SOFTWARE DEFINED RADIO (SDR) ARCHITECTURE FOR CONCURRENT MULTI-SATELLITE COMMUNICATIONS

Mamatha R. Maheshwarappa

Submitted for the Degree of
Doctor of Philosophy
from the
University of Surrey

Surrey Space Centre
Faculty of Electronics and Physical Sciences
University of Surrey
Guildford, Surrey GU2 7XH, UK

August 2016
Software Defined Radio (SDR) Architecture for Concurrent Multi-Satellite Communications
Mamatha R. Maheshwarappa

© Mamatha R. Maheshwarappa 2016
ACKNOWLEDGEMENTS

This thesis represents a milestone in more than 3 years of work at UniS and especially within the Surrey Space Centre (SSC). Since my first day during MSc on September 26th, 2011 I have felt at home at UniS. I have been given unique opportunities including - working at SSTL for STRaND-1, post launch activities for De-OrbitSail, AMSAT activities such as Bring Your Own Board (BYOB) day and presenting/manning SSC stand at different conferences. Throughout these years, I have learned about many research areas/subsystems of a satellite though student presentations and working on different projects. This is the result of advice, experiences and motivation by dozens of remarkable individuals who I wish to acknowledge.

First and foremost, I wish to thank my supervisor, Dr Christopher Bridges. He has been supportive since the days I began working on MSc thesis. Ever since, Chris has supported me not only by providing a research assistantship over almost three years, but also academically and emotionally through the rough road to finish this thesis. He helped me come up with the thesis topic, win three different scholarships and guided me throughout. And during the most difficult times when writing this thesis, he gave me the moral support and the freedom I needed to move on. His words can always inspire me and keep me motivated. Without his kind and patient instruction, it is impossible for me to finish this thesis. Thank you!

I must offer my profound gratitude to Professor Sir Martin Sweeting, co-supervisor and a mentor for his insightful comments and encouragement. My sincere thanks goes to Dr Mark Bowyer, Senior Expert at Airbus Defence and Space Ltd. Dr Bowyer has offered me unreserved help, valuable and constructive suggestions during my research. His willingness to give his time so generously has been very much appreciated. I would like to offer special thanks to Dr Brian Yeomans, who has been a great help in my initial days of my research.

I am also indebted to Karen Ford who has been a source of love and energy always. A special acknowledgement goes to my office mates Eli and Marcello who have been supportive in every way. Furthermore, I have to offer my special thanks to OBDH group – Becky, Alex and Yaser who came up with critical questions and constructive advice to improve my research during our group meetings. There are numerous students and academics that have given me technical and personal support during my studies: Prof Aglietti, Prof Palmer, Dr Saaj, Chiara, Richard, Martin, Ed, Andrea, Luke, Jim, Ale, Giuseppe and Asma. Without all these exceptional friends and academics my understanding of satellite systems, in all forms, would never have given me broad knowledge and the unending passion for innovation through research I now
Software Defined Radio (SDR) Architecture for Concurrent Multi-Satellite Communications
Mamatha R. Maheshwarappa

have. I would also like to thank my dearest friends outside SSC – Arunthathi, Ravi, Armin and Gowhar who have encouraged me throughout my PhD both in good and bad times. Also, I would like to acknowledge my current colleagues at QinetiQ Matthew Cosby, Paul Thompson and Tom Morris who have been very supportive and understanding.

I would like to sincerely thank all my sponsors – SSC, Overseas Research Scholarship (ORS) and SSTL.

I finish with India, where the most basic source of my life energy resides: my family. I have an amazing family, unique in many ways, and the stereotype of a perfect family in many others. Their support has been unconditional all these years; they have given up many things for me to be at the UK; they have cherished with me every great moment and supported me whenever I needed it. My final and special thanks go to my husband, Arpan, who makes my life complete with his unconditional love and strength....
SDRs have emerged as a viable approach for space communications over the last decade by delivering low-cost hardware and flexible software solutions. The flexibility introduced by the SDR concept not only allows the realisation of concurrent multiple standards on one platform, but also promises to ease the implementation of one communication standard on differing SDR platforms by signal porting. This technology would facilitate implementing reconfigurable nodes for parallel satellite reception in Mobile/Deployable Ground Segments and Distributed Satellite Systems (DSS) for amateur radio/university satellite operations.

This work outlines the recent advances in embedded technologies that can enable new communication architectures for concurrent multi-satellite or satellite-to-ground missions where multi-link challenges are associated. This research proposes a novel concept to run advanced parallelised SDR back-end technologies in a Commercial-Off-The-Shelf (COTS) embedded system that can support multi-signal processing for multi-satellite scenarios simultaneously. The initial SDR implementation could support only one receiver chain due to system saturation. However, the design was optimised to facilitate multiple signals within the limited resources available on an embedded system at any given time. This was achieved by providing a VHDL solution to the existing Python and C/C++ programming languages along with parallelisation so as to accelerate performance whilst maintaining the flexibility. The improvement in the performance was validated at every stage through profiling.

Various cases of concurrent multiple signals with different standards such as frequency (with Doppler effect) and symbol rates were simulated in order to validate the novel architecture proposed in this research. Also, the architecture allows the system to be reconfigurable by providing the opportunity to change the communication standards in soft real-time. The chosen COTS solution provides a generic software methodology for both ground and space applications that will remain unaltered despite new evolutions in hardware, and supports concurrent multi-standard, multi-channel and multi-rate telemetry signals.

Key words: Software defined radio, front/back end challenges, GNURadio, Commercial-Off-The-Shelf (COTS), Field Programmable Gate Array (FPGA)
Software Defined Radio (SDR) Architecture for Concurrent Multi-Satellite Communications
Mamatha R. Maheshwarappa

3.2 DESIGN APPROACH ............................................................................................................. 48

3.2.1 Band Pass Filter (BPF) .................................................................................................. 49
3.2.2 Low Noise Amplifier (LNA) ......................................................................................... 51
3.2.3 Duplexer .......................................................................................................................... 51
3.2.4 I and Q signal ................................................................................................................... 51
3.2.5 Local Oscillator (LO) ..................................................................................................... 51
3.2.6 Low Pass Filter (LPF) ..................................................................................................... 52
3.2.7 Variable Gain Amplifier (VGA) ...................................................................................... 52
3.2.8 Analogue-to-Digital (ADC) & Digital to Analogue-to-Converters (DAC) .................... 52
3.2.9 Baseband Processing ....................................................................................................... 52

3.3 SDR HARDWARE PLATFORM OPTIONS ....................................................................... 53

3.3.1 Zedboard and MyriadRF ............................................................................................... 54
3.3.2 Zedboard and AD-FMCOMMS3-EBZ .......................................................................... 58

3.4 ARCHITECTURE IMPLEMENTATION AND VALIDATION ............................................... 61

3.4.1 Frame Structure ............................................................................................................. 61
3.4.2 Modulation ...................................................................................................................... 62
3.4.3 Forward Error Correction (FEC) ................................................................................... 63
3.4.4 Encoder Steps and Requirement Analysis ....................................................................... 65
3.4.5 Data Rate and Symbol Rate Calculations: ................................................................. 67
3.4.6 Memory Calculations ..................................................................................................... 68
3.4.7 Real-time Satellite Signals Received on a Dongle Connected to a Regular PC/Laptop ................................................................................................................................. 69
3.4.8 European Student Earth Orbiter (ESEO) Transmitter Signals Received on Dongle and Regular PC ........................................................................................................... 74
3.4.9 Transceiver on Embedded Systems (AD-FMCOMMS3-EBZ and Zedboard) ............ 75

3.5 SUMMARY .......................................................................................................................... 82

4 PERFORMANCE ANALYSIS ............................................................................................... 83

4.1 SOFTWARE PROFILING .................................................................................................... 83
Software Defined Radio (SDR) Architecture for Concurrent Multi-Satellite Communications
Mamatha R. Maheshwarappa

6.2.1 Parallel Architecture – Alternative 1 ................................................................. 133
6.2.2 Parallel Architecture - Alternative 2 Refinement .............................................. 135
6.2.3 Parallel Architecture - Alternative 3 Refinement .............................................. 136
6.2.4 Parallel Architecture - Alternative 4 Refinement .............................................. 137
6.3 Implementation and Validation of Parallel Architecture – Alternative 1 ............. 139
6.3.1 Emulation of Two Signals .................................................................................. 140
6.4 General Results - Discussion ............................................................................... 145
6.4.1 Memory Requirement ....................................................................................... 146
6.4.2 Throughput ....................................................................................................... 148
6.4.3 FPGA Sizing Requirements ............................................................................. 149
6.5 Auto Configuring .................................................................................................. 151
6.5.1 Case 1: Signal Unknown ................................................................................... 151
6.5.2 Case 2: Signal of Interest .................................................................................. 152
6.6 Summary ............................................................................................................... 154

7 Conclusions ............................................................................................................. 155
7.1 Contribution to the State-of-the-Art ..................................................................... 158
7.2 Future Work ......................................................................................................... 158
7.2.1 Concurrent Multiple Transmit Channels ......................................................... 159
7.2.2 Parallel Architectures ...................................................................................... 159
7.2.3 Auto-Configuring ............................................................................................. 159
7.2.4 Test Plan .......................................................................................................... 159

References .................................................................................................................. 161
Appendix 1 – Front End Options ................................................................................. 172
Appendix 2 – Space SDRs .......................................................................................... 176
Appendix 3 – SDRs for Small Satellite Applications ...................................................... 178
Appendix 4 – Terrestrial SDRs .................................................................................... 180
Appendix 5 – Smart Fusion2 and Funcube Dongle ....................................................... 183
Appendix 6 – Implementation Steps and Challenges .................................................. 185
LIST OF FIGURES

FIGURE 1-1 SPACEWORKS’ PROJECTION OF SMALL SATELLITES TILL 2020 [3] ........................................... 2
FIGURE 1-2 CATEGORIZATION OF THIRTY-NINE MULTI-SATELLITE MISSIONS [6] ................................. 2
FIGURE 1-4 SCREENSHOT OF GPREDICT SHOWING THE NUMBER OF SMALL SATELLITES IN SPACE ............. 3
FIGURE 1-5 RADAR VIEW OF THE ANTENNA SHOWING DIFFERENT SATELLITES IN VISIBILITY (LEFT)
   VHF/UHF ANTENNA SETUP AT UNI S (RIGHT) .................................................................................. 4
FIGURE 1-7 EVOLUTION OF SOFTWARE DEFINED RADIOS .................................................................. 6
FIGURE 1-8 FLOWCHART SHOWING THE OUTLINE OF THIS THESIS .................................................... 11
FIGURE 2-5 EDSN MISSION CONCEPT [55] ...................................................................................... 13
FIGURE 2-6 SOLARA/SARA SATELLITE CONCEPT [65] ................................................................. 13
FIGURE 2-7 XSOLANTRA SIDE VIEW [67] ...................................................................................... 14
FIGURE 2-8 ONE WEB CONSTELLATION [68] ...................................................................................... 14
FIGURE 2-10 CUBESATS LAUNCHED EACH YEAR (2000 - PRESENT) .................................................. 16
FIGURE 2-11 TREND IN CUBESAT FREQUENCY BANDS1 .................................................................. 17
FIGURE 2-12 COMMONLY USED MODULATION TECHNIQUES1 ...................................................... 17
FIGURE 2-13 COMMONLY USED DATA RATES ............................................................................... 18
FIGURE 2-14 CAMSAT RECEPTION REPORT ON SEP 20 2015 8:10 UTC, JA1OGZ, AKIRA ............. 19
FIGURE 2-16 3D CONE OF SSC GROUND STATION ANTENNA ..................................................... 20
FIGURE 2-18 FOUR ACTIVE LEO SMALL SATELLITE CONSTELLATIONS ........................................... 22
FIGURE 2-19 FIVE SATELLITES (YOAGAN: 20B_40110, 20A_40109, 20C_40111,
   FLOCK:1C_12_40739, 2B_4_40953) FROM TWO DIFFERENT CONSTELLATIONS OVER SSC
   GROUND STATION .............................................................................................................. 22
FIGURE 2-20 HIGH-LEVEL ABSTRACTION OF THE SDR FORUM TIER DEFINITION ..................... 25
FIGURE 2-22 SDR ARCHITECTURE WITH BASEBAND SoC AND RF SoC .......................................... 27
FIGURE 2-23 THE SEVEN-LAYER OSI REFERENCE MODEL [98] .................................................. 28
FIGURE 2-24 ZERO IF RECEIVER AND TRANSMITTER ARCHITECTURE [106] ............................. 28
Software Defined Radio (SDR) Architecture for Concurrent Multi-Satellite Communications
Mamatha R. Maheshwarappa

Figure 3-20 Transmitted Signal from Setup 2 .................................................. 74
Figure 3-21 Decoded Signal from ESEO Transmitter (Date: 21/09/2014 Time: 13:30:00) .... 75
Figure 3-22 Functional Block Diagram [174] ................................................. 76
Figure 3-23 IIO and LibIIO Framework [187] .................................................. 77
Figure 3-24 Signal Transmitted from Setup 3 ................................................... 78
Figure 3-25 Setup 3 Block Diagram .................................................................. 79
Figure 3-26 FFT Plot of the Samples Received from AD9361 (Date: 21/09/2014 Time: 13:30:00) .............................................................................. 79
Figure 3-27 Data Received on Zedboard (Date: 14/11/2014 Time: 10:12:36) ............... 80
Figure 3-28 Transmission Verified on Rohde & Schwarz VSG ............................ 81
Figure 3-29 Constellation Plot of the Signal Received from Test-Bed 3 ...................... 81
Figure 4-1 Software Optimisation Process ....................................................... 84
Figure 4-2 Encoder Overview .......................................................................... 90
Figure 4-3 Transmitter Pseudo Code .............................................................. 90
Figure 4-4 Signal Received on SDR Sharp at Different Data Rates 1k2, 2k4, 4k8, 9k6 and 19k2 (from left to right) .............................................................. 92
Figure 4-5 Computationally Intensive Transmitter Blocks at 9k6 on Dual Core ARM Cortex A9 .............................................................................. 93
Figure 4-6 Absolute CPU Consumption - Transmitter ........................................ 95
Figure 4-7 Profiling Results on Dual Core ARM Cortex A9 ............................... 96
Figure 4-8 Profiling Results on Octa Core ARM Cortex A15 & A7 ....................... 97
Figure 4-9 Profiling Results on Dual Core Intel x86 ........................................... 97
Figure 4-10 Overview of the Decoder ............................................................... 98
Figure 4-11 Receiver Pseudo Code .................................................................. 99
Figure 4-12 Computationally Intensive Receiver Blocks at 1k2 on Octa Core ARM Cortex A15 & A7 .............................................................. 101
Figure 4-13 Response to gprof on Dual Core ARM Cortex A9 .......................... 102
Figure 4-14 Absolute CPU Consumption - Receiver ........................................ 103
Software Defined Radio (SDR) Architecture for Concurrent Multi-Satellite Communications
Mamatha R. Maheshwarappa

FIGURE 4-15 SUCCESS RATE COMPARISON ON DIFFERENT ARCHITECTURES ........................................ 103
FIGURE 4-16 PROFILING RESULTS ON OCTA CORE ARM CORTEX A15 & A7 ........................................ 104
FIGURE 4-17 PROFILING RESULTS ON DUAL CORE INTEL x86 .......................................................... 105
FIGURE 5-1 REVISED ARCHITECTURE - DDC BLOCK MOVED TO FPGA FABRIC ................................. 109
FIGURE 5-2 OVERVIEW OF DQT FUNCTION IN THIS APPLICATION ................................................. 111
FIGURE 5-3 DOWNSAMPLER PSEUDO CODE ....................................................................................... 111
FIGURE 5-4 ABSTRACT VIVADO BLOCK ............................................................................................... 112
................................................................................................................................................................ 113
FIGURE 5-6 POST - DQT PROFILING RESULTS ON DUAL ARM CORTEX A9 (ABSOLUTE CPU TIME) ... 114
FIGURE 5-7 POST DQT PROFILING RESULTS ON ARM CORTEX A9 (RELATIVE CPU TIME) ............. 115
FIGURE 5-8 SUCCESS RATE COMPARISON - POST DQT IMPLEMENTATION ....................................... 116
FIGURE 5-9 NORMALISED CIC FILTER RESPONSE ............................................................................. 119
FIGURE 5-10 FFT PLOT (A) SOFTWARE DECIMATION (B) HARDWARE DECIMATION ............................ 119
FIGURE 5-11 MODIFIED REFERENCE DESIGN .................................................................................... 122
FIGURE 5-12 RECONFIGURABLE DDC BLOCK IMPLEMENTED IN FPGA ............................................. 122
FIGURE 5-13 PSEUDO CODE OF THE PYTHON SCRIPT ....................................................................... 124
FIGURE 5-14 PSEUDO CODE OF BROADCASTER BLOCK ..................................................................... 126
................................................................................................................................................................ 127
................................................................................................................................................................ 128
FIGURE 6-1 PARALLEL RECEPTION OF MULTIPLE SATELLITES ON SDR SHARP ............................ 134
FIGURE 6-2 PARALLEL ARCHITECTURE - ALTERNATIVE 1 ............................................................... 135
FIGURE 6-3 PARALLEL ARCHITECTURE - ALTERNATIVE 2 ............................................................... 136
FIGURE 6-4 PARALLEL ARCHITECTURE - ALTERNATIVE 3 ............................................................... 136
FIGURE 6-5 PARALLEL ARCHITECTURE - ALTERNATIVE 4 ............................................................... 137
FIGURE 6-6 PSEUDO CODE OF THE PYTHON SCRIPT USED TO CHANGE THE TRANSCIEVER STANDARDS .......................................................... 139
FIGURE 6-7 TWO SETUPS USED TO VALIDATE THE PARALLEL ARCHITECTURE WITH 2 (ZYNQ + AD-FMCOMMS3-EBZ) ........................................................................................................... 140

XIII
Software Defined Radio (SDR) Architecture for Concurrent Multi-Satellite Communications
Mamatha R. Maheshwarappa

Figure 6-8 Two Signals at Different Data Rates & Frequencies (A) 1K2 and 2K4 (B) 1K2 and 19K2 (Date: 20/09/2015, Time: 15:32:00, X-axis: 10 KHz, Y-axis: 10 dB) ........................................ 141

Figure 6-9 Two Signals at Different Data Rates and Frequencies (Moving at 50Hz/s) (A) 1K2 and 2K4 (B) 1K2 and 19K2 (Date: 20/09/2015, Time: 15:00:00, X-axis: 10 KHz, Y-axis: 10 dB) ........................................................................................................ 141

Figure 6-10 Setup of Case 1: 2 x DDCs Connected to 2 x Rx Ports ................................................................. 142

Figure 6-11 Setup of Case 1: 2 x DDCs Connected to Single Rx Port ............................................................... 143

Figure 6-12 Transmitter - 2 Signals Decoded (Date: 24/09/2015, Time: 11:50:00) ........................................ 144

Figure 6-13 Analysis of Signals at Different Rates that Can be Accommodated in 1.536 MHz 146

Figure 6-14 Memory Required for Each Signal at Different Symbol Rates and at Different Stages .......................................................................................................................... 147

Figure 6-15 Memory Requirement Analysis .................................................................................................... 148

Figure 6-16 Access Time v/s Symbol Rates ..................................................................................................... 149

Figure 6-17 FPGA Hardware - Overhead Analysis ......................................................................................... 151

Figure 6-18 Flowchart for Auto Configuring Concurrent Multi-Satellite Reception ............................ 153

Figure 7-1 Electra Transceiver Block Diagram ............................................................................................ 176

Figure 7-2 SCAN Test-bed (Image Credit: NASA) .................................................................................. 177

Figure 7-3 Block Diagram of the Functional Interactions of the RF Subsystem [34] .................. 177
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 2-1</td>
<td>Communication Requirement of Current and Future Missions</td>
<td>15</td>
</tr>
<tr>
<td>Table 2-2</td>
<td>List of Constellations Considered for Analysis</td>
<td>21</td>
</tr>
<tr>
<td>Table 2-3</td>
<td>Traditional Radios v/s SDRs</td>
<td>31</td>
</tr>
<tr>
<td>Table 2-4</td>
<td>List of Terrestrial FPGAs</td>
<td>36</td>
</tr>
<tr>
<td>Table 2-5</td>
<td>List of Space Graded FPGAs [129]</td>
<td>36</td>
</tr>
<tr>
<td>Table 2-6</td>
<td>Back-end Signal Processing Software Options</td>
<td>39</td>
</tr>
<tr>
<td>Table 3-1</td>
<td>Parametric Comparison of Test-beds</td>
<td>60</td>
</tr>
<tr>
<td>Table 3-2</td>
<td>Signal Diversity among CubeSats [142]</td>
<td>61</td>
</tr>
<tr>
<td>Table 3-3</td>
<td>Memory Requirement at Different Stages of Decoder at 1k2</td>
<td>68</td>
</tr>
<tr>
<td>Table 3-4</td>
<td>Memory Requirement at Different Stages of Decoder at 19k2</td>
<td>69</td>
</tr>
<tr>
<td>Table 3-5</td>
<td>SSC Ground Station Setup</td>
<td>70</td>
</tr>
<tr>
<td>Table 4-1</td>
<td>Comparison of Static Analysis Tools</td>
<td>86</td>
</tr>
<tr>
<td>Table 4-2</td>
<td>Encoder Functions and Their Tasks</td>
<td>91</td>
</tr>
<tr>
<td>Table 4-3</td>
<td>Comparison of Different Platforms</td>
<td>94</td>
</tr>
<tr>
<td>Table 4-4</td>
<td>Decoder Functions and Their Tasks</td>
<td>100</td>
</tr>
<tr>
<td>Table 4-5</td>
<td>Comparison of Instruction Set Across Different Platforms</td>
<td>106</td>
</tr>
<tr>
<td>Table 5-1</td>
<td>Improvement Achieved with Sample DDC Block on FPGA</td>
<td>115</td>
</tr>
<tr>
<td>Table 5-2</td>
<td>Overhead Analysis of DQT Implementation</td>
<td>117</td>
</tr>
<tr>
<td>Table 5-3</td>
<td>Details of the Blocks in Reconfigurable DDC</td>
<td>123</td>
</tr>
<tr>
<td>Table 6-1</td>
<td>Comparison of Different Alternatives of Parallel Architecture</td>
<td>138</td>
</tr>
<tr>
<td>Table 6-2</td>
<td>Bandwidth Required for Different Symbol Rates</td>
<td>145</td>
</tr>
<tr>
<td>Table 6-3</td>
<td>Overhead Analysis after 4 DDC Blocks</td>
<td>150</td>
</tr>
</tbody>
</table>
Software Defined Radio (SDR) Architecture for Concurrent Multi-Satellite Communications
Mamatha R. Maheshwarappa

TABLE 7-1 MRO/ELECTRA MODES, FUNCTIONS, AND PERFORMANCE ............................................. 176

TABLE 7-2 ELECTRA – FREQUENCIES .......................................................................................... 176

TABLE 7-3 OVERVIEW OF SCAN TEST-BED SPECIFICATIONS [34] .................................................... 177
LIST OF ACRONYMS

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAReST</td>
<td>Autonomous Assembly of a Reconfigurable Space Telescope</td>
</tr>
<tr>
<td>ADC</td>
<td>Analogue to Digital Converter</td>
</tr>
<tr>
<td>AM</td>
<td>Amplitude Modulation</td>
</tr>
<tr>
<td>AMSAT</td>
<td>Amateur Radio and Amateur Satellite</td>
</tr>
<tr>
<td>ARM</td>
<td>Advanced RISC Machines</td>
</tr>
<tr>
<td>ATSC</td>
<td>Advanced Television Systems Committee</td>
</tr>
<tr>
<td>AX.25</td>
<td>Amateur X.25</td>
</tr>
<tr>
<td>ASIC</td>
<td>Application Specific Integrated Circuit</td>
</tr>
<tr>
<td>AWGN</td>
<td>Additive White Gaussian Noise</td>
</tr>
<tr>
<td>BPF</td>
<td>Band Pass Filter</td>
</tr>
<tr>
<td>BPSK</td>
<td>Binary Phase Shift Keying</td>
</tr>
<tr>
<td>CAB</td>
<td>Configurable Analogue Blocks</td>
</tr>
<tr>
<td>CCSDS</td>
<td>Consultative Committee for Space Data System</td>
</tr>
<tr>
<td>CDMA2K</td>
<td>Code Division Multiple Access (3rd Generation cellular/radio technology)</td>
</tr>
<tr>
<td>CIC</td>
<td>Cascaded Integrator Comb</td>
</tr>
<tr>
<td>CMOS</td>
<td>Complementary Metal Oxide Semiconductor</td>
</tr>
<tr>
<td>COTS</td>
<td>Commercially Off-The-Shelf</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
</tr>
<tr>
<td>CRI</td>
<td>Cryptography Research Incorporated</td>
</tr>
<tr>
<td>CW</td>
<td>Continuous Wave</td>
</tr>
<tr>
<td>DAC</td>
<td>Digital to Analogue Converter</td>
</tr>
<tr>
<td>DC</td>
<td>Direct current</td>
</tr>
<tr>
<td>DDC</td>
<td>Direct Digital Converter</td>
</tr>
<tr>
<td>DDR</td>
<td>Double Data Rate</td>
</tr>
</tbody>
</table>
Software Defined Radio (SDR) Architecture for Concurrent Multi-Satellite Communications
Mamatha R. Maheshwarappa

DDS - Direct Digital Synthesizer
DECT - Digital Enhanced Cordless Telecommunications
DMA - Direct Memory Access
DQFN - Depopulated Very- Thin Quad Flat-Pack No-Leads
DQT - Digital Quadrature Tuning
DSP - Digital Signal Processing
DVB - Digital Video Broadcasting
DSS - Distributed Satellite Systems
EDAC - Error Detection and Correction
EDGE - Enhanced Data for GSM Evolution
EER - Envelope Elimination and Restoration
ENVM - Embedded Non-Volatile Memory
EVDO - Enhanced Voice-Data Optimized
ESA - European Space Agency
FCC - Federal Communications Commission
FCD - FunCube Dongle
FEC - Forward Error Correction
FFT - Fast Fourier Transform
FIR - Finite Impulse Response
FM - Frequency Modulation
FPAA - Field Programmable Analogue Array
FPGA - Field Programmable Gate Array
FSL - Free Space Loss
FSK - Frequency Shift Keying
GEO - Geostationary Earth Orbit
GPIO - General Purpose Input Output
GPP - General Purpose Processor
GPRS - General Packet Radio Service
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPS</td>
<td>Global Positioning System</td>
</tr>
<tr>
<td>GPSS</td>
<td>Global Positioning Satellite System</td>
</tr>
<tr>
<td>GSM</td>
<td>Global Systems for Mobile communication</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>HDLC</td>
<td>High-Level Data Link Control</td>
</tr>
<tr>
<td>HLS</td>
<td>High Level Synthesis</td>
</tr>
<tr>
<td>HPA</td>
<td>High Power Amplifier</td>
</tr>
<tr>
<td>HR</td>
<td>Hardware Radio</td>
</tr>
<tr>
<td>HSPA</td>
<td>High Speed Packet Access</td>
</tr>
<tr>
<td>I2C</td>
<td>Inter-Integrated Circuit</td>
</tr>
<tr>
<td>IARU</td>
<td>International Amateur Radio Union</td>
</tr>
<tr>
<td>iDEN</td>
<td>Integrated Digital Enhanced Network</td>
</tr>
<tr>
<td>IEEE</td>
<td>Institute of Electrical and Electronics Engineers</td>
</tr>
<tr>
<td>I/O</td>
<td>Input and Output</td>
</tr>
<tr>
<td>I/Q</td>
<td>In-phase and Quadrature phase</td>
</tr>
<tr>
<td>IF</td>
<td>Intermediate Frequency</td>
</tr>
<tr>
<td>IIR</td>
<td>Infinite Impulse Response</td>
</tr>
<tr>
<td>IIO</td>
<td>Industrial Input/Output</td>
</tr>
<tr>
<td>IMEC</td>
<td>Interuniversity Micro-Electronics Centre</td>
</tr>
<tr>
<td>IP</td>
<td>Intellectual Property</td>
</tr>
<tr>
<td>ISS</td>
<td>International Space Station</td>
</tr>
<tr>
<td>ISR</td>
<td>Ideal Software Radio</td>
</tr>
<tr>
<td>ITU</td>
<td>International Telecommunication Union</td>
</tr>
<tr>
<td>LEO</td>
<td>Lower Earth Orbit</td>
</tr>
<tr>
<td>LNA</td>
<td>Low Noise Amplifier</td>
</tr>
<tr>
<td>LOS</td>
<td>Line Of Sight</td>
</tr>
<tr>
<td>LPDDR</td>
<td>Low-Power, Double-Data-Rate</td>
</tr>
<tr>
<td>LPF</td>
<td>Low Pass Filter</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>------------------------------------------------</td>
</tr>
<tr>
<td>LPT</td>
<td>Low Power Transceiver</td>
</tr>
<tr>
<td>LSRAM</td>
<td>Large static random access memory</td>
</tr>
<tr>
<td>LTE</td>
<td>Long Term Evolution</td>
</tr>
<tr>
<td>LUT</td>
<td>Look Up Table</td>
</tr>
<tr>
<td>MAC</td>
<td>Media Access Control</td>
</tr>
<tr>
<td>MCU</td>
<td>Master Control Unit</td>
</tr>
<tr>
<td>MEO</td>
<td>Medium Earth Orbit</td>
</tr>
<tr>
<td>MIMO</td>
<td>Multiple Input Multiple Output</td>
</tr>
<tr>
<td>MMCM</td>
<td>Mixed Mode Clock Manager</td>
</tr>
<tr>
<td>MMUART</td>
<td>Multi-Mode UARTs</td>
</tr>
<tr>
<td>MRAM</td>
<td>Magneto-resistive Random Access Memory</td>
</tr>
<tr>
<td>MPMB</td>
<td>Multi-Protocol Multi-Band</td>
</tr>
<tr>
<td>MQFP</td>
<td>Metric Quad Flat Package</td>
</tr>
<tr>
<td>MRO</td>
<td>Mars Reconnaissance Orbiter</td>
</tr>
<tr>
<td>MSS</td>
<td>Microcontroller Sub-System</td>
</tr>
<tr>
<td>MT</td>
<td>Montana</td>
</tr>
<tr>
<td>NASA</td>
<td>National Aeronautics and Space Administration</td>
</tr>
<tr>
<td>NFS</td>
<td>Network File System</td>
</tr>
<tr>
<td>NoC</td>
<td>Network on Chip</td>
</tr>
<tr>
<td>OSS</td>
<td>Open Source Software</td>
</tr>
<tr>
<td>PA</td>
<td>Power Amplifier</td>
</tr>
<tr>
<td>PC</td>
<td>Personal Computer</td>
</tr>
<tr>
<td>PHY</td>
<td>Physical layer</td>
</tr>
<tr>
<td>PN</td>
<td>Pseudo Noise</td>
</tr>
<tr>
<td>QAM</td>
<td>Quadrature Amplitude Modulation</td>
</tr>
<tr>
<td>QPSK</td>
<td>Quadrature Phase Shift Keying</td>
</tr>
<tr>
<td>QoS</td>
<td>Quality of Service</td>
</tr>
<tr>
<td>RF</td>
<td>Radio Frequency</td>
</tr>
</tbody>
</table>

XX
Software Defined Radio (SDR) Architecture for Concurrent Multi-Satellite Communications
Mamatha R. Maheshwarappa

RISC - Reduced Instruction Set Computing
SatNOGS - Satellite Networked Open Ground Station
SCM - Spatial Channel Models
SCR - Software Controlled Radio
SDR - Software Defined Radio
SDSoC - Software Defined System on Chip
SEU - Single Event Upset
SGMII - Serial Gigabit Media Independent Interface
SNR - Signal to Noise Ratio
SoC - System on Chip
SOM - System On Module
SPI - Serial Peripheral Interface
SPS - Symbols Per Second
SRAM - Static Random Access Memory
SSB - Single Side Band
SSC - Surrey Space Centre
STK - Systems Tool Kit
STRaND - Surrey Training Research and Nano-satellite Demonstrator
STRS - Space Telecommunications Radio System
TD-SCDMA - Time Division Synchronous Code Division Multiple Access
TDRSS - Tracking and Data Relay Satellite System
TETRA - TErrestrial TRunked RAdio
TFTP - Trivial File Transfer Protocol
TLEs - Two Line Elements
TMR - Triple Modular Redundancy
TSE - Triple Speed Ethernet
UART - Universal Asynchronous Receiver/ Transmitter
UAV - Unmanned Air Vehicle
Software Defined Radio (SDR) Architecture for Concurrent Multi-Satellite Communications
Mamatha R. Maheshwarappa

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UHF</td>
<td>Ultra High Frequency</td>
</tr>
<tr>
<td>UMTS</td>
<td>Universal Mobile Telecommunications System</td>
</tr>
<tr>
<td>UniS</td>
<td>University of Surrey</td>
</tr>
<tr>
<td>USA</td>
<td>United States of America</td>
</tr>
<tr>
<td>USAF</td>
<td>United States Air Force</td>
</tr>
<tr>
<td>USB</td>
<td>Universal Serial Bus</td>
</tr>
<tr>
<td>USR</td>
<td>Ultimate Software Radio</td>
</tr>
<tr>
<td>VDMA</td>
<td>Video Direct Memory Access</td>
</tr>
<tr>
<td>VSG</td>
<td>Vector Signal Generator</td>
</tr>
<tr>
<td>VGA</td>
<td>Variable Gain Amplifier</td>
</tr>
<tr>
<td>VHDL</td>
<td>VHSIC Hardware Description Language</td>
</tr>
<tr>
<td>VHF</td>
<td>Very High Frequency</td>
</tr>
<tr>
<td>VHSIC</td>
<td>Very High Speed Integrated Circuit</td>
</tr>
<tr>
<td>WCDMA</td>
<td>Wideband Code Division Multiple Access</td>
</tr>
<tr>
<td>WLAN</td>
<td>Wireless Local Area Network</td>
</tr>
<tr>
<td>WPAN</td>
<td>Wireless Personal Area Network</td>
</tr>
</tbody>
</table>
1 INTRODUCTION

The increasing number of amateur radio/university class small satellites has created a limited spectrum for amateur satellite use, due to the inherent overlapping coverages caused by the limited discrimination of practical ground station antennas and the reduced spacing between satellites. There is therefore a need for an adaptable software platform to enable concurrent multiple-signal reception, which can dynamically change frequencies, data rate, and occupied bandwidth, and implement various radio-frequency (RF) protocol standards. In this research, possible applications are identified in the context of the fixed/mobile ground-to-space links (with the technology implemented on the ground segment), however, in the future, the techniques developed should be applicable to the satellite platforms themselves.

A Software Defined Radio (SDR) with commercial-off-the-shelf (COTS) open source hardware and software tools, implemented on low resource embedded systems is proposed as a feasible platform on which to implement this capability. Referring to the evolution of transceivers and current state-of-the-art systems, the combination of a COTS RF front-end and digital-signal processing (DSP)/baseband back-end system-on-chip (SoC) technologies, along with open source software tools, appears to be a promising solution for the implementation of such a platform. New field-programmable gate-array (FPGA) intellectual property (IP) cores have been developed in order to improve the central processing unit (CPU) load. Such improvements have achieved up to 36% load reduction at 1200 bits-per-second (bps), 31% at 2400 bps, 21% at 4800 bps and 0.7% at 19200 bps date-rates, and this has allowed the design, implementation and validation of multiple signals using a novel pipeline architecture. This technology therefore appears suitable to support concurrent multiple decoders for both fixed/mobile ground station applications and, in the future, for implementation on a satellite platform, such as a CubeSat.

Small satellites, have moved on from being a niche tool for space engineering research and development to now being used to implement practical/commercial missions, such as disaster monitoring, Earth observation, communications and navigation services [1]. Small satellites are
Software Defined Radio (SDR) Architecture for Concurrent Multi-Satellite Communications
Mamatha R. Maheshwarappa

attractive due to their reduced build time, lower cost, potential for more frequent launch opportunities, and greater involvement of small industries and universities [2]. This is evident in Spaceworks’ projection (Figure 1-1) based on announced and future plans of developers and programs indicating 3000 nano/microsatellites will be launched from 2016 through 2022 [3].

![Figure 1-1 Spaceworks' Projection of Small Satellites till 2020 [3]](image)

Also, there is an increasing trend of implementing small satellite formations/constellations due to [4, 5] their relative simplicity, low cost and low mass, which makes them the ideal candidates for low Earth orbit (LEO) constellations [6]. Figure 1-2 shows a categorisation of future missions based on the number of satellites and the type of mission [12, 13].

![Figure 1-2 Categorisation of Thirty-Nine Multi-Satellite Missions [6]](image)
The increasing number of CubeSats has also caused the shortage of available frequencies in VHF, UHF and S-bands due to rapid increase in the number of CubeSats orbiting the Earth. On December 12, 1961 OSCAR-1 was successfully deployed as a secondary payload from Discoverer 36 into Low Earth Orbit (LEO) and was received by over 570 amateur radio operators around the world during its 22 day lifetime [7]. The designation OSCAR is still in use today and stands for “Orbiting Spacecraft Carrying Amateur Radio”. This marked the beginning of over 50 years of space operations for amateur radio enthusiasts. Universities, government and commercial institutions developing CubeSats and smallsats for the purposes of performing space-based research still rely heavily on amateur communication technology and frequency allocations.

This has created strain for licensing and coordinating organisations, such as the Federal Communications Commission (FCC) and the International Amateur Radio Union (IARU) [8]. As mentioned at several recent workshops [9, 10], the FCC and International Telecommunication Union (ITU) are in process of reviewing licensing for small satellites. It is, therefore, important to develop an efficient way of utilising this limited bandwidth resource. This expected shortage of bandwidth has prompted researchers to explore new ways of efficiently using limited bandwidth [11, 12]. At the current and projected growth rates, the existing small satellite communication systems will not be able to fully support these challenges into the future. Also, it has been recognised, that spectrum will become significantly more crowded [13] to a point where interference is much more likely and where simple reception techniques are likely to be more prone to failure and/or be too error prone.

Figure 1-3 Screenshot of Gpredict Showing the Number of Small Satellites in Space

Already, the interference between communication links has increased due to overlapping coverages caused by the increase in the number of satellites orbiting the Earth. This is
Software Defined Radio (SDR) Architecture for Concurrent Multi-Satellite Communications
Mamatha R. Maheshwarappa

demonstrated in Figure 1-4, a screenshot of Gpredict [14] – real time tracking and prediction algorithm driven by Two Line Elements (TLEs) downloaded at NORAD/CelesTrak [15] as of 2014. A TLE is a data format encoding a list of orbital elements of an Earth-orbiting object for a given point in time, the epoch. The United States Air Force (USAF) tracks all detectable objects in Earth orbit, creating a corresponding TLE for each object, and makes available TLEs for non-classified objects on the website of CelesTrak.

This problem could be met to some extent by using switched beam arrays [16], phased arrays and smart antennas [17] and digital beam forming [18] which allows reception from multiple satellites simultaneously by applying a phase shift across the array aperture [19]. There is a growing need to efficiently increase the number of communication links with a given setup (antenna and high power amplifier (HPA)) and make the most of the available resources. Programmes such as SatNOGs [20] exist, their motivation is to solve this problem using a crowd funded/crowd sourced/large scale/cloud based approach. However, this in itself does not provide any assurance that important mission data will be received and be available to an intended user in a timely fashion. In contrast, we want to look at the problem of optimally utilising a single antenna at a single geographic site. The underlying technology, though initially ground based, may also be applied to future low cost CubeSat constellations.

Figure 1-4 Radar View of the Antenna Showing Different Satellites in Visibility (Left) VHF/UHF Antenna Setup at UniS (Right)
Software Defined Radio (SDR) Architecture for Concurrent Multi-Satellite Communications
Mamatha R. Maheshwarappa

Discussion: If we had, in the distant future, a reasonably priced ground based phased array with a large number of high gain beams (e.g. > 30 beams at > 30 dBi), then all satellites could continue to operate on a single receive chain for each band in operation. In practice, it is significantly cheaper to use one or more simple Yagi type antennas (of medium gain 10-15dBi) and support concurrent multi-channel operation using a pre-agreed (IARU coordinated) frequency plan. Therefore, previously accepted communications setups with dedicated hardware for a particular mission and frequency band communicating to one spacecraft at a given time now becomes the bottleneck. Further, the diversity of signal standards in operation, mean that no single commercial ground modem has come to dominate the small satellite/CubeSat market.

Therefore, this thesis will address the issues discussed earlier:

- The ground communication systems, currently in use, do not support the concurrent reception from multiple satellites and need to be replaced by parallel architecture approaches.
- A re-configurable communication module is needed to support in-flight changes to increase flexibility, allowing the implementation of different signal processing elements and the ability to recognise and mitigate interference issues without altering the system hardware.

Therefore, the solution, proposed in this thesis, focuses on:

1. Reconfigurable platform – adaptive (performance, reliability) for mission needs.
2. Multifaceted hybrid computing – Agile mix of COTS; fixed & reconfigurable.
3. Scalable building blocks – address needs of fixed/mobile ground station.

One of the possible strategies to solve these issues is to equip satellites/ground segments with a Software Defined Radio (SDR) – that is a "radio in which some or all the physical layer functions are software defined" [21]. SDRs offer functionalities that are otherwise hard to achieve, such as re-configurability, adaptability and autonomy, which in-turn enables a limited amount of subsystem redesign and which can be developed towards a generic satellite communication solution. The benefits offered by SDRs and the continuous advances in commercial digital electronics, have triggered interest in advanced communication systems for small satellites [22]. This increase in subsystem performance can potentially offer satellite communications to loosen constraints on modulation parameters based on link conditions, frequency bands, Doppler uncertainties and data rates at minimum cost, thereby making dynamic multiband access and sharing possible.

On the other hand, recent advances in signal processing combined with those in antenna technology provide Multiple-Input Multiple-Output (MIMO) capabilities [23], which create
opportunities in enhancing the throughput of wireless networks. Both SDR and MIMO technologies, together, enable next generation wireless networks [24], such as mesh networks, to support dynamic and adaptive bandwidth sharing of time, frequency, and space [25]. This flexibility and adaptability comes, however, at the expense of power consumption and complexity. Since its conception in 1995 [21], the growth of terrestrial SDRs has been “exponential” as seen in Figure 1-7. They have evolved, not only in technology, from being solely receivers to transceivers but also in size, mass and power requirements. This range of options is continually increasing and some of these will undoubtedly find mainstream acceptance in one form or another in the future terrestrial/space SDRs.

![Graph](image.png)

**Figure 1-5 Evolution of Software Defined Radios**

Conventionally, space SDRs were primarily seen as a payload for bigger satellites, such as STS-107 and Mars Reconnaissance Orbiter (MRO) [26], or as a test bed on International Space Station (ISS) [27]. Very few SDRs have flown on small satellite missions [28-30] due to complexity in technology, extreme power requirements and cost.

In the recent years there has been considerable growth in re-programmable SDRs with frequency and modulation agility as seen in [31, 32]. Though the technology advances can improve bandwidth and computing capabilities, their power requirement still remains as a major concern for small satellite missions.

Despite space being an expensive business, the cost of building satellites has been reduced through two significant trends: namely, the use of COTS parts and miniaturisation. More than 30 years of research at the University of Surrey has shown that small satellite missions can be conducted relatively quickly and inexpensively by using COTS technologies, or they can be
enhanced by using advanced technologies [33]. As a result, today, Surrey Satellite Technology Limited (SSTL), a spin-out company from the University of Surrey, is the leader in developing and manufacturing small satellites by using COTS components as they are both small in mass and low in cost [34, 35]. These COTS components are tested through strict environmental tests such as vibration, thermal vacuum and radiation tests to ensure that they are qualified for space applications. The natural progression of component miniaturisation in modern day electronics has also helped in reducing the physical size and mass of building small satellites so that even smaller satellites can be developed which utilise state-of-the-art integrated electronics solutions (for example: STRaND-1 [36] – the UK’s first CubeSat), also reducing the power consumption. This research proposes one such COTS solution to address the future problems of ever increasing band utilisation and encouraging the small satellite community.

1.1 Research Motivation

SDR is beneficial for space applications as it provides the flexibility that will allow deployed satellite communication equipment to be software upgradable to more advanced on-board algorithms and communication standards. This will allow communication functionality changes and multiple uses during lifetime of the satellite mission. Although examples of how to design constellations of small satellites exist and demonstrate their value for Earth and planetary observation [37-39], most studies focus on the benefits that the global distribution of measurements will provide and not on the the potential complexities and mission vulnerabilities of computing in the constellation.

The International Telecommunication Union (ITU) oversees the frequency allocations within the radio spectrum and sets aside some for the amateur radio use. Amateur/University satellites/CubeSats are tend to use these frequencies, and may be authorised to do so, provided they meet the strict requirements of the amateur radio satellite service. These are coordinated by the International Amateur Radio Union (IARU). Radio amateurs use various transmission modes such as Morse code, data and voice. Frequency allocations, and bandwidth vary from country to country and between ITU regions and frequency bands [45]. For example, in the VHF range, the bandwidth of individual transmissions allocated to the space operation services (Earth-to-space) shall not exceed ±25 kHz. The ITU also deals with interference issues. For instance, in the L-band (1215-1260 MHz), active spaceborne sensors in the Earth exploration-satellite and space research services shall not cause harmful interference to, claim protection from, or otherwise impose constraints on operation or development of the radiolocation service, the radio navigation-satellite service and other services allocated on a primary basis. Radio communication services operating within industrial-scientific-medical (ISM) bands must accept harmful interference that may be caused by these applications.
The studies that have addressed computing aspects in [40] and [41] primarily rely on a large number of ground stations in their networking strategies [42]. This study intends to complement the earlier work by considering a communication channel that relies on a single ground station to receive multiple satellites, with different signals, simultaneously. Such an approach could greatly expand the capability of distributed satellite systems (DSSs) [43-45]. To the best of author’s knowledge, the implementation of SDRs for concurrent multiple small satellite scenarios has not been investigated before. The main purpose of the proposed platform is to enable the concurrent reception of multiple signals from multiple satellites operating in constellations, clusters, and formations of small satellites such as CPOD [64] and EDSN [65].

1.2 Aim and Objectives

The aim is to create a new class of SoC-based SDR to receive signals from multiple satellites, at a single ground-station, within the mass and power constraints of an embedded system.

The research is aimed at:

1. Advancing the state-of-the-art in SDR systems for concurrent multi-channel communications.

2. Creating a new generic software methodology and framework for SDR applications that informs future evolutions in hardware and software.

The main objectives of this research are as follows:

1. Investigate state-of-the-art terrestrial SDR technologies that could be adapted for concurrent multi-satellite applications and exploring the current terrestrial and space SDR missions.

2. Review and classify the mission requirements involving multiple satellites.

3. Investigate the key bottlenecks in terrestrial systems in terms of application requirements, algorithmic capabilities and hardware limitations.

4. Port and characterise the performance of existing SDR software chains in an embedded system.

5. Improve processing speed and memory performance by parallelisation in Field Programmable Gate Arrays (FPGAs) targeting high performance at low power.

6. Demonstrate concurrent multi-channel reception through parallel processing by combined SDR and SoC system.
7. Validate parallel processing chain through experimentation with real-time recorded satellite signals from CubeSats such as FUNcube-1 [46], and generated multi-satellite signals.

1.3 Novelty and Contributions

The main contribution of this work is the development of a new parallel architecture using advanced SDR techniques on modern SoC based embedded system to enable flexible telemetry, tracking and command (TT&C) communication for amateur radio/university class small satellites and receive multiple concurrent satellite signals of different standards (such as different modulation techniques, data rates, frequencies) on ground system/distributed satellite systems.

The work described in this thesis has been (or is being) published in the following Journal Papers and proceedings:


- M. Maheshwarappa, M. Bowyer and C. P. Bridges, “SDR Performance of Different CPU Cores for Small Satellite Applications” revision submitted for manuscript acceptance in the Journal ‘IEEE transactions on Aerospace and Electronic Systems’. [Accepted]

- M. Maheshwarappa and C. P. Bridges, ‘Software Defined Radios (SDRs) for Small Satellites’ Proceeding of the NASA/ESA Conference on Adaptive Hardware and Systems at University of Leicester, 2014. DOI: 10.1109/AHS.2014.6880174


1.4 Outline of the Thesis

This thesis is divided into the following chapters:

In Chapter 2, a literature review is presented on the increasing number of satellite missions – including multi-satellite systems. The gaps in the existing communication systems are identified and the evolution of communication systems is reviewed. SDR is proposed as a potential solution to fill these gaps, along with the challenges associated with the present state-of-the-art technologies (both hardware/software and at system level). Finally, the unique engineering environment with the complex set of challenges associated with space, is discussed.

In Chapter 3, the drivers and requirements of this research are defined, followed by an investigation of different front-/back-end COTS platforms from two perspectives:

Firstly, from the perspective of the hardware capabilities, in terms of speed, resource availability, number of CPUs; Secondly, from the perspective of ease of use in terms of porting, reference design and software support. A new transceiver architecture is proposed, based on the chosen platforms, along with a detailed description of the transceiver implementation and validation processes.

In Chapter 4, the analysis and partitioning of different blocks in an implemented transceiver program and processor resource management, is carried out through profiling. The behaviour of these blocks is also studied on different platforms and at different symbol rates. Finally, the chapter summarises the findings.

In Chapter 5, based on the previous findings, the architecture is revised in order to efficiently utilise the FPGA firmware and take advantage of its flexibility and speed - towards achieving parallel reception. Implementation and validation of the amended architecture is discussed, along with post-implementation analysis through profiling.

Finally, in Chapter 6, the different stages of parallel architecture are discussed, based on the work carried out throughout the research. Specific scenarios are simulated for validation of the novel parallel architecture. Finally, the future work is discussed towards improving the current work. Chapter 7 describes the conclusions of this thesis. A flowchart illustrating the contents of this thesis is shown in Figure 1-8.
Software Defined Radio (SDR) Architecture for Concurrent Multi-Satellite Communications
Mamatha R. Maheshwarappa

Introduction

Figure 1-6 Flowchart Showing the Outline of this Thesis
2 MULTI-SATELLITE SYSTEMS & SDR TECHNOLOGIES

This chapter reviews the communication challenges associated with an increasing number of small satellites in space, and the relevant background associated with the on-going technology evolution. Section 1.1 introduces the current trend in the frequency band utilisation, modulation techniques and data rates used in CubeSats, along with the need for a low cost flexible ground station to support multi-satellite communications. The evolution of transceivers from traditional to Software Defined Radios (SDRs) in an effort to meet the communication challenges discussed, is reviewed in Section 1.2. Section 1.3 summarises the evolution of different technologies towards enabling SDR systems. This section also aims at describing the challenges and constraints experienced by the designers and implementers of SDR based systems. Existing state-of-the-art space SDRs are discussed along with a review of generic problems for space in Section 1.4. Finally, section 1.5 gives the summary of the chapter.

2.1 Introduction

Small satellites are becoming a way to perform scientific and technological missions more affordably [47]. The objectives of these missions are very ambitious and are driven by new complexities such as limited bandwidth resource and interference between adjacent channels [48] which require multi-mode operation of wireless transceivers [49]. In addition, there are still challenges to reliably increase the communication window [50], mission throughput and supporting concurrent multiple signal scenarios. These issues can be found in various combinations from fixed to mobile ground stations.
2.1.1 Multi-Satellite Missions

The growth of technologies offers small satellites the opportunity to improve the way space missions develop and operate transceivers for communication networks in space. The ability to change the operating characteristics of a radio through software once deployed to space, offers the flexibility to adapt to new science opportunities, recover from anomalies within the science payload or communication system, and potentially reduce development cost and risk, by adapting generic space platforms to meet specific mission requirements. However, the flexibility and adaptability comes at the expense of power consumption and complexity in integrating previously separated building blocks on an integrated flexible radio platform. Here are few imminent missions that would benefit from such a flexible platform:

2.1.1.1 Edison Demonstration of Smallsat Networks (EDSN)

The EDISON mission will launch a loose formation of eight CubeSats as seen in Figure 2-5 into orbit approximately 500 km above Earth. The aim of this mission is to develop the technology to send multiple, advanced, yet affordable nanosatellites into space with cross-link communications (i.e. inter-satellite links – ISLs) to enable a wide array of scientific, commercial, and academic research at low cost and development time. Each EDSN nanosatellite is a 1.5 unit CubeSat with dimensions of about 10 x 10 x 17 cm and a mass of about 2 kg. EDSN will demonstrate a communication concept in which the individual satellites will share their collected data and one of the satellites using an ISM frequency band link and will transmit the data to a ground station on UHF [51] [51, 52].

2.1.1.2 Solar Observing Low-frequency Array for Radio Astronomy/Separated Antennas Reconfigurable Array (SOLARA/SARA)

SOLARA's primary objective is to make solar observations. SOLARA/SARA will be composed of at least 12 individual spacecraft, seen in Figure 2-6. Radio astronomy, especially at long wavelengths and low intensities, is highly sensitive to terrestrial radio noise communication and thus the
Software Defined Radio (SDR) Architecture for Concurrent Multi-Satellite Communications  
Mamatha R. Maheshwarappa

constellation would need to be placed as far as possible from Earth in a Distant Retrograde Orbit (DRO) \( \sim 1 \) million km away from Earth. SARA will use small, low power patch antennas integrated into each unit of SOLARA to form a high EIRP (Equivalent Isotropically Radiated Power) distributed antenna by combining the signals from each individual antenna in phase [53]. A synthesized beam composed of many spacecraft antennas will allow higher data rate communications for CubeSats without the need for deployable/inflatable antennas or high power amplifiers. Each spacecraft consists of two S-band channels, one for satellite-to-ground communication and the other for the ISL [54].

### 2.1.1.3 ExtraSolar Observing Low-frequency Array of Nano Satellites for Radio Astronomy (XSOLANTRA)

![Figure 2-3 XSOLANTRA Side View](image)

XSOLANTRA in Figure 2-7 is a conceptual study of a student designed, built, and tested micro-satellite mission to a DRO at 1.2 million kms from Earth. XSOLANTRA will look at Electron Cyclotron Maser Emission generated by the interaction between stellar wind and planetary magnetosphere from which interior composition and atmospheric shielding can be inferred. The science instrument for XSOLANTRA is the entire array of fourteen 3U CubeSats operating together as an interferometer. SHERPA, called Mothership will hold these 14 CubeSats until it is inserted into the DRO orbit. The CubeSats will communicate with the Mothership via a UHF relay. This range is between 0.4 – 0.6 GHz. The primary constraint is the numerous frequencies involved in this mission. The first range involves science collection in the 0.1 MHz to 10 MHz range, UHF for communication between Mothership and CubeSats and Mothership includes X-band and L-band for downlink and uplink respectively [55], which not only affects the hardware, but also the power and bit rate.

### 2.1.1.4 One Web

![Figure 2-4 One Web Constellation](image)

One Web is a constellation of approximately 648 satellites expected to provide global internet broadband service to individual customers by 2019 [56]. Each satellite in the constellation will have a mass of 175–200 kg. The 720 operational satellites will orbit in 20 different orbital planes at...
1,200 km altitude as seen in Figure 2-8. The satellites will operate in the Ku band (12-18 GHz). Each satellite will generate 6 Gbps of throughput. The antenna on each smallsat will be a phased array antenna measuring approximately 36 x 16 cm (14.2 x 6.3 in) and will provide internet access at 50 Mbps. It will use a technique called “progressive pitch” in which the satellites are slightly turned to avoid interference with Ku-band satellites in geostationary orbit.

Table 2-1 gives an overview of the frequency band, range, power and data rate targeted in the missions discussed. The recent OneWeb constellation with approximately 648 satellites has created concerns about the amount of electromagnetic interference that the constellation could add to existing terrestrial transceivers.

<table>
<thead>
<tr>
<th>Table 2-1 Communication Requirement of Current and Future Missions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Downlink</strong></td>
</tr>
<tr>
<td><strong>EDSN</strong></td>
</tr>
<tr>
<td>Range : 450-500 km</td>
</tr>
<tr>
<td>Power : 1W</td>
</tr>
<tr>
<td>Data rate : 9.6/38.4 kbps</td>
</tr>
<tr>
<td>Range : 1 x 10^6 km</td>
</tr>
<tr>
<td><strong>XSOLANTRA</strong></td>
</tr>
<tr>
<td>Range : 1.2 x 10^6 km</td>
</tr>
<tr>
<td>Power : 140 W</td>
</tr>
<tr>
<td>Data rate : 37.7 Mbps</td>
</tr>
<tr>
<td><strong>One Web</strong></td>
</tr>
<tr>
<td>Range : 1200 km</td>
</tr>
<tr>
<td>Data rate : 6 Gbps</td>
</tr>
</tbody>
</table>

2.1.2 Ground Station Systems

The increasing number of satellites (70% increase in the past three years) as seen in Figure 2-10, in LEO, occupying the amateur radio spectrum, together with the multi-satellite missions with variety of modulation techniques, data rates and protocols [57] proposed across the CubeSat community demands, the inclusion of a multitude of communication standards onto a single platform. The most common bands for the Amateur satellite service are the 2m (VHF) and 70 cm (UHF) bands [58]. Figure 2-11 highlights the trend in the frequencies for CubeSat...
Software Defined Radio (SDR) Architecture for Concurrent Multi-Satellite Communications
Mamatha R. Maheshwarappa

communications. Licensing of these spacecraft is accomplished through the Federal Communication Commission (FCC) and the spacecraft radio operations are governed by Federal Regulations [59]. Figure 2-12 and Figure 2-13 shows the most commonly used modulation technique and data rates respectively for downlink and satellite beacons. The most popular modulation schemes employed by the currently active CubeSats [56] are Audio Frequency Shift Keying (AFSK) [60] and Frequency Shift Keying / Gaussian Minimum Shift Keying (FSK/GMSK) [61].

Figure 2-5 CubeSats Launched Each Year (2000 - Present)
The ‘other’ modulation schemes depicted in the figure include a mixture of less frequently used modulation schemes, mostly consisting of Single Side Band (SSB) Amplitude Modulation (AM) and Frequency Modulation (FM) used for voice operations by the satellites that provide a voice relay service to licensed amateur radio operators. Also included in the ‘other’ group is the CW (Morse Code) beacons utilised by some of the spacecraft. The most popular data rates are mainly 1200 and 9600 bps but in the recent years, CubeSats are aimed at using higher data rates, for example the Flock-1 satellites (120 Mbps) [62] and SSTL’s Sapphire (4Mbps)[63].
The increasing number of clusters/formations/constellations of small satellites is leading to a scenario, as seen in Figure 2-14, where multiple satellites (in this case 6 of the CAMSAT (CAS-3) mission) are over a ground station at the same time. The reception reported on 20th September 2015 by JA1OGZ shows a noisy VHF spectrum with the centre frequencies of each satellite separated by ~65 KHz and the actual signal is overlaid by noise and harmonics of adjacent channels, which makes the detection/decoding of the valid signal challenging. The present technology cannot fully support concurrent multi-satellite reception at the same time using a single setup.

---

1 Survey as of December 2015
Software Defined Radio (SDR) Architecture for Concurrent Multi-Satellite Communications
Mamatha R. Maheshwarappa

Figure 2-9 CAMSAT Reception report on Sep 20 2015 8:10 UTC, JA1OGZ, Akira

2.1.2.1 Systems Tool Kit (STK) Analysis

STK is a 2D/3D environment used to model systems such as aircraft, satellite ground vehicles, and their sensors to evaluate their performance in real or simulated time [64]. STK 9.2 was used to understand the congestion of active satellites in space, based TLEs from SpaceTrak [65]. The 3D projection of the cone (30 deg) as in Figure 2-16 represents average antenna cone angle (typically varies between 25 – 40 deg) and the dots represent active satellites orbiting Earth.
The earlier study in Section 2.1.1 suggests that the trend in using constellations/formations of small satellites means there is a shift away from single satellite missions and a move towards the widespread adoption of multiple-satellite networks with rapid replenishment. Four existing active constellations in LEO were chosen in this simulation to understand the conjunctions caused by a constellation of satellites over a given ground station (in this case it is the Surrey Space Centre, and Guildford, UK). These constellations were chosen as they communicate in the frequency bands of prime interest to the small/amateur radio satellite community (VHF, UHF and S-bands). Also, the signals from these LEO satellites particularly suffer from the effects of Doppler shift and interference (as satellites in LEO travel at a high speed of 7.5 km/s relative to the ground and due to greater number of satellites working in the same frequency band when compared to medium Earth orbit (MEO) or geostationary Earth orbit (GEO)), which makes the detection of these signals more challenging. The four constellations are listed in Table 2-2 and different coloured lines in Figure 2-18 represent different constellations.
<table>
<thead>
<tr>
<th>Constellation</th>
<th>No of satellites</th>
<th>Ground to Satellite Link</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLOCK 1 (Planet Labs Inc) [62]</td>
<td>36 nanosatellite (3U CubeSats)</td>
<td>Telemetry: VHF (1200 bps) Two way link: S-band (115 kbps) X-band : 120 Mbps</td>
<td>Earth Imaging</td>
</tr>
<tr>
<td>Yaogan (CAST) [66]</td>
<td>37 remote sensing satellites</td>
<td>TT&amp;C: S-band Two way link : X-band</td>
<td>Remote Sensing</td>
</tr>
<tr>
<td>Shijian (CAST) [67]</td>
<td>20 mini satellite</td>
<td>S-band (1 – 4 Mbps)</td>
<td>Scientific research and technological experiments</td>
</tr>
<tr>
<td>Formosat [68] (Taiwan/USA/SSTL)</td>
<td>6 microsatellite</td>
<td>S-band Uplink: 32 kbps S-band Downlink: 2 Mbps</td>
<td>Remote sensing</td>
</tr>
</tbody>
</table>

The potential for interference is minimal if satellites are distributed in different planes compared to orbits that are very close to each other, as in the case of FLOCK_1E series (shown in Figure 2-18). Figure 2-19 shows the 3D projection of the antenna cone with 5 satellites (3 from Yaogan series and 2 from FLOCK-1). The challenge here is not only that there are multiple satellites at a time, but also these satellites belong to different constellations with varied communication standards. It is also important to note that these satellites are at different altitudes and ranges which means the Doppler effect is different for each satellite. Therefore, the technology not only needs to support concurrent multiple signals but also be able to adapt to different standards, Doppler effects and mitigate interferences. Traditionally, a ground station has a separate setup for each mission – including a dedicated transceiver (e.g. ICOM [69]) and modem (Terminal Node Controller [70]). In the recent years, the trend has been to move towards a wide-band front-end (e.g. Dongles [71]) which can be plugged in to a laptop/PC with a GNURadio [135] that can record signals of different standards and process them separately.
However, this adaptation requires a ground station to be based on a flexible/generic reconfigurable architecture. This architecture would also suit a worldwide distributed system, such as ESA’s Global Educational Network for Satellite Operations (GENSO) system [48] and Satellite Networked Open Ground Station (SatNOGS) [27], where updates containing the software for communicating with new signals could be shared with different distant stations without the need for hardware upgrades, and could aid the multi-satellite missions discussed earlier.
2.2 Multiple Input Multiple Output (MIMO)

MIMO technology is a wireless technology that uses multiple transmitters and receivers to transfer more data at the same time [225]. MIMO systems have been a focus of academic and industrial research in the last decade [72, 73] as they offer the promise of increased capacity, high spectral efficiency and high gains. The fundamental basis of MIMO techniques is to overcome multi-path propagation in the radio channel. Multi-path is an effect, which arises when the radio signal travels from transmitter to receiver via multiple paths rather than a single, dominant Line Of Sight (LOS) path. The multiple paths occur due to reflection and scattering from objects such as buildings, trees and the general geographic features. The paths interfere at the receiver to cause Rayleigh fading [74]. If the multiple antenna elements are sufficiently separated, the fading at different elements may be largely uncorrelated, allowing diversity reception.

In terrestrial wireless communication, MIMO has demonstrated significant results. Channel modelling for terrestrial MIMO wireless communication systems has received research attention within the last two decades resulting in a number of standardised MIMO [75] spatial channel models (SCM) [76, 77]. However, as far as satellite channels are concerned, it is not as straightforward as for terrestrial communications. Satellite channels rarely produce independent delay profiles required for the MIMO concept to work. It is dominated by LOS (Line of Sight) and higher FSL (Free Space Loss) [78].

A generalised MIMO model was proposed in [34] using a deterministic channel modelling approach for satellite formation flying systems. This model was based on the summation of sinusoids approach, which modelled the non-LOS component as a summation of rays departing the satellites and arriving the ground receiving station with different angle of arrivals, angle of departures, delay of arrivals, and complex amplitudes. This simplified channel modelling approach could not account for the effect of rays that share common or very closely spaced multipath propagation parameters. In order to overcome this, a cluster based modelling approach called the Spatial Channel Model (SCM) was proposed [79]. This model accounted for both Earth and satellite propagation parameters such as delay of arrival, angle of arrival, angle of departure, delay spread, and power angular spectrum and space communication effects including ionospheric power loss and ionospheric angular deviations. An improved signal quality and capacity was achieved by the use of the Orthogonal Frequency Division Multiplexing (OFDM) technique [80], which made the signals more robust against delay spread in frequency selective channel.
However, the MIMO system is limited to single/multiple reception of similar, or the same, sources. It does not support reception of signal from different satellites with varied communication standards and signals from satellites of different constellations at the same time. Therefore, along with MIMO there is a potential need for a universal programmable hardware that can, not only provide better signal quality and capacity, but also provide generic communication platforms for concurrent multi-satellite reception.

2.3 General Communication Systems

Wireless radios for the mobile communication systems have passed through several generations, from traditional radios to Ultimate Software Radios (USRs). Here follows an overview of the evolution of transceiver architecture towards providing a single platform that could be reprogrammed to process a variety of signals:

The SDR forum, a non-profit corporation set up to support the development, deployment and use the open architectures for advanced wireless systems, has developed a multi-tiered definition of SDR. Tier 0 represents ‘traditional’ radio hardware and forms a baseline reference. The uppermost tier, Tier 4, represents the ‘ultimate’ version of SDR. Reality falls somewhere in the middle. For most applications, the state-of-the-art SDR currently aligns with Tier 1 and Tier 2 definitions. Virtually all modem wireless communications equipment may be classified as being software-controlled radios (i.e., Tier 1).

The review is mainly based on [81] and [82].

In order to aid visualising the concepts of SDR Forum’s tier definitions, it is possible to consider the functionality contained within a radio to identify, in broad terms, how the concepts fit within a radio. Figure 2-20 illustrates an abstraction of the 5 Tier definition [83], where the length of the arrow indicates the proportion of the software content within the radio. For example, it can be seen that, at Tier 0 there is very little software element by virtue of the length of the arrow being minimal. Conversely, at Tier 4 the USR, where the entire signal at the output of the antenna has been digitised and operates within a software environment.
Note: Length of arrows indicates software contents of radio

Figure 2-13 High-Level Abstraction of the SDR Forum Tier Definition
2.3.1 Tier 4 - Ultimate Software Radio (USR)

USR represents the 'blue-sky' vision of SDR and are defined for comparison purposes only. It accepts fully programmable traffic, controls information and supports operation of broad range of frequencies and can switch from one air-interface/application to another in milliseconds.

2.3.2 Tier 3 - Ideal Software Radio (ISR)

In ISR, programmability extends to the entire system with analog conversion only at the antenna, speaker and microphones. The concept of ISR first appeared with the work of Mitola [84, 85] in 1995. He proposed to create a radio to adjust to several communication scenarios automatically. A signal incident on the antenna port is routed to a low noise amplifier (LNA) through a circulator which is then digitised for further signal processing. Demodulation and decoding are accomplished in a number of modulation formats and access schemes using a DSP processor/FPGA. This concept has driven many researchers to study software defined approaches [86].

The ideal software radio has following features:

1. The modulation scheme, channelisation, protocols, and equalisation for transmit and receive are all determined in software within the digital processing subsystem.

2. The ideal circulator is used to separate transmit and receive path signals, without the usual frequency restrictions placed upon this function. Ideal circulators have all the ports with same Voltage Standing Wave Ratio (VSWR) and isolation is constant across the frequency bands. A non-ideal properties of a circulator would have effect on the design due to the frequency dependency of the circulator parameters, the feedback associated with multiple reflections between circulator ports and terminators, and the possibility of spurious pass-bands in the circulator characteristics way from the desired operating frequency range [87].

3. Anti-alias and reconstruction filtering is clearly required in this architecture in order to restrict the bandwidth of the desired signal in a wide spectrum.

Since ISR is too far from reality due to several hardware and software limitations such as ADC sampling rate, processing speed etc.

2.3.3 Tier 2 - Software Defined Radio (SDR)

A SDR is a form of transceiver in which ideally all aspects of its operation are determined using versatile, general-purpose hardware whose configuration is under software control as seen in
Software Defined Radio (SDR) Architecture for Concurrent Multi-Satellite Communications  
Mamatha R. Maheshwarappa

Figure 2-22. Such a solution allows inexpensive, efficient interoperability between the available standards and frequency bands. SDRs provide software control of a variety of modulation techniques, wide-band or narrow-band operation, communication security functions, and signal requirements of current and evolving standards over a broad frequency range [88].

Thus, the signal path can be reconfigured in software without requiring any hardware modifications. The frequency bands covered may still be constrained at the front-end, requiring a switch in the antenna system and wide band analogue-to-digital converters/digital-to-analogue converters (ADCs/DACs). This research focuses on moving towards SDR, whilst taking the advantage of its re-configurability in order to process multiple signals of different standards simultaneously. It is reasonable to assume that software defined radios are “radios that provide software control of a variety of modulation techniques, wideband or narrowband operation, communications security functions (such as frequency hopping), and signal requirements of current and evolving standards over a broad frequency range”.

Figure 2-14 SDR Architecture with Baseband SoC and RF SoC

It is important that SDR is not confused with application software and other software not associated with the radio. Thus, considering the open systems interconnection (OSI) reference model, shown in Figure 2-23, SDR refers in general to functionality within the physical and data link layers and perhaps parts of the network layer but functionality in the higher layers in not specific to SDR.
2.3.4 Tier 1 - Software Controlled Radio (SCR)

Only the control functions of an SCR are implemented in software, thus only limited functions are changeable using software. Typically, this extends to interconnects, power levels etc. but not to frequency bands and/or modulation types etc. Direct Conversion/zero-IF receiver [89], as shown in Figure 2-24 are the best examples for SCRs. This architecture allows the RF signal to be converted to baseband directly.

![Software Defined Radio (SDR) Architecture for Concurrent Multi-Satellite Communications](image)

**Figure 2-15 The seven-layer OSI reference model [98]**

**Figure 2-16 Zero IF Receiver and Transmitter Architecture [106]**
The received signal is selected at RF by a band-pass filter, and then it is amplified by an LNA, as in the previous architecture. Finally, it is directly down converted to baseband by a mixer (or two mixers with a delay of 90° between them) and converted to the digital domain using a straightforward ADC.

The advantages of this architecture are:

1. Reduction in the number of analogue components compared to hardware radio allows the use of a filter having much less stringent specification than the image-reject filter and makes the high-level integration easier.

2. Can make use of a high level of integration, making it a common architecture for multiband receivers such as the one described in [90] and for complete transceiver architectures as in [91] and [92].

The disadvantages are:

1. Direct translation can generate some issues, such as DC offset.

2. Other issues are related to second-order inter-modulation products that are generated around DC, and, since the mixer output is baseband signal, it can be easily corrupted by the large flicker noise [93] / “pink” noise (signal with a frequency spectrum such that the power spectral density is inversely proportional to the frequency), of the mixer.

3. Carrier leakage and phase gain mismatch in full duplex operation.

The previous generation was configuration similar to zero-intermediate frequency (zero-IF) known as the low-IF architecture [94] in which the RF signals are mixed down to a non-zero low to moderate IF instead of going directly to DC.

2.3.5 Tier 0 - Hardware Radio (HR)/Classic Super-Heterodyne Radio

The radio is implemented using hardware components only and cannot be modified except through physical intervention. The first generation classic super heterodyne architecture as seen in Figure 2-25 is the most common configuration used in RF receivers. This configuration includes two down-conversion stages, i.e., the RF received signal is first demodulated to an intermediate frequency (IF) and then converted to baseband signal [95]. The received signal is first filtered by a pre-selection filter and (after amplified by the low-noise amplifier, (LNA)) passes through another filter to reduce the image frequency effects before the first translation from RF to IF. After this stage, the signal is again down converted from IF to baseband, where it is converted to the digital domain for processing.
The advantages of this architecture are:

1. Less overall software complexity with fewer processing blocks at the digital domain, and requiring less complex calibration.

2. Requires slower ADCs.

The disadvantages are:

1. A number of fabrication technologies are used, making full on-chip integration difficult. They are usually designed to a specific channel, preventing the expansion of the receiving band for various modulation formats and occupied formats.

2. The amount of circuitry, low integration level and the difficulty to implement multimode operation generally prevent the use of super heterodyne transmitters in SDR applications.

2.3.6 SDRs v/s Traditional Radios

Table 2-3 presents some comparisons between SDRs and traditional radios. In the recent years, Software Radio and Software Defined Radio [96] technology has promised to revolutionise the communication industry by delivering low-cost, flexible software solutions for communication
Software Defined Radio (SDR) Architecture for Concurrent Multi-Satellite Communications
Mamatha R. Maheshwarappa

Protocols. In this decade, the introduction of FPGA SoC and, most recently, RF programmable transceiver SoC can fulfill the early promise.

Table 2-3 Traditional Radios v/s SDRs

<table>
<thead>
<tr>
<th>Pros:</th>
<th>Cons:</th>
<th>Pros:</th>
<th>Cons:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software defined radios</td>
<td></td>
<td>Traditional radios</td>
<td></td>
</tr>
<tr>
<td>➢ Flexible design: Multi-band/multi-mode</td>
<td>➢ Complexity in software</td>
<td>➢ Limited processing and thus selection of processor/controller/ADC is less critical</td>
<td>➢ Fixed design: Single-band/single-mode</td>
</tr>
<tr>
<td>➢ Software based reconfigurable platform</td>
<td>➢ Vulnerable to software threats</td>
<td>➢ Cheap and readily available</td>
<td>➢ Complexity in hardware</td>
</tr>
<tr>
<td>➢ Upgradable during mission lifetime</td>
<td>➢ Faster FPGAs or DSP processors and Larger bandwidth ADCs are required</td>
<td>➢ Less power consumption</td>
<td>➢ More analogue components</td>
</tr>
<tr>
<td></td>
<td></td>
<td>➢ Power consumption is high</td>
<td>➢ Cross talk between the narrow bands due to aging</td>
</tr>
</tbody>
</table>

The Federal communications Commission (FCC) views SDRs as the result of an evolutionary process from purely hardware based equipment to fully software based equipment [97]. The paper issued by FCC Technical Advisory Council (TAC) cited that the evolution in SDRs has a potential role in the development of a board array of services and applications in telecommunications and information transfer technologies [98]. Advances in both hardware and software technology are making SDRs a reality [99, 100].

Attributes of space SDR architecture include:

1. High power consumption and complexity in software.
2. Requires faster FPGAs and larger bandwidth ADCs/DACs.
3. Post-launch re-programmability to support control, configuration, re-configurability and new application installation.
4. Provides flexibility to support parallel reception from multiple-satellites.
5. Scalability, extendibility, and modular design to support evolution over time.
6. Ability to allow latest application/signal development to support new features and services without hardware upgrades.
7. Affordability to promote commercially available computer software and hardware products/standards.

2.4 SDR Challenges

The process of taking a SDR from initial concept and requirements to reality is fraught with many challenges and difficulties. Although the theoretical research for SDRs is blooming, with many interesting results, hardware implementation and system development are progressing at a slower pace, because of the complexities involved in designing and developing SDRs at different levels - hardware, software and system.

2.4.1 Hardware Challenges

The hardware challenges include the analogue (front end), ADC/DAC and digital domains (back end), particularly in wideband implementation as listed:

2.4.1.1 RF Front End Challenges

A key bottle neck in SDR has always been, and continues to be, the frequency-agile RF front-ends that can easily be coupled with the parts of the SDR that carry out the digital processing – be they pure software systems or a mix of hardware and software. The SDR transceiver should be able to use any available band, adapt to multiple signals from multiple satellites and modulation schemes, switch quickly between links, and communicate with two or more points at a time. It should have a wide dynamic range (so that it has the ability to handle a large interferer), whilst simultaneously receiving the desired signal. Therefore, the RF section needs to be particularly flexible.

A well-designed SDR receiver seeks a balance between the LNA noise factor (NF) (<3dB) and high linearity (C/I > 50dB). C/I (Carrier to Interference) is defined as the ratio of average received modulated carrier power to the average received adjacent co-channel interference power. Communications system components, such as power amplifiers, often exhibit nonlinearity. Due to nonlinear characteristics, the modulation sidebands interact and produce intermodulation distortion. Thus, interference may be created through intermodulation distortion within the transmission system that will degrade bit error rate at the receiver.

Since, SDRs can receive very wide band, it is likely that they are more susceptible to intermodulation distortions. This can be prevented by having narrow digital filters around the
signal of interest. The system needs to achieve a high LNA and mixer linearity, adequate filtering, low Local Oscillator (LO) phase noise and spurious signal generation, and to have a good blocker immunity [103]. At the transmitter, the key requirements are high power amplifier (PA) linearity and low noise, necessary for low adjacent channel leakage power, high PA efficiency and heat removal, and low filter insertion loss, to reduce power consumption.

The commercial use of SDR has been restricted to providing ‘partial software upgradability’ within a given family of wireless standards. This has been due to technological bottlenecks at the RF front end and its inability to be reconfigurable. However, with recent innovations in enabling wideband RF front ends and soft transceivers, SDR can move beyond ‘partial reconfigurability’ to ‘multiprotocol multiband reconfigurability’. This section looks at the innovations that are driving this transition and analyses the critical factors needed to ensure its adaptability to meet the space challenges discussed later in this chapter.

Many innovations at the RF front end and improvements in the baseband processing capabilities are now beginning to bring about a change in the perception of SDR, such as moving the digital domain as close as possible to the antenna. There exists a wide range of options to solve the front-end problems and challenges in a SDR and a representation of these available options can be found in Figure 2-26.

Appendix 1 gives the details of these front-end options that list the following parameters: release date, architecture, RF tuning range, protocols used, block diagram, application and the cost.

![Figure 2-18 Front End Options](image)
2.4.1.2 ADC and DAC Challenges

In an ideal SDR, the RF signal is converted into the digital domain as close to the antenna as possible. In this way, the processing is handled by the digital signal processing. Therefore, the ADC requires higher analogue input bandwidth and sampling rate i.e., double the required signal bandwidth to meet the Nyquist criteria. However, there is a trade-off between the analogue input bandwidth and sampling rate v/s power consumed, as seen in Figure 2-27 and Figure 2-28. Though the power is not a main constraint now for the Ground Station, it will be a major constraint in future space segment implementations.
2.4.1.3 Baseband Challenges

Similarly, the baseband processing has seen different approaches being applied over the past few years. The digital section of wireless terminals is greatly benefited by the rapid advancement of semiconductor technologies. Miniaturisation allows increased complexity, leading to better performance and more integrated functionality of digital circuits. Therefore, digital processors and their driving software can easily cope with the demands of modern multi-mode wireless technologies.

With real-time constraints, the challenge in baseband architecture and design is to:

1. Achieve sufficient computational capacity, processing wide-band high bit rate signals within acceptable size, mass and power constraints of a CubeSat.

2. Adapt to the harsh interference and noise conditions by instantly changing parts of the signal processing through loading different software modules, in order to still maintain adequate bit error rates.

3. Meet software architectural challenges such as parallelism, protocol handling, porting and code structure. Broadly the parallelism can be broken into three types: Instruction Level Parallelism (ILP) \([104]\), Data-Level Parallelism (DLP) \([105]\), and Task-Level Parallelism (TLP) \([106]\). Different SDR platforms exploit different types of parallelism. This research will also focus on improving the parallelisation, thus achieving high performance.

The most common approach for a re-configurable baseband unit is to use DSP processors \([107]\), application specific integrated circuits (ASICs) \([108]\) and FPGAs \([109]\), also known as the firmware approach. Though DSP processors are arguably the original enabling technology for SDR, they are best suited to the less computationally intensive forms of signal processing, rather than very high-speed applications. They are often utilised for involved, off-line processing of data, which has been acquired and undergone initial processing/storage by a different type of device such as an FPGA or an ASIC. ASICs are typically compact designs that utilise low hardware resources and have low power consumption. They can be used to develop components such as microprocessors, memory units or even SoCs. The major drawback of embedded ASICs is that they have a long time-to-market and incur high start-up costs. In addition, the hardware structure cannot be modified after the chips are manufactured.

Both ASICs, and FPGAs have undergone a revolution in recent years, both in terms of performance and cost. Although FPGAs use a similar design approach and implement the same functions as ASICs, they allow the designer greater flexibility, reduced design time and the
Software Defined Radio (SDR) Architecture for Concurrent Multi-Satellite Communications
Mamatha R. Maheshwarappa

possibility to reuse existing solutions in developing new products. Furthermore, due to their inherent parallelism, FPGAs are commonly being used to perform computationally intensive digital signal processing functions, thereby fitting with the requirements of SDR very well. Table 2-4 and Table 2-5 show example state-of-the-art ground and space qualified FPGAs respectively. Not surprisingly, the number of Digital Signal Processing (DSP) blocks and the number of look-up tables (LUTs) in the space-qualified FPGAs are much less than those of the commercial ones.

Table 2-4 List of Terrestrial FPGAs

<table>
<thead>
<tr>
<th>Vendor</th>
<th>Xilinx</th>
<th>Altera</th>
<th>Microsemi</th>
<th>Altera</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device</td>
<td>Virtex 7</td>
<td>Stratix V GT</td>
<td>SF2 (SoC)</td>
<td>Cyclone IV</td>
</tr>
<tr>
<td>Technology</td>
<td>28nm HPL</td>
<td>14 nm 3D Tri-Gate transistor</td>
<td>CRI portfolio</td>
<td>60nm</td>
</tr>
<tr>
<td>Clock frequency (MHz)</td>
<td>1600</td>
<td>500</td>
<td>166</td>
<td>500</td>
</tr>
<tr>
<td>LUT number</td>
<td>1955K</td>
<td>234,720</td>
<td>146,124</td>
<td>114,480</td>
</tr>
<tr>
<td>Transceiver bandwidth (Data rate / number of transceiver channels)</td>
<td>2,784Gb/s (Full Duplex)</td>
<td>28.05 Gbps / 4 12.5 Gbps / 32</td>
<td>16 x 5GBPS</td>
<td>8 x 3.125 Gbps</td>
</tr>
<tr>
<td>DSP block number</td>
<td>3600</td>
<td>512</td>
<td>240</td>
<td>360</td>
</tr>
</tbody>
</table>

Table 2-5 List of Space Graded FPGAs [129]

<table>
<thead>
<tr>
<th>Vendors</th>
<th>Xilinx</th>
<th>Aeroflex</th>
<th>Microsemi</th>
<th>Microsemi</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device</td>
<td>Virtex 5 (FX)</td>
<td>Eclipse</td>
<td>RTG4G150</td>
<td>RTAX-S/SL</td>
</tr>
<tr>
<td>Technology</td>
<td>SRAM-65nm</td>
<td>CMOS Antifuse – 250nm</td>
<td>65 nm</td>
<td>0.15μm</td>
</tr>
<tr>
<td>Clock Frequency (MHz)</td>
<td>550</td>
<td>150</td>
<td>300</td>
<td>125</td>
</tr>
<tr>
<td>LUT number</td>
<td>81,920</td>
<td>4,002</td>
<td>151,824</td>
<td>30k – 500k</td>
</tr>
<tr>
<td>Flip-flop number</td>
<td>81,920</td>
<td>4,002</td>
<td>151,824</td>
<td>30k – 500k</td>
</tr>
<tr>
<td>DSP block number</td>
<td>320</td>
<td>4002</td>
<td>462</td>
<td>120</td>
</tr>
<tr>
<td>Power (W) at 100 MHz</td>
<td>3.1 (typ.), 7.7 (max)</td>
<td>1.8 (typ.), 2.5 (max)</td>
<td>0.49 (typ.), 11.766 (max)</td>
<td>1.13(typ.), 14.47 (max)</td>
</tr>
</tbody>
</table>
Also, one of the observations, as shown in Figure 2-29, is that ground application FPGAs implement nearly 32kLUT per consumed watt, whereas space qualified ones achieve far fewer per consumed watt. Note: they are all below the ground implementation trend line.

![Figure 2-21 Available Space Qualified FPGAs: Size v/s Power Consumption](image)

The performed survey on FPGAs has highlighted a considerable market delay for the space-qualified devices with respect to their commercial counterparts. Such delays can be roughly estimated by assuming that the specific commercial component presents the requested features for the considered application. Therefore, this research mainly focuses on having the flexibility on the ground in order to meet the high power SDR requirements.

### 2.4.2 Software Challenges

Signal processing deals with the analysis, interpretation, and manipulation of signals. Processing of such signals includes filtering, storage and reconstruction, separation of information from noise, compression, and feature extraction. In communication systems, signal processing is mostly performed at OSI layer 1, the physical layer (modulation, equalisation, multiplexing, radio transmissions, etc.) [110]. In SDRs, the major task of signal processing is sensing the spectrum reliably and efficiently in order to detect the unused spectrum and sharing...
it without harmful interference to other users, filtering, prediction and obtaining the data information.

These DSP challenges have been met using various software and firmware approaches over the years. DSP algorithms have long been run on standard computers, as well as on specialised processors, called digital signal processors, using software languages such as C/C++, and on purpose-built hardware, such as application-specific integrated circuit (ASICs). Currently, there are additional technologies used for digital signal processing including more powerful general purpose microprocessors, FPGAs, digital signal controllers (mostly for industrial applications such as motor control), and stream processors where hardware description languages (HDL) such as VHDL and Verilog play a major role.

These languages are often better for applications with tight resources that must be closely managed. They force one to consider and specify even the smallest details, such as memory assignments and threads. A good programmer can use this low-level control to eliminate the overhead in most higher level implementations. At this level, one can take advantage of the target architecture or host operating system properties to achieve greater performance.

However, there has been an invasion of DSP/Control frameworks such as Matlab/Simulink [111] and NI Labview [111, 112] (DSP for Matlab and Labview [113]) in the recent years. In measurement and control applications, programming is just one task of a system designer. Engineers often do not have the time to keep up with or rewrite old software to support the advancements in computing and measurement hardware, operating systems, and so on. They add value by figuring out how to acquire, manipulate, and present real-world data - not by coming up with new ways to handle memory allocations and thread pools. By using Matlab/LabVIEW, one can build on top of tested, supported, and maintained libraries of lower level code.

Table 2-6 shows a comparison of two different implementation approaches: those using custom/low level languages (VHDL [114]) and those using a high level general purpose processor (GPP) based approach using high level languages with many supplied blocks (GNURadio [115]). The “ideal” approach is seen as a compromise between these two solutions.

However, the challenge here is the lack of software support and good documentation. VHDL and GNURadio platforms have readily available blocks that can be directly used. However, they come with massive overhead as the blocks are too generic. Therefore, this research aims at developing custom blocks where necessary to utilise the resources efficiently.
### Table 2-6 Back-end Signal Processing Software Options

<table>
<thead>
<tr>
<th>VHDL</th>
<th>Ideal platform</th>
<th>GNU Radio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cycle accurate</td>
<td>Real time</td>
<td>Soggy soft real time</td>
</tr>
<tr>
<td>Huge design effort up front</td>
<td>Ease of use for demodulator designer</td>
<td>Low design effort assuming block are available</td>
</tr>
<tr>
<td>Long design/debug cycle (years)</td>
<td>Moderate design cycle months</td>
<td>Short design cycle days</td>
</tr>
<tr>
<td>Optimum resource and power utilisation</td>
<td>Good resource and power utilisation</td>
<td>Relies on huge CPU and memory resources</td>
</tr>
<tr>
<td>Very reliable</td>
<td>Good reliability</td>
<td>Moderate reliability</td>
</tr>
</tbody>
</table>

#### 2.4.3 System Challenges

The design and implementation of SDR systems requires the combination of many disparate disciplines including hardware, firmware, software, RF, DSP and functionalities as discussed:

**SDR: High-level and low-level operation**

Some definitions of SDR consider high-level operations which includes modifying signal attributes under the control of software. While others consider its operation at a much lower level and focus on how the physical layer processing is implemented. The first definition describes software configured radio and, while it encompasses SDR, arguably applies to almost all modern radio equipment. However, the latter definition is much more prescriptive, clearly describing what is meant by the term SDR.

**SDR: Field re-programmability**

Field re-programmability is another key feature of a SDR, software updates may be performed ‘over-the-air’ and without any intervention of the user.

**SDR: Dynamic Changes in Functionality**

SDR is sometimes discussed in the context of a radio that can modify its operation dynamically to deliver maximum perceived performance to the user and optimal spectral efficiency. In particular, SDR systems have the potential to offer reconfigurable, multi-mode operation capabilities.
Software Defined Radio (SDR) Architecture for Concurrent Multi-Satellite Communications
Mamatha R. Maheshwarappa

The SDR discipline encourages the separation of signal from the underlying platform, typically at the expense of increased system complexity. SoC, multicore and multiprocessor SoC (MPSoC) based platforms are best suited for such applications with increased system-level efficiency. However, the challenge here is to efficiently utilise the resources and optimise the design. In order to resolve this challenge Xilinx has introduced SDSoC, a new C/C++ development environment.

2.4.3.1 Software Defined System on Chip (SDSoC)

The SDSoC development environment provides an Application Specific Standard Product (ASSP) – a C/C++ like programming experience – to improve productivity for application development, system architecture definition and platform creation [116]. SDSoC delivers system level profiling, automated software acceleration in programmable logic, system connectivity generation, and libraries to speed programming (see Figure 2-30). It has the ability to create a high-level representation of the system and to decide which portions are to be implemented in software and which are to be realised in hardware. Furthermore, to have this solution presented in such a way that one can use it without having to call in their hardware counterparts.
The enabling technology is High Level Synthesis (HLS), which can take a high-level representation in C/C++ and compile/synthesize it into an equivalent RTL (register transfer level) representation that can subsequently be used by traditional synthesis technology to generate the ultimate hardware realisation. SDSoC is the manifestation of the intelligence to manage SoC resources that would benefit SDR applications to include different standards in future. The SDSoC environment is not used in this work as it was only released recently (in 2015) when the majority of the implementation work had been completed but it is noted that it will definitely be useful in the future for profiling and automated software acceleration.

2.4.4 Space Challenges

In the last decade, SDRs were mainly seen in larger ESA or NASA space missions, used as a payload or as a test-bed [26, 27] (as depicted in Figure 2-31) and were less competitive when compared to the ones used on the ground. However, in the recent years there have been a few SDRs used in small satellites applications as listed in Appendix 2.
Along with these SDRs which have flown, or are ready to fly, there are other SDRs that are under development for satellite applications: e.g. AstroSDR [32], NanoDock SDR [31], GAMALINK [117] and STI-PRX-01 [118]. The key point here is none of the existing space/terrestrial SDRs have demonstrated concurrent multi-satellite reception.

Electra is the first programmable software radio that has been developed exclusively for space missions [26]. It was primarily targeted at the Mars Exploration Missions, which include various micro-spacecraft short-range communication links, such as orbit–lander, orbiter–rover, orbiter–microprobe, orbiter–balloon, ST-4 (orbiter to/from Lander link), ST-3 (inter-spacecraft links) and multiple proposed discovery missions (e.g., balloons, gliders, probes). The radio is designed to operate over a range of data rates from 1 kbps - 4 Mbps (compare this to terrestrial ones capable of 50 – 100 Mbps), and the frequency uncertainties accommodate up to 20 kHz whilst the terrestrial ones can typically accommodate up to 28 MHz. Further details such as the transceiver block diagram, specifications and frequencies can be found in Appendix 2.

The SCaN Test-bed consists of reconfigurable and reprogrammable SDR transceivers/transponders operating at S-band, Ka-band and L-band, along with the required RF/antenna systems necessary for communications. The three SDRs will provide S-band duplex RF links directly with the ground, S-band duplex RF links with the TDRSS (Tracking and Data Relay Satellite System), Ka-band duplex with TDRSS and L-band receive-only with the GPSS (Global Positioning Satellite System) [96]. Further details such as the transceiver block diagram and specifications can be found in Appendix 2.
Space presents a unique engineering environment with a new set of problems to overcome. But how does this affect us when putting a SDR in space for inter-satellite or ground to satellite links?

SDRs for Distributed Satellite Systems (DSS) will provide flexibility that will allow deployed satellite communication transceivers to be software upgraded according to advances in algorithms and communication standard. However, SDRs for space applications pose many challenges, some of them causing SDR to evolve slower than anticipated:

### 2.4.4.1 Radiation effects

Although radiation is not the primary concern in the research, it will be a concern for space implementation. Many COTS devices have significant Single Event Upset (SEU) tolerance issues in space [74]. An SEU occurs when a charged sub-atomic particle causes a (reversible) state change in an electronic component. The SEU rate for LEO orbits is of the order of \(1 \times 10^{-5}\) SEU/bit/day for typical COTS CMOS memory technologies. Well known and proven SEU mitigation techniques include: Triple Modular Redundancy (TMR) – which corrects complete erasure by majority voting, Error Detection and Correction (EDAC) codes include parity, hamming codes, cyclic codes and Double Error Detection, Double Error Correction, and memory scrubbing [119].

### 2.4.4.2 Frequency uncertainties with Doppler tracking capabilities

Two Line Elements (TLEs) is a data format encoding a list of orbital elements of an object around Earth for a given point in time. TLEs are often wrong at the beginning of the launch as NORAD (the North American Aerospace Defense Command – now the Continental Air Defense Command (CONAD)) may take a few days to correctly identify any new object in space. This may extend to a few weeks when the objects are small (CubeSats) and/or many (30+ satellites are deployed from a single launch [83, 85]). It is often necessary to establish communication with a satellite soon after the launch. If the satellite's transmitter is designed to be turned on by ground command, only above a desired ground station (which is common in CubeSat missions, due to their typical lack of power), then knowing where to point the ground antenna and when to send the “turn on” command is critical.

Due to aging, temperature and Doppler effects the frequencies might shift/change from the allotted centre frequency. Frequencies might vary from few kHz to MHz depending on the operating band.
2.4.4.3 Signal fading

The relative velocity between satellites in different orbits varies with time. The communication channel of in-space transmission is mainly characterised by free-space loss and thermal noise of the electronics, presumed to be Additive White Gaussian Noise (AWGN) [39] and signal fading due to mobility, antenna pointing, phase propagation delay, attenuation and refraction due to varying plasma density causing multi-path effects. Also, the signals pass through the ionosphere with effects such as scintillation, fading and Faraday rotation [76]. Between 300 MHz and 3 GHz, in which S and L band lie, severe disruptions are possible during a solar storm [120] which could affect satellite communications.

2.4.4.4 Re-configurability time

Satellites in LEO with orbital periods of ~ 98 minutes revolve ~ 14.7 times/day. So, on an average they visit a particular mid-latitude location on Earth 5-6 times/day for about 5-15 minutes each, depending on the elevation. Re-configurability includes change/replacement of software modules on-board the satellite after launch which is one of the main attributes of a SDR. It is therefore crucial to utilise this time efficiently to carry out different tasks such as downloading beacon/telemetry and payload data, tele-commanding the satellite and reconfiguring the software modules when required.

2.4.4.5 Interference with Adjacent Channels

The architecture proposed needs to adapt transmission and receiver parameters to avoid interference and maximise spectral efficiency. To avoid causing interference, numerous techniques can be used and combined such as frequency tuning [103] (adaptive frequency hopping, dynamic frequency selection and RF band switching), channel aggregation [121], time multiplexing [122], power control [123] and modulation and coding for QoS adaptability [124]. SDR will be based on strong cross layer interactions. For example, the SDR management involves intelligent use of spectrum based on anticipating the demand for spectrum by different satellites and the number of satellites in view at a given point of time.

2.5 Summary

The first section of this chapter has discussed the growing trend in multi-satellite missions by reviewing and classifying the multi-satellite mission requirements which is one of the research objectives. Primarily, the current issues with increasing number of CubeSats with statistical analysis of different modulation schemes and data rates are reviewed. It is evident that there is
an increasing strain in the amateur frequency bands and the spectrum allocation due to future missions such as QB50 and OneWeb. There is an indisputable need for a generic communication platform to support multiple signals from multiple satellites with varied standards simultaneously. The multi-satellite problem is illustrated using STK tool where 4 existing constellations such as FLOCK-1, Yaogan, Shijian and Formosat were considered to understand the overlap of satellites’ communication window.

The next two sections provide an overview of MIMO and different tiers of radios starting from traditional super-heterodyne architectures (Tier 0) to USR (Tier 4) along with a trade-off in terms of power, complexity, and heritage that can fit into the requirements of parallel reception with varied modulation techniques, data rates and frequency bands. SDR is a radio platform where the signal processing path is performed, all or in part, using programmable devices such as FPGAs, DSP processors and general purpose processors. The trend is not just in SDR, but also in FPGAs and reprogrammable RF frontend configurations. As such, the attributes of the transmitter/receiver signal can be changed in software and without any physical changes to the hardware. Ultimately, as more and more of the signal path is digitised, SDRs offer superior tuning capabilities more than traditional radio architectures.

The implication of MIMO for SDR implementation of wireless communication systems can be seen as a complementary technology – as a system’s engineering tool that can used where appropriate. MIMO can benefit significantly from adaption to the channel, and adaption is, of course, a feature of SDR implementations.

The next section of this chapter addressed the front-, back-end, systems and space challenges and explored current state-of-the-art embedded devices that are relevant for this research.

Traditionally, SDRs did not include a configurable front-end but high-performance components can now allow bandwidth upto 12 GHz. Various ADC/DAC are reviewed to show the trade-off between input analogue bandwidth, sampling rate and power. Though the review shows ADC/DAC with higher bandwidth and sampling rate, they come at the expense of power and CPU consumption. Similarly, at the back end, various processing devices such as DSP processors, ASICs and FPGAs are reviewed along with space graded and terrestrial FPGAs. The recent technologies such as SDSoC could be used in future to accelerate the performance of SDRs.

In conclusion, this chapter has focused on investigating state-of-the-art terrestrial SDR technologies to understand the requirements, algorithmic capabilities and hardware limitations in order to adapt for concurrent multi-satellite applications which is part of the research objectives.
3 SDR PLATFORM IMPLEMENTATION & REQUIREMENT ANALYSIS

SDRs have evolved from a conceptual solution for enabling multiple radio applications (functions) to run on a single platform, to a practical solution for which, today, many products are commercially available. Section 3.1 defines the drivers and requirements of this research followed by Section 3.2 containing an overview on the design approach and the recent technologies that can be used in this approach. Section 3.3 introduces novel concepts with recent test-bed options for improving the flexibility of SDRs in space applications. Detailed design, implementation and validation methods of the proposed new architecture are discussed in Section 3.4. Finally, the requirement analysis is carried out for encoding and decoding of the FUNcube-1 telemetry signal (based on binary phase shift keying (BPSK) modulation and Convolutional Encoding Viterbi Decoding – CEVD).

3.1 Drivers and Requirements

To better support the rapid increase in CubeSat numbers (and CubeSat constellations), the problem of multi-channel, multi-rate, multi-signal, transmission and reception of TT&C links (and future inter-CubeSat data links) is considered. At present many low cost SDR solutions for CubeSats have their heritage in Amateur Radio and Amateur Satellite (AMSAT) communities. By nature, these solutions have evolved from the application of SDRs to process audio band signals,
Software Defined Radio (SDR) Architecture for Concurrent Multi-Satellite Communications
Mamatha R. Maheshwarappa

using well established amateur radio transceivers to perform low noise amplification, band selection, tuning, down-conversion, and baseband filtering.

In the foremost years, the baseband digital signal processing used custom made modems and was often limited to simple Forward Error Correction (FEC) [125]-less signals. These systems can be considered as single threads of processing that must be duplicated entirely to support concurrent transmission and/or reception to multiple satellites in a single pass or field of view.

The advent of many hobby and semi-professional SDR transceivers such as HackRF [126]/BladeRF [127]/ LimeSDR [128] and the Ettus USRP series [129] can be seen as good hardware enablers for this problem space. However, in their default and or factory defined configurations they only support a single transmit and receive path. Furthermore, in normal operation, they rely on a high performance general purpose computer (attached by USB or Ethernet) to carry out the bulk of the modem processing.

When using such high performance computers to realise the modem processing, the trend is to use and become reliant on vast signal implementation frameworks such as Matlab/Simulink, NI LabView and GNURadio. In doing so, the required power and memory consumption spirals upwards, and the ability to analyse the underlying reliability/assurance spirals downwards.

This direction is incompatible with many CubeSats where mass (up to 1.33 kg), power (2W with body mounted solar cells) and volume (1 litre) for 10 cm cube (1U) [226] are at a premium. It is perhaps also incompatible in a ground station, operating many satellites, where the reliability of embedded system must be expected.

Another trend that has emerged in recent years is the rapid rise of System on Chip (SoC) technologies to address many RF and/or DSP applications. It is now possible to realise a compact and capable SDR, based on a single RF SoC and Single BaseBand SoC (incorporating FPGA, Vector DSP and GPP elements). SoC technology is an excellent platform enabler for our problem space and opens up the possibility of a common underlying design that could be used for ground and space applications. It is important to note that at present, to master SoC technology requires a steep learning curve.

In firmware engineering, it is common to use low level hardware description languages such as VHDL or Verilog. Such languages and associated teaching impose a much stricter discipline on modem design and implementation of various digital signal processing algorithms. However, with such a low level approach it is not always easy to manage increasingly complex system designs, to the point that many vendors are now advocating the use of C/C++ for all phases of design capture. On one side of the argument, it can be seen that application of low level design
techniques and optimised platform electronics are compatible with the mass (up to 1.33 kg), power (up to 2W with body mounted solar cells), volume (1 litre) (for a 10 cm cube (1U)) constraints imposed by CubeSat platform [226]. However, a modem design that takes many years to complete, is not compatible with time scale constraints of most CubeSat programmes.

On the other side of the argument, it can be seen that high level/model based design techniques and general purpose computing are compatible with time scale constraints imposed, but are far less likely to meet requirements for mass, power and volume.

In this work, we focus on a compromise design and implementation methodology, targeted at our problem space, that best balances the use of established and emerging design capture techniques, employs the latest RF and base-band (BB) system on chip technologies and which partitions functionality across FPGA, DSP processor and GPP processing elements in manner that reflects a compromise between power consumption and minimising design time. In the process of this work, the basis for a light-weight framework to speed the implementation of future CubeSat signals on SoC technologies is established.

High level of integration and small size are key objectives in SDRs. In order to achieve these objectives it is feasible to move most of the data processing to digital domain. This research will aim at implementing new algorithms/methodologies on SoC platform which include:

1. A baseline algorithm to perform concurrent reception of multiple signals from multiple satellite scenarios (as discussed in Section 2.1.1), capable of performing signal processing such as decimation, down conversion and decoding. It will be implemented in C/C++, and its implementation will also provide information on its computational requirements, time/CPU, memory, I/O and data rate resources.

2. One of the main challenges is the coexistence of several standards in the spectrum. In this scenario of receiving from different satellites, the receiver must be able to differentiate the signals coming in from different channels with varied signal standards.

3. This research also aims to combine the open-source software and SDR platform hardware to create an embedded transceiver to fully understand the computing needs such as CPU and time.

### 3.2 Design Approach

A detailed Zero-IF architecture for multi-mode applications is proposed in Figure 3-1.

VHF/UHF bands are selected for uplink/downlink as currently ~80% of the CubeSats use VHF/UHF frequencies for ground-links (Figure 2-11); there are more ground facilities/ amateur
communities to communicate in these bands across the world which would help to increase the communication window and it is easier and cheaper to establish a VHF/UHF ground station when compared to other bands.

SatNOGS is made possible by the “RTL DONGLE” (no transmit) and the “FunCubeDongle” (FCD) (narrow band), GNURadio and 3D printed antenna models. Though it has made great inroads to this area, it is perhaps limited by the power consumption required by conventional laptop or desktop computers, and inefficiency of GNURadio. In the recent years, S-band is becoming more popular in the CubeSat community for Ground-to-Earth communications and ISLs in order to achieve higher data rates. VHF/UHF and S-band require separate antennas and thus separate RF Front end.

The focus of this research is to move the digital domain as close as possible to the antenna towards achieving ideal SDR scenario as discussed in Section 2.3. As there are technical limitations such as providing large front-end frequency bandwidth, high ADC sampling rate and baseband processing, the closest we can get in the multi-tiered definitions is SDR. Being conscious about the recent technology developments and their abilities to perform multiple functions, the architecture is grouped into different sections viz: front-end and back-end, including different processing blocks [130, 131]:

### 3.2.1 Band Pass Filter (BPF)

A BPF selects a required band of frequencies from the input signal. BPFs can be characterised by Q-factor, which is the inverse of the fractional bandwidth. There are narrow-band and wide-band filters [132]. 90% of the CubeSat signals are narrow band therefore the BPFs should have a high-Q factor, as given by Eq. 3.1:

$$Q = \frac{f_0}{f_2 - f_1} \quad \text{Eq. 3.1}$$

Where $f_0$ is centre frequency; $f_2$ and $f_1$ are the cut-off frequencies. The multi-mode analogue mode of a SDR would need reconfigurable filters in order to match the requirements of diverse satellite standards.
Software Defined Radio (SDR) Architecture for Concurrent Multi-Satellite Communications
Mamatha R. Maheshwarappa

Figure 3-1 Detailed Transceiver
3.2.2 Low Noise Amplifier (LNA)

A LNA is an electronic amplifier that amplifies a very low-power signal without significantly degrading its signal to noise ratio. The LNA is an amplifier that is constrained to minimally affect the RF signal in terms of non-linear distortion and noise corruption. Designers minimise the additional noise by considering trade-offs that include impedance matching, choosing the amplifier technology and selecting low-noise biasing conditions [133]. The requirements of a wideband, digital controlled, low noise, high linearity and low power consumption make the LNA design for SDR receiver challenging [134]. For wideband LNA design, input matching and low noise figure are the two critical considerations.

3.2.3 Duplexer

A duplexer (more generally a diplexer) provides the means for simultaneous operation having separate transmit (Tx) and receive (Rx) frequencies when using a common antenna [135] which is essential for concurrent multi-satellite communications. For instance, the STRaND-1 CubeSat has a UHF Rx and VHF Tx and in such a case the “duplexer” provides sufficient filtering to prevent both transmitter carrier power and wide band noise from desensitizing the associated receiver. Through use of the correct branch cable lengths and careful loop coupling adjustments, the duplexer can be tuned for a wider bandwidth response to accommodate multiple frequency transmitters and receivers suitable for SDR applications.

3.2.4 I and Q signal

“I” and “Q” are the In-phase and Quadrature components of a signal. A signal \( x(t) \) can be represented as a vector with magnitude and phase angle as in Eq 3.2 [159]:

\[
x(t) = I(t) + jQ(t)
\]

Eq. 3.2

The use of I and Q channels allows for processing of signals near DC or zero frequency and reduces the sampling rate requirements of the ADC/DACs.

3.2.5 Local Oscillator (LO)

A LO is a device that generates a sinusoidal signal with a frequency such that the receiver is able to generate the correct resulting frequency, or intermediate frequency (IF), for further amplification and conversion into, for example, audio detection. The frequency of the LO is mixed with the incoming signal from the antenna to produce an IF signal. Depending on the received frequency there may be more than one LO and more than one IF stages especially in traditional super heterodyne architectures [136, 137]. In this application, the LO is implemented in the digital part of the system, DDS_Compiler (Refer to section 5.3).
3.2.6 Low Pass Filter (LPF)

The LPF allows low frequency signals from 0 Hz to its cut-off frequency, \( f_c \), to pass while blocking those any higher. The resistance and the capacitance define the time constant \( (T = RC) \) which then determines the cut-off frequency, i.e. the frequency at which the filter attenuates the signal [138] (see Eq. 3.3).

\[
f_c = \frac{1}{2\pi RC}
\]  

Eq. 3.3

In this application, a LPF is used as an anti-aliasing filter before the signal sampler to restrict the bandwidth of a signal over the band of interest.

3.2.7 Variable Gain Amplifier (VGA)

VGAs are signal-conditioning amplifiers with electronically settable voltage gain. The analogue VGAs are linearly controllable in terms of both dB and magnitude and are used mainly in traditional super-heterodyne architectures, while digital control offers similar control, but in discrete steps. VGAs affect the shape of the passband response as well as the cut-off frequency of the LPF, particularly when the bandwidth of the VGA is approximately the same as the LPF cut-off frequency. This is because the gain of VGA starts to roll-off at frequencies lower than the -3dB bandwidth of the VGA. Therefore, in order to have an overall flat response of the baseband, the bandwidth of the VGA should be much higher than the cut-off frequency of the LPF [139].

3.2.8 Analogue-to-Digital (ADC) & Digital to Analogue-to-Converters (DAC)

An ideal SDR would conceptually be able to switch between applications in milliseconds, therefore the data converters would have to be directly on the antenna which is not really a practical solution as the ADC would need extremely high bandwidth to operate at RF in order to meet the Nyquist criteria. Therefore, in reality some analogue front end (with frequency conversion) must be used before the ADC in the receive path and after the DAC in the transmit path that does the appropriate frequency translation. Placing the data converters as close to the antenna as possible provides the most flexibility but must be traded off against performance limitations [140].

3.2.9 Baseband Processing

Following ADC sampling, signals that are the integral multiples of the sampling frequency will alias on top of the desired channel at DC. Those channels that are not at exact harmonics of the sampling frequency will also alias in the Nyquist band but away from the desired channel. Thus, a strong attenuation is needed at multiples of the sampling frequency while the attenuation
away from it is more relaxed. In narrowband receivers, anti-aliasing and anti-blocking are done by RF preselect filters, LNA tuned load and baseband filters. The preselect filter is a discrete off-chip filter whose centre frequency and pass-band are fixed. Therefore, the preselect filters cannot be used in a SDR which covers a wide range of frequencies. A tuneable filter such as Micro Electro Mechanical Systems (MEMS) filter could be suitable for SDRs in the future.

Removing the preselect filter exposes the RF front-end to the whole range of incoming signals and imposes tough linearity requirements which needs to be moved to baseband/DSP filters [141]. The remaining tasks such as modulation/demodulation, encoding/decoding are carried out using FPGA/ASICS/CPUs/and graphics processing units (GPUs), which are addressed in more detail in Section 2.4.1.3.

### 3.3 SDR Hardware Platform Options

The aim of this section is to give the reader an overview of possible options towards achieving a SDR as discussed in Section 2.3.3 with strict requirements such as functionality, cost, volume and power consumption. The trade-off criteria are: transmit and receive capabilities on the same chip, FPGA resources, high throughput, well-documented reference design/software support by the hardware supplier, ease of use and ease in porting the software tools on to the chosen test-bed. The front-/back-end blocks discussed so far can be dealt with independently with recent technologies as seen in Figure 3-2 (more details can be found in Appendix 3 and 4).

![Figure 3-2 Recent Technologies for Front and Back End](image-url)
The software development was carried out on different combinations of front-/back-end technologies in order to choose the most suitable platform. The initial work was carried out on a smaller less powerful FPGA, the Microsemi Smart Fusion 2, using the SF2-STARTER-KIT_ES development board [142] and FunCube Dongle (FCD) [143] (Details discussed in Appendix 5).

FCDs have good RF performance but are narrow band (96-192 KSPS). The hugely successful RTL dongles have lower performance but are wider band (DAB and DVH-T sample rates), and more recently, the AirSpy series offer better performance [144], 12-bit conversion and sample rates up to 10 MSPS. At present, though most support a good tuning range (VHF, UHF and L-band), the compact dongles suffer from being Rx only.

In many ways the growth in “Dongles” has fuelled the growth in more complex open hardware crowd-funded SDR designs such as HackRF and BladeRF. Most recently, a new HF only, Rx only, design called KiwiSDR [145], supports 4 concurrent narrow band channels and a single wideband channel for producing “waterfall” plots.

Similar COTS SDRs that can be used as a platform towards achieving the objectives of this research are shown in Figure 3-3 and their specifications can be found in Appendix 4.

Relying on an external high performance (high power) computer, the Ettus E310 (Zynq + AD9361) is very close to what we need, but it is too expensive. The PicoZed SDR [146], and Matchstiq [147] would not be suitable for the space environment in their present form. The Nuand BladeRF was too primitive in terms of software development when this work was undertaken and included only one transmit and receive channel.

![Figure 3-3 COTS SDRs – (a) EPIQ Matchstiq [60] (b) Nuand BladeRF [61] (c) ETTUS USRP E310](image)

3.3.1 Zedboard and MyriadRF

To overcome the limitations of the previous hardware platform (SF2+FCD) such as limited FPGA resources, no floating point/vector instructions and only Rx functionality, a new platform
was considered (Zedboard [148] along with Lime Microsystems’ Zipper and MyriadRF boards [128]) as seen in Figure 3-4.

Floating point DSP processors yield greater precision than the fixed point, increases the dynamic range available for the application and ensures greater accuracy in the end result. Also, the latter has an advantage of high throughput and a direct interface which supports higher data rates and transmit functionality using the MyriadRF.

![Figure 3-4 Interface of Zedboard and Zipper Board](image)

### 3.3.1.1 Zedboard

Zedboard is an evaluation and development board based on the Xilinx Zynq-7020 extensible processing platform, combining a dual Cortex-A9 Processing System (PS) with Programmable Logic (PL) cells, the Zynq 7020 [149]. Modern COTS processors provide the utmost in performance and energy efficiency, but are susceptible to ionizing radiation in space, whereas RadHard processors are virtually immune to this radiation but are more expensive, larger, less energy-efficient, and generations behind in terms of speed and functionality. Using COTS devices for critical data processing, supported by simpler RadHard devices for monitoring and management of the COTS devices, and augmented with novel uses of fault-tolerant hardware, software, information, and networking within and between the COTS devices, the resulting system can maximise performance and reliability (called performability) while minimizing energy consumption and cost.
One such technology is being developed by Centre for High-Performance & Reconfigurable Computing (CHREC) at the University of Florida and at Brigham Young University working closely with NASA using the Zynq 7020 [150]. This hybrid space computer features an innovative combination of three technologies: COTS devices, radiation hardened (RadHard) devices, and fault tolerant computing [151]. This is to enable a new class of future Goddard Space Flight Centre (GSFC) missions by developing technology for small spacecraft architectures, mission concepts, component subsystem hardware, and development methods.

The fusion of on-board peripherals and expansion capabilities as seen in Figure 3-5 makes it an ideal platform for SDR applications as it can be modified to fit into other applications without changing the hardware.

![Figure 3-5 Zedboard System Design](image)

This configuration was planned to implement a division of software functions in both the dual-core ARM processors and associated FPGA fabric. A system diagram was generated as an initial step towards implementation using Xilinx Processing System (XPS) as seen in Figure 3-6.
The processing system (2 x Cortex – A9) controls the signal and data flow of the blocks over the Advanced eXtensible Interface 4 (AXI4). The distribution of the functions such as up/down conversion, decimation and interpolation between the FPGA fabric and dual-processor is based on performance tests of blocks on Zedboard. The performance test would be repeated for other blocks in the future to estimate where bottlenecks exist. This helps in optimising the usage of the on-board resources available and thus allowing multiple threads to be implemented on the same hardware, which is one of the requirements for concurrent multi-satellite reception.

![Figure 3-6 Xilinx XPS System Diagram](image)

### 3.3.1.2 Myriad RF

On the RF programmable transceiver SoC, initial evaluation took place using the Lime Micro Myriad RF containing the LMS6002D RF SoC [152]. More recent development has taken place using the Analog Devices’ AD-FMCOMMS3-EBZ containing the newer AD9361 RF SoC [153].

The LMS6002 was the first commercial widely available RF SoC transceiver which supported both transmit and receive. However, the early support did not include adequate development tools, software cores, drivers, libraries and worked examples. This was a potential front-end but had reduced supported on Zedboard at the time this work was undertaken. More recently, the LMS7002 has more sophisticated baseband DSP and on-chip DDC/DUC blocks that include digital NCO tuning, which is not found on AD9361. The LMS7002M has a continuous operation spectrum of 100 kHz to 3.8 GHz, and is software configured up to 120 MHz RF bandwidth. It integrates 12-bit ADCs and DACs, LNAs, filters and mixers to provide two transmit and receive paths, enabling 2x2 MIMO operation.
3.3.2 Zedboard and AD-FMCOMMS3-EBZ

A reconfigurable channel or band select filter is one of the key elements of multi-mode SDR as discussed in Section 3.1. The next step in the evolution process is to implement fully reconfigurable filters that can seamlessly adapt to the spectral characteristics of a wide variety of communication standards while maintaining a relatively low hardware overhead, low power consumption and high cost efficiency. This leads to the idea of using Analog Devices’ AD-FMCOMMS3-EBZ for implementing the fully reconfigurable filters. The overview of the AD-FMCOMMS3-EBZ can be seen in Figure 3-7 which consists of dual Rx and Tx ports.

![Figure 3-7 AD_FMCOMMS3-EBZ Overview [173]](image)

It is evident from the literature that the flexibility of the original SDR concept [154] comes at the price of excessive demand for computational capacity, power and resources. Even compromise approaches, usually summarised under the term of SDR, require more Million Instructions Per Second (MIPS) than a mono-processor [155]. Therefore, a novel architecture is proposed based on the multi-core signal processing system as seen in Figure 3-8.

This architecture consists of a Base-Band (BB) SoC paired with Radio Frequency (RF) SoC. The BB SoC contains FPGA fabric and ARM dual-core Cortex A9 processor. As discussed earlier the Avnet Zedboard containing the Xilinx Zynq 7020 FPGA SoC was chosen due to its low-cost and well supported back-end signal processing functionalities along with AD-FMCOMMS3-EBZ. The two boards seen in Figure 3-9 (and constituent SoCs) communicate via conventional parallel I/O under DMA for high speed sampled data (up to ~123 complex MSPS) and Serial Peripheral Interface (SPI)
The partly filtered and partly decimated samples received from the RF SoC are passed to the FPGA fabric (within the BB SoC) for further processing. The samples received over SPI are stored in an internal SRAM, dual A9 processor cores are connected to the FPGA fabric by high speed SoC Advanced eXtensible Interface (AXI) bus.

**Figure 3-8 Multi-Core Architecture**

**Figure 3-9 AD-FMCOMMS3-EBZ and Zedboard [53]**
Table 3-1 gives the trade-off among three platforms discussed above:

**Table 3-1 Parametric Comparison of Test-beds**

<table>
<thead>
<tr>
<th></th>
<th>SF2 + FCD</th>
<th>(Zedboard + MyriadRF)</th>
<th>(Zedboard + AD-FMCOMMS3-EBZ)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CPU</strong></td>
<td>ARM Cortex M3 – 100 MHz (2/5)</td>
<td>Dual ARM Cortex – A9  – 800 MHz (4/5)</td>
<td>Dual ARM Cortex – A9  – 800 MHz (4/5)</td>
</tr>
<tr>
<td><strong>FPGA Details</strong></td>
<td>DSP blocks: None</td>
<td>DSP blocks: 220</td>
<td>DSP blocks: 220</td>
</tr>
<tr>
<td></td>
<td>LUTs: 4</td>
<td>LUTs: 53200</td>
<td>LUTs: 53200</td>
</tr>
<tr>
<td></td>
<td>SoC: SF2 (2/5)</td>
<td>SoC: Zynq (5/5)</td>
<td>SoC: Zynq (5/5)</td>
</tr>
<tr>
<td><strong>Rx RF SoC</strong></td>
<td>Elonics e4000 – limited bandwidth (2/5)</td>
<td>LMS6002D – single Tx port (3/5)</td>
<td>AD9361 – Dual Rx Ports (5/5)</td>
</tr>
<tr>
<td><strong>Tx RF SoC</strong></td>
<td>New board design needed (0/5)</td>
<td>LMS6002D - single Tx port (3/5)</td>
<td>AD9361 – Dual Tx ports (5/5)</td>
</tr>
<tr>
<td><strong>Reference Design</strong></td>
<td>None – 0/5</td>
<td>Very primitive – 1/5</td>
<td>Yes – 4/5</td>
</tr>
<tr>
<td><strong>Ease of Use</strong></td>
<td>2/5</td>
<td>3/5</td>
<td>4/5</td>
</tr>
<tr>
<td><strong>Ease of Porting</strong></td>
<td>1/5</td>
<td>2/5</td>
<td>3/5</td>
</tr>
<tr>
<td><strong>Total Score</strong></td>
<td>9</td>
<td>21</td>
<td>30</td>
</tr>
</tbody>
</table>

It is evident that the Zedboard + AD-FMCOMMS3-EBZ stands out with better processor, bigger FPGA fabric, good tool support and reference design as a starting point. Both the MyriadRF and the AD-FMCOMMS3-EBZ have transmit RF SoC while there is a need for new Tx design in SF2+FCD combination. Based on all these metrics, the Zedboard + AD-FMCOMMS3-EBZ was chosen for further developments. Also, this platform supports: 2x2 MIMO which is useful to demonstrate concurrent multi-satellite reception and cheaper when compared to other options with similar configuration as seen in Appendix 4.

The next section focuses on implementation, testing, debugging, decoding and validation of the proposed architecture on the chosen hardware platform.
3.4 Architecture Implementation and Validation

As a first step towards validating the architecture, a simple receiver demodulator decoder for CubeSat beacon telemetry was implemented.

The FUNcube-1 (AO-73) CubeSat was chosen as it provides a good starting point for our work because its telemetry beacon is documented and addressed by number of Open Source Software (OSS) demodulator decoder implementations [156].

FUNcube-1 (AO73) is a complete educational 1U CubeSat with the goal of enthusing and educating young people about radio, space, physics and electronics. FUNcube-1, now registered as a Dutch spacecraft, was successfully launched from Russia on a DNEPR rocket on Nov 21st 2013 at 07:11:29 UTC. More than 900 stations, including many at schools and colleges around the world, have received and decoded the telemetry [46].

It is important to note that the reference design chosen is implemented and validated on the chosen testbed (Zedboard and AD-FMCOMMS3-EBZ) for the first time.

<table>
<thead>
<tr>
<th>Frame Formats</th>
<th>AX25, HDLC, DVB-S2, Custom</th>
</tr>
</thead>
<tbody>
<tr>
<td>Excess Bandwidth</td>
<td>0.2 – 1.0</td>
</tr>
<tr>
<td>Modulation Type</td>
<td>FSK, BPSK, AFSK, GMSK, MSK, OQPSK</td>
</tr>
<tr>
<td>Modulation Order</td>
<td>2, 4, 6, 8, 16, 32, 64</td>
</tr>
<tr>
<td>FEC Coding</td>
<td>None, Convolution, Reed-Solomon, Turbo, Viterbi</td>
</tr>
<tr>
<td>Unique words</td>
<td>None, pre-defined word length</td>
</tr>
<tr>
<td>Interleaving</td>
<td>Bit-interleaving algorithmic (deterministic)</td>
</tr>
<tr>
<td>Symbol Rate</td>
<td>1200, 2400, 4800, 9600, 19200</td>
</tr>
</tbody>
</table>

Table 3-2 shows the signal diversity common among CubeSats, the underlined (and marked in red) attributes are those which are implemented in this reference design and others could be implemented in order to make the chosen SDR platform a generic platform that can support any CubeSat.

As a first step towards validating the re-configurability of the design, the symbol rates were changed dynamically every 5s and these changes were recorded on SDR sharp [157](discussed
Software Defined Radio (SDR) Architecture for Concurrent Multi-Satellite Communications
Mamatha R. Maheshwarappa

in Section 4.2.1). This reference modulator/demodulator is based on BPSK modulation comprising of two Reed Solomon (160, 128) encoders, the scrambler, the convolutional encoder and the interleave blocks [158].

3.4.1 Frame Structure

Figure 3-10 shows the FUNcube-1 telemetry (TLM) frame – 5s which is 6000 symbols at 1200 sps: 5200 symbols of encoded data and 800 Overhead Symbols, which are composed of a 32 symbol CCSDS defined [158] unique word with the remaining 768 set to 1's. Once differentially encoded, as part of the DBPSK modulation process, the stream of alternating 101010’s produce a CW carrier and 600 Hz sub-carriers that produce an audible tone for 0.7s making the data transmission 4.3s.

The use of DBPSK modulation and differential detection allows simple robust low cost reception, at the expense of 2.2 dB loss in additive white Gaussian noise (AWGN) performance.

The occupied bandwidth is given by Eq. 3-1 where \( R_s \) is the symbol rate and \( r \) is the excess bandwidth or roll-off factor (\( r = 0.5 \) for FUNcube-1).

\[
BW = R_s (1+r) \quad \text{Eq. 3-1}
\]

\[
= 1200 (1+0.5)
\]

\[
= 1800Hz
\]

3.4.2 Modulation

Binary Phase Shift Keying (BPSK), as used by the AO-40 telemetry beacon, is one of the oldest and most widely used modulation schemes in space communications. The coherent BPSK demodulators commonly used with (LEO) amateur radio satellite telemetry often have trouble maintaining carrier lock on a fading signal. The AMSAT phase system describes “Phase 3” as having reliable telemetry and command systems and refers to highly elliptical orbit satellites. Not only are the amplitude variations a problem for the carrier tracking loop, but there can be a significant Doppler component due to antenna motion, especially at S-band, and there can be
sudden phase jumps associated with specular reflections off spacecraft components. Strong FEC will not help if the demodulator cannot track the signal.

Therefore, differentially coherent demodulation (DBPSK) is implemented. A coherent detector extracts its carrier phase reference from a relatively wide sliding window of received symbols, but this window is a big target for a channel phase disturbance. DBPSK only uses the phase of the symbol immediately before the one we are demodulating, so a short disturbance only affects a few symbols. For DBPSK to work, the data has to be encoded separately at the transmitter i.e., to encode a 1-bit not as a particular carrier phase, but as a change of phase from one symbol to the next. Similarly, a 0-bit is encoded as no phase change. That allows the user to recover the data by directly comparing pairs of received symbols.

3.4.3 Forward Error Correction (FEC)

Methods for dealing with errors on one-way data links are called 'Forward Error Correction'. This is achieved by adding redundancy to the data in a prescribed way [159]. The recipient then exploits this additional information to detect and correct errors in the received message. FEC coding techniques are classified as either block codes or convolutional codes [160]. The detailed classification of FEC coding is shown in Figure 3-11. The classification depends on the presence or absence of memory.

1. A block code has no memory, since it collects and therefore isolates \(k\)-bits in a buffer prior to processing:
   a. The number of sample points collected prior to processing is far less than required for a block code. The delay through the encoder is therefore far less.
   b. The encoder acts on the serial bit stream as it enters the transmitter.
   c. Each bit in the output stream is not only dependent on the current bit, but also on those processed previously. This implies a form of memory.
   d. The performance is less sensitive to Signal-to-Noise Ratio (SNR) variations than that of block codes. In situations of limited power, where SNR would be a concern, the preferred method for achieving FEC is therefore to use convolutional codes.
The satellite signals received on the ground are less reliable due to antenna off-pointing, spin induced fading, and the spectrum is very noisy, especially in the VHF/UHF bands as seen in Figure 3-12, thus, making selection/decoding of signal highly challenging.

In such cases, FEC coding provides substantial improvement in the reliability and would permit the acquisition of telemetry using lower gain antennas than in the un-coded format cases [182] as illustrated in Figure 3-13.

It is noted that un-coded telecommand/telemetry (TC/TM) is still common in many CubeSats.
FEC helps to reduce the average signal-to-noise (SNR) requirements to permit the use of lower antenna gain/system noise temperature (G/T) ground stations and to overcome fading due to attitude instability, partially deployed antennas, pointing inaccuracies and the relative speed between satellites in space.

In our reference design includes a second layer for interleaving after the convolutional encoder, just before the modulator to overcome fading channels. Up to 16% of the errors in the demodulated channel symbols can be corrected by error control coding [178].

3.4.4 Encoder Steps and Requirement Analysis

One particular scheme, common among several CubeSats is derived mostly from from Phil Karn’s well known AO-40 design and implementation [KA9Q] [178].

Figure 3-14 gives an overview of the encoder where each frame in this format contains 256 bytes of user information. Each block of 256 data bytes is byte interleaved into two systematic (160, 128) Reed-Solomon codewords. The even-numbered bytes of the user data form the 128 data bytes of the first Reed-Solomon codeword, and the odd-numbered data bytes form the second Reed-Solomon codeword. The (160, 128) Reed-Solomon code is shortened from the standard (255, 223) Reed-Solomon code specified by the CCSDS. (CCSDS 101.0-B-5, section 3). The shortening is accomplished by zero-padding the first 95 symbols of the data field of each Reed-Solomon codeword.
After Reed-Solomon coding, the user data is pseudo-randomized (scrambled) by XORing with the pseudo-random number (PN) sequence generated by the CCSDS-specified polynomial as in Eq. 3-2.

\[ h(x) = x^8 + x^7 + x^2 + x + 1 \]  \hspace{1cm} \text{Eq. 3-2.}

The scrambled Reed-Solomon codewords are next byte-interleaved and encoded using the CCSDS standard constraint length 7, rate 1/2 \((k=7, \ r=1/2)\) convolutional code. The convolutionally encoded data is next written by rows into a block interleaver with dimensions of 80 rows by 65 columns, starting with the second row. Each interleaver element is one bit, for a total interleaver size of 5200 bits or 650 bytes. The first row of the interleaver is reserved for a fixed 65-bit synchronization vector generated by the first 65 bits of the PN sequence produced by the polynomial (Eq. 3-3)

\[ s(x) = x^7 + x^3 + 1 \]  \hspace{1cm} \text{Eq. 3-3.}
Finally, the encoded data is up-sampled twice from 9.6 KSPS to 96 KSPS (this is to match Alex Csete’s decoder) and from 96KSPS to 1.536 MSPS (to meet a minimum input sample rate induced by the AD9361 RF Transceiver SoC) for transmission. Alex Csete is a radio amateur (OZ9AEC) who has worked on several open source SDR software tools. One of these includes his contribution towards the FUNcube telemetry decoder for Linux used in a Dashboard application. As Alex’s code was implemented with FCD Pro, the input sampling rate is confined to 96 KSPS. Therefore, the encoder and decoder blocks had to be changed to meet the hardware requirements (the minimum sampling rate that can achieved on Analog Devices’ development board is 1.5 MSPS).

3.4.5 Data Rate and Symbol Rate Calculations:

Following the encoder steps discussed in section 3.4.4 the symbol rate is calculated accounting for overheads due to the Viterbi encoder, sync vector and CW tone. The input data rate is 256 bytes/s.

FUNcube-1 Data Rate is 256 bytes per 5s = 2048 bits per 5s

= 409.5 bps

Three sources of overhead in FC signal

- Viterbi Flush 6 bits per 2560 bits
- Sync (65 bits) and pad (3 bits) 68 channel bits per 5132 channel bits
- CW tone (800 symbols) per 5200 symbols

The relation between the data rate and symbol rate is given by Eq. 3-4:

\[ Symbol\ Rate = \frac{Data\ Rate}{m \cdot CR_v \cdot CR_{rs}} \cdot (1 + Overhead_{SV})(1 + Overhead_{CW})(1 + Overhead_{VF}) \] Eq.3-4

Where Modulation order, \( m = \log_2(2) \); Viterbi Rate, \( CR_v = 1/2 \); Reed Solomon, \( CR_{rs} = 128/160 \);
SV = Sync Vector; CW = Carrier Wave; VF = Viterbi Flush

\[
Symbol\ Rate = \left( \frac{409.5}{1 \times \left( \frac{1}{2} \right) \left( \frac{128}{160} \right)} \right) \left( 1 + \frac{68}{5132} \right) \left( 1 + \frac{800}{5200} \right) \left( 1 + \frac{6}{2560} \right)
\]

\[ Symbol\ Rate = 1200 \text{ symbols/s} \]

Note: The symbol rates vary depending on the coding schemes chosen.
3.4.6 Memory Calculations

The total memory bandwidth required needs to consider all the various sampling rates, intermediate/decimated sample rates and symbol rates in operation and is the sum of what is listed in Table 3-3. 1.536 MSPS is the minimum input sampling rate that can be achieved on AD9361, for 5s the number of input samples is 7.68 Msamples with (16 bit I + 16 bit Q). The input is decimated at two stages: firstly, by a factor of 16 which is hardware decimation; secondly by a factor of 10 which is a software decimation. Decimation is carried out at two stages to match the decoder, which takes input of 96 KSPS discussed later in Section 3.4.9.

<table>
<thead>
<tr>
<th>Table 3-3 Memory Requirement at Different Stages of Decoder at 1k2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>---------------------------</td>
</tr>
<tr>
<td>Input Data</td>
</tr>
<tr>
<td>DDC (s/w decimation = 16)</td>
</tr>
<tr>
<td>s/w decimation = 10</td>
</tr>
<tr>
<td>s/w decimation by 8 (8 times oversampled)</td>
</tr>
<tr>
<td>Demodulated Data</td>
</tr>
</tbody>
</table>

The Zedboard has 256 KB of on-chip memory (Zynq-7020), 560 KB of extensible block RAM and 1 GB of external memory. On-chip memory is too small (256 KB + 560 KB of extensible block RAM) to store the input samples (30.72 MB) before decimation. Therefore, the input samples need to be transferred to an external memory before they are decimated which involves additional time (it takes 7.3ns with data access up to 64 bit width to write/read from external memory i.e., 28.032 ms to write 30.72 MB). After down-sampling the input stream of samples by 160, one complete packet (192 KB) of 5s at 1200 symbols/s can be stored in the on-chip memory available, and up to 4 packets of 5s at 1200 symbols/s can be stored by combining on-chip memory and extensible block RAM. This is the case when the satellite signals are at 1200 symbols/s whereas when there are signals at a higher data rate, the memory requirement increases.

An example case at 19,200 symbols/s is discussed in Table 3-4. This demonstrates that only one decimated packet of 5s at 19,200 symbols/s can be stored by combining on-chip memory and
extensible block RAM. Therefore, the architecture has to account for external memory read/write time at each stage, which increases the CPU consumption. This design demands the signal to be oversampled due to constraints on the front-end and to meet the reference decoder, which makes it harder on the CPU. However, this issue can be overcome by use of a different front-end or using other CubeSat communication standards.

**Table 3-4 Memory Requirement at Different Stages of Decoder at 19k2**

<table>
<thead>
<tr>
<th></th>
<th>Number of samples/s</th>
<th>Number of Samples for 5s</th>
<th>Memory Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Data</td>
<td>1.536 MSPS</td>
<td>7.68 Msamples</td>
<td>30.72 MB</td>
</tr>
<tr>
<td>DDC (s/w decimation = 1)</td>
<td>1.536 MSPS</td>
<td>7.68 Msamples</td>
<td>30.72 MB</td>
</tr>
<tr>
<td>s/w decimation = 10</td>
<td>0.1536 MSPS</td>
<td>7.68 Msamples</td>
<td>3.072 MB</td>
</tr>
<tr>
<td>s/w decimation by 8 (8 times oversampled)</td>
<td>19.2 KSPS</td>
<td>96 Ksamples</td>
<td>384 KB</td>
</tr>
<tr>
<td>Demodulated Data</td>
<td>6553.6 bps</td>
<td>32768 bps</td>
<td>4096 B</td>
</tr>
</tbody>
</table>

3.4.7 Real-time Satellite Signals Received on a Dongle Connected to a Regular PC/Laptop

The chosen Open Source Software (OSS) starting point to form a “reference implementation” is Alex Csete’s FUNcube decoder (fcdec) available on github [161]. This code base, targeted for Linux, is designed to work offline using sample files captured from the FUNcube Pro Dongle. The default sample rate is 96 KSPS. It was modified to run in soft real time on a modern x86 laptop. With further effort, it was possible to create a soft real implementation based on the higher (and more representative) sample rates available from the RTL-2832 Dongle [162].

This evolved into a reference implementation called "rtl-fcdec". This was tested for interoperability against FUNcube-1 reference signals [163] up-sampled, stored and played back on a Rohde & Schwarz SMBV100 Vector Signal Generator (VSG) [164]. The block diagram of setup 1 is shown in Figure 3-15, which includes the Surrey Space Centre (SSC) ground station antenna setup connected to a LNA followed by a splitter to divide the signal into two paths; one connected to the dongle and the other to the spectrum analyser. Table 3-5 shows the ground station setup used to track FUNcube-1.
Table 3-5 SSC Ground Station Setup

<table>
<thead>
<tr>
<th>Equipment</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Antenna</td>
<td>144 and 430 Medium gain circular polarised yagis</td>
</tr>
<tr>
<td>Low Noise Amplifier (LNA)</td>
<td>SP-7000 Preamp (+20 dB)</td>
</tr>
<tr>
<td>Rotator</td>
<td>G-5500 Controller</td>
</tr>
</tbody>
</table>

Figure 3-15 Setup 1 Block Diagram

The signal received from FUNcube-1(real-time) centred at 145.935 where the Doppler effect is evident is shown in Figure 3-16 and the constellation plot of the same signal is shown in Figure 3-17. The signal thus received was decoded on a Personal Computer (PC) using the reference implementation 'rtl-fcdec'.

The signal received from FUNcube-1(real-time) centred at 145.935 where the Doppler effect is evident is shown in Figure 3-16 and the constellation plot of the same signal is shown in Figure 3-17. The signal thus received was decoded on a Personal Computer (PC) using the reference implementation 'rtl-fcdec'.
The steps performed in the Decoder are as follows:

1. Convert 96 KSPS 16-bit real sampled signals into frequency domain using 8192 FFT.
2. Find the bin with maximum magnitude.
3. Extract 204 frequency bins around selected the chosen bin (Centrebin).
4. Convert back to time domain (inverse FFT).
5. Down sample (and filter) from 96 KSPS to 9600 SPS at this point we have 1200 bps (DBPSK) which is 8 times oversampled.
6. Complex down-conversion to remove residual 1200 Hz offset.
7. Carrier phase recovery (just complex multiplication with last symbol when using DPSK)
8. Correlating for 65 symbol unique word.
9. 5200 bits of aligned output into FEC Decoder.
As a next step, the Dongle and the PC were replaced by AD-FMCOMMS3-EBZ and Zedboard with the same decoder program, however it failed to decode the signals. Despite this, we could see the Doppler shift in the frequency as seen in Figure 3-18. The Eq. 3-5 relates the centre-bin and frequency:

\[
\text{Frequency} = \frac{\text{Centrebin}}{8192} \times 96000 \quad \text{Eq. 3-5}
\]

\[
= \frac{200}{8192} \times 96000
\]

\[
= 2343 \text{ Hz}
\]

Initially, the centre-bin values are arbitrary as the frequency is not locked (the satellite is below the horizon), but once the signal from the satellite is strong and the satellite is visible to the ground station, the frequency is locked (centre-bin value around 200) and stays locked until the satellite is below the horizon (i.e. for about 10 minutes). A slight drift in the centre-bin value can be noticed which is due to the Doppler effect (changes from 200 to 180). As the signal strength drops slowly towards the end of the pass the centre-bin values start to vary again. This confirmed that we were tracking the right signal from FUNcube-1 on the AD-FMCOMMS3-EBZ.
The frequency corresponding to the centre-bin value of 200 is 2343 Hz which is the offset from DC. In order to investigate the problem of decoding in detail there was a need for a transmitter that could transmit at any desired time without the need to wait for a good elevation pass. The second setup includes one such transmitter, which was built in-house adhering to the FUNcube-1 parameters.
3.4.8 European Student Earth Orbiter (ESEO) Transmitter Signals Received on Dongle and Regular PC

The second setup, as seen in Figure 3-19, is same as the previous one, except that the real satellite is replaced by the (ground based) European Student Earth Orbiter (ESEO) transmitter implemented on an embedded system [165]. The transmitter transmits a carrier for ~5 seconds and stays silent for ~3 seconds. Figure 3-20 shows the signal received on the spectrum analyser centred at 90.014 MHz – as expected we are not seeing the Doppler effect in the received signal as the signal is from a stationary source and not from a real satellite. These signals were decoded successfully on a regular PC/Laptop as seen in Figure 3-21.

![Diagram](image)

**Figure 3-19 Setup 2 Block Diagram**

![Image](image)

**Figure 3-20 Transmitted Signal from Setup 2**
Having confidence in decoding the signal from ESEO transmitter, the next step was to import the decoder (rtl-fcdec) on to the chosen embedded system (Zedboard + AD-FMCOMMS3-EBZ) which is also referred as Setup 3.

### 3.4.9 Transceiver on Embedded Systems (AD-FMCOMMS3-EBZ and Zedboard)

The Analog Devices AD-FMCOMMS3-EBZ has a so-called “bare metal system” and Linux OS based device drivers accompanied by some application examples. For this work, we have proceeded with the Zynq ARM Linux OS based approach, assuming at least, that the integration and testing of any application related OSS may be simplified. The functional blocks of the system are presented in Figure 3-22. The core is programmable through an AXI-lite interface. The data path consists of an AXI Video Direct Memory Access (VDMA) and Direct Memory Access (DMA) interfaces for the transmit and receive path respectively. The digital interface consists of 12 bits of Double Data Rate (DDR) data and supports full duplex operation in all configurations up to 2×2 input/output ports.

The transmit and receive data paths share a single clock. The data is sent or received based on the (programmable) configuration from separate transmit and to separate receive chains. In the transmit direction, complex I and Q signals are generated for each RF port. The digital source could either be an internal Direct Digital Synthesizer (DDS) or from the external DDR via VDMA. The internal DDS phase and frequency are programmable. In the receive direction, each component of the received data is passed to a pseudo-random noise (PN) monitor. The monitors

---

**Figure 3-21 Decoded Signal from ESEO Transmitter (Date: 21/09/2014 Time: 13:30:00)**
validate the digital interface signal capture and timing. The data are then optionally DC-filtered, corrected for I/Q offset and phase mismatches and are written to the external DDR memory via DMA. The device control and monitor signals are interfaced to a GPIO module. The Serial Peripheral Interface (SPI) signals are controlled by a separate AXI based SPI core.

To this end, Analog Devices provide a Linux device driver, dependent on and accessed, using the Linux industrial I/O (IIO) framework [166]. Linux IIO allows user space signal applications to configure/query/sample-stream to-and-from the AD9361 using familiar UNIX calls (open/close/read/write/ioctl) and, perhaps more preferred, by a user space library called libiio. The Linux libiio provides a modern high performance abstraction to all IIO devices including the AD9361.

The library abstracts the low-level details of the hardware, and provides a complete programming interface as seen in Figure 3-23, that can be used for advanced projects.

The library is composed by one high-level API, and several backends:

1. the “local” backend, which interfaces the Linux kernel through the sysfs virtual filesystem,

2. the “network” backend, which interfaces the iiod server through a network link.
The IIO Daemon (IIOD) server is an example of an application that uses libiio. It creates a libiio context that uses the “local” backend, and then share it on the network to any client application using the “network” backend of libiio and connected to the server.

Using IIOD, it proved straightforward at first to produce an fcdec variant called ‘iio-fcdec’ reusing much of the proven ‘rtl-fcdec’ code base. A practical problem encountered stems from the lowest filtered decimated sample rate, of order 1.5 MSPS that can be output from AD9361 RF SoC. To address this, the AD9361 is configured to produce a multiple of an oversampled symbol rate (e.g. n x 1200) that is conveniently larger than the 1.5 MSPS limit imposed. In this implementation, 1.536 MSPS was chosen, which derives from 16 x 96 KSPS. Now, within the Baseband SoC, the received sample stream is decimated by 16 (initially in software, but to be moved to firmware). The resulting 96 KSPS sample stream has sufficient bandwidth to allow residual LO breakthrough and IQ imbalance artifacts to be simply discarded, halving the available bandwidth to 40 kHz, but with sufficient remaining bandwidth to address spacecraft Doppler and oscillator uncertainties.

For greatest flexibility and simple access to floating point arithmetic, the 96 KSPS sample stream is processed in software within the ARM Cortex A9. The first signal processing step is coarse carrier acquisition, performed using an 8192 point Fast Fourier Transform (FFT).
results in a further 96 KSPS sample stream that is approximately band centred on the largest (wanted) carrier. A software based Finite Impulse Response (FIR) filter, 27-taps long, containing a low-pass impulse response, is used to further filter and decimate the signal by factor of 10 to 9.6 KSPS and offset by 1.2 kHz from baseband (for amateur radio heritage reasons). At this stage the underlying signal is down-converted to baseband and matched filtered, followed by carrier phase recovery. Finally, following symbol timing recovery, a 1200 symbol stream is produced and passed to the decoder. Further, it has been possible to create a reference encoder called ‘iio-fcenc’. Successful interoperability testing of iio-fcenc and rtl-fcdec took place and Figure 3-24 shows the signal being received on a FUNcube Pro+ dongle. This signal was decoded using rtl-fcdec on a Linux machine first before testing with ‘iio-fcdec’ (the same as Figure 3-21).

![Figure 3-24 Signal Transmitted from Setup 3](image)

The AD-FMCOMMS3-EBZ provides the flexibility to transmit at any desired frequency within the range of 70 MHz to 6.0 GHz. This has an advantage over traditional transmitters which involve unique hardware for each frequency, thereby demonstrating the SDR attributes mentioned earlier, and meets the requirements discussed in Section 3.1. The freedom to choose the frequency in software helps in compensating for thermal drift, clock timing and Doppler effects.

The different transmitter platforms available to test the receiver chain are (Figure 3-25); real satellite with Doppler frequency shift, constant transmitter (setup 2) at 96.014 MHz and the
transmitter that can be tuned to any desired frequency within the range. The receiver was tested with test setup 2 and the frequency offset was tracked with an average error of 789.75 Hz (calibrated with a reference signal generator). The samples were captured and an FFT was plotted to check the spectrum as seen in Figure 3-26, the received spectrum resembled the spectrum transmitted at the right frequency offset (10kHz from DC) along with other interferences.

![Setup 3 Block Diagram](image)

**Figure 3-25 Setup 3 Block Diagram**

![FFT Plot of the Samples Received from AD9361](image)

**Figure 3-26 FFT Plot of the Samples Received from AD9361 (Date: 21/09/2014 Time: 13:30:00)**
The transmitted signals from ESEO transmitter (Setup 2) and the signals that were looped back from Setup 3 were successfully received on the Zedboard and decoded as seen in Figure 3-27.

![Figure 3-27 Data Received on Zedboard (Date: 14/11/2014 Time: 10:12:36)](image)

The reception was also verified on a Rohde & Schwarz FSV3 Vector Signal Generator (VSG) as seen in Figure 3-28 and the constellation plot of the BPSK signal can be seen in Figure 3-29. The Error Vector Magnitude (EVM) is ~2% which is within acceptable values for low order modulations. The carrier frequency offset is 225 Hz from the centre frequency (145.935 MHz) suggesting absolute accuracy of AD-FMCOMMS3-EBZ crystal to be ~1.5 ppm.

The implementation and validation carried out so far has achieved the goal of combining the state-of-the-art SDR hardware and open source software tool. It has made the first step towards a new communication platform on embedded systems aimed at small satellite missions.
Software Defined Radio (SDR) Architecture for Concurrent Multi-Satellite Communications
Mamatha R. Maheshwarappa

Figure 3-28 Transmission Verified on Rohde & Schwarz VSG

Figure 3-29 Constellation Plot of the Signal Received from Test-bed 3
3.5 Summary

Firstly, the drivers and requirements of this research were formulated: combining open source hardware and software tools, reconfigurable receiver platform in order to adapt to different satellite signals with varied standards to perform concurrent multi-satellite reception. Second, a detailed transceiver block diagram was discussed that can support the requirements along with a survey of present technologies to support the transceiver architecture.

The next section physically reviewed the test-beds chosen by carrying out an implementation on each platform. Analog Devices’ AD-FMCOMMS3-EBZ and Xilinx Zedboard were chosen in order to implement the standard signals that can be tested and analysed. Once the hardware was chosen, the FUNcube-1 (AO-73) CubeSat was selected to provide a good starting point for our work because its telemetry beacon is documented and addressed by a number of Open Source Software (OSS) demodulator decoder implementations. A FUNcube-1 decoder reference signal was modified to run in soft real time, and further, the decoder was integrated with the libiio library from Analog Devices in order to validate the signals on an embedded system, which is one of the objectives. The transmit and receive signals were tested separately for its working, and validated on different platforms.

On-board memory calculations were carried out for different symbol rates and at different stages of implementation to understand the requirements. It is important to note that the chosen reference design is oversampled (160 times) and therefore needs external memory to accommodate the signal before decimation (the internal memory is limited to 256 KB + 560 KB of extendable block RAM and the input samples before decimation require 30 MB). In the future, the parallel architecture for concurrent multiple satellite reception should account for the on-board resources such as memory, time and CPU, while handling signals of different standards as they would have different requirements. Hence, the implemented and tested design is now taken forward for performance analysis and characterisation, which is discussed in Chapter 4.
4 PERFORMANCE ANALYSIS

Starting from an existing C/C++ open source code base, a reference signal is adapted, analysed and partitioned, based on the empirical profiling of processor resources utilised on a range of different multi-core general-purpose processors. From this analysis, and partitioning assumptions, some basic requirements for an embedded platform for parallel multi-satellite reception can be established. Profiling of the C/C++ based reference signal design is carried out on dual, quad and octa-core CPUs with the aim of moving minimum functionality from General Purpose Processor (GPP) Software to FPGA firmware, in order to meet performance goals, maximise flexibility and minimise expense associated with implementation of many variant signals. Section 4.1 gives an overview of the software profiling along with an analysis of different profiling tools and their performances. Section 4.2 and section 4.3 highlight the computationally intensive transmitter blocks and receiver blocks respectively, along with the analysis of the response of different platforms and section 4.4 summarises the chapter's findings.

4.1 Software Profiling

Building optimised software systems is both an art and an engineering discipline. Software construction is an inherently recursive process, where system designers and developers iterate between problem understanding and realisation of the solution as seen in Figure 4-1. Hotspots are areas in the code that take a long time to execute. The benchmark is a program that is used to objectively evaluate performance of an application and provide repeatable application behaviour for use with performance analysis tools.
A superficial understanding of behaviour is often insufficient, particularly for embedded systems, where performance is tightly coupled to variations in the execution environment, such as the load on shared resources and hardware clock speeds. To build predictable and optimised embedded systems, we need tools that can help (1) improve understanding of execution weight of each block and (2) show the dependency on increasing symbol rates.

**Figure 4-1 Software Optimisation Process**

The key steps to find performance bottlenecks are:

1. Determining how the system resources, such as time and CPU are being utilised to identify system-level bottlenecks.
2. Measuring the execution time for each module and function in the application.
3. Determining how various standards (in our case symbol rates) running on the system affect the performance.
4. Identifying the most time-consuming function calls and call sequences within the application.

There are two basic approaches to behavioural analysis: static and dynamic. The static analysis tools do not modify the binary image of an application, and instead rely on techniques such as source code instrumentation or sampling to obtain their results [167]. The type of information that can be obtained from any specific static analysis tool is largely a function of the type of evaluation techniques that are employed. For instance, source code instrumentation tools such as gprof can provide rudimentary timing data about the various functions defined within an application, while highly advanced sampling tools such as qprof can provide detailed statistical
reports about shared library usage and kernel-level interactions [168]. While such data can certainly simplify the task of tracking down performance analysis problems, other issues inherent to these types of tools prevent their use in certain situations.

By virtue of the fact that static analysis tools are incapable of modifying a running program, any statistical data that a developer wishes to collect must be specified prior to when the application is running [169]. Furthermore, most static analysis tools report results asynchronously, meaning that if a performance issue arises halfway through the instrumented run of an application the developer will not be notified until after the entire run has completed. As a result, any performance issues that require real-time feedback to diagnose cannot typically be detected by these types of tools.

The use of static analysis tools can cause system slowdown due to the overhead of gathering statistics, and thus can have effects upon application performance. Furthermore, this external code can potentially change the behavior of a running program by introducing performance issues that did not exist prior to analysis or falsely alleviating those that previously existed in the program. Despite these issues, these types of tools have become invaluable in the real world due to the useful statistics that they can gather. The aim here is to identify the most time-consuming functions and execution time of each module that can be achieved using static methods, despite the limitations. Also, at this stage the analysis does not demand real-time feedback.

Detailed descriptions of each of the major subtypes of static analysis tools are provided in Table 4-1, starting with the earliest compile-time instrumentation tools and proceeding onward towards modern sampling and compound tools. Along the way, specific implementations of each subtype are discussed in an effort to demonstrate the effectiveness of each static analysis technique in the real world.

Alternatively, dynamic analysis tools rely on binary-level alterations to facilitate the gathering of statistical data from an application [170]. Such alterations are inserted while the application is running so that accurate statistics can be gathered in real-time [171]. This, in turn, enables dynamic analysis tools to provide insights into program performance that would not be possible to obtain via static examination techniques. As a result, programs tend to run slower while being analysed due to the increased overhead when compared to static analysis tools caused by the insertion or activation of performance monitoring routines. Beyond this, the "random" insertion of code into a binary system can affect the flow of instructions through a processor pipeline, thus modifying the performance characteristics of the application.
4.1.1 Performance Tools

The capabilities that a specific performance tool can provide vary widely depending upon the types of analysis technique that it implements. A set of four classifications based upon the basic types of analysis techniques are shown in Table 4-1:

<table>
<thead>
<tr>
<th>Subtype</th>
<th>Features</th>
<th>Shortcomings</th>
<th>Example Tools</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compile-time Instrumentation Tools (CITs)</td>
<td>• Instruments applications at the source-code level.</td>
<td>• Typically cannot gather statistics at the library or kernel level.</td>
<td>Gprof</td>
</tr>
<tr>
<td></td>
<td>• Can gather call counts for each function in an application.</td>
<td>• Requires that an application’s entire source tree be available to instrument properly.</td>
<td>Prof</td>
</tr>
<tr>
<td></td>
<td>• Can generate call graphs to show flow of control through an application.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Obtains data in a precise manner; does not rely on statistical methods.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sampling Tools (STs)</td>
<td>• Instruments applications via statistical sampling.</td>
<td>• All data obtained is approximate at best due to sampling methods employed.</td>
<td>Qprof</td>
</tr>
<tr>
<td></td>
<td>• Can gather call counts for each function in an application.</td>
<td>• Typically cannot generate call graphs.</td>
<td>Oprofile</td>
</tr>
<tr>
<td></td>
<td>• Can determine how much time was spent in each portion of an application.</td>
<td>• Can require the presence of specialised timing hardware to obtain reliable results.</td>
<td>Prospect</td>
</tr>
<tr>
<td></td>
<td>• Many implementations are able to obtain statistics at both the library and kernel level.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Hardware Counter Tools (HCTs)
- Instruments applications via statistical sampling.
- Can make use of hardware counters embedded in modern microprocessors to characterise applications via their hardware usage.
- Many implementations can characterise hardware usage at both the library and kernel level.
- All data obtained is approximate at best due to sampling methods employed.
- Number of hardware counters available limit the types of statistics that can be obtained in a single instrumented run.
- Requires the presence of specialised hardware that may not be available on all platforms.

## Compounded Tools (CTs)
- Combine one or more static analysis techniques into a single tool.
- Can provide developers with a multi-facted view of their application.
- Provides inherent benefits of each technique they implement.
- Run the risk of being a “jack-of-all-trades”, but a master of none.
- Can be more complex to operate than their single-use counterparts.

<table>
<thead>
<tr>
<th>Perfsuite</th>
<th>DCPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sunfire Link</td>
<td></td>
</tr>
<tr>
<td>Perfmon</td>
<td></td>
</tr>
<tr>
<td>Statsmod</td>
<td></td>
</tr>
</tbody>
</table>

| Intel vTune |
| AMD code analyser |

As can be seen in the table, each type of static analysis tool has its own unique set of features and shortcomings. Despite this, each classification of tool shares the quality that it functions in a static manner, meaning that no modification of an applications' binary image is performed when it is analysed [172]. As a result, these tools are typically only capable of generalised analysis and not ideal for use in cases where highly detailed performance statistics need to be gathered. In the vast majority of situations, however, a static tool exists that is capable of pinpointing any type of performance issue a developer encounters, provided that it is not buried under multiple layers of abstraction. Our application requires one such tool that highlights the blocks in the architecture that consume a lot of CPU time without going into libraries/kernel level details and
which can be used on an embedded Linux environment. Therefore, GNU gprof was chosen in this research in order to identify critical regions, determine the blocks that need to be optimised, vectorised and/or moved to FPGA firmware.

4.1.2 Gprof

Gprof is a cross-platform performance analysis tool that is capable of serving both as a CIT and a sampling tool. To perform a compile-time analysis, gprof relies on two components: an augmenting compiler and a data analysis wrapper. The augmenting compiler is used to insert a monitoring function into a targeted application at the source level along with a set of calls to the monitor that are injected prior to each function call in the program. The resulting instrumented binary is then executed via the wrapper, causing raw performance data to be collected each time a function call is triggered. This data is then analyzed after the program terminates and presented to the end-user [173].

The statistical data collected by gprof is at the function level and primarily consists of call counts, call graphs, and other related information. Such results can be used to reconstruct the internal structure of an application and identify where performance bottlenecks might exist. For instance, if a single function has a relatively high call count compared to others in the application, optimising it would likely result in a significant impact on performance [174].

The aim of this particular exercise in this research is to exploit the flexibility that can be achieved by common FPGA cores (digital down converter/fast Fourier transform (DDC/FFT) [175]) to optimise the implementation in order to accommodate more than one signal path on the baseband SoC, while maximising remaining functionality in software. The use of GNU gprof assists in making the choices in an educated and incremental fashion. During profiling, the packet/frame decoding, success rates are recorded to later aid results reconciliation. In this approach, the symbol rate is increased to (and beyond) the point that CPU starvation sets in i.e., when the symbol rates are 9k6 and 19k2. Using a block based signal, realised mostly in pure software, the observed effects of CPU starvation are not catastrophic, and rather a graceful degradation in performance occurs.

GNU-based profiling framework consists of:

1. An instrumentation library for collecting and recording data.
2. A compiler (GNU) that inserts calls to the instrumentation library into and application code; and
3. A post-processing tool called gprof for viewing the collected profile data.

When used with an application, the gprof framework provides the following information:
1. The relative time spent in each routine.
2. The number of times a routine was invoked.
3. A list of the parent routines that invoke a given routine.
4. A list of the child routine a given routine invokes.
5. An estimate of the cumulative time spent in the child routines invoked by a given routine.

4.1.3 Profiling Setup Overview

This section focuses on understanding the time taken to execute each function in the transceiver implementation carried out in Chapter 3. This includes both platform functions (front-end initialisation and configuration) and transceiver functions (decoding, demodulation, down conversion, up conversion, encoding etc). In order to understand the bottlenecks for both DSS and ground station applications, different platforms were selected for backend processing retaining Analog Devices’ front-end throughout:

1. Single embedded platform implementing platform and transceiver functions on single OS.
2. Two embedded platforms used to partition platform (including RF ADC/DAC) and transceiver functions (including core modem) onto separate CPU’s, separated by high speed Digital IF.
3. One embedded platform and one high performance GPP platform (Desktop PC) used to partition platform and transceiver functions, such that a reference performance can be assessed for transceiver functions across the widest possible range of symbol/data rates.

As the AD-FMCOMMS3-EBZ development board could not be integrated to all three platforms – the digitised samples were streamed to a remote platform (using Ethernet Digital IF – IIOD in this case) in order to better understand the separation of platform (sample based) and transceiver function (symbol based) overheads. Such platform overheads include high sample rate buffering and user/kernel space device driver steps, extending the “hardware in the loop” approach, to best inform the later implementation partitioning (discussed in Chapter 5) previously unseen in the literature.

Thus, we have 3 profiling/partitioning steps that we systematically compare: We learn from low data rate, low symbol rate implementation on the single embedded platform (Step 1), but expect performance limitations. We use Step 2 to allow higher symbol rates to be considered, still using representative embedded platforms, but where the relative platform and waveform
overheads can be better separated and assessed/analysed independently. Finally, we have Step 3 that allows the widest range of symbol rates to be considered, but on a potentially non-representative computational platform.

4.2 Transmitter Profiling

The FUNcube-1 encoder reference implementation was integrated within the ADI IIO SDR framework, which evolved into a standalone binary. This new part of the work is called 'iio-fcenc' which is written in C. An overview of the encoder steps is shown in Figure 4-2 (details of encoder steps were discussed in Section 3.4). The user data is RS encoded, interleaved and convolutional encoded before up sampling from 96 KSPS to 1.536 MSPS for transmission.

![Figure 4-2 Encoder Overview](image)

The processing steps required for transmit chain are captured in the pseudo code shown in Figure 4-3, and descriptions of each function are provided in Table 4-2. The transmit blocks were initially implemented on ARM Cortex A9 processors as it was a straightforward task to get the reference design in C to run on the processor. Gprof, the chosen static profiling tool was used to profile these blocks in order to understand where the bottleneck exists.

---

**Pseudo code:**

Begin

1. Initialise the Encoder
2. Each block of 256 bytes is RS encoded, followed by Interleaving and convolutional encoding to 650 bytes
3. Check for parity and perform FIR filtering.
4. Complex up conversion form 9.6 KSPS to 96 KSPS
5. Up convert from 96KSPS to 1.536 MSPS
6. Configure and stream samples using ADI IO framework.

End
# Table 4-2 Encoder Functions and Their Tasks

<table>
<thead>
<tr>
<th>Functions</th>
<th>Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Main()</strong></td>
<td>Configures and streams samples using ADI IO framework.</td>
</tr>
<tr>
<td><strong>FCsample</strong></td>
<td>Up converting the sample from 96KSPS to 1.536 MSPS</td>
</tr>
<tr>
<td><strong>TxGetNextSample</strong></td>
<td>Performs filtering (Finite Impulse Response (FIR))</td>
</tr>
<tr>
<td><strong>Parity</strong></td>
<td>Return 1 if x has odd parity, 0 if even in order to interleave into two systematic Reed-Solomon codewords.</td>
</tr>
<tr>
<td><strong>Encode_and_interleave</strong></td>
<td>Internal function to convolutionally encode and block-interleave one byte (no scrambling)</td>
</tr>
<tr>
<td><strong>Scramble_and encode</strong></td>
<td>Internal function to scramble a byte, convolutionally encode and block interleave</td>
</tr>
<tr>
<td><strong>Interleave_symbol</strong></td>
<td>Internal function to write one binary channel symbol into the block interleaver and update the pointers</td>
</tr>
<tr>
<td><strong>Mod255</strong></td>
<td>Reduce argument modulo 255 without a divide</td>
</tr>
<tr>
<td><strong>Encode_byte</strong></td>
<td>This function is called with each user data byte to be encoded into the current frame</td>
</tr>
<tr>
<td><strong>Encode_parity</strong></td>
<td>This function is called 64 times after the 256 bytes have been passed to update_encoder. Each call scrambles, encodes and interleaves one byte of Reed-Solomon parity</td>
</tr>
<tr>
<td><strong>Init_encoder</strong></td>
<td>This program initialises the encoder, it only has to be called once at program alert</td>
</tr>
</tbody>
</table>

Successful testing of iio-fcenc took place for different symbol rates such as 1k2, 2k4, 4k8, 9k6 and 19k2. A shell script was written to automatically change the symbol rate every 5s and these signals were received on a FUNcube Pro+ dongle and spectrum analysis was performed using SDRSharp [24] as seen in Figure 4-4. It is evident that the bandwidth increases with the symbol.
rate and the software can switch into different symbol rates dynamically which is one of the requirements as discussed in Chapter 3. The FUNcube-1 signal has 0.7s of overhead (10101) every 5s.

Using gprof, it was possible to generate a flow of the computationally intensive transmitter blocks implemented (as seen in Figure 4-5) on the Xilinx Zynq – Processing System where main() is streaming the samples and FCsample() is performing up-sampling which reports the maximum CPU consumption. Each block contains:

a. The name of the function which is calling other functions.
b. Percentage of the total time that is propagated to this function from previous functions.
c. The percentage of the total running time of the program used by this function.
d. Number of times this function was invoked by the previous function.
e. The numbers in between the blocks are: the percentage of the total running time left for other functions and number of times the following functions was invoked by the previous function.

The percentage of the total running time of the program used by each function is plotted against the CPU time in order to understand the relative time accounted for individual functions, which will aid in distributing the resources efficiently. As discussed earlier in Section 4.1.3, the profiling includes three setups: Single embedded platform, two embedded platforms, and one embedded platform and one high performance GPP platform.
Software Defined Radio (SDR) Architecture for Concurrent Multi-Satellite Communications
Mamatha R. Maheshwarappa

The processed response of the receiver functions to gprof was different on Intel x86 (on high performance platform) and ARM Cortex A9 (single embedded platform) which is discussed later in Section 4.3, thus making the like-for-like comparison of the receiver blocks difficult. Consequently, the samples were streamed over a network to another embedded system [Odroid-XU Lite] which has Octa – ARM Cortex A15 Quad Core and A7 Quad Core processors. The response to the functions on Octa Core was similar to Intel x86, thus making the like-for-like comparison possible.

Therefore, both transmitter and receiver programs are executed for different data rates and on different platforms discussed in Table 4-3 such as Zedboard, Odroid-XU Lite and Dell Optiplex 745 to understand the function distribution for higher symbol rates. This also helps in understanding the limitations of any embedded system over Desktop PCs and to explore the options of efficiently utilising the resources such as CPU time and memory on board. It is important to note that the setup for Dell Optiplex 745 and Odroid XU Lite were different from Zedboard. With Intel and Octa core A15/A7 sample streaming was over Ethernet while with Zedboard, streaming and physical SDR was on same CPU (ARM Cortex A9). This clearly shows that the sample based overhead changed the behaviour of ARM Cortex A9 due to saturation.
Table 4-3 Comparison of Different Platforms

<table>
<thead>
<tr>
<th></th>
<th>Dell Optiplex 745 (One Embedded and one High Performance GPP Platform)</th>
<th>Odroid XU Lite (Two different Embedded Platforms)</th>
<th>Zedboard (Single Platform)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor</td>
<td>Intel x86</td>
<td>ARM Cortex A15 &amp; A7</td>
<td>ARM Cortex A9</td>
</tr>
<tr>
<td>Number of Cores</td>
<td>Dual</td>
<td>Octa – Quad A15 &amp; Quad A7</td>
<td>Dual</td>
</tr>
<tr>
<td>CPU Frequency</td>
<td>2.13 GHz</td>
<td>A15 – 1.4 GHz; A7 – 1.2 GHz</td>
<td>700 MHz</td>
</tr>
<tr>
<td>Linux Version</td>
<td>3.13.0</td>
<td>3.4.98</td>
<td>3.15.0</td>
</tr>
<tr>
<td>System type</td>
<td>64-bit</td>
<td>32-bit</td>
<td>32-bit</td>
</tr>
<tr>
<td>Application</td>
<td>Identical Application from Source</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Setup</td>
<td>Zedboard (iio-phy)</td>
<td>Intel waveform (iio-lib)</td>
<td>Odroid waveform (iio-lib)</td>
</tr>
</tbody>
</table>

4.2.1 Absolute CPU Consumption

The absolute CPU consumption is recorded by using 'top' command which provides the ongoing look at processor activity in real-time. Figure 4-6 gives the comparison of the absolute CPU consumption on dissimilar platforms while the encoder is running at varied symbol rates. It is evident that the CPU consumption increases along with an increase in the symbol rates. It is clear that only one signal can be transmitted at 9k6 and 19k2 on the Zedboard. It is important to note that the physical layer actions are involved on the same processor in this case and therefore the difference in the behaviour can be observed. The behaviour appears to be linear on the ARM Cortex A9 operating at 700 MHz as the streaming and physical SDR is on same CPU. However, the behaviour is relatively similar on the ARM Cortex A15/A7 operating at 1.4/1.2 GHz and on the Intel x86 operating at 2.13 GHz as the physical layer actions are not involved. In addition, this behaviour can be observed on relative CPU consumption plots of the encoder program across the platforms. That is the ‘main’ function which is handling the physical SDR.
function such as initialising and defining front-end parameters (frequency, filters, input/output voltage) is prominent on the Dual Core ARM Cortex A9 while it is negligible on the other two platforms.

![Graph showing absolute CPU consumption for different symbol rates and platforms]

**Figure 4-6 Absolute CPU Consumption - Transmitter**

### 4.2.2 Relative CPU Consumption

Gprof, the static profiling tool is used to determine the CPU time taken by one function with respect to another function along with number of function calls and their duration. As described earlier, the profiling setup involves three different platforms:

#### 4.2.2.1 Single Embedded Platform

This setup includes Zedboard and the AD-FMCOMMS3-EBZ where both physical layer functions and the higher layer functions such as decimation, frequency correction and decoding were carried out on the same Dual Core processors. Figure 4-7 gives the relative comparison of the CPU consumption by different transmitter functions on the Dual Core ARM Cortex A9. FCSample() which does up-sampling, and main(), responsible for streaming the samples and managing buffers, are the most dominant functions, while other functions are almost negligible. The main() and FCSample() functions contribute ~50% towards the CPU consumption at 1k. The relative contribution of FCSample() increases whereas main() decreases linearly with
symbol rate. Though it appears that the CPU time taken by main() is reducing with the symbol rate, it is a relative measure i.e., the main() is contributing the same, as the input sampling rate is the same, but the FCsample() is consuming more CPU time as it is processing more symbols each time the symbol rate increases.

![Graph of Relative CPU Time vs Symbol Rate](image)

**Figure 4-7 Profiling Results on Dual Core ARM Cortex A9**

### 4.2.2.2 Two Different Embedded Platforms

Here, the setup includes two embedded systems: Zedboard + AD-FMCOMMS3-EBZ and Odroid board. The behaviour of the transmitter functions on “the octa cores” ARM Cortex A15 and A7 as shown in Figure 4-8 is not performing any physical layer functions. Here the sample streaming is quicker compared to ”the dual core” ARM Cortex A9 and therefore the FCsample() dominates over main(). As Ordoid is only working on the symbols but not the input samples the main() is less prominent when compared to the previous case. Also, the processor speed is high (1.4 GHz and 1.2 GHz) compared to Zedboard (700 MHz) which makes the data handling faster. Again, this is a relative measure and therefore the FCsample() appears to dominate. It is important to note that other functions have negligible effect when compared to these two functions. Also, since the aim here is to identify the most dominant function/s, other functions are not discussed in detail.
4.2.2.3 One Embedded Platform and One High Performance GPP

On the Intel x86 (Figure 4-9), the sample streaming is the fastest due to high speed processor (2.13 GHz) and therefore FCsample() is the only function contributing towards 80-100% of the relative CPU time.
The rate of change of CPU consumption by FCsample() reduces on faster platforms with an increase in the symbol rate. This clearly shows that the Ethernet I/O is working more efficiently on the Intel platform.

In conclusion, though the behaviour is slightly different on different platforms due to varied processor speed and configurations, the dominant block on all three platforms is found to be FCsample() which performs up-sampling. This could be moved to FPGA firmware (HDL) for optimisation in the future, thereby reducing the ARM Cortex A9 CPU cycles/time/memory, in order to enable simultaneous multiple-signal transmission.

### 4.3 Receiver Profiling

Similar to the transmitter, the FUNcube-1 decoder reference implementation was integrated within the ADI IIO SDR framework which evolved into a standalone binary. This new part of the work is called ‘iio-fcdec’ which is written in C++. An overview of the receiver is shown in Figure 4-10, which includes acquiring the valid signal in the spectrum followed by phase recovery, matched filter and frame detection. Further, the received convolutional data is decoded using a Viterbi decoder, the data is de-interleaved and the Reed-Solomon code is decoded. Viterbi decoded convolutional codes provide coding gain on Gaussian channels with random channel bit errors. In order to protect the data from these errors there needs to be an error detection/correction code. Each block contains 255 8-bit symbols; 223 symbols contain user data and 32 symbols contain parity computed according to the code specification. A (255, 223) RS code not only provides reliable error detection, it can also correct up to (255-223)/2 = 16 symbol errors in each block.
The Pseudo code (Figure 4-11) includes the key receiver steps describing the decoder steps in C++ and Table 4-4 gives the description of each function and the tasks performed by the receiver blocks. The baseline codes were referred from different places and therefore the transmitter blocks are written in C and the receiver blocks are in C++. In the case of C, importance is given to the steps or procedure of the program while C++ focuses on the data rather than the process. C is regarded as a low-level language (difficult interpretation & less user friendly) while C++ has features of both low-level (concentration on what is going on in the machine hardware) and high-level language (concentration on the program itself), hence is regarded as a middle-level language. Also, it is easier to implement/edit the code in the case of C++ for this reason. This makes C++ a better choice for receiver blocks as the data handling section in the receiver is complex when compared to the transmitter.

**Pseudo code:**

```
Begin
   Convert 96 KSPS 16-bit real sampled signals into frequency domain using 8192 FFT.
   Find the frequency bin with maximum magnitude.
   Extract 204 frequency bins around the chosen bin (Centrebin).
   Convert back to time domain (inverse FFT).
   Down sample (and filter) from 96 KSPS to 9600 SPS; at this point we have 1200 bps (DBPSK) which is 8 times oversampled.
   Complex down-conversion to remove residual 1200 Hz offset.
   Carrier phase recovery (consisting of complex multiplication with last symbol when using DPSK)
   Correlating for 65 symbol unique word.
   5200 bits of aligned output into FEC Decoder.
End
```

*Figure 4-11 Receiver Pseudo Code*

The computationally intensive functions in the receiver chain at 1k2 are seen in Figure 4-12 with details such as name of the function, percentage of the total time that is propagated to this function from previous function, number of times this function was invoked and the percentage of total time left for other functions.
Table 4-4 Decoder Functions and Their Tasks

<table>
<thead>
<tr>
<th>Function</th>
<th>Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main()</td>
<td>Configures and access ADI IIO – sample stream at 1.536 MSPS &amp; down samples from 1.536 MSPS to 96 KSPS.</td>
</tr>
<tr>
<td>CTryDecode::go(float*)</td>
<td>Converts 96KSPS 16-bit real sampled signals into frequency domain using 8192 Fast Fourier Transform (FFT). Finds the bin with maximum magnitude. Extracts 204 frequency bins around selected Centrebin.</td>
</tr>
<tr>
<td>CDecoder::ProcessFFT(float (*)[2], unsigned int)</td>
<td>Converts back to time domain (Inverse FFT)</td>
</tr>
<tr>
<td>CDecoder::RxDownSample(float)</td>
<td>Down samples (and filter) from 96000 to 9600SPS. At the entry to this function a 1200 sps, a AO-40 waveform is 80 times oversampled.</td>
</tr>
<tr>
<td>CDecoder::RxPutNextUCSample(float)</td>
<td>Complex down conversion to remove residual 1200 Hz offset and Root Raised Cosine FIR. Carrier Phase recovery (complex multiplication with last symbol when using DPSK). Correlating for 65 symbol unique word and 3 pad bits. 5200 bits of aligned output into FEC Decoder.</td>
</tr>
<tr>
<td>FECDecode</td>
<td>Decodes the data and on success prints the frame.</td>
</tr>
<tr>
<td>Viterbi277</td>
<td>Viterbi decoder with a length of 7 and a rate of ½.</td>
</tr>
</tbody>
</table>
Figure 4-12 Computationally Intensive Receiver Blocks at 1k2 on Octa Core ARM Cortex A15 & A7
The processed response of the receiver functions to gprof was different on the Octa Core ARM Cortex A15 and A7 (Figure 4-12) and the ARM Cortex A9 (Figure 4-13) and thus making the like-for-like comparison of the receiver blocks difficult. There is a separation between the application and the libiio Linux driver, shown as a parallel thread in Figure 4-13, which includes shutdown(), _init() and frame dummy. Therefore, as mentioned earlier, the samples were streamed over a network to another embedded system [Odroid-XU Lite] which has Octa – ARM Cortex A15 Quad Core and A7 Quad Core processors to make the like-for-like comparison possible. The difference in the response was due to processor saturation and this issue was resolved by moving the computationally intensive block to an FPGA. Once the computationally intensive blocks were moved to an FPGA the response of the ARM Cortex A9 was similar to that of the Octa Core and Intel x86 (discussed in Chapter 5).

4.3.1 Absolute CPU Consumption

Similar to transmitter, the absolute CPU consumption is recorded by using ‘top’ command which provides the on-going look at processor activity in real-time. Figure 4-14 shows the absolute CPU consumption on dissimilar platforms while the decoder is running at varied symbol rates. The decoder consumes more than 50% of the CPU at 1k2 on the ARM Cortex A9 and reaches almost 100% appearing linear which results in low success rate as the symbol rate increases (Figure 4-15). Again, the physical layer functions are initialised on the Zedboard therefore the ARM Cortex A9 has both sample and symbol overhead. However, the samples are routed to other boards via Ethernet and thus the Odroid and Desktop PC do not suffer from the overhead caused by physical layer functions. The success rate is defined as the number of successfully decoded packets per 100 packets received on each platform. At each symbol rate, 100 packets were transmitted from a different source (Desktop PC) and received on the Zedboard+AD-
FMCOMMS3-EBZ boards to see how many packets were completely decoded. As the CPU consumption increased, the success rate decreased which is evident in Figure 4-15. The behaviour on the ARM Cortex A15 and A7 reaches ~50% at higher data rate. This resulted in unsuccessful decoding at 9k6 and 19k2 as is shown in Figure 4-15 and the differences in profiling behaviour are shown in Figure 4-16. The Intel x86 is well within 50% even at 19k2 thus ensuring 100% success rate.

Figure 4-14 Absolute CPU Consumption - Receiver

Figure 4-15 Success Rate Comparison on Different Architectures
4.3.2 Relative CPU Consumption

As the behaviour on the ARM Cortex A9 was different when compared to other platforms, it was not possible to get the relative performance data on the Zedboard. Therefore, on the receiver end we have only two platforms:

4.3.2.1 Two Different Embedded Platforms

The two different embedded platforms are: the Zedboard + AD-FMCOMMS3-EBZ and the Odroid boards. Among the different functions discussed earlier, main() decimates the samples from 1.536 MSPS to 96 KSPS and becomes less dominant (as a relative measure) as the symbol rate increases; whereas the function go(), which converts 96KSPS 16-bit real sampled signals into the frequency domain using a 8192 Fast Fourier Transform (FFT) is more prevalent, thus suppressing other functions. This is because the FFT (contained in CTryDecode::go(float*)) occurs more often as the symbol rate increases. Unlike in transmitter case, we can notice the effect of other functions such as CDecoder::RxPutNextUCSample, which does complex down conversion to remove the residual 1200 Hz offset and FIR filtering, carrier phase recovery and unique word correlation; CDecoder::RxDownSample, which does further down-sampling from 96KSPS to 96 bps decoding and CDecoder::ProcessFFT, which converts back to the time domain, contributing to the performance profile.

![Graph showing relative CPU consumption](image-url)

*Figure 4-16 Profiling Results on Octa Core ARM Cortex A15 & A7*
4.3.2.2 One Embedded Platform and One High Performance GPP

The embedded platform is the Zedboard (which does physical layer initialisations such as gain, voltage control, assigning the number of channels, etc.) and the high performance GPP includes Desktop PC with Dual Core Intel x86 for backend processing. The behaviour is quite similar to that of the Odroid board, where main()’s down-sampling of the the input sampling rate from 1.536 MSPS to 96 KSPS dominates over other functions. However, the FFT is less prevalent compared to the Octa Core ARM Cortex A15 and A7, as seen in Figure 4-17 this is because Intel x86 is more efficient with vector instructions.

In addition, the compilers were found to be different across the platforms, as seen in Table 4-5. There is a difference in the instruction sets used across various architectures to perform similar functions and was observed using ‘objdump’- part of the GNU Binutils [176] used to display information about object files. On the Intel x86 architecture ‘move’ instructions dominate over ‘add’ functions, whereas on the ARM architectures ‘add’ instructions are called more frequently. This may suggest that memory operations are the key, reducing the number of read/write operations to the memory and decimating the samples would make the design more efficient. There are 9 instructions that are common across all platforms, and 131 common instructions among the ARM platforms for identical applications. We suspect that the level of compiler optimisation on the Intel GPP platform may be higher than that of the embedded platform, but this is nonetheless a reference for what can be achieved with pure SDR implementation.
Using this methodology, with our current reference signal, we end up with the simplest and most obvious solution, i.e. move the high rate sample interpolation/decimation functions into the FPGA, which is achieved in Chapter 5. However, this exercise was carried out in order to quantify these numbers so that we may use them later to compare with the profiling results obtained by moving the computationally intensive blocks on to the FPGA.

4.4 Summary

This chapter outlines the different types of software profiling tools available – gprof was selected, as the aim here was to highlight the block in the architecture that consumes the most CPU time, without going into library/kernel level details. This profiling of an adaptive SDR architecture, was obtained using gprof on different platforms such as single/multiple embedded systems and a combination of embedded systems and a general Desktop PC for reference. Also, this exercise was carried out at varied symbol rates such as 1k2, 2k4, 4k8, 9k6 and 19k2. The initial implementation is in C/C++, due to the reduced implementation time when compared to VHDL of simple blocks such as decoding/encoding/demodulation and modulation. Profiling using gprof tabulates the relative and absolute performances along with success rates due to CPU saturation.

Though we can achieve more than one transmitter thread on the Zedboard for lower symbol rates such as 1k2, 2k4 and 9k6, on the receiver end, the CPU consumption reaches 50% with
just one receive thread. This illustrates that concurrent multiple reception is not possible on the chosen platform (Zedboard+AD-FMCOMMS3-EBZ) in this software configuration and it has relatively low success rate. However, multiple reception can be achieved easily on a normal Desktop PC. However, since the aim here is to achieve the concurrent multi-satellite reception on an embedded system, there was a need to quantify the blocks that required more CPU time so that they could be rearranged. The obtained performance results demonstrate the need to move blocks demanding higher computation capacity such as up/down sampling.

Therefore, these blocks need to be moved to programmable logic on the FPGA fabric embedded inside the Xilinx Zynq, in order to reduce the CPU consumption. The architecture needs to be revised in order to efficiently utilise the FPGA firmware and take advantage of the flexibility and fast reconfigurable hardware logic. The FPGA firmware can be re-configured to include one or more DDC blocks. The reference design includes the functionality from Analog Devices, which fetches the samples from the RF SoC interface core and provides them to the Zynq PS for further processing.

A sample DDC block can be implemented in between the RF core (AXI_AD9361) and the sample packer block, which packs I and Q signals from different channels, before the signal is stored in Direct Memory Access (DMA) memory. Other blocks, such as modulation/demodulation, frequency/phase correction and packet handling, which are computationally less intensive, can be retained on the ARM Cortex A9 processors as an initial step. Implementation on a FPGA will come at a cost of more power consumption and use more on-chip resources. The next chapter focuses on the implementation of the DDC block on a FPGA and the overhead analysis for power and on-chip resources.
5 FPGA IMPLEMENTATION

One of the main conclusions, based on the profiling results obtained in Chapter 4, is that the Digital Down Converter (DDC) and Digital Up Converter (DUC) blocks directly impact the volume and rate of memory operations and the CPU consumption. Therefore, the architecture is now revised in order to efficiently utilise the FPGA firmware and take advantage of its flexibility and speed. The FPGA firmware was re-configured to include one or more DDC block(s) to perform the higher sample rate computational tasks. Having multiple DDCs would facilitate the parallelisation required to handle multiple signals at any given time. The use of a simple DDC block, based on discarding samples, reduces the sample rate entering the embedded software domain, allowing software performance benefits, including the use of higher symbol rates, to be assessed. The implementation of a simple DDC block, along with post-profiling results, is discussed in section 5.2. The validation of a more complicated DDC block, with dynamically configurable registers that can be tuned to different centre frequencies and decimation rates (which is one of the main characteristics of a SDR), along with its overhead analysis, and the prospects of accommodating more than one DDC block in order to support parallel reception, is presented in section 5.3. Finally a summary is presented in section 5.4.

5.1 Introduction

The embedded platform reference design includes the core from Analog Devices, which fetches the samples from the AD9361 RF/ADC core and provides them to the Zynq PS for further processing. The DDC block was inserted between the RF SoC interface block (AXI_AD9361) and the sample packer block (util_adc_pack), prior to the AXI DMA interfacing block (AXI_DMA).
Other blocks such as modulation/demodulation, frequency/phase correction and packet handling, which are computationally less intensive, were retained on the ARM Cortex A9 processors.

The C/C++ implementation discussed earlier (Chapter 3) was carried out on the reference Linux OS based design provided by Analog Devices. However, it was necessary to understand the reference design and the process of creating a new image with the DDC block integrated to the FPGA fabric. This was not a straight-forward task, as there are several releases of the reference design, Linux OS, Bootloader and the documentation on adding a new block and creating a new image is very thin. Details of the implementation steps and challenges can be found in Appendix 6. The revised architecture includes a DDC block in the FPGA as seen in Figure 5-1, with the rest unchanged. The DUC block is still retained in the ARM Cortex A9 processor, as the focus here was mainly on the receiver. Also, the profiling results suggest that the transmitter consumes less CPU (Absolute CPU time) when compared to the receiver and multiple transmitter threads could be tested even in the original implementation.

![Figure 5-1 Revised Architecture - DDC Block moved to FPGA Fabric](image)
5.2 Digital Down Converter (DDC)

The Digital Down Converter (DDC) allows the frequency band of interest to be moved down the spectrum so that the sample rate can be reduced. Filter requirements and further processing on the signal of interest become more easily realisable [177]. The DDC consists of three subcomponents: Direct Digital Synthesizer (DDS), a Low Pass Filter (LPF) and a downsampler Digital Quadrature Tuner (DQT) [178]. The DDS generates a complex sinusoid at the intermediate frequency. Multiplication of the intermediate frequency with the input signal creates images centred on the sum and difference frequencies depending on the sign of the DDS. A conventional analogue conversion produces sum and difference frequencies, but in a complex baseband implementation the sum or difference frequency can be chosen and isolated. Here, the LPF is used solely to reject noise. The most common choice is a FIR filter for low amounts of decimation (typically less than ten) or a cascaded integrator comb (CIC) filter followed by a FIR filter for larger downsampling ratios [179]. In this new form, the signal can readily be downsampled using a DQT and is more convenient for further processing due to reduced number of blocks needed.

Being digital gives many advantages, including:

1. Digital stability – not affected by temperature or manufacturing processes.
2. Controllability – all aspects of the DDC are controlled using software. The local oscillator can change frequency very rapidly indeed – in many cases a frequency change can take place on the next sample which can be compensated in software.
3. Size – A single ADC can feed many DDCs, without splitting losses seen in the analogue domain, a boon for multi-carrier applications. Multiple DDCs can be implemented in an FPGA device, so multiple channels can be implemented – or additional circuitry could be added.

However, there are some disadvantages:

1. ADC speeds are limited. At present it is not possible to digitise higher frequencies, such as L and S-bands with low cost, low power ADCs. There are techniques to extend the range of ADCs, but often it is simpler to use analogue circuits to bring the carrier down to an IF that digital circuits can then manage [180].
2. ADC’s dynamic range is limited. In many communications systems, the signal’s amplitude can vary greatly. Fast ADCs often only have 12 bits of resolution – giving an absolute maximum dynamic range of 72 dB. It is often better to use analogue circuits in
conjunction with the ADC to implement AGC functions to ensure that this range is best used [181].

In time, more and more systems will use predominantly digital technology. Our application benefits from a digital down sampler, flexibility, and re-configurability achieved by a FPGA. The existing DDC blocks in Vivado are mainly intended for wide band applications including CDMA2000 and WCDMA standards and are not useful for narrowband satellite communication applications [182]. Therefore, a custom block (simple downsampler (DQT)) was implemented in order to analyse the computational dependencies.

### 5.2.1 Digital Quadrature Tuner (DQT)

Figure 5-2 gives an overview of the DQT function in this application, the minimum filtered decimated rate is 1.536 MSPS (output from the AD9361 RF SoC) but the required sampling rate (to match Alex Csete’s decoder) is 96 KSPS. Therefore, the down-sampling rate of the DQT block is 16.

Here is the simple downsampler pseudo code, as seen in Figure 5-3, implemented on Zynq Programmable Logic (in VHDL), as a first step towards checking the performance improvement.

**Pseudo code:**

```vhdl
begin
    Wait for the valid lines to be asserted (rising_edge) from AD9361 core
    Start the counter
    Wait for the counter to reach the decimation rate defined
    Send the data corresponding to the decimation rate
    Assert the valid line and the flag
    Set the counter to zero
End
```

**Figure 5-3 Downsampler Pseudo Code**
Software Defined Radio (SDR) Architecture for Concurrent Multi-Satellite Communications
Mamatha R. Maheshwarappa

The abstract Vivado block is shown in Figure 5-4, with the input/output lines. The behaviour of these input/output lines is shown in Figure 5-5, where the decimation is set to 4 and the valid lines are asserted after 4 valid lines from the AD9361 core. It is important to note that the valid lines decide the decimation rate. The packer block (where the I/Q signals of different channels are packed together) performs logical-OR on the valid lines and asserts the pack_sync and pack_dvalid. Therefore, valid lines from different DQT blocks have to be in sync at the desired decimation rate. Figure 5-4 demonstrates the valid lines from two DQTs (one on I and one on Q) asserted at the same time and the pack_dsync and pack_dvalid lines going high at the falling edge of ddc_valid_out (i.e., the next clock cycle). Once the pack_sync and pack_dvalid are asserted, the data from the two DQT blocks (16 bits each) on channel 1 and the two DQT blocks on channel 2 are packed to 64 bits data.

![Abstract Vivado Block](image)

Figure 5-4 Abstract Vivado Block
Figure 5-5 Simulation Results Validating the Valid Lines
5.2.2 Post-DQT Profiling Results

Profiling was repeated to understand the improvement achieved once the DQT block was implemented on the FPGA fabric. Figure 5-6 shows the percentage reduction in the absolute CPU consumption at different symbol rates and varying firmware based decimation rates. This improvement allows parallel reception of up to four signals at 1k2 while it was limited to one earlier. Similarly, up to two signals at 2k4 can be decoded simultaneously in place of a single signal.

![Figure 5-6 Post-DQT Profiling Results on Dual ARM Cortex A9 (Absolute CPU Time)](image)

The 1st digit in the x-axis stands for hardware decimation and the 2nd digit for software decimation. It is unambiguous that as the hardware decimation increases, the CPU consumption decreases. Table 5-1 summarizes the improvement achieved at different symbol rates. The improvement is slightly less at higher symbol rates, as the symbol overhead increases for the same number of input samples. Through this exercise we have reduced the input sample overhead in order to match Analog Devices’ requirements and Alex Csete’s decoder.
Table 5-1 Improvement Achieved with Sample DDC Block on FPGA

<table>
<thead>
<tr>
<th>Symbol Rate</th>
<th>% Reduction in the Absolute CPU consumption</th>
</tr>
</thead>
<tbody>
<tr>
<td>1k2</td>
<td>36.76%</td>
</tr>
<tr>
<td>2k4</td>
<td>31.14%</td>
</tr>
<tr>
<td>4k8</td>
<td>21.5%</td>
</tr>
<tr>
<td>9k6</td>
<td>0.7%</td>
</tr>
</tbody>
</table>

Similar progress can be seen in relative performance measures as seen in Figure 5-7. The block `main()`, which was contributing towards 90% of the CPU usage, is now reduced to 40% with hardware decimation at 1k2. Other functions appear to contribute more time as they are the relative measure.

Figure 5-7 Post DQT Profiling Results on ARM Cortex A9 (Relative CPU Time)
The decoder which was unsuccessful in decoding the signals is now 100% successful even at higher date rates, as seen in Figure 5-8. The reduction in the CPU utilisation not only helps in accommodating more signals, but also in ensuring reliable signal decoding. Also, this has helped in solving the issue discussed in section 4.3 regarding the processed response of the receiver functions to gprof. The reduction in the CPU consumption has helped to get the same response to gprof on the ARM Cortex A9 as on the Octa core ARM Cortex A15 and A7 (Figure 4-12), thus making the like-for-like comparison of the receiver blocks possible.

![Figure 5-8 Success Rate Comparison - Post DQT Implementation](image)

Table 5-2 shows the FPGA processor logic utilisation before and after the front-end DQT function was moved to firmware and includes the percentage increase in FPGA utilisation that results. Adding a sample DQT block to the original increased, slightly, the power consumption and the hardware requirements. The total overhead of on-chip power is 1.4% and with a 1-2% increase in memory LUTs and DSP48, which is negligible. This gives confidence to implement the architecture with multiple channels in order to provide a parallel system to receive multiple signals at the same time.
Implementation on the FPGA fabric shows that nearly half of the hardware is still available. Therefore, along with DDCs, the next highly computationally intensive block, that is the Fast Fourier Transform (FFT), which is now in software, can also be implemented in hardware in the future. In this way, the software can accommodate more decoding threads and thus aid parallelisation. Here, we use the Zynq 7020 but in the case of a greater number of signals with higher data rates, a larger FPGA may be selected such as the Zynq 7045.

Table 5-2 Overhead Analysis of DQT Implementation

<table>
<thead>
<tr>
<th>Original Design (Software DDC)</th>
<th>With DQT Block on FPGA</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Power:</strong></td>
<td><strong>Power:</strong></td>
<td><strong>Power:</strong></td>
</tr>
<tr>
<td>- Total On-Chip Power: 2.2 W</td>
<td>- Total On-Chip Power: 2.231 W</td>
<td>- Total On-Chip Power: 1.4%</td>
</tr>
<tr>
<td>- Dynamic Power : 2.03 W</td>
<td>- Dynamic Power : 2.06 W</td>
<td>- Dynamic Power : 1.47%</td>
</tr>
<tr>
<td>- Device Static : 0.17 W</td>
<td>- Device Static : 0.171 W</td>
<td>- Device Static : 0.58%</td>
</tr>
<tr>
<td><strong>Post Implementation:</strong></td>
<td><strong>Post Implementation:</strong></td>
<td><strong>Post Implementation:</strong></td>
</tr>
<tr>
<td>- Flip Flop : 19%</td>
<td>- Flip Flop : 19%</td>
<td>- Flip Flop : 0</td>
</tr>
<tr>
<td>- LUT : 24%</td>
<td>- LUT : 24%</td>
<td>- LUT : 0</td>
</tr>
<tr>
<td>- Memory LUT : 4%</td>
<td>- Memory LUT : 5%</td>
<td>- Memory LUT : 1%</td>
</tr>
<tr>
<td>- I/O : 61%</td>
<td>- I/O : 61%</td>
<td>- I/O : 0</td>
</tr>
<tr>
<td>- BRAM : 6%</td>
<td>- BRAM : 6%</td>
<td>- BRAM : 0</td>
</tr>
<tr>
<td>- DSP48 : 31%</td>
<td>- DSP48 : 33%</td>
<td>- DSP48 : 2%</td>
</tr>
<tr>
<td>- BUFG : 28%</td>
<td>- BUFG : 28%</td>
<td>- BUFG : 0</td>
</tr>
<tr>
<td>- MMCM : 50%</td>
<td>- MMCM : 50%</td>
<td>- MMCM : 0</td>
</tr>
</tbody>
</table>
5.2.3 Cascaded Integrator Comb (CIC) Filter

The CIC filter was chosen over FIR (Finite Impulse Response) filter as CIC filters are hardware-efficient for multirate implementations [200] with structures that use only adders, subtractors and delay elements. Also, for lower decimation rates, as in our case (16), CIC filters are preferred over FIR filters. The CIC filter has a low-pass response that results from filtering an input signal with a cascade of $N$ unit-amplitude, rectangular windows of length $RM$. The magnitude response of the CIC filter is given by Eq. 5-1:

$$H(f) = \left[\frac{\sin(\pi RMf)}{\sin(\pi f)}\right]^N$$

Eq. 5-1.

Where $N =$ number of CIC stages; $R =$ rate change; $M =$ differential delay in the comb stages of the filter, and the nulls in the magnitude response are at integer multiples of $f = 1 / (RM)$. The parameters provide the passband characteristics over the frequency range from zero to a predetermined cut-off frequency $f_c$. This pass band frequency range is typically the bandwidth of interest, occupied by the signal undergoing filtering. The CIC filter response to different configurations of the number of CIC stages ($M$) and differential delay in the comb stages ($N$) can be seen in Figure 5-9.
The larger the number of cascaded stages and differential delays, the more attenuated the magnitude response of the side lobes becomes. The DQT block was replaced by the existing Vivado CIC block in the reference design and the block was configured to different cases as listed in Figure 5-9. The configuration we would to expect to work was $N=5$ and $M=2$ (the red line) and this case was validated by plotting the FFT of the samples collected after hardware decimation, as seen in Figure 5-10(b), which matches the FFT plot of samples as seen in Figure 5-10(a) in software decimation (original implementation). Also, the signal was decoded successfully.
However, the existing CIC Compiler has many drawbacks, including:

1. The input sampling frequency is limited to 6.144 MSPS which means the signal had to be oversampled 4 times more than the Analog Devices’ requirements (1.536 MSPS). The input sampling rate using a CIC filter is constrained to be higher than the AD9361 minimum rate of 1.536 MHz if we want to retain decimation rates between 1 and 16 (this is in order to achieve like-for-like comparison for performance analysis). If we can live with decimation rates between 4 and 16, then the minimum decimation rate of 4 imposed by the CIC LogiCore is not a problem.

2. Also, there was no proper documentation available in order to understand the dependency on the sampling rate and clock frequency at the time of development as the block was relatively new (released on 18th November, 2015 [183]).

3. The complexity was increased when the decimation rate of the CIC block had to be programmed via a complex AXI register.

Since the CIC block provided by Vivado Design Suite is not very reliable for our application and is still in its infancy with limited configuration flexibility, a custom DDC block had to be implemented in order to dynamically re-configurable the parameters to fit different signals as per requirements. Re-configurability is one of the main advantages of a SDR, which in our case supports the concurrent multi-satellite reception through adaption.

### 5.3 Dynamically Reconfigurable DDC

With the emergence of partially and dynamically re-configurable FPGAs, SoC architects are given a new degree of freedom in system level design as these allow multiple applications to time-share a portion of an FPGA, while the rest of the device continues to operate unaffected. While conventional SoCs require the number, type and location of hardware modules to be defined at design time, dynamically reconfigurable FPGAs allow these parameters to be adapted at runtime [184]. This new degree of freedom, however, also raises new issues that need solving. Among these are the handling of dynamic reconfiguration, problems related to online placement of hardware modules, and the question of suitable communication structures. There has been notable research effort addressing these issues [185] when considering applying this to the physical layer processing architecture on SDR systems, however, there has been relatively little research in this context concentrating on DSP processor-FPGA hybrid systems [186, 187] and Network on Chip (NoC) systems [188].
Software Defined Radio (SDR) Architecture for Concurrent Multi-Satellite Communications
Mamatha R. Maheshwarappa

SDR is a technique used to support multiple communication standards and services with a single programmable terminal device. Normally, a SDR platform employs a set of programmable hardware devices, such as a FPGA or DSP processors, to perform different radio functions to meet the requirements of multiple standards, with these radio functions being controlled or defined by software. SDR systems can benefit from dynamic re-configurability due to the reduced complexity and increased design flexibility engendered, as different transceiver applications can be handled by configuring different blocks to different attributes such as frequency, decimation rate and phase.

This research takes advantage of this flexibility to make the DDC block reconfigurable, in order to simultaneously receive multiple signals with varied parameters, such as different modulation techniques, data rates and frequencies, which is the main objective of this research as discussed in Section 1.2.

Figure 5-11 shows the reference design from Analog Devices and the blocks highlighted in red are the ones modified. The samples from the AXI-AD9361 core are decimated by the DDC block before giving them to the packer (util_adc_pack) and the packed samples are stored in the memory for further processing performed on the Zynq-PS.

An Integrated Logic Analyser (ILA) is used for the analysis of the different signals and timing.

Figure 5-12 shows the complete reconfigurable DDC block, which consists of the blocks described in Table 5-3. The benefits of this architecture are primarily that it reduces the CPU consumption as the block is moved to PL from PS providing scope to accommodate concurrent multiple signals on the Zynq SoC, and reduces the number of hardware devices in the physical layer compared to traditional systems involving separate hardware for different signals. Also, this makes the design efficient both in terms of power and cost. The minimum number of channels that the packer accepts is two, and therefore two DDC blocks were integrated between the AXI_AD9361 and util_adc_pack.
Software Defined Radio (SDR) Architecture for Concurrent Multi-Satellite Communications
Mamatha R. Maheshwarappa

Figure 5-11 Modified Reference Design

Figure 5-12 Reconfigurable DDC Block Implemented in FPGA
Table 5-3 Details of the Blocks in Reconfigurable DDC

<table>
<thead>
<tr>
<th>Sl. No</th>
<th>Block Name</th>
<th>Type of the Block</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Digital Quadrature Tuner (DQT)</td>
<td>Custom Block</td>
<td>Down converts the incoming samples to match the decoder from 1.536 MSPS to 96 KSPS.</td>
</tr>
<tr>
<td>2.</td>
<td>Digital Direct Synthesizer (DDS)</td>
<td>Existing Vivado Block</td>
<td>DDS component synthesizes sine and cosine carrier frequencies which modulates the I and Q data. This helps in tuning the frequency to any desired signal in the spectrum and compensate for Doppler shift in the frequency.</td>
</tr>
<tr>
<td>3.</td>
<td>Decimation Rate Tuner</td>
<td>Custom Block</td>
<td>This is a register, which controls the decimation rate (1, 2, 4, 8 and 16) of the DQT block for varying symbol rates in this case 19k2, 9k6, 4k8, 2k4 and 1k2 respectively.</td>
</tr>
<tr>
<td>4.</td>
<td>Phase Tuner</td>
<td>Custom Block</td>
<td>Similar to Decimation Rate Tuner, which provides the phase input to the DDS block and in turn to the complex multiplier.</td>
</tr>
<tr>
<td>5.</td>
<td>Complex multiplier</td>
<td>Existing Vivado Block</td>
<td>This block performs complex multiplication (64 bit) of the DDS output and the incoming data from AD9361 core.</td>
</tr>
<tr>
<td>6.</td>
<td>Broadcaster</td>
<td>Custom Block</td>
<td>Distributes the upper 16 bits ([63:48] and [31:16]) of both the words to I and Q channel of DQT blocks.</td>
</tr>
<tr>
<td>7.</td>
<td>Concat</td>
<td>Existing Vivado Block</td>
<td>Concatenates the I and Q signals from AD9361 core.</td>
</tr>
</tbody>
</table>

The custom blocks such as DQT, Decimation Rate Tuner, Phase Tuner and Broadcaster are explained in detail in the following sections.
5.3.1 Python Software Controller

When a new register is added, the kernel has to be aware of the base address of the register that we want to configure and therefore each time a register is added, the devicetree has to be modified in order to provide the details of the block. However, with the following python script, the base address of the register can be added and configured in software and in real-time. The Pseudo code shown in Figure 5-13 shows the snippet of the Python script used to configure the DDC blocks.

Pseudo code:

Parameters : decimation rate and frequency offset (rfreq)
Initialisation : Registers with base addresses
- Phase bits = 16
- Input sampling frequency Fs = 1.536e6
Computation :
- Open()
  - Calculate ifreq = -(rfreq/Fs) * ((2**phase_bits) / 2)
  - Write the defined decimation rate to the respective register
  - Write the defined frequency offset to the respective register
- close()

Figure 5-13 Pseudo Code of the Python Script

Based on the analysis carried out in Chapter 2 on CubeSat standards, the most popular data rates are 1k2, 2k4, 4k8, 9k6 and 19k2, and the most common frequency bands used are mainly the VHF, UHF and S-bands. Therefore, decimation rates varying from 16 (1k2) to 1 (19k2) were tested.

The phase calculations are given by Eq. 5-2:

\[
ifreq = \left( \frac{rfreq}{Fs} \right) \times \left( \frac{2^{\text{phase bits}}}{2} \right) \quad \text{Eq. 5-2.}
\]
Software Defined Radio (SDR) Architecture for Concurrent Multi-Satellite Communications
Mamatha R. Maheshwarappa

Where \( ifreq \) is increment frequency, \( rfreq1 \) is the frequency offset of Signal 1, \( Fs \) is the input sampling frequency (1.536 MSPS) and the Phase bits are 16.

5.3.2 Decimation Control

As discussed in Chapter 3 and 4, the decimation was previously carried out mainly in software and the analysis using gprof suggested that it was consuming the highest CPU capacity and therefore it was decided to move it to the FPGA. It is important to analyse the improvement stage by stage and to be able to accommodate all the CubeSat signals varying from 1200 bps to 19200 bps. Therefore, there is a need for a reconfigurable decimation block.

An AXI register that controls the decimation rate of the DQT block was implemented. Valid lines play a very important role, as discussed earlier in Section 5.2.1, in the entire reference design. Figure 5-15 shows how the decimation rate changes due to difference in the time at which the valid lines are asserted.

The packer block from Analog Devices performs logical-OR on the valid lines from DQT-I and DQT-Q of the different channels. Therefore, the falling edge of all the DDC blocks should occur at the same time (highlighted in Figure 5-15) – this was achieved by changing the process statement in VHDL inside the DQT block to ‘rising_edge’ of the clock rather than just ‘clk’. Also, the change in decimation rate from 2 to 4 can be noticed in Figure 5-16, which was done real time. This illustrated that these blocks can be made reconfigurable in real-time to suit the requirements. This was one of the main objectives of this work as discussed in Section 1.2.

5.3.3 Phase Control

Similar to decimation control is the phase control block, where the block provides the frequency offset of a signal to the DDS block that modulates the I and Q data, based on the input frequency. The output of the DDS block is complex multiplied with I and Q signals from the AXI_AD9361 core, thus giving 64 bits output. 16 upper bits of both the words are provided to the DQT-I and DQT-Q blocks respectively to down-sample the incoming samples.

By way of example, where the required frequency offset is 1KHz, the increment frequency \( (ifreq) \) is calculated by Eq. 5-3:

\[
ifreq = \left(\frac{rfreq}{Fs}\right) \times \left(\frac{2^{phase\ bits}}{2}\right)
\]

\[ Eq.5-3. \]
Software Defined Radio (SDR) Architecture for Concurrent Multi-Satellite Communications
Mamatha R. Maheshwarappa

\[ ifreq = \left( \frac{1000}{1.536 \times 10^6} \right) x \left( \frac{2^{16}}{2} \right) \]

\[ ifreq = 21.333 \text{ Hz} \]

Where \( ifreq \) is increment frequency, \( rfreq \) is the frequency offset of a signal, \( Fs \) is the input sampling frequency (1.536 MSPS) and the Phase bits is 16 bits in this case.

5.3.4 Broadcaster

The output of the complex multiplier was 64 bits, which had to be divided between the DQT-I and DQT-Q blocks which are 16-bit input blocks. Therefore, the upper 16 bits of both the words were chosen. Other combinations can be tried to get different signal levels and this is another parameter that can be made programmable in the future.

The existing broadcaster block in Vivado was primitive and it was compatible only with Vivado 2015.3 but the reference design from Analog Devices was released for Vivado 2014.2 at the time this work was taken up. The block had compatibility issues and the valid lines were not propagated as desired. The valid lines had different frequencies varied over different clock cycles and thus the packer block was not able to pack the incoming samples appropriately. In order to resolve this issue, a custom block was created in Vivado 2014.2. Figure 5-14 shows the pseudo-code for the broadcaster block.

**Pseudo code:**

```
Parameters : Input data = 64 bits, Output data = 16 bits
Valid line, Clk
Initialisation : input_data_width = 64
               Output data width = 16
               Counter = 0
Computation :
begin
  Assign the upper 16 bits of the 1\textsuperscript{st} word (input) to output_1
  Assign the upper 16 bits of the 2\textsuperscript{nd} work (input) to output_2
end
```

**Figure 5-14 Pseudo Code of Broadcaster Block**
**Figure 5-15**: Invalid Decimation Rate due to Difference in the Time at which Valid Lines are Asserted
Figure 5-16 Valid Decimation Rate and Real Time Decimation Rate Switching (from 2 to 4)
5.3.5 Overhead Analysis

Table 5-4 shows the second stage analysis of the FPGA processor logic utilisation before and after the reconfigurable DDC function was implemented. Adding a reconfigurable DDC block to the original reference design increased the power consumption and the hardware requirements. The total overhead of on-chip power is 13.14% with a 5% increase in flip-flops and memory LUTs, 8% increase in LUTs, 18% increase in block-RAM (BRAM) and 3% increase in DSP blocks. This analysis suggests that more DDC/DUC blocks could be implemented in order to aid parallel reception. Here, we use the Zynq 7020 but in the case of a greater number of signals with higher data rates, a larger FPGA such as the Zynq 7045 may be selected.

Table 5-4 Overhead Analysis of Re-configurable DDC Block Implementation

<table>
<thead>
<tr>
<th>Power:</th>
<th>With DDC Block on FPGA</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total On-Chip Power: 2.2 W</td>
<td>Total On-Chip Power: 2.489 W</td>
<td>Overhead: 13.14%</td>
</tr>
<tr>
<td>Dynamic Power: 2.03 W</td>
<td>Dynamic Power: 2.309 W</td>
<td>Overhead: 13.74%</td>
</tr>
<tr>
<td>Device Static: 0.17 W</td>
<td>Device Static: 0.180 W</td>
<td>Overhead: 5.88%</td>
</tr>
</tbody>
</table>

Post Implementation:

<table>
<thead>
<tr>
<th>Power:</th>
<th>With DDC Block on FPGA</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flip Flop: 19%</td>
<td>Flip Flop: 24%</td>
<td>Overhead: 5%</td>
</tr>
<tr>
<td>LUT: 24%</td>
<td>LUT: 32%</td>
<td>Overhead: 8%</td>
</tr>
<tr>
<td>Memory LUT: 04%</td>
<td>Memory LUT: 09%</td>
<td>Overhead: 5%</td>
</tr>
<tr>
<td>I/O: 61%</td>
<td>I/O: 61%</td>
<td>Overhead: 0%</td>
</tr>
<tr>
<td>BRAM: 06%</td>
<td>BRAM: 24%</td>
<td>Overhead: 18%</td>
</tr>
<tr>
<td>DSP48: 31%</td>
<td>DSP48: 34%</td>
<td>Overhead: 03%</td>
</tr>
<tr>
<td>BUFG: 28%</td>
<td>BUFG: 28%</td>
<td>Overhead: 0%</td>
</tr>
<tr>
<td>MMCM: 50%</td>
<td>MMCM: 50%</td>
<td>Overhead: 0%</td>
</tr>
</tbody>
</table>

FPGA Implementation
5.4 Summary

This chapter has focused on improving the resources available by taking advantage of the flexibility and speed of FPGAs. The implementation was started with a simple DQT block to investigate the complexity involved in integrating a custom block into the available reference design. The port implementation results indicate the radical improvement that can be achieved on FPGAs. The significant improvement being at lower data rates such as 36.76% at 1k2, 31.14% at 2k4 and 21.5% at 4k8. This comes at a cost of 1.4% more on-chip power and 1-2% increase in on-chip resources. Therefore, it has been concluded that for this reference design, moving the Front End DDC function alone, from software to firmware, is sufficient to allow concurrent multiple satellite reception at typical CubeSat telemetry rates.

Further, the DDC block was made reconfigurable so that the real-time configuration needs can be upgraded dynamically according to the requirements. The implementation carried out so far allows the data rate and frequency to be changed in real-time. Other metrics such as adding/removing the filters and increasing the number of DDC blocks within the fabric can be achieved along similar lines. The idea here is to demonstrate the technology of reconfigurability within the limited resources available in an embedded system. This was achieved with 13.14% more on-chip power and 5-18% on-chip resource.

In conclusion, the analysis in section 5.3.5 shows that multiple signals can be received and decoded simultaneously, which takes us to the next chapter.
6 Parallel Architecture

The ability to support multiple concurrent communication channels per RF band is a fundamental aim of this thesis. Such SDR platforms that can achieve this, typically employ a channeliser (reviewed in section 6.1) to extract channels from the received RF band for follow-on baseband processing, and/or to insert channels into the RF band for transmission. This chapter elaborates on a new SoC solution for parallel reception to support concurrent multiple satellite signals, as described in section 6.2. Implementation and validation of the proposed parallel architecture is discussed in section 6.3, along with different case studies. Auto configuration of the parallel architecture is proposed in section 6.4 in order to dynamically match the configuration parameters to the spectrum. The chapter is summarised in section 6.5.

6.1 Channelisation

Channelisation is a process whereby the received signal bandwidth is divided into sub-bands. The three predominant channelisation architectures used in digital communications systems are Digital Down Conversion (DDC), Frequency Domain Filtering, and Polyphase FFT Filter Banks [189].

6.1.1 Digital Down Converter (DDC):

In this technique, the wideband signal is mixed with a synthesized carrier, at or near the carrier frequency of the channel of interest to baseband the channel. The resulting signal is then filtered and decimated to isolate and extract the channel of interest from the wideband signal and reduce the overall sample rate to the minimum necessary to support the channel.
Depending on the bandwidth of the channel of interest, one or two different filtering and decimation techniques are employed [190]. For wider bandwidth signals, requiring decimations of 8 or less, a wideband FIR filter is used directly following the baseband mixer. The output of this filter is then decimated by an appropriate amount. For narrow band signals, the filtering and decimation functions are typically split into multiple stages. The initial stage is provided by a Hogenauer filter, also known as a CIC filter, which provides for reasonable first stage channel isolation, while minimising the number of complex operations that must be performed prior to decimation. Follow on filtering is typically then provided by a narrowband FIR filter, which would typically have programmable taps to allow this filtering to be optimised for the channel of interest.

The primary advantages of the DDC techniques are the flexibility in selecting both the carrier frequency and channel bandwidth. However, for complex channel structures, where both wideband and narrowband channels may occupy the same input signal, a mix of down converter technologies may be required, complicating the architecture of the channeliser block.

6.1.2 Frequency Domain Filtering (FDF)

The FDF approach makes use of the properties of the Fast Fourier Transform (FFT) to simplify the baseband conversion, filtering, and decimation functions identified in the DDC approach. In this technique, input data is buffered into overlapping blocks, with an FFT performed on each of the blocks [191].

Using the FDF channelisation approach, a large number of both wideband and narrowband operations can coexist in the same channeliser structure, providing for improved flexibility and higher channel density than is the case with the DDC technique. However, to make optimal use of this capability, a programmable device such as a DSP processor or FPGA must be used for the baseband channelisation processing to support the dynamic loading of baseband filtering and inverse FFT components of various sizes on a per channel basis.

6.1.3 Polyphase FFT Filter Bank

The Polyphase FFT Filter Bank (PFFB) channeliser improves upon the efficiency of the frequency domain filtering technique by assuming redundancy within the frequency plan of the wideband channel [192]. This structure makes use of a polyphase filter to isolate and decimate the various channels, and then employs an FFT to efficiently convert each channel to baseband. Although this technique is limited to channel structures consisting of equally spaced channels, it is extremely efficient, requiring only a single FIR filtering structure and a small FFT, with the FFT typically driving the complexity of the overall channeliser.
If redundancy exists in the channel structure, then the polyphase FFT filter bank appears to be the most efficient choice, and if flexibility is required, then the FDF approach seems to make the most sense. The channeliser in this application needs to be flexible enough to accommodate all of the carrier/bandwidth combinations supported by the network architecture, and possibly allow for the dynamic reallocation of channel resources within this architecture during operation. FDF and DDC channelisation techniques offer similar capability in terms of flexibility for any type of channel spacing and bandwidth, but the FDF approach is useful for both narrow and wide band channels. Also, the mixing operation is performed on the block of data vs. continuous time processing, which makes zero carrier offsets for individual blocks of data, creating a rotating phase offset between each block, if the carrier cycle at the ADC sample rate is not an integer number of block size.

Therefore, a new architecture, which is flexible, suitable for both narrow and wide band signals with different standards, is proposed in the next section. This is a combination of the discussed methods. This approach consists of different stages in order to understand the challenges involved, and its pros and cons.

6.2 Parallel Configuration

The selection of SDR technology over a standard hardware radio is primarily due to the need for flexibility and re-configurability. As an alternative to building multiple hardware demodulators and decoders matching the existing CubeSat communication standards and then having to physically upgrade the system for new spacecraft, the upgrade can now be accomplished in the software through a SDR. This could be done in one of the following stages:

6.2.1 Parallel Architecture – Alternative 1

Figure 6-2 shows the first alternative of the parallel architecture, which is combination of the FDF and DDC methods discussed earlier. This reference design is optimised for testing ADC’s. This includes testing single process and cooperative threads tasking of multiple demodulators.

The objective of the parallel architecture is to receive the signal from a scenario as depicted in Figure 6-1, with more than one satellite, each with different modulation techniques, data rates and centre frequencies. To acquire the desired signal present in the spectrum, an asynchronous approach is proposed, with a software FFT to scan the spectrum, and software scripts that will define the configuration of the frontend such as gain, filters, bandwidth and centre frequency. Once the signal is detected, the DDC is configured to a desired decimation rate and centre frequency.
The next stage in the architecture is a synchronous process with parallel wrappers consisting of DDC blocks with respective registers to configure the blocks on firmware. Each valid signal in the spectrum is mapped to a separate wrapper, based on the available on-chip resources.

Here, the multiple chains of DDCs are included within one synchronous domain for simplicity. However, the Vivado valid_in/valid_out approach does allow a hybrid sync/async design paradigm.

Each signal is stored at different offset addresses in memory, accessed by DMA, based on the RF band centre frequency, channel plan and prevailing Doppler offset. The last stage is proposed to be asynchronous, as the signal stored in memory can be accessed independently by the decoder thread performing co-operative tasking of multiple threads running on dual core processors.

Using a single programmable baseband SoC to execute several baseband processing programs at the same time, can produce benefits in terms of increased hardware reuse, shared software kernel functions and use of shared information, such as link state and channel parameters.

However, in order to avoid data loss, dropped packets or frames, the processor must have enough resources to support the worst case load in all supported standards simultaneously. Therefore, here is the technology that is tested on basic Zynq 7020 but in case of a greater number of signals with higher data rates, a larger FPGA with more LUTs and DSP blocks could be used (e.g. the Zynq 7045).
6.2.2 Parallel Architecture - Alternative 2 Refinement

The initial asynchronous part, including the FFT and hardware configurable software scripts, remain the same throughout the different alternatives.

The 2nd architecture has extension to support higher rates along with the ease of use supported by alternative 1 architecture. This includes separate threads for separate processes as seen in Figure 6-3. The aim here is mainly performance improvement within an envelope of low power resources. This architecture includes a software demultiplexer (demuxer) to route the signal to different decoders. The performance could be better as separate threads handle the signals separately, and if possible, these threads can be run on different cores to increase the efficiency. However, this needs additional effort to replace the packer block with a simple multiplexer and to deal with issues related to valid lines, which would make the design much simpler.
6.2.3 Parallel Architecture - Alternative 3 Refinement

The alternative 3 architecture, as seen in Figure 6-4, provides better extension to support, ease of use and reliability within a defined performance envelope for CubeSat applications.

As the CubeSat data rates are very low (1k2, 2k4, 4k8, 9k6 and 19k2 bps) extendable block RAM (560 KB) is sufficient to store 2180 demodulated signals at 1k2 and 136 demodulated signals at 19k2 (refer to Section 3.4.6) and therefore DMA is not needed. This includes writing data directly to a memory without involving a multiplexer/packer, which make the design more reliable. The goal here is software reliability/scalability and performance improvement. Also, most of the work is done in firmware which means the efficiency is increased and thus, it can receive and decode more signals. More demodulator chains can be run on dual ARM Cortex A9 processors.
6.2.4 Parallel Architecture - Alternative 4 Refinement

The alternative 4 architecture, as seen in Figure 6-5, includes all the objectives from the previous alternatives, such as ease of use, performance, software reliability/scalability and higher data rates. Though the architecture looks really simple, both the DDC and demodulator blocks have to be really intelligent to read and write the data to and from the DMA system using multiple instances of the DMA engine. Since there is no hardware and software multiplexer. Instead, the demodulator threads should analyse which one is busy and assign the next signal to the thread that is free.
Every alternative discussed in this section has its advantages and disadvantages, as depicted in Table 6-1. This is a qualitative analysis based on the experience gained through software and firmware implementations carried out earlier in Chapter 3 and Chapter 4. However, a quantitative comparison can only be provided with a specific implementation. The implementation of different alternatives of the proposed parallel architectures includes the following issues:

1. Time – the implementations come with several engineering issues, which are time consuming.
2. Validation – the validation of all different standards such as symbol rate/modulation technique/frequency/coding scheme is highly challenging as there is vast variety of CubeSat signals (discussed in Section 2.1.2).
3. Expertise – the implementation of Alternative 2 and Alternative 3 parallel architectures requires the reference design to be changed greatly which demands greater experience.

<table>
<thead>
<tr>
<th>Alternative 1</th>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Fewer modifications in the reference design.</td>
<td>• Involves packer block which is not reliable and issues related with valid lines.</td>
</tr>
<tr>
<td></td>
<td>• DMA can be accessed by different decoder threads independently.</td>
<td>• Packers limits the number of channels that can be handled to 16.</td>
</tr>
<tr>
<td>Alternative 2</td>
<td>• Solves issues related to packer block and valid lines.</td>
<td>• Requires an intelligent multiplexer block implemented in firmware.</td>
</tr>
<tr>
<td></td>
<td>• Supports higher number of channels and symbol rates.</td>
<td>• Software multiplexer has to be capable of mapping demodulator threads to appropriate memory location.</td>
</tr>
<tr>
<td>Alternative 3</td>
<td>• Every signal has a dedicated memory and therefore easy to access.</td>
<td>• Since the memory is fixed, signal with higher data rates have a limitation.</td>
</tr>
<tr>
<td></td>
<td>• Less complex in terms of firmware implementation.</td>
<td>• Firmware implementation is complex with AXI register and</td>
</tr>
<tr>
<td>Alternative 4</td>
<td>memory handling.</td>
<td>FIFO blocks.</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------</td>
<td>--------------</td>
</tr>
<tr>
<td>- Requires less firmware resources as there are fewer blocks.</td>
<td>- DDC blocks and demodulator threads have to intelligent to analyse the bus and assign the signals.</td>
<td></td>
</tr>
<tr>
<td>- Can accommodate more number of DDC blocks and therefore more number of channels.</td>
<td>- Again, since the memory is fixed/pre-assigned, signal with higher data rates have a limitation.</td>
<td></td>
</tr>
</tbody>
</table>

Due to these constraints, alternative 1 is implemented and tested with different transceiver standards such as frequency and symbol rate, and alternatives 2, 3 and 4 architectures are not implemented during this research. However, these architectures can be considered as part of the future work based on this thesis.

6.3 Implementation and Validation of Parallel Architecture – Alternative 1

The alternative 1 architecture, described in section 6.2.1, was implemented and validated with different test cases. Two setups comprising the same hardware (Zynq + AD-FMCOMMS3-EBZ) were used, as shown in Figure 6-7, but different transceiver standards were generated using python scripts:

1. To change the frequency in real time to simulate Doppler effect.
2. To change the data rates and frequency in real time in order to simulate signals from different satellites.

The Pseudo code in Figure 6-6 describes the python script:

```
Pseudo code:

Parameters   :  decimation rate and frequency offset (rfreq)
Computation :
Open()
Calculate ifreq = - (rfreq/Fs) * ((2**phase_bits) / 2)
Write the defined decimation rate to the respective register
Write the defined frequency offset to the respective register
Close()
```

Figure 6-6 Pseudo Code of the Python Script used to change the Transceiver Standards

Parallel Architecture
6.3.1 Emulation of Two Signals

Different combination of signals were transmitted from two different boards as it would saturate the processor if both the signals were transmitted from a single board and Digital Up Converters are not implemented in the FPGA as this thesis mainly concentrates on the receive chain. The signals from two these transmitters were received initially on the SDR Sharp to test the transmission. The different combinations are:

1. Two signals at different symbol rates and at different centre frequencies without Doppler effect. Figure 6-8 (a) shows symbol rates of 1k2 and 2k4, and Figure 6-8 (b) shows symbol rates of 1k2 and 19k2.
2. Two signals with Doppler effect varying at 50 Hz/s and centred at different frequencies. Figure 6-9 (a) shows the Doppler effect at 1k2 and 2k4 where the rate of change of frequency is prominent as the bandwidth is smaller when compared to Figure 6-9 (b) where the drift is less prominent due to larger bandwidth.

In reality, an open-source satellite tracking and orbital prediction program, such as “predict”, can be integrated, to acquire the location and direction information of the satellites and to therefore obtain the Doppler information. The receiver, on the other hand, was able to track the frequency offset accurately and decode the signals.
Software Defined Radio (SDR) Architecture for Concurrent Multi-Satellite Communications
Mamatha R. Maheshwarappa

Figure 6-8 Two Signals at Different Data Rates & Frequencies (a) 1k2 and 2k4 (b) 1k2 and 19k2
(Date: 20/09/2015, Time: 15:32:00, X-axis: 10 KHz, Y-axis: 10 dB)

Figure 6-9 Two Signals at Different Data Rates and Frequencies (moving at 50Hz/s) (a) 1k2 and 2k4 (b) 1k2 and 19k2 (Date: 20/09/2015, Time: 15:00:00, X-axis: 10 KHz, Y-axis: 10 dB)
The algorithm implemented on the receiver for concurrent multiple signal reception includes:

1. Software scripts, which will configure the front-end such as gain, filters and bandwidth.
2. Software FFT, which scans the spectrum and finds the signal(s).
3. Each signal is fed into a different DDC block depending on the configuration (discussed in 6.3.1.1).
4. Each DDC block has a register, which can be configured using a Python script (Figure 6-6).
5. The decimated signal is stored at different offset addresses using DMA.
6. The memory is accessed by different demodulator threads, which extract the signal around the chosen centre frequency.

The demodulator could be different for different satellite signals, but in this case, the same FUNcube -1 modulation/coding technique is used, in order to reduce the time in implementing various standards. However, standards such as frequency, symbol rate were changed to test the receiver's adaptability, along with Doppler effect. The received signals were decoded on a single board using two different configurations.

6.3.1.1 Configuration 1: 2 x DDC > 2 x Rx

Configuration 1 is shown in Figure 6-10, where the signals from both the transmitters (1st signal has an offset of 10 kHz offset, while the 2nd signal has 20 kHz offset) are connected to a combiner, then the signal is transmitted via a coaxial cable to a splitter. The output from the splitter is connected to two Rx ports of the same board. These ports are connected to individual DDC blocks within the firmware, where the signal is decimated for further processing.

Figure 6-10 Setup of Case 1: 2 x DDCs connected to 2 x Rx Ports
6.3.1.2 Configuration 2: 1 x DDC > 2 x Rx

Figure 6-11 shows the 2nd configuration where the output of the combiner is connected to one of the Rx ports on the AD-FMCOMMS3-EBZ, which in-turn is connected to 2 different DDC blocks within the firmware.

![Figure 6-11 Setup of Case 1: 2 x DDCs connected to single Rx Port]

Figure 6-12 shows successful operation of the parallel system at different rates and frequencies. Freq_off clearly indicates two signals; one of which has an offset of 10 kHz and second signal has 20 kHz offset and both have different messages (Satellite – 1 and Satellite - 2).
Figure 6-12 Transmitter - 2 Signals Decoded (Date: 24/09/2015, Time: 11:50:00)
The following steps were carried out in order to incorporate two demodulators in a single thread:

1. Separate buffers were created to store the decimated data.
2. FFT in software was included only for Case -1 : 2 x DDC > 2 x Rx.
3. For Case 2: 1 x DDC > 2 x Rx, software FFT was taken out due to the simplistic nature of this implementation based on assumption of one stable maximum in a given window.
4. Therefore, the centre frequency of the receiver had to be accurate in order to decode both the signals.

However, this can be overcome with alternative 2, 3 or 4 parallel architectures, as they have separate demodulator threads.

**6.4 General Results - Discussion**

As discussed earlier in Section 3.4.4, the minimum input sampling rate of the AD9361 RF front-end is 1.536 MSPS. In order to see the worst-case requirements, the entire spectrum (1.536 MHz) is assumed to be occupied by available signals. For the reference signal chosen, the bandwidth required at different symbol rates is depicted in Table 6-2.

<table>
<thead>
<tr>
<th>Symbol Rate (bps)</th>
<th>Bandwidth Required (Hz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>1800</td>
</tr>
<tr>
<td>2400</td>
<td>3600</td>
</tr>
<tr>
<td>4800</td>
<td>7200</td>
</tr>
<tr>
<td>9600</td>
<td>14400</td>
</tr>
<tr>
<td>19200</td>
<td>28800</td>
</tr>
</tbody>
</table>

As satellites in LEO are moving at ~7.5 km/s, the signal received at the ground station experiences the Doppler effect. Therefore, the signal bandwidth needs to account for the Doppler effect - which is +/- 2kHz from the centre frequency for VHF frequencies at LEO [193].

Figure 6-13 shows the number of signals that can be accommodated in 1.536 MHz with and without Doppler. At 1k2, as many as 850 satellites can fit into 1.536 MHz without taking into account the Doppler effect. However, as the symbol rate increases, the number of signals that
Software Defined Radio (SDR) Architecture for Concurrent Multi-Satellite Communications
Mamatha R. Maheshwarappa

can be accommodated decreases. The number decreases further when considering the Doppler effect.

![Graph showing analysis of signals at different rates](image_url)

**Figure 6-13 Analysis of Signals at Different Rates that can be accommodated in 1.536 MHz**

### 6.4.1 Memory Requirement

Based on the analysis of number of signals that can be housed in a spectrum of 1.536 MHz, the memory required to store these signals is calculated at different stages as listed:

1. **After the hardware decimation** – the hardware decimation rate varies depending on the symbol rate (for 1k2 signal the decimation rate is 16 while for 19k2 it is 1).
2. **Software decimation** – this is done in order to match Alex Csete’s decoder (Decimation rate is 10).
3. **Complex down conversion to remove residual 1200 Hz offset** (Decimation rate is 8).
4. **Demodulator data for decoding**.

The initial memory required to store 1.536 MS (16 bit I + 16 bit Q) for 5s is 30.72 MB which exceeds the on-chip memory (256 KB + 560 KB of extendable block RAM) and therefore the signal is stored in an external memory. However, after decimation the memory required to store 1k2 signal is significantly less (1.92 MB). There is an increase in the memory required as the
Software Defined Radio (SDR) Architecture for Concurrent Multi-Satellite Communications
Mamatha R. Maheshwarappa

Symbol rate increases as seen in Figure 6-14, especially after the hardware and software decimation. It is important to note that after the 2nd level of decimation the memory required is considerably less, hence the data can be stored in on-chip memory. Also, this allows the storage of more than one signal at a time in the on-chip FPGA BRAM.

![Figure 6-14 Memory Required for Each Signal at Different Symbol Rates and at Different Stages](image)

Now that we have the total number of signals that can be accommodated in the spectrum (Figure 6-13) and the memory required for each signal at different symbol rates (Figure 6-14), we can find the memory requirement of these signals if they were to be stored on Zedboard.

These numbers are obtained by multiplying the number of signals as in Figure 6-13 with the respective memory requirement, as seen in Figure 6-14. The memory required increases as the symbol rate increases and at 19k2 it reaches 1.443 GB as seen in Figure 6-15 (which needs all the memory available on Zedboard – 256 KB on-chip + 560 KB extendable block RAM and 1 GB external memory). Therefore, any further increase in the symbol rate, or the number of signals, would require another board. It can be concluded that, for the chosen scenario, the memory requirement is not a bottleneck for the concurrent multi-satellite reception on the chosen hardware.
Having considered the memory required, the next step is to understand the memory throughput. The worst case condition here is writing to and reading from the external memory. It takes 7.3ns with data access up to 64 bit width to write/read from external memory i.e., 28.032 ms to write/read 30.72 MB data. Figure 6-16 shows the analysis of the time taken to write the complete data generated, as in Figure 6-15, to an external memory.

However, the rate at which the data is coming in is 1.536 MSPS which requires 6.144 MB/s of memory and if this data has to be stored in an external memory, which is the worst case, it takes about 5.606 ms to write/read on a single channel but the Zynq 7020 chip has 8-channel DMA with 4 channels dedicated to parallel loading (PL) that supports multiple transfer types. Therefore, the rate at which the data is coming in can be handled by the hardware in parallel.

The extreme case is at 19k2, which takes about 1.317 s for the data transfer, as seen in Figure 6-16, and could be shared between 8 channels which will take 164.6 μs per channel. The parallel architecture needs to take care of this access time in the design. So far, with 4 DDCs, memory handling was not an issue, but nonetheless, it is worth considering these numbers in any future design.
6.4.3 FPGA Sizing Requirements

Another constraint, along with memory, is the FPGA fabric and power consumption, as CubeSats are limited in power if we are considering space application. Table 6-3 shows the third stage analysis of the FPGA processor logic utilisation before and after the 4 reconfigurable DDC functions were implemented. Adding 4 reconfigurable DDC blocks to the original reference design, increased the power consumption and the hardware requirements further.

The total overhead of on-chip power is 23.14% with 7% increase in flip-flops, 10% increase in LUTs, 5% increase in memory LUTs, 29% increase in BRAMs and 5% increase in DSP blocks.

It is interesting to see the number of block RAMs increasing faster than the other metrics, with the increase in the number of DDCs, as shown in Figure 6-16. Therefore, BRAMs will be one of the first bottlenecks for this design, and we may have to use the external memory. On average, there is an increase of 15% BRAM for every 2 DDC blocks added, which suggests that the FPGA can accommodate 8-10 additional DDC blocks within the FPGA. However, if the data is routed to the external memory instead of using BRAM, then up to 32 DDC blocks can be accommodated, as the increase in the flip-flops, LUTs and DSP48 is 4-5% for every 2 DDC blocks added.

In conclusion, the FPGA fabric would be the bottleneck for this design. In order to solve this, a bigger FPGA or more than one Zynq chip (around 7 to fit 264 signals) could be used.
Table 6-3 Overhead Analysis after 4 DDC Blocks

<table>
<thead>
<tr>
<th>Power</th>
<th>Original Design (Software DDC)</th>
<th>With 4 DDC Blocks on FPGA</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Power: 2.2 W</td>
<td>Power: 2.709 W</td>
<td>Power: 23.14%</td>
</tr>
<tr>
<td>Total On-Chip Power</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dynamic Power</td>
<td>: 2.03 W</td>
<td>Dynamic Power: 2.521 W</td>
<td>Dynamic Power: 24.18%</td>
</tr>
<tr>
<td>Device Static</td>
<td>: 0.17 W</td>
<td>Device Static: 0.188 W</td>
<td>Device Static: 10.58%</td>
</tr>
</tbody>
</table>

Post Implementation:

<table>
<thead>
<tr>
<th></th>
<th>Original Design (Software DDC)</th>
<th>With 4 DDC Blocks on FPGA</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Flip Flop: 19%</td>
<td>Flip Flop: 26%</td>
<td>Flip Flop: 07%</td>
</tr>
<tr>
<td></td>
<td>LUT: 24%</td>
<td>LUT: 34%</td>
<td>LUT: 10%</td>
</tr>
<tr>
<td></td>
<td>Memory LUT: 0.04%</td>
<td>Memory LUT: 0.09%</td>
<td>Memory LUT: 0.05%</td>
</tr>
<tr>
<td></td>
<td>I/O: 61%</td>
<td>I/O: 61%</td>
<td>I/O: 0%</td>
</tr>
<tr>
<td></td>
<td>BRAM: 0.06%</td>
<td>BRAM: 0.35%</td>
<td>BRAM: 0.29%</td>
</tr>
<tr>
<td></td>
<td>DSP48: 0.31%</td>
<td>DSP48: 0.36%</td>
<td>DSP48: 0.05%</td>
</tr>
<tr>
<td></td>
<td>BUFG: 0.28%</td>
<td>BUFG: 0.28%</td>
<td>BUFG: 0%</td>
</tr>
<tr>
<td></td>
<td>MMCM: 0.50%</td>
<td>MMCM: 0.50%</td>
<td>MMCM: 0%</td>
</tr>
</tbody>
</table>

There is a 10% increase in the power consumption, which is less concerning for ground station applications, but would induce significant effect on a CubeSat. So the power budget on a CubeSat would limit the number of DDCs that could be implemented on the FPGA - the overall consumption with 4 DDC blocks is 2.709 W.
6.5 Auto Configuring

Having understood the resources available, the next step as part of the future work is to automate and dynamically match the channel conditions through following steps:

6.5.1 Case 1: Signal Unknown

1. Scan the available spectrum using a FFT: The algorithm should be able to scan a wide spectrum in order to find multiple satellite signals at a time.
2. Locate valid signals: The next step would be to locate the right energy levels of valid signals and record the centre frequency.
3. Create a channel that attempts to match the spectral shape: Once the energy is detected, then the filters have to match the spectral shape of the signal and not lose any valid information.
4. Perform modulation recognition: Modulation recognition is a very important step to know if the signal is of interest and/or we have decoder chain available. The decision can be made based on the spectral shape and bandwidth.
5. Demodulate the signal if the database has the desired demodulator thread.

6. Adjust the configuration to keep bit error rate at a given threshold, as discussed in Section 3.4.3.

6.5.2 Case 2: Signal of Interest

The proposed algorithm aims at allowing access to concurrent multiple-reception after the signal of interest is converted to a complex baseband:

1. Extract the Acquisition of Signal (AOS) information of various satellites communication systems in the frequency range: 70 MHz to 6.0 GHz from “predict” at any given time.

2. Receive the complex signal from the AD9361.

3. Perform a FFT on the signal received to find the FFT peak or peaks.

4. If the difference between the FFT peak and the noise level is greater than or equal to a defined threshold:
   a. Perform signal-power estimation.
   b. Record the data (signal power, magnitude of each FFT bin used in the power estimation, and the local time)
   c. Store it in the memory allotted, based on the data rate and the bandwidth.
   d. Perform low-speed signal processing tasks such as modulation/demodulation and packet handling.

This algorithm is described for one of the signals in the spectrum. Since the high-speed signal processing blocks are handled by the FPGA, the processor can accommodate more than one encoder/decoder at a time (the number of signals will be decided once the performance test is performed). While the process is being carried out, the next set of signals will be detected by performing a FFT, and will be stored in the memory (FIFO) for further processing. Both of these cases are illustrated in the flowchart shown in Figure 6-18:
Front end changes include improving the LNA gain and voltage levels.

\[ \text{Eb/No} \quad \text{as per Section 3.4.3 and Figure 3-14} \]

Signal of interest is the one in the catalogue stored onboard.

Valid signal – not present in the catalogue but has SNR > 10 dB

Hardware includes flip-flops, LUTs and DSP48 on the FPGA.

---

**Figure 6-18 Flowchart for Auto Configuring Concurrent Multi-Satellite Reception**
6.6 Summary

This chapter proposes novel parallel architectures, with different alternatives of evolution, that can support concurrent multi-satellite reception after analysing the different channelisation methods available. The implementation investigated in detail in this thesis is limited to one alternative, due to time constraints, and this design is validated with two signals with varied data rates and frequencies, including the Doppler effect. In the field of CubeSats, the tracking/identification/decoding of dissimilar signals at the same time, and demonstrating this on an embedded platform has been achieved for the first time.

A scenario is examined, where the spectrum is anticipated to be packed with signals of different standards and an analysis has been carried out to understand if there are any bottlenecks with regards to hardware, memory and throughput requirements. The analysis shows that there is on average a 10% increase in the power consumption, 15% increase in BRAM, 4% increase in the flip flops, 5% increase in the LUTs and 2% increase in the DSP48 for every two DDC blocks implemented. Therefore, upto 8-10 DDCs blocks could be accommodated on the Zedboard. However, if external memory is used, then up to 36 DDC blocks could be accommodated on one Zynq SoC.

In order to accommodate a greater number of signals, a bigger FPGA or more Zynq SoCs could be used. It is estimated that 7 would be sufficient to accommodate 264 signals. Also, the memory analysis suggests that there is enough room to accommodate all the signals in the spectrum (including on-chip and external memory) – assuming typical CubeSat communication techniques. The 8 parallel DMA channels provide throughput, which is adequate to meet our design requirements for the symbol rates discussed in this research. The throughput analysis would form a baseline for any future design involving higher data/symbol rates.

Finally, as part of the future work, the auto configuration of concurrent multiple satellite signals is proposed, based on the requirement analysis carried out earlier.
7 CONCLUSIONS

SDR is a technology in fast evolution and which is receiving great interest in the space industry. It can be used to develop a reconfigurable radio system, where the parameters are selected in a dynamic manner, providing the freedom to change the parameters on the ground, as well as on-board a satellite, even after the launch, where we are focusing mainly on amateur radio/university class small satellites.

We can predict that SDR systems will be heterogeneous in terms of computing and processing resources, in order to deal with a wide variety of radio applications. This implies many research activities are needed in the fields of multi-processing and heterogeneous computing. The research described in this thesis proposes one such new approach to support concurrent multiple satellite signals based on SoC FPGA reconfiguration.

A key aspect of the proposed solution to the problem presented is to take the advantage of COTS components. The target is to support multiple small satellite operations in the context satellite formations or constellations, where there may be many satellites present in the field of view of a ground-station antenna, simultaneously, and where a multiplicity of communications protocols, data rates and modulation schemes may be in operation.

To achieve this, different methodologies were first tried on a simulation tool such as GNURadio and then the appropriate blocks were implemented on a baseband SoC to further the state-of-the-art and apply the work to space applications. Such a solution allows inexpensive, efficient, interoperability between the available standards and technologies. Also, this research is ahead of NASA’s SDR communication platform for the next LEO standard S-band transceiver which is
one of the targeted technologies in next 5 – 10 years as mentioned in NASA Communication and Navigation Systems Roadmap [194] and [195].

Chapter 2 investigated small satellites and their common trends to support future technologies in DSS, with fast development times, new found heritage, cheap, and low mass COTS interfaces.

A review of current and future satellite missions concludes that there is need for a generic communication platform that can handle concurrent multiple satellite signals, various modulation techniques, data rates and frequency bands that can fit in to the requirements of small satellite systems.

SDR technology is proposed to be the future generic platform for the space-to-ground and intersatellite communications as the SDR paradigm is already entering into the space segment – albeit slowly. This is driven by three main factors. Firstly, the fast evolution of wireless standards has caused a shorter time-to-market, which makes a programmable SDR solution attractive. Secondly, SDRs offer the flexibility that will allow deployed satellite communication transceivers to be software upgraded according to advances in algorithms and communication standards. Thirdly, the use of COTS components in space has reduced the parts costs and development time of systems.

An overview of current state-of-the-art front-end/back end technologies has been presented along with their challenges, including trade-offs of various ADC/DAC options. Research into the space challenges for SDRs in LEO distributed satellite systems includes various techniques to meet the interference and frequency uncertainty problems.

Chapter 3 investigated the trade-off in architecture design that can be used to meet the diverse requirements/challenges supporting inter-satellite and ground-to-satellite links on the same hardware platform. This section provided a review of the existing/on-going work within application portability, application development, the underlying middleware platform and alternative architectures that can be adapted for the space applications. SDRs for space demand dynamic techniques in the software to provide more flexible mediation and reliability. Implementation on various platforms was used to validate the technology, and this illuminates different ways to test the architecture and also the resources required to perform certain tasks.

It was clear that the desktop reference transceiver application can be implemented on a modern embedded system, which would not only aid in upgrading the traditional ground stations but also could potentially be implemented on a small satellite. Further to this, with the goal of enabling embedded systems for multi-mode communication, the technology is proposed, implemented and validated along with the memory requirement analysis.

CONCLUSIONS
The aim here is to combine the state-of-the-art SDR hardware and open source software tools towards achieving a new generic communication platform for space applications. The implementation of a new combined system-on-chip (SoC) and SDR communication platform enables a reduction in cost, as well as mass. Different parallelisation techniques for ADC/DAC/FPGA will enable a reduction in power consumption by improving the computational capacity, which is an important factor in the design of small satellites. Current work has looked into various possible approaches to implement the required digital signal processing.

Chapter 4 highlights different types of software profiling tools and performance analysis of the implemented blocks using gprof. This exercise helped to quantify the CPU requirements of each block in the transceiver and the failure rates due to CPU starvation. The results confirmed that the reference signal design could not support more than one receive chain as the design was largely implemented in software. The CPU starvation of processing resources also affected the error rate and gprof response as the symbol rate increased. This analysis proved useful, later, to quantify the improvement achieved by rearranging the transceiver blocks. The results of this work are published in the IEEE Transactions on Aerospace and Electronic Systems Journal. The revised architecture is implemented in Chapter 5 and validated with the reference signal.

Chapter 5 includes a FPGA targeted DQT block to start with, which showed a great improvement in the CPU consumption – 36.76% at 1k2, 31.14% at 2k4, 21.5% at 4k8 and 0.7% at 19k2 with 1-2% hardware overhead and 1.4% power overhead. This affirmed that implementing more than one DDC block was possible, and could facilitate concurrent multi-signal reception. The existing Vivado DDC block was found to be not suitable for the design and therefore a custom block was designed and tested at each stage. In order to take the full advantage of a software radio and to move towards SDR Tier 3, as discussed in Chapter 2, it has to be reconfigurable/adaptable to different standards and therefore the DDC block was made dynamically reconfigurable by having a custom register to configure the block for different frequencies and symbol rates. This was achieved with a 13.14% power overhead and 5-18% hardware overhead, compared to the reference design.

The improvements achieved in Chapter 5 allowed the design to support concurrent multiple satellite signals, and this was extended in Chapter 6, covering a novel parallel architecture with different alternatives for concurrent multi-satellite reception.

The implementation of the parallel architecture was tested by emulation of two signals with different standards such as varied data rates, frequencies and Doppler effect. The next step was
to analyse the memory, throughput and FPGA requirements for the worst case – that is when the entire spectrum is occupied by signals of different standards.

The analysis shows that memory and throughput are not the bottlenecks for the symbol rates that this research is concentrating on - 1k2, 2k4, 4k8, 9k6 and 19k2. However, the FPGA fabric is a concern, as the DDC blocks require a certain number of flip-flops, LUTs and DSP48 blocks.

It is calculated that up to 36 DDC blocks can be accommodated on the Zynq 7020 FPGA fabric. Hence, in order to fit in more signals a bigger FPGA (Zynq 7045) or more than one Zynq 7020 is recommended.

These results have been submitted for publication in the IEEE Transactions on Aerospace and Electronic Systems journal.

In conclusion, this thesis demonstrates the concept of combining state-of-the-art low cost SDR hardware and open source software tools, towards achieving a new generic communication platform for small satellite communications. Potential applications of the proposed embedded system architecture are for the ground station for concurrent multi-satellite communications, and deployable mobile ground station networks and can be further extended to distributed satellite systems. In general, the author is confident that the flexibility features included in the proposed architecture constitutes not only an evolution in modern ground-satellite/inter-satellite communications systems, but also a paradigm shift towards enabling higher-layer protocols in order to provide better performances than is the case in traditional telemetry and telecommand architectures.

### 7.1 Contribution to the State-of-the-art

The prime contribution has been that of developing a new parallel architecture using advanced SDR techniques on a modern SoC based embedded system, to enable flexible TT&C communication for small satellites and to receive multiple concurrent satellite signals of different standards (such as different modulation techniques, data rates, frequencies used by amateur/university satellites) on a ground system, with future potential to space application in distributed satellite systems.

### 7.2 Future Work

Due to time constraints, this study had to be paused at this point, but the future work that will lead to a SDR that can be flown on any DSS is discussed in this section. However, majority of the
research issues are addressed in this study, and these can form a baseline for solution to the related engineering problems in the future:

7.2.1 Concurrent Multiple Transmit Channels

As seen in Chapter 5, the concentration was mainly on multi-satellite reception due to:

1. The discovery that the transmit chains consumed less CPU resource compared to the receiver blocks, which allowed concurrent multiple transmission within the existing design, if required. Reception was therefore the key problem to examine.
2. There are legal issues associated in transmitting to concurrent multiple/other satellites not owned by the University of Surrey. Working on receive only simplified these issues.
3. However, multiple transmission would still be valid for DSS scenarios and this could be implemented in similar way as discussed in Chapter 5.

7.2.2 Parallel Architectures

The alternatives 2, 3 and 4, as discussed in section 6.2, can be implemented for improved flexibility and performance as only alternative 1 is implemented, validated and analysed at the moment.

7.2.3 Auto-Configuring

Auto-configuring is definitely required for the ground station as it is unrealistic to keep in track of all the satellites, and for DSS, as these satellites have to work autonomously in space. The flowchart is proposed in section 6.5 with analysis of memory, throughput and FPGA analysis presented in section 6.4.

7.2.4 Test Plan

The SDR developed during this research needs to be validated by performing the following compatibility and environment tests before being proposed for any DSS/targeted missions discussed in section 2.1.1.

7.2.4.1 Compatibility test

1. SDR AGC characterisation testing: The SDR needs analogue and digital AGC to be characterised over temperature ranges between -45°C to +60 °C for LEO [196]. This characterisation testing will help to create several algorithms to estimate the SDR receiving power, which is the desirable capability for on orbit characterisation.
2. SDR reconfigurable parameters: A SDR typically can have more reconfigurable parameters than the ones discussed in Chapter 5 such as different coding schemes,
modulation techniques and protocols. The numerous reconfigurable modes can very quickly increase the amount of pre-flight verification testing that must be completed. Reconfigurable parameters should be limited by the amount of test time available and the level of risk assumed as configurations that are not tested have a higher risk of problems [197].

3. Repairs enabled by software/firmware upgrades: one of the benefits of a SDR is the ability to load new software post-launch. During the pre-launch it is necessary to verify and characterise several software versions for different modes [198].

4. System level testing needs to concentrate on compatibility issues such as frequency stability, transmit power, spurious signal emission/susceptibility etc.

5. On the transmitter side, the power amplifier output power and compression curve are two useful metrics. Also, careful estimation of the drive power based on the voltage generated by DAC, compression curves for different modulation schemes. The platform may include transmitted power sensor which essentially measures voltage on the power amplifier output. In the context of Surrey, the transmitter can be tested with STRaND-1 engineering model satellite for its working.

### 7.2.4.2 Environment test

1. Typical tests for a SDR during thermo-vacuum testing include bit-error rate (BER) characterization, receiver acquisition frequency range, thresholds and timing, Doppler tracking rate and range, output frequency stability, receive performance in the presence of continuous wave and modulated interferers, transmitter error vector magnitudes, transmit power level and transmit spectrum plots [196].

2. Electromagnetic interference (EMI) testing of the SDR is required to ensure successful operation in the presence of EMI [196].

3. Vibration test is the third major environment test, and this test ensures that the SDRs can withstand the vibrations and shocks that occur during launch.

4. Radiation tolerance is an important consideration for SDR operation in space. Typically, individual parts would be radiation tested for total ionising dose and single event effects as was done for STRaND-1[199].
REFERENCES


REFERENCES


[35] SSTL. Available: https://www.sstl.co.uk


Software Defined Radio (SDR) Architecture for Concurrent Multi-Satellite Communications
Mamatha R. Maheshwarappa


REFERENCES


[56] OneWeb, "Building the World's Largest Constellation of Satellites ".

[57] M. Swartwout,"The First One Hundred CubeSats: A Statistical Look."


REFERENCES
REFERENCES


Software Defined Radio (SDR) Architecture for Concurrent Multi-Satellite Communications
Mamatha R. Maheshwarappa


REFERENCES
Software Defined Radio (SDR) Architecture for Concurrent Multi-Satellite Communications
Mamatha R. Maheshwarappa


[117] T. Space. *Gamalink Inter-Satellite Link*. Available: [https://indico.esa.int/indico/event/133/contribution/14/material/0/0.pdf](https://indico.esa.int/indico/event/133/contribution/14/material/0/0.pdf)


Software Defined Radio (SDR) Architecture for Concurrent Multi-Satellite Communications  
Mamatha R. Maheshwarappa

[149] Xilinx, "Zynq 7000 All Programmable SoC Overview."

REFERENCES
REFERENCES


[158] T. C. Coding, "Consultative Committee for Space Data Systems."


[165] ESA. *European Student Earth Orbiter (ESEO)*. Available: [http://www.esa.int/Education/European_Student_Earth_Orbiter](http://www.esa.int/Education/European_Student_Earth_Orbiter)


Software Defined Radio (SDR) Architecture for Concurrent Multi-Satellite Communications
Mamatha R. Maheshwarappa


Software Defined Radio (SDR) Architecture for Concurrent Multi-Satellite Communications
Mamatha R. Maheshwarappa


REFERENCES
# Appendix 1 – Front End Options

<table>
<thead>
<tr>
<th><strong>SDR (Vendors)</strong></th>
<th><strong>SCALDIO (IMEC)</strong></th>
<th><strong>AA1001 (ASIC Ahead)</strong></th>
<th><strong>BW1102 (Bitwave)</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Release date</strong></td>
<td>Feb, 2007</td>
<td>Jan 2007</td>
<td>Sep 2008</td>
</tr>
<tr>
<td><strong>Architecture</strong></td>
<td>Reconfigurable transceiver (Front end) Processor: 200MHz ARM926</td>
<td>Single chip programmable radio (Front end)</td>
<td>Single Transceiver (Front end)</td>
</tr>
</tbody>
</table>
| **RF tuning range** | 174 MHz to 6 GHz  
Channel bandwidth : 1MHz-40MHz | 700 MHz – 6 GHz, | 700MHz – 3.8GHz (Bandwidths from 25 kHz to 20 MHz) |
| **Protocols**     | Future cellular, WLAN, WPAN  
Power consumption: 60 to 120mA  
Vcc : 1.2V | Multiple frequency, IEEE 802.16e | GSM, GPRS, EDGE, WCDMA, HSPA, 1xRTT, CDMA2K, EVDO, DECT, 802.11b/g, 802.16d/e, GPS. |
| **Mechanical**    | Active area of 7.7mm² | Single chip | 7x7 PBGA, 144 Balls, 0.5 Spacing |
| **Block Diagram** | ![Block Diagram 1](image1) | ![Block Diagram 2](image2) | ![Block Diagram 3](image3) |
| **Application**   | Future cellular, WLAN, WPAN, broadcast and positioning standards. | Wireless data devices and mobile phones. | Femtocells, Handset, Laptops and Gaming Devices |
| **Cost**          | $18 for 10,000 units | | |
Software Defined Radio (SDR) Architecture for Concurrent Multi-Satellite Communications  
Mamatha R. Maheshwarappa

<table>
<thead>
<tr>
<th>SDR (Vendors)</th>
<th><strong>Cobra (IMEC)</strong></th>
<th><strong>FunCube Dongle Pro (Funcube)</strong></th>
<th><strong>FunCube Dongle Proplus (Funcube)</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Release date</strong></td>
<td>June, 2010</td>
<td>2010</td>
<td>2012</td>
</tr>
<tr>
<td><strong>Architecture</strong></td>
<td>Cobra is the new version of Scaldio</td>
<td>Limited to 96 kHz I/Q baseband sampling</td>
<td>192kHz sampling rate</td>
</tr>
<tr>
<td><strong>Frequency Band</strong></td>
<td>Upto 6 GHz; Speed : Upto 1 Gbits/s</td>
<td>0.64 - 1100 MHz, 1270 - 1700 MHz</td>
<td>150 – 260 MHz, 410 MHz – 2.15 GHz</td>
</tr>
<tr>
<td><strong>Protocols</strong></td>
<td>IEEE802.11n to .11ac WLAN; LTE to LTE-advanced cellular; and DVB-T/H to DVB-T2 broadcasting</td>
<td>Amateur radio and TETRA</td>
<td>Amateur radio and TETRA</td>
</tr>
<tr>
<td><strong>Mechanical</strong></td>
<td>5 mm²</td>
<td>USB Dongle; Powered by PC's USB port</td>
<td>USB 1x type A male connection</td>
</tr>
<tr>
<td><strong>Power consumption</strong></td>
<td>40 – 100 mW</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Block Diagram</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Application</strong></td>
<td>Multiple asynchronous concurrent data streams on mobile handsets, base stations and small cells.</td>
<td>Any application that understands a standard stereo soundcard configured for quadrature, or I/Q.</td>
<td>Any application that understands a standard stereo soundcard configured for quadrature, or I/Q.</td>
</tr>
<tr>
<td><strong>Cost</strong></td>
<td>----</td>
<td>$ 160</td>
<td>$ 200</td>
</tr>
</tbody>
</table>

Appendix 1 – Front end options

173
## Software Defined Radio (SDR) Architecture for Concurrent Multi-Satellite Communications

**Mamatha R. Maheshwarappa**

### SDR (Vendors)

<table>
<thead>
<tr>
<th>SDR (Vendors)</th>
<th>LMS6002D (Lime Microsystems)</th>
<th>FPAA (Anadigm)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Release date</strong></td>
<td>Dec 2012</td>
<td>Dec 2004</td>
</tr>
<tr>
<td><strong>Architecture</strong></td>
<td><img src="Diagram1.png" alt="Diagram" /></td>
<td>Fully differential architecture</td>
</tr>
<tr>
<td><strong>Frequency Band</strong></td>
<td>0.3 – 3.8 GHz</td>
<td>60 kHz – 600 MHz</td>
</tr>
<tr>
<td><strong>Protocols</strong></td>
<td>Broadband wireless communication devices for WCDMA/HSPA, LTE, GSM, CDMA2000, IEEE® 802.16(x) radios</td>
<td>--</td>
</tr>
</tbody>
</table>
| **Mechanical** | 120 pin DQFN package: 9 mm x 9 mm  
Power: 1.2 V and 3.3 V | 44 lead MQFP package; 12 mm x 12 mm  
Power: 1.8 W |
| **Application** | Femtocell and Pico-cell base stations and Repeaters | Real-time software control of analogue system peripheral |
| **Cost** | $110 | Chip cost: $5; Development kit: $199 |

---

Appendix 1 – Front end options
## Appendix 1 – Front end options

<table>
<thead>
<tr>
<th>SDR (Vendors)</th>
<th>LMS7002M (Lime Microsystems)</th>
<th>AD9364 (Analog Devices)</th>
<th>AD9361 (Analog Devices)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Release date</strong></td>
<td>October 2014</td>
<td>April 2014</td>
<td>2014</td>
</tr>
<tr>
<td><strong>Architecture</strong></td>
<td>Field Programmable Radio Frequency (FPRF) chip with dual transceiver ideal for MIMO</td>
<td>RF 1 x 1 transceiver; Dual receivers: 6 differential or 12 single ended inputs</td>
<td>RF 2 x 2 transceiver; Dual receivers: 6 differential or 12 single-ended inputs</td>
</tr>
<tr>
<td><strong>Frequency Band</strong></td>
<td>100 kHz – 3.8 GHz</td>
<td>70 MHz to 6.0 GHz</td>
<td>70 MHz to 6.0 GHz</td>
</tr>
<tr>
<td>RF Bandwidth: 120 MHz (through analog ports)</td>
<td>Tunable channel bandwidth: &lt;200 kHz to 56 MHz</td>
<td>Tunable channel bandwidth: &lt;200 kHz to 56 MHz</td>
<td></td>
</tr>
<tr>
<td>65 MHz (through digital interface)</td>
<td>Sampling rate: 160/640 MSPS</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Protocols</strong></td>
<td>Supports both TDD and Full Duplex FDD</td>
<td>Supports both TDD and Full Duplex FDD</td>
<td>Supports both TDD and Full Duplex FDD</td>
</tr>
<tr>
<td><strong>Mechanical</strong></td>
<td>1.5 x 11.5 mm aQFN 261-pin package</td>
<td>10 mm x 10 mm; 144 ball chip scale package ball grid array (CSP_BGA)</td>
<td>10 mm x 10 mm; 144 ball chip scale package ball grid array (CSP_BGA)</td>
</tr>
<tr>
<td>Operating Voltage: 1.8 V</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Block Diagram</strong></td>
<td><img src="image" alt="Block Diagram" /></td>
<td><img src="image" alt="Block Diagram" /></td>
<td><img src="image" alt="Block Diagram" /></td>
</tr>
<tr>
<td><strong>Cost</strong></td>
<td>--</td>
<td>$130.00</td>
<td>$175.00</td>
</tr>
</tbody>
</table>
APPENDIX 2 – SPACE SDRS

Figure 7-1 Electra Transceiver Block Diagram

Table 7-2 Electra – Frequencies

<table>
<thead>
<tr>
<th>Channel Number</th>
<th>CCSDS Forward Return Frequency (MHz)</th>
<th>MRO Preset Forward Return Frequency (MHz)</th>
<th>CCSDS Return Frequency (MHz)</th>
<th>MRO Preset Return Frequency (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>437.1</td>
<td>437.1</td>
<td>401.585625</td>
<td>401.585625</td>
</tr>
<tr>
<td>1</td>
<td>435.6</td>
<td>435.6</td>
<td>404.4</td>
<td>404.4</td>
</tr>
<tr>
<td>2</td>
<td>439.2</td>
<td>439.2</td>
<td>397.5</td>
<td>397.5</td>
</tr>
<tr>
<td>3</td>
<td>444.0</td>
<td>444.0</td>
<td>393.9</td>
<td>393.9</td>
</tr>
<tr>
<td>4</td>
<td>435 to 450</td>
<td>435 to 450</td>
<td>390 to 405</td>
<td>401.4</td>
</tr>
<tr>
<td>5</td>
<td>435 to 450</td>
<td>435 to 450</td>
<td>390 to 405</td>
<td>402.6</td>
</tr>
<tr>
<td>6</td>
<td>444 to 450</td>
<td>444 to 450</td>
<td>390 to 405</td>
<td>402.6</td>
</tr>
<tr>
<td>7</td>
<td>442</td>
<td>442</td>
<td>393</td>
<td>393</td>
</tr>
<tr>
<td>8</td>
<td>444.2</td>
<td>444.2</td>
<td>390</td>
<td>390</td>
</tr>
<tr>
<td>9</td>
<td>444</td>
<td>444</td>
<td>393</td>
<td>393</td>
</tr>
<tr>
<td>10</td>
<td>445</td>
<td>445</td>
<td>395</td>
<td>395</td>
</tr>
<tr>
<td>11</td>
<td>446</td>
<td>446</td>
<td>395.5</td>
<td>395.5</td>
</tr>
<tr>
<td>12</td>
<td>447</td>
<td>447</td>
<td>396</td>
<td>396</td>
</tr>
<tr>
<td>13</td>
<td>448</td>
<td>448</td>
<td>399</td>
<td>399</td>
</tr>
</tbody>
</table>

Table 7-1 MRO/Electra Modes, Functions, and Performance

<table>
<thead>
<tr>
<th>Capability</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Protocol</td>
<td>Prox-1 (reliable and expected link layer protocols)</td>
</tr>
<tr>
<td>Frequencies</td>
<td>See next section (including Table 2-5)</td>
</tr>
<tr>
<td>Modes of operation</td>
<td>Half-duplex Rx and Tx (no Prox-1 protocol in half duplex)</td>
</tr>
<tr>
<td></td>
<td>Full-duplex transceiver</td>
</tr>
<tr>
<td>Full-duplex carrier modes</td>
<td>Coherent, noncoherent</td>
</tr>
<tr>
<td>Transceiver RF output power</td>
<td>5.0 W full duplex, 7.0 W half duplex</td>
</tr>
<tr>
<td>Circuit loss, EUT to antenna</td>
<td>−0.42 dB</td>
</tr>
<tr>
<td>Receiver thresholds, at antenna</td>
<td>−130.8 dBm (1 kbps) to −99.6 dBm (1024 kbps) coded</td>
</tr>
<tr>
<td></td>
<td>−126.0 dBm (1 kbps) to −91.1 dBm (2048 kbps) uncoded</td>
</tr>
<tr>
<td>Carrier modulation modes</td>
<td>Suppressed carrier, residual carrier (60 deg mod index)</td>
</tr>
<tr>
<td>Modulation types</td>
<td>Residual carrier binary phase-shift keying (BPSK) with bi-phase-L (Manchester). Suppressed-carrier BPSK</td>
</tr>
<tr>
<td>Frequency reference</td>
<td>Ultra stable oscillator</td>
</tr>
<tr>
<td>Rx and Tx symbol rates</td>
<td>1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048 kbps. Also, adaptive data rate mode</td>
</tr>
<tr>
<td>Received signal power range</td>
<td>−140 to −70 dBm</td>
</tr>
<tr>
<td>Encoding</td>
<td>Uncoded, (k = 7, r = 1/2) convolutional, differential symbol coding</td>
</tr>
<tr>
<td>Decoding</td>
<td>Uncoded, (k = 7, r = 1/2) convolutional (3-bit soft decode)</td>
</tr>
<tr>
<td>Scrambling/descrambling</td>
<td>V.38</td>
</tr>
<tr>
<td>Acquisition and tracking loop</td>
<td>Second-order PLL, with loop bandwidth 10 Hz to 10 kHz (for received signal from −140 dBm to −70 dBm)</td>
</tr>
<tr>
<td>Tracking range and rate</td>
<td>±20 kHz, ±200 Hz/s</td>
</tr>
</tbody>
</table>
Software Defined Radio (SDR) Architecture for Concurrent Multi-Satellite Communications
Mamatha R. Maheshwarappa

Figure 7-2 SCaN Test-bed (Image credit: NASA)

Table 7-3 Overview of SCaN Test-bed Specifications [27]

<table>
<thead>
<tr>
<th>SDR Parameter / Manufacturer</th>
<th>JPL</th>
<th>GD</th>
<th>Harris</th>
</tr>
</thead>
<tbody>
<tr>
<td>General Purpose and Signal Processing Modules</td>
<td>66 MHz SPARC processor and 2 Xilinx Virtex II FPGAs</td>
<td>60 MIPS Coldfire processor and 1 Xilinx Virtex II Pro FPGA</td>
<td>700 MIPS Power PC processor and 4 Xilinx Virtex IV FPGAs</td>
</tr>
<tr>
<td>RF Module</td>
<td>Two (10-bit) 50 Mbps DAC and one 10-bit 50 Mbps ADC, S-band duplexer (2.7–2.9 GHz, 6 MHz tuning, and GPS receive at L1, L2 and L5)</td>
<td>S-band duplexer (2.1–2.3 GHz, 6 MHz tuning)</td>
<td>Two 12-bit 300 MHz DAC and one 300 MHz ADC, Ka-band (transmit 25.5–25.7 GHz, 220 MHz tuning, receive: 22.5–22.7 GHz, 50 MHz tuning)</td>
</tr>
<tr>
<td>Memory Unit</td>
<td>128 MByte SDRAM and 512 MByte flash</td>
<td>128 MByte SDRAM and 4 MByte EEPROM; also 1 MByte Flash on SSD as experiment</td>
<td>256 MByte SDRAM</td>
</tr>
<tr>
<td>OE (Operating Environment)</td>
<td>RIEM operating system: OE complies with STRS, V1.02</td>
<td>VxWorks operating system: OE complies with STRS, V1.02</td>
<td>VxWorks operating system: OE complies with STRS, V1.02</td>
</tr>
<tr>
<td>Command &amp; Telemetry</td>
<td>MIL-STD-1553</td>
<td>MIL-STD-1553</td>
<td>SpaceWire</td>
</tr>
<tr>
<td>Data Format</td>
<td>SpaceWire</td>
<td>SpaceWire</td>
<td>SpaceWire</td>
</tr>
<tr>
<td>Data Rate Class</td>
<td>10’s Mbps</td>
<td>10’s Mbps</td>
<td>100’s Mbps</td>
</tr>
<tr>
<td>Output Power Amplifier</td>
<td>10 W</td>
<td>10 W</td>
<td>Ka converter drives 40 W TWTA</td>
</tr>
</tbody>
</table>

Figure 7-3 Block Diagram of the Functional Interactions of the RF Subsystem [27]
## APPENDIX 3 – SDRs for Small Satellite Applications

<table>
<thead>
<tr>
<th>(SDR)University/Space Industry</th>
<th>CSDR (Saint Louis University)</th>
<th>Frontier Radio (Applied Physics Laboratory)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Satellite</td>
<td>SLUCube</td>
<td>Radiation Belt Storm Probes (RBSP)</td>
</tr>
<tr>
<td>Launch</td>
<td>--</td>
<td>August 30, 2012</td>
</tr>
<tr>
<td>Architecture</td>
<td>![Architecture Diagram]</td>
<td>![Architecture Diagram]</td>
</tr>
<tr>
<td>Frequency Band</td>
<td>VHF and UHF</td>
<td>S-band (2025 – 2120 MHz)</td>
</tr>
<tr>
<td>Data Rate</td>
<td>19.6 kbps on VHF and 56 kbps on UHF</td>
<td>1 bps – 1.25 Mbps (uncoded)</td>
</tr>
<tr>
<td>Space Qualified</td>
<td>--</td>
<td>Yes</td>
</tr>
<tr>
<td>Image</td>
<td><img src="image.png" alt="Image" /></td>
<td><img src="image.png" alt="Image" /></td>
</tr>
<tr>
<td>Size &amp; Mass</td>
<td>PC104 spec / 51.2g</td>
<td>1.8 kg</td>
</tr>
<tr>
<td>Power</td>
<td>319 mW</td>
<td>5W</td>
</tr>
<tr>
<td>Cost</td>
<td>$461.64</td>
<td>--</td>
</tr>
</tbody>
</table>
### Software Defined Radio (SDR) Architecture for Multi-Satellite Communications

**Mamatha R. Maheshwarappa**

#### Appendix 3 – SDRs for Small Satellite Applications

<table>
<thead>
<tr>
<th>(SDR)University/Space Industry</th>
<th>SDR (GAMALINK)</th>
<th>Modular SDR (BitBeam)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Satellite</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>Launch</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td><strong>Architecture</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><img src="image" alt="SDR (GAMALINK) Diagram" /></td>
<td><img src="image" alt="Modular SDR (BitBeam) Diagram" /></td>
<td></td>
</tr>
<tr>
<td>Frequency Band</td>
<td>0.3 – 3.8 GHz</td>
<td>100 – 6000 MHz</td>
</tr>
<tr>
<td></td>
<td>Signal Bandwidth: 1.5 – 28 MHz</td>
<td>Rx Daughter Board: 2.0–2.1, 2.4–1.5 GHz</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tx Daughter Board: 8.1–8.5 GHz</td>
</tr>
<tr>
<td>Data Rate</td>
<td>1k – 1M Symbols/s (BPSK/QPSK/8PSK)</td>
<td>100 kbps -200 Mbps</td>
</tr>
<tr>
<td>Space Qualified</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td><strong>Image</strong></td>
<td><img src="image" alt="SDR (GAMALINK) Image" /></td>
<td><img src="image" alt="Modular SDR (BitBeam) Image" /></td>
</tr>
<tr>
<td>Size &amp; Mass</td>
<td>3.0” square</td>
<td>90.2 x 96 x 13.8 mm / 97 g</td>
</tr>
<tr>
<td>Power</td>
<td>1.2W</td>
<td>1 - 4 W</td>
</tr>
</tbody>
</table>
# Appendix 4 - Terrestrial SDRs

<table>
<thead>
<tr>
<th>SDR (Vendors)</th>
<th>Matchstiq (EPIQ solutions)</th>
<th>Bitshark Express Rx (EPIQ solutions)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Release date</strong></td>
<td>2013</td>
<td>Not yet released</td>
</tr>
<tr>
<td><strong>Frequency Band</strong></td>
<td>300 MHz – 3.8 GHz; Bandwidth: 28 MHz</td>
<td>300 MHz – 4 GHz; Bandwidth: 50 MHz</td>
</tr>
<tr>
<td><strong>Protocols</strong></td>
<td>GSM, iDEN, CDMA2K/EVDO, UMTS, TD-SCDMA, WiFi/802.11b/g, WiMAX/802.16 d/e</td>
<td>GSM, iDEN, CDMA2K/EVDO, UMTS, TD-SCDMA, WiFi/802.11b/g, WiMAX/802.16 d/e</td>
</tr>
<tr>
<td><strong>Mechanical</strong></td>
<td>Size: 2.2” x 4.6” x 0.9” Power: &lt;3 W</td>
<td>--</td>
</tr>
</tbody>
</table>
| **Application** | • Stand-Alone in UAV  
• Wireless interface to Android Host  
• Wired interface to PC | • Stand-Alone in UAV  
• Wireless interface to Android Host  
• Wired interface to PC |

## Block Diagrams

- **Matchstiq** (EPIQ solutions) Block Diagram
- **Bitshark Express Rx** (EPIQ solutions) Block Diagram

---

Appendix 4 – Terrestrial SDRs 180
### Software Defined Radio (SDR) Architecture for Multi-Satellite Communications

Mamatha R. Maheshwarappa

#### Appendix 4 – Terrestrial SDRs

<table>
<thead>
<tr>
<th>Cost</th>
<th>$ 6500</th>
<th>$ 6300</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SDR (Vendors)</strong></td>
<td><strong>BladeRF (Nuand)</strong></td>
<td><strong>USRP E310 (Ettus Research)</strong></td>
</tr>
<tr>
<td><strong>Release date</strong></td>
<td>September, 2013</td>
<td>May 2015</td>
</tr>
<tr>
<td><strong>Architecture</strong></td>
<td>--</td>
<td>2 x 2 MIMO</td>
</tr>
<tr>
<td><strong>Frequency Band</strong></td>
<td>300 MHz – 3.8 MHz; Bandwidth : 28 MHz</td>
<td>70 MHz – 6 GHz; Bandwidth: 56 MHz</td>
</tr>
<tr>
<td><strong>Protocols</strong></td>
<td>GSM and LTE picocell, GPS Rx, ATSC Tx, Bluetooth/Wi-Fi</td>
<td></td>
</tr>
<tr>
<td><strong>Mechanical</strong></td>
<td>Power: 5V DC</td>
<td>133 x 68 x 26.4 mm; 375 g</td>
</tr>
<tr>
<td><strong>Block Diagram</strong></td>
<td>--</td>
<td></td>
</tr>
<tr>
<td><strong>Application</strong></td>
<td>Open hardware platform for hobbyists, and professionals to explore and experiment with the multidisciplinary facets of RF communication.</td>
<td>Mobile and embedded applications</td>
</tr>
</tbody>
</table>
### Software Defined Radio (SDR) Architecture for Multi-Satellite Communications

Mamatha R. Maheshwarappa

<table>
<thead>
<tr>
<th>Cost</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$ 650</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$ 3130</td>
</tr>
</tbody>
</table>

Appendix 4 – Terrestrial SDRs
APPENDIX 5 – SMART FUSION2 AND FUNCUBE DONGLE

This includes GPIO, USB, filtering and porting of existing SDR code towards configuring the Linux kernel and the USB driver interface. The SF2 SOM is intended to provide a flexible platform for embedded applications with SF2 SoC FPGA, built on 65nm process technology coupled with a full-fledged Linux software execution environment running on the 166 MHz ARM Cortex-M3 SF2 processor core, advanced security processing accelerators, DSP blocks and SRAM. However, SF2 does not support hardware floating point or vector instructions which can improve performance. Initial tests were carried out on the SF2 which looked into configuring the GPIO block and MMUART_0, programming SF2 M2S050T on the starter kit board, using the SF2 MSS configurator. For this application, Soft Console [134] and Libero [135] Software running on a Windows development host were used.

Interfacing USB/FCD

Having understood the basics of the Linux kernel and the process of building the kernel, next task was to interface FunCube Dongle (FCD) Pro, the hardware setup is shown in Figure 1, the first step towards realising the SDR.

![Figure 1 SF2 with FCD](image)

FCD was chosen as the front end for the test purpose and since it is USB audio device, few features such as support for host-side USB, USB device class-devices and USB mass storage support under USB support in the Linux kernel configuration were selected to detect the FCD.
Conclusion on SF2 + FCD

The practical work demonstrated here has initially looked into porting and characterising the performance of existing SDR software chains in an embedded system towards achieving the objectives of this research. The initial work includes:

1. A hardware test-bed that has been formulated using the SF2 starter kit as a platform to investigate the existing tools and demonstrating Linux software environment.

2. FCD is proposed in literature and now practically used to show that the commercially available front end technologies work not only with Desktop-PCs but also with the embedded system such as SF2.

However, this implementation of the concept is missing key SDR features:

1. M3 has no floating point/vector instructions which is necessary for handling complex I and Q signals.

2. SF2 FPGA resources are limited compared to modern Altera/Xilinx alternatives, which also have much more powerful hard core(s).

A completely bare metal hand crafted design on SF2 would do the job, but would be significant effort, much VHDL/C from scratch (The latest Lime SDR is based on the bare metal hand crafted design which was not available when the implementation was started).
APPENDIX 6 – IMPLEMENTATION STEPS AND CHALLENGES

Figure 2. Files on a SD Card for Zedboard Bootup

The reference HDL was downloaded from Analog Devices’ repository [34] which has only the sources, bit/elf files were generated by running the .tcl scripts. The libraries required for AD-FMCOMMS2-3 board and the project were built as per [35]. DDC blocks (I and Q) based on CIC filters were integrated with the reference design. This design was validated, synthesized and implemented to generate system.bit file.

2. File generated: First Stage Boot Loader (FSBL.elf); Tool Used: SDK 2014.2.
Once the system.bit was generated, the HDL design was exported to SDK to create a new application project using Zynq FSBL.

3. The u-boot.elf (bootloader) was provided by Analog Devices[36]. Alternatively, this can be independently built for the Zedboard.

Above files were added to the partition list in Create Zynq Boot Image dialog to generate BOOT.BIN file.

This files were generated by cloning the right version/branch repository as mentioned in Table 1 and built on a Linux PC as per [25].
Table 1: Page Margins for Letter and A4 Submissions

<table>
<thead>
<tr>
<th>Repository</th>
<th>Version</th>
<th>Branch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux</td>
<td>Linux</td>
<td>2014_R2</td>
</tr>
<tr>
<td>HDL</td>
<td>2014.2</td>
<td>hdl_2014_R2</td>
</tr>
</tbody>
</table>

6. uEnv.txt contains the base address of all the files and it was provided by Analog Devices.