F2++,   K2
        ++,   M2+
288                                         fr14=r10*r13; fr17=r14+r15;;                         //
          M3+,    A2
                                            fr6=r22*r4;    fr18=r6-r7;;                         //
          M1++,   A1+
290
      yr31:28=q[j0+=4]; yr13:12=l[k7+=k9];fr15=r11*r12; fr24=r0+r16, fr26=r0-r16;;             // F1+++,
          M4+,    A3
292   xr31:28=q[j0+=4]; xr13:12=l[k7+=k9]; fr7=r23*r5;    fr25=r1+r17, fr27=r1-r17;;           // F2+++, K2
        +++, M2++,   A4
      q[j1+=4]=r25:24;                      fr14=r22*r5;  fr19=r14+r15;;                        // S1,
          M3++,   A2+
294
  .align_code 4;
296 _BflyLastLoop:
      q[j2+=4]=r27:26;                      fr6=r30*r12;  fr16=r6-r7;;                         // S2----,
          M1-,    A1--
298
      yr3:0=q[j0+=4];   yr5:4=l[k7+=k9];   fr15=r23*r4;  fr24=r8+r18,  fr26=r8-r18;;          // F1,
          M4--,   A3---
300   xr3:0=q[j0+=4];   xr5:4=l[k7+=k9];   fr7=r31*r13;  fr25=r9+r19,  fr27=r9-r19;;          // F2,     K2
      ,     M2-,    A4---
      q[j1+=4]=r25:24;                      fr14=r30*r13; fr17=r14+r15;;                       // S1---,
          M3-,    A2--
302   q[j2+=4]=r27:26;                      fr6=r2*r4;    fr18=r6-r7;;                         // S2---,
          M1,     A1-

304   yr11:8=q[j0+=4];  yr13:12=l[k7+=k9];fr15=r31*r12; fr24=r20+r16, fr26=r20-r16;;           // F1+,
          M4-,    A3--
      xr11:8=q[j0+=4];  xr13:12=l[k7+=k9];fr7=r3*r5;    fr25=r21+r17, fr27=r21-r17;;          // F2+,    K2
      +,    M2,     A4--
306       q[j1+=4]=r25:24;                    fr14=r2*r5;   fr19=r14+r15;;                     // S1--,
              M3,     A2-
      q[j2+=4]=r27:26;                      fr6=r10*r12;  fr16=r6-r7;;                         // S2--,
          M1+,    A1
308
      yr23:20=q[j0+=4]; yr5:4=l[k7+=k9];   fr15=r3*r4;   fr24=r28+r18, fr26=r28-r18;;          // F1++,
          M4,     A3-
310   xr23:20=q[j0+=4]; xr5:4=l[k7+=k9];   fr7=r11*r13;  fr25=r29+r19, fr27=r29-r19;;          // F2++,   K2
      ++,   M2+,    A4-
      q[j1+=4]=r25:24;                      fr14=r10*r13; fr17=r14+r15;;                       // S1-,
          M3+,    A2
312   q[j2+=4]=r27:26;                      fr6=r22*r4;   fr18=r6-r7;;                         // S2-,
          M1++,   A1+
```

```
314     yr31:28=q[j0+=4];yr13:12=l[k7+=k9]; fr15=r11*r12; fr24=r0+r16,  fr26=r0-r16;;      // F1+++,
            M4+,   A3
        xr31:28=q[j0+=4];xr13:12=l[k7+=k9]; fr7=r23*r5;   fr25=r1+r17,  fr27=r1-r17;;      // F2+++, K2
        +++, M2++,  A4
316 .align_code 4;
          if NLC0E, jump _BflyLastLoop;
318       q[j1+=4]=r25:24;  fr14=r22*r5;  fr19=r14+r15;;                          // S1,           M3
        ++,  A2+

320   q[j2+=4]=r27:26;                  fr6=r30*r12;  fr16=r6-r7;;                          // S2----,
            M1-, A1--

322                                     fr15=r23*r4;  fr24=r8+r18,  fr26=r8-r18;;          //
            M4--,  A3---
                                        fr7=r31*r13;  fr25=r9+r19,  fr27=r9-r19;;          //
            M2-,   A4---
324   q[j1+=4]=r25:24;                  fr14=r30*r13; fr17=r14+r15;;                        // S1---,
            M3-,   A2--
      q[j2+=4]=r27:26;                                fr18=r6-r7;;                        // S2---,
                  A1-

326
                                        fr15=r31*r12; fr24=r20+r16, fr26=r20-r16;;         //
            M4-,   A3--
328                                                   fr25=r21+r17, fr27=r21-r17;;         //
                  A4--
        q[j1+=4]=r25:24;                             fr19=r14+r15;;                      // S1--,
                A2-
330   q[j2+=4]=r27:26;;                                                                   // S2--

332                                                   fr24=r28+r18, fr26=r28-r18;;         //
                  A3-
                                                      fr25=r29+r19, fr27=r29-r19;;         //
                  A4-
334   q[j1+=4]=r25:24;;                                                                   // S1-
      q[j2+=4]=r27:26;;                                                                   // S2-
336
      j11=[j27+0x19];;                          // j11=COMPLEX or REAL, off the stack
338   comp(j11,COMPLEX);;                       // Complex or Real?
    .align_code 4;
340   if jeq, jump _FFTEpilogue;;               // If Complex, done

342 //****************************** Real re-combine ******************************

344 //j17=N/2, j7=output

346   k8=k31+_twiddles; j0=j31+j7;;                 // k8->twiddles, j0->internal buffer
      k9=ashiftr k9; j10=j31+j7;;                 // k9=twiddle stride, j10->internal buffer
348
```

```
    j14=j17+j17;;                                   // j14=N (N/2 complex values)
350 j14=j14-4;;                                     // j14=N-4 real=N/2-2 complex
    j1=j0+j14;;                                     // j1->internal buffer+(N/2-2)
352 j14=j10+j14;;                                   // j14->internal buffer+(N/2-2)

354 j29 = ashiftr j17;;                             // j29=N/4
    k15=k31+MAX_FFT_SIZE/4; j30=ashiftr j29;;       // k15=N/4*twiddle_stride, j30=N/8
356
    j30 = ashiftr j30;;                             // N/16
358
    k8=k8+k9; r0=l[j7+j17];;                        // k8->twiddles+1, get G(N/4)
360 j0=j0+2; k12=k8+k15;;                            // j0->internal buffer+1, k12->twiddles+N/8+1
    LC0=j30; fr0=r0+r0; j2=j0+j29;;                 // LC0=N/16, compute F(N/4)=2*conj(G(N/4)), j2->
      internal buffer+1+N/8
362 j3=j1-j29;;                                     // j3->internal buffer+3N/8-2
    xfr0=-r0; j10=j10+2; k10=yr0;;                  // j10->internal buffer+1, k10=Im(F(N/4))
364 j12=j10+j29;;                                   // j12->internal buffer+N/8+1
    if LC0E; j13=j14-j29; k11=xr0;;                 // LC0=N/16-1, j13->internal buffer+3N/8-2, k11=
      Re(F(N/4))
366
    yr3:0=DAB q[j0+=4];;                            // Prime the DAB
368 xr3:0=DAB q[j2+=4];;                            // Prime the DAB

370 yr3:0=DAB q[j0+=4];;                            // yr0=Re(G(n)), yr1=Im(G(n)), yr2=Re(G(n+1)), yr3=Im(
      G(n+1))
    xr3:0=DAB q[j2+=4];;                            // xr0=Re(G(n+N/8)), xr1=Im(G(n+N/8))
372                                 // xr2=Re(G(n+1+N/8)), xr3=Im(G(n+1+N/8))
    yr7:4=q[j1+=-4]; xr9:8=l[k12+=k9];;             // yr4=Re(G(N/2-(n+1))), yr5=Im(G(N/2-(n+1)))
374                                 // yr6=Re(G(N/2-n)), yr7=Im(G(N/2-n))
                                    // twiddles(n+N/8) - want to mult by sin(x)-icos(x)
376 xr7:4=q[j3+=-4]; xr11:10=l[k12+=k9];;             // xr4=Re(G(N/2-(n+1+N/8))), xr5=Im(G(N/2-(n
      +1+N/8)))
                                    // xr6=Re(G(N/2-(n+N/8))), xr7=Im(G(N/2-(n+N/8)))
378                                 // twiddles(n+1+N/8)
    if LC0E; fr16=r0+r6, fr20=r0-r6; yr9:8=l[k8+=k9];;      // LC0=N/16-2, r16=Re(G(n)+conj(G(N/2-
      n))), r20=Re(G(n)-conj(G(N/2-n)))
380                                 // twiddles(n)
    fr18=r2+r4, fr22=r2-r4; yr11:10=l[k8+=k9];;         // r18=Re(G(n+1)+conj(G(N/2-(n+1)))), r22=
      Re(G(n+1)-conj(G(N/2-(n+1))))
382                                 // twiddles(n+1)
    fr24=r20*r9; fr21=r1+r7, fr17=r1-r7;;           // r24=s(n)*Re(G(n)-conj(G(N/2-n)))
384                                 // r17=Im(G(n)+conj(G(N/2-n))), r21=Im(G(n)-conj(G(N/2-n)))
    fr26=r22*r11; fr23=r3+r5, fr19=r3-r5; xr3:0=DAB q[j2+=4];;   // r26=s(n+1)*Re(G(n+1)-conj(G(N
      /2-(n+1))))
386                                 // r19=Im(G(n+1)+conj(G(N/2-(n+1)))), r23=Im(G(n+1)-conj(G(N/2-(
      n+1))))
                                    // xr3:0=next G(n+2+N/8), G(n+3+N/8)
388 fr25=r21*r8;  yr3:0=DAB q[j0+=4];;              // r25=c(n)*Im(G(n)-conj(G(N/2-n))), yr3:0=
```

```
    next G(n+2), G(n+3)
  fr27=r23*r10;;                                  // r27=c(n+1)*Im(G(n+1)-conj(G(N/2-(n+1))))
390 fr24=r24+r25; fr25=r21*r9; yr7:4=q[j1+=-4];;          // r24=Re(-i*exp(2*pi*i*n)(G(n)-conj(G(N
    /2-n))))
                                  // r13=s(n)*Im(G(n)-conj(G(N/2-n)))
392                               // yr7:4=next G(N/2-(n+2)), G(N/2-(n+3))
  fr26=r26+r27; fr27=r23*r11; xr7:4=q[j3+=-4];;          // r26=Re(-i*exp(2*pi*i*(n+1))(G(n+1)-conj
    (G(N/2-(n+1)))))
394                               // r27=s(n+1)*Im(G(n+1)-conj(G(N/2-(n+1))))
                                  // xr7:4=next G(N/2-(n+2+N/8)), G(N/2-(n+3+N/8))
396 fr13=r20*r8; fr12=r16+r24, fr30=r16-r24;;            // r13=c(n)*Re(G(n)-conj(G(N/2-n))), r12=Re(
    F(n)), r30=Re(F(N/2-n))
  fr15=r22*r10; fr14=r18+r26, fr28=r18-r26;;              // r15=c(n+1)*Re(G(n+1)-conj(G(N/2-(n+1)))
    )
398                               // r14=Re(F(n+1)), r28=Re(F(N/2-(n+1)))
  fr13=r25-r13; xr9:8=l[k12+=k9];;                  // r13=Im(-i*exp(2*pi*i*x)(G(n)-conj(G(N/2-n))))
    , next twiddles(n+2+N/8)
400 fr15=r27-r15; xr11:10=l[k12+=k9];;                 // r15=Im(-i*exp(2*pi*i*x)(G(n+1)-conj(G(N/2-(
    n+1)))))
                                  // next twiddles(n+3+N/8)
402 .align_code 4;
  _combine_stage:
    fr16=r0+r6, fr20=r0-r6; yr9:8=l[k8+=k9];;          // r16=Re(G(n+2)+conj(G(N/2-(n+2)))), r20=Re
    (G(n+2)-conj(G(N/2-(n+2))))
                                  // next twiddles(n+2)
406   fr18=r2+r4, fr22=r2-r4; yr11:10=l[k8+=k9];;          // r18=Re(G(n+3)+conj(G(N/2-(n+3)))), r22=
    Re(G(n+3)-conj(G(N/2-(n+3))))
                                  // next twiddles(n+3)
408   fr13=r13+r17, fr31=r13-r17;;                  // r13=Im(F(n)), r31=Im(F(N/2-n))
    fr15=r15+r19, fr29=r15-r19; l[j12+=2]=xr13:12;;       // r15=Im(F(n+1)), r29=Im(F(N/2-(n+1))),
     store F(n+N/8)
410   fr24=r20*r9; fr21=r1+r7, fr17=r1-r7; q[j14+=-4]=yr31:28;; // r24=s(n+2)*Re(G(n+2)-conj(G(N/2-(
    n+2))))
                                  // r21=Im(G(n+2)+conj(G(N/2-(n+2)))), r17=Im(G(n+2)-conj(G(N/2-(
    n+2))))
412                               // store F(N/2-n), F(N/2-(n+1))
    fr26=r22*r11; fr23=r3+r5, fr19=r3-r5; xr3:0=DAB q[j2+=4];;  // r26=s(n+3)*Re(G(n+3)-conj(G(N
    /2-(n+3))))
414                               // r23=Im(G(n+3)+conj(G(N/2-(n+3)))), r19=Im(G(n+3)-conj(G(N/2-(
    n+3))))
                                  // xr3:0=next G(n+4+N/8), G(n+5+N/8)
416   fr25=r21*r8; yr3:0=DAB q[j0+=4];;                // r25=c(n+2)*Im(G(n+2)-conj(G(N/2-(n+2))))
                                  // yr3:0=next G(n+4), G(n+5)
418   fr27=r23*r10; q[j13+=-4]=xr31:28;;                 // r27=c(n+3)*Im(G(n+3)-conj(G(N/2-(n+3))))
                                  // store F(N/2-(n+N/8)), F(N/2-(n+1+N/8))
420   fr24=r24+r25; fr25=r21*r9; l[j10+=2]=yr13:12;;       // r24=Re(-i*exp(2*pi*i*x)(G(n+2)-conj(G
    (N/2-(n+2)))))
                                  // r25=s(n+2)*Im(G(n+2)-conj(G(N/2-(n+2)))), store F(n)
```

```
422     fr26=r26+r27; fr27=r23*r11; l[j10+=2]=yr15:14;;          // r26=Re(-i*exp(2*pi*i*x)(G(n+3)-conj(G
        (N/2-(n+3)))))
                                    // r27=s(n+3)*Im(G(n+3)-conj(G(N/2-(n+3)))), store F(n+1)
424     fr13=r20*r8; fr12=r16+r24, fr30=r16-r24; l[j12+=2]=xr15:14;;// r13=cos(n+2)*Re(G(n+2)-conj(G(N
        /2-(n+2))))
                                    // r12=Re(F(n+2)), r30=Re(F(N/2-(n+2))), store F(n+1+N/8)
426     fr15=r22*r10; fr14=r18+r26, fr28=r18-r26; xr7:4=q[j3+=-4];; // r15=cos(n+3)*Re(G(n+3)-conj(G(N
        /2-(n+3))))
                                    // r14=Re(F(n+3)), r28=Re(F(N/2-(n+3)))
428                                 // xr7:4=next G(N/2-(n+4+N/8)), G(N/2-(n+5+N/8))
        fr13=r25-r13; xr9:8=l[k12+=k9]; yr7:4=q[j1+=-4];;       // r13=Im(-i*exp(2*pi*i*x)(G(n+2)-conj(G
        (N/2-(n+2)))))
430                                 // next twiddles(n+4+N/8)
                                    // yr7:4=next G(N/2-(n+4)), G(N/2-(n+5))
432 .align_code 4;
        if NLC0E, jump _combine_stage(P); fr15=r27-r15; xr11:10=l[k12+=k9];;// r15=Im(-i*exp(2*pi*i*x)
        (G(n+3)-conj(G(N/2-(n+3)))))
434                                 // next twiddles(n+5+N/8)

436   fr16=r0+r6, fr20=r0-r6; yr9:8=l[k8+=k9];;             // r16=Re(G(n+4)+conj(G(N/2-(n+4)))), r20=Re
        (G(n+4)-conj(G(N/2-(n+4))))
                                    // next twiddles(n+4)
438   fr18=r2+r4, fr22=r2-r4; yr11:10=l[k8+=k9];;             // r18=Re(G(n+5)+conj(G(N/2-(n+5)))), r22=
        Re(G(n+5)-conj(G(N/2-(n+5))))
                                    // next twiddles(n+5)
440   fr13=r13+r17, fr31=r13-r17;;                    // r13=Im(F(n+2)), r31=Im(F(N/2-(n+2)))
      fr15=r15+r19, fr29=r15-r19;;                    // r15=Im(F(n+3)), r29=Im(F(N/2-(n+3)))
442   fr24=r20*r9; fr21=r1+r7, fr17=r1-r7; yr1:0=l[j31+j7];;       // r24=s(n+4)*Re(G(n+4)-conj(G(N/2-(
        n+4))))
                                    // r21=Im(G(n+4)+conj(G(N/2-(n+4)))), r17=Im(G(n+4)-conj(G(N/2-(
        n+4))))
444                                 // yr0=Re(G(0)), yr1=Im(G(0))
      fr26=r22*r11; fr23=r3+r5, fr19=r3-r5;;                // r26=s(n+5)*Re(G(n+5)-conj(G(N/2-(n+5))))
446                                 // r23=Im(G(n+5)+conj(G(N/2-(n+5)))), r19=Im(G(n+5)-conj(G(N/2-(
        n+5))))
      fr25=r21*r8;;                           // r25=cos(x)*Im(G(n)-conj(G(N/2-n)))
448   fr27=r23*r10; l[j12+=2]=xr13:12;;               // r27=cos(x)*Im(G(n)-conj(G(N/2-n)))
                                    // store F(n+2+N/8)
450   yfr0=r1+r0; yr1=lshift r1 by -32; q[j14+=-4]=yr31:28;;       // yr0=Re(G(0))+Im(G(0)), yr1=0=Im(F
        (0))
                                    // store F(N/2-(n+2)), F(N/2-(n+3))
452   yfr0=r0+r0; q[j13+=-4]=xr31:28;;                  // yr0=Re(F(0))
                                    // store F(N/2-(n+2+N/8)), F(N/2-(n+3+N/8))
454   fr24=r24+r25; fr25=r21*r9; l[j10+=2]=yr13:12;;          // r24=Re(-i*exp(2*pi*i*x)(G(n+4)-conj(G
        (N/2-(n+4)))))
                                    // r25=s(n+4)*Im(G(n+4)-conj(G(N/2-(n+4)))), store F(n+2)
456   fr26=r26+r27; fr27=r23*r11; l[j10+=2]=yr15:14;;          // r26=Re(-i*exp(2*pi*i*x)(G(n+5)-conj(G
        (N/2-(n+5)))))
```

```
                                    // r27=s(n+5)*Im(G(n+5)-conj(G(N/2-(n+5)))), store F(n+3)
458 fr13=r20*r8; fr12=r16+r24, fr30=r16-r24; l[j12+=2]=xr15:14;;  // r13=c(n+4)*Re(G(n+4)-conj(G(N
    /2-(n+4))))
                                    // r12=Re(F(n+4)), r30=Re(F(N/2-(n+4))), store F(n+3+N/8)
460 fr15=r22*r10; fr14=r18+r26, fr28=r18-r26; l[j31+j7]=yr1:0;;   // r15=c(n+5)*Re(G(n+5)-conj(G(N
    /2-(n+5))))
                                    // r14=Re(F(n+5)), r28=Re(F(N/2-(n+5)))
462                                 // store F(0)
    fr13=r25-r13; l[j7+j17]=k11:10;;                    // r13=Im(-i*exp(2*pi*i*x)(G(n+4)-conj(G(N/2-(
    n+4)))))
464                                 // store F(N/4)
    fr15=r27-r15;;                          // r15=Im(-i*exp(2*pi*i*x)(G(n+5)-conj(G(N/2-(n+5)))))
466
    fr13=r13+r17, fr31=r13-r17;;                    // r13=Im(F(n+4)), r31=Im(F(N/2-(n+4)))
468 fr15=r15+r19, fr29=r15-r19; l[j12+=2]=xr13:12;;         // r15=Im(F(n+5)), r29=Im(F(N/2-(n+5))),
     store F(n+4+N/8)
    q[j14+=-4]=yr31:28;;                        // store F(N/2-(n+4)), F(N/2-(n+5))
470 q[j13+=-4]=xr31:28;;                        // store F(N/2-(n+4+N/8)), F(N/2-(n+5+N/8))

472 l[j10+=2]=yr13:12;;                         // store F(n+4)
    l[j10+=2]=yr15:14;;                         // store F(n+5)
474 l[j12+=2]=xr15:14;;                         // store F(n+5+N/8)

476 //****************************** Epilogue *********************************
  _FFTEpilogue:
478
    mPOPQ(yR27:24)
480 mPOPQ(yR31:28)
    mPOPQ(xR27:24)
482   mPOPQ(xR31:28)
    mPOPQ(j19:16)
484 mRETURN

486 //******************** End Label For Statistical Profiling *****************
  _FFT32.end:
```

Listing 8: dspCode/fft_flp32.asm

```
1 /*  fft32.asm

3  Prelim rev. October 19, 2003 - BL
     Rev. 1.0 - added real inputs case - PM
5
   This is assembly routine for the Complex radix-2 C-callable FFT on TigerSHARC
7  family of DSPs.

9  I. Description of Calling.
```

```
11    1. Inputs:
        j4 -> input (ping-pong buffer 1)
13      j5 -> ping-pong buffer 1
            j6 -> ping-pong buffer 2
15      j7 -> output
        j27+0x18 -> N = Number of points
17      j27+0x19 -> REAL or COMPLEX

19    2. C-Calling Example:
        fft32(&(input), &(ping_pong_buffer1), &(ping_pong_buffer2), &(output), N, COMPLEX);
21
      3. Limitations:
23      a. All buffers must be aligned on memory boundary which is a multiple of 4.
        b. N must be between 32 and MAX_FFT_SIZE.
25      c. If memory space savings are required and input does not have to be
           preserved, ping_pong_buffer1 can be the same buffer as input.
27      d. If memory space savings are required, output can be the same buffer
           as ping_pong_buffer2 if the number of FFT stages is even (i.e.
29         Log2(N) is even) or the same as ping_pong_buffer1 if the number of
           FFT stages is odd (i.e. Log2(N) is odd).
31
      4. MAX_FFT_SIZE can be selected via #define. Larger values allow for more choices
33       of N, but its twiddles will occupy more memory.
      5. This C - callable function can process up to 64K blocks of data on TS201
35       (16K blocks on TS101) because C environment itself necessitates memory.
         Therefore, if more input points are necessary, assembly language development
37       may become a must. On TS201, a block of memory is 128K words long, so
         maximum N is 128K real points or 64K complex points. TS101 contains
39       only 2 blocks of data memory of 64K words and 4 buffers must be
         accommodated. Therefore, maximum N is 32K real words or 16K complex words.
41

43  II. Description of the FFT algorithm.

45    1. The input data is treated as complex interleaved N-point.
      2. Due to re-ordering, no stage can be done in-place.
47    3. The bit reversal and the first two stages are combined into
         a single loop. This loop takes data from input and stores it
49       in the ping-pong buffer1.
      4. Each subsequent stage ping-pongs the data between the two ping-pong
51       buffers. The last stage uses FFT output buffer for its output.
      5. Although the FFT is designed to be called with any point size
53       N <= MAX_FFT_SIZE by subsampling the twiddle factors, for ADSP-TS20x
         processors, the best cycle optimization is achieved when MAX_FFT_SIZE=N.
55       For ADSP-TS101 all choices of MAX_FFT_SIZE are equally optimal.

57
```

```
   III. Description of the REAL FFT algorithm.

     1. The input data is treated as complex interleaved N/2-point. The N/2 point complex
        FFT will be computed first. Thus, N is halved, now number of points = N/2.

     2. Details and source code of the N/2 point complex FFT are in II above.

     3. Real re-combine:
        Here the complex N/2-point FFT computed in the previous steps is recombined to
        produce the N-point real FFT. If G is the complex FFT and F is the real FFT,
        the formula for F is given by:

        F(n) = 0.5*(G(n)+conj(G(N/2-n)))-0.5*i*exp(-2*pi*i*n/N)*(G(n)-conj(G(N/2-n))).

        From this the following can be derived:

        conj(F(N/2-n)) = 0.5*(G(n)+conj(G(N/2-n)))+0.5*i*exp(-2*pi*i*n/N)*(G(n)-conj(G(N/2-n))).

        Thus, this can be computed in (n,N/2-n) pairs, as follows (dropping factor of 2):

        G(n) ------------------------------------>------------------------------>------> F(n)
                                                \ +/                          \ +/
                                                 \/                            \/
                                                 /\                            /\
                          conj                  / -\  exp(-2*pi*i*n)*i        / -\   conj
        G(N/2-n) -------> conj(G(N/2-n))------->------------------------------>------> F(N/2-n)

        This is very efficient on the TigerSHARC architecture due to the add/subtract
        instruction.


   IV. For all additional details regarding this algorithm and code, see EE-218
        application note, available from the ADI web site.

  */
//********************************** Includes ********************************

#include "FFTDef.h"
 #include "defts201.h"

 //************************* Externs ***********************************

 .extern _twiddles;

 //****************************** FFT Routine ******************************
.section program;
 .global _IFFT32;

```

```
_IFFT32:

//********************************* Prologue **********************************

    mENTER
    mPUSHQ(j19:16)
      mPUSHQ(xR31:28)
    mPUSHQ(xR27:24)
    mPUSHQ(yR31:28)
    mPUSHQ(yR27:24)

//********************************** Setup ************************************
    j17 = [j27 + 0x18];;                    //j17 = N
    j11 = [j27 + 0x19];;                    // j11=COMPLEX or REAL, off the stack

    comp(j11,COMPLEX);;                      // Complex or Real?
    if jeq, jump _FFTStages1and2;;
    j17=ashiftr j17;;                        // if Real, half N
//*****************************************************************************
_FFTStages1and2:

    j11 = j31 + j17;;                                  // j11=N

    xr3=j11; k7=k31+_twiddles;;
    k1=j11; j8=lshiftr j11;;                            // k1=N, j8=N/2
    j9=lshiftr j8; xr0=MAX_FFT_SIZE; xr3=LD0 r3;;              // j9=N/4, compute the twiddle
      stride
    k8=lshiftr k1; xr0=LD0 r0; xr1=j11;;
    k8=lshiftr k8; xr1=LD0 r1; xr2=(31-3);;                   // k8=N/4, Compute Stages-3
    k0=j4; k10=lshiftr k8; xr1=r1-r0; xr0=lshift r0 by -32;;         // k0->input, xr1=bit
      difference between MAX and N
    k10=lshiftr k10; xr0=bset r0 by r1; xr30=r2-r3;;             // k10=N/16, xr30=Stages-3
    k10=k10-1; xr0=lshift r0 by 2; LC1=xr30;;                 // k10=N/16-1, LC1=Stages-3
    k9=xr0; k4=k31+(MAX_FFT_SIZE/4-1);;
    k4=not k4; j10=lshiftr j9;;                         // initial twiddles pointer mask, j10=N/8

//***************** Bit Reverse and Stages 1 & 2 ****************************

    k5=lshiftr k1;;                                 // k5=N/2
    j0=j31+j6; k6=k6-k6;;                             // j0->ping_pong_buffer2
    j1=j0+j9; LC0=k10;;                               // j1->ping_pong_buffer2+N/4, LC0=N/16-1
    j2=j1+j9; k1=k0+k5;;                              // j2->ping_pong_buffer2+N/2, k1->input+N/2
    j3=j2+j9; k2=k1+k5;;                              // j3->ping_pong_buffer2+3N/4, k2->input+N
    j12=j3+j9; k3=k2+k5;;                             // j12->ping_pong_buffer2+N, k3->input+3N/2
    j13=j12+j9; k5=lshiftr k5;;                           // j13->ping_pong_buffer2+5N/4, k5=N/4
    r15=0x32107654;; //Fan Ren modified for IFFT @2100328
    j14=j13+j9; r1:0=q[k0+k6];;                         // j14->ping_pong_buffer2+3N/2
    r1:0=permute(r1:0,r15);; //Fan Ren modified for IFFT @2100328
```

```
    j15=j14+j9; r3:2=q[k2+k6];;                          // j15->ping_pong_buffer2+7N/4
153 r3:2=permute(r3:2,r15);; //Fan Ren modified for IFFT @2100328


155
    r5:4=q[k1+k6];;
157 r5:4=permute(r5:4,r15);; //Fan Ren modified for IFFT @2100328
    r7:6=q[k3+k6];;
159 r7:6=permute(r7:6,r15);; //Fan Ren modified for IFFT @2100328

161 k6=k6+k5 (br); fr0=r0+r2, fr20=r0-r2;;
    r9:8=q[k0+k6]; fr2=r1+r3, fr29=r1-r3;;
163 r9:8=permute(r9:8,r15);; //Fan Ren modified for IFFT @2100328
    r11:10=q[k2+k6]; fr4=r4+r6, fr21=r4-r6;;
165 r11:10=permute(r11:10,r15);; //Fan Ren modified for IFFT @2100328
    r13:12=q[k1+k6]; fr5=r5+r7, fr28=r5-r7;;
167 r13:12=permute(r13:12,r15);; //Fan Ren modified for IFFT @2100328

169 r16=0x32107654;; //Fan Ren modified for IFFT @2100328
    r15:14=q[k3+k6]; fr18=r8+r10, fr22=r8-r10;;
171 r15:14=permute(r15:14,r16);; //Fan Ren modified for IFFT @2100328
    k6=k6+k5 (br); fr19=r9+r11, fr31=r9-r11;;
173 fr26=r12+r14, fr23=r12-r14;;
    fr27=r13+r15, fr30=r13-r15;;
175
    fr20=r20+r28, fr28=r20-r28;;
177 fr29=r29+r21, fr21=r29-r21;;
    fr22=r22+r30, fr30=r22-r30;;
179 fr31=r31+r23, fr23=r31-r23;;

181 .align_code 4;
  _Stages1and2Loop:
183   r15=0x32107654;; //Fan Ren modified for IFFT @2100328
      r1:0=q[k0+k6]; q[j2+=4]=yr23:20; fr16=r0+r4, fr24=r0-r4;;
185   r1:0=permute(r1:0,r15);; //Fan Ren modified for IFFT @2100328
      r3:2=q[k2+k6]; q[j3+=4]=xr23:20; fr17=r2+r5, fr25=r2-r5;;
187   r3:2=permute(r3:2,r15);; //Fan Ren modified for IFFT @2100328
      r5:4=q[k1+k6]; q[j14+=4]=yr31:28; fr18=r18+r26, fr26=r18-r26;;
189   r5:4=permute(r5:4,r15);; //Fan Ren modified for IFFT @2100328
      r7:6=q[k3+k6]; q[j15+=4]=xr31:28; fr19=r19+r27, fr27=r19-r27;;
191   r7:6=permute(r7:6,r15);; //Fan Ren modified for IFFT @2100328

193   k6=k6+k5 (br); q[j0+=4]=yr19:16; fr0=r0+r2, fr20=r0-r2;;
      r9:8=q[k0+k6]; q[j1+=4]=xr19:16; fr2=r1+r3, fr29=r1-r3;;
195   r9:8=permute(r9:8,r15);; //Fan Ren modified for IFFT @2100328
      r11:10=q[k2+k6]; q[j12+=4]=yr27:24; fr4=r4+r6, fr21=r4-r6;;
197   r11:10=permute(r11:10,r15);; //Fan Ren modified for IFFT @2100328
      r13:12=q[k1+k6]; q[j13+=4]=xr27:24; fr5=r5+r7, fr28=r5-r7;;
199   r13:12=permute(r13:12,r15);; //Fan Ren modified for IFFT @2100328
```

```
201     r16=0x32107654;; //Fan Ren modified for IFFT @2100328
        r15:14=q[k3+k6]; fr18=r8+r10, fr22=r8-r10;;
203     r15:14=permute(r15:14,r16);; //Fan Ren modified for IFFT @2100328
        k6=k6+k5 (br); fr19=r9+r11, fr31=r9-r11;;
205     fr26=r12+r14, fr23=r12-r14;;
        fr27=r13+r15, fr30=r13-r15;;
207
        fr20=r20+r28, fr28=r20-r28;;
209     fr29=r29+r21, fr21=r29-r21;;
        fr22=r22+r30, fr30=r22-r30;;
211 .align_code 4;
        if NLC0E, jump _Stages1and2Loop;
213     fr31=r31+r23, fr23=r31-r23;;

215   q[j2+=4]=yr23:20; fr16=r0+r4, fr24=r0-r4;;
      q[j3+=4]=xr23:20; fr17=r2+r5, fr25=r2-r5;;
217   q[j14+=4]=yr31:28; fr18=r18+r26, fr26=r18-r26;;
      q[j15+=4]=xr31:28; fr19=r19+r27, fr27=r19-r27;;
219
      q[j0+=4]=yr19:16;;
221   q[j1+=4]=xr19:16;;
      q[j12+=4]=yr27:24;;
223   q[j13+=4]=xr27:24;;

225 //************************ Stages 3 to Log2(N)-1  ******************************

227   j0=j31+j6; k5=k31+0;;

229 .align_code 4;
    _StageLoop:
231     yr3:0=q[j0+=4];    k3=k5 and k4;;                                    // F1,     K1
        xr3:0=q[j0+=4];    r5:4=l[k7+k3];;                                  // F2,     K2
233     LC0=k10;           k5=k5+k9;;                                       //         K3,
          M1

235     yr11:8=q[j0+=4];   k3=k5 and k4;     fr6=r2*r4;;                        // F1+,
        K1+
        xr11:8=q[j0+=4];   r13:12=l[k7+k3]; fr7=r3*r5;;                      // F2+,    K2+,
          M2
237                                      fr14=r2*r5;;                       //
         M3
        j1=j31+j5;         k5=k5+k9;         fr6=r10*r12;   fr16=r6-r7;;     //         K3+,
          M1+,    A1
239
        yr23:20=q[j0+=4]; k3=k5 and k4;     fr15=r3*r4;;                     // F1++,   K1
        ++,   M4
241     xr23:20=q[j0+=4]; r5:4=l[k7+k3];    fr7=r11*r13;;                    // F2++,   K2
```

```
    ++,  M2+
                                        fr14=r10*r13; fr17=r14+r15;;                    //
       M3+,   A2
243 j2=j1+j11;          k5=k5+k9;         fr6=r22*r4;   fr18=r6-r7;;                    //         K3
    ++,  M1++,  A1+

245 yr31:28=q[j0+=4]; k3=k5 and k4;    fr15=r11*r12; fr24=r0+r16, fr26=r0-r16;;      // F1+++, K1
    +++, M4+,   A3
    xr31:28=q[j0+=4]; r13:12=l[k7+k3]; fr7=r23*r5;   fr25=r1+r17, fr27=r1-r17;;      // F2+++, K2
    +++, M2++,  A4
247     q[j1+=4]=r25:24;                   fr14=r22*r5;  fr19=r14+r15;;                  // S1,
          M3++,  A2+


249
  .align_code 4;
251 _BflyLoop:
      q[j2+=4]=r27:26;  k5=k5+k9;         fr6=r30*r12;  fr16=r6-r7;;                   // S2----,K3
    -,   M1-,   A1--
253
      yr3:0=q[j0+=4];   k3=k5 and k4;    fr15=r23*r4;  fr24=r8+r18,  fr26=r8-r18;;    // F1,    K1
    ,    M4--,  A3---
255   xr3:0=q[j0+=4];   r5:4=l[k7+k3];   fr7=r31*r13;  fr25=r9+r19,  fr27=r9-r19;;    // F2,    K2
    ,    M2-,   A4---
      q[j1+=4]=r25:24;                   fr14=r30*r13; fr17=r14+r15;;                  // S1---,
         M3-,   A2--
257   q[j2+=4]=r27:26;  k5=k5+k9;         fr6=r2*r4;    fr18=r6-r7;;                   // S2---, K3
    ,    M1,    A1-

259   yr11:8=q[j0+=4];  k3=k5 and k4;    fr15=r31*r12; fr24=r20+r16, fr26=r20-r16;;   // F1+,   K1
    +,   M4-,   A3--
      xr11:8=q[j0+=4];  r13:12=l[k7+k3]; fr7=r3*r5;    fr25=r21+r17, fr27=r21-r17;;   // F2+,   K2
    +,   M2,    A4--
261        q[j1+=4]=r25:24;                   fr14=r2*r5;   fr19=r14+r15;;                 // S1
    --,          M3,    A2-
      q[j2+=4]=r27:26;  k5=k5+k9;         fr6=r10*r12;  fr16=r6-r7;;                   // S2--,  K3
    +,   M1+,   A1
263
      yr23:20=q[j0+=4]; k3=k5 and k4;    fr15=r3*r4;   fr24=r28+r18, fr26=r28-r18;;   // F1++,  K1
    ++,  M4,    A3-
265   xr23:20=q[j0+=4]; r5:4=l[k7+k3];   fr7=r11*r13;  fr25=r29+r19, fr27=r29-r19;;   // F2++,  K2
    ++,  M2+,   A4-
      q[j1+=4]=r25:24;                   fr14=r10*r13; fr17=r14+r15;;                  // S1-,
         M3+,   A2
267   q[j2+=4]=r27:26;  k5=k5+k9;         fr6=r22*r4;   fr18=r6-r7;;                   // S2-,   K3
    ++,  M1++,  A1+

269   yr31:28=q[j0+=4]; k3=k5 and k4;    fr15=r11*r12; fr24=r0+r16,  fr26=r0-r16;;    // F1+++, K1
    +++, M4+,   A3
```

```
        xr31:28=q[j0+=4]; r13:12=l[k7+k3]; fr7=r23*r5;    fr25=r1+r17,   fr27=r1-r17;;   // F2+++, K2
      +++, M2++,  A4
.align_code 4;
            if NLC0E, jump _BflyLoop;
            q[j1+=4]=r25:24; fr14=r22*r5;  fr19=r14+r15;;                              // S1,
      M3++,  A2+

    q[j2+=4]=r27:26;                    fr6=r30*r12;  fr16=r6-r7;;                     // S2----,
          M1-, A1--

    j0=j31+j5;                          fr15=r23*r4;  fr24=r8+r18,  fr26=r8-r18;;      //
          M4--,  A3---, swap ping-pong pointers
      j5=j31+j6;                        fr7=r31*r13;  fr25=r9+r19,  fr27=r9-r19;;      //
          M2-,   A4---
    q[j1+=4]=r25:24;                    fr14=r30*r13; fr17=r14+r15;;                   // S1---,
          M3-,   A2--
      q[j2+=4]=r27:26;                                  fr18=r6-r7;;                   // S2---,
                  A1-

      j6=j31+j0;                        fr15=r31*r12; fr24=r20+r16, fr26=r20-r16;;     //
          M4-,   A3--
                                                      fr25=r21+r17, fr27=r21-r17;;     //
                  A4--
          q[j1+=4]=r25:24;                            fr19=r14+r15;;                   // S1--,
                  A2-
    q[j2+=4]=r27:26;                    fr24=r28+r18, fr22=r28-r18;;              // S2--
           A3-

    j0=j31+j6;                          fr25=r29+r19, fr23=r29-r19;;     //                    A4
      -
      q[j1+=4]=r25:24; k5=k31+0;;                                                      //
       S1-
.align_code 4;
      if NLC1E, jump _StageLoop;
    q[j2+=4]=r23:22; k4=ashiftr k4;;                                                   // S2-,
      shift the mask

//****************************** Last stage ******************************
    k9 = ashiftr k9;;//in this manner any MAX_FFT_SIZE can be used

    yr3:0=q[j0+=4]; yr5:4 = l[k7+=k9];;                                  // F1,

    xr3:0=q[j0+=4];    xr5:4=l[k7+=k9];;                                           // F2,     K2
    j1=j31+j7;         fr6=r2*r4; LC0=k10;;                              //              M1


    yr11:8=q[j0+=4];  yr13:12=l[k7+=k9];;
            // F1+
```

```
303   xr11:8=q[j0+=4];  xr13:12=l[k7+=k9];  fr7=r3*r5;;                                    // F2+,    K2
        +,    M2
      j2=j1+j11;                        fr14=r2*r5;;                                  //
      M3
305                                       fr6=r10*r12;   fr16=r6-r7;;                   //
          M1+,    A1

307   yr23:20=q[j0+=4]; yr5:4=l[k7+=k9];   fr15=r3*r4;;                                  // F1++,
          M4
      xr23:20=q[j0+=4]; xr5:4=l[k7+=k9];    fr7=r11*r13;;                                 // F2++,   K2
        ++,   M2+
309                                       fr14=r10*r13; fr17=r14+r15;;                  //
          M3+,    A2
                                          fr6=r22*r4;    fr18=r6-r7;;                   //
          M1++,   A1+
311
      yr31:28=q[j0+=4]; yr13:12=l[k7+=k9];fr15=r11*r12; fr24=r0+r16, fr26=r0-r16;;      // F1+++,
          M4+,    A3
313   xr31:28=q[j0+=4]; xr13:12=l[k7+=k9]; fr7=r23*r5;    fr25=r1+r17, fr27=r1-r17;;     // F2+++, K2
        +++, M2++,   A4
      r11=0.015625;;           //Fan Ren modified for IFFT @2100328
315   fr25=r25*r11;r10=0x32107654;;    //Fan Ren modified for IFFT @2100328
      fr24=r24*r11;;           //Fan Ren modified for IFFT @2100328
317   r25:24=permute(r25:24,r10);;  //Fan Ren modified for IFFT @2100328
        q[j1+=4]=r25:24;  fr14=r22*r5;  fr19=r14+r15;;                    // S1,           M3++,   A2+
319



321


323 .align_code 4;
    _BflyLastLoop:
325     r11=0.015625;;           //Fan Ren modified for IFFT @2100328
        fr27=r27*r11;r10=0x32107654;;    //Fan Ren modified for IFFT @2100328
327     fr26=r26*r11;;           //Fan Ren modified for IFFT @2100328
        r27:26=permute(r27:26,r10);;  //Fan Ren modified for IFFT @2100328
329     q[j2+=4]=r27:26;                     fr6=r30*r12;   fr16=r6-r7;;                  // S2----,
            M1-,    A1--

331     yr3:0=q[j0+=4];   yr5:4=l[k7+=k9];   fr15=r23*r4;   fr24=r8+r18,   fr26=r8-r18;;    // F1,
            M4--,   A3---
        xr3:0=q[j0+=4];   xr5:4=l[k7+=k9];   fr7=r31*r13;   fr25=r9+r19,   fr27=r9-r19;;    // F2,     K2
        ,     M2-,    A4---
333     fr25=r25*r11;;             //Fan Ren modified for IFFT @2100328
        fr24=r24*r11;;             //Fan Ren modified for IFFT @2100328
335     r25:24=permute(r25:24,r10);;  //Fan Ren modified for IFFT @2100328
        q[j1+=4]=r25:24;                     fr14=r30*r13; fr17=r14+r15;;                  // S1---,
            M3-,    A2--
337     fr27=r27*r11;;             //Fan Ren modified for IFFT @2100328
```

```
    fr26=r26*r11;;              //Fan Ren modified for IFFT @2100328
339 r27:26=permute(r27:26,r10);;  //Fan Ren modified for IFFT @2100328
    q[j2+=4]=r27:26;                    fr6=r2*r4;    fr18=r6-r7;;                        // S2---,
        M1,    A1-
341
    yr11:8=q[j0+=4];  yr13:12=l[k7+=k9];fr15=r31*r12; fr24=r20+r16, fr26=r20-r16;;     // F1+,
        M4-,   A3--
343 xr11:8=q[j0+=4];  xr13:12=l[k7+=k9];fr7=r3*r5;    fr25=r21+r17, fr27=r21-r17;;     // F2+,   K2
    +,   M2,    A4--
    r31=0.015625;;          //Fan Ren modified for IFFT @2100328
345 fr25=r25*r31;r30=0x32107654;;   //Fan Ren modified for IFFT @2100328
    fr24=r24*r31;;              //Fan Ren modified for IFFT @2100328
347 r25:24=permute(r25:24,r30);;  //Fan Ren modified for IFFT @2100328
      q[j1+=4]=r25:24;                    fr14=r2*r5;   fr19=r14+r15;;                   // S1--,
          M3,    A2-
349 fr27=r27*r31;;              //Fan Ren modified for IFFT @2100328
    fr26=r26*r31;;              //Fan Ren modified for IFFT @2100328
351 r27:26=permute(r27:26,r30);;  //Fan Ren modified for IFFT @2100328
    q[j2+=4]=r27:26;                    fr6=r10*r12;  fr16=r6-r7;;                        // S2--,
        M1+,   A1
353
    yr23:20=q[j0+=4]; yr5:4=l[k7+=k9];  fr15=r3*r4;   fr24=r28+r18, fr26=r28-r18;;     // F1++,
        M4,    A3-
355 xr23:20=q[j0+=4]; xr5:4=l[k7+=k9];  fr7=r11*r13;  fr25=r29+r19, fr27=r29-r19;;     // F2++,  K2
    ++,  M2+,   A4-
    fr25=r25*r31;;              //Fan Ren modified for IFFT @2100328
357 fr24=r24*r31;;              //Fan Ren modified for IFFT @2100328
    r25:24=permute(r25:24,r30);;  //Fan Ren modified for IFFT @2100328
359 q[j1+=4]=r25:24;                    fr14=r10*r13; fr17=r14+r15;;                     // S1-,
        M3+,   A2
    fr27=r27*r31;;              //Fan Ren modified for IFFT @2100328
361 fr26=r26*r31;;              //Fan Ren modified for IFFT @2100328
    r27:26=permute(r27:26,r30);;  //Fan Ren modified for IFFT @2100328
363 q[j2+=4]=r27:26;                    fr6=r22*r4;   fr18=r6-r7;;                        // S2-,
        M1++,  A1+

365 yr31:28=q[j0+=4];yr13:12=l[k7+=k9]; fr15=r11*r12; fr24=r0+r16,  fr26=r0-r16;;      // F1+++,
        M4+,   A3
    xr31:28=q[j0+=4];xr13:12=l[k7+=k9]; fr7=r23*r5;   fr25=r1+r17,  fr27=r1-r17;;      // F2+++, K2
    +++, M2++,  A4
367 r11=0.015625;;          //Fan Ren modified for IFFT @2100328
    fr25=r25*r11;r10=0x32107654;;   //Fan Ren modified for IFFT @2100328
369 fr24=r24*r11;;              //Fan Ren modified for IFFT @2100328
    r25:24=permute(r25:24,r10);;  //Fan Ren modified for IFFT @2100328
371 .align_code 4;
      if NLC0E, jump _BflyLastLoop;
373   q[j1+=4]=r25:24;  fr14=r22*r5;  fr19=r14+r15;;                                  // S1,          M3
    ++,  A2+
```

```
375 fr27=r27*r11;;              //Fan Ren modified for IFFT @2100328
    fr26=r26*r11;;              //Fan Ren modified for IFFT @2100328
377 r27:26=permute(r27:26,r10);;  //Fan Ren modified for IFFT @2100328
    q[j2+=4]=r27:26;                    fr6=r30*r12;  fr16=r6-r7;;                          // S2----,
        M1-, A1--
379
                                        fr15=r23*r4;  fr24=r8+r18,  fr26=r8-r18;;           //
        M4--,  A3---
381                                     fr7=r31*r13;  fr25=r9+r19,  fr27=r9-r19;;           //
        M2-,   A4---
    fr25=r25*r11;;              //Fan Ren modified for IFFT @2100328
383 fr24=r24*r11;;              //Fan Ren modified for IFFT @2100328
    r25:24=permute(r25:24,r10);;  //Fan Ren modified for IFFT @2100328
385 q[j1+=4]=r25:24;                    fr14=r30*r13; fr17=r14+r15;;                        // S1---,
        M3-,   A2--
    fr27=r27*r11;;              //Fan Ren modified for IFFT @2100328
387 fr26=r26*r11;;              //Fan Ren modified for IFFT @2100328
    r27:26=permute(r27:26,r10);;  //Fan Ren modified for IFFT @2100328
389 q[j2+=4]=r27:26;                                fr18=r6-r7;;                            // S2---,
              A1-
391                                     fr15=r31*r12; fr24=r20+r16, fr26=r20-r16;;          //
        M4-,   A3--
                                                    fr25=r21+r17, fr27=r21-r17;;          //
              A4--
393 fr25=r25*r11;;              //Fan Ren modified for IFFT @2100328
    fr24=r24*r11;;              //Fan Ren modified for IFFT @2100328
395 r25:24=permute(r25:24,r10);;  //Fan Ren modified for IFFT @2100328
      q[j1+=4]=r25:24;                              fr19=r14+r15;;                        // S1--,
            A2-
397 fr27=r27*r11;;              //Fan Ren modified for IFFT @2100328
    fr26=r26*r11;;              //Fan Ren modified for IFFT @2100328
399 r27:26=permute(r27:26,r10);;  //Fan Ren modified for IFFT @2100328
    q[j2+=4]=r27:26;;                                                                       // S2--
401
                                                    fr24=r28+r18, fr26=r28-r18;;          //
              A3-
403                                                 fr25=r29+r19, fr27=r29-r19;;          //
              A4-
    fr25=r25*r11;;              //Fan Ren modified for IFFT @2100328
405 fr24=r24*r11;;              //Fan Ren modified for IFFT @2100328
    r25:24=permute(r25:24,r10);;  //Fan Ren modified for IFFT @2100328
407 q[j1+=4]=r25:24;;                                                                       // S1-
    fr27=r27*r11;;              //Fan Ren modified for IFFT @2100328
409 fr26=r26*r11;;              //Fan Ren modified for IFFT @2100328
    r27:26=permute(r27:26,r10);;  //Fan Ren modified for IFFT @2100328
411 q[j2+=4]=r27:26;;                                                                       // S2-
```

```
413   j11=[j27+0x19];;                                  // j11=COMPLEX or REAL, off the stack
      comp(j11,COMPLEX);;                               // Complex or Real?
415 .align_code 4;
      if jeq, jump _FFTEpilogue;;                       // If Complex, done
417
    //****************************** Real re-combine ******************************
419
    //j17=N/2, j7=output
421
      k8=k31+_twiddles; j0=j31+j7;;                       // k8->twiddles, j0->internal buffer
423   k9=ashiftr k9; j10=j31+j7;;                       // k9=twiddle stride, j10->internal buffer

425   j14=j17+j17;;                                   // j14=N (N/2 complex values)
      j14=j14-4;;                                   // j14=N-4 real=N/2-2 complex
427   j1=j0+j14;;                                   // j1->internal buffer+(N/2-2)
      j14=j10+j14;;                                 // j14->internal buffer+(N/2-2)
429
      j29 = ashiftr j17;;                             // j29=N/4
431   k15=k31+MAX_FFT_SIZE/4; j30=ashiftr j29;;           // k15=N/4*twiddle_stride, j30=N/8

433   j30 = ashiftr j30;;                             // N/16

435   k8=k8+k9; r0=l[j7+j17];;                          // k8->twiddles+1, get G(N/4)
      j0=j0+2; k12=k8+k15;;                           // j0->internal buffer+1, k12->twiddles+N/8+1
437   LC0=j30; fr0=r0+r0; j2=j0+j29;;                   // LC0=N/16, compute F(N/4)=2*conj(G(N/4)), j2->
        internal buffer+1+N/8
      j3=j1-j29;;                                   // j3->internal buffer+3N/8-2
439   xfr0=-r0; j10=j10+2; k10=yr0;;                    // j10->internal buffer+1, k10=Im(F(N/4))
      j12=j10+j29;;                                 // j12->internal buffer+N/8+1
441   if LC0E; j13=j14-j29; k11=xr0;;                   // LC0=N/16-1, j13->internal buffer+3N/8-2, k11=
        Re(F(N/4))

443   yr3:0=DAB q[j0+=4];;                            // Prime the DAB
      xr3:0=DAB q[j2+=4];;                            // Prime the DAB
445
      yr3:0=DAB q[j0+=4];;                            // yr0=Re(G(n)), yr1=Im(G(n)), yr2=Re(G(n+1)), yr3=Im(
        G(n+1))
447   xr3:0=DAB q[j2+=4];;                            // xr0=Re(G(n+N/8)), xr1=Im(G(n+N/8))
                                    // xr2=Re(G(n+1+N/8)), xr3=Im(G(n+1+N/8))
449   yr7:4=q[j1+=-4]; xr9:8=l[k12+=k9];;               // yr4=Re(G(N/2-(n+1))), yr5=Im(G(N/2-(n+1)))
                                    // yr6=Re(G(N/2-n)), yr7=Im(G(N/2-n))
451                                 // twiddles(n+N/8) - want to mult by sin(x)-icos(x)
      xr7:4=q[j3+=-4]; xr11:10=l[k12+=k9];;             // xr4=Re(G(N/2-(n+1+N/8))), xr5=Im(G(N/2-(n
        +1+N/8)))
453                                 // xr6=Re(G(N/2-(n+N/8))), xr7=Im(G(N/2-(n+N/8)))
                                    // twiddles(n+1+N/8)
455   if LC0E; fr16=r0+r6, fr20=r0-r6; yr9:8=l[k8+=k9];;        // LC0=N/16-2, r16=Re(G(n)+conj(G(N/2-
```

```
    n))), r20=Re(G(n)-conj(G(N/2-n)))
                                    // twiddles(n)
457 fr18=r2+r4, fr22=r2-r4; yr11:10=l[k8+=k9];;          // r18=Re(G(n+1)+conj(G(N/2-(n+1)))), r22=
    Re(G(n+1)-conj(G(N/2-(n+1))))
                                    // twiddles(n+1)
459 fr24=r20*r9; fr21=r1+r7, fr17=r1-r7;;              // r24=s(n)*Re(G(n)-conj(G(N/2-n)))
                                    // r17=Im(G(n)+conj(G(N/2-n))), r21=Im(G(n)-conj(G(N/2-n)))
461 fr26=r22*r11; fr23=r3+r5, fr19=r3-r5; xr3:0=DAB q[j2+=4];;    // r26=s(n+1)*Re(G(n+1)-conj(G(N
    /2-(n+1))))
                                    // r19=Im(G(n+1)+conj(G(N/2-(n+1)))), r23=Im(G(n+1)-conj(G(N/2-(
    n+1))))
463                                 // xr3:0=next G(n+2+N/8), G(n+3+N/8)
    fr25=r21*r8;  yr3:0=DAB q[j0+=4];;                  // r25=c(n)*Im(G(n)-conj(G(N/2-n))), yr3:0=
    next G(n+2), G(n+3)
465 fr27=r23*r10;;                              // r27=c(n+1)*Im(G(n+1)-conj(G(N/2-(n+1))))
    fr24=r24+r25; fr25=r21*r9; yr7:4=q[j1+=-4];;          // r24=Re(-i*exp(2*pi*i*n)(G(n)-conj(G(N
    /2-n))))
467                                 // r13=s(n)*Im(G(n)-conj(G(N/2-n)))
                                    // yr7:4=next G(N/2-(n+2)), G(N/2-(n+3))
469 fr26=r26+r27; fr27=r23*r11; xr7:4=q[j3+=-4];;         // r26=Re(-i*exp(2*pi*i*(n+1))(G(n+1)-conj
    (G(N/2-(n+1)))))
                                    // r27=s(n+1)*Im(G(n+1)-conj(G(N/2-(n+1))))
471                                 // xr7:4=next G(N/2-(n+2+N/8)), G(N/2-(n+3+N/8))
    fr13=r20*r8; fr12=r16+r24, fr30=r16-r24;;            // r13=c(n)*Re(G(n)-conj(G(N/2-n))), r12=Re(
    F(n)), r30=Re(F(N/2-n))
473 fr15=r22*r10; fr14=r18+r26, fr28=r18-r26;;            // r15=c(n+1)*Re(G(n+1)-conj(G(N/2-(n+1)))
    )
                                    // r14=Re(F(n+1)), r28=Re(F(N/2-(n+1)))
475 fr13=r25-r13; xr9:8=l[k12+=k9];;                  // r13=Im(-i*exp(2*pi*i*x)(G(n)-conj(G(N/2-n))))
    , next twiddles(n+2+N/8)
    fr15=r27-r15; xr11:10=l[k12+=k9];;                  // r15=Im(-i*exp(2*pi*i*x)(G(n+1)-conj(G(N/2-(
    n+1)))))
477                                 // next twiddles(n+3+N/8)
  .align_code 4;
479 _combine_stage:
    fr16=r0+r6, fr20=r0-r6; yr9:8=l[k8+=k9];;           // r16=Re(G(n+2)+conj(G(N/2-(n+2)))), r20=Re
    (G(n+2)-conj(G(N/2-(n+2))))
481                                 // next twiddles(n+2)
    fr18=r2+r4, fr22=r2-r4; yr11:10=l[k8+=k9];;         // r18=Re(G(n+3)+conj(G(N/2-(n+3)))), r22=
    Re(G(n+3)-conj(G(N/2-(n+3))))
483                                 // next twiddles(n+3)
    fr13=r13+r17, fr31=r13-r17;;                    // r13=Im(F(n)), r31=Im(F(N/2-n))
485 fr15=r15+r19, fr29=r15-r19; l[j12+=2]=xr13:12;;       // r15=Im(F(n+1)), r29=Im(F(N/2-(n+1))),
     store F(n+N/8)
    fr24=r20*r9; fr21=r1+r7, fr17=r1-r7; q[j14+=-4]=yr31:28;; // r24=s(n+2)*Re(G(n+2)-conj(G(N/2-(
    n+2))))
487                                 // r21=Im(G(n+2)+conj(G(N/2-(n+2)))), r17=Im(G(n+2)-conj(G(N/2-(
    n+2))))
```

```
                                    // store F(N/2-n), F(N/2-(n+1))
489     fr26=r22*r11; fr23=r3+r5, fr19=r3-r5; xr3:0=DAB q[j2+=4];;  // r26=s(n+3)*Re(G(n+3)-conj(G(N
        /2-(n+3))))
                                    // r23=Im(G(n+3)+conj(G(N/2-(n+3)))), r19=Im(G(n+3)-conj(G(N/2-(
        n+3))))
491                                 // xr3:0=next G(n+4+N/8), G(n+5+N/8)
        fr25=r21*r8; yr3:0=DAB q[j0+=4];;           // r25=c(n+2)*Im(G(n+2)-conj(G(N/2-(n+2))))
493                                 // yr3:0=next G(n+4), G(n+5)
        fr27=r23*r10; q[j13+=-4]=xr31:28;;            // r27=c(n+3)*Im(G(n+3)-conj(G(N/2-(n+3))))
495                                 // store F(N/2-(n+N/8)), F(N/2-(n+1+N/8))
        fr24=r24+r25; fr25=r21*r9; l[j10+=2]=yr13:12;;      // r24=Re(-i*exp(2*pi*i*x)(G(n+2)-conj(G
        (N/2-(n+2)))))
497                                 // r25=s(n+2)*Im(G(n+2)-conj(G(N/2-(n+2)))), store F(n)
        fr26=r26+r27; fr27=r23*r11; l[j10+=2]=yr15:14;;      // r26=Re(-i*exp(2*pi*i*x)(G(n+3)-conj(G
        (N/2-(n+3)))))
499                                 // r27=s(n+3)*Im(G(n+3)-conj(G(N/2-(n+3)))), store F(n+1)
        fr13=r20*r8; fr12=r16+r24, fr30=r16-r24; l[j12+=2]=xr15:14;;// r13=cos(n+2)*Re(G(n+2)-conj(G(N
        /2-(n+2))))
501                                 // r12=Re(F(n+2)), r30=Re(F(N/2-(n+2))), store F(n+1+N/8)
        fr15=r22*r10; fr14=r18+r26, fr28=r18-r26; xr7:4=q[j3+=-4];; // r15=cos(n+3)*Re(G(n+3)-conj(G(N
        /2-(n+3))))
503                                 // r14=Re(F(n+3)), r28=Re(F(N/2-(n+3)))
                                    // xr7:4=next G(N/2-(n+4+N/8)), G(N/2-(n+5+N/8))
505     fr13=r25-r13; xr9:8=l[k12+=k9]; yr7:4=q[j1+=-4];;      // r13=Im(-i*exp(2*pi*i*x)(G(n+2)-conj(G
        (N/2-(n+2)))))
                                    // next twiddles(n+4+N/8)
507                                 // yr7:4=next G(N/2-(n+4)), G(N/2-(n+5))
    .align_code 4;
509     if NLC0E, jump _combine_stage(P); fr15=r27-r15; xr11:10=l[k12+=k9];;// r15=Im(-i*exp(2*pi*i*x)
        (G(n+3)-conj(G(N/2-(n+3)))))
                                    // next twiddles(n+5+N/8)
511
      fr16=r0+r6, fr20=r0-r6; yr9:8=l[k8+=k9];;            // r16=Re(G(n+4)+conj(G(N/2-(n+4)))), r20=Re
        (G(n+4)-conj(G(N/2-(n+4))))
513                                 // next twiddles(n+4)
      fr18=r2+r4, fr22=r2-r4; yr11:10=l[k8+=k9];;            // r18=Re(G(n+5)+conj(G(N/2-(n+5)))), r22=
        Re(G(n+5)-conj(G(N/2-(n+5))))
515                                 // next twiddles(n+5)
      fr13=r13+r17, fr31=r13-r17;;                  // r13=Im(F(n+2)), r31=Im(F(N/2-(n+2)))
517   fr15=r15+r19, fr29=r15-r19;;                  // r15=Im(F(n+3)), r29=Im(F(N/2-(n+3)))
      fr24=r20*r9; fr21=r1+r7, fr17=r1-r7; yr1:0=l[j31+j7];;      // r24=s(n+4)*Re(G(n+4)-conj(G(N/2-(
        n+4))))
519                                 // r21=Im(G(n+4)+conj(G(N/2-(n+4)))), r17=Im(G(n+4)-conj(G(N/2-(
        n+4))))
                                    // yr0=Re(G(0)), yr1=Im(G(0))
521   fr26=r22*r11; fr23=r3+r5, fr19=r3-r5;;              // r26=s(n+5)*Re(G(n+5)-conj(G(N/2-(n+5))))
                                    // r23=Im(G(n+5)+conj(G(N/2-(n+5)))), r19=Im(G(n+5)-conj(G(N/2-(
        n+5))))
```

```
523   fr25=r21*r8;;                                  // r25=cos(x)*Im(G(n)-conj(G(N/2-n)))
      fr27=r23*r10; l[j12+=2]=xr13:12;;                  // r27=cos(x)*Im(G(n)-conj(G(N/2-n)))
525                                  // store F(n+2+N/8)
      yfr0=r1+r0; yr1=lshift r1 by -32; q[j14+=-4]=yr31:28;;       // yr0=Re(G(0))+Im(G(0)), yr1=0=Im(F
        (0))
527                                  // store F(N/2-(n+2)), F(N/2-(n+3))
      yfr0=r0+r0; q[j13+=-4]=xr31:28;;                   // yr0=Re(F(0))
529                                  // store F(N/2-(n+2+N/8)), F(N/2-(n+3+N/8))
      fr24=r24+r25; fr25=r21*r9; l[j10+=2]=yr13:12;;           // r24=Re(-i*exp(2*pi*i*x)(G(n+4)-conj(G
        (N/2-(n+4)))))
531                                  // r25=s(n+4)*Im(G(n+4)-conj(G(N/2-(n+4)))), store F(n+2)
      fr26=r26+r27; fr27=r23*r11; l[j10+=2]=yr15:14;;           // r26=Re(-i*exp(2*pi*i*x)(G(n+5)-conj(G
        (N/2-(n+5)))))
533                                  // r27=s(n+5)*Im(G(n+5)-conj(G(N/2-(n+5)))), store F(n+3)
      fr13=r20*r8; fr12=r16+r24, fr30=r16-r24; l[j12+=2]=xr15:14;;  // r13=c(n+4)*Re(G(n+4)-conj(G(N
        /2-(n+4))))
535                                  // r12=Re(F(n+4)), r30=Re(F(N/2-(n+4))), store F(n+3+N/8)
      fr15=r22*r10; fr14=r18+r26, fr28=r18-r26; l[j31+j7]=yr1:0;;   // r15=c(n+5)*Re(G(n+5)-conj(G(N
        /2-(n+5))))
537                                  // r14=Re(F(n+5)), r28=Re(F(N/2-(n+5)))
                                     // store F(0)
539   fr13=r25-r13; l[j7+j17]=k11:10;;                    // r13=Im(-i*exp(2*pi*i*x)(G(n+4)-conj(G(N/2-(
        n+4)))))
                                     // store F(N/2)
541   fr15=r27-r15;;                             // r15=Im(-i*exp(2*pi*i*x)(G(n+5)-conj(G(N/2-(n+5)))))

543   fr13=r13+r17, fr31=r13-r17;;                     // r13=Im(F(n+4)), r31=Im(F(N/2-(n+4)))
      fr15=r15+r19, fr29=r15-r19; l[j12+=2]=xr13:12;;           // r15=Im(F(n+5)), r29=Im(F(N/2-(n+5))),
         store F(n+4+N/8)
545   q[j14+=-4]=yr31:28;;                          // store F(N/2-(n+4)), F(N/2-(n+5))
      q[j13+=-4]=xr31:28;;                          // store F(N/2-(n+4+N/8)), F(N/2-(n+5+N/8))
547
      l[j10+=2]=yr13:12;;                           // store F(n+4)
549   l[j10+=2]=yr15:14;;                           // store F(n+5)
      l[j12+=2]=xr15:14;;                           // store F(n+5+N/8)
551
  //****************************** Epilogue *********************************
553 _FFTEpilogue:

555   mPOPQ(yR27:24)
      mPOPQ(yR31:28)
557   mPOPQ(xR27:24)
        mPOPQ(xR31:28)
559   mPOPQ(j19:16)
      mRETURN
561
  //******************** End Label For Statistical Profiling *****************
563 _IFFT32.end:
```

Listing 9: dspCode/ifft_flp32.asm

```
//*****************************************************************
//      Init routine for the Danube FFT
//      Init.c
//*****************************************************************

//************************ Includes ***********************************
#include <sysreg.h>
#include <builtins.h>
#include "defts201.h"

//************************ Externs ************************************

//********************* Function declares *****************************

//*****************************************************************
void init( void )
{
//---------------------------- Enable Cache ---------------------------------

   asm("CACMDALL = 0x00000000;;");//CACMD_EN;;");
//   asm("BTBEN;;");

//     __builtin_sysreg_write(__SQCTLST, SQCTL_NMOD);
//     __builtin_sysreg_write(__CACMDB, CACMD_EN);

}
//*****************************************************************
```

Listing 10: dspCode/init.c

```
//#define INTV_WORDS 4800
#define INTV_WORDS 12
#define SEED 34642

#define SYMBOLS_PER_INTV 12
//#define SYMBOL_SEP 67
#define SYMBOL_SEP 7
#define NEXT_POS(x) x=x+256
#define PREV_POS(x) x=x-256
//#define LAST_WORD 4733
#define LAST_WORD 5
#define LAST_BIT 47
```

```
unsigned long long interleaver(unsigned long long int);
unsigned long long deinterleaver(unsigned long long int input);
```

Listing 11: dspCode/interleaver_new.h

```
#include "interleaver_new.h"
#include "input.h"
#include <stdio.h>

unsigned long long int interleaver(unsigned long long int input){
  section ("data10b")
  static unsigned long long int BUFFER_PING[INTV_WORDS], BUFFER_PONG[INTV_WORDS];
  static unsigned long long int *OUT_PING = BUFFER_PING, *OUT_PONG = BUFFER_PONG, *temp;
  int i=0, j=0;
  static int output_index=0;
  static int ping_index=0;
  static int outputPos = 0;
  unsigned long long int output;
  unsigned long long int bit_ext;
  unsigned int pos_len = 0x00000001;
  //printf("input: %012llx\n",input);
  if (output_index==INTV_WORDS){
    // Reset the interleaver state
    output_index = 0;
    outputPos = 0;
    ping_index = 0;

    // Switch the input buffers
    temp = OUT_PING;
    OUT_PING = OUT_PONG;
    OUT_PONG = temp;
  }

  //Write the input bit to the input buffer.

  for (i=0;i<BITS_IN_SYMBOL;i++){
    // Interleave
    bit_ext = __builtin_fext2_ze(input,pos_len)<<(outputPos);
    //printf("%d,\n",ping_index*48+outputPos);
    OUT_PING[ping_index]|=bit_ext;

    NEXT_POS(pos_len);

    ping_index = (ping_index+SYMBOL_SEP);
    if (ping_index>=INTV_WORDS){
      ping_index-=INTV_WORDS;
      outputPos=(outputPos+1)%BITS_IN_SYMBOL;
```

```
    }
44
  }
46 //printf("\n");%

48 // Get the output from the output buffer.
  output = OUT_PONG[output_index];
50 OUT_PONG[output_index] = 0; // Re-initalise element for the interleaver.
  output_index++;
52 return output;
}
54


56 unsigned long long int deinterleaver(unsigned long long int input){
  section ("data4b")
58 static unsigned long long int INPUT_PING[INTV_WORDS], INPUT_PONG[INTV_WORDS], OUTPUT_PING[
    INTV_WORDS], OUTPUT_PONG[INTV_WORDS];
  static unsigned long long int *IN_PING = INPUT_PING, *IN_PONG = INPUT_PONG, *OUT_PING =
    OUTPUT_PING, *OUT_PONG = OUTPUT_PONG, *temp;
60 int i=0;
  static int ping_index=0;
62 static int input_pong_index=0;
  static int outputPos = 0; // Last position of interleaver. Simulated by MATLAB
64 static int output_ping_index = 0;
  unsigned long long int output=0;
66 unsigned long long int bit_ext;
  unsigned int pos_len = 0x0000001;
68

  if (ping_index==INTV_WORDS){
70   // Reset the interleaver state
    ping_index = 0;
72

    // Switch the input buffers
74   temp = IN_PING;
    IN_PING = IN_PONG;
76   IN_PONG = temp;
  }
78

  IN_PING[ping_index] = input;
80

  //Write the input bit to the input buffer.
82 for (i=0;i<BITS_IN_SYMBOL;i++){
    // Extract the deinterleave bit
84   bit_ext = (IN_PONG[input_pong_index]>>outputPos)&(0x1);
    output = __builtin_fdep2(output,bit_ext,pos_len);
86   NEXT_POS(pos_len);

88   // Reverse of the in%terleaver operation
```

```
      input_pong_index += SYMBOL_SEP;
90    if (input_pong_index>=INTV_WORDS){
        input_pong_index-=INTV_WORDS;
92      outputPos=(outputPos+1)%BITS_IN_SYMBOL;
      }
94
    }
96  ping_index++;
    return output;
98 }
```

Listing 12: dspCode/interleaver_new.c

```
 #include "input.h"
2
 /*************for writeLVDS()*****************/
4#define PREAMBLE_SIZE      640   // 640 / 160 = 4
 #define PREAMBLE_TIMES     4   // we transmit 4 preamble symbols, than 12 times for samples
6#define IFFT_SIZE       128

8// To clarify, 1 symbol = 48 tones + 12 padding bits + 4 subcarrier pilots = 64 tones, 1 ifft_out
     = 64 tones + 16 CPs

10void preamble_transmit(float*, int);
 void data_transmit(float*, float*, float*);
12
 void getLVDS(float received[SAMPLE_SIZE], float transmitted[SAMPLE_SIZE]);
```

Listing 13: dspCode/lvds.h

```
 #include "lvds.h"
2
 float preamble_sequence[PREAMBLE_SIZE] =
4  #include "preambles.txt"

6void getLVDS(float received[SAMPLE_SIZE], float transmitted[SAMPLE_SIZE]){
   int i;
8  for (i=0; i<(SAMPLE_SIZE); i++){
     received[i] = transmitted[i];
10 }
 }
12
 void preamble_transmit(float writeLVDS_output[SAMPLE_SIZE], int i){
14 int offset = i * SAMPLE_SIZE;
   int j;
16 for(j=0; j<(SAMPLE_SIZE); j++){
```

```
    writeLVDS_output[j] = preamble_sequence[j+offset];
  }
 }

 void data_transmit(float ifft_out_cp[NUM_CP], float ifft_out[IFFT_SIZE], float writeLVDS_output[
    SAMPLE_SIZE]){
  int i;
  for (i=0; i<NUM_CP; i++){
    writeLVDS_output[i]=ifft_out_cp[i];
  }
  for (i=NUM_CP; i<IFFT_SIZE + NUM_CP; i++){
    writeLVDS_output[i]=ifft_out[i-NUM_CP];
  }
 }
```

Listing 14: dspCode/lvds.c

```
#include "input.h"

#define WORD32 unsigned int // to create a 32-bit space
#define WORD64 unsigned long long
#define CMPNUM unsigned long long // to creat a  64 bits space, higher 32 bits for real and lower
    32 bits for imagine

#define TRUE    1
#define FALSE   0
#define DEBUG   TRUE
#define PLUS_1  0x000000003F800000 // imaginary part before, so it can be written to the lower
    bits
#define MINUS_1 0x00000000BF800000
#define ZERO    0x0000000000000000
#define MASK  0x00000001
#define LENGTH  16 // the length of data flow in Bits, just assume it is 32 now
#define MAP_SIZEOUT 128

void mapper(WORD64 input_bits, float buffout[MAP_SIZEOUT]);
void init_map_buffout(float buffout[MAP_SIZEOUT]);
void demapper(float buffin[MAP_SIZEOUT], WORD64 *buffout);
```

Listing 15: dspCode/mapper.h

```
/*******************************Upadated on 10th April***************
 To modify the input data type from two 32 bits to a 64 bits,
which contains 48 bits of one symbol, in order to be more efficient.
 This update costs 434 cycles for mapper and 234 for demapper.
However, the 24 bits one costs 435 cycles for mapper and 229 for demapper.
```

```
 So, not a great progress on efficiency of the mapper and demapper. :)
--Kainan
 *****************************************************************************/


//************************ Includes ***********************************
 #include "mapper.h"
#include "builtins.h"
 #include <stdio.h>

 //************************ Externs ************************************
int map_tbl[N_SYMBITS] = { 2, 4, 6, 8, 10, 12, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40,
      44, 46, 48, 50, 52, 76, 78, 80, 82, 84, 88, 90, 92, 94, 96, 98, 100, 102, 104, 106, 108, 110,
      112, 116, 118, 120, 122, 124, 126 };
 //To record which order in buffout[] should be write data in

 /************************
0: pad bit
 1: pilot subcarriers 1
2: pilot subcarriers -1
 3: data
************************/
 void init_map_buffout(float buffout[MAP_SIZEOUT]){
  int i;
  for (i = 0; i< MAP_SIZEOUT; i++){
    buffout[i] = 0;
  }
  buffout[14] = 1;
  buffout[42] = -1;
  buffout[86] = 1;
  buffout[114] = 1;
}

void mapper(WORD64 input_bits, float buffout[MAP_SIZEOUT]){
  WORD64 input_bits_reorder;
  int cur_bit, cur_addr;
  int i;
  //input_bits = __builtin_compose_64(buffin[1], buffin[0]);
  //input_bits_reorder = (input_bits & 0xffffff) | ((input_bits >> 8) & 0xffffff000000);
  // 63==========================================0
  //  sixteen 0 |  low 24 bits | high 24 bits  |
  // ============================================
  input_bits_reorder = ((input_bits>>(N_SYMBITS/2))&0xffffff)|((input_bits<<(N_SYMBITS/2))&0
    xffffff000000); // fftshift for OFDM processing
  init_map_buffout(buffout);
  for (i = 0; i<N_SYMBITS; i++)
  {
    cur_addr = map_tbl[i];
```

```
51      if (((input_bits_reorder & 0x1) == 1)){
          buffout[cur_addr] = 1;
53      }
        else{
55        buffout[cur_addr] = -1;
        }
57      input_bits_reorder = input_bits_reorder >> 1;
    }
59  }

61  //0 ====================
    //  |                    | 0
63  //  | buffin1 complex    |
    //  |                    | 23
65  //  ====================
    //  |                    | 0
67  //  | buffin0 complex    |
    //  |                    | 23
69  //64====================


71
    void demapper(float buffin[MAP_SIZEOUT], WORD64 *buffout){
73    // Hard decision maker.
      int cur_addr, i;
75    WORD64 input_bits_preorder = 0;

77    for (i = 0; i<N_SYMBITS; i++)
      {
79      cur_addr = map_tbl[i];
        if (buffin[cur_addr] > 0){
81        //input_bits_preorder = (input_bits_preorder>>1) | 0x800000000000;
          input_bits_preorder = (input_bits_preorder>>1) | 0x800000000000;
83      }
        else{
85        input_bits_preorder = input_bits_preorder >> 1;
        }
87    }

89    (*buffout) = ((input_bits_preorder>>24)&0xffffff)|((input_bits_preorder<<24)&0xffffff000000); //
         shift again to get back original order
      // 63==========================================0
91    //  sixteen 0 |  high 24 bits | low 24 bits  |
      // ============================================
93  }
```

Listing 16: dspCode/mapper.c

```
#include "input.h"

unsigned int memRead(unsigned int*);
void memWrite(unsigned int, unsigned int RECEIVED[TOT_SYMBOL_NUM]);
```

Listing 17: dspCode/memory.h

```
#include <stdio.h>
#include "memory.h"
#include "convolutional.h"

void memWrite(unsigned int vit_output, unsigned int RECEIVED[TOT_SYMBOL_NUM]){
  static int super_frame_num = 0;
  static int output_index=0;
  static unsigned int offset = -EFF_CONV_SIZE;
  int loc;

  if (super_frame_num==0){
    (offset+=EFF_CONV_SIZE);
    super_frame_num++;
    // Dont give the flush bits to the receiver memory!
    return;
  }
  loc = (offset-1-(output_index%EFF_CONV_SIZE))%TOT_SYMBOL_NUM;
  //printf("loc = %d\nsymb_num = %d\neff_conv_size = %d\n", loc,TOT_SYMBOL_NUM,EFF_CONV_SIZE);
  RECEIVED[loc]=vit_output;

  super_frame_num=(super_frame_num+1)%CONV_SIZE;
  output_index=output_index+1;

}

unsigned int memRead(unsigned int *INPUT){
  static int input_index=0;
  static int num_flushes=0;
  static int super_frame_num=1;
  unsigned int output;
  // Flush the convolutional encoder
  if (super_frame_num==CONV_SIZE){
    super_frame_num=1;
    return 0x000000;

  }

  // Never will be transmitting so many symbols that overflow is a problem
  if (input_index >= TOT_SYMBOL_NUM){
```

```
40      output = 0x000000;
    } else {
42      output = INPUT[input_index];
    }
44
    input_index=(input_index+1);
46  super_frame_num++;
    return output;
48 }
```

Listing 18: dspCode/memory.c

```
 #include <stdio.h>
2#include <stdlib.h>
 #include <math.h>
4#include "builtins.h"
 #define PI          3.1415926
6#define PRINT_LINE_LEN  20
 #define TRANS_SIZE     128
8#define RANDNUM01      (float)rand() / 2147483647.0 // to get a random number between 0 and 1. ps.
     rand() is [0 2^31-1]
 #define RANDGAUSS_NORM  sqrtf(-2*logf(RANDNUM01))*cosf(2*PI*RANDNUM01) // Box-Muller method, float
      data type, normal distribution N(0,1)
10#define Pr          1.118 // received power
 #define Bw          5.0e6 // channel bandwidth
12#define Rx_amp       1.0574 // Pr = A^2, wehre A is the amplitude of the received signal A = sqrtf(
     Pr)
 //#define SPACENOISE RANDGAUSS*3.2171e-11
14//#define CHANNELGAIN 2.84e-6

16void printArray(unsigned long long int*, int);
 void printArrayInt(unsigned int*, int);
18int __builtin_count_ones(int);
 float ber_calc(unsigned int*, unsigned int*);
20void channel_awgn(float*, float*, float);
```

Listing 19: dspCode/testbench.h

```
 #include "testbench.h"
2#include "input.h"

4void printArray(unsigned long long int in[], int size) {
   int i;
6  for (i = 0; i < size; i++) {
     // Can add in 0x before % size
8    printf("%d: 0x%016llx ", i,in[i]);
```

```
    if ( (i+1)%PRINT_LINE_LEN == 0 ) {
      printf("\n");
    }
  }
  printf("\n");
}

void printArrayInt(unsigned int in[], int size) {
  int i;
  for (i = 0; i < size; i++) {
    // Can add in 0x before % size
    printf("%d: 0x%08x ", i,in[i]);
    if ( (i+1)%PRINT_LINE_LEN == 0 ) {
      printf("\n");
    }
  }
  printf("\n");
}


float ber_calc(unsigned int INPUT[TOT_SYMBOL_NUM], unsigned int OUTPUT[TOT_SYMBOL_NUM]){
  float length = TOT_SYMBOL_NUM * 48.0; // the total number of bits of the input and output
  int i;
  int error_sum = 0;
  int error_bits;
  int error_num;
  float ber;
  for(i = 0; i < TOT_SYMBOL_NUM; i++){
    error_bits = INPUT[i] ^ OUTPUT[i]; // error bits are 1 and correct bits are 0
    error_num = __builtin_count_ones(error_bits); // count 1s, namely count the number of error
    bits
    error_sum = error_sum + error_num;
  }
  ber = (float)error_sum / length;
  return ber;
}

void channel_awgn(float transmitted[TRANS_SIZE], float received[TRANS_SIZE], float snr_db){
//  channel model: receive = k * transmit + n; where k is the channel gain and n is white noise
  int i;
  float snr = alog10f(snr_db); // convert dB into unit (W/W)
  float fi = sqrtf(Pr / (snr * Bw)); // standard deviation of the noise
  for(i = 0; i<TRANS_SIZE;i++){
    received[i] = Rx_amp * transmitted[i] + fi * RANDGAUSS_NORM;
  }
}
```

Listing 20: dspCode/testbench.c

C MATLAB Code

C.1 AFE Driver

```
%% ELEN90070 FYP
% Code runs the  SDR receive process
% Written by Callum Maltby (539190) 27/4/16

% Set properties
Fs = 520841;
samplePerFrame = 1280;
numFrames = 200;
runTime = numFrames*samplePerFrame/Fs;

fprintf('\n=========================\n Capturing RF signals \n=========================\n');

% Run the receiver simulink model
simOut = sim('zynqRadioQPSKRxAD9361AD9364SL_unchanged', runTime);

fprintf('\n=========================\n Finished receiving \n=========================\n');

% Convert complex timeseries to .mat
convertAFESamples;

% Run OFDM synchronisation
OFDMSynchronizationExample;
```

Listing 21: matlabCode/runSdrRx.m

```
function [FER, BER, txMsgInBitsAll, rxMsgInBitsAll] = calculateOFDMBER(msg, decMsgInBits,
    numFramesDetected)
% CALCULATEOFDMBER: BER calculation based on the repeatedly transmitted
% message and the detected frames.

% Copyright 2014 The MathWorks, Inc.

% Reconstruct transmitted bit message
txMsgInBits     = coder.const(double(dec2bin(msg, 7).'));
txMsgInBits     = txMsgInBits(:) - 48;
txMsgInBitsAll = repmat(txMsgInBits, 1, numFramesDetected);
bitMsgLen       = length(txMsgInBits);

% Remove padded bits from detected frames
rxMsgInBits     = reshape(decMsgInBits, round(length(decMsgInBits)/numFramesDetected),
```

```
    numFramesDetected);
rxMsgInBitsAll = rxMsgInBits(1:bitMsgLen, :);

% FER calculation
numFrmErr = sum(any(txMsgInBitsAll ~= rxMsgInBitsAll));
FER = numFrmErr/numFramesDetected;

% BER calculation
numBitErr = sum(txMsgInBitsAll(:) ~= rxMsgInBitsAll(:));
BER = numBitErr/(bitMsgLen*numFramesDetected);

% [EOF]
```

Listing 22: matlabCode/calculateOFDMBER.m

```
clear all
close all
%% OFDM Synchronization
% This example shows a method for digital communication with OFDM
% synchronization based upon the IEEE 802.11a standard.  System objects
% from the Communication System Toolbox are utilized to provide OFDM
% modulation and demodulation and help synchronization functionality.  In
% particular, this example illustrates methods to address real-world
% wireless communication issues like carrier frequency recovery, timing
% recovery, and frequency domain equalization.

%   Copyright 2013-2014 The MathWorks, Inc.

%% Implementations
% This example describes a MATLAB implementation of OFDM synchronization,
% based upon the IEEE 802.11a standard [ <#9 3> ].

%% Introduction
% The IEEE 802.11a standard describes the transmission of an OFDM modulated
% signal for information exchange between systems in local and metropolitan
% area networks.  This example utilizes the physical layer outlined by that
% standard, specifically the preamble symbols and the OFDM grid structure.
%
% The purpose of this example is:
%
% * To model a general OFDM wireless communication system that is able to
% successfully recover a message, which was corrupted by various simulated
% channel impairments.
%
% * To illustrate the use of key Communications System Toolbox(TM) tools
% for OFDM system design and OFDM symbol synchronization
%
```

```
% * To illustrate the performance benefits of MATLAB Coder(TM)

%% Initialization
% Adjustable transmitter parameters including the payload message in each
% frame that consists of several OFDM symbols and the number of transmitted
% frames.
message = 'Live long and prosper, from the Communications System Toolbox Team at MathWorks!';
numFrames = 1e2;

% Adjustable channel parameters
EbN0dB = 12;            % Channel noise level (dB)
frequencyOffset = 1e4;  % Frequency offset (Hz)
phaseOffset = 15;       % Phase offset (Degrees)
delay = 80;             % Initial sample offset for entire data stream (samples)

% Display recovered messages
displayRecoveredMsg = true;

% Enable scope visualizations
useScopes = false;

% Check for MATLAB Coder license
%useCodegen = checkCodegenLicenseOFDM;
useCodegen = false;


% By default the transmitter and receiver functions will be recompiled
% between every run, which is not always necessary.  To disable receiver
% compilation, change "compileIt" to false.
%compileIt = useCodegen;
compileIt = false;

%% Code Architecture for the System
% This example models a digital communication system based upon the IEEE
% 802.11a standard [ <#9 3> ].  The system is broken down into four
% functions: generateOFDMSignal, applyOFDMChannel, receiveOFDMSignal, and
% calculateOFDMBER.
%
% 1) generateOFDMSignal: set up and step an OFDMTransmitter System object.
% The object converts the payload message into a bit stream which is first
% PSK modulated, then OFDM modulated, and finally prepended by preamble
% OFDM symbols to form an individual frame. The transmitter repeats this
% frame |numFrames| times.
%
% 2) applyOFDMChannel: models the channel with carrier offset, timing
% offset, and additive white Gaussian noise (AWGN).
%
% 3) receiveOFDMSignal: set up and step an OFDMReceiver System object. The
```

```
% object models a series of components at the receiver, including timing
% recovery, carrier frequency recovery, channel equalization, and
% demodulation. The object can also be configured to show multiple scopes
% to visualize the receiver processing. The output of the OFDMReceiver
% object's step method is the decoded bit stream from those detected
% frames.
%
% 4) calculateOFDMBER: calculate the system frame error rate (FER) and bit
% error rate (BER) based on the original payload message in each frame and
% the bit output from the OFDMReceiver System object.

%% Description of the Individual Components and Algorithms
% *Transmitter*
%
% The OFDMTransmitter System object generates an OFDM signal based upon the
% IEEE 802.11a standard with a supplied ASCII payload. Each transmission
% frame is made up of several OFDM symbols, including preamble and data
% symbols. Identical frames are repeated by the transmitter based on the
% value supplied. Frames are padded to fill the OFDM grid when necessary.
%
% *Channel*
%
% This component simulates the effects of over-the-air transmission. It
% degrades the transmitted signal with both phase and frequency offset, a
% delay to mimic channel delay between transmitter and receiver, and AWGN.
% The noise level of the AWGN is given in dB.
%
% *Receiver*
%
% This OFDMReceiver System object recovers the original transmitted payload
% message. It is divided into four primary operations in this order:
%
% 1) Timing Recovery: This component is responsible for determining the
% sample location of the start of a given frame. More specifically, it
% utilizes a known preamble sequence in the received frame found through a
% cross-correlation. The cross-correlated data will contain a specific
% peak arrangement/spacing which allows for identification. The preamble
% itself is designed to produce this specific shape in the time domain.
% This identification method is based upon [ <#9 1> ]. The locatePreamble
% method of the object, which is responsible for this operation, uses a
% normalized minimum peak height, and a minimum number of required peaks to
% provide a possible preamble match.
%
% 2) Carrier Frequency Recovery: Frequency estimation is accomplished by
% calculating the phase difference in the time domain between halves of the
% long portion of the 802.11a preamble. This phase difference Phi is then
% converted to a frequency offset. This is a common technique originally
% published by Schmidl and Cox [ <#9 2> ]. This implementation of the
```

```
% phase measurement assumes that the true offset is within pi, or one
% frequency bin of the FFT.  In the case of 802.11a a bin is 312.5kHz wide.
%
% 3) Frequency Domain Equalization: Since the frequency estimate can be
% inaccurate, additional phase rotation will exist at the subcarrier level
% of the OFDM symbol.  As well as phase rotations, channel fading will also
% affect the received signal.  Both of these impairments are corrected by a
% frequency domain equalizer.  The equalizer has two stages, utilizing both
% preamble and pilot data.  First, the received payload is equalized
% through the use of taps generated from the received long preamble
% samples.  Then the pilot subcarriers are extracted, and interpolated in
% frequency to provide a full channel estimate. The payload is next
% equalized using these pilot estimates.
%
% 4) Data Decoder: Finally the OFDM subcarriers are demodulated and then,
% PSK demodulated into bits, from which the original payload message can be
% recovered.
%
% *BER Calculation*
%
% This component calculates the system FER and BER based on the original
% payload message and the decoded bit stream from the detected frames at
% the receiver. The undetected frames are not counted in the calculation.
%
% *Display of Recovered Message*
%
% The recovered message at the receiver is displayed for each detected
% frame. Since the original message length is not sent to the receiver, the
% padded bits in each frame are also recovered into characters and
% displayed. So you may see up to 7 meaningless characters at the end of
% each recovered message.
%
% *Scopes*
%
% * constellation diagrams showing the received signal before and after
% frequency domain equalization
%
% * vector plot of the equalizer taps used for a given frame
%
% * a spectrum analyzer displaying detected frames of data
%
% * a time plot displaying the start of detected frames
%
% * a time plot displaying the frequency estimate of the transmitter's
% carrier offset for detected frames
%% OFDM Synchronization Test Overview
% A large data vector is regenerated for a given EbN0 value by the
% generateOFDMSignal function. This data is then passed through the
```

```
177% applyOFDMChannel function which introduces several common channel
  % impairments.  Finally the data is passed to the receiver for recovery.
179% The receiveOFDMSignal function operates by processing data on a
  % frame-by-frame basis.  This processing mechanism is self-contained for
181% performance benefits when using code generation and for code simplicity.
  % This script by default generates code for the transmitter and receiver
183% functions; this is accomplished by using the *codegen* command provided
  % by the MATLAB Coder(TM) product. The *codegen* command translates
185% MATLAB(R) functions to a C++ static or dynamic library, executable, or to
  % a MEX file, producing a code for accelerated execution. The generated C
187% code runs several times faster than the original MATLAB code.
  %
189% During operation, the receiver will display a series of plots
  % illustrating certain synchronization results and effects on the signal.
191
  % Compile transmitter with MATLAB Coder
193 if compileIt
      codegen generateOFDMSignal -args {coder.Constant(message), coder.Constant(numFrames)}
195 end

197% Generate transmission signal
  if useCodegen
199     [txSig, frameLen] = generateOFDMSignal_mex(message, numFrames);
  else
201     [txSig, frameLen] = generateOFDMSignal(message, numFrames);
  end
203
  % Pass signal through channel
205 rxSig = applyOFDMChannel(txSig, EbN0dB, delay, frequencyOffset, phaseOffset);
  % Compile receiver with MATLAB Coder
207 if compileIt
      codegen  receiveOFDMSignal -args {rxSig, coder.Constant(frameLen), coder.Constant(
       displayRecoveredMsg), coder.Constant(useScopes)}
209 end

211% Read in data that was output by the .xls receiver
  load Data_dump/data_dump_ofdm.mat
213 data_dump = [data_dump; data_dump];
  rxSig = data_dump(1:138080).*02/0.01;
215 %load rxSdrOut.mat
  %rxSig = complexSdrOut(1:138080).*0.2/0.01;
217
  % Recover signal
219 if useCodegen
      [decMsgInBits, numFramesDetected] = receiveOFDMSignal_mex(rxSig, frameLen, displayRecoveredMsg
      , useScopes);
221 else
      [decMsgInBits, numFramesDetected] = receiveOFDMSignal(rxSig, frameLen, displayRecoveredMsg,
```

```
    useScopes);
end

% Calculate average BER
[FER, BER, txMsgBits, rxMsgBits] = calculateOFDMBER(message, decMsgInBits, numFramesDetected);
fprintf('\nAt EbNo = %5.2fdB, %d frames detected among the %d transmitted frames with FER = %f and
    BER = %f\n', ...
    EbN0dB, numFramesDetected, numFrames, FER, BER);

% Find which bits are in error and display their position in the symbol
[rows, cols] = size(txMsgBits);
fprintf('The message bits in error are: ');
for i = 1:rows
    for j = 1:cols
        if txMsgBits(i, j) ~= rxMsgBits(i, j)
            fprintf('%d ', rem(i,48));
        end
    end
end
fprintf('\n');

%% Summary
% This example utilizes several MATLAB System objects to simulate digital
% communication with OFDM over an AWGN channel. It shows how to model
% several parts of the OFDM system such as modulation, frequency
% estimation, timing recovery, and equalization. The simulation also
% displays information about the operation of the synchronization
% algorithms through a series of plots. This example also utilizes code
% generation, allowing the simulation to run several times faster than the
% original MATLAB code.

%% Appendix
% The following System objects are used in this example:
%
% * <matlab:edit([matlabroot,'\toolbox\comm\commdemos\OFDMTransmitter.m']) OFDMTransmitter.m>
% * <matlab:edit([matlabroot,'\toolbox\comm\commdemos\OFDMReceiver.m']) OFDMReceiver.m>
% * <matlab:edit([matlabroot,'\toolbox\comm\commdemos\OFDMScopes.m']) OFDMScopes.m>
%
% The following helper functions are used in this example:
%
% * <matlab:edit([matlabroot,'\toolbox\comm\commdemos\checkCodegenLicenseOFDM.m'])
    checkCodegenLicenseOFDM.m>
% * <matlab:edit([matlabroot,'\toolbox\comm\commdemos\generateOFDMSignal.m']) generateOFDMSignal.m
    >
% * <matlab:edit([matlabroot,'\toolbox\comm\commdemos\applyOFDMChannel.m']) applyOFDMChannel.m>
% * <matlab:edit([matlabroot,'\toolbox\comm\commdemos\receiveOFDMSignal.m']) receiveOFDMSignal.m>
% * <matlab:edit([matlabroot,'\toolbox\comm\commdemos\processOFDMScopes.m']) processOFDMScopes.m>
% * <matlab:edit([matlabroot,'\toolbox\comm\commdemos\calculateOFDMBER.m']) calculateOFDMBER.m>
```

```
% * <matlab:edit([matlabroot,'\toolbox\comm\commdemos\getOFDMPreambleAndPilot.m'])
    getOFDMPreambleAndPilot.m>

%% References
% # Minn, H.; Zeng, M.; Bhargava, V.K., "On timing offset estimation for
% OFDM systems," Communications Letters, IEEE , vol.4, no.7, pp.242,244,
% July 2000
% # Schmidl, T.M.; Cox, D.C., "Robust frequency and timing synchronization
% for OFDM," Communications, IEEE Transactions on , vol.45, no.12,
% pp.1613,1621, Dec 1997
% # IEEE Std 802.11a, "Part 11: Wireless LAN Medium Access Control (MAC)
% and Physical Layer (PHY) Specifications," 1999.

displayEndOfDemoMessage(mfilename)
```

Listing 23: matlabCode/OFDMSynchronizationExample.m

C.2 Coding Scheme Simulator and Data Extraction

```
% ELEN90070 FYP
% Script calculates BER for a number of OFDM trials
% Written by Callum Maltby (539190) on 29/5/16

% Test files are kept in 'Trial <i>' folders
% Live test files are kept in the local directory

clc
clear all
close all

% Assert output settings
rootDir = 'C:\Users\Callum\Dropbox\Uni\UMSP CubeSat FYP\Laptop Files\BER';
%rootDir = 'D:\Uni\UMSP CubeSat FYP\Laptop Files\BER';
displayBER = 0;
displayHex = 0;
printHexSpaces = 1;
plotBER = 1;
plotCodingGain = 1;
liveTest = 0;
otherTrialDemo = 0;
displaySpectrum = 0;
displayErrorPos = 0;
printText = 0;
showStatus = 1;
```

```
% Define experimental Settings
startTrial = 12;%12
endTrial = 30;%30;
numTrials = endTrial - startTrial + 1;

% Define simulation parameters
numStates = 8;
numFrames = 1000;   %1000 % Increasing the number of frames just averages out the result
numProbPoints = 500;
minBerPower = -5; %-5
maxBerPower = -0.4; % -0.4 for BER curve
maxCodingGainPower = -1.5; %-1
filterSpan = numProbPoints/10;

% Display BER for live test
if (liveTest)
    fprintf('For the live test:\n');
    [berChan, berData] = oneTransBerCalc('txEncoderOutput.dat', 'rxDecoderInput.dat', '
    txBinaryMessage.dat', 'rxBinaryMessage.dat', 1, displayHex, printHexSpaces, -1,
    displayErrorPos, printText);
end

% Quit if only live test is desired
if (~otherTrialDemo && ~plotCodingGain)
    return;
end

% Make space to store the BER values
BERch = zeros(1,numTrials);
BERdata = zeros(1,numTrials);

% Extract data from each run
for i = startTrial:endTrial

    if ( (displayErrorPos == 1) || (displayHex == 1) || (printText == 1) )
        fprintf('\n----Trial %d----\n\n', i);
    end

    subFolder = strcat('Trial', {' '}, num2str(i) );

    txEnc = char( fullfile ( rootDir, subFolder, 'txEncoderOutput.dat' ) );
    rxDec = char( fullfile ( rootDir, subFolder, 'rxDecoderInput.dat' ) );
    txMsg = char( fullfile ( rootDir, subFolder, 'txBinaryMessage.dat' ) );
    rxMsg = char( fullfile ( rootDir, subFolder, 'rxBinaryMessage.dat' ) );

    [berChan, berData] = oneTransBerCalc(txEnc, rxDec, txMsg, rxMsg, displayBER, displayHex,
    printHexSpaces, i, displayErrorPos, printText);
    BERch(i) = berChan;
```

```
    BERdata(i) = berData;

    % Display received frame spectrum
    if (displaySpectrum)
        Fs = 520841;
        rxTimeDomainPath = char( fullfile ( rootDir, subFolder, 'rxTimeDomain.dat' ) );
        load(rxTimeDomainPath);
        X = zeros(1, length(rxTimeDomain)/2);
        for j = 1:length(rxTimeDomain)/2;
            X(j) = rxTimeDomain(2*j-1) + 1i*rxTimeDomain(2*j);
        end

        z = fft(X);
        z = fftshift(z);
        N = length(X);
        f = [-N/2:N/2-1]/N.*Fs;
        figure(i)
        plot(f/1000,abs(z))
        xlabel('Baseband frequency [kHz]');
        ylabel('Linear amplitude');
        title('Received OFDM Signal Spectrum');
    end

end

% Elements were indexed by their Trial number
BERch = BERch(startTrial:endTrial);
BERdata = BERdata(startTrial:endTrial);

% Plot BERch against BERdata for experimental and theoretical results
if (plotBER || plotCodingGain)
    % Generate theoretical BER curves

    % Use the last test file to generate data
    fileID = fopen(txMsg,'r');
    A = textscan(fileID, '%s');
    fclose(fileID);
    B = A{1};
    C = reshape(B, [], 1);
    D = strrep(C,',', '');
    E = strrep(D,'}', '');
    E = strrep(E,'{', '');
    F = strvcat( strrep(E,';', '')' );
    [rows,cols] = size(F);
    G = F(:,3:cols);
    H = reshape(G', 1, []);
    input = hexToBinaryVector(H, 4*length(H) ) ;

```

```
ERROR_PROB = logspace(minBerPower, maxBerPower, numProbPoints);

BERchTheorFlag = -1;
% Adjust the error rate
for k = 1:length(ERROR_PROB)
    if (showStatus)
        fprintf('Theoretical BER: %.2f%% done\n', 100.0*k/length(ERROR_PROB) );
    end
    numBitErrors = 0;
    numBitErrorsDecoded = 0;
    % Send multiple frames
    for i = 1:numFrames
        % Generate the convolutional code output. The most popular code,
        % implemented here, has k = 4 and R = 1/2
        encoderOutput = convEncoder(input);

        [receiverInput, errors] = channelNoise(encoderOutput, ERROR_PROB(k) );
        numBitErrors = numBitErrors + errors;

        receiverOutput = convDecoder_v3(receiverInput, numStates);

        % Trim off the flushing zeros
        output = receiverOutput(1, 1:length(input) );

        % Find errors in binary message
        bitDiff = xor(input, output);
        numBitErrorsDecoded = numBitErrorsDecoded + sum(bitDiff);
    end
    BERchTheor(k) = numBitErrors/(length(receiverInput)*numFrames);
    BERdataTheor(k) = numBitErrorsDecoded/(length(input)*numFrames);

    % Assert a flag for the coding gain curve to start at
    if ( (BERchTheorFlag == -1) && (BERchTheor(k) > 10^maxCodingGainPower) )
        BERchTheorFlag = k;
    end

end

% Plot experimental and theoretical results together
if (plotBER)
    figure(1)
    scatter(BERch, BERdata);
    hold on
    xlabel('Channel BER');
    ylabel('Data BER');
    title('BER plot');
    plot(BERchTheor, BERdataTheor, 'r');
    legend('Experimental', 'Simulated');
```

```
        set(gca,'xscale','log')
169     hold off
    end
171

173 if (plotCodingGain)
        berBpsk =
    [0.499994358104165;0.499993669688756;0.499992897273962;0.499992030610310;0.499991058197699;0.499989967132
    e-05;3.87210821552205e-06];
175     ebn0 = -100:10';
        delta = 10^-5;
177     for k = 1:BERchTheorFlag
            if (showStatus)
179             fprintf('Coding Gain:     %.2f%% done\n', 100.0*k/BERchTheorFlag );
            end
181         % Take channel BER to find SNR
            if (BERchTheor(k) > (0.5 - delta) )
183             uncodedSNR(k) = -20;
            elseif (BERchTheor(k) < delta )
185             uncodedSNR(k) = 12;
            else
187             uncodedSNR(k) = interp1(berBpsk, ebn0, BERchTheor(k), 'spline');
            end
189
            if (BERdataTheor(k) > (0.5 - delta) )
191             codedSNR(k) = -20;
            elseif (BERdataTheor(k) < delta )
193             codedSNR(k) = 12;
            else
195             codedSNR(k) = interp1(berBpsk, ebn0, BERdataTheor(k), 'spline');
            end
197         %{
            k
199         BERchTheor(k)
            uncodedSNR(k)
201         BERdataTheor(k)
            codedSNR(k)
203         %}
            %codedSNR(k) = interp1(berBpsk, ebn0, BERdataTheor(k), 'cubic');
205         codingGain(k) = codedSNR(k) - uncodedSNR(k)
        end
207
        %{
209     figure(2)
        plot(uncodedSNR, codingGain, 'b');
211     grid on
        xlabel('Uncoded SNR [dB]');
213     ylabel('Coding Gain [dB]');
```

```
        title('Coding Gain');
        legend('Simulated Results');
        %}

        % Sort arrays
        [uncodedSNRSorted, sortIndex] = sort(uncodedSNR);
        codingGainSorted = codingGain(sortIndex);
        % Moving average filter
        codingGainSmoothed = smooth(uncodedSNRSorted, codingGainSorted, filterSpan, 'moving'); %
    ,1,'rloess');

        figure(3)
        plot(uncodedSNRSorted, codingGainSorted, 'b', uncodedSNRSorted, codingGainSmoothed, 'r');
        grid on
        xlabel('Uncoded SNR [dB]');
        ylabel('Coding Gain [dB]');
        title('Coding Gain');
        legend('Simulated Results', 'Moving Average Filter');

    end

    %uncodedSNRSorted =
    [redacted]

    %codingGainSorted =
    [redacted]

end
```

Listing 24: matlabCode/berCalcDemo.m

```
% ELEN90070 FYP
% Frame BER Calculation
% Written by Callum Maltby (539190) on 28/5/16

% Required Files
%  txEncoderOutput.dat
%  rxDecoderInput.dat
%  txBinaryMessage.dat
%  rxBinaryMessage.dat

function [berChan, berData] = oneTransBerCalc(txEnc, rxDec, txMsg, rxMsg, displayBER, displayHex,
    printHexSpaces, trialNum, displayErrorPos, printText)

    % ========================= %
    %  Calculate channel BER  %
    % ========================= %
```

```
% Generate bit vector for encoder output
fileID = fopen(txEnc,'r');
A = textscan(fileID, '%s');
fclose(fileID);

B = A{1};
C = hexToBinaryVector(B);
[rows,cols] = size(C);

txEncoderOutputBits = [];
for i = 1:rows/2
    newVec = [C(2*i, (cols/2+1):cols), C(2*i-1,:)];
    txEncoderOutputBits = [txEncoderOutputBits, newVec];
end

if (displayHex)
    txEncoderOutputHex = char(binaryVectorToHex(txEncoderOutputBits));
    fprintf('--txEncoderOutputHex--\n');
    for j = 1:length(txEncoderOutputHex)
        fprintf('%c', txEncoderOutputHex(j));
        if (printHexSpaces)
            if ( rem(j,4) == 0 )
                fprintf(' ');
            end
            if ( rem(j,12) == 0 )
                fprintf('\n');
            end
        end
    end
    fprintf('\n');
end

% Generate bit vector for decoder input
fileID = fopen(rxDec,'r');
A = textscan(fileID, '%s');
fclose(fileID);

B = A{1};
C = hexToBinaryVector(B);
[rows,cols] = size(C);

rxDecoderInputBits = [];
for i = 1:rows/2
    newVec = [C(2*i, (cols/2+1):cols), C(2*i-1,:)];
    rxDecoderInputBits = [rxDecoderInputBits, newVec];
end

```

```
    if (displayHex)
        rxDecoderInputHex = char(binaryVectorToHex(rxDecoderInputBits) );
        fprintf('--rxDecoderInputHex--\n');
        for j = 1:length(txEncoderOutputHex)
            fprintf('%c', rxDecoderInputHex(j) );
            if (printHexSpaces)
                if ( rem(j,4) == 0 )
                    fprintf(' ');
                end
                if ( rem(j,12) == 0 )
                    fprintf('\n');
                end
            end
        end
        fprintf('\n');
    end

    berChanDiff = xor(txEncoderOutputBits,rxDecoderInputBits);
    berChan = sum( berChanDiff/length(berChanDiff) );

    % Print out bits in error
    if (displayErrorPos)
        fprintf('Errors in berChanDiff have indices: \n');
        for k = 1:length(berChanDiff)
            if ( berChanDiff(k) == 1 )
                fprintf('%d,', k);
            end
            if (printHexSpaces)
                if ( rem(k,20) == 0 )
                    fprintf('\n');
                end
            end
        end
        fprintf('\n\n');
    end

    %====================%
    %  Calculate data BER  %
    %====================%

    % Generate bit vector for Tx data source
    fileID = fopen(txMsg,'r');
    A = textscan(fileID, '%s');
    fclose(fileID);

    B = A{1};
    C = reshape(B, [], 1);
    D = strrep(C,',', '');
```

```
D = strrep(D,'{', '');
D = strrep(D,'}', '');
D = strvcat( strrep(D,';', '')' );
[rows,cols] = size(D);
D = D(:,3:cols);
E = reshape(D', 1, []);
txMsgBinary = hexToBinaryVector(E, 4*length(E) );

if (displayHex)
    txMsgHex = char(binaryVectorToHex(txMsgBinary) );
    fprintf('--txMsgHex--\n');
    for j = 1:length(txMsgHex)
        fprintf('%c', txMsgHex(j) );
        if (printHexSpaces)
            if ( rem(j,6) == 0 )
                fprintf('\n');
            end
        end
    end
    fprintf('\n');
end


% Generate bit vector for Rx data sink
fileID = fopen(rxMsg,'r');
A = textscan(fileID, '%s');
fclose(fileID);

B = A{1};
C = reshape(B, [], 1);
D = strrep(C,',', '');
D = strrep(D,'{', '');
D = strrep(D,'}', '');
D = strvcat( strrep(D,';', '')' );
[rows,cols] = size(D);
D = D(:,5:cols);
E = reshape(D', 1, []);

rxMsgBinary = hexToBinaryVector(E, 4*length(E) );

if (displayHex)
    rxMsgHex = char(binaryVectorToHex(rxMsgBinary));
    fprintf('--rxMsgHex--\n');
    for j = 1:length(rxMsgHex)
        fprintf('%c', rxMsgHex(j) );
        if (printHexSpaces)
            if ( rem(j,6) == 0 )
                fprintf('\n');
```

```
                end
            end
        end
        fprintf('\n');
    end

    % Calculate transmission BER
    berDataDiff = xor(txMsgBinary,rxMsgBinary);
    berData = sum( berDataDiff/length(berDataDiff) );

    % Print out bits in error
    if (displayErrorPos)
        fprintf('Errors in berDataDiff have indices: \n');
        for k = 1:length(berDataDiff)
            if ( berDataDiff(k) == 1 )
                fprintf('%d,', k);
            end
            if (printHexSpaces)
                if ( rem(k,20) == 0 )
                    fprintf('\n');
                end
            end
        end
        fprintf('\n\n');
    end

    % Display experimental BER for each trial
    if (displayBER)
        fprintf('Trial %d: Channel BER = %.6f\n', trialNum, berChan);
        fprintf('Trial %d:    Data BER = %.6f\n\n', trialNum, berData);
    end

    % Print received text
    if (printText)
        WORD_NUM = 11;
        rxStr = [];
        for i = 1:WORD_NUM
            receivedSymbol = B{i};% Input received symbols, assume 2 chars are for 1 letter
            %binArray = reshape(txMsgBinary
            char1 = hex2dec((receivedSymbol(5:6)));
            char2 = hex2dec((receivedSymbol(7:8)));
            char3 = hex2dec((receivedSymbol(9:10)));

            if length(rxStr) >= WORD_NUM*3 % 33
                rxStr = char(char1,char2,char3)'; %empty the previous and start a new word
            else
                rxStr = [rxStr,char(char1,char2,char3)']; % keep going on the next symbol
            end
```

```
        end
209         fprintf('Trial %d: Message is: %s\n', trialNum, rxStr)

211     end

213 end
```

Listing 25: matlabCode/oneTransBerCalc.m

```
1 % FYP UMSP Convolutional encoding function
 % Code written by Callum Maltby (539190)
3 % Code started 18/1/2016 and finished on 18/1/2016

5 % The parameters are: k = 4 and R = 1/2

7 function output = convEncoder(input)
      % Define parameters particular to this standard implementation
9     maskA = [1, 0, 0, 1];
      maskB = [1, 1, 1, 1];
11    m = length(maskA);
      state = zeros(1,m);
13
      % Add zeroes to the input to 'flush' the registers with zeroes
15    input = [input, zeros(1,m)];
      output = zeros(1, 2*length(input));
17
      % Process every input bit
19    for i = 1:length(input)
          % Shift the input signal through the register state array
21        state = [input(i), state(1:m-1)];
          %fprintf('%d ', bi2de(state,'left-msb'));
23        % Produce vectors of the binary inputs to the A and B modulo 2adders
          outAVect = state & maskA;
25        outBVect = state & maskB;
          % Compute the binary outputs of these adders
27        output(2*i-1) = rem(sum(outAVect), 2); % outA
          output(2*i) = rem(sum(outBVect), 2);   % out B
29    end
  end
```

Listing 26: matlabCode/convEncoder.m

```
 % FYP UMSP Convolutional encoding function
2 % Code written by Callum Maltby (539190)
 % Code started 13/1/2016 and finished on dd/mm/2016
4
```

```
% The input code block shall be decoded back to the original message. The
% parameters are: k = 7 and R = 1/2

function output = convDecoder_v3(input, numStates)

    % These lookup tables will be stored in memory
    VIT_TUPLES = [0,3;1,2;1,2;0,3;3,0;2,1;2,1;3,0];
    TUPLE_COSTS = [0,1,1,2;1,0,2,1;1,2,0,1;2,1,1,0];
    VIT_STATES = [0,1;2,3;4,5;6,7;0,1;2,3;4,5;6,7];

    len = length(input);
    % Make space to assign the state path markers
    backwardPoint = zeros(numStates, len/2);
    % In the simplest form, hold all state's path metrics
    PM = zeros(numStates, 2);
    for k = 2:numStates
        PM(k, 1) = Inf;
    end

    currentPMCol = 2;
    prevPMCol = 1;

    for i = 2:len/2
        % Scan all possible states
        % The -1 and +1 will be removed for C
        currentPMCol = ~(currentPMCol-1) + 1;
        prevPMCol = ~(prevPMCol-1) + 1;
        % Compute the decimal representation of the two-bit input tuple.
        % Each of these tuples need to be incremented by one for use as an array index
        tuple = 2*input(2*i-3) + input(2*i-2)+1;
        for j = 1:numStates
            % Find the two possible previous states
            stateAlpha = VIT_STATES(j,1)+1;
            stateBeta = VIT_STATES(j,2)+1;
            % Evaluate the path metric costs
            costAlpha = TUPLE_COSTS(VIT_TUPLES(j,1)+1, tuple) + PM(stateAlpha, prevPMCol);
            costBeta =  TUPLE_COSTS(VIT_TUPLES(j,2)+1, tuple) + PM(stateBeta, prevPMCol);
            % Assign the backward pointer to the lowest cost path
            if ( costAlpha <= costBeta )
                backwardPoint(j, i) = stateAlpha;
                PM(j, currentPMCol) = costAlpha;
            else
                backwardPoint(j, i) = stateBeta;
                PM(j, currentPMCol) = costBeta;
            end
        end
        %PM
        %backwardPoint
```

```
    end

    % Find the end state with the lowest path metric
    min = 0;
    index = 1;
    for j = 1:numStates
        if PM(j, currentPMCol) < min
            index = j;
            min = PM(j, currentPMCol);
        end
    end

    % Decode the message using backwardPoint
    output = zeros(1,len/2);
    for k = len/2:-1:2
        output(k-1) = (index > 4);
        index = backwardPoint(index, k);
    end

end
```

Listing 27: matlabCode/convDecoder_v3.m

```
% FYP UMSP channel noise function
% Code written by Callum Maltby (539190)
% Code started 30/10/2015 and finished on dd/mm/yyyy

% The channel shall introduce binary bit errors, either correlated (bursty)
% or non correlated.

function [output, errors] = channelNoise(input, prob)
    errors = 0;
    for i = 1:length(input)
        if (rand() < prob) % An error occurred
            output(i) = xor(input(i), 1);
            errors =  errors + 1;
        else
            output(i) = xor(input(i), 0);
        end
    end
end
```

Listing 28: matlabCode/channelNoise.m

C.3 Antenna Ray Tracing

```
function varargout = antenna_GUI_cont(varargin)
% ANTENNA_GUI_CONT MATLAB code for antenna_GUI_cont.fig
%      ANTENNA_GUI_CONT, by itself, creates a new ANTENNA_GUI_CONT or raises the existing
%      singleton*.
%
%      H = ANTENNA_GUI_CONT returns the handle to a new ANTENNA_GUI_CONT or the handle to
%      the existing singleton*.
%
%      ANTENNA_GUI_CONT('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in ANTENNA_GUI_CONT.M with the given input arguments.
%
%      ANTENNA_GUI_CONT('Property','Value',...) creates a new ANTENNA_GUI_CONT or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before antenna_GUI_cont_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to antenna_GUI_cont_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help antenna_GUI_cont

% Last Modified by GUIDE v2.5 30-Sep-2015 17:01:08

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @antenna_GUI_cont_OpeningFcn, ...
                   'gui_OutputFcn',  @antenna_GUI_cont_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
```

```
% --- Executes just before antenna_GUI_cont is made visible.
function antenna_GUI_cont_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to antenna_GUI_cont (see VARARGIN)

% Choose default command line output for antenna_GUI_cont
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% This sets up the initial plot - only do when we are invisible
% so window can get raised using antenna_GUI_cont.
if strcmp(get(hObject,'Visible'),'off')
    plot(rand(5));
end

% UIWAIT makes antenna_GUI_cont wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = antenna_GUI_cont_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
axes(handles.axes1);
cla;
norm_scale = 0.002; % scales the normal vector displayed on the plot
ratio_sec_to_prim=str2double(get(handles.ratio,'String')); % handles.param.radius; % ratio of
    secondary reflector size to primary reflector size. default 0.1

% paraboloid parameters
d = str2double(get(handles.d,'String'));% default 0.85
radius = str2double(get(handles.r,'String')); % handles.param.radius; % in meters, default 0.237
```

```
f = 1/(4*d); % focal point of the parabola

% spherical parameters
r = radius;

% find angle psi
tanj_angle = atan(2*d*radius);
psi = 2*tanj_angle;

% ---------------------- Hyperboloid Design ----------------------------
h = str2double(get(handles.h,'String')); % handles.param.h;
feed_offset = str2double(get(handles.feedhorn_offset,'String')); %distance from base of parabola
    back to feed_horn focus
c = (f+feed_offset)/2; % distance from center of the hyperbola to the focus of the parabola.
a = h-(c-feed_offset); % we can vary a (the offset) to get a good design!! restricted by interior
    of cubesat, default 0.274

e = c/a; % eccentricity of the hyperbola
b = sqrt((c)^2-a^2);

% feeder angle requirement
theta = 2*atan(((e-1)*tan(psi/2))/(e+1))*180/pi

% generate the parabola based on the input parameters
x1 = -radius:0.001:radius;
y1 = d*x1.^2;

% calculate the volume of the parabola
i = round(length(x1)/2);
vol_p = 0;
while i < length(x1)
    vol_p = vol_p + x1(i)^2 * (y1(i)-y1(i-1));
    i = i+1;
end
vol_p = vol_p*pi

% generate the hyperboloid to match the paraboloid geometry
x4 = x1*ratio_sec_to_prim;
y4 = sqrt(a^2*((x4.^2)./b^2+1))+c-feed_offset;

% ---------------------- Ellipsoid Design ----------------------------
% generate the transparent sphere(ellipse) based on the input parameters
% (x/a)^2 + (y-H/b)^2 = 1

offset = d*radius^2;
y_hyp_max = y4(length(y4));
x_hyp_max = x4(length(x4));

```

```
xp = radius;
yp = d*radius^2

H = (2*d*yp*(x_hyp_max^2-xp^2)-(y_hyp_max^2-yp^2))/(2*(d*(x_hyp_max^2-xp^2)-(y_hyp_max-yp)))
b_ellipse = sqrt((yp-H)^2-2*d*(yp-H)*xp^2)
a_ellipse = sqrt(xp^2-(yp-H)/(2*d))

y2 = offset:0.0001:(b_ellipse+H); % from the horizontal axis to the top of the ellipse
x2 = a_ellipse*sqrt(1-((y2-H)/b_ellipse).^2);
x3 = -a_ellipse*sqrt(1-((y2-H)/b_ellipse).^2);

% calculate the volume of the ellipsoid
i = 2;
vol_e = 0;
while i < length(y2)
    vol_e = vol_e + x2(i)^2 * (y2(i)-y2(i-1));
    i = i+1;
end
vol_e = vol_e*pi

% volume of the antenna
fprintf('Volume of the inflatable antenna component (excluding feedhorn) is: %d m^3\n %d mL\n',
    vol_e+vol_p, 100^3*(vol_e+vol_p));

axes(handles.axes1);
hold on
par_obj = plot(x1,y1,'k','LineWidth',4);
sph1_obj = plot(x2,y2,'r',x3,y2,'r','LineWidth',4);
hyp_obj = plot(x4,y4,'k','LineWidth',3);
axis([-r-0.1,r+0.1,0,r+d*r^2+0.1]);

PAR = [x1' y1'];
HYP = [x4' y4'];
ELL = [x2' y2'];
csvwrite('PAR.csv',PAR);
csvwrite('HYP.csv',HYP);
csvwrite('ELL.csv',ELL);
% ---------- Ray Tracing Model Verification ----------
j=1;
for i = -radius:0.039:radius
    % plot the incident rays
    ray_x = i;
    if (ray_x<radius/10)&&(ray_x>-radius/10)
        continue;
    end
    ray_yint = d*ray_x^2;
    x = [ray_x ray_x];
    y = [ray_yint 10];
```

```
186     plot(x,y,'g','LineWidth',2);

188     % find the reflected rays
        angle = atan(2*d*ray_x);
190     rot_M = [cos(-2*angle) sin(-2*angle); -sin(-2*angle) cos(-2*angle)];
        ROT_COORD = rot_M*[(x-ray_x);(y-ray_yint)]; %need to translate before rotating
192     x_rot = ROT_COORD(1,:)+ray_x;
        y_rot=ROT_COORD(2,:)+ray_yint;
194
        % plot the reflected rays
196     plot(x_rot,y_rot,'b','LineWidth',2);

198     % find the intersections of the reflected rays
        L1 = [x4;y4];
200     L2 = [x_rot; y_rot];
        intX=InterX(L1,L2);
202     if isempty(intX)
            continue;
204     else
            % plot(intX(1),intX(2),'ro');
206     end


208


210     % determine the angle of incidence on the hyperbola
        hyp_tanj = intX(1)/(intX(2)-(c-feed_offset))*a^2/b^2; % dy/dx = x/y*b^2/a^2
212     hyp_angle = atan(-1/hyp_tanj); % angle of the normal vector
        u = [cos(hyp_angle) sin(hyp_angle)]; % unit vector describing normal of hyperbola
214     quiver(intX(1),intX(2),norm_scale*u(1),norm_scale*u(2),'k');
        v = [(x_rot(2)-x_rot(1)),(y_rot(2)-y_rot(1))]; % vector describing incident ray
216     v=v/sqrt(v(1)^2+v(2)^2); % make v a unit vector.
        inc_angle = acos(u*v'); % dot product formula to calculate angle between
218
        % generate the second set of reflected rays
220     x_feed = [ray_x x_rot(2)];
        y_feed = [ray_yint y_rot(2)];
222
        rot_M = [cos(-2*inc_angle) sin(-2*inc_angle); -sin(-2*inc_angle) cos(-2*inc_angle)];
224     ROT_COORD = rot_M*[(x_feed-x_rot(2)); (y_feed-y_rot(2))];
        x_feed = ROT_COORD(1,:) + intX(1);
226     y_feed = ROT_COORD(2,:) + intX(2);
        plot(x_feed,y_feed,'m','LineWidth',2);
228
        j=j+1;
230 end

232 title('UMSP Cassegrain Antenna');
```

```
% ---------------------- CALCULATE THE 2D cuts ----------------------

a = a_ellipse; % major ellipsoid axis, symmetric in the x and y directions.
b = b_ellipse; % minor ellipsoid axis in the z directions.
h = 0.000001;
N= str2double(get(handles.N,'String')); % The number of sections used to construct the antenna

DIST_TO_PARAB = H-yp; % the distance from the center of the ellipsoid to the top of the parabolic
    dish.

dist(1) = 0;
j=2;

% calculates the numeric solution to the line integral of an ellipse from
% the x-y plane to height, b.
for z=(-DIST_TO_PARAB+h):h:(b-h) % at x=a f_t would evaluate to infinity. Stop 1 sample before
    this.
    f_t = sqrt(1+z^2*a^2/(b^2*(b^2-z^2)));
    dist(j) = dist(j-1) + f_t*h;
    j=j+1;
end

z = (-DIST_TO_PARAB):h:(b-h);
% figure;
% plot(z,dist);
% xlabel('z (m)');
% ylabel('Mapped length of cut (m)');
% str = sprintf('For a step size, h = %d',h);
% title(str);


h2 = 0.0001; % resolution of the section's cut.
j=1;
H = dist(length(dist)) % height of the section

for c = 0:h2:H;
    % need z(c) to input into the equation for width of section at height
    % z.
    i = 1;
    while (dist(i)<c)
        i = i + 1;
    end

    % get the z value from the index: linear mapping from i.
    i_offset = length(-DIST_TO_PARAB:h:0);
    z(j) = (i-i_offset)*h;

```

```
    width(j) = 2*pi*a*sqrt(1-z(j)^2/b^2)/N; % compute the width of the section at a given height,
    z.
    j = j+1;
end

c = 0:h2:H;
% figure;
% plot(width,c)
% xlabel('width (m)');
% ylabel('height (m)');
% str = sprintf('Width of %d cut(s) vs height', N);
% title(str);

lhs_net = -width/2;
rhs_net = width/2;
figure(1);
cla;
hold on;
plot(lhs_net,c);
plot(rhs_net,c);
csvwrite('env.txt',[rhs_net',c']);
str = sprintf('One of N=%d 2D cuts to produce an ellipsoid with a=b=%2.2d, c=%2.2d',N,a,b);
title(str);
xlabel('m');
ylabel('m');
hold off;

% generate the paraboloid cuts

zp = d*radius^2;

len_par(1) = 0;

h = 0.000001;
j=2;
for z = (0+h):h:zp
    f_z = sqrt(1+1/(4*d*z));
    len_par(j) = f_z*h + len_par(j-1);
    j=j+1;
end

z = 0:h:zp;
% figure;
%plot(z,len_par);
%title('Mapping of z to line length of parabola segment');
%xlabel('m');
%ylabel('m');
```

```
h2 = 0.0001; %resolution of the paraboloid's cut. 1/10th of a mm.

H_P = len_par(length(len_par)) % height of the section
 j = 1;
for c = 0:h2:H_P;
    % need z(c) to input into the equation for width of section at height
    % z.
    i = 1;
    while (len_par(i)<c)
        i = i + 1;
    end

    % get the z value from the index: linear mapping from i.
    z = (i-1)*h;

    width_p(j) = 2*pi*sqrt(z/d)/N; % compute the width of the section at a given height, z.
    j = j+1;
 end
c = 0:h2:H_P;
 length(width_p)
length(c)

%figure;
 %plot(width_p,c);
% str = sprintf('Material width vs height for N=%2d sections',N);
 % title(str);
% xlabel('m');
 % ylabel('m');
lhs_par = -width_p/2;
 rhs_par = width_p/2;

 figure(2);
cla;
 hold on;
plot(lhs_par,c);
 plot(rhs_par,c);
csvwrite('refl.txt',[rhs_par',c']);
 hold off;
str = sprintf('One of N=%2d 2D cuts required to make a paraboloid of radius r=%2.2d and curvature
    d = %2.2d',N,radius,d);
 title(str);
xlabel('m');
 ylabel('m');



% ------------------------------------------------------------------
```

```
function FileMenu_Callback(hObject, eventdata, handles)
% hObject    handle to FileMenu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --------------------------------------------------------------------
function OpenMenuItem_Callback(hObject, eventdata, handles)
% hObject    handle to OpenMenuItem (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
file = uigetfile('*.fig');
if ~isequal(file, 0)
    open(file);
end

% --------------------------------------------------------------------
function PrintMenuItem_Callback(hObject, eventdata, handles)
% hObject    handle to PrintMenuItem (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
printdlg(handles.figure1)

% --------------------------------------------------------------------
function CloseMenuItem_Callback(hObject, eventdata, handles)
% hObject    handle to CloseMenuItem (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
selection = questdlg(['Close ' get(handles.figure1,'Name') '?'],...
                     ['Close ' get(handles.figure1,'Name') '...'],...
                     'Yes','No','Yes');
if strcmp(selection,'No')
    return;
end

delete(handles.figure1)


% --- Executes on selection change in popupmenu1.
function popupmenu1_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns popupmenu1 contents as cell array
%        contents{get(hObject,'Value')} returns selected item from popupmenu1


```

```
% --- Executes during object creation, after setting all properties.
function popupmenu1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

set(hObject, 'String', {'plot(rand(5))', 'plot(sin(1:0.01:25))', 'bar(1:.5:10)', 'plot(membrane)',
    'surf(peaks)'});


% --- If Enable == 'on', executes on mouse press in 5 pixel border.
% --- Otherwise, executes on mouse press in 5 pixel border or over pushbutton1.
function pushbutton1_ButtonDownFcn(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on slider movement.
function slider1_Callback(hObject, eventdata, handles)
% hObject    handle to slider1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider


% --- Executes during object creation, after setting all properties.
function slider1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to slider1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end


% --- Executes on slider movement.
function slider2_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to slider2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider


% --- Executes during object creation, after setting all properties.
function slider2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to slider2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end


% --- Executes on slider movement.
function slider4_Callback(hObject, eventdata, handles)
% hObject    handle to slider4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider


% --- Executes during object creation, after setting all properties.
function slider4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to slider4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end


% --- Executes on slider movement.
function slider5_Callback(hObject, eventdata, handles)
% hObject    handle to slider5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider


% --- Executes during object creation, after setting all properties.
function slider5_CreateFcn(hObject, eventdata, handles)
% hObject    handle to slider5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end



function d_Callback(hObject, eventdata, handles)
% hObject    handle to d (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of d as text
%        str2double(get(hObject,'String')) returns contents of d as a double


% --- Executes during object creation, after setting all properties.
function d_CreateFcn(hObject, eventdata, handles)
% hObject    handle to d (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function h_Callback(hObject, eventdata, handles)
% hObject    handle to h (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of h as text
%        str2double(get(hObject,'String')) returns contents of h as a double
```

```
564
% --- Executes during object creation, after setting all properties.
566 function h_CreateFcn(hObject, eventdata, handles)
% hObject    handle to h (see GCBO)
568% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
570
% Hint: edit controls usually have a white background on Windows.
572%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
574     set(hObject,'BackgroundColor','white');
end
576


578
function r_Callback(hObject, eventdata, handles)
580% hObject    handle to r (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
582% handles    structure with handles and user data (see GUIDATA)

584% Hints: get(hObject,'String') returns contents of r as text
%        str2double(get(hObject,'String')) returns contents of r as a double
586


588% --- Executes during object creation, after setting all properties.
function r_CreateFcn(hObject, eventdata, handles)
590% hObject    handle to r (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
592% handles    empty - handles not created until after all CreateFcns called

594% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
596 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
      set(hObject,'BackgroundColor','white');
598 end


600


602 function ratio_Callback(hObject, eventdata, handles)
% hObject    handle to ratio (see GCBO)
604% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
606
% Hints: get(hObject,'String') returns contents of ratio as text
608%        str2double(get(hObject,'String')) returns contents of ratio as a double


610
% --- Executes during object creation, after setting all properties.
```

```
function ratio_CreateFcn(hObject, eventdata, handles)
% hObject    handle to ratio (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function N_Callback(hObject, eventdata, handles)
% hObject    handle to N (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of N as text
%        str2double(get(hObject,'String')) returns contents of N as a double


% --- Executes during object creation, after setting all properties.
function N_CreateFcn(hObject, eventdata, handles)
% hObject    handle to N (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function feedhorn_offset_Callback(hObject, eventdata, handles)
% hObject    handle to feedhorn_offset (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of feedhorn_offset as text
%        str2double(get(hObject,'String')) returns contents of feedhorn_offset as a double


% --- Executes during object creation, after setting all properties.
function feedhorn_offset_CreateFcn(hObject, eventdata, handles)
% hObject    handle to feedhorn_offset (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

Listing 29: matlabCode/antenna_GUI_cont.m