A Reusable Signal Processing Architecture for Satellite Based Communication Systems

by

Jakobus Stephanus Botha

Thesis presented in partial fulfilment of the requirements for the degree of Master of Science in Electronical Engineering at Stellenbosch University

Supervisor: Dr G-J van Rooyen
Department of Electrical and Electronic Engineering

March 2011

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Date: March 2011

Abstract

Keywords: digital signal processing, embedded systems, telecommunications, satellite technology

The rapid growth of the telecommunications industry is a worldwide phenomenon with people and computers generating and transmitting more and more information daily. Despite this growth, there are still areas in South Africa which lack terrestrial communications coverage. People inhabit these rural areas and their essential communication needs are not met. Satellite based communication coverage can provide a valuable service in these circumstances.

In this thesis, the design of a satellite-based communications payload – which makes use of software defined radio techniques – is presented in terms of the Open Systems Interconnect layer structure. A robust hardware platform using a space-qualified on-board computer, a Xilinx Virtex-5 Field Programmable Gate Array (FPGA) and a Freescale digital signal processor (DSP) is designed, implemented and thoroughly tested. A device driver is designed for hardware and firmware components. A prototype ground station is also designed and constructed using a low-power PC, a Xilinx Spartan-3E FPGA, a Freescale DSP and radio frequency hardware.

A wide range of testing methodologies were successfully utilised to deploy a functional system which is critically evaluated in the last chapter.

Uittreksel

Sleutelwoord: syferseinverwerking, toegewyde stelsels, telekommunikasie, satelliettegnologie

Die vinnig groeiende telekommunikasieindustrie is 'n wêreldwye verskynsel waarin mense en rekenaars daagliks meer en meer data genereer. Ten spyte van die groei, is daar nog steeds gebiede in Suid-Afrika wat aan 'n gebrek van aardse kommunikasiedekking lei. Mense bewoon dié areas maar daar word nie aan hul noodsaaklike kommunikasiebehoeftes voldoen nie. Satelliet-gebaseerde kommunikasiedekking kan 'n waardevolle diens in hierdie omstandighede wees.

Hierdie tesis beskryf die ontwerp van 'n ruimtegebaseerde kommunikasieloonvrag – wat gebruik maak van sagteware-gedefinieerde radiotegnieke – aangebied in terme van die Open Systems Interconnect laagstruktuur. 'n Robuuste apparatuurplatform wat gebruik maak van 'n ruimte-gekwalifiseerde rekenaar, 'n Xilinx Virtex-5 Veldprogrameerbare Hek-Skikking (VPHS) en 'n Freescale syferseinverwerker is ontwerp, geïmplementeer en deeglik getoets. 'n Toestelbestuurder moes ontwerp word vir die apparatuur- en fermatuur-komponente. 'n Prototipe grondstasie is ook ontwerp en gebou met behulp van' n lae-krag PC, 'n Xilinx Spartan-3E VPHS, 'n Freescale seinverwerker en radiofrekwensie apparatuur.

'n Wye verskeidenheid van toetsmetodes is suksesvol benut om 'n funksionele stelsel te ontwikkel wat krities geëvalueer word in die laaste hoofstuk.

Acknowledgements

- Most of my thanks goes to Dr G-J van Rooyen for his inspiration and guidance as my supervisor.
- I also extend heartfelt thanks to the Department of Communications of the South African government for showing a strong interest in extraterrestrial communications systems and for initialising this project.
- I also acknowledge my friend Adrian Cooke for introducing me to Dr Riaan Wolhuter after a particularly pleasant hike in Stettynskloof.
- I would like to thank Dr Wolhuter for his support during the project. His many interesting stories and his wisdom provided me with inspiration during my studies.
- I would also like to thank the many friends I've made in the E&E faculty. Some inspiring and thought provoking lunches have helped me grow as a person. Especially those with Albert Visagie, Pieter Holtzhauzen, Charl Botha and Edward de Villiers.
- My parents deserve a lot of my thanks for their continued support during my studies.
- Lastly I would like to thank my colleague Ewald van der Westhuizen for the long hours of team work during the project's integration phase.

Contents

List of Figures

List of Tables

Nomenclature

Acronyms

ADC	Analogue-to-Digital Converter
ASE	Aircraft-to-Satellite Emulator
CAN	Controller Area Network
CMOS	Complementary Metal-Oxide-Semiconductor
CPG	Clock Pulse Generator
CPU	Central Processing Unit
DAC	Digital-to-Analogue Converter
DRAM	Dynamic Random Access Memory
DSP	Digital Signal Processor
EDC	Error Detection and Correction
EMC	Electromagnetic Compatibility
EMI	Electromagnetic Interference
FFT	Fast Fourier Transform
FIFO	First In First Out
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
I/Q	In-phase/Quadrature
ISM	Industrial, Scientific and Medical
ISR	Interrupt Service Routine
KUL	Katholieke Universiteit Leuven
LEO	Low Earth Orbit
LNA	Low Noise Amplifier
LO	Local Oscillator
LVCMOS	Low Voltage CMOS
LVDS	Low Voltage Differential Signaling
MCCPL	Multiple Channel Communications Payload
OBC	On Board Computer
PA	Power Amplifier
PLL	Phase Locked Loop
POSIX	Portable Operating System Interface for Unix
QPSK	Quadrature Phase Shift Keying
RF	Radio Frequency

SAA	Steerable Antenna Array
SH4	SuperH-4
SIC	Signal Interface Card
SRAM	Static Random Access Memory
SPI	Serial Peripheral Interface
TM	Telemetry
UART	Universal Asynchronous Receiver/Transmitter
USB	Universal Serial Bus
VHDL	VHSIC Hardware Description Language
VHSIC	Very-High-Speed Integrated Circuits

Terms of Reference

This project began during a commissioning by the Department of Communications of the South African government – with the University of Stellenbosch as contractors – for a multiple channel communications system that would possibly provide telecommunications coverage in remote areas of South Africa. It was to be implemented as a payload on a low earth orbit satellite.

The project funding was canceled by the Department of Communications after the first year and the project continued in academic form. The Katholieke Universiteit Leuven (KUL) subsequently expressed interest in developing a payload for a low earth orbit satellite using a Steerable Antenna Array (SAA). Much of the development done for the Department of Communications could be used as-is for the KUL project.

Specific constraints were placed on the project by KUL and the project supervisors:

- two Xilinx field programmable gate array boards, an ML501 and a Spartan-3E were to be used for the research done by Francois Olivier into low density parity check codes,
- a Freescale DSP developement kit was to be used for the signal processing tasks as the hardware and design experience was available within the group,
- a Renesas SH4, from the engineering model of Sumbandila Satellite's communications payload, was to be used as on-board computer as it was immediately available and supported by Sun Space,
- the space readiness in terms of radiation tolerance and vibration resistance of the hardware was only to be investigated during a subsequent phase of the project,
- KUL would supply the steerable antenna array which was to be tested,
- a minimum data rate of 19 200 bits per second was required by KUL for the communications link and
- the error correction code rate used by Francois Olivier would double the effective data baud rate, resulting in a physical layer baud rate of 38 400 bits per second.

The specific objectives identified for the part of the project documented in this thesis, were:

- physical integration and interconnection of all hardware components,
- low level software design on the Renesas SH4 to provide software interfaces to off-board hardware resources,
- low level software design on the Freescale DSP to provide a software interface to the external hardware,
- firmware design on the Xilinx FPGAs to enable interfacing with external components,
- radio frequency component selection to enable transmission of data over the channel and
- design of a prototype ground station which would be used for testing of the SAA.

Chapter 1

Introduction

1.1 Project Background

The rapid growth of the telecommunications industry is a worldwide phenomenon with people and computers generating and transmitting more and more information each day. Despite this rapid growth, there are still areas in South Africa without communications coverage. People inhabit these rural areas, yet certain essential communication needs are often not met. For example, rural hospitals may find it very difficult to communicate with specialists or suppliers in urban areas. Low-cost, low-bandwidth satellite communications coverage can be a valuable tool in these circumstances. Such a communications system can also, for example, play a central role in the transfer of aggregated data from distributed data capturing units.

SUNSAT, the first locally built satellite, was an Orbital Satellite Carrying Amateur Radio which used frequencies in the Very High Frequency and Ultra High Frequency bands for communication with terrestrial ground stations [6]. The second locally built satellite, Sumbandila, has a low-cost, low-bandwidth communications system on board [12].

The communications payload on the Sumbandila satellite was designed and implemented by Stellenbosch University (SU) and Sun Space and Information Systems with the Department of Communications (DoC) of the South African government as client. The system consists of an On Board Computer (OBC) and a VHF-UHF Communications Unit (VUCU). The VUCU is a full duplex communications unit using separate frequency bands for the uplink and the downlink, the design of which was heavily influenced by SUNSAT. However, the frequencies allocated by the International Telecommunications Union (ITU) and the Independent Communications Authority of South Africa for the uplink and downlink are roughly 0.2 MHz apart and the unit is unable to communicate in full-duplex mode as the transmitted signal's power is much higher than the received signal's, therefore swamping the receiver. Half duplex therefore had to be implemented in software. This meant that the effective

data throughput rate of the payload fell to under 4800 bps, which allows only fairly short text messages to be sent.

Also aboard Sumbandila is an experimental payload consisting of various scientific experiments [45]. Among the experiments are:

- An instrument to measure the Earth's atmospheric phenomena below 30 kHz, developed by the University of KwaZulu-Natal,
- The Southern African Amateur Radio Satellite Association has placed an amateur radio service on board which communicates in voice beacon, digipeater and parrot modes,
- The Nelson Mandela Metropolitan University has an experiment to measure the non-linear vibration effects of strings in microgravity and
- Stellenbosch University has an experiment on board which will measure the effects of radiation on electronics.

Central to these experiments was the Stellenbosch University's Software Defined Radio (SDR) group's Signal Interface Card (SIC) which controls the electronics and provides an interface between the experiments and an OBC running Linux. The results of the experiments are then transmitted to earth via a special version of the VUCU.

The DoC then commissioned SU to design a broadband communications payload for possible use on a future satellite. The requirement from the DoC was a high-speed, multi-channel communications system that could be used with low-cost ground stations that could be deployed throughout the country. Three phases spanning three years were alloted for the completion of the project.

In the first year a technological proof of concept of the payload was to be built. During the second year an engineering model would be built. The final phase would see rigorous space readiness testing being done with a flight model being built as final deliverable.

1.2 Project Evolution

Before the end of the first phase the DoC canceled the project. This happened before the research was completed and put an end to the funding which meant the project continued in an academic form from then on.

Shortly thereafter KUL approached Stellenbosch University with a proposed project to test their SAA on an LEO satellite for their In-Situ Hyper Spectral II (IS-HSII) project. KUL has a potentially novel approach to beam steering from space and they needed a platform on which they could test their

antenna. An agreement was reached that Stellenbosch University would provide KUL with a hardware platform which they could used for testing their SAA.

Building space compliant hardware was not possible because there was no existing satellite specification; there were no system constraints such as weight and available space from which a design could begin.

The solution to this is to use Stellenbosch University's Jora lightweight two seat glider. A prototype payload platform would be constructed and used to the test the SAA on the Jora. The prototype would use space compliant hardware where possible but full space compliance would only be considered after the first successful aircraft flight test.

Due to the similarities between the MCCPL and the KUL SAA payload it was decided that the MCCPL architecture would, for the most part, be a good starting point for the design and was used as such. It was also decided that the payload would be a good platform on which to continue the SDR modem design.

1.3 Proposed Solution

Much of the hardware and expertise required to implement the KUL SAA testing platform was already available within the group. This was in part due to the MCCPL project. The MCCPL system design was therefore an ideal starting point for the design of the SAA platform. A modified version of the MCCPL architecture was presented to Katholieke Universiteit Leuven by F. Olivier, G-J van Rooyen and R. Wolhuter [47].

The proposal listed the following as key components:

- an aircraft-to-satellite emulator (ASE), which would translate Jora avionics (yaw, pitch and roll) into satellite telemetry,
- the SH4 on-board computer, a space qualified processor unit,
- a software defined radio (SDR) modem to enable data communication and
- a ground station which would communicate with the SAA platform.

The proposal was accepted by KUL. It contained specifications which provided clear objectives for the platform. Among the specifications are:

- the RF operational band used would be the S-band portion of the Industrial, Scientific and Medical (ISM) band,
- a minimum baud rate of 19 200 bps was required for the communications link,

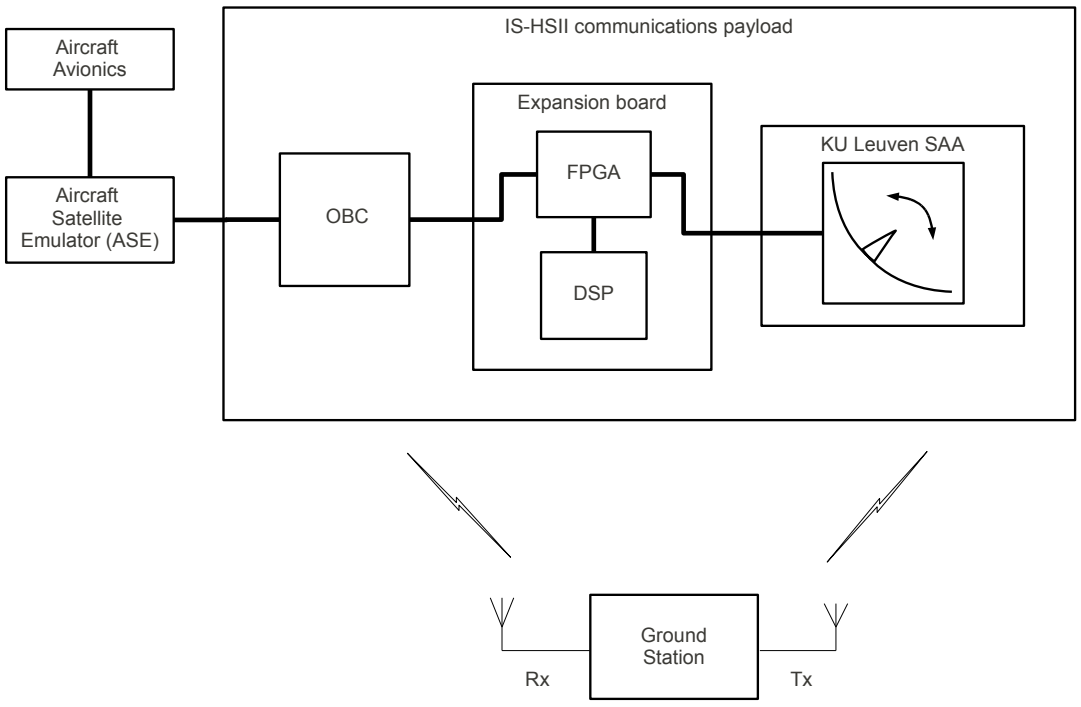

Figure 1.1: Overview of the SAA platform.

- a low-density constellation modulation method such as Quadrature Phase Shift Keying (QPSK) would be used for the SDR modem design,
- the project was to include aircraft borne flight hardware and
- Stellenbosch University would develop and supply a ground station for communicating with the payload.

Due to the size of the project, a team of engineers was required for the project. The work done which formed a part of this thesis is clearly outlined in subsequent chapters. An overview of the airborne platform together with a ground station is presented in Figure 1.1. Reconfigurability was kept in mind during the course of the thesis to allow reuse of developed components.

1.4 Structure of This Thesis

In Chapter 2 a methodology for complex system design and implementation is derived. A useful model for explaining concepts throughout the thesis is presented. The methodology is then used to implement a high level design of the payload and ground station systems.

Chapter 3 provides a more in-depth view of the platform overview presented in Figure 1.1. The detailed design of the payload is also presented

in this chapter. Software and hardware components are designed from system constraints and low level schematics are presented. Chapter 4 follows the same procedure and provides the same level of detail for the ground station system.

In Chapters 5 and 6 the implementation of the payload and ground station systems are presented. Low level smoke tests are performed to verify simple implementations where necessary and these tests are documented here. Limited integration between components is done.

The integration of the subsystems is then discussed in Chapter 7. The final integration tests are performed and documented in this chapter. A high level ground-to-air simulation test is also described in this chapter.

In Chapter 8 the results of the system tests are summarized. The testing results are critically analyzed and recommendations to noted problems are made based on the analysis. Future work which can be done on the project is also presented.

Chapter 2

Methodology and Perspective

In this chapter an overview of the platform required by KUL for the testing of their antenna is presented. The platform's intended use is as a communications payload on an LEO satellite. As stated in the terms of reference, however, stringent requirements for harsh space conditions were not considered extensively. Nonetheless, the system is referred to as a payload throughout this text.

The payload is split into subsystems, where possible, to allow modular design, implementation, and testing; the subsystems are all presented in this chapter and are used to separate the work which formed part of this thesis from work which was done by others.

Firstly a useful model is presented to the reader to assist in the description of the subsystems during subsequent chapters. Thereafter an overview of the payload is presented. The satellite earth station, also referred to as the ground station, is then described.

2.1 Methodology

The methodology derived in this section describes the design process, the implementation process and the testing process, all of which are presented in subsequent chapters.

2.1.1 Outline

The design of a new communications system is a complex process. There is usually not a predefined sequence of steps to follow which lead you to a correct or optimal solution. There are, however, certain system constraints (stated in the terms of reference) which form a base from which the design can begin. Furthermore, there exist specific outcome criteria; a set of functional requirements for the payload are known.

The outcome criteria are abstract concepts – top level specifications which do not entail implementation details. Since the objective of this thesis is ultimately the construction and testing of an operational system, implementation details need to be considered extensively. Based upon these two givens (known top level specifications and required real world implementation) a work methodology for the entire thesis is formed.

2.1.2 Derivation

When implementing a system, all components should be smoke tested before integration. This is to cater for components that do not perform according to specification. From this, it is evident that the most effective methodology for this thesis is top-level design followed by bottom-up implementation.

Once the lowest level subsystems have been tested successfully, integration begins. After each step of integration a new subsystem is formed. This subsystem must then be tested. This iterative process continues until the entire system has passed an end-to-end test and is performing according to specification. There is a feedback path from the implementation stage whereby the top-level design can be updated if, for example, a component does not perform as expected.

2.1.3 Presentation

The derived methodology is strictly adhered to throughout the thesis. The detailed level design of the entire system down to component level was performed first and is presented in the two design chapters. Each design chapter contains a detailed figure of the design which clearly distinguishes between the parts of the system that were implemented as a part of this thesis and the parts that were implemented by others. The implementation of the bottom level components is described in the two implementation chapters. The integration and end-to-end system testing is described in the final testing chapter.

2.2 OSI Model

The International Standards for Organisation developed a model called the Open Systems Interconnect (OSI) model as a tool for subdividing communications systems [27]. The model consists of divisions called layers; the layers are stacked in a consecutive hierarchy. Each layer interfaces with the layers above and below it: it provides services to layers other than itself and it requests "services" from other layers. This is useful in partitioning designs, developing robust independent blocks, and implementing top-down or bottom-up design.

For this project a simplified version of the OSI Model that does not include the session, presentation and network layers is used. This is because full com-

munications functionality and testing of the SAA is provided without them. The simplified model is depicted in figure 2.1. In summary, the layers are:

- **Physical Layer**: At the bottom of the OSI Layer is the physical layer. This layer is described in terms of the raw transmission of bits between nodes in the communication network. Logical concepts such as frames and packets do not exist at the physical layer. Furthermore, not all physical devices and interfaces form part of the physical layer, only those that form the lowest layer of transmission.
- **Data Link Layer**: The Data Link Layer is the first layer above the physical layer and is the layer which builds frames from data words. The Data Link Layer is the protocol layer which transfers data between nodes. It might provide the ability to detect and correct errors that occurred in the physical layer.
- **Transport Layer**: The Transport Layer provides end-to-end communication services for applications. It also provides services such as reliability, i.e. it ensures that data from the application layer is transmitted reliably.
- **Application Layer**: High level system and user applications reside in the application layer. The application layer has no concept of physical error correction, low level data words or frames. It uses logical addressing.

2.3 Top Level Payload Design

This section uses the two top level payload objectives (of communicating with a system in space and testing the SAA) to provide a setting for the rest of the

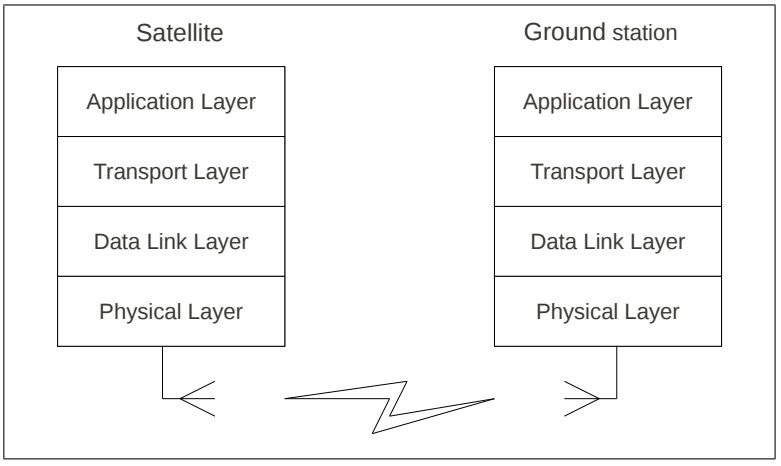

Figure 2.1: A simplified OSI model.

thesis. The objectives of the payload place certain constraints on the design. The purpose of the payload is twofold: firstly it must serve as a platform on which KUL's SAA can be tested. Secondly, the payload must enable the transmission of useful data between ground stations i.e. it must function as a communications payload. These two objectives are now described in detail.

2.3.1 Testing of the KUL Antenna

The SAA subsystem is supplied by KUL. It is depicted in Figure 2.2. The SAA itself is intended to be used as a receiver and consists of $m \times n$ individual receiving elements. Each element is a Right-Hand Circularly Polarised (RHCP) antenna with a wavelength of $416.\dot{6} \times 10^{-12}$ m. Digital Beam Steering (DBS) techniques are used to alter the radiation pattern of the array to improve the performance of the RF link. The DBS calculations are done on the digitised signal by an FPGA with an Ethernet interface.

The beam steering is controlled by sending x-y coordinates to the steering control logic on the FPGA which performs phase and gain calculations to create a radiation pattern with a peak at the received coordinates. A transmitter must therefore be designed to transmit an RF signal to the SAA. Testing will then involve varying the angle of incidence between this transmitter and the SAA over time and measuring the gain of the SAA while altering its radiation pattern.

Ultimately the SAA will be on board a Low Earth Orbit (LEO) satellite. The orbital path of the satellite results in a varying angle of incidence as the

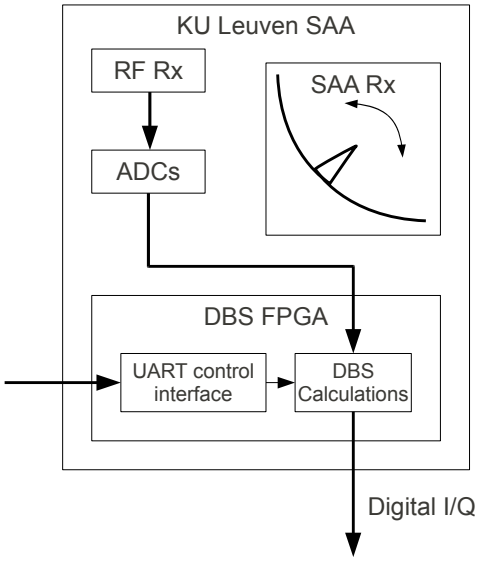

Figure 2.2: The SAA subsystem.

satellite passes overhead. Whilst simple tests to vary the angle of incidence of the received RF signal can be carried out over long distances between two terrestrial points, a more realistic test would be to fly overhead with the payload mounted on an aircraft. This kind of test would reconstruct signal attenuation near the horizon.

It was recognised that to test the payload on an aircraft would require a satellite emulator to convert avionic data into orbital data. This aircraft emulator was designed and implemented by Iwan Kruger [31].

2.3.2 Communications Link

The SAA implements quadrature mixing to achieve greater spectral efficiency and make the system less prone to channel noise. This means that the received RF signal is demodulated into separate In Phase (I) and Quadrature (Q) components. The I and Q signals are then digitised by a pair of Analogue to Digital Converters (ADCs) in the SAA subsystem. These digitised I and Q signal samples then require further demodulation to extract the raw data bit stream. These samples are presented to external components via an Ethernet interface.

The two interfaces to the SAA subsystem are a bidirectional control interface for sending and receiving beam steering commands and an Ethernet interface which delivers the digitised signal samples. This means a processor is required to send commands over the serial interface and some type of demodulation communications hardware is required to demodulate the signal samples. The major subsystems required to implement the payload communications system are therefore:

- a processor,
- a modulator/demodulator and
- an FPGA.

The FPGA is used for signal routing and data marshaling between components. The modem used is an extension of the modem written by Ewald van der Westhuizen from the multiple channel communications payload [7]. The processor used is the SH4 On Board Computer from the Sumbandila Communications Payload [16].

Error Detection and Correction (EDAC) is used in communications systems to improve bit error rates. A data protocol is used to ensure reliable data transfer. EDAC methods for the payload were investigated by Francois Olivier [36]. These methods are currently being implemented by Riaan Wiid [46].

It was also noted that the scheduling system on Sumbandila's communications payload, written by H. R. Gerber and A. Cooke [12], used a suboptimal scheduling mechanism. If the orbital parameters (two line elements) and the

precise location of the ground station are known, it is possible to design an efficient scheduler. This was done by John Gilmore [22].

To implement a useful communications link, three more components are required:

- an uplink transmitter,
- a downlink transmitter and
- a downlink receiver

The downlink transmitter implementation had no direct effect on the testing of the SAA itself. Therefore an off-the-shelf data radio would be used to implement the downlink. The uplink transmitter and downlink receiver systems form part of the ground station, an overview of which will be presented next.

2.4 Ground Station Overview

The ground station's primary function is to transmit an RF signal to the payload in such a manner that the operation of the SAA can be tested accurately. A secondary function of the ground station is to transmit some kind of useful data over the communications link to another ground station. A specific list of functional system requirements will now be constructed from these broadly stated requirements.

The SAA acts as an RF receiver system centered at 2.45 GHz. It expects a waveform that is Right Hand Circularly Polarised (RHCP) – clockwise in the direction of propagation. The SAA performs quadrature down mixing and sampling to deliver baseband digital I and Q signals on its output. The functional system requirements for the ground station can now be listed. They are:

- the ground station must transmit an RF signal of the correct amplitude, frequency and polarisation,
- the RF signal must contain a complex I/Q message signal and
- the I/Q message signals must in turn contain modulated information.

These three specifications form the criteria against which all other design decisions will be made. These design decisions are further discussed in Chapter 4.

Chapter 3

Payload Design

In this chapter the detailed design of all the components on the payload is presented. In the first section an overview of the payload provides background for the subsequent sections. Each major component is presented in its own section and is followed by a detailed design in terms of the OSI model, where deemed useful to the discussion. The SDR modem, for example, exists purely on the physical layer, so its design will not be handled in terms of the OSI layers. Figure 3.1 is a diagram of the payload.

3.1 Payload Design Overview

The payload consists of individual interconnected components. To provide a clear picture of the payload's operation and the interconnection of the components, all components will be described briefly. Due to the large nature of the project, the implementation of certain components fell outside the scope of this thesis. These components and subsystems were the responsibility of other team members. This section describes exactly which components formed part of this thesis and which did not.

At the core of the payload is the SAA subsystem designed by KUL which must be tested. The SAA subsystem has the following components:

- the physical antenna array,
- RF receiver hardware for down mixing and demodulation,
- ADCs for sampling the baseband signals and
- an FPGA which performs beam steering calculations and provides interfaces to the greater payload.

The SAA interface uses a Ethernet physical layer chip [34] to transmit I/Q samples to the greater payload. The components on the payload that formed a part of this thesis are:

- data marshaling firmware which routes the digital I/Q samples from the SAA to the SDR modem via the FPGA,
- firmware to route the demodulated data from the SDR modem to the OBC,
- a device driver on the OBC acting as an interface to the FPGA and
- firmware to route the data from the OBC to the downlink data radio.

The remainder of the work that forms part of this thesis was done on the ground station, discussed in later sections.

3.2 The SH4 On-Board Computer

As stated in the terms of reference, the OBC to be used for this project was designed and built by SunSpace. It is space qualified and has withstood the harsh space environment on the Sumbandila Satellite which is in orbit at the time of writing [41]. The OBC is based on a 32-bit RISC processor, the Renesas SH7750R [39] (referred to as the SH4 in this thesis). The OBC features a dual redundant CAN interface, NAND flash non-volatile memory, a bi-directional Low Voltage Differential Signaling (LVDS) I/O port and a memory expansion port.

The actual OBC unit used for this thesis was the same OBC from the Sumbandila Communications Payload engineering model. It was loaded with the QNX operating system making it ready for use immediately. Although the Linux operating system has been ported to the SH4 OBC in [45], it was decided to use QNX for reasons explained below, despite the fact that running Linux on the OBC has certain advantages (such as code portability and development flexibility).

The decision to use QNX as operating system was based on the following two points. Firstly, QNX is a commercial operating system which means that professional support is available from QNX Software Systems. Secondly, other engineers would be involved in the development of the payload, so a reliable operating system was required. If the operating system required development it could cause delays in other research projects.

The SH4 OBC provides two serial ports which can be used. One of the serial ports, `/dev/ser1`, is permanently mapped to a terminal by the QNX image [17]. The second serial port is mapped to a serial device, `/dev/ser2`, which can be used as a general purpose Universal Asynchronous Receiver/Transmitter (UART), but which was used by Kruger's Aircraft Satellite Emulator to send beam steering commands to the SAA subsystem [31].

3.3 OBC Design in Terms of the OSI Model

A view of the software components and hardware interfaces required on the OBC are now presented. Most of the OBC-related work done for this thesis was on the physical layer of the OSI model, yet the application and data link layer components are briefly described to provide an understanding of the system.

3.3.1 Application Layer

System software running on the application layer is required to provide the communications payload with store and forward capability. Store and forward means that useful data can be transmitted between ground stations via the payload. When multiple ground stations are trying to communicate with the payload simultaneously, some type of scheduling system is required. The scheduling system implemented by H.R. Gerber and A. Cooke [12] simply chose a ground station from a master list at random until all ground stations were serviced. It was realised that, if the location of the ground stations and the orbit of the satellite is known, it is possible to design a much more efficient, potentially optimal scheduling system. This area of research did not form part of the work done in this thesis, however, and was done by J. Gilmore [22].

3.3.2 Transport Layer

Data from the application software must be transmitted (and received) over the channel. This is done by using a very simple transport protocol to ensure the reliable delivery of packets at the receiver's transport layer. The protocol used to accomplish this is the Automatic Repeat Request (ARQ) protocol, implemented by J. Gilmore and R. Wiid [21].

3.3.3 Data Link Layer

The ARQ packets from the transport layer must be sent to the physical layer. This is done by splitting the packets up into frames as per the Telecommand and Telemetry (TM) protocol designed by the European Cooperation for Space Standardization [18]. A single TM frame is 84 bytes in length. The TM protocol was implemented by F. Olivier, E. van der Westhuizen and R. Wiid [46].

3.3.4 Physical Layer

TM frames from the data link layer on the OBC need to be physically transported to the FPGA which relays them to the modem for modulation. A suitable electrical interface must be chosen between the OBC and the FPGA.

A software interface must be designed allowing other programs on the OBC to use this interface.

Electrical Interface

The SH4 OBC has the following physical interfaces:

- an LVDS I/O interface,
- two Controller Area Netwrok (CAN) bus interfaces,
- two serial ports and
- a memory expansion port.

Considerations for the design of the interfaces include: sufficient data rates, electrical compatibility between physical components and features such as addressing, interrupts and DMA.

The CAN bus interfaces are intended for connecting the "major components" of the satellite and are reserved for this purpose. Of the two serial ports, one is mapped to a Unix serial console which is useful for connecting to the SH4. The other serial port is reserved for sending control commands to the SAA. The LVDS I/O is useful for high speed point to point data transfers. While it could be used for our interface to the FPGA, it would require additional addressing wrappers because more than one address on the FPGA needs to be accessed. This leaves the expansion port, which, as is described below, meets the requirements.

The expansion port comes in the form of a PC-104+ connector into which a daughter board typically slots. This connector provides a 32-bit data bus which is memory mapped as external memory to the SH4's address space. It also provides an active low power up reset pin, $\overline{\text{RESET}}$, that allows the daughter board to reset to a known state at startup. There is also an interrupt pin, $\overline{\text{IRL3}}$, that allows the daughter board to generate hardware interrupts. Four chip select pins are also provided to allow multiple components to be connected to the expansion port on the same address and data bus.

All of the pins on the expansion port are connected to LVCMOS-3.3 buffers, the 74ALVC162245 from Fairchild Semiconductor [19]. These buffers then connect to the "outside world". This protects the SH4 from accidental damage.

The maximum data rate of the expansion port is a crucial consideration to ensure that data transfers between the OBC and the FPGA occur timeously. Calculating the theoretical maximum data rate of the expansion port is not trivial. The expansion port has its own clock which determines the speed at which data transfers occur. The maximum internal clock to bus clock ratio equals $\frac{1}{2}$. To determine the maximum internal clock speed, it is noted in [17] that the internal clock speed is determined by the external mode pins MD[2..0]. These pins have a fixed value of 011 and cannot be adjusted. From [39], Table

10.3(2), this means that the on-chip Phase Locked Loop (PLL) is activated with a multiplication factor of 12. The SH4 input clock pin is fed by a 16 MHz crystal. The internal clock frequency is therefore fixed at 192 MHz and the maximum bus clock speed is therefore 96 MHz.

The SH4 has a 32-bit virtual address space. The virtual address is divided into five areas according to the upper address value. External memory comprises a 29-bit address space, divided into eight areas. Various memory modules can be connected to the seven areas of external address, and chip select signals $\overline{\text{CS}}$[0-6] are used to multiplex all the data buses. $\overline{\text{CS}}$[2-5] are available on the expansion port. $\overline{\text{CS2}}$ was chosen and its corresponding external area is Area 2. All Area 2 address (0x8000000 - 0xC000000) accesses will activate $\overline{\text{CS2}}$.

Given a clock frequency, f_c, the theoretical upper bound on the transfer data rate, r, is determined by c_m, the number of cycles per memory access and c_i, the number of cycles for inter-area accesses. This is given by

$$r = \frac{f_c}{c_m + c_i} \times w \,\text{bps} \tag{3.3.1}$$

where w is the word size (32 bits in this case).

The required data rate across the interface is determined by the baud rate specification of 19 200 bps. The required bidirectional data rate across the expansion port is therefore 38 400 bps because the physical interface is half duplex.

Assuming external memory is accessed as synchronous SRAM, the SRAM timing diagram, provided in [39] is consulted. It is evident that a no-wait memory access lasts for two bus clock cycles. The minimum inter-area access wait period is one clock cycle. Substituting these values into equation 3.3.1 yields

$$r = \frac{96 \times 10^6}{3} \times 32 = 1024 \times 10^6 \,\text{bps} \tag{3.3.2}$$

However, since the expansion port is memory mapped directly to the SH4's data and memory buses, accessing it will be much slower than this upper limit because memory accesses are shared with other processes running on the SH4. It is not possible to say exactly what the effective throughput will be – this will have to be measured.

It is, however, possible to say that the upper limit is five orders of magnitude faster than the required data rate so it seems feasible that the expansion port will not become congested.

The total required pin count for implementing an SRAM interface to the OBC on the FPGA is 40: 32 pins for data, three pins for the clock, read and write strobes respectively, $\overline{\text{RESET}}$, $\overline{\text{IRL3}}$, a single chip select and two address pins. Whilst only one pin is required for addressing – for reading and writing

to the modem – an extra pin is included in the design to allow for address expansion.

Resource Manager

Data transfers over the expansion port must be managed by a piece of dedicated software. Software which is dedicated to managing hardware is called a Resource Manager in QNX terminology. The resource manager must provide an interface to client software enabling access to a physical resource, in this case the expansion port. A multi threaded, interrupt driven, full duplex resource manager was developed as part of this thesis and is discussed in Section 3.4.

3.4 OBC Resource Manager

The QNX kernel is a micro-kernel which means that the resource manager is a user level program. The client connecting to the resource manager is the TM protocol written by Wiid *et al.* [46]. The sequence in which the client software interacts with the resource manager, the operating system and the physical device is designed to hide implementation details of the resource manager from the client. The sequence is presented in Figure 3.2.

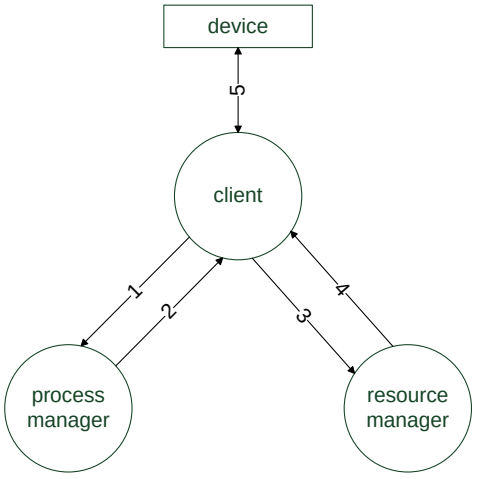

Figure 3.2: Client software device access sequence.

When the resource manager is launched, it registers a device in the namespace such as `/dev/exp`. When a client wants to access the device, the following steps take place:

- the client initially queries the process manager (a module responsible for pathname management) using the *open()* call and asks it to look up the name `/dev/exp`,
- in step 2 the process manager responds with some values which uniquely identify an open communication channel to the resource manager,
- the client then connects to the resource manager on this channel,
- once the resource manager receives the connection it generally responds with a success (and *open()* returns with a file descriptor) or an error,
- the client then uses this file descriptor to communicate with the device through *read()* and *write()* calls.

The Unix standard of exposing a device as a file is used in QNX. The standard *open()*, *read()*, *write()* and *devctl()* file operations are implemented. The TM protocol implementation consists of two threads, namely a reader thread and a writer thread, which attempt to transfer TM frames from and to the ground station respectively.

Note that the design must cater for two separate transmit paths at this stage. The first transmit path is for the downlink data radio (an off-the-shelf component used to rapidly implement a working prototype) while the second transmit path is for a dedicated DAC that will eventually form part of the downlink transmitter chain.

The resource manager therefore exposes a serial device, `/dev/ser3`, to allow data to be transmitted to the downlink data radio. It was decided to implement both devices, `/dev/exp` and `/dev/ser3` with a single resource manager. This is because the data radio connects to a UART on the ML501 FPGA board and all traffic to it must also cross the expansion port. Stated differently, there is only a single physical device – the expansion port – and it is managed by a single resource manager which provides two logical devices.

The required behaviour of the resource manager is presented in Figure 3.3 as a use case diagram.

3.4.1 Software Polling vs Hardware Interrupts

Because the CPU is shared with other processes, such as the ground station scheduler, CPU efficiency is an important design criterion for the resource manager. The possible responses of the resource manager, specifically to a *read()* system call, are analysed to determine which response method uses the least CPU cycles. The result is extrapolated to the *write()* system call response.

When a client makes a read call to the resource manager to read from the modem, there is either data available or not. If there is nothing available, the *read()* function can exit with a return value indicating that zero bytes were

read. In this case, the read thread would need to poll the resource manager until data became available – an unwanted scenario implying wasted CPU cycles. Alternatively, the resource manager blocks the reading thread, and the resource manager returns with data when it is available.

If the reading thread is blocked, the resource manager can poll the FPGA to determine whether data is available. This is another scenario where CPU cycles would be wasted. A third scenario exists wherein the resource manager is designed to be driven by hardware interrupts. When the read blocks, the resource manager can reply at a later stage after an interrupt has indicated that data is available. This interrupt driven design is clearly the most efficient choice and the design thereof is now analysed.

3.4.2 Interrupt Driven Resource Manager Design

The SH4 has 4 external interrupt pins, IRL[3-0], whereby interrupts can be generated. The pins can be configured as a single interrupt source, with the levels determining the interrupt priority. A level of 0000 indicates the highest level interrupt request. Alternatively, each interrupt pin can be configured as an independent interrupt source, with the priority of each interrupt source also configurable. The expansion port, however, provides only a single interrupt pin, IRL3, so it was decided to configure it as an independent interrupt source.

When generating an interrupt on the IRL3 pin, the FPGA must assert the interrupt until the interrupt is acknowledged. This is because the interrupts on the SH4 are level triggered. Once the interrupt has been acknowledged, the process generating the interrupt on the FPGA must allow the SH4 some time to process the interrupt before generating another one. A countdown timer is used for this. The process is depicted in Figure 3.4. A required component in an interrupt driven resource manager is an Interrupt Service Routine (ISR). The ISR runs at the highest possible scheduling priority. The ISR must therefore be lightweight and simple, performing only critical tasks, with the "real" work being scheduled to worker threads. The ISR is responsible for:

- masking (disabling) further interrupts,
- determining the source of the interrupt,
- notifying the FPGA that the interrupt was received,
- updating a data structure shared between the ISR and some of the threads in the resource manager,
- signaling the resource manager that some kind of event has occurred.

Determining the source of the interrupt must be done in software because there is only a single interrupt pin available yet there are multiple interrupt

conditions. The simplest way to do this is by reading a status register on the FPGA. The process is described in detail in Section 5.3.5.

The action of reading the status register can be used as an interrupt acknowledge. If the value of the status register is read while the FPGA is generating an interrupt, that signifies that the ISR has received the interrupt. To prevent accidental acknowledgement of the interrupt, it is ensured that the ISR is the only part of the resource manager which ever accesses the status register.

Because the ISR runs with the highest priority available, it should not perform tasks such as block data transfer. A dedicated thread is therefore included in the resource manager design for this purpose. When an interrupt is generated, the ISR acknowledges the interrupt by reading the value of the status register. The value of the status register is then passed to the worker thread. The worker thread can then use this value to determine the source of the interrupt. When the worker thread reads this value, care must be taken to ensure that the value is not updated by another interrupt. The thread therefore disables interrupts when checking this value, and enables them once it is finished handling the interrupt. Data accesses to this shared value should therefore be kept to a minimum.

In the idle state, the worker thread waits for a signal from the ISR that there is "work" to be done. QNX provides two methods for doing this. In the first method the ISRs sole purpose is to signal the thread. In the second method the ISR performs some critical task and then signals the thread. The second method is used in this design.

The worker thread is what reads and writes data from the FPGA buffers. It checks the value of the status register (passed to it by the ISR) and decides which buffers to service based on this value. The data it has read from the buffers must then be made available to the software client, in this case the TM protocol threads. If the worker thread tries to pass the data directly to the client, and if the client is not immediately ready to receive the data, this data will be overwritten when the worker thread reads new data. The data is therefore stored in circular buffers until the client is ready to read it. The data flow between the various buffers is depicted in Figure 3.5.

3.5 Downlink Transmitter

An off the shelf data radio was used as a downlink transmitter to simplify the implementation of the prototype. A data radio is a device which accepts data bits at the input and outputs a modulated RF signal which is fit for propagation over the channel. A data radio with a data rate of 19 200 bps, transmit power of 1 W, receiver sensitivity of -110 dBm operating in the 900 MHz ISM band was sought. These link budget requirements were drawn up by Iwan Kruger [30]. The ISM band was chosen because off-the-shelf components which

operate in it are readily available. The XTend OEM RF module was identified as a suitable data radio. It is a 1 W, 900 MHz RF module that accepts serial data at its input via a UART at a data rate of up to 115 200 bps. The receiving module then delivers the same data at its output, assuming the two modules are within range.

3.6 The Software Defined Radio

The modem deals purely with the manipulation of physical data bits. It does not have any Error Detection and Correction (EDAC) functionality, does not implement any protocols or run any application software. It is therefore situated, in its entirety, in the physical layer of the OSI model.

The function of the software modem is to act as a translator between digital information and radio frequency communication. The detailed design and implementation of the modem was done by Ewald van der Westhuizen [44]. The work done in this thesis involved integrating the modem to the rest of the payload.

A brief overview of the modulation techniques used are presented to provide an understanding of how the modem connects with the greater payload. Thereafter a detailed analysis of the physical interfaces is done.

3.6.1 Modem Overview

The stream of bits at the modem's input is a digital pulse train. Any aspect of this signal can be modulated – amplitude, frequency or phase. The three basic digital modulation schemes which use these aspects are Amplitude Shift Keying (ASK), Frequency Shift Keying (FSK) and Phase Shift Keying. One of these had to be chosen as a modulation scheme. It is evident from [37] that Phase Shift Keying achieves better power efficiency and bandwidth efficiency than the ASK or FSK schemes.

A more complex form of Phase Shift Keying namely Quadrature Phase Shift Keying (QPSK) provides greater bandwidth efficiency than ordinary PSK. Only a brief description of QPSK will be given because this thesis did not entail the implementation of QPSK. In QPSK modulation, a cosine carrier's phase is varied in time. Given two carriers, typically called the I and Q signals, each carrier's phase will change once within a time slot. A change in phase represents a symbol transition. A symbol therefore consists of two bits.

Differential QPSK is a method whereby only relative symbol transitions are used to demodulate the data. This is useful when incoherent demodulation is required. D-QPSK was used as the final modulation scheme for the modem.

A sampling frequency of 230 400 Hz is required for successful demodulation of a received signal in this application [44]. Also specified in the SDR modem design is a sample size of 16 bits. To calculate the data flow rate for the modem

at the given sampling frequency, the data rates of the following data paths are summed:

- unmodulated incoming data from the OBC,
- modulated data to be sent to the DACs,
- signal samples from the ADCs that need to be demodulated and
- demodulated data to the OBC.

The rate of the user data from the OBC is the effective data rate specified in the terms of reference, 19 200 bps. The data destined for the OBC flows at the same rate. The total data rate of I and Q samples being sent to the DACs can be calculated as $230\,400 \times 16 \times 2 = 7\,372\,800$ bps. The data rate for incoming samples from the ADCs is the same. The effective bidirectional data rate is therefore $38\,400 + 7\,372\,800 \times 2 = 14\,784\,000$ bps.

For this thesis, however, only the uplink signal was generated using the SDR modem. The downlink was implemented using an off-the-shelf radio. The effective data rate that was implemented is thus 7 392 000 bps.

3.6.2 General Purpose I/O Pins

The DSP has 34 I/O pins which can be programmed to function as general purpose I/O. These pins can be programmed to behave in an arbitrary manner. For example, code could be written to implement a Universal Asynchronous Receiver/Transmitter (UART) using these pins. However, dedicated hardware usually provides faster data rates and requires less code to be written. Based upon this, it was decided by the author that one of the dedicated and more flexible hardware interfaces would be considered.

3.6.3 Host Interface

The Host Interface is a byte wide interface supporting 8-, 16- or 24-bit word data transfers. It has rich features such as:

- host commands whereby the host (in this case the FPGA) can issue commands to the DSP,
- a choice of either polling, interrupt-driven or Direct Memory Access (DMA) transfer modes and
- a maximum data rate of roughly 81 MB/s.

The Host Interface lacks built-in addressing hardware, which means an addressing scheme must be implemented. A relatively simple scheme would be to add an 8 bit control byte to each 16 bit sample and use the interface in

24 bit mode. Contained in this control byte would be an address which the FPGA can use to determine whether the data is destined for the DAC, the ADC or the OBC.

The Host Interface meets all the requirements and provides useful additional functionality. It was therefore chosen above the other interfaces and the FPGA firmware was designed around the Host Interface. However, during implementation, unexpected behaviour was encountered. Specifically, a double assert of the data strobe by the DSP would occur during the middle of a three byte transfer. This caused entire words to be discarded when being transferred over the interface. The data strobe is normally asserted as an active low signal once to indicate a strobe. The double strobe phenomenon is defined as two such strobes occuring directly after one other, causing the host device to recognise it as two seperate strobes (while only a single strobe was expected).

After extensive investigation this double assertion waveform was measured with a logic analyser and a copy of the measurement was sent to Freescale to get assistance. Direct contact was made with a Freescale engineer who was unable to explain the behaviour. The double data strobe assertion phenomenon was also independently confirmed by Ewald van der Westhuizen on a separate Freescale development board. The Host Interface was thus deemed unusable.

3.6.4 Enhanced Synchronous Serial Interface (ESSI)

The DSP has two full-duplex ESSI ports allowing serial communication between devices at a maximum rate of 44 MB/s [20]. The ESSI does not feature any dedicated addressing hardware so, as with the Host Interface, additional complexity must be added to implement addressing. The two ESSI ports were designed for 6 channel digital surround sound applications as each ESSI can function with a single receiver and drive three separate transmitters.

3.6.5 External Memory Expansion Port

The DSP expansion port can be used for either memory expansion or as memory mapped I/O. It consists of a 24-bit address bus, a 24-bit data bus, and control lines that provide features such as chip select and bus arbitration. The expansion port supports the implementation of external Static Random Access Memory (SRAM) or Dynamic Random Access Memory (DRAM). The implementation of DRAM on an FPGA is more complex than that of SRAM. Furthermore, the DRAM timing specifications were not present in the DSP's data sheet. The SRAM timing waveforms were available so it was decided to investigate the implementation of an SRAM interface over the expansion port.

Read and write operations over the SRAM interface are synchronous with the DSP's internal core clock. The DSP's internal clock runs at 275 MHz. A single SRAM access takes one clock cycle. Up to 31 wait states can be inserted by programming the Bus Control Register. An infinite number of wait states

can be added by the $\overline{\text{TA}}$ control signal. The expansion port is therefore capable of providing the required data rate.
The total required pin count for implementing an SRAM interface to the DSP on the FPGA is 30: 24 pins for the data bus, a pin for the clock signal, a read strobe and a write strobe. Whilst only 2 pins are required for addressing (this is because only 3 addresses on the FPGA are used, namely DAC write, ADC read and OBC read / write) three pins are used in the design to allow for address expansion.

3.6.6 SRAM Code

The DSP is the bus master of the SRAM interface. The modem has two main function calls namely *modulate()* and *demodulate()*. Each of these function calls requires two data buffers. There are therefore four buffers containing data on the DSP. The simplest implementation of data transfer is round robin software polling, although this wastes clock cycles unnecessarily. Interrupt and DMA driven transfers are also possible, although it was not possible to get DMA transfers working during the implementation phase. The DMA controller was configured as per the data sheet, the source address, destination address and block size values loaded and the DMA controller was told to initiate the transfer yet the data was never transferred to the destination address. The cause of this problem remains unknown. For this reason it was decided to use interrupt driven transfers over the SRAM interface where possible.

Four dedicated hardware interrupt lines are available on the DSP. The FPGA can generate an interrupt when it has data in its buffers which the DSP must read. The corresponding ISR reads a block of data from the FPGA and places it in a buffer. When the DSP has a block of data that it wants to transmit to the FPGA, however, it must poll a status register on the FPGA to determine whether there is space in its buffers to accept the data. If there is space in the buffer it performs a write, otherwise it retries at a later stage. This polling will only occur once a block of data is available in the appropriate buffers, which means not many clock cycles are wasted on polling in this direction. The data flow of this design is depicted in Figure 3.6.

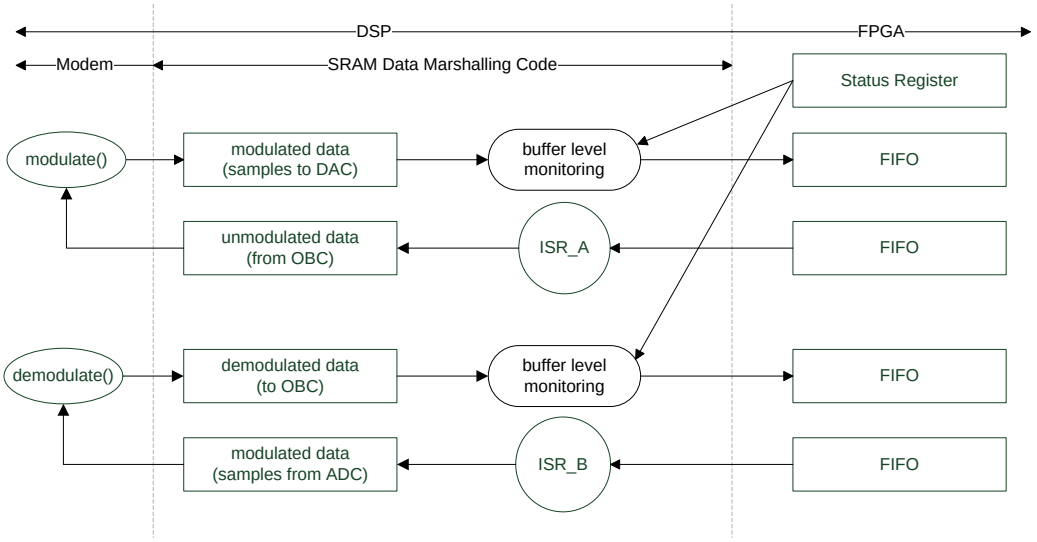

Figure 3.6: DSP-FPGA dataflow diagram.

3.7 The Field Programmable Gate Array

The choice of an FPGA was driven by two main requirements. Because the FPGA's primary function was marshaling the flow of the various data between components, enough I/O pins were a primary requirement. Furthermore, functional blocks such as First In First Out (FIFO) buffers were required to allow reliable data transfer between clock domains. Thirdly, PCB layout was to be avoided where possible during the prototyping phase due to the complexity of the system.

Two FPGA boards were immediately identified as being available. The first was a prototype version of the Signal Interface Card (SIC) from the experimental payload research on Sumbandila Satellite. The second was a Xilinx Virtex-5 ML501 development board [49]. The SIC has a PC104+ connector which slots into the SH4 OBC making it ideal for this project and was chosen above the Xilinx board for this reason. However, during implementation, it was discovered that the FPGA on the SIC, an Actel ProAsicPlus, did not have functional FIFO blocks. This was thoroughly investigated. Actel's documentation indicated problems with the FIFO blocks and stated that they had been obsoleted and should be "used with caution" [5]. The fact that Actel's Asynchronous FIFO cores would discard words occasionally was also confirmed in a private communication with Heiko Berner from SunSpace. This meant that the Xilinx ML501 board was used in the implementation of the payload prototype.

3.8 FPGA Design in Terms of the OSI Model

The FPGA deals primarily with the physical transport of data between the major subsystems in the payload. Also implemented on the FPGA are EDAC modules implemented by Wiid. The FPGA therefore spans the Physical and Data Link layers.

3.8.1 Physical Layer

It is noted that the ML501 FPGA board has two selectable I/O standards: LVCMOS-2.5 and LVCMOS-3.3. The LVCMOS-3.3 standard is what both the OBC and the DSP use. This means that FPGA I/O pins can be connected directly to both the OBC expansion port and the DSP expansion port.
The ML501 has the following available interfaces:

- an ethernet interface,
- an expansion header,
- an RS-232 serial port,
- a Universal Serial Bus (USB) host port and
- a USB peripheral port.

As described in Section 2.3.1, the ethernet interface is used for receiving baseband I/Q samples from the KUL SAA. The serial part is also unavailable, as it is used to connect to the downlink data radio. That leaves the USB port and the expansion header. The clear choice in this case is the expansion header because it contains a total of 78 single ended general purpose I/O pins. As calculated earlier in the chapter, we require 40 pins to implement the SRAM interface to the OBC and 30 pins for the DSP SRAM interface. This leaves 8 pins for debugging purposes.

3.8.2 Data Link Layer

Unmodulated data in the form of a TM frame from the OBC is sent through various stages before it gets passed to the modem for modulation. Firstly a Cyclic Redundancy Check gets calculated and added to the TM frame. The data is then passed to a pseudo randomiser to prevent a DC component from forming in the modulated signal. The randomised data is then passed to an Attach Synchronisation Marker (ASM) module. These three modules were implemented by Wiid. The SRAM interface to the OBC is connected to the CRC module, and the ASM module is connected to the DSP SRAM interface. The payload firmware is depicted in Figure 3.7. Objects bound in dashed line boxes represent off board components and components that did not form a part of the work done for this thesis.

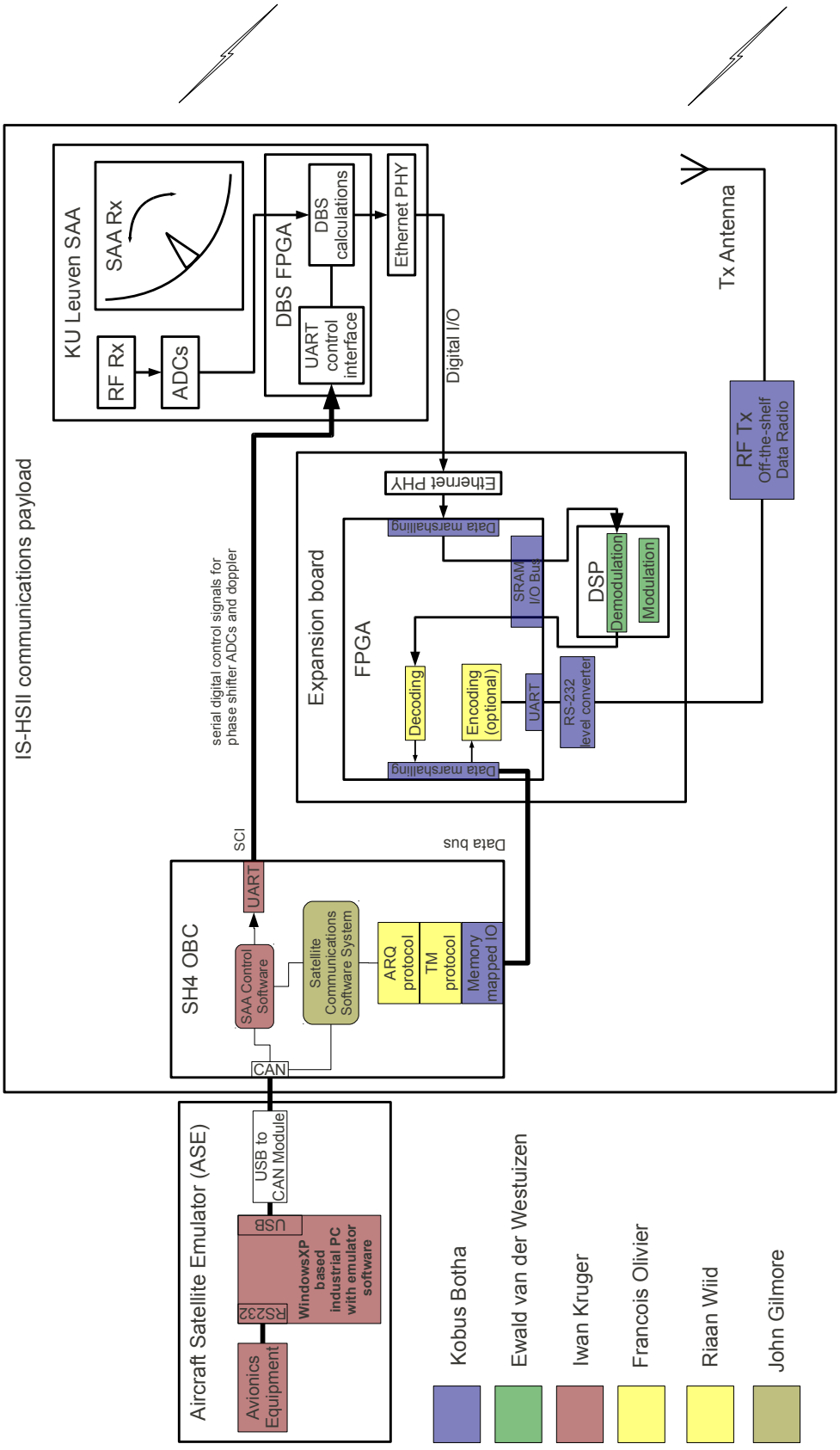

Figure 3.1: The greater payload.

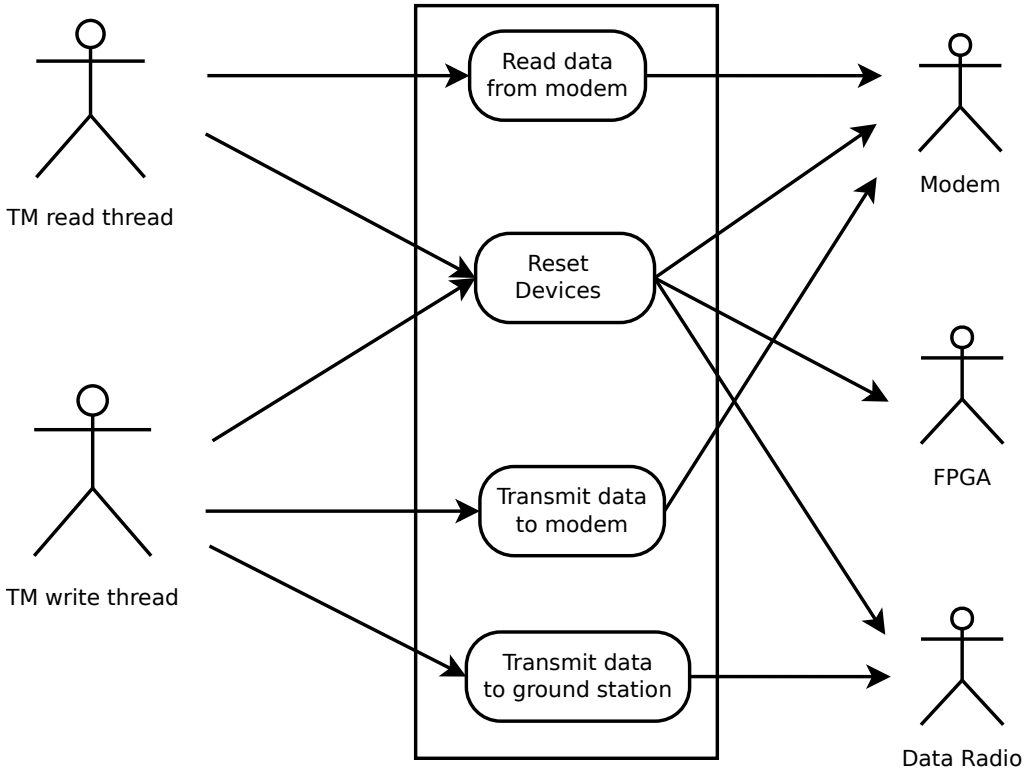

Figure 3.3: Use case diagram for the resource manager.

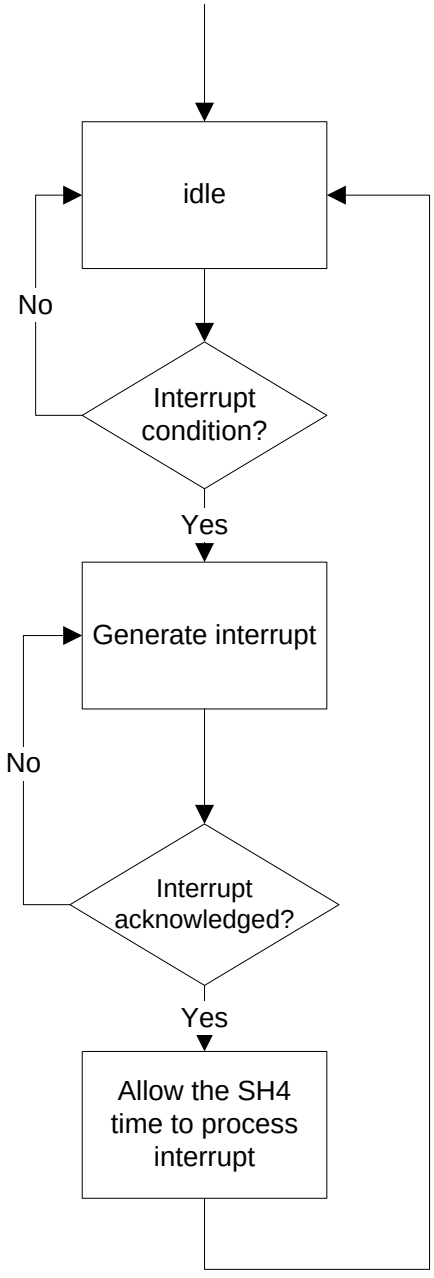

Figure 3.4: Interrupt generation sequence.

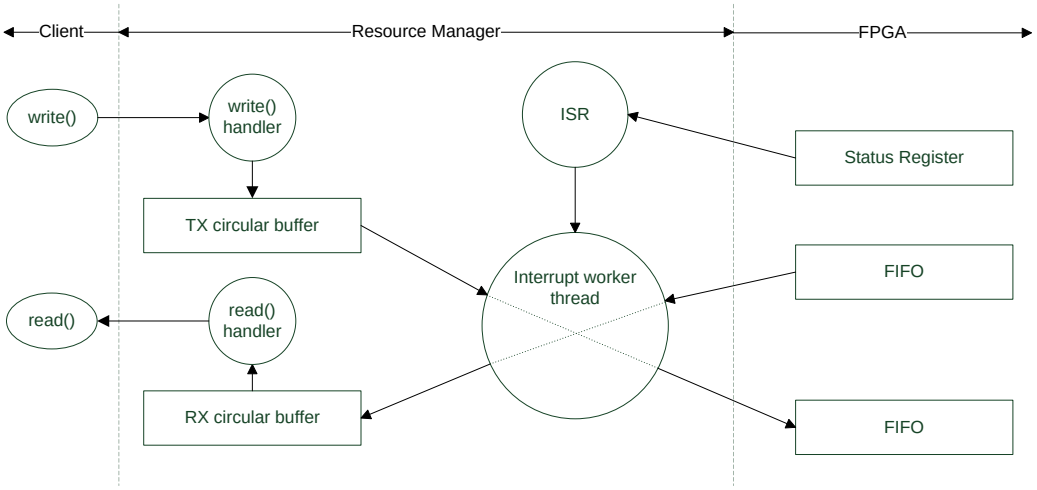

Figure 3.5: ISR integration diagram.

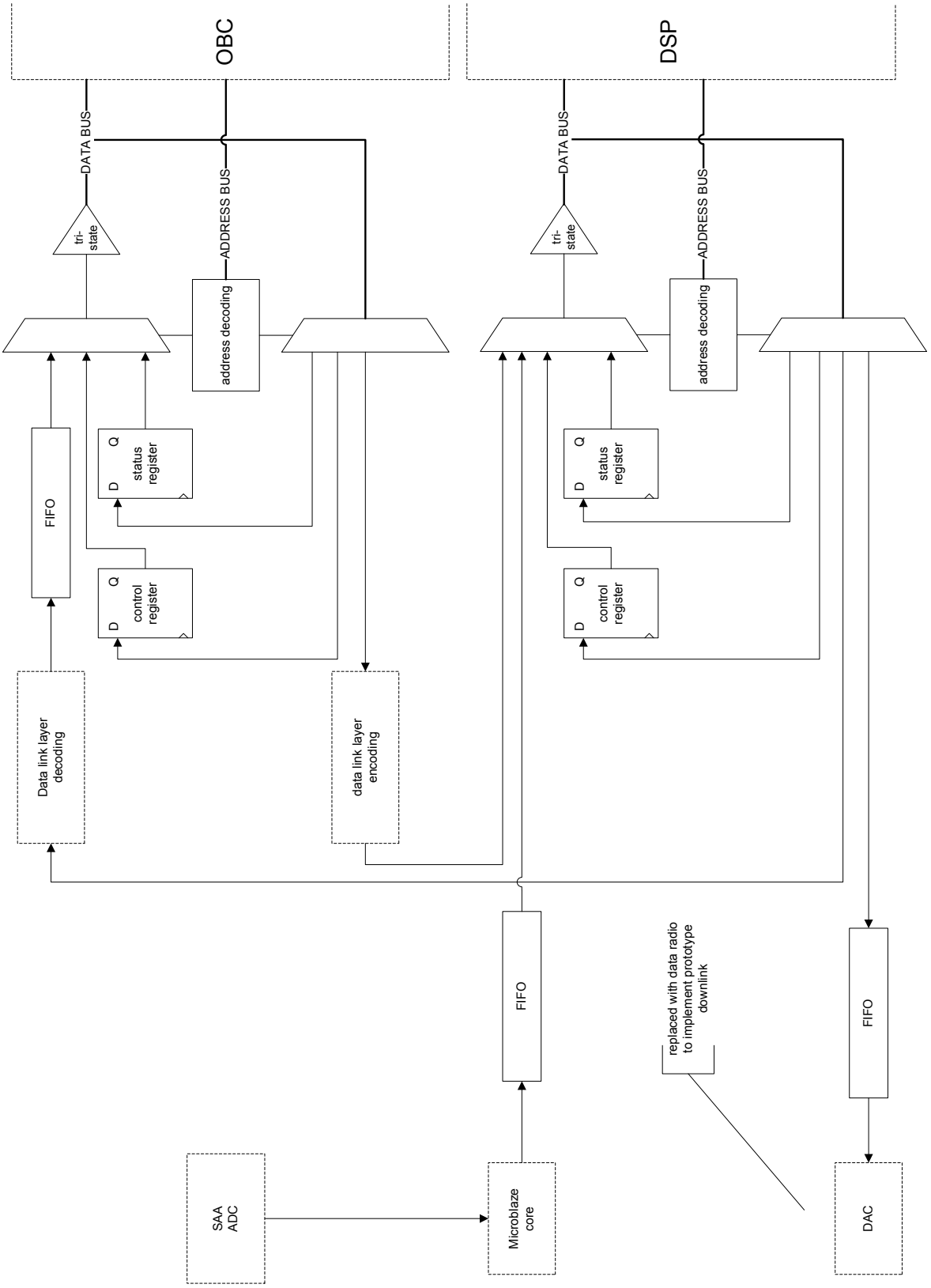

Figure 3.7: Payload FPGA firmware.

Chapter 4

Ground Station Design

In this chapter the design of the ground station will be presented. The purpose of the ground station is to transmit an RF signal to the payload in such a way that the testing of the Steerable Antenna Array on the payload is possible, while demonstrating the ability to transmit information to a remote ground station via the communications link. These given functions are used as a basis from which a list of required hardware components is constructed. Each component is then discussed in its own section.

There are two important figures in this chapter namely Figures 4.2 and 4.1. They provide an overview of the ground station data flow paths and a detailed view of the design, respectively. Figure 4.1 is especially important because it defines which parts of the system formed a part of the work done for this thesis and which parts were implemented by others. The implementation of the components and interfaces which formed a part of this thesis are presented in Chapter 5.

4.1 Necessary Hardware

The four ground station criteria outlined in Chapter 2 are repeated here for the reader's convenience:

- The ground station must transmit an RF signal of the correct amplitude, frequency and polarisation to the SAA uplink receiver,
- the RF signal must contain a complex I/Q message signal,
- the I/Q message signal must in turn contain modulated data, i.e. data from environmental sensors and
- the ground station must be able to receive data from the downlink transmitter.

From these criteria a list of necessary hardware components follows:

- RF tools such as antennas and amplifiers are required to generate the desired RF Signal for propagation,
- an I/Q modulator will be required to generate the complex I/Q signal from the two quadrature baseband signals,
- some type of modem is required to modulate the user data into baseband I/Q signals,
- a digital to analogue converter is required to feed the modem samples into the I/Q mixer,
- a data processor such as a PC computer will be required to generate and manage the useful information,
- RF tools for the downlink receiver,
- hardware "glue" such as an FPGA is required to manage and connect all the digital interfaces of the above components.

Visualising the end-to-end flow of data through the system provides a good idea of the kind of interfaces required between components. This data flow will be described using an example of data being sent from ground station A to ground station B. The two ground stations do not have a line of sight path and need to transmit the data via the payload. The transmit path of A will illustrate the uplink and the received path of B the downlink. The transmit and receive paths are illustrated in blue and green respectively in Figure 4.2.

To begin with, environmental data such as weather data are sensed, sampled and transmitted to ground station A via an ad hoc network. The data is collected on the ground station PC by application software. This data must then be transmitted to the modem for modulation. It passes through the data link layer where the data is broken into packets using the simple Automatic Repeat reQuest (ARQ) protocol. These packets are passed to the Telemetry (TM) protocol where they are separated into frames. The frames are sent to the physical layer for channel coding. A Cyclic Redundancy Check (CRC) is added to the TM frames which then get randomised and sent to the modem.

The modem modulates the data bits and sends the modulated data back to the FPGA as samples. The FPGA then sends these samples to the Digital to Analogue (DAC) converter to escape the digital domain. The output of the DAC is connected to an I/Q modulator which generates the real signal from the composite complex baseband signals. The output of the modulator gets mixed to the desired carrier frequency. The RF signal then gets amplified to the desired level and transmitted via an antenna.

The RF signal propagates through free space and reaches the payload. To simplify the scenario the payload will be represented by a bent pipe. The data gets retransmitted by the payload to ground station B.

Ground station B's receiving antenna directs the RF signal to an off-the-shelf data radio. The data radio demodulates the signal and sends the raw data to the PC (via the FPGA) where the application layer software processes it. Note that the data radio has its own built-in EDAC.

The uplink and downlink are required to run in full duplex mode. The above processes are depicted running simultaneously in Figure 4.2.

The various components in the ground station design will now be analysed.

4.2 Ad Hoc Sensor Network

Designing an ad hoc sensor network was not the main goal of this thesis. However, it was noted that the basic requirements for each node are a transducer, in the form of a sensor, and an Analogue to Digital Converter (ADC) to sample the analogue values. Furthermore, each node requires some kind of radio equipment to communicate with other nodes. Finally there must be a single node in range of the ground station which will handle the transfer of data from all the other nodes to the ground station PC.

An immediate solution for a node was identified as one consisting of a Renesas micro-controller (with built-in ADC) connected to an RF Digital RFDP8 [40]. The RFDP8 is a complete 2.4 GHz transceiver with a built-in ad hoc protocol available at low-cost from RF Design at the time of writing. The RFDP8 has a UART which allows it to connect to either the Renesas board or a PC, using some level conversion circuitry such as the MAX232.

It was decided that this was a sufficient design for the purposes of this thesis. The design was implemented by Christopher Gautun from ENSEIRB Matmeca.

4.3 PC

The choice of PC was simplified by the fact that a lightweight PC was sought since the ground station would ultimately be deployed in the field. That excluded machines such as desktops and servers. The Fit-PC is a very small PC which requires a regulated DC power supply [2]. It consists of an AMD Geode 500 MHz CPU, 256 MB of RAM, a 160 GB disk drive, 4x USB ports and comes preloaded with Ubuntu Linux. This would be sufficient for running the application and data link layer software and provides physical interfaces to the outside world. A Fit-PC was immediately available for development from the Sumbandila Satellite Communications Payload project. It was therefore decided to use the Fit-PC as a ground station PC.

4.4 FPGA

The following criteria must be met by the FPGA used for the ground station:

- it must have a sufficient number of Input/Output pins to interconnect all the components, and
- it must have sufficient capacity for implementing the EDAC firmware for the data link layer.

Two Xilinx Spartan-3E evaluation kits were immediately available for use. The Spartan-3E kit provides 43 dedicated I/O pins. It also has USB and serial digital data interfaces. The FPGA is a crucial component which connects the electrical interfaces of the digital components to each other. Each of these interfaces is now analysed.

4.4.1 FPGA to Fit-PC Interface

The Fit-PC has 4x USB ports as digital data interfaces. It does not provide any other interfaces such as serial or parallel ports or expansion buses. Implementing serial port data transfer is considerably simpler than USB transfer. A serial data transfer design was therefore chosen. A USB to serial bridge, using the PL-2303 microchip, can be used to provide a serial port as a data interface. Given our data rate of 19200 baud, the serial port would be sufficient as baud rates of 115 200 are feasible.

Both the Fit-PC and the FPGA are Data Communications Equipment (DCE) devices (all this means is that both the FPGA and the Fit-PC receive and transmit data) so a null-modem cable must be used to connect the two. A null-modem cable has its receive (Rx) and transmit (Tx) lines swapped at one end. This enables the Fit-PC to receive and transmit data to the FPGA and vice versa.

The Spartan-3E kit provides two serial ports in the form of DB-9 connectors, together with level conversion circuitry to convert from UART TTL levels to serial port RS-232 levels. A UART would be implemented in VHDL on the FPGA. The UART requires 2 pins on the FPGA (transmit and receive).

Regarding the flow control of data over the serial interface, the DB-9 connectors on the Spartan-3E board have no connections for the hardware flow control pins. Flow control therefore has to be implemented in software. On the Fit-PC the serial port device driver handles the flow control. The driver is well documented and will not be explained here [32]. The serial port is opened in asynchronous, full duplex mode with XON/XOFF software flow control. On Linux systems this is accomplished by issuing the following command:

```
stty --file=/dev/ttyUSB0 ixon
```

The /dev/ttyUSB0 is the UART to USB bridge supplied by the PL2303 chip and recognised as such by the Linux kernel. The ixon option tells the serial port to enable XON/XOFF flow control. This means the driver will listen for the XOFF character (ASCII 17) and stop transmitting when this character is received. Similarly, it will resume transmission when the XON character (ASCII 19) is received. Because the XON/XOFF characters may appear in normal data transfer, an escape code is used to transmit these characters. The serial port device driver is POSIX compliant so open(), read(), write() and ioctl() calls will be used to manipulate the serial port and ultimately the FPGA as well as the modem.

4.4.2 FPGA to DSP Interface

As discussed in Chapter 3 there is a plethora of interfaces to the Freescale DSP. The interface to the modem will be the same SRAM interface used on the payload and described in Chapter 3. The SRAM interface requires 30 I/O pins, leaving 13 available.

4.4.3 FPGA to DAC Interface

The Spartan-3E has an on-board four channel DAC, the National LTC2624. This simplifies interfacing to the DAC as no glue logic or adapter circuitry is needed. The LTC2624 has a well documented serial interface used to load and update its registers [33]. The serial interface is connected directly to the FPGA.

4.4.4 Firmware

Once all the components have been connected to the FPGA, the firmware is what ensures that the electrical interfaces conform to the specifications. Writing firmware in VHDL is a simple four step process. Firstly the design is performed in hardware using low level components such as logic gates and flip-flops. Secondly, VHDL is written to implement this hardware design. Thirdly, a VHDL test bench is written which simulates the performance of the hardware in software. Finally, the VHDL is synthesised (analogous to compilation of code) and programmed onto the FPGA.

Before beginning the hardware design it is noted that the various digital components run at different clock speeds. The notable clock speeds in the system are the baud rate of the UART to the PC, the sampling rate of the DAC and the clock speed of the DSP chip. As done on the firmware for the payload, FIFO buffers will be used to ensure that data is transferred reliably between these clock domains.

Off-the-shelf, open source UARTs written in VHDL are freely available from numerous sources [4], [3]. The Constant (K) Compact UART written by

Ken Chapman of Xilinx is a fully functional, freely available VHDL module consisting of transmit and receive macros written for Spartan, Virtex and Virtex-E devices [9]. Each macro handles data in little endian mode with one start bit, eight data bits and one stop bit. Each macro also contains a built-in 16 byte FIFO. These macros are perfectly suited for communicating with the UART on the Fit-PC.

Unmodulated data received from the Fit-PC must be stored in a FIFO which will be read from to transmit data to the modem. Software flow control must be implemented in the firmware to avoid an overflow of this FIFO. The FIFO has a programmable threshold. This threshold can be used to trigger the sending of an XOFF command to the device driver on the Fit-PC to indicate that a full condition is approaching. Once the FIFO is safely below this threshold, an XON command can be sent to resume reception.

In addition to the XON/XOFF commands, the firmware must be able to respond to other commands from the Fit-PC. One of them is the startup "HELLO" sequence which is used by the system software on the PC to verify hardware connectivity. To aid in the processing of these commands, the PicoBlaze core from Xilinx was used [50]. The PicoBlaze core is an 8-bit RISC soft processor core which is freely available from Xilinx. The core has a basic set of assembler instructions with which it is programmed. The assembly code for the PicoBlaze processor was written by Ewald van der Westhuizen.

The ground station FPGA firmware is depicted in Figure 4.3.

4.5 Modem

A QPSK modulator is required to modulate the user data into baseband I/Q signal samples which can be converted into an analogue signal and sent to the quadrature mixer. The top level modem design was done by Dr Gert-Jan van Rooyen, Ewald van der Westhuizen and Kobus Botha during the multiple channel communications payload project [7]. It was implemented in the C language on a Freescale DSP56321 by Ewald van der Westhuizen with the assistance of Dr van Rooyen.

The modem generates symbols from the user data two bits at a time. These bit pairs, which are the symbol values, are used to generate a message signal pulse train. The message pulse train is sent through a raised-cosine filter to smooth the phase transitions. The output of the filter is the message signal. A baseband carrier wave is then generated using DDS and mixed with the message signal to create the modulated baseband signal samples which can be sent to the DAC.

Table 4.1: Quadrature mixer comparison.

Feature	**HMC495LP3**	**COM-4001**
Frequency Range (MHz)	$250 - 3\,800$	$902 - 928$ / $2\,025 - 2\,500$
Output Power (dBm)	-9	-9
Integrated LO	No	Yes
Connectorized	Yes	Yes

4.6 DAC

The modulated baseband signal has a bandwidth of 19 200 Hz. The Nyquist-Shannon sampling theorem states that, for a signal with a bandwidth of F Hz, samples spaced less than $\frac{1}{2F}$ seconds apart will be a lossless representation of the signal. Typically in practice this is extended to $\frac{1}{3F}$ and $\frac{1}{4F}$ (a technique known as oversampling, typically used to create safety margins to cater for filter roll off). The DAC sample rate must therefore have a minimum value of $4 \times 19\,200 = 76\,800$ samples per second. Furthermore, the samples from the modem represent 16 bits each, so the ideal DAC will have a sample word size of 16 bits.

4.7 Quadrature Upmixer

As explained in Chapter 3, the SAA expects a QPSK signal centered at 2.45 GHz. The modulated baseband signals from the modem must therefore be mixed up to 2.45 GHz. QPSK signals can be combined to form a single real signal using a technique known as quadrature upmixing [8]. The HMC495LP3 from Hittite Microwave Corporation was identified as being a suitable device capable of performing direct quadrature upmixing to frequencies between 250 and 3 800 MHz [24]. Another quadrature mixer that was identified is the COM-4001 from ComBlock [11]. Both were feasible for use with the ground station yet, due to time constraints a connectorised evaluation module was sought, preferably with a built-in local oscillator (LO). This was to simplify the implementation of the transmitter chain and to avoid a potentially lengthy circuit layout process. The comparison between these two modules is summarised in Table 4.1.

It was decided to evaluate the COM-4001 because it does not require an external LO.

4.8 RF Hardware

The RF output signal from the quadrature upmixer requires amplification to meet the equivalent isotropically radiated power of 1 W (30 dBm) required by the link budget. Assuming an antenna gain of 6 dB, this means a total

gain of $30 - 6 - (-9) = 33$ dBm is needed. Sourcing an amplifier with a gain of 33 dBm was not possible. What is typically done to achieve this kind of gain is that the low-power signal is amplified using successive stages. The first amplifier, referred to as a Low Noise Amplifier (LNA), lifts the signal enough to be sufficiently amplified by a second amplifier, a Power Amplifier (PA). The noise injected into the signal by the first amplifier in the stage gets multiplied by the second one, hence the low noise figure of the first amplifier.

Off-the-shelf components were sought to minimise costs. Two products, both from Hittite Microwave, were identified as being suitable. An LNA, the Hittite HMC286, provides 19 dB of gain for signals in the $2.3 - 2.5$ GHz range and is available as a connectorised evaluation module [23]. The Hittite HMC755LP4E is a 1 W PA that provides a gain of 31 dB for signals in the $2.3 - 2.8$ GHz range and is also available as a connectorised evaluation module [25].

The HMC286 has a maximum allowable input power of 0 dBm, so it is compatible with the COM-4001. However, the HMC286's output 1 dB compression point is at 6 dBm. With a gain of 19 dB this means that input signals with a power greater than $6 - 19 = -13$ dBm will cause the LNA to operate in the compression range and the input signal will become distorted. This is not a problem, however, because the COM-4001 has a soft programmable output power.

If the LNA is driven with a -14 dBm signal from the COM-4001, the output will be 5 dBm. Adding the 31 dB from the PA gives 36 dBm. This is also not a problem as the gain of the PA can be attenuated using control voltages to provide the required output power.

4.9 Transmitter (Uplink) Antenna

Antenna design did not form a part of the work done for this thesis, and the designed and construction of the antenna was done by Wynand van Eeden. The antenna specifications are:

- the radiated wave must be circularly polarised as required by KUL,
- the antenna should display high return loss when fed a 2.45 GHz signal as this means that the signal is radiated effectively at this frequency,
- the radiation pattern should be omnidirectional to eliminate the need for antenna tracking as this simplifies the ground station implementation.

4.10 Receiver (Downlink) Antenna

A 900 MHz omnidirectional antenna was chosen for the RF downlink. The Aerocomm 0600-00019 is a 915 MHz omnidirectional half wave dipole antenna

with a 2 dBi gain [1]. Two antennas were obtained for implementing the downlink transmitter and receiver.

4.11 Downlink Data Radio

The required data rate for the downlink is 19 200 bps. The XTend™ OEM RF module provides a data rate of up to 115 200 bps and was used for this thesis [14].

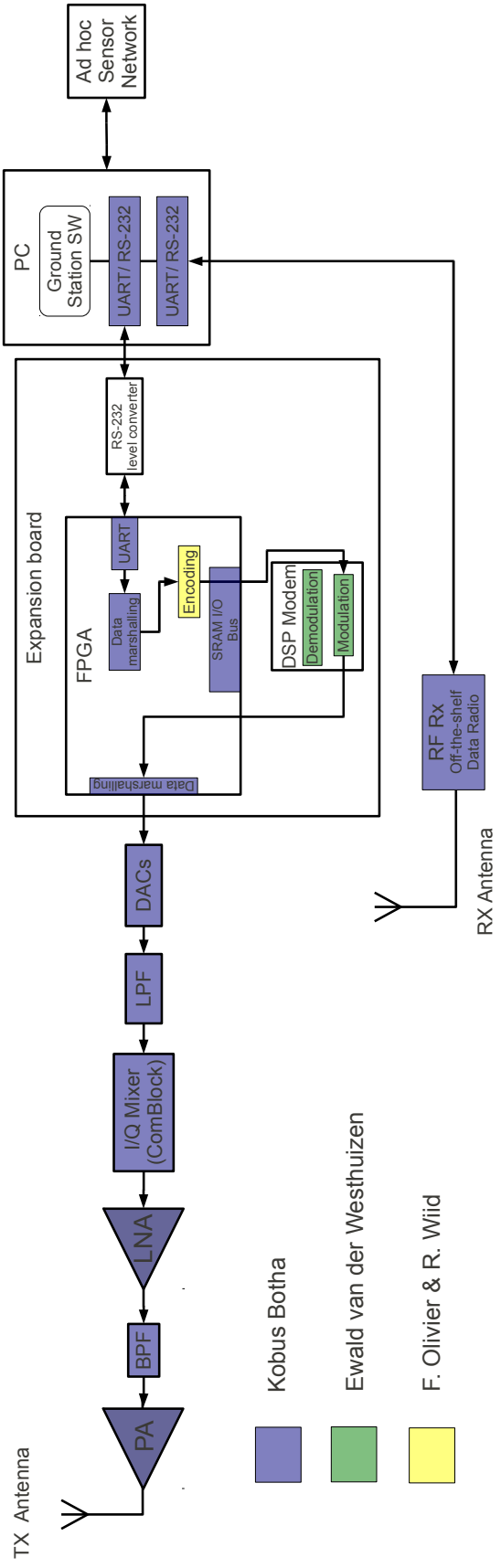

Figure 4.1: Functional overview of the ground station.

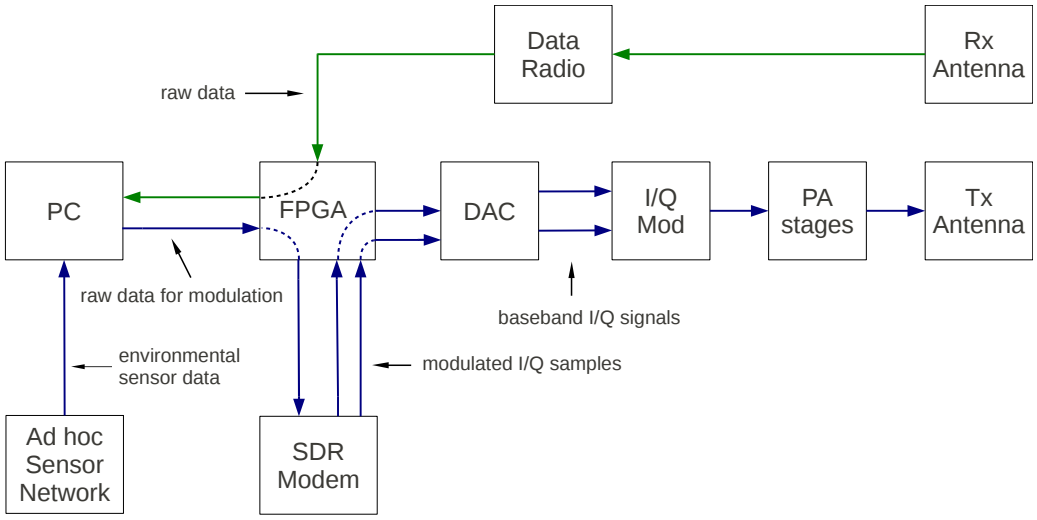

Figure 4.2: Ground station overview.

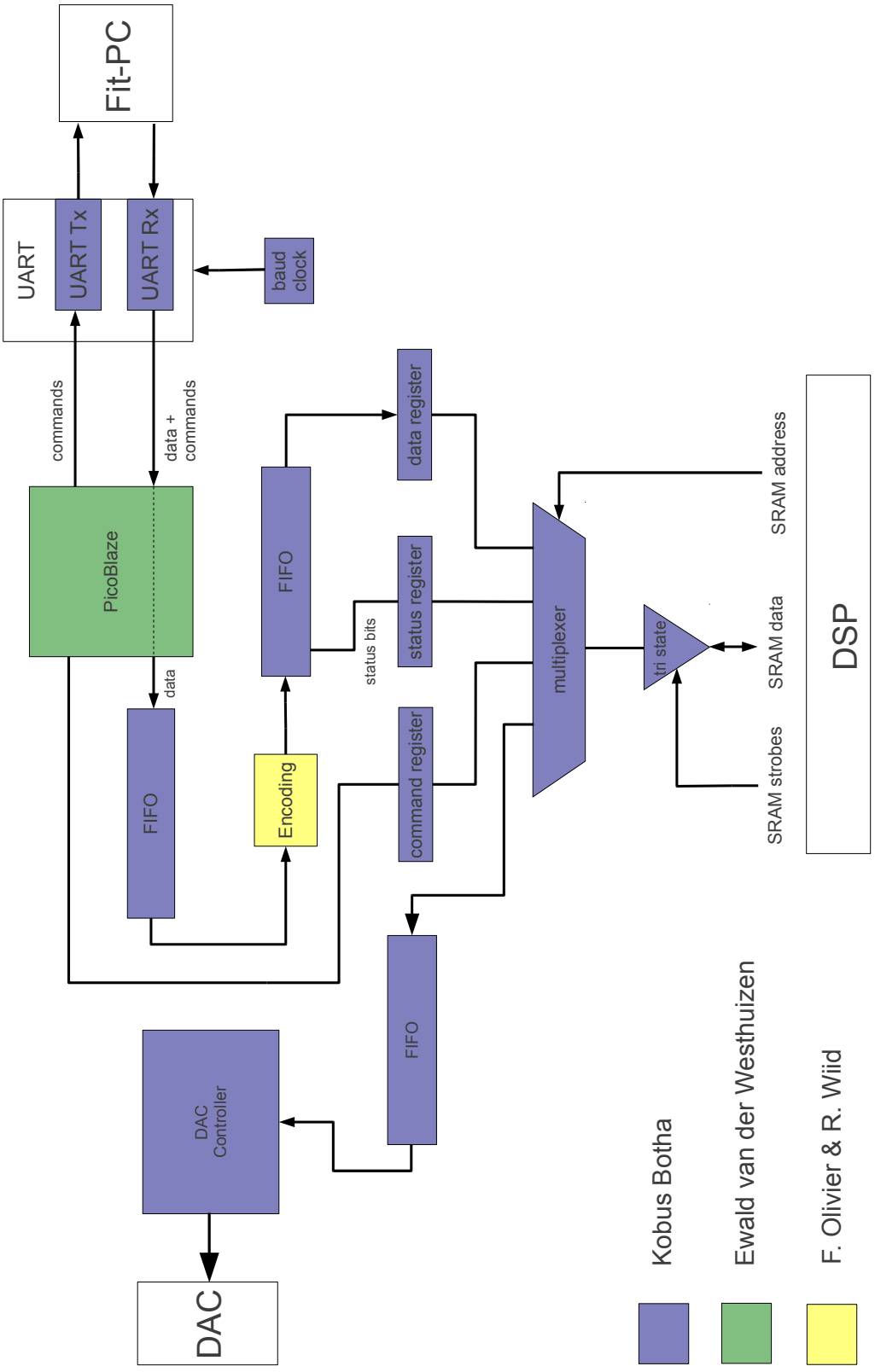

Figure 4.3: Ground station FPGA firmware.

Chapter 5

Payload Implementation

This chapter explains how the design presented in Chapter 3 was implemented. The bottom-up construction of a system involves implementing the lowest level components first. Smoke tests are performed on all components before integration to any other components. Once smoke testing is passed, components are then connected to each other to form subsystems of increased complexity.

These subsystems are then connected to each other and tested. This process is repeated until all the components are connected to form a single system. The tests done for each component are described at the end of that component's implementation description in this chapter. These smoke tests are only intended to verify the operation of single components.

The integration of all the components into a single payload system is presented at the end of the chapter, along with a few integration tests. Final integration tests are documented in detail in Chapter 7.

5.1 Development Environment

The SH4, FPGA and DSP all have their own set of development tools and set up requirements. These are briefly presented in this section.

5.1.1 Accessing the OBC

Accessing the QNX image on the SH4 is possible through the CAN bus [15]. Dedicated Cygnal CAN controllers in combination with SunSpace's CAN driver on the SH4 make this possible. However, typical PCs do not have CAN bus terminals. An intermediate device, the Ground Station Equipment - Ethernet Interface (GSE-EI) module from SunSpace, was used to access the CAN bus. The GSE-EI converts Ethernet packets into CAN packets. The Ethernet to CAN tunnel set up by the GSE-EI is configured by the Lua CAN Interface (LCI) software provided by SunSpace.

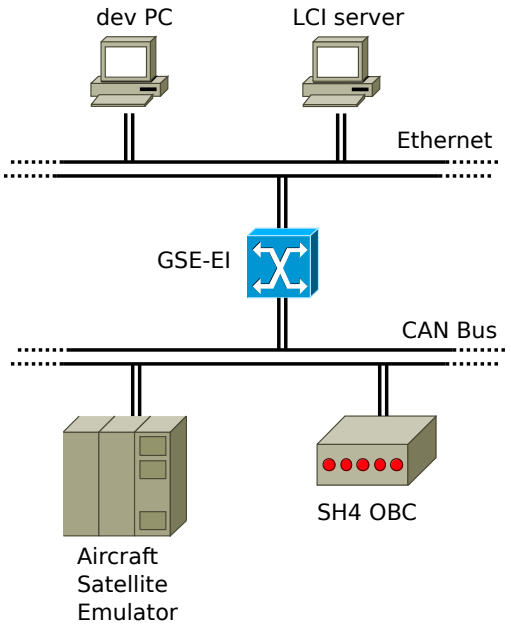

Figure 5.1: CAN bus development environment.

A dedicated PC running the LCI software on the same Ethernet segment is required to use the GSE-EI. Once LCI is running and the QNX image on the SH4 has booted, data can be transferred to and from the SH4 using standard IP tools such as ftp and rsync. The CAN bus development environment is depicted in Figure 5.1. The SH4 can also be accessed via a serial terminal which is always active. This serial terminal is crucial when debugging boot problems as it displays a boot progress log.

5.1.2 QNX Programming

QNX supplies a cross-compiler allowing SH4 executables to be compiled on various other architectures such as x86. The cross-compiler supports C and C++ code. POSIX compliant C code was written where ever possible for code portability as certain software components could be reused on the ground station. A Linux version of the QNX Integrated Development Environment (IDE) was used to write and compile code with.

Executables were transferred to the SH4 using the QNX connection manager, qconn, and can either be executed from within the IDE or from the command line during a telnet or rsh session. The programs are stored on the OBC's non-volatile memory and are configured to execute at start up, using the SunSpace ProcessMonitor component, putting the SH4 in a known state after a cold start [42].

5.1.3 ML501 FPGA Programming

Xilinx provides the Integrated Synthesis Environment (ISE) software suite for generating bitstream programming files from VHDL code. The Xilinx Virtex-5 family FPGAs use flash-erase EEPROM technology for programming firmware. The firmware must therefore be loaded after each cold start. Six configuration interfaces are available for loading a generated bitstream file onto the FPGA. Serial configuration via JTAG was used to load new firmware onto the FPGA during development cycles. Once a version of the firmware was considered stable it was loaded onto a compact flash card. The image from the compact flash is loaded onto the FPGA after a cold start by dedicated flash hardware on the ML501 board, ensuring that the system is in a known state.

When generating bitstream files in ISE, it is crucial to note that unused pins on the FPGA are connected to ground by default. This could be catastrophic if the pin is ever connected to a positive supply pin. ISE must be instructed to let unused pins float as this decreases the possibility of accidental damage.

5.1.4 DSP Programming

Motorola CLASS COF (.CLD) files are loaded onto the DSP via the DSP56300 Hardware Debugger which is part of the Motorola DSP Software Development Tools suite. CLD files can be compiled using an assembler, a C compiler or a C++ compiler. Motorola provides a free-to-use assembler which was used for initial testing of the DSP's host interface. The Tasking software suite, which provides C and C++ compilers, was subsequently purchased. The DSP software was therefore written in C, compiled and linked by the Tasking IDE and uploaded to the DSP by the hardware debugger through the JTAG port using a parallel port cable. Stable versions of the code were written to the Atmel AT29LV010A-20TC chip which provides $128\,\mathrm{K} \times 8\,\mathrm{bits}$ of CMOS flash memory from which the DSP boots, putting the DSP in a known state after a cold start.

5.2 Electrical Interface Connections

Physical connects must be made between the various components of the payload. The implementation of these connections is now described.

5.2.1 FPGA-OBC

The SH4 OBC PC104+ connector is a socket type quad-row 30 pin connector. An adapter board must therefore be constructed which slots into this connector by means of another PC104+ connector. Similarly, to connect to the ML501, an adapter board must be constructed which slots onto the two XGI headers, which are triple-row 30 pin headers with standard 0.1" spacing. Some of the

pins on this adapter board will connect to the DSP and some will connect to the OBC. Furthermore, one of the two adapter boards should contain standard 0.1" header rows which piggyback on the standard I/O pins, allowing convenient measurement of voltage levels on the I/O pins. This is to assist in debugging.

PCB layouts were made for both adapter boards. The OBC adapter board was tested using a continuity test. The ML501 adapter board was tested in two ways. An input test was done by synthesising firmware with a single line of VHDL:

```
LED <= in0 or in1 or in2 ... or inN;
```

where N was the last pin. One of the LEDs on the board would then light up whenever a 3.3 V signal was applied to an I/O pin. The ability for each pin to act as an input was verified in this way. As an output test, an M-bit counter was output on M output pins. The frequency at which each pin toggled was then measured using an oscilloscope. This test verified the ability of each output pin to drive a signal.

5.2.2 FPGA-DSP

The DSP56311 Evaluation kit provides access to the DSP's address, data and control buses in the form of three seperate 0.1" headers. Ribbon cable and 0.1" header sockets were used for connecting the ML501 adapter board pins directly to the DSP.

5.2.3 FPGA-SAA

The interface supplied by KUL to their SAA is an Ethernet interface. Ewald van der Westhuizen implemented a PicoBlaze module on the Virtex-5 FPGA which handles the reception of ADC samples from the SAA over Ethernet. All that is required to connect the ML501 board to the SAA subsystem is an Ethernet cable.

5.2.4 FPGA-Data Radio

The Xtend OEM accepts UART TTL levels at its data port. This is not compatible with the RS-232 levels of the ML501 serial port connector. A level translation board was therefore constructed for the Xtend module to allow it to connect to the ML501 board's RS-232 port.

5.3 OBC Resource Manager Implementation

The resource manager was written in the C language as it is a well documented language for writing QNX applications. QNX also provide well documented C

development libraries.

QNX provides an InterProcess Communication (IPC) mechanism called message passing which can be used to send data in a reliable manner between processes. Using message passing involves creating a message channel for incoming messages, listening for incoming messages on that channel, and then replying to any messages received on the channel. The resource manager uses message passing for all IPC tasks.

5.3.1 Expansion Port Setup

The first thing that the resource manager does is to ensure that the timing of the expansion port hardware agrees with what the firmware expects. When accessing the expansion port the SH4 is the bus master. All timing is dependent on the speed of the bus clock, CKIO, which is set up using the frequency control registers. The SRAM wait states are controlled by the Bus State Controller (BSC).

CKIO setup

A 16 MHz crystal oscillator is fed to the SH4 which has an on-chip PLL for generation of internal and external frequencies. The Clock Pulse Generator (CPG) has three clocks which are set externally by the external mode pins MD[2..0]. The CPG is set to clock operating mode 3 by these pins. The default frequency of CKIO is therefore 64 MHz (and the internal clock runs at 192 MHz). During testing of the adapter board, however, electromagnetic interference (EMI) was detected on the adapter board. The existence of EMI on a data bus is easily detected using the transmission of an up counter value over the bus. When the counter is 3 bits wide, all the values are transferred over the interface successfully. However, as the counter becomes wider, the number of parallel wires containing a logic 1 increases. In other words, a 3 bit counter will transition from 111 to 000 and a 15 bit counter from 111111111111111 to 000000000000000.

When a large number of bits on the bus change their value, EMI will manifest as a "remnant" voltage on certain wires. This is due to mutual induction – the current in one wire creates a magnetic field, and that magnetic field affects the adjacent wires [28]. For example, a change from 2^{15} to 0 fails because the actual value on the bus is, for example, 000000010000000 when the value is latched. The stray 1 is due to the induced magnetic field.

A simple solution to measured EMI is reducing the clock speed of the interface. This provides the wires time to settle to their correct values. For this reason the frequency of CKIO was halved. Substituting this new value of CKIO into Equation 3.3.1 yields

$$r = \frac{32 \times 10^6}{3} \times 32 \simeq 341 \times 10^6 \,\text{bps} \tag{5.3.1}$$

Table 5.1: Simplified OBC expansion port firmware model.

Address	Read	Write	Description
0x00	✓	✗	TM receive register
0x01	✗	✓	TM transmit register
0x02	✓	✗	status register
0x03	✓	✓	control register

The required data rate of 7 392 000 bps still falls within this upper limit and the effective data rate over the expansion port was not impacted negatively by this change. This was verified during subsequent measurements.

After halving of CKIO's frequency, EMI effects were not measured anymore at latch time and reliable data transfers were achieved. The setting of CKIO involves writing set up values of 0xA567, 0x5A00 and 0xE23 to registers WTCNT, WTCSR and FRQCR respectively.

BSC setup

The bus state controller controls the number of wait states inserted during SRAM access operations over the expansion port. The minimum number of wait states is aimed for to maximise the data throughput rate. The BSC inserts the same number of wait states for either read or write operations.

For a write operation, data becomes valid on the first falling edge of CKIO after the assertion of $\overline{\text{WE2}}$. The flip-flops in firmware only latch data on rising edges of CKIO. This means that the data must be latched after the first rising edge of CKIO when $\overline{\text{WE2}}$ is asserted. Zero wait states are needed to accomplish this.

For read operations, the SH4 expects valid data on the data bus after N rising edges of CKIO, where N is the number of wait states. However, the data must first be "popped" out of the FIFO onto the data bus into a register. This takes two clock cycles: one for popping from the FIFO, and one for latching into the register. Two wait states must therefore be inserted. This is done by writing 0x400 into the WCR2 register (see Figure A.1 in Appendix A).

5.3.2 Simplified Firmware Interface

A simplified model of the firmware, summarised in Table 5.1, is used to hide firmware implementation details from the resource manager. The resource manager reads and writes to four registers when accessing the expansion port. The resource manager also contains an interrupt handler which is triggered by the firmware. The rest of the firmware implementation is hidden from the resource manager. This model allows the greater firmware transmit and receive paths to be implemented in any manner without needing to update the resource manager. Each of the registers is now detailed.

Table 5.2: OBCS bit definitions.

Bit	Name	Reset Value	Description
31-8	Reserved	-	Not used
7	TMWPF	0	Programmable full threshold of the TM write FIFO
6	TMWPE	1	Programmable empty threshold of the TM write FIFO
5	TMWF	0	Full threshold of the TM write FIFO
4	TMWE	1	Empty threshold of the TM write FIFO
3	TMRPF	0	Programmable full threshold of the TM read FIFO
2	TMRPE	1	Programmable empty threshold of the TM read FIFO
1	TMRF	0	Full threshold of the TM read FIFO
0	TMRE	1	Empty threshold of the TM read FIFO

OBC Expansion Port Status Register (OBCS)

The status register OBCS is a 32-bit read only register that is used by the resource manager to poll the status of the buffers on the FPGA. The Telemetry (TM) write FIFO is the buffer containing TM frame – which was sent from the TM write module on the SH4 – and must be written to the EDAC firmware components. The TM read buffer contains decoded data from the EDAC components which are destined for the TM protocol on the SH4. The bits of the OBCS register are defined in Table 5.2. The bit names used conform to the Xilinx FIFO specification [48] and they are depicted in Figure 5.8.
The status register is read by the resource manager interrupt handler. The interrupt processor thread then checks the TMWPF bit to determine if the TM write FIFO has available space. The interrupt processing thread also checks the TMRE bit to determine whether there is data on the FPGA waiting to be read.

OBC Control Register (OBCC)

The control register OBCC is a 32-bit read/write register used for the OBC to set flags which can be read by the FPGA. The bits of the OBCC register are defined in Table 5.3. Currently the only bit used in this register is RST, at position 0, which allows the OBC to reset the FPGA and DSP hardware. Writing a 1 to this bit will generate a firmware reset on the next rising edge of the expansion port bus clock.

Table 5.3: OBCC bit definitions.

Bit Number	Bit Name	Reset Value	Description
31-1	Reserved	-	Not used
0	RST	0	User programmable reset

TM Registers

When writing to the TM write register, data is clocked into the TM write FIFO. Reading from the TM read register reads data from the TM read FIFO.

5.3.3 Resource Manager Components

QNX provides various layers which are used to build resource managers. These layers (as named within the QNX documentation) are:

- iofunc layer (the top layer)
- resmgr layer
- dispatch layer
- thread pool layer (the bottom layer)

iofunc layer

This layer provides the standard set of functions which provide the device with a POSIX "appearance" to the outside world. This layer ensures that all POSIX calls made to the device return without error. This layer was used because it removes the need to implement every single POSIX function. Only four of the POSIX file functions, *open(), read(), write()* and *devctl()* are implemented.

The iofunc layer also provides helper functions which are used throughout the resource manager. These helper functions simplify the implementation of the device functions. All *iofunc_ ** functions are part of the iofunc layer.

resmgr layer

The resmgr layer handles most of the resource manager details. It waits for and examines incoming messages, and calls the appropriate handler to process each received message. Functions beginning with *resmgr_* form part of the resmgr layer.

The resmgr_attach() function is used to create a name in the pathname space:

```
resmgr_attach (dpp, &resmgr_attr, "/dev/exp", _FTYPE_ANY, 0,
               &cfuncs, &ifuncs, &attr)
```

The *resmgr_block()* function blocks the resource manager until a client connects to it. When a client has connected, *resmgr_handler()* is called to process the message by calling the appropriate handler function (open, write, read, etc).

dispatch layer

The dispatch layer is useful for handling other types of IPC available in QNX, such as pulses, and is not used in the implementation of the resource manager.

thread pool layer

This layer allows the resource manager to respond to multiple simultaneous read or write calls. Because the OBC expansion port is physically a half duplex interface, it is only necessary to service one of the TM reader or writer threads at a time. This layer is therefore not used in the resource manager implementation. The resource manager is, however, multi-threaded, but this is to allow efficient handling of the interrupts, and the standard POSIX pthread library was used for this purpose.

5.3.4 POSIX File Functions

The *io_open()* handler function implementation is reduced to a single line of code, provided that the *iofunc_open_default()* helper function is used:

```
int io_open (resmgr_context_t *ctp, io_open_t *msg,
             RESMGR_HANDLE_T *handle, void *extra) {
   return (iofunc_open_default(ctp, msg, handle,extra));
}
```

The operations performed by the read and write function handlers are expressed as flowcharts in Figures 5.2 and 5.3 respectively.

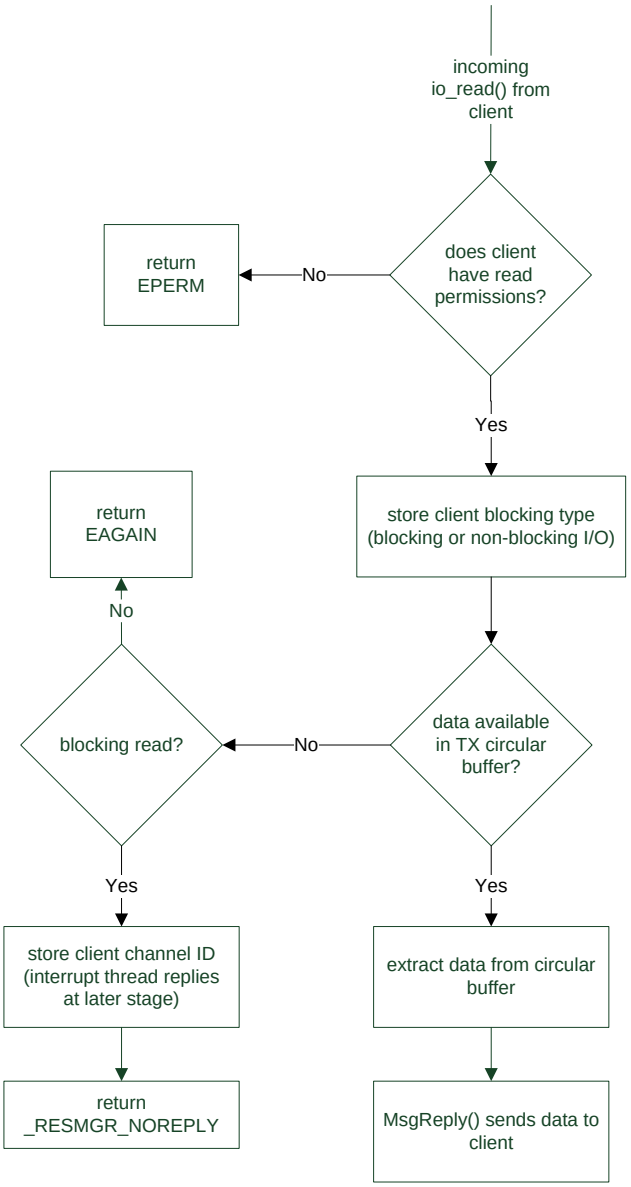

Figure 5.2: io_read() operations.

The *io_devctl()* function handler checks the integer value passed to it in a case statement and takes action accordingly. At the time of writing the only value recognised is 0, which causes the resource manager to write a value of 0x01 into the OBCC register. This value generates a reset on the FPGA and the DSP hardware. Adding more devctl operations is as simple as adding more clauses to the case statement in this function.

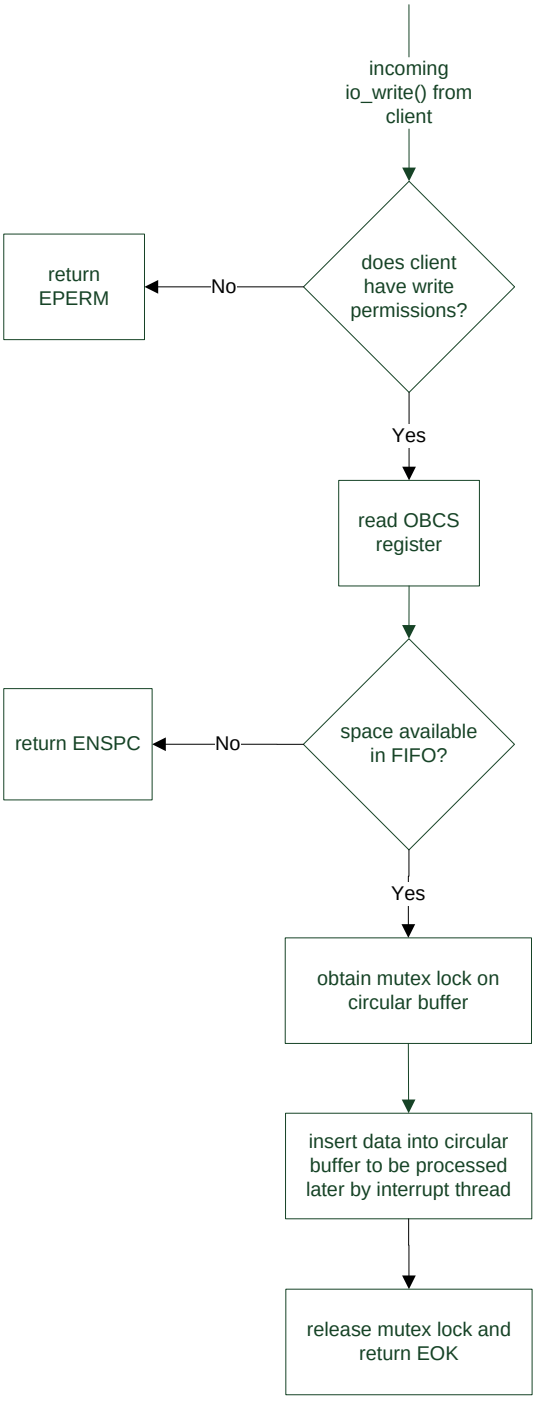

Figure 5.3: io_write() operations.

5.3.5 Interrupt Handler

The interrupt handler module consists of two components: an Interrupt Service Routine (ISR) and an interrupt processing thread.

ISR

The ISR runs with the highest possible priority so it must be kept short to avoid creating a bottleneck. The ISR:

- services the hardware by reading a register,
- updates a data structure shared between the ISR and the interrupt processing thread and
- signals the resource manager application that an event has occurred.

All of these tasks are accomplished with the following code:

```
const struct sigevent *intHandler(void *arg, int id) {
  InterruptMask(intr,intId);
  status_reg = regbase[REG_STATUS];
  return (&event);
}
```

The first thing the ISR does is to mask any further interrupts. Thereafter it reads the status register from the FPGA and in doing so notifies the FPGA that the interrupt was received. It then returns a pointer to a special type of eventm SIGEV_INTR, that only has a meaning for the interrupt processing thread. The event structure is initialised in the resource manager's *main()* function.
Attaching the ISR to the interrupt handler is handled by the QNX operating system and is done using the *InterruptAttach()* function (where *intHandler()* is the ISR) :

```
intId = InterruptAttach( intr, intHandler,
                         (void *) &status_reg, 0, 0);
```

A pointer to status_reg, a 32-bit unsigned integer variable, is passed to InterruptAttach. This couples the ISR to the variable.

Interrupt Processing Thread

This thread does the actual "work" when an interrupt is received. The thread waits for notification from the ISR that an interrupt occurred as part of a *while(1)* loop:

```
InterruptWait (NULL, NULL);
```

When *InterruptWait()* unblocks, the ISR has returned a SIGEV_INTR, indicating that some form of work needs to be done. The interrupt processing thread then checks the value in status_reg. If there is data waiting to be read from the TM read FIFO, it must be read from the FIFO and inserted in the receive circular buffer to be accessed by later by a *read()* call. Similarly, if there is data in the transmit circular buffer from a previous *write()* call, that data must be written to the TM write FIFO (given that there is space in the FIFO). Finally the thread unmasks interrupts by calling the *InterruptUnmask()* function.
The conditions under which interrupts are generated are explained in Section 5.4, but a problematic scenario was identified wherein interrupts would constantly be generated when the TM write FIFO was waiting for data. This was remedied by implementing selective interrupt unmasking.

Interrupts are unmasked under four conditions: when the interrupt processing thread launches, after a block of data is written from the TX circular buffer to the TM write FIFO, after a block of data is read from the TM read FIFO into the RX circular buffer and after a successful *io_write()* call. These four conditions ensure that buffer overflows do not occur and that the unnecessary processing of interrupts is avoided.

As is evident from the figures, there are two threads manipulating the circular buffers at the same time. The circular buffers were not designed for "asynchronous" read and write access. A mutex was therefore used to ensure that only one thread would be accessing a circular buffer at any time.

5.3.6 Testing

Because the design of the resource manager is so tightly coupled to the design of the FPGA firmware, testing the resource manager also involved testing the firmware. This is classified as an integration test and is described in Section 5.7.

5.4 FPGA Firmware

In this section the components of the firmware are implemented and tested. Certain integration tests between components are inevitable at this stage to verify that the firmware is operating as expected. These tests are described in this section.

5.4.1 OBC SRAM Interface Implementation

The expansion port on the SH4 OBC provides a 32-bit memory mapped SRAM type interface to external devices. The SH4 is the bus master for all transfers over this interface. When $\overline{\text{CS2}}$ goes low, it indicates that the SH4 is using

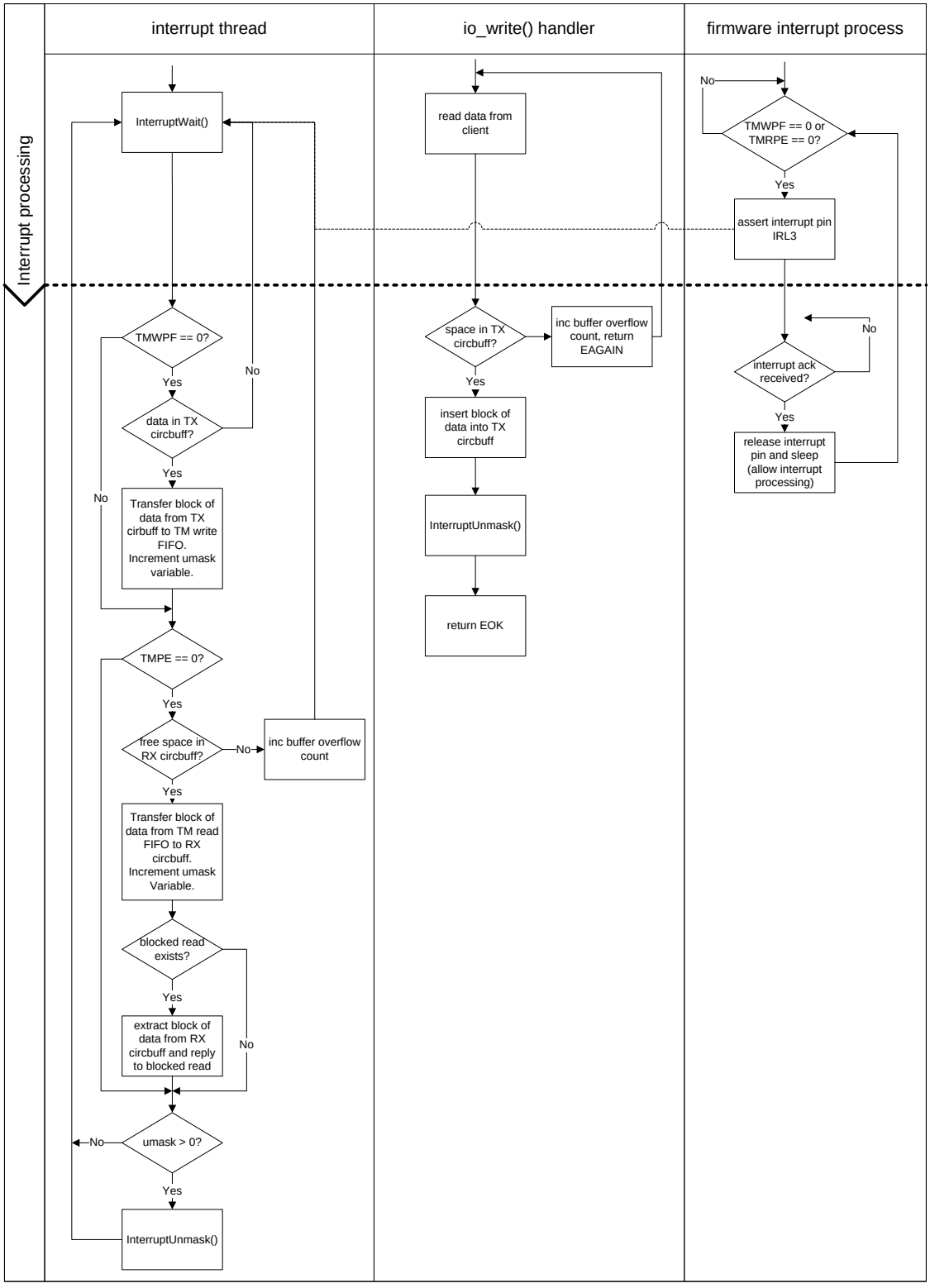

Figure 5.4: Interrupt interaction detail: write().

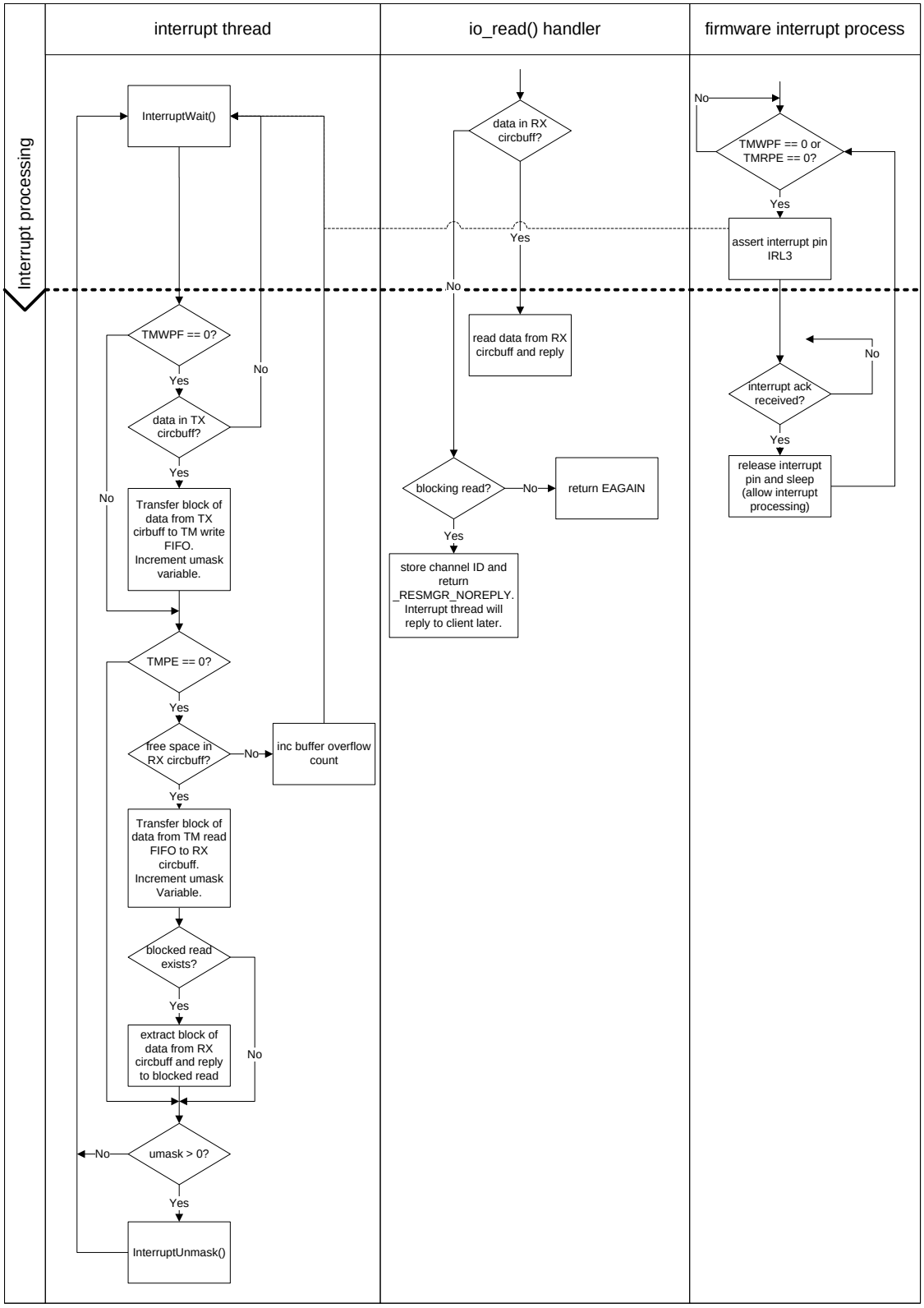

Figure 5.5: Interrupt interaction detail: read().

the expansion port to access external memory area 2. As described in Section 5.3.1, the timing of the waveforms on the expansion port are determined by control registers on the SH4.

To verify the setup values written to these control registers, the expansion port waveforms were measured using a logic analyser. The waveform timings for the read and write operations are presented in Figures 5.6 and 5.7 respectively.

The simplest test is to test the reading and writing of data between the OBC and the FPGA by using a single register.

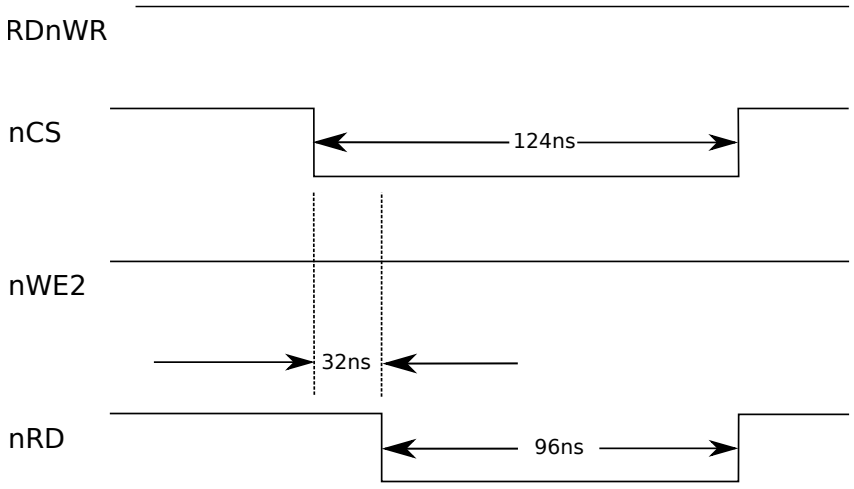

Figure 5.6: Expansion port read timings.

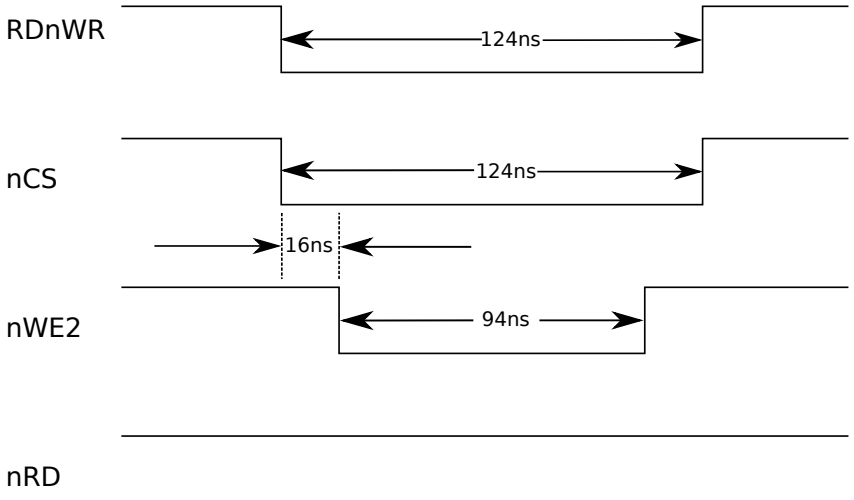

Figure 5.7: Expansion port write timings.

5.4.2 FIFO Implementation

The FIFOs generated by the Xilinx core generator are first tested with a VHDL test bench in the ISE simulator to verify their operation. Once their operation has been verified they are incorporated into the simple register based firmware. Data is latched into the FIFO on a synchronous rising edge of the wr_en and wr_clk input ports. This means that wr_en must be high for a single clock period or else the value at the data input will be written twice. This is accomplished using the edge detector circuit depicted in Figure 5.10.

When implementing FIFOs, the unwanted conditions are underflows and overflows. These are considered critical errors and are prevented in the design. They are not dealt with other than incrementing a counter.

TM Read FIFO Underflow

If a TM Read underflow occurs, stale values will be read from the FIFO and inserted into the RX circular buffer by the interrupt thread. The interrupt thread therefore checks the value of the programmable empty threshold before reading a block of data. The programmable empty threshold is specified by an integer input port to the FIFO. When the number of words in the FIFO is less than or equal to this integer, the programmable empty bit has a value of one. Under all other circumstances the bit has a value of zero. The threshold is set to the size of the data block minus one. The threshold is depicted in Figure 5.8.

TM Write FIFO Underflow

If a TM write underflow occurs, the FEC encoder will encode a stale value. The process that drives the FEC encoder therefore checks the empty bit. If the empty bit is zero, there is at least one word in the FIFO, and that word is sent to the encoder.

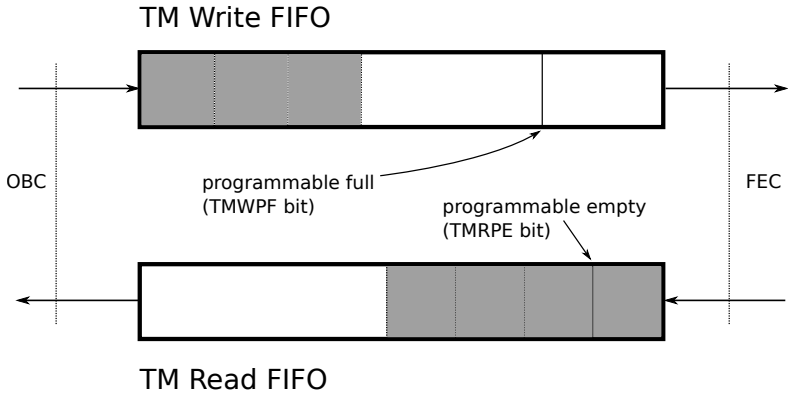

Figure 5.8: FIFO thresholds.

TM Read FIFO Overflow

If a TM read overflow occurs, the FEC decoder will overwrite words that have not been read yet. This is avoided by checking the full flag. If the flag is zero, there is space in the FIFO and a single word is written to the FIFO.

TM Write FIFO Overflow

A TM write overflow is avoided by checking the programmable full bit. When this bit is zero, the word count in the FIFO is less than or equal to the programmable full threshold value. The bit equals one otherwise. The programmable full threshold is set to 75%. The threshold is depicted in Figure 5.8.

5.4.3 Control Process

A control process is implemented in the firmware to monitor the value in the control register and to monitor the interrupt conditions. When the reset bit in the control register is set, this process generates a reset signal which reinitialises all state machines, registers and buffers.

When the interrupt conditions are met, this process handles the generation of interrupts. The process generates an interrupt when:

- the TM Read FIFO programmable empty bit equals zero and
- the TM Write FIFO programmable full bit equals zero.

The thresholds determining these bits are depicted in Figure 5.8. Both interrupt conditions are satisfied in the diagram and an interrupt will be generated.

After generating an interrupt, the process waits for an interrupt acknowledge, which is generated when the interrupt processing thread reads the value of the status register. When the interrupt acknowledge is received, the interrupt pin is released (de-asserted). The process then pauses briefly to allow processing of the interrupt before checking the interrupt conditions again. This is repeated ad infinitum.

5.4.4 Testing

VHDL test benches were used to verify the behaviour of the edge detector, register and FIFO components.

5.5 DSP SRAM Interface

The DSP SRAM interface on the DSP56311 EV Kit does not contain a clock i.e. it is asynchronous. Asynchronous SRAM access is therefore implemented

in firmware by using an edge detector similar to that used on the OBC SRAM interface.

During implementation of the modem by Ewald van der Westhuizen it became apparent that the DSP56311 did not support a sufficiently high clock speed for real time modem operation. The DSP56321 is a faster chip and was placed on the EV Kit, toegether with a faster 20 MHz crystal.

The timing of the SRAM interface waveforms is determined by the speed at which the DSP56321 runs and the number of wait states configured in the Address Atribute Register. The DSP56321 has a digital PLL which is set to run at 275 MHz by the modem software. A single read/write cycle over SRAM takes one clock cycle. The ML501 FPGA is fed by a 100 MHz crystal so wait states must be inserted.

The maximum number of wait states, 31, were inserted to provide a safety margin against EMI effects. With 31 wait states, 24-bit read/write operations take 32 clock cycles. This provides a data rate of $\frac{275\times10^6}{32} \times 24 = 24.59\,\text{MB/s}$ which is greater than the required data rate calculated in Chapter 3. A value of 0x1F is written to the Bus Stace Control (BSC) register to insert 31 wait states during SRAM accesses.

External SRAM access is memory mapped to I/O by writing 0x14C11 to register AAR0. All addresses in the range 0x10000 - 0x1F000 are treated as external SRAM memory addresses.

5.5.1 Testing

A VHDL test bench was used to verify that the operation and timing of the DSP SRAM firmware module functioned correctly.

5.6 Downlink Data Radio Implementation

The XTend OEM module supports an RF data rate of 115 200 bps and is accessed via a UART. An RS-232 level converter is used to enable interfacing with standard serial ports. All of the SH4 OBC serial ports are connected to components. A single RS-232 port on the ML501 FPGA board is available and was identified as being a suitable electrical interface to use for the XTend module.

A UART module, based on Ken Chapman's Compact-K UART [9], was implemented in firmware on the ML501 FPGA. This UART was then accessed via the resource manager on the SH4. The resource manager exposed the data radio as `/dev/ser3`. The baud rate was hard coded into the firmware as 115 200 bps but it would be possible to implement a programmable baud rate UART using a configuration register.

The downlink data radios support both hardware and software flow control. However, by setting the RF baud rate of the radios to 115 200 bps and the data

interface baud rate to 57 600 bps, it is possible to prevent buffer overflows from occurring by designing the resource manager to check the status of the UART FIFO on the FPGA before allowing client software to perform a write.

5.7 Low Level Integration Testing

In this section the lowest level intercomponent testing is described. Basic building blocks are verified and electrical stability of inter-component connectivity is tested.

5.7.1 OBC SRAM Interface Low Level Loopback Rest

This is the simplest test to confirm that 32-bit data words can be transferred between the OBC and the FPGA.

Test Setup and Procedure

The gated flip-flop in Figure 5.9 is synthesised on the FPGA, with its output looped back to the expansion port data bus. The input is gated with the $\overline{\text{CS2}}$ signal to ensure that only Area 2 writes on the expansion port are handled. The $\overline{\text{WE2}}$ enable signal from the SH4 must be delayed by one clock cycle because the data is only valid after the second rising edge of CKIO [39]. The test setup is depicted in Figure 5.9. SunSpace provides convenient command line utilities for accessing memory locations on the SH4. Mempoke writes to memory and

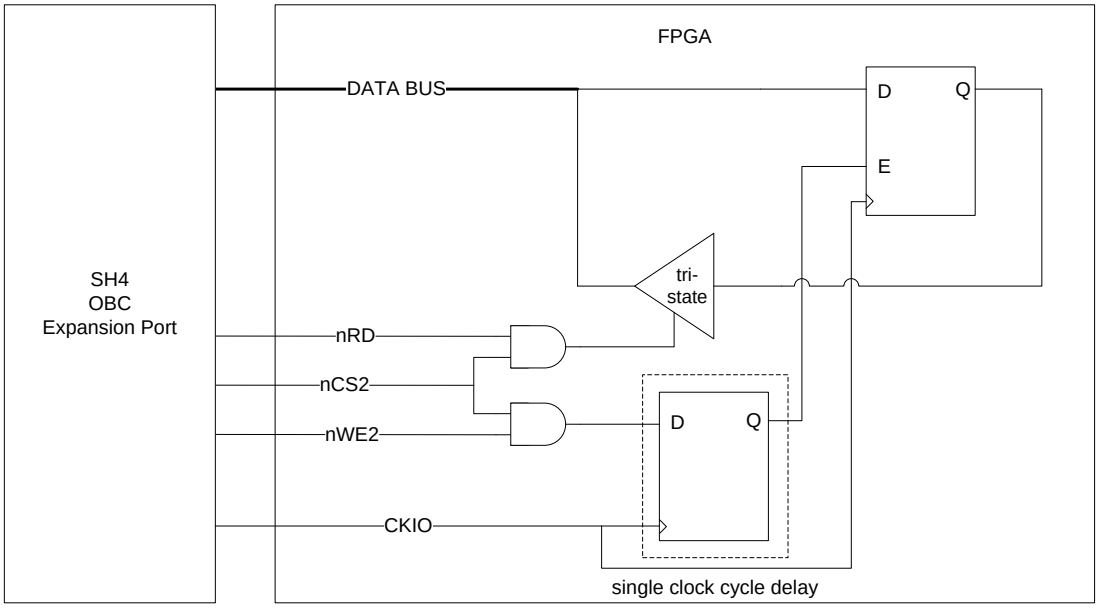

Figure 5.9: Register loopback test.

mempeek reads from memory. Because the expansion port is implemented as memory-mapped I/O, writing to the firmware register is accomplished with:

```
mempoke -r 32 0x88000000 42
```

The -r switch specifies 32-bit memory access. All addresses in the 0x88000000 range are Area 2 addresses. Reading from the register is done with:

```
mempeek -r 32 0x88000000
```

Expected results

The value written with mempoke should be returned by the mempeek command.

Actual results

The output of the commands display the result:

```
# mempoke -r 32 0x88000000 5
MemPoke, v0.1, Compiled on May  7 2007 11:41:08
Writing to address = 0x88000000, value = 0x00000005
# mempeek -r 32 0x88000000
0x88000000: 0x00000005 (5) [0000  ...   0000 0101]
```

Analysis of Results

From the output of mempeek it seems that the implementation of the register was successful. However, the full range of 32-bit values should be transferred, read and verified repeatedly by an automated stress test program to ensure that no glitches or EMI effects exist on the interface. This was done during the FIFO test described next.

5.7.2 OBC SRAM with FIFO Loopback Test

This is the simplest test using a single FIFO and some registers.

Test Setup and Procedure

The firmware depicted in Figure 5.10 was synthesised on the FPGA. A C program was written which would write N 32-bit values, where N is the capacity of the FIFO (in this case the FIFO can hold 256 32-bit words), and then read N words. The read values were compared to the written values and an error was raised if the values did not match. This process was placed inside a *for()* loop which ran 2^{32} times. After each iteration of the loop, each element in the collection of values to send was incremented. This test ensured that the entire range of values were tested.

Expected Results

The 256 values read from the FIFO should correspond to the values sent to the FIFO.

Actual Results

The values read from the FIFO were equal to the values written to the FIFO.

Analysis of Results

All synthesised components were operating as expected.

5.7.3 DSP SRAM with FIFO Loopback Test

The testing procedures for the DSP SRAM interface tests were very similar to the OBC SRAM interface tests and will not be described here. Both the single register and FIFO loopback tests passed and it was concluded that data could be reliably transferred between the DSP and the FPGA. The firmware used for the FIFO loopback test is depicted in Figure 5.11.

5.7.4 Data Radio Testing

This smoke test is to verify that the RS-232 level conversion circuitry is functional and that data can be sent over the air via the radios.

Test Setup and Procedure

The test was performed using a Fit-PC, two USB to RS-232 converters and two XTend data radios with antennas. Both radios were set up in transparent

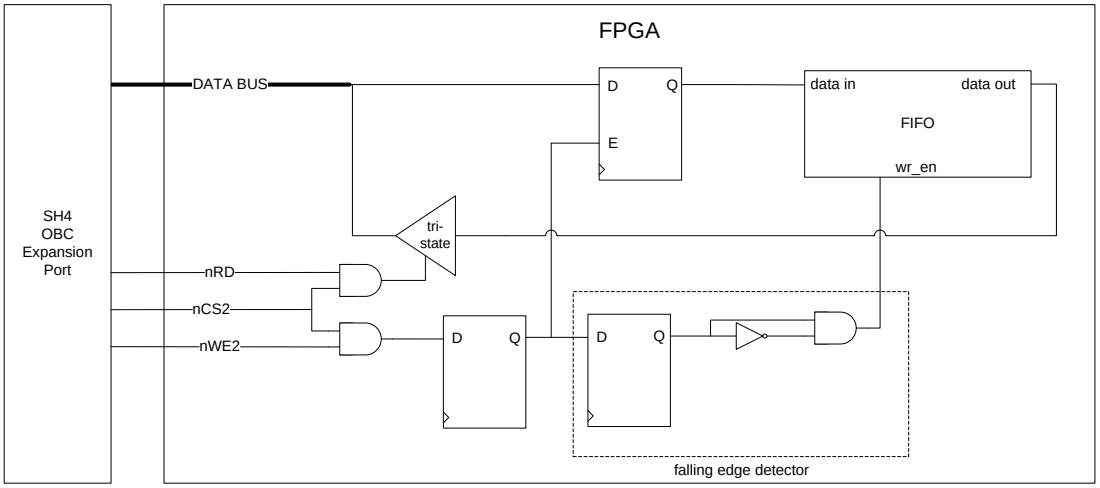

Figure 5.10: OBC single FIFO loopback test.

mode. Data was transmitted to one serial port with:

```
$ echo "1234567890" > /dev/ttyUSB0
```

The other serial port was monitored for incoming data:

```
$ cat /dev/ttyUSB1
```

Expected Results

The transmitted string should be received without loss.

Actual Results

The following output was noted:

```
$ cat /dev/ttyUSB1
1234567890
```

Analysis of Results

The test string was successfully transmitted over the air using the data radios. The radios perform according to specification. Note that this test also verifies that the downlink receiver is functioning properly.

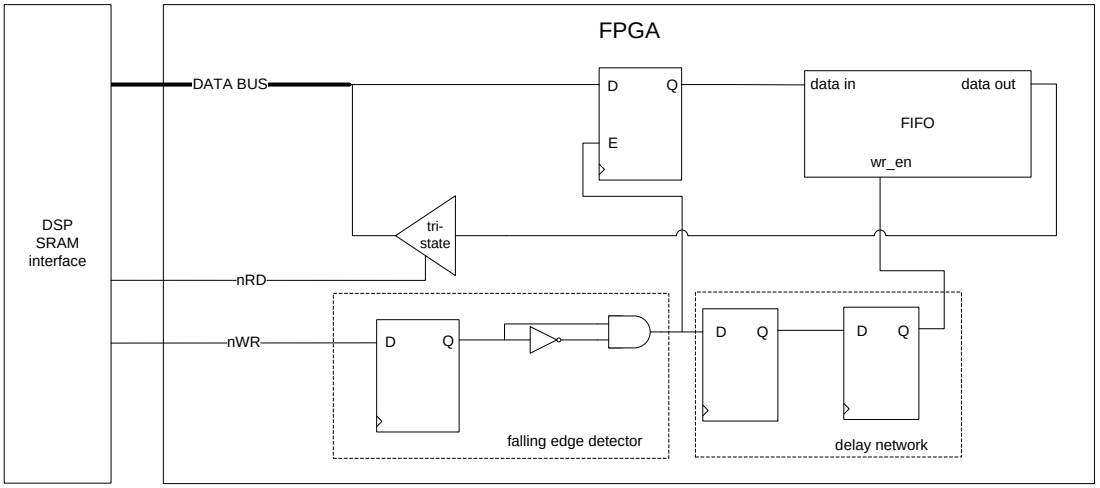

Figure 5.11: DSP single FIFO loopback test.

Chapter 6

Ground Station Implementation

The implementation of the ground station system presented in Chapter 4 is explained in this chapter. As with the implementation of the payload, the top-level components are constructed from the ground up. Basic smoke tests of the bottom level components are performed after each implementation step. Detailed integration and system testing is presented in Chapter 7.

In general, component choice is more flexible for the ground station because:

- single event effects (such as latchups caused by ions or electro-magnetic radiation) do not have to be considered,
- there is more physical space for component layout and
- more power is available on the ground.

The ground station RF hardware is discussed in terms of the uplink transmitter and the downlink receiver. The firmware on the FPGA is then discussed as is the modem integration.

6.1 Downlink Receiver

The XTend OEM module used for the downlink was connected to the ground station PC via a Prolific pl2303 USB-to-UART converter [38]. The pl2303 does not provide hardware flow control. Software flow control is disabled on the PC driver. Flow control is not required because the radio only receives data. The baud rate of the serial port is set to 57 600 bps with the following command:

```
stty 57600 -F /dev/ttyUSB0 ixoff
```

The smoke testing of the data radios performed in Chapter 5 showed that this setup was sufficient to allow reception of data on the downlink using the XTend module.

6.2 Uplink Transmitter

The uplink transmitter is required to transmit the modulated data bits as a complex I/Q signal of appropriate power and frequency. To achieve this the following tools must be used:

- a means to transport the raw data bits from the PC to the DAC,
- a DAC to escape the digital domain of the modulator,
- a reconstruction filter at the DAC output,
- a quadrature mixer to create a complex RF signal from the I/Q signals,
- an amplifier stage to achieve the required signal strength and
- an antenna.

As outlined in Chapter 4, the Spartan-3E FPGA development board is used for physical data routing between the various components with digital interfaces.

6.2.1 DAC Implementation

As described in Chapter 4, a 16-bit DAC was sought with a minimum sample rate of 76 800 samples per second. Because the Spartan-3E board was used there was a constraint on the number of available output pins. A 16-bit, 500 MSps DAC evaluation board was available from the Multiple Channel Communications Payload project, but this DAC has 32 parallel data inputs. The Spartan-3E board has a total of 43 I/O pins [51], 32 of which are required for interfacing to the DSP modem. An alternative solution was therefore sought.

The Spartan-3E board has an on-board quad-channel, 12-bit, voltage output DAC: the Linear Technology LTC2624. The LTC2624 has a 2-wire serial input which makes it suitable for the limited Spartan-3E board I/O pin count.

6.2.2 DAC DDS Output Test

Evaluation of DAC performance requires analysis of many parameters [29]. This test's focus was on ensuring that the firmware was operating the DAC correctly. The focus was not on the DACs performance.

Test Setup and Procedure

To test the implementation of the DAC firmware a Direct Digital Synthesis (DDS) module was used to generate 12-bit sine and cosine values at a sampling rate of 230 400 with a signal frequency of 38 400 Hz. These were then fed to two of the output channels. The outputs were measured using an oscilloscope.

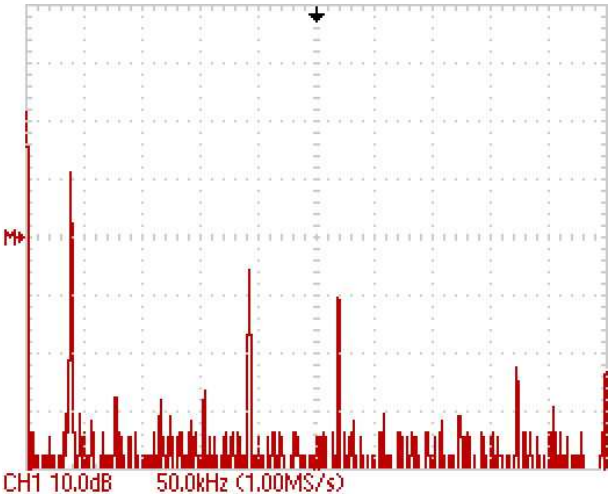

Figure 6.1: FFT of a single DDS output channel.

Expected Results

It was expected to see a discrete valued time signal on the DAC outputs.

Actual Results

The Fast Fourier Transform (FFT) of the sinusoidal output is displayed in Figure 6.1.

Analysis of Results

The main frequency component is clearly visible along with a fifth harmonic with roughly 15 dB of attenuation. This test confirmed that the DACs were being operated correctly by the firmware.

6.2.3 Reconstruction Filter Implementation

The Sallen-Key filter topology can be used to implement second-order active filters [43] with linear phase responses. The phase response is important because phase modulated signals will be passed through the filter. The low-pass configuration used is depicted in Figure 6.2. The cutoff frequency for this circuit is given as $f = 1/2\pi RC$, with $C_1 = \sqrt{2}C$ and $C_2 = \sqrt{2}C/2$. The desired cutoff frequency is 38 400 Hz as this is the bandwidth of the modulated message signal from the modem.

Using standard 1 K resistors and choosing $C_1 = 3.9\,\text{nF}$ results in $C_2 = 1.95\,\text{nF}$. A value of 1.95 nF was not readily available, so 1.85 nF was used instead. LF351 wide-bandwidth operational amplifiers with supply voltages of $\pm 10\,\text{V}$ were used [35].

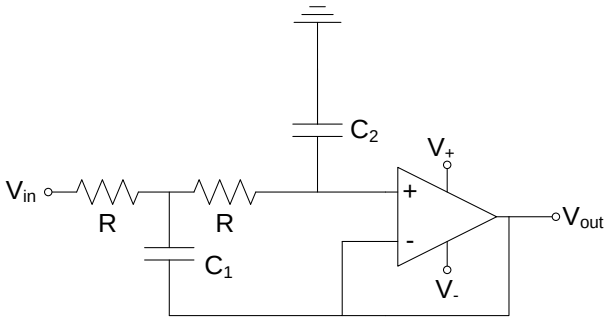

Figure 6.2: Sallen-Key low pass filter.

6.2.4 Reconstruction Filter Frequency Response Test

The filter circuit is now tested to verify the cutoff frequency.

Test Setup and Procedure

The same 38 400 Hz signal generated previously is fed to the filter input. The FFT of the filter's output is taken.

Expected Results

It is expected that the frequency component at 38 400 Hz remain unattenuated. Any higher frequencies should be filtered out.

Actual Results

The FFT of the filter output is displayed in Figure 6.3.

Analysis of Results

Only the desired frequency component is present. When compared with Figure 6.1, it is evident that the components at higher frequencies have been filtered out. The filter is therefore suitable for use with the rest of the system.

6.2.5 Quadrature Mixer Implementation

The ComBlock COM-4001 was used as a quadrature upmixer. The COM-4001 has a built-in mixer to mix the complex baseband signal to the desired carrier frequency, the output of which can then be fed to an amplification stage. It expects 1 V peak-to-peak input signals with a 0.85 V DC offset [11]. This is because the inputs of the COM-4001 are fed to operational amplifiers which do not operate well near the 0 V or 3.3 V rails. In practice, the COM-4001 can

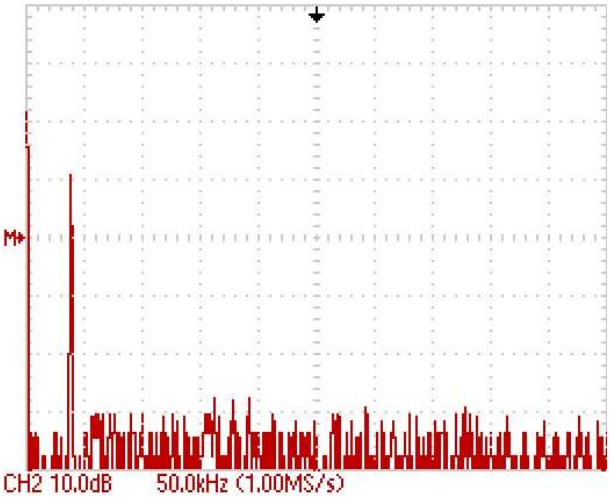

Figure 6.3: FFT of a single reconstruction filter output channel.

accept a DC offset anywhere within 0.35V of the rails. This means that, for a $1\,V_{pp}$ signal,

$$0.35 \leq V_{in} \leq 2.95 \tag{6.2.1}$$

Ideally, the full-scale output of the DAC must satisfy these criteria. The full-scale output of the DAC is defined by five reference pins:

- REF LO, the floor (zero) of the full scale output
- REF A, B, C and D, each of which defines the ceiling of the full scale output for the respective channel.

The obvious solution is to connect 0.35 V to REF LO, and 1.35 V to REF C and D, where channels C and D are the output channels of the I and Q message signals, respectively. According to the data sheet, however, the maximum value of REF LO is 100 mV [33]. The obvious solution is therefore not feasible. Two other possibilities present themselves:

- add an analogue DC offset
- add a digital DC offset

Construction of an analogue circuit requires circuit design, sourcing of components and testing. This is more time consuming than implementing a digital solution using VHDL. Adding a constant digitally, however, would involve discarding a portion of the full scale output provided by the DACs. This loss of resolution is not ideal, so it was decided to add the DC offset using analogue components.

The output of the DACs are AC waveforms. The simplest circuit for adding DC offset to an AC waveform is presented in Figure 6.4. The DC voltage at

V_{out} is given by the relation:

$$V_{out} = V_{in} \times \frac{R_2}{R_1 + R_2} \tag{6.2.2}$$

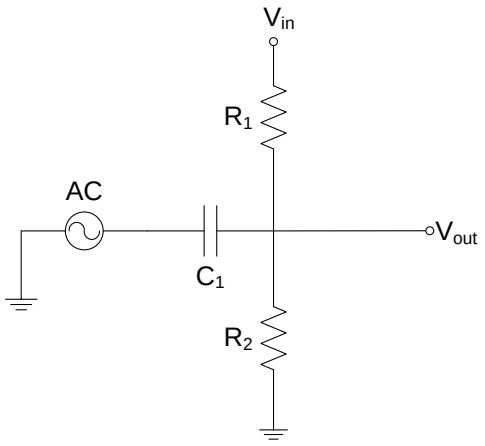

Figure 6.4: DC offset adding circuit.

The capacitor C_1 in Figure 6.4 decouples the DAC output from the rest of the circuit and forms a high pass filter (together with the load impedance connected to it). The choice of a value for C_1 is not an exact calculation, but it can be estimated as follows. The input impedance of the COM-4001 is not known. It is only known that it has a high impedance. Assume it to be 1 MΩ. The equivalent impedance of the voltage divider is $R_1 \parallel R_2 = 500\,\Omega$. So C_1 connects to a load of about 500 Ω. C_1 therefore has a minimum value of 16 nF so that the 3 dB point will be below the lowest frequency component of 19 200 Hz.
The values for R_1 and R_2 are determined by two factors:

- the value of V_{out} in Equation 6.2.2 must be greater than or equal to 0.85 V and
- the current flowing in the voltage divider should be large compared with the current flowing into the COM-4001 input [26].

$V_{in} = +1.8$ V is used because a fixed regulator with this value as output is readily available and closer to the required DC offset than the +5 V supplies of the COMBLOCKs and the Spartan-3E. Substituting $V_{in} = 1.8$ V, $R_1 = 1$ KΩ and $V_{out} = 0.85$ V into Equation 6.2.2 yields:

$$R_2 = \frac{0.85}{0.95} R_1 = 894\,\Omega$$

The closest standard value to 894 is 820, but the DC bias should be above 0.85 V so R_2 was chosen as 1 KΩ, giving a DC bias of 0.9 V.

6.2.6 DAC Output Test

The loaded and unloaded AC outputs of the DAC are measured in this test.

Test Setup and Procedure

The circuit in Figure 6.4 was constructed and connected to both of the DAC outputs. The system was powered up and the outputs measured with an oscilloscope. A 500 Ω resistor was connected between V_{out} and ground and the measurement repeated.

Expected Test Results

A 1 V peak-to-peak sinusoid with a DC offset of 0.9 V should be measured in both cases.

Actual Results

It was verified that the peak-to-peak value was 1 V and that the DC offset was 0.9 V. A 500 Ω resistor was connected between V_{out} and ground and the measured waveform remained unchanged.

Analysis of Results

This test verified that the circuit added the desired DC offset of 0.9 V and did distort the output signal under load.

6.2.7 DAC COM-4001 DDS Test

In this integration test the DC offset circuit was connected to the input of the COM-4001.

Test Setup and Procedure

The DC offset circuit from the DAC is connected to the input of the COM-4001. The FPGA subsystem is powered up. Then the COM-4001 system is powered up. The sine and cosine outputs from the DAC are measured to ensure that they remain unchanged under load.

Expected Results

It is expected that the same AC waveforms are seen at the outputs of the DACs after the COM-4001 is powered up.

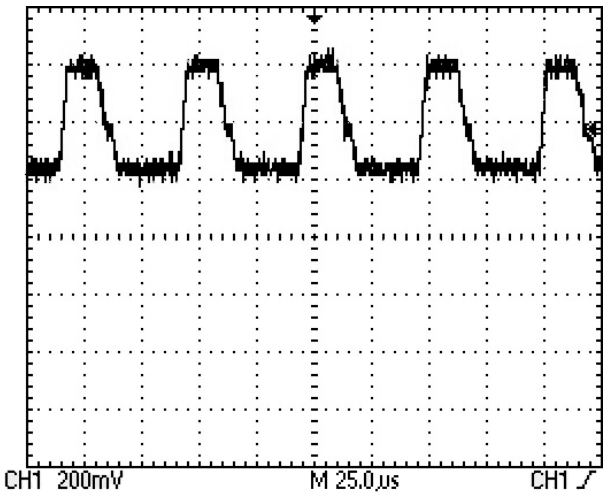

Figure 6.5: Distortion noted on DAC outputs.

Actual Results

When power was supplied to the COM-4001, the signal distortion depicted in Figure 6.5 was noted on the DAC outputs. Specifically, clamping was noted at voltages near 0.35 and 1.35 V .

Analysis of Results

The clamping could have been caused by the following conditions:

- the DAC output voltage headroom near the 0 V rail was too high or
- the DC-bias level was lower than the value the COM-4001 expected, i.e. the value in the data sheet was wrong.

Increasing the DC-bias offset to 1V did not affect the distortion of the waveform, neither did decreasing the peak-to-peak value of the signal. For this reason it seemed as if the DAC was having difficulty maintaining output voltages near the lower rail. This could be due to the fact that the DAC was sinking too much current at these voltages [33]. It was therefore decided to place a buffer between the DAC output and the COM-4001 input.

The standard LM324 operational amplifier was chosen because it was readily available. A voltage follower circuit was implemented. The output signal remained distorted except the clamping was happening at the positive rail too. A differential amplifier with unity gain was also tested with failure.

Due to time-constraints it was decided to add the DC-offset in the digital domain, although further tests that could possibly solve this problem are:

- supplying the LM324 with rails other than 0 and 5 V, such as ± 6 V,

- testing with a different operational amplifier,
- testing higher DC-offset values.

6.2.8 Digital DC-offset Implementation

Adding the DC-offset digitally involves a two step process of scaling and summing. Firstly, REF C and REF D are both set to 1.35 V. Each sample is then multiplied with a scaling factor of $\frac{1}{1.35} \simeq 0.74$. After this, a constant value of 0.35 is added to each sample. The process is depicted in Figure 6.6.

To implement this in VHDL fixed point arithmetic was used. The full scale value of each 12-bit DAC output is 2^{12}. The fixed point equivalents of the scaling factor and the DC offset constant are $2^{12} \times 0.74 \simeq 3031$ and $\frac{0.35}{1.35} \times 2^{12} \simeq 1062$ respectively. The equivalent hardware pipeline was constructed using the standard Spartan-3 multiplier and adder cores. The sample outputs were first fed into the multiplier and multiplied by the scaling constant. When multiplying two 12-bit fixed point integers the result is a 24-bit value. The 12 least significant bits are truncated and the remaining 12-bit integer is fed to the adder block. The output of the adder is sent directly to the DAC output which is then DC coupled to the COM-4001 input. The firmware implementation of this is depicted in Figure 6.7.

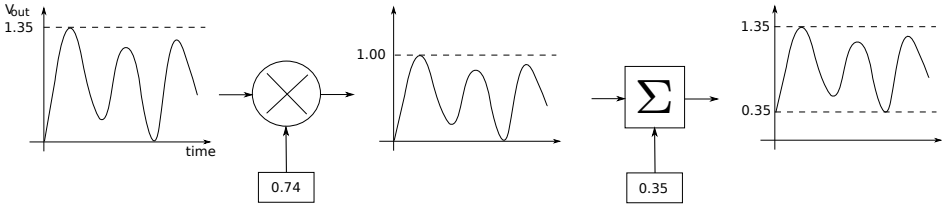

Figure 6.6: Analogue representation of DAC output scaling.

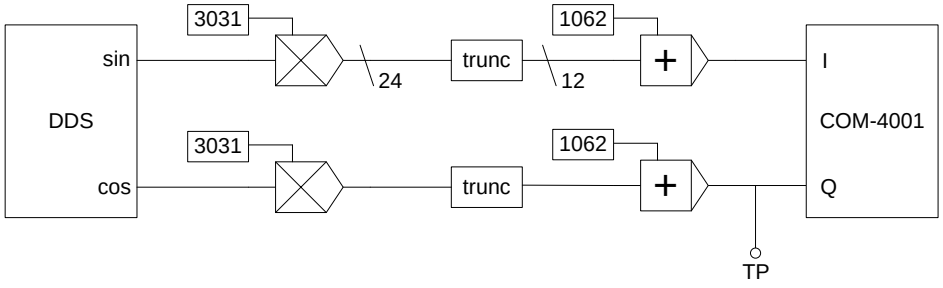

Figure 6.7: Digital DC offset.

6.2.9 Digital DC-offset Test

In this test the digital DC offset firmware componented was connected to the input of the COM-4001 and tested.

Test Setup and Procedure

The digital DC offset firmware is synthesised and loaded on the Spartan-3E. The outputs from the DAC are connected to the inputs of the COM-4001. The FPGA subsystem is powered up. Then the COM-4001 system is powered up. The sine and cosine outputs from the DAC are measured. The RF signal at the output of the COM-4001 is measured using a power meter.

Expected Results

It is expected that the undistorted sinusoidal AC waveforms are seen at the outputs of the DACs after the COM-4001 is powered up. A -9 dBm RF signal is expected at the output of the COM-4001.

Actual Results

When power was supplied to the COM-4001, no signal distortion was noted. A -6.7 dBm signal was measured at the output of the COM-4001.

Analysis of Results

This result verified that the DAC output scaling was implemented correctly and that the correct DAC output voltage was maintained. It also seemed that the I/Q signals were being modulated properly and that the complex baseband signal was being mixed to the correct frequency.

6.2.10 RF Amplification Stage Implementation

As described in the Chapter 4, the Hittite HMC286 and HMC755LP4E were identified as being sufficient components for creating a 1W RF signal. All three components are fully connectorised so implementation involved supplying power to the components and connecting them to each other.

The HMC286 LNA has a single supply pin and a fixed gain of 19 dB. The output 1 dB compression point is at 6 dBm. This means that, if the -9 dBm output of the COM-4001 were connected to the HMC286, the output would be driven into compression. The COM-4001 output gain is therefore adjusted in software to a value of $6 - 19 = -13$ dBm. Another 3 dB of headroom is allowed. Setting the COM-4001 output to -16 dBm is done by setting the gain control parameter to 550 in the control software.

Table 6.1: PA pin voltages.

Pin	Value (V)
VCS	5
VEN1-3	3.5
Vcc1-3	5

Table 6.2: RF power measurements.

Point	Value (dBm)
COM-4001 RF output	−17.94
LNA RF output	1.79
PA RF output	28.93

6.2.11 RF Amplification Stage Test Measurements

The output power of the COM-4001, the LNA and the PA are measured in this test. A basic spectral examination of each component's RF output signal is also performed.

Test Setup and Procedure

The COM-4001 module is supplied with 5 V and connected to the reconstruction filter from the DACs on the Spartan-3E board. The RF output of the COM-4001 is measured. The LNA supply pin is connected to a +3 V supply and connected to the COM-4001. The output of the LNA is then measured. The PA pins are then set to the voltages in Table 6.1, as per the data sheet [25], and the PA's RF input is connected to the LNA RF output. The PA RF output is then measured.

Expected Test Results

A measured gain of 19 dB is expected from the LNA. A measured gain of 31 dB is expected from the PA. Linear frequency response is expected from all the components.

Actual Test Results

The measured output power at each point in the RF chain is presented in Table 6.2.

Note that a 30 dB attenuator was connected to the output of the PA as the Rhode & Schwarz spectrum analyzer used for the measurements has a maximum input power threshold of 20 dBm. The spectrum analyzer measurements are provided in Appendix B. No obvious spectral distortion was noted at any of the measurement points.

Analysis of Results

The gain supplied by the LNA is $1.79 - (-17.94) = 19.73\,\text{dB}$ so it is operating with slightly more gain than expected. The gain of the PA is $28.93 - 1.79 = 27.14\,\text{dB}$ which is less than the maximum specified gain of 31 dB. This discrepancy could be explained by excessively high operating temperatures. During testing it was noted that the evaluation board was very hot to the touch. According to the product data sheet, a maximum gain of 28 dB is achievable at high temperatures.

Assuming a maximum gain of 27 dBm (measured), an antenna gain of $30 - 27 = 3\,\text{dB}$ is required to achieve the required equivalent isotropically radiated power of 1 W specified by the link budget.

Chapter 7

Integration and Testing

In this chapter the various bottom level components of the system are connected and tested to form stable subcomponents. These subcomponents are then connected to each other in an iterative manner. After each step, larger, more complex subcomponents are formed and tested. Ground station and payload loopback tests are documented and presented here.

7.1 Payload Loopback Tests

The two SRAM interfaces are first stress tested to ensure the physical interface is reliable. Thereafter, and continuing from the OBC SRAM with FIFO loopback test presented in Chapter 5, components get added to the system being tested until all the payload components form a single unit which can be tested.

7.1.1 Resource Manager Integration Stress Test

In this test 1.5 GB of data was transferred over the expansion port to verify 100% reliability of the physical layer interface between the SH4 OBC and the ML501 FPGA board. Key components under testing were:

- the resource manager,
- the physical layer connection between the OBC and the ML501 board and
- the firmware on the FPGA.

Test Setup and Procedure

Figure 7.1 is a simplified representation of the firmware that was synthesised onto the ML501 board FPGA for this test. The interrupt driven resource manager was started on the SH4. A test program was written to test the

behaviour of the resource manager. The test program opens the expansion port file using *open()*, spawns a reader thread and a writer thread and then exits. The writer thread transmits a block of 21 bytes using *write()* and checks the return code:

```
wd = write(fd, sendblock, sizeof(sendblock));
```

If a retry error was returned, the program delays for 10 milliseconds and retries on the next iteration. If no error was returned, the data values in the block are incremented. The reader thread reads data from the device as fast as possible. It performs blocking reads:

```
rd = read(fd, &rcvblock, sizeof(rcvblock));
```

Once a block of data is read, the values in the block are checked to see if any errors occurred. The reader thread must have known values from the writer thread with which to compare the read values. Instead of implementing a complex sliding window scheme to synchronize the transmitted values with the reader thread, the writer thread increments each value in the data block after each write operation. The reader thread can then use the following conditions to test data it has read:

$$d_{k+20} - d_k = 1 \tag{7.1.1}$$

and

$$d_{k+1} - d_k = 1 \tag{7.1.2}$$

where d_k represents a single 32-bit data word containing an integer value. This process is then allowed to run for 2 hours.

Expected Results

If data is being transmitted successfully, no errors will be reported by the reader thread of the test program when checking the relations in Equations 7.1.1 and 7.1.2.

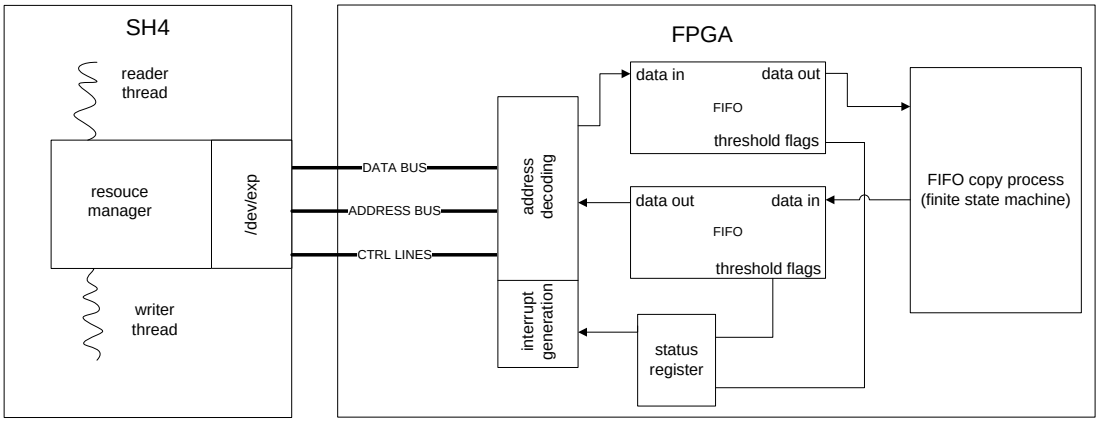

Figure 7.1: Resource manager with dual FIFO loopback test.

Actual Results

After running for two hours, a total of 1.5 GB of data was transmitted over the interface and no errors were detected.

Analysis of Results

The effective throughput of the interface was 10 000 read/write cycles every 8 seconds, where a read/write cycle consisted of transmitting a 21-element block of 32-bit words i.e. the data rate was $\frac{10\,000}{7} \times 21 \times 32 \times 2 \times \frac{1}{1024 \times 8} \simeq 234\,\text{KB/s}$. The interface was tested extensively and no errors were found.

7.1.2 DSP Expansion Port Integration Stress Test

The testing procedure for the DSP SRAM interface stress test was very similar to the OBC SRAM interface stress test: data was written to a FIFO on the FPGA from the OBC. The data is then sent to the modem for modulation. An internal loopback in the modem copies the modulated samples to the demodulator. The demodulated data is then sent back to the FPGA where the reader thread on the SH4 reads the values. The test setup is depicted in Figure 7.9. The read values are compared to the transmitted values and an error is flagged if there is a discrepancy. When an error is flagged, the code halts and a red LED on the DSP is illuminated indicating error. The stress test did not result in any invalid data values and it was concluded that data could be reliably transferred between the DSP and the SH4 via the FPGA.

7.2 Ground Station Integration Tests

Firstly, all the components that implement the downlink are integrated and tested. Secondly, a loopback test including the quadrature upmixer is performed and a problem with the SDR modem is identified. Finally, the LNA and PA are integrated and tested.

7.2.1 Data Radio Integration

On the ground station the data radio was connected directly to the Fit-PC. On the payload however, all the RS-232 ports on the SH4 were in use so the data radio was connected to an available RS-232 port on the ML501 FPGA board. An integration test was required to verify that the resource manager and FPGA firmware interfaced with the radio properly.

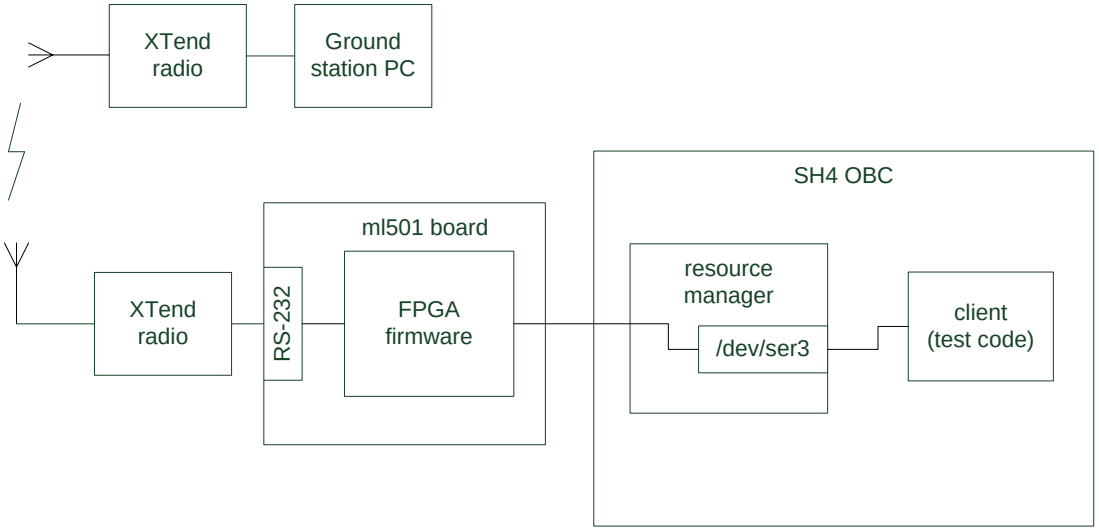

Figure 7.2: Data radio integration test.

Test Setup and Procedure

The resource manager on the SH4, /usr/sbin/devc-exp, was started. The data radio was connected to the RS-232 port of the ML501 board and powered up. The ML501 board was powered up and loaded with the payload firmware. A block of data was transmitted over the downlink via the resource manager from within a C program on the SH4:

```
fd = open ("/dev/ser3", O_RDWR));
rd = write(fd, sendblock, sizeof(sendblock));
```

Note that /dev/ser3 is not a physical serial port, but a logical one exposed by the resource manager. On the ground station PC, the serial port was constantly monitored for incoming data:

```
cat /dev/ttyUSB0
```

The test setup is depicted in Figure 7.2.

Expected Results

The data block transmitted from the SH4 should be received on the ground station PC.

Actual Results

The data was successfully received on the ground station PC.

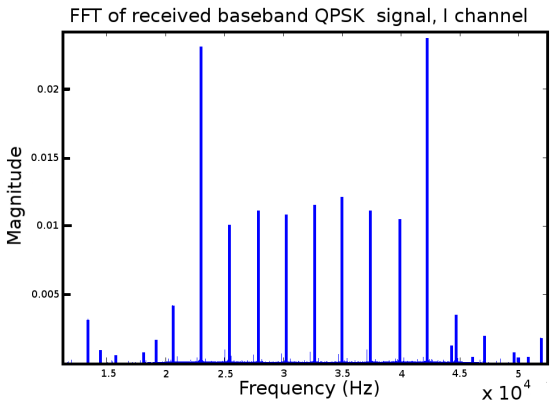

Figure 7.3: Fast Fourier Transform of the COM-3001 test point 1 signal.

Analysis of Results

POSIX compliant interfaces are created for the downlink allowing higher layer programs to transmit data seamlessly to the ground.

7.3 Quadrature Mixing Integration Test

The integration testing of the quadrature mixer modules is now described.

Test Setup and Procedure

In this test the filtered analogue baseband I/Q message signals from the modem are sent to the COM-4001 quadrature upmixer module. The RF output of the COM-4001 module is attenuated and fed to the COM-3001 quadrature downmixer receiver. The COM-3001 has two 10-bit ADC outputs and two analogue test points (I and Q). The test points expose the ADC inputs and contain the baseband I/Q message signals. One of the test points was sampled using a data capture unit and software written by JJ Cronjé [13]. The fast Fourier transofrm of a modulated message signal was also generated using Matlab.

Expected Results

Upon inspection, the frequency "footprint" of the received and transmitted signals should be identical.

Actual Results

The spectrum of the test point 1 signal on the quadrature downmixer is shown in Figure 7.3. The spectrum of the modulated signal which was generated in Matlab is shown in Figure 7.4.

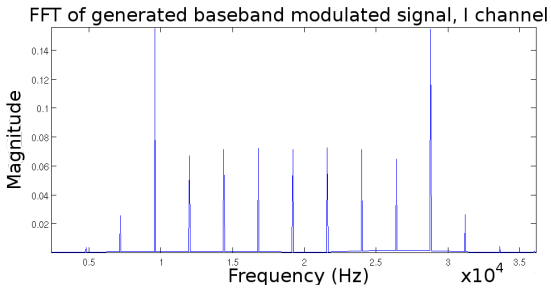

Figure 7.4: Fast Fourier Transform of a baseband modulated signal generated in Matlab.

Analysis Results

When the two spectrums are compared it is evident that, except for some attenuation and carrier leakage, the transmitted signal was received successfully. The attenuation can be adjusted by the gain control voltage pin on the COM-3001 module. The carrier leakage is discussed in the following test. This test verified that both the COM-3001 and the COM-4001 are operating according to specification.

7.4 Ground Station Loopback Test

In this test the aim is to integrate the ground station components with the RF components.

Test Setup and Procedure

The components are connected as depicted in Figure 7.5. The necessary components for this test are:

- a Fit-PC acting as ground station PC,
- a Spartan-3E FPGA board acting as ground station FPGA,
- the DSP modem,
- the reconstruction filter for the DAC outputs,
- the COM-4001 and COM-3001 RF modules and
- a second Spartan-3E FPGA board acting as a loopback "KUL" FPGA.

The PC transmits imitation TM frames to the ground station Spartan-3E FPGA via an on-board UART. The frames represent the raw user data which need to be modulated and are sent to the modem. The modulated samples from the modem are sent to the DAC via the FPGA. The output

of the DAC is connected to the reconstruction filter. The filtered baseband I/Q signals are then connected to the quadrature upmixer. An attenuator is connected to weaken the signal before connecting it to the COM-3001 input. The demodulated I/Q signals are then captured on another Spartan-3E FPGA board. They are sent to the DAC as a test point. The samples are also passed to the ground station Spartan-3E board via the on-board Ethernet. These samples are then passed to the modem for demodulation. The demodulated data is then transmitted to the PC where it is compared to the transmitted data.

Expected Results

It is expected that the transmitted data is looped back successfully i.e. the received data should match the transmitted data.

Actual Results

The modem was unable to demodulate the incoming data successfully. Samples were being sent to the modem for demodulation so all other components in the chain were operational.

Analysis of Results

The FFT of a single channel at the output of the reconstruction filter was taken and is presented in Figure 7.6. Figure 7.7 depicts the received signal as seen at the output of the COM-3001 module. It is evident that there is a frequency component – near 12.5 kHz – in the received signal which is not present in the

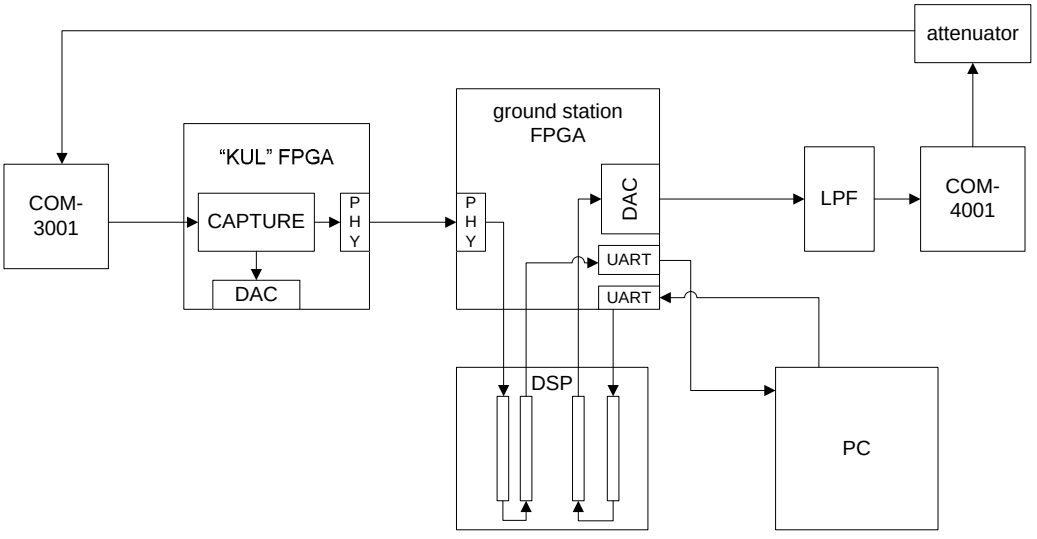

Figure 7.5: Ground station RF integration.

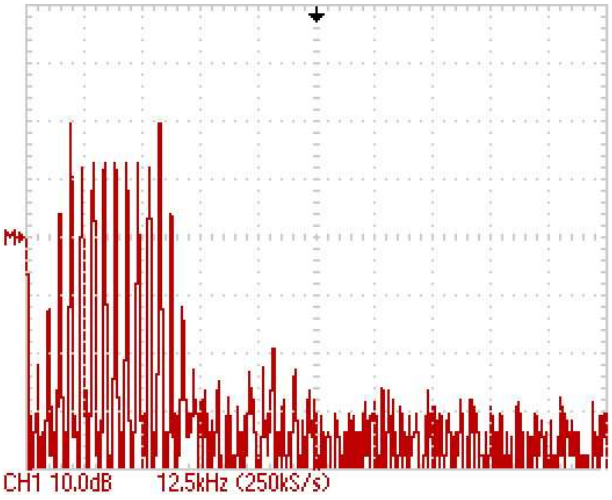

Figure 7.6: FFT of a modulated uplink message signal.

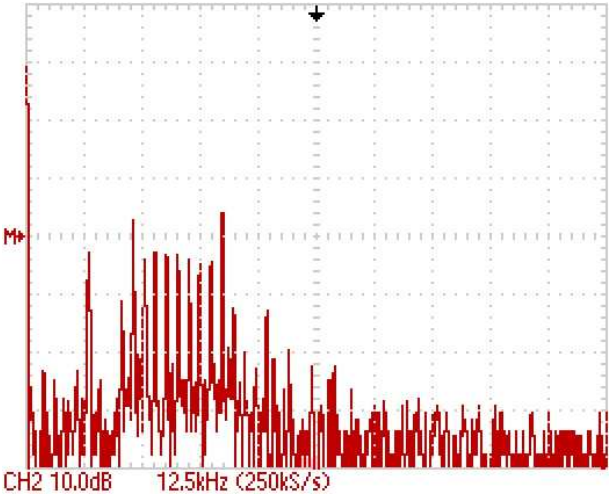

Figure 7.7: FFT of a received baseband signal.

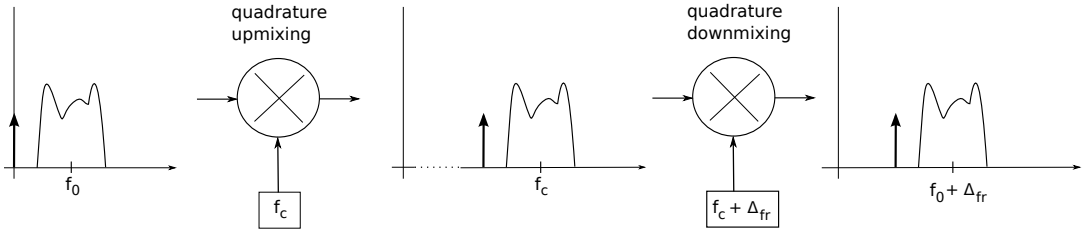

Figure 7.8: Frequency domain representation of carrier leakage.

transmitted signal. The modem is locking onto this component and not the message signal and is therefore unable to demodulate. The presence of the frequency component is now explained. As described in Chapter 5, the COM-4001 quadrature upmixer expects a DC offset of 0.85 V at its input. This DC component translates to a component at 0 Hz in the frequency domain. When the baseband signal is then mixed up to the carrier frequency of 2.45 GHz, the DC component also gets mixed up.

Upon reception of the RF signal, the COM-3001 mixes the signal down to baseband. However, because the COM-4001 and COM-3001 modules have their own independent oscillators, there is a discrepancy between the upmixing and downmixing frequencies. This is easily understood after taking a closer look at the the 40 MHz oscillators on the COMBLOCK modules [11]. Due to frequency tolerance of the oscillators and the Phase Locked Loop (PLL) multiplication factor, a difference will be noted in the two local oscillators. The maximum frequency difference, Δ_{fr}, between the two local oscillators can be calculated with the following formula:

$$\Delta_{fr} = f_0 \times mf(\lceil \Delta_f \rceil - \lfloor \Delta_f \rfloor) \tag{7.4.1}$$

The multiplication factor, mf, is $2.45 \times 10^9 / 40 \times 10^6 = 61.25$. Furthermore, with a maximum frequency tolerance, Δ_f, of ± 50 ppm [10], and a nominal frequency, f_0, of 40×10^6 Hz, this could result in a difference of $40 \times 10^6 \times 61.25(100/10^6) = 245\,000$ Hz. Luckily both units are operating at the same temperature, so the difference will not be so extreme.

The mixing process and the resulting unwanted frequency component is depicted in Figure 7.8.

At the time of writing it was not known what the solution to this problem is, but it could be useful to mix up to a higher intermediate frequency to ensure that the message signal is not near the DC component. The DC component could then be filtered out.

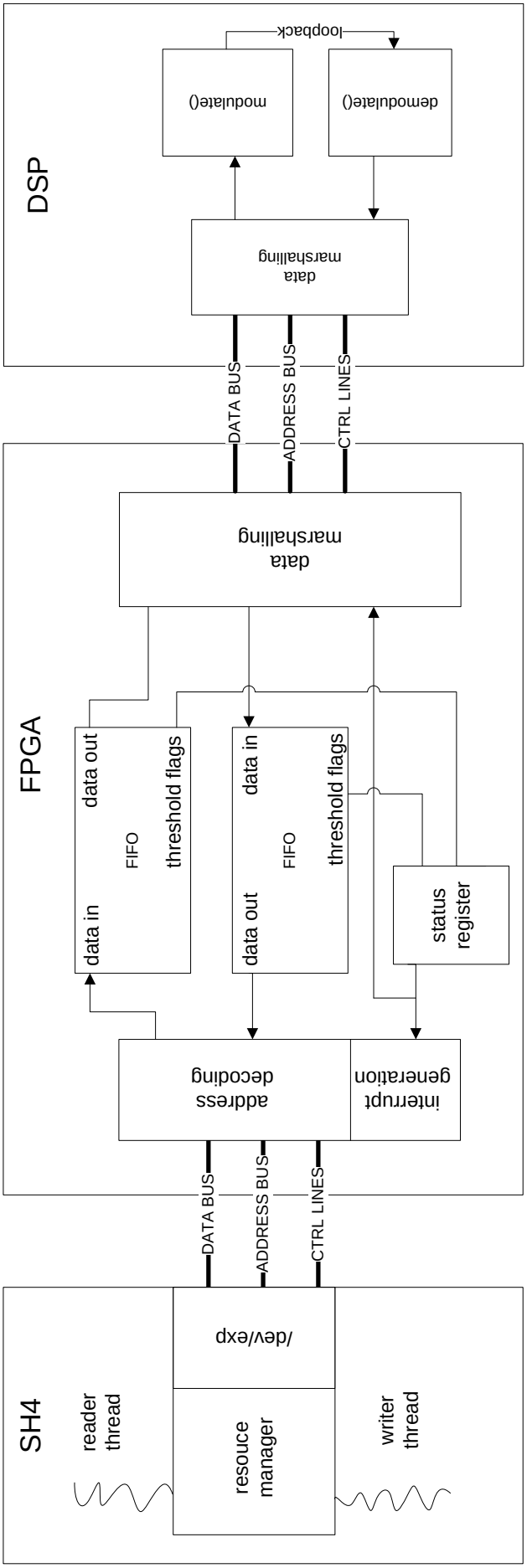

Figure 7.9: SH4 OBC, FPGA and DSP loopback test.

Chapter 8

Conclusion

This thesis has presented the a reusable signal processing architecture which is implemented in the form of a satellite communications payload for a LEO satellite.

8.1 Summary of Completed Work

This thesis was presented in three sections. The work began with an analysis of the proposed solution which led to a detailed design. The detailed design then required implementation. Thirdly, the implementation was tested.

The first section showed that, from a design perspective, a novel airborne signal processing platform was feasible. The platform was shown to be able to test a steerable antenna array and implement an extraterrestrial communications link. A prototype remote sensing ground station was presented as an example use case of the communications link.

Section two showed that implementation of both the payload and the ground station was feasible. A functional airborne signal processing architecture was constructed, tested and documented. A robust, space-qualified onboard computer was combined with a signal processor and a high-end FPGA to demonstrate that the implementation of a satellite-based communications payload using software defined radio and forward error correction technology is feasible.

Section three used rigorous testing to verify the reliability of the payload architecture. Integration of other research projects such as the software defined radio modem was shown to be feasible, even if the projects were not fully functional at the time of writing.

A variety of buses and interfaces such as CAN, SRAM and Ethernet, were made available for future projects. A reliable and extendable resource manager for the SH4 OBC was developed which allows software on the SH4 to interface with a wide range of hardware. Stress testing was used to transfer large amounts of data throughout all the components in the system and this

showed that the physical layer was error-free.

8.2 Recommendations

The signal processing architecture was designed to be reconfigurable. Full system integration with the various hardware components was demonstrated. Solving difficulties with the SDR modem and implementing FEC modules will complete the entire communications system. Certain improvements, optimisations and tasks were identified during the course of the thesis:

- it should be possible to switch the payload into a test mode wherein all non-critical system functionality is disabled and a test signal is transmitted to the ground to assist in debugging should the payload not operate as expected,
- the downlink data radio should be replaced with the SDR modem, a DAC, a reconstruction filter, a quadrature upmixer and a power amplifier,
- once the SDR modem integration is successful, a dedicated SH4 daughter board consisting of the Virtex-5 FPGA and the Freescale DSP should be designed and built,
- more extensive testing with more than one ground station should be done to test the scheduling software implemented by J. Gilmore,
- the EDAC modules implemented by R. Wiid should be integrated into the system once the SDR modem is operational,
- once the components have been finalized, space qualification of the components should be investigated,
- the integration of the SAA should be tested in a controlled environment before the first flight test and
- the cost of the ground station transmitter could be reduced by designing a quadrature upmixer PCB to replace the COM-4001 module.

8.3 Final Comments

Standard interfacing techniques, protocols and off-the-shelf components were used wherever possible to save development time and reduce project costs. It was shown that the system worked correctly through the use of thorough testing. It was shown that the SDR modem needs further development. The

design goal of creating a reusable signal processing architecture for satellite-based communications systems was achieved. The South African space industry can use the platform for further research into digital signal processing in space based communications systems.

Appendix A

Timing Diagrams

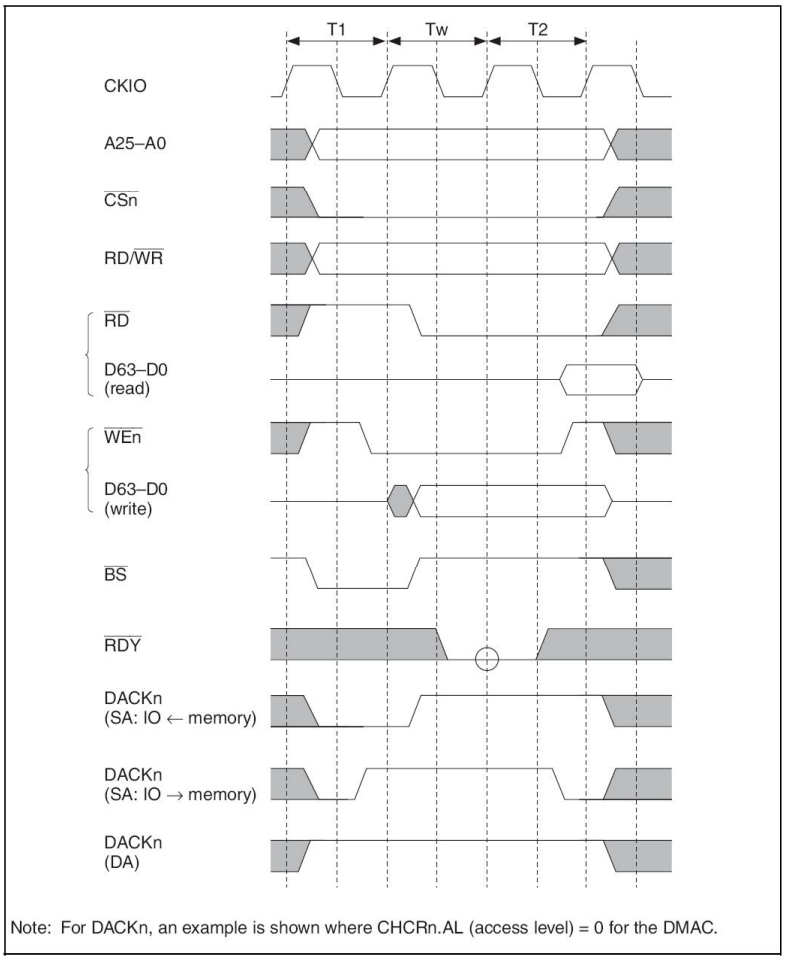

Figure A.1: SRAM interface wait timing (software wait only)

Appendix B

Ground Station RF Power Measurements

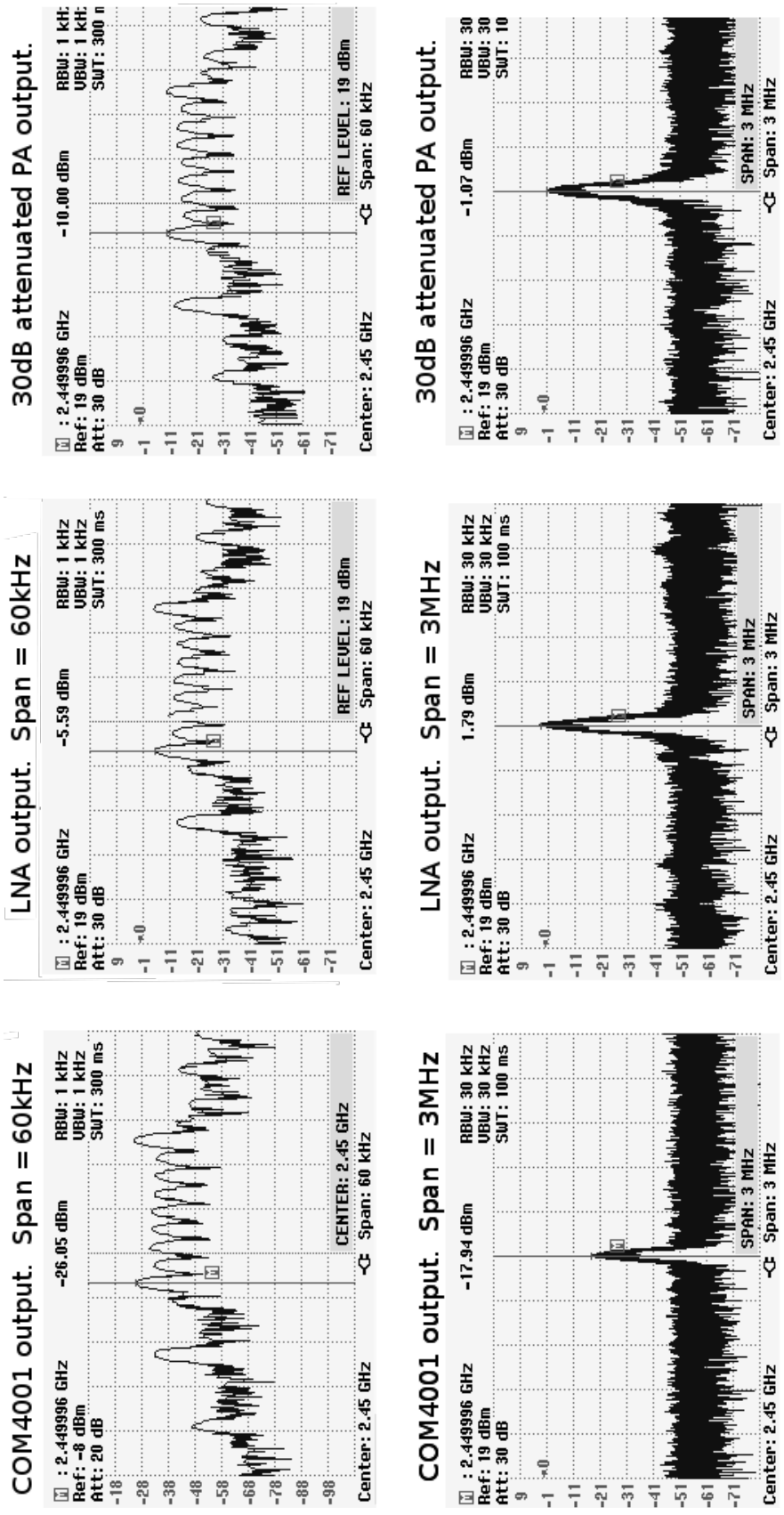

Figure B.1: Ground station RF measurements.

Bibliography

[1] "Aerocomm fixed mount antenna - mmcx." http://www.rfdesign.co.za/pages/5645456/Products/868-MHz-Wireless-Products/Antennas.asp.

[2] "Fit-pc." http://www.fit-pc.com/.

[3] "A simplified vhdl uart." http://esd.cs.ucr.edu/labs/uart/uart.html.

[4] "Uart 16550 core." http://opencores.org/project,uart16550.

[5] ACTEL, CA. *Libero IDE Online Help*.

[6] AMSAT, "SUNSAT-OSCAR 35." http://www.amsat.org/amsat-new/satellites/satInfo.php?satID=53.

[7] BOTHA, K., VAN DER WESTHUIZEN, E., and VAN ROOYEN, G.-J., "A Digital Modem Card for a Multi-channel Satellite Communications Payload." *presented at SATNAC 2008*.

[8] CHANG, K., *Encyclopedia of RF and Microwave Engineering*. New Jersey: John Wiley and Sons, Inc., 2005.

[9] CHAPMAN, K., "200 MHz UART with internal 16-byte buffer." http://www.xilinx.com/support/documentation/application_notes/xapp223.pdf.

[10] CITIZEN. *AT Cut Crystal Unit*.

[11] COMBLOCK, Maryland. *COM-4001-C/D Data Sheet*.

[12] COOKE, A., "Rural Email System for the Sumbandila Satellite." Master's thesis, University of Stellenbosch, 2007.

[13] CRONJÉ, J., "Software Architecture Design of a Software Defined Radio System." Master's thesis, University of Stellenbosch, 2004.

[14] DIGI, MN. *XTend OEM RF Module Product Data Sheet*.

[15] DREYER, G., *Generic Specification: CAN Node*. SunSpace and Information Systems, Stellenbosch.

[16] DREYER, G., *Specification for the On Board Computer SH4-6U-01*. SunSpace and Information Systems, Stellenbosch.

[17] DREYER, G., *User manual for the SH4 OBC*. SunSpace and Information Systems, Stellenbosch.

[18] EUROPEAN COOPERATION FOR SPACE STANDARDIZATION, Noordwijk. *Space data links - Telecommand protocols, synchronization and channel coding*.

[19] FAIRCHILD SEMICONDUCTOR, ME. *74ALVC162245 Data Sheet*.

[20] FREESCALE SEMICONDUCTOR, TX. *DSP56321 Technical Data*.

[21] GILMORE, J., "Detailed design: Satellite transport protocol." IS-HSII Project Documentation, 2010.

[22] GILMORE, J., "Development of a Satellite Communications Software System and Scheduling Strategy." Master's thesis, University of Stellenbosch, 2010.

[23] HITTITE, MA. *HMC286 / 286E Data Sheet*.

[24] HITTITE, MA. *HMC495LP3 / 495LP3E Data Sheet*.

[25] HITTITE, MA. *HMC755LP4E Data Sheet*.

[26] HOROWITZ, P. and HILL, W., *The art of electronics*. Second edition. Cambridge: Cambridge University Press, 1996.

[27] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. *ISO/IEC standard 7498-1:1994*.

[28] JOHNSON, H. W. and GRAHAM, M., *High-speed digital Design: A Handbook of Black Magic*, pp. 29–36. New Jersey: Prentice Hall, 1993.

[29] KESTER, W., *Evaluating High Speed DAC Performance*. Analog Devices, MA.

[30] KRUGER, I., "Ku leuven is-hsii project linkbudget." IS-HSII Project Documentation, 2008.

[31] KRUGER, I., "Detailed design: Aircraft satellite emulator." IS-HSII Project Documentation, 2010.

[32] LAWYER, D. S., "Linux serial howto." http://tldp.org/HOWTO/Serial-HOWTO.html.

[33] LINEAR TECHNOLOGY, CA. *LTC2624 Data Sheet*.

[34] NATIONAL SEMICONDUCTOR, CA. *DP83865 Gig PHYTER Data Sheet*.

[35] NATIONAL SEMICONDUCTOR, CA. *LF351 Wide Bandwidth JFET Input Operational Amplifier*.

[36] OLIVIER, F., "An LDPC Error Control Strategy for Low Earth Orbit Satellite Communication Link Applications." Master's thesis, University of Stellenbosch, 2009.

[37] PROAKIS, J. G., *Wiley Encyclopedia of Telecommunications*. New Jersey: John Wiley and Sons, Inc., 2003.

[38] PROLIFIC, Taiwan. *PL-2303 Product Data Sheet*.

[39] RENESAS. *SH7750 Series Hardware Manual*.

[40] RF DIGITAL CORPORATION, CA. *RFDP8 RF Module*.

[41] SUNSPACE, "New software for ADCS processor." http://sumbandilamission.blogspot.com/2009/09/new-software-for-adcs-processor.html.

[42] SUNSPACE AND INFORMATION SYSTEMS, Stellenbosch. *Process Monitor User Documentation*.

[43] TEXAS INSTRUMENTS, TX. *Analysis of the Sallen-Key Architecture: Application Report*, 2002.

[44] VAN DER WESTHUIZEN, E., "Detailed design of a software defined radio modem." IS-HSII Project Documentation, 2009.

[45] VAN WYK, J. F., "Reusable software defined radio platform for micro-satellites." Master's thesis, University of Stellenbosch, 2008.

[46] WIID, R., VAN DER WESTHUIZEN, E., and OLIVIER, F., "Detailed design: Tm protocol." IS-HSII Project Documentation, 2010.

[47] WOLHUTER, R., VAN ROOYEN, G.-J., and OLIVIER, F., "IS-HSII project proposal for ground station development and systems integration with aircraft based satellite emulator." IS-HSII Project Documentation, 2008.

[48] XILINX, CA. *LogiCORE IP FIFO Generator v6.1*.

[49] XILINX, CA. *ML501 Evaluation Platform*.

[50] XILINX, CA. *PicoBlaze 8-bit Embedded Microcontroller User Guide*.

[51] XILINX, CA. *Spartan-3E Starter Kit Board User Guide*.