Improvements in CPU & FPGA Performance for Small Satellite SDR Applications

Mamatha R. Maheshwarappa, *Student Member, IEEE,* Mark D. J. Bowyer, *Member, IEEE* and Christopher P. Bridges, *Member, IEEE*

***Abstract*— The ongoing evolution in constellation/formation of CubeSats along with steadily increasing number of satellites deployed in Lower Earth Orbit (LEO), demands a generic reconfigurable multimode communication platforms. As the number of satellites increase, the existing protocols combined with the trend to build one control station per CubeSat become a bottle neck for existing communication methods to support data volumes from these spacecraft at any given time. This paper explores the Software Defined Radio (SDR) architecture for the purposes of supporting multiple-signals from multiple-satellites, deploying mobile and/or distributed ground station nodes to increase the access time of the spacecraft and enabling a future SDR for Distributed Satellite Systems (DSS). Performance results of differing software transceiver blocks and the decoding success rates are analysed for varied symbol rates over different cores to inform on bottlenecks for Field Programmable Gate Array (FPGA) acceleration. Further, an embedded system architecture is proposed based on these results favouring the ground station which supports the transition from single satellite communication to multi-satellite communications.**

***Index Terms* — Central Processing Unit (CPU), Field Programmable Gate Array (FPGA), Satellite communication, Software Defined Radio (SDR), System-on-chip (SoC).**

I. INTRODUCTION

SMALL satellites are fast becoming a way to perform scientific and technological missions more affordably due to reduced build time, more frequent launch opportunities, larger variety of missions, more rapid expansion of the technical and/or scientific knowledge base and greater involvement of small industries/universities [1, 2]. Furthermore, there is an ongoing evolution of multiple small satellite scenarios such as FLOCK-1 [3], QB50 [4], Autonomous Assembly of a Reconfigurable Space Telescope (AAReST) [5], Surrey Training Research and Nano-Satellite Demonstrator (STRaND -2) [6] and Edison Demonstration of Smallsat Network (EDSN) [7]. The objectives of these missions are very ambitious and are driven by new complexities which require multi-mode operation of wireless transceivers [8].

This paragraph of the first footnote will contain the date on which you submitted your paper for review. It will also contain support information, including sponsor and financial support acknowledgment. For example, “This work was supported in part by the U.S. Department of Commerce under Grant BS123456”.

M. R. Maheshwarappa and C. P. Bridges are with the Surrey Space Centre, Faculty of Engineering and Physical Sciences (FEPS), University of Surrey, Guildford, United Kingdom - GU27XH (e-mail: m.maheshwarappa@surrey.ac.uk; c.p.bridges@surrey.ac.uk).

M. D. J. Bowyer is with Airbus Defence and Space Ltd., Anchorage Road, Portsmouth, Hampshire, United Kingdom – PO35PU.

This work aims at three specific application areas. Firstly, the ground station that can handle multiple satellite signals at any given time as seen in Fig. 1. The increasing number of satellites in Lower-Earth Orbit (LEO) occupying Amateur Radio Spectrum together with variety of modulation techniques, data rates and protocols [9] used across the CubeSat community demands the integration of a multitude of communication standards onto a single platform. This is compounded by the problem of crowded spectrum [10] which is driving research on more efficient use of the available spectrum e.g., by de-confliction or Cognitive Radio (CR) techniques. For all such applications, a universal programmable hardware is desirable, which intensifies the interest in Software Defined Radio (SDR) in recent years [11]. Such an SDR must be robust in noisy and/or contested spectrum and make maximum use of a priori information to minimise initial acquisition and detection bandwidths.

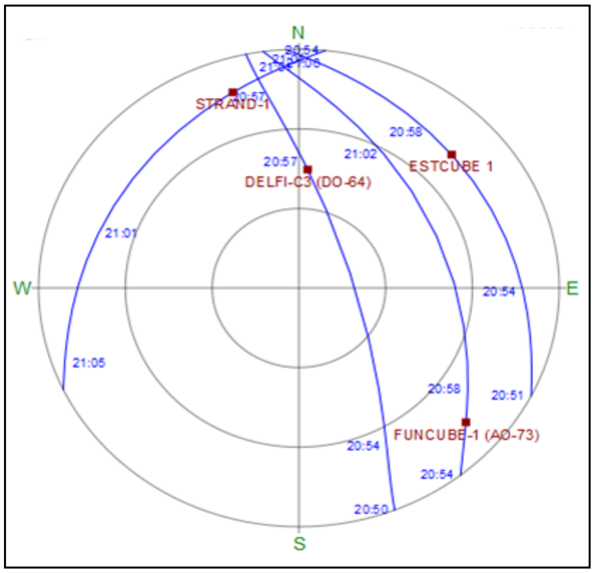


Fig. 1. Radar View of the Antenna Showing Different Satellites in Visibility

Secondly, the need for deployable mobile ground station network for the purposes of increased access time such as ESA’s Global Educational Network for Satellite Operations (GENSO) system [12] and Satellite Networked Open Ground Station (SatNOGS) [13]. A ground station based on SDR hardware is suitable for worldwide distributed systems, where updates containing the software for communicating with new waveforms could be shared among different distant stations without the need for hardware upgrades.

Finally, a candidate embedded design is presented as a possible enabler of the future SDR for distributed satellite communication systems. The growth of SDR offers small satellites the opportunity to improve the way space missions develop and operate transceivers for communication network in space as seen in [36] and [40]. 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 and recover from anomalies within the science payload or communication system e.g., in Global Navigation Satellite Systems (GNSS) receivers as in [38, 39]. Also, potentially reduce development cost and risk by adapting generic space platforms to meet specific mission requirements. However, the flexibility and adaptability comes with an expense of power consumption and complexity in integrating previously separated building blocks on a single die.

The objectives of this paper are:

1. SDR implementation and profiling analysis of SmallSat Telemetry, Tracking and Command (TT&C) waveform on state-of–the-art Radio Frequency (RF) (Analog Devices AD9361) and Base Band SoC (Xilinx Zynq) based architecture, with emphasis on ground system multi-satellite reception.
2. Profiling of C/C++ based reference waveform design 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 expenses associated with implementation of many variant waveforms.
3. Using a low cost Zynq SoC solution (the Zedboard), the desired multi-satellite reception can accommodate up to 4 concurrent satellites by moving waveform independent front-end tuning, filtering and decimation functions from software to firmware, leaving waveform dependent matched filtering, demodulation and decoding functions in software.

This paper is an extension of the work carried out in [33] and [34] where a novel SDR architecture on an embedded system is proposed as seen in section 2. The implementation and validation process of the proposed transceiver architecture is briefed in section 3 (more details on transceiver implementation and validation can be found in [34]). The focus of this paper is to understand the CPU load caused by each transceiver block as discussed in section 4. Further in section 5, an improvement in the design is achieved by re-distributing the transceiver blocks within the SoC. Lastly, section 6 summarises the contributions and future work.

II. Transceiver Architecture

For over two decades, SDR technology has promised to revolutionise the communication industry by delivering low cost, flexible software solutions for communication protocols [9]. In this decade, the introduction of BB SoC and, most recently, RF programmable transceiver SoC can fulfill the early promise. Also, open source simulation tool such as GNURadio [35] is widely used to implement low-cost digital beacon receiver based on SDR [37], Emergency Managers Weather Information Network (EMWIN) and Low-Rate Information Transmission (LRIT) Software Receiver using GNURadio [41]. GNURadio was used in this research initially to understand the working of the existing/generated filters, channel codes, synchronization elements, equalisers, demodulators, decoders and other processing blocks using pre-recorded or generated data as addressed in [33].

Towards achieving the attributes discussed in the previous section, this work proposes a new SDR architecture on an embedded system as seen in Fig. 2 [33]. This architecture

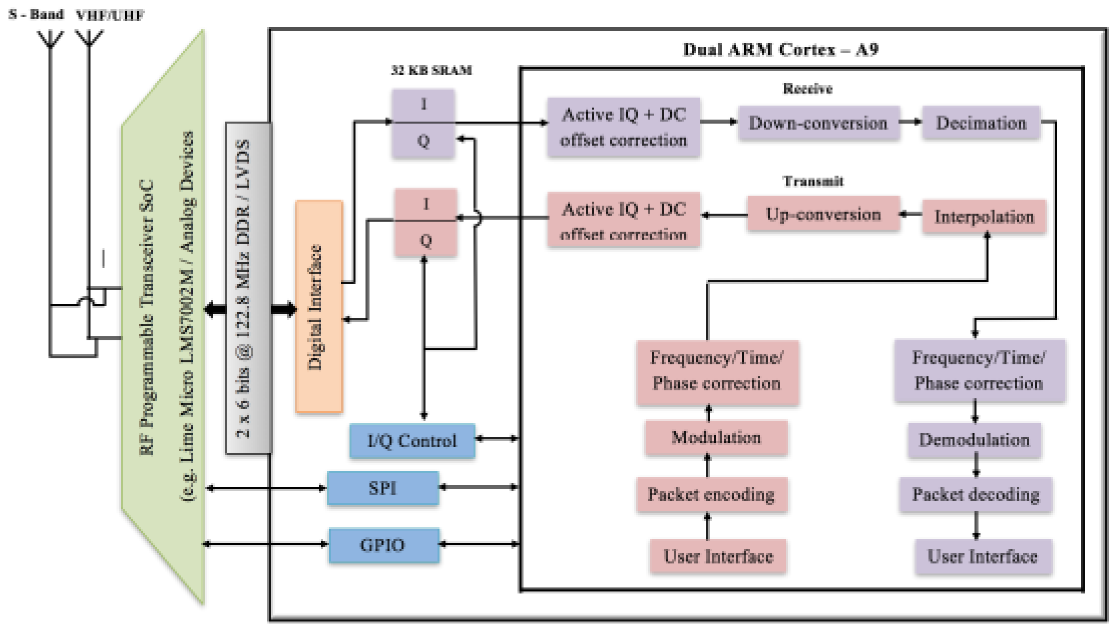

S - Band
VHF/UHF
RF Programmable Transceiver SoC
(e.g. Lime Micro LMS7002M / Analog Devices
2 x 6 bits @ 122.8 MHz DDR / LVDS
Digital Interface
32 KB SRAM
I
Q
I
Q
I/Q Control
SPI
GPIO
Dual ARM Cortex – A9
Receive
Active IQ + DC offset correction
Down-conversion
Decimation
Transmit
Active IQ + DC offset correction
Up-conversion
Interpolation
Frequency/Time/ Phase correction
Modulation
Packet encoding
User Interface
Frequency/Time/ Phase correction
Demodulation
Packet decoding
User Interface

Fig. 2. SDR Architecture Implemented on Zynq

consists of a BB SoC paired with RF SoC. The BB SoC contains FPGA fabric and ARM dual-core Cortex A9 processor. For initial development, the Avnet Zedboard containing the Xilinx Zynq 7020 FPGA SoC [14] is chosen providing a low-cost and well supported back-end for the signal processing functionalities.

On the RF programmable transceiver SoC, initial evaluation took place using the Lime Micro Myriad RF containing the LMS6002D RF SoC [15]. More recent development has taken place using the Analog Devices AD-FMCOMMS3-EBZ containing the newer AD9361 RF SoC [16]. It is hoped that future developments will incorporate the latest and most capable Lime Micro SoC, the LMS7002M [17]. The two boards (and constituent SoCs) communicate using conventional parallel I/O for high speed sampled data (up to ~123 complex MSPS) and Serial Peripheral Interface (SPI) for configuration, control and monitoring. Detailed description of the SDR architecture can be found in [33] and [34].

III. Implementation

As a first step towards validating the architecture, a simple coder modulator/demodulator decoder reference model for a well-known CubeSat beacon telemetry was implemented. The FUNcube-1 (AO-73) CubeSat [18] provides a good starting point for our work because the telemetry beacon is documented and addressed by number of Open Source Software (OSS) demodulator decoder implementations written in C/C++.

A. *Transmitter*

The particular scheme, from AO-40 heritage [19], common among several CubeSats [18], is based on Binary Phase Shift Keying (BPSK) modulation and a robust concatenated code comprising Viterbi (Rate 1/2) [20] and two Reed Solomon (160,128) blocks [21]. Much work here derives from Phil Karn’s well-known AO-40 design and implementation [KA9Q] [19]. The Analog Devices AD-FMCOMMS3-EBZ has bare metal and Linux Operating System (OS) based device drivers accompanied by application examples. For this work, we have started with the Zynq ARM Linux OS based approach as the integration and test of application related OSS may be simplified. To this end, Analog Devices provide a capable AD9361 Linux device driver, dependent on and accessed, using Linux industrial I/O (IIO) framework [22]. Linux IIO allows user space waveform applications to configure/query/sample-stream to and from the AD9361 using familiar UNIX calls (open/close/read/write/ ioctl) and perhaps, and more preferred, by a user space library called libiio [23]. The Linux libiio provides a modern high performance abstraction to all IIO devices including the AD9361. Using IIO, it has been possible to create a soft real time reference encoder called “iio-fcenc”.

Successful interoperability testing of iio-fcenc took place for different symbol rates such as 1.2K, 2.4K, 4.8K, 9.6K and 19.2K. Fig. 3 show the signals being received on a FUNcube Pro+ dongle and spectrum analysis performed using SDR Sharp [24].

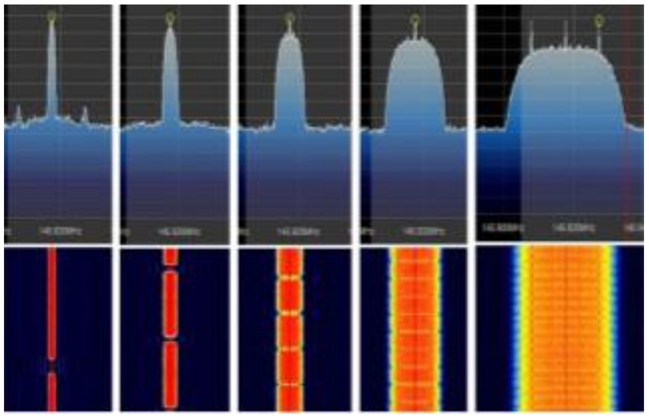


Fig. 3. Signal Received on SDR Sharp at Different Data Rates 1.2K, 2.4K, 4.8K, 9.6K and 19.2K (from left to right)

It was possible to run iio_fcenc on different platforms of varying architecture and core capacity, including – ARM Cortex 15 and ARM Cortex A7, ARM Cortex A9 and Intel x86. The transmission was also verified on a Rohde & Schwarz FSV3 Vector Signal Generator (VSG) [25] as seen in Fig. 4 and the constellation plot of the BPSK signal can be seen in Fig. 5. 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 AD-FMCOMMS3-EBZ provides the flexibility to transmit at any desired frequency within the range of 70 MHz to 6.0 GHz. Also, the freedom to adjust centre frequencies and sample rates under software control helps compensating thermal drift, clock timing and Doppler Effects. This architecture demonstrates the SDR attributes such as post-launch re-configurability, scalability and affordability to promote commercially available computer software and hardware products/standards which was not achievable by traditional transmitters.

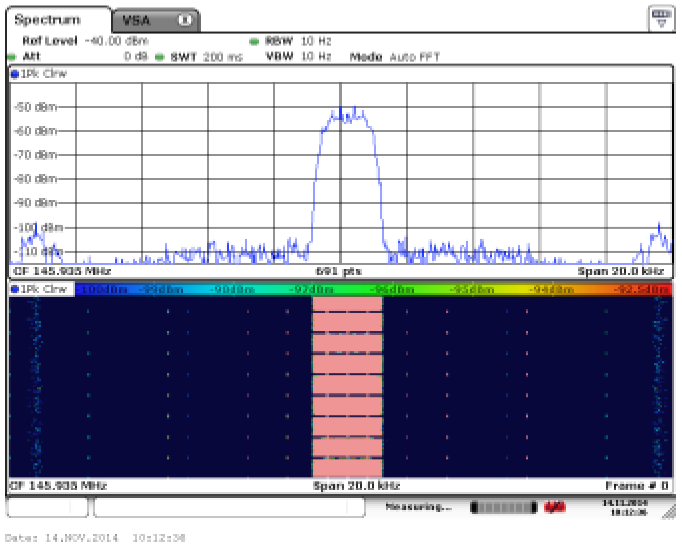
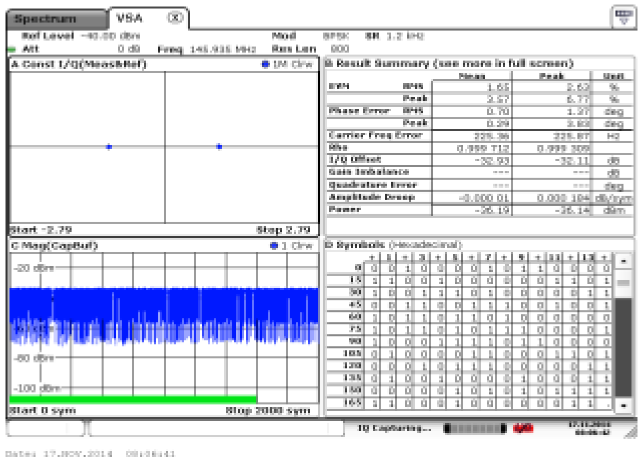


Fig. 4. Transmission Verified on Rohde & Schwarz VSG



B Result Summary (see more in full screen)

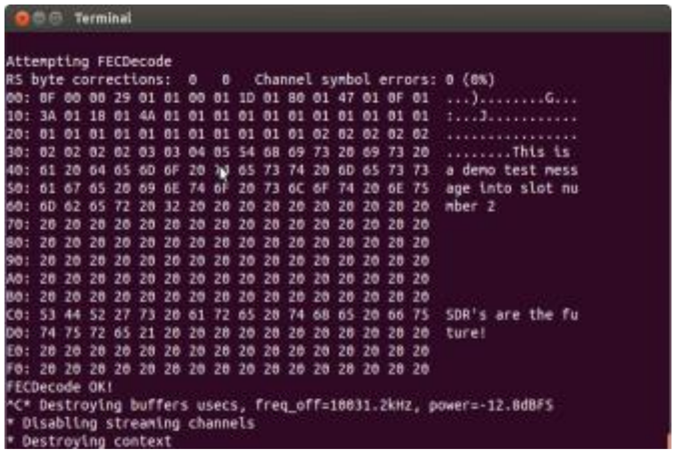
		Mean	Peak	Unit
EVM	RMS	1.65	2.62	%
	Peak	2.57	6.77	%
Phase Error	RMS	0.70	1.27	deg
	Peak	0.29	3.93	deg
Carrier Freq Error		225.36	225.87	Hz
Rho		0.999 712	0.999 309	
I/Q Offset		-32.93	-32.11	dB
Gain Imbalance		---	---	dB
Quadrature Error		---	---	deg
Amplitude Droop		-0.000 01	0.000 104	dB/sym
Power		-26.19	-26.14	dBm

Fig. 5. Constellation Plot of the Transmitted Signal

B. Receiver

The chosen OSS starting point to form a “reference implementation” is Alex Csete’s FUNcube Decoder (fcdec) available on github [26]. This C/C++ code base, targeted for Linux, is designed to work offline using sample files captured from the FUNcube Pro Dongle [24]. Using IIO, along with fcdec it has been possible to create a soft real time reference decoder called “iio-fcdec” similar to “iio-fcenc”. This was tested for interoperability against FUNcube-1 reference waveforms up-sampled, stored and played back on a Rohde & Schwarz SMBV100 VSG [25].

The transmitted signals were looped back to the receiver port to transmit and receive the signals simultaneously. Fig. 6 shows the decoded packets from the loopback test. It was also possible to run iio_fcdec on an x86 PC and Odroid-XU Lite (Octa - ARM Cortex A15 Quad Core and ARM Cortex A7 Quad Core) [27] and stream samples from Zedboard (which is running iiod by default) over Ethernet network to compare the performance of the blocks on different processors. Different symbol rates (1.2K, 2.4K, 4.8K, 9.6K and 19.2K) were achieved by changing the interpolation ratio and decimation ratio in iio_fcenc and iio_fcdec respectively similar to what was achieved on the Zedboard.



```
Attempting FECDecode
RS byte corrections:  0   0   Channel symbol errors: 0 (0%)
00: 0F 00 00 29 01 01 00 01 1D 01 80 01 47 01 0F 01  ...)........G...
10: 3A 01 18 01 4A 01 01 01 01 01 01 01 01 01 01 01  :...J...........
20: 01 01 01 01 01 01 01 01 01 01 01 02 02 02 02 02  ................
30: 02 02 02 02 03 03 04 05 54 68 69 73 20 69 73 20  ........This is 
40: 61 20 64 65 6D 6F 20 74 65 73 74 20 6D 65 73 73  a demo test mess
50: 61 67 65 20 69 6E 74 6F 20 73 6C 6F 74 20 6E 75  age into slot nu
60: 6D 62 65 72 20 32 20 20 20 20 20 20 20 20 20 20  mber 2
70: 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
80: 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
90: 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
A0: 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
B0: 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
C0: 53 44 52 27 73 20 61 72 65 20 74 68 65 20 66 75  SDR's are the fu
D0: 74 75 72 65 21 20 20 20 20 20 20 20 20 20 20 20  ture!
E0: 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
F0: 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
FECDecode OK!
^C* Destroying buffers usecs, freq_off=10031.2kHz, power=-12.0dBFS
* Disabling streaming channels
* Destroying context
```

Fig. 6. Decoded Signal

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 an integer multiple of an oversampled symbol rate (e.g. 40x1.2K) that is conveniently larger than the 1.5 Msps limit imposed. In this implementation, 1.536 Msps was chosen that derives from 16 x 96 ksps. Therefore, the received sample stream is decimated by 16. The resulting 96 ksps sample stream has sufficient bandwidth to allow sufficient bandwidth to address spacecraft Doppler and oscillator uncertainties but discard LO breakthrough and IQ imbalance artifacts by halving the available bandwidth to ~40 kHz. The 96 ksps sample stream is processed in software for flexibility and simple access to floating point arithmetic. This receiver processing is embedded (on the Zedboard’s ARM Cortex A9) or streamed remotely to a more powerful host such as Intel x86 and Odroid-XU Lite as in Table I using sample streaming provided by Analog Device’s iiod [28].

TABLE I
COMPARISON OF DIFFERENT PLATFORMS

	Dell Optiplex 745	Odroid XU Lite	Zedboard
Processor	Intel x86	ARM Cortex A15 & A7	ARM Cortex A9
Number of Cores	Dual	Octa – Quad A15 & Quad A7	Dual
CPU Frequency	2.13 GHz	A15 – 1.4 GHz A7 – 1.2 GHz	700 MHz
Linux Version	3.13.0	3.4.98	3.15.0
System type	64-bit	32-bit	32-bit
Application	Identical Application from Source		

The first signal processing step is coarse carrier acquisition performed using an 8192 point Fast Fourier Transform (FFT). This 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 heritage reasons). At this stage the underlying signal is down-converted to baseband and matched filtered followed by carrier phase recovery. Finally, from symbol timing recovery a 1.2K symbol stream is produced and passed to the decoder. As the receiver input signal bandwidth is limited to ~40 KHz by the reference design the symbol rate was limited to 19.2K (and still includes excess bandwidth for Doppler uncertainty).

IV. PROFILING

Profiling can decompose and tabulate the execution weight of each block in the compiled C/C++ program. We are using GNU gprof [29] to identify critical regions, determine which blocks need to be optimised, vectorised and/or moved to FPGA firmware (HDL). The aim here is to exploit vectorised instructions within the BB SoC hardcore (ie. ARM NEON

VLIW capability [30]) and FPGA softcores (DDC/FFT [31]) to optimise the implementation in order to accommodate more than one signal path on the BB SoC. GNU gprof helps in making the above 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 data rate is increased to (and beyond) the point that CPU starvation sets in. Using a block based waveform realised in pure software, the observed effects of CPU starvation are not catastrophic rather a graceful degradation occurs.

A. *Transmitter Profiling*

Fig. 7 shows the flow of the computationally intensive transmitter blocks implemented on the Xilinx Zynq – Processing System where main() which is streaming the samples and FCsample() which is performing up-sampling reports the maximum CPU consumption.

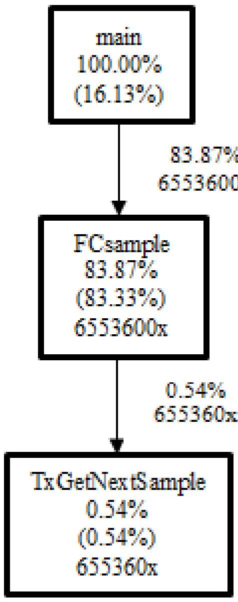


Fig. 7. Computationally Intensive Transmitter Blocks

During profiling, both transmitter and receiver programs are executed for different data rates and on different platforms discussed in Table I such as Zedboard, Odroid-XU Lite and Dell Optiplex 745 to understand the function distribution for higher symbol rates.

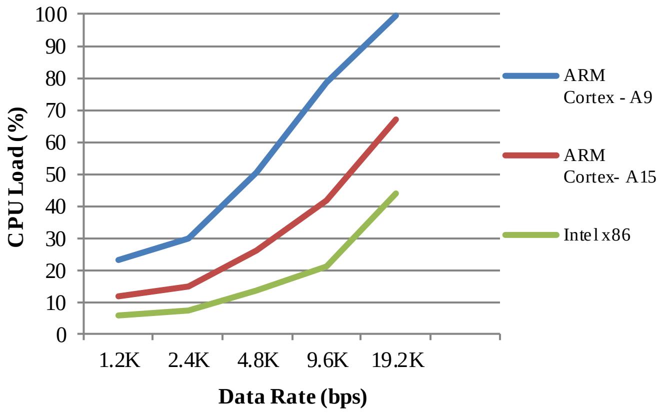


Fig. 8. Absolute CPU Consumption – Transmitter

Fig. 8 gives the comparison of the absolute CPU consumption on dissimilar platforms while the encoder is running at varied data rates. It is evident that the CPU consumption increases along with an increase in the symbol rates. The behaviour appears linear on ARM Cortex A9 operating at 700 MHz, quadratic on ARM Cortex A15/A7 operating at 1.4/1.2 GHz and cubic on Intel x86 operating at 2.13 GHz. This behaviour can also be observed on relative CPU consumption plots of the encoder program across the platforms.

Table II gives the relative comparison of the CPU consumption by different transmitter functions on Dual Core ARM Cortex A9. FCsample() which is up-sampling and main() responsible for streaming the samples and managing buffers are the two dominant functions, with other functions being negligible. Though main() and FCsample() contribute ~50% towards the CPU consumption at 1.2K, the relative contribution of FCsample() increases whereas main() decreases linearly with symbol rate.

The behavior of these functions on “the quad cores” ARM Cortex A15 and A7 appears quadratic. Here the sample streaming is quicker compared to “the dual core” ARM Cortex A9 and therefore the FCsample() dominates over main(). On Intel x86, the sample streaming is the fastest and therefore FCsample() is the only function contributing towards 80-100% of the relative CPU load. The rate of change of CPU consumption by FCsample() reduces on faster platforms with an increase in the symbol rate. Since this is a relative measure of CPU consumption, other functions apart from main() and FCsample() are less prevalent (almost negligible) on all platforms.

TABLE II
RELATIVE COMPARISON OF THE CPU CONSUMPTION BY DIFFERENT TRANSMITTER FUNCTIONS ON DUAL CORE ARM CORTEX A9 (1), DUAL CORE INTEL x86 (2) AND OCTA CORE ARM CORTEX A15 AND A7 (3)

Functions	**1.2K (bps)**			**2.4K (bps)**			**4.8K (bps)**			**9.6K (bps)**			**19.2K (bps)**		
	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3
Main()	52.1	16.25	31.57	42.75	10.33	18.82	30.21	5.16	8.49	20.53	1.8	2.8	14.34	1.18	0.9
FCsample ()	46.99	83.68	66.34	62.38	90.71	78.39	74.7	95.5	88.41	83.19	97.72	94.84	90.04	98.63	98.16

The aim here was to identify the most dominant block which is distinctly FCsample() - performing up-sampling, and this will be moved to FPGA firmware (HDL) for optimisation and thereby enabling multiple-signal transmission.

B. *Receiver Profiling*

Similarly, Fig. 9 shows the computationally intensive functions in the receiver chain (on Intel x86). The down sampling is done in three different stages: main(), in the function called go and RxDownSample() which are the most dominant followed by ProcessFFT() where the appropriate signal is selected. Fig.10 shows the absolute CPU consumption on dissimilar platforms while the decoder is running at varied data rates. The decoder consumes more than 50% CPU at 1.2K on ARM Cortex A9 and reaches almost 100% (appears linear) resulting in low success rate as the symbol rate increases (Fig 11). The behaviour appears quadratic on ARM Cortex A15 and A7 and reaches ~50% at higher data rate.

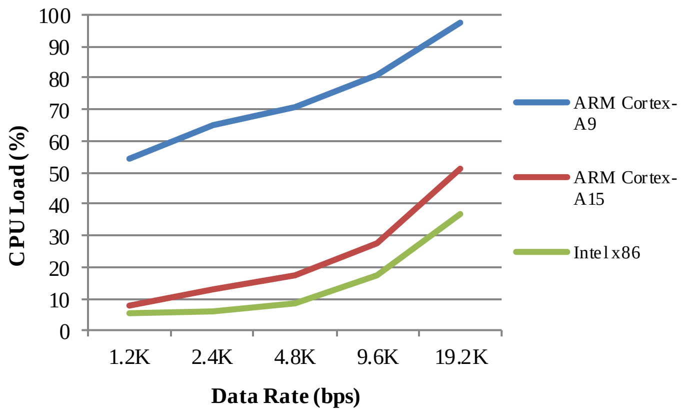


Fig. 10. Absolute CPU Consumption – Receiver

This results in unsuccessful decoding at data rates 9.6K and 19.2K as shown in Fig.11 along with differences in profiling behaviour as shown in Table III. Whereas Intel x86 which exhibits cubic behaviour is well within 50% even at 19.2K and ensuring 100% success rate.

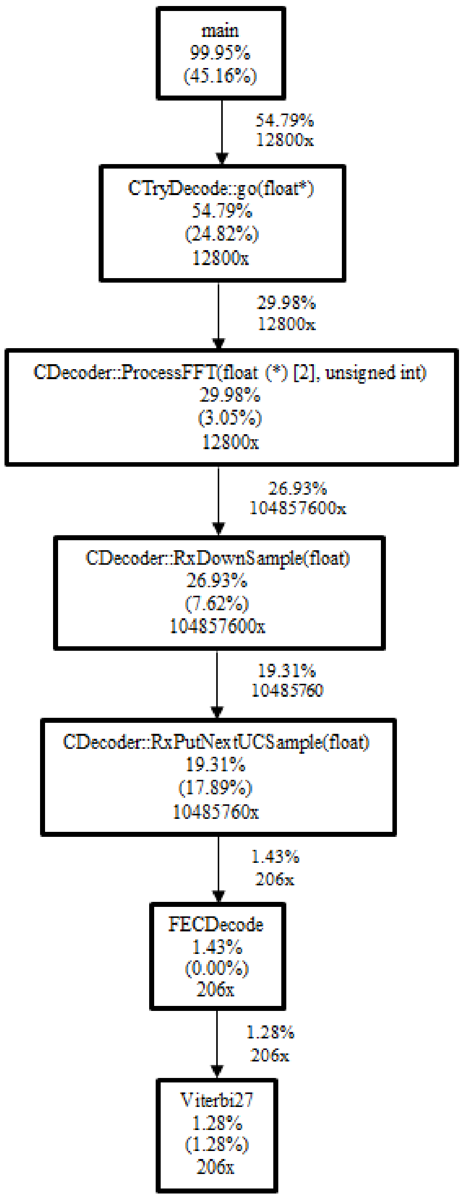


Fig. 9. Computationally Intensive Receiver Blocks

TABLE III
RELATIVE COMPARISON OF THE CPU CONSUMPTION BY DIFFERENT RECEIVER FUNCTIONS ON DUAL CORE ARM CORTEX A9 (1), DUAL CORE INTEL X86 (2) AND OCTA CORE ARM CORTEX A15 AND A7 (3)

Functions	**1.2K (bps)**			**2.4K (bps)**			**4.8K (bps)**			**9.6K (bps)**			**19.2K (bps)**		
	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3
Main()	89.6	61.07	60.24	84.05	54.45	55.82	74.58	50.49	49.02	60.23	47.19	42.56	54.07	44.3	38.97
CTryDecode::go(float*)	6.73	16.42	22.41	8.26	17.33	24.91	14.41	18.94	27.68	25	21.16	32.5	29.63	23.49	36.38
CDecoder::RxPutNextUCSample	1.83	12.88	6.75	3.13	13.97	5.2	3.81	14.87	8.53	5.68	17.47	6.14	8.15	18.52	8
CDecoder::RxDownSample	0.92	5.65	5.06	3.13	8.83	5.5	3.81	8.7	6.78	5.11	8.98	6.37	5.19	9.21	7.53
CDecoder::ProcessFFT	0.61	1.54	5.06	0.85	2.23	6.39	3.39	2.9	8.53	3.98	3.21	5.97	2.96	3.89	7.62

The function main() is decimating the samples from 1.536 Msps to 96 ksps becomes less dominant as the data rate increases on all three platforms whereas the function go() which does further down-sampling from 96 ksps to 96 bps along with Reed Solomon and Viterbi (embedded within FECDecode which works on hard bits but capable of working on soft symbols) is more prevalent on the ARM Cortex A15 and A7 when compared to ARM Cortex A9 and Intel x86 and thus suppressing other functions as seen in Table III. Although there is a significant difference in the relative CPU consumption between two different stages of down-sampling [main() and go()] at 1.2K, the difference gradually reduces and they consume almost the same CPU (~50%) at 19.2K on ARM Cortex A15 and A7 unlike on ARM Cortex A9 Intel x86 where the down-sampling from 1.536 Msps to 96 ksps dominates the other functions. The significant observation here is that the CPU consumption of down-sampling [main()] on ARM Cortex A9 is more than 50% at different data rates and reaches maximum of 90% at 1.2K where the signal is down-sampled to a greater value (16).

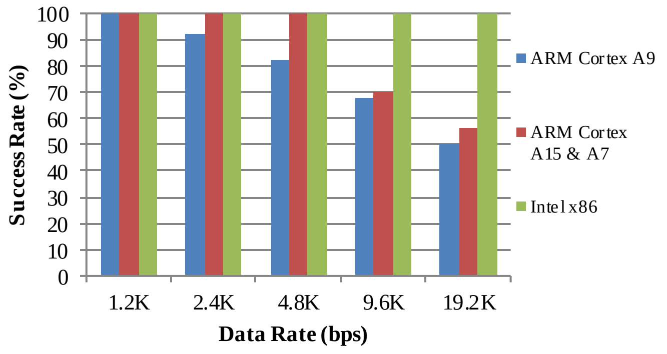


Fig. 11. Success Rate Comparison on Different Architectures

In addition, the compilers were found to be different across the platforms as seen in Table IV. There is a difference in the instruction sets used across various architectures to perform similar functions and was observed using ‘objdump’. On 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 to the memory and decimating the samples would make the design more efficient.

V. Implementation of DDC Block in FPGA

Based on the profiling results obtained earlier it is evident that the up-sampling and down-sampling are computationally intensive blocks in the transceiver. The architecture was 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 the sample DDC block. The reference design includes the core from Analog Devices which fetches the samples from RF SoC interface core and provides them to Zynq PS for further processing. The sample DDC block was implemented in between RF core (AXI_AD9361) and sample packer block which packs I and Q

TABLE IV
Comparison of Instruction Set Across Different Platforms

	Dell Optiplex 745	**Odroid XU Lite**	**Zedboard**
C Compiler	GCC 4.8.2	GCC 4.8.2	GCC 4.6.3
No. of different Instructions	88	150	144
No. of Similar Instructions across the platforms	9	9	9
	-	131	131
Dominant Instructions (top 5)	mov(693) callq(186) add(130) movss(123) cmp(100)	add(238) ldr(235) mov(156) movw(146) movt(141)	ldr(256) add.w(246) movw(176) add(169) mov(155)

signals from different channels before the signal is stored in Direct Memory Access (DMA). Other blocks such as modulation/demodulation, frequency/phase correction and packet handling which are computationally less intensive were retained in ARM Cortex A9 processors.

A. Post-Profiling Results

Once the DDC block was implemented on the FPGA fabric, profiling was repeated to understand the improvement achieved. Fig. 12 shows the percentage reduction in the absolute CPU consumption at different data rates. This improvement in the average load allows parallel reception of up to 5 signals (without accounting for instantaneous peak load), so upto 4 signals could be a better expectation at 1.2K while it was limited to one earlier. Similarly, two signals at 2.4K can be decoded simultaneously in place of single signal.

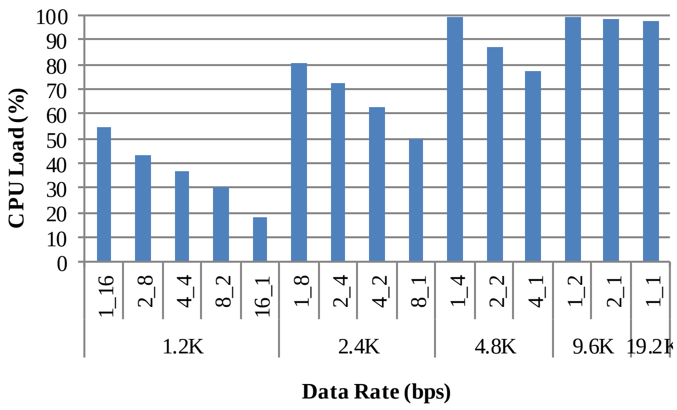


Fig. 12. Absolute CPU Consumption - ARM Cortex A9

TABLE V
Improvement Achieved with Sample DDC Block on FPGA

Data Rate	**Average reduction in the CPU consumption**
1.2K	36.76%
2.4K	31.14%
4.8K	21.5%
9.6K	0.7%

TABLE VI
POST PROFILING RESULTS ON DUAL CORE ARM CORTEX A9

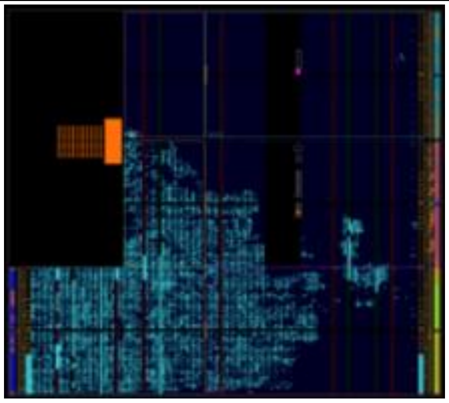
Functions	1.2K (bps)					2.4K (bps)				4.8K (bps)			9.6K (bps)		19.2K (bps)
h/w_s/w	**1_16**	**2_8**	**4_4**	**8_2**	**16_1**	**1_8**	**2_4**	**4_2**	**8_1**	**1_4**	**2_2**	**4_1**	**1_2**	**2_1**	**1_1**
Main()	89.6	83.25	71.09	63.74	50	84.05	73.57	61.49	56.72	74.58	60.23	57.14	60.23	52.59	54.07
CTryDecode::go(float*)	6.73	9.42	18.75	21.98	30.77	8.26	17.62	22.41	28.36	14.41	23.39	27.07	25	28.15	29.63
CDecoder::RxPutNextUCSample	1.83	4.19	4.69	7.69	8.97	3.13	3.96	6.32	5.97	3.81	7.6	7.52	5.68	10.37	8.15
CDecoder::RxDownSample	0.92	1.57	3.91	5.49	6.41	3.13	2.64	5.75	4.48	3.81	5.26	6.02	5.11	5.93	5.19
CDecoder::ProcessFFT	0.61	1.05	1.56	1.1	3.85	0.85	1.32	4.02	4.48	3.39	2.34	1.5	3.98	2.96	2.96

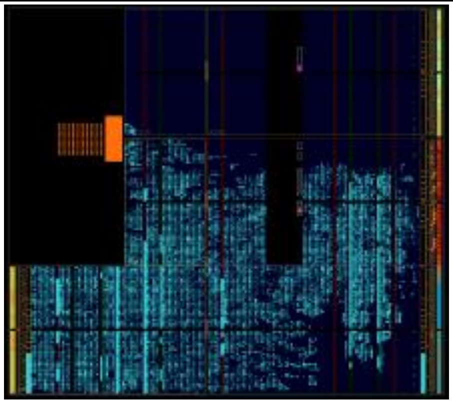
The 1st digit in the x-axis stands for hardware decimation and the 2nd digit for software decimation. It is unambiguous that as the hardware decimation increases the CPU consumption decreases. Table V summarizes the improvement achieved at different data rates. Similar progress can be seen in relative performance measures as seen in Table VI, main() which was contributing towards 89.6% of the CPU is now reduced to 50% with hardware decimation at 1.2K, from 84.05% to 56.72 % at 2.4K, 74.58% to 57.12% at 4.8K and from 60.23% to 52.59% at 9.6K.

Table VII shows FPGA processor logic utilisation before and after the front end DDC function being moved to firmware and includes the percentage increase in FPGA utilisation that results. Adding a sample DDC block to the original increased the power consumption and the hardware requirements. Total overhead of on-chip power is 13.14% with 5% increase in flip-flops and memory LUTs, 8% increase in LUTs, 18% increase in BRAMs and 3% increase in DSPs. This analysis suggests that approximately 4-5 DDC/DUCs can be implemented in order to aid parallel reception. Here, we

TABLE VII
OVERHEAD ANALYSIS OF DDC IMPLEMENTATION

Original Design (Software DDC)		With DDC Block on FPGA		Overhead	
Power:		**Power:**		**Power:**	
• Total On-Chip Power	: 2.2 W	• Total On-Chip Power	: 2.489 W	• Total On-Chip Power	: 13.14%
• Dynamic Power	: 2.03 W	• Dynamic Power	: 2.309 W	• Dynamic Power	: 13.74%
• Device Static	: 0.17 W	• Device Static	: 0.180 W	• Device Static	: 5.88%
Post Implementation:		**Post Implementation:**		**Post Implementation:**	
• Flip Flop	: 19%	• Flip Flop	: 24%	• Flip Flop	: 05%
• LUT	: 24%	• LUT	: 32%	• LUT	: 08%
• Memory LUT	: 04%	• Memory LUT	: 09%	• Memory LUT	: 05%
• I/O	: 61%	• I/O	: 61%	• I/O	: 0%
• BRAM	: 06%	• BRAM	: 24%	• BRAM	: 18%
• DSP48	: 31%	• DSP48	: 34%	• DSP48	: 03%
• BUFG	: 28%	• BUFG	: 28%	• BUFG	: 0%
• MMCM	: 50%	• MMCM	: 50%	• MMCM	: 0%


Implementation on FPGA Fabric


Implementation on FPGA Fabric

use the Zynq 7020 but in case of more number of signals with higher data rates, a larger FPGA may be selected [32].

VI. Conclusions

This paper presents the design and implementation of an adaptive SDR architecture on different platforms with varied symbol rates such as 1.2K, 2.4K, 4.8K, 9.6K and 19.2K. C/C++ was preferred over VHDL for initial implementations due to the reduced implementation time 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. Also, the functions exhibit diverse behaviour such as linear/quadratic and cubic on dissimilar platforms. The obtained performance results demonstrate the need to move blocks demanding higher computation capacity such as up/down sampling blocks.

The sample DDC block was moved to FPGA and the post-profiling results show the improvement in the performance thereby facilitating more than one signal at any given time. The significant improvement being at lower data rates such as 36.76% at 1.2K and 31.14% at 2.4K. This comes with a cost of 13.14% more on-chip power and 5 -15 % 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 multiple satellite reception at typical CubeSat telemetry rates.

Future work includes the implementation of the proposed design with n-stage pipeline architecture on FPGA SoC as shown in Fig. 13 based on different stages of transmission synchronisation. The objective of the pipeline architecture is to receive the signal from more than one satellite operating at different modulation techniques, data rates and centre frequencies. RF SoC would acquire the desired signal present in the spectrum with predefined software configuration of the frontend such as gain, filters, bandwidth and centre frequency.

The next stage in the architecture includes parallel wrappers of DDCs consisting of Digital Quadrature Tuner (DQT) and Cascaded Integrator Comb (CIC) blocks. Each valid signal in the spectrum is mapped to separate channel based on the available on-chip resources. Each signal is stored under different offset address in the DMA which is configured according to the pre-calculated memory requirements. The last stage in the architecture is proposed to be asynchronous as the signal stored in the DMA can be accessed independently using different decoder threads running on dual core processors.

Using a single programmable baseband SoC to execute several baseband processing programs at the same time can benefit in 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 combined FPGA logic and processor must have the resources to support the worst case load in all supported standards simultaneously.

In conclusion, this paper 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 satellite communications. Potential applications of the proposed embedded system architecture are the ground station for multi-satellite

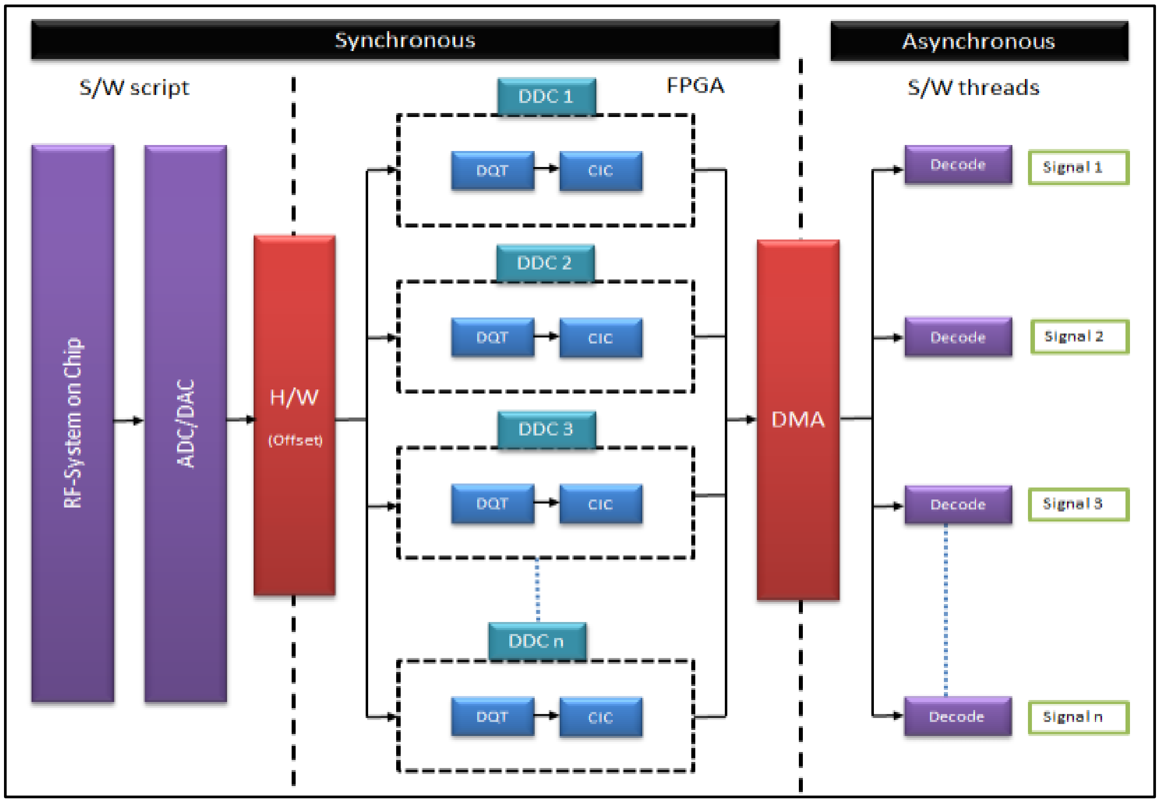


Fig. 13. Future Pipeline Architecture

communications, deployable mobile ground station network and can be further extended to distributed satellite system

ACKNOWLEDGMENT

The authors would like to thank Dr. Brian Yeomans, Research Fellow, Surrey Space Centre for his valuable time, discussions and contributions towards the paper.

REFERENCES

[1] R. Sandau, K. Brieß, and M. D'Errico, "Small satellites for global coverage: Potential and limits," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 65, no. 6, pp. 492-504, Nov, 2010.

[2] D. J. Barnhart, T. Vladimirova, and M. N. Sweeting, "Very-Small-Satellite Design for Distributed Space Missions," *Journal of Spacecraft and Rockets*, vol. 44, no. 6, pp. 1294-1306, Nov, 2007.

[3] (2015) Planet Labs: FLOCK 1 - CubeSat Satellite Constellation, San Francisco, California, USA [Online] Available: https://www.planet.com/flock1/

[4] H. Bedon, C. Negron, J. Llantoy, C. M. Nieto, and C. O. Asma, "Preliminary internetworking simulation of the QB50 cubesat constellation," in Communications (LATINCOM), 2010 IEEE Latin-American Conference on, 2010, pp. 1-6.

[5] C. Underwood, S. Pellegrino, V. Lappas, C. Bridges, B. Taylor, S. Chhaniyara, T. Theodorou, P. Shaw, M. Arya, and J. Breckinridge, "Autonomous Assembly of a Reconfiguarble Space Telescope (AAReST) – A CubeSat/Microsatellite Based Technology Demonstrator," presented at Small Satellite Conference, Utah State University, Logan, UT, USA, 2013. [Online]. Available: http://digitalcommons.usu.edu/cgi/viewcontent.cgi?article=2952&context=smallsat

[6] C. P. Bridges, B. Taylor, N. Horri, C. I. Underwood, S. Kenyon, J. Barrera-Ars, L. Pryce, and R. Bird, "STRaND-2: Visual inspection, proximity operations &amp; nanosatellite docking," in *Proc. Aerospace Conference, 2013 IEEE*, 2013, pp. 1-8.

[7] H. B. Smith, S. H. K. Hu, and J. J. Cockrell, "NASAs EDSN Aims to Overcome the Operational Challenges of CubeSat Constellations and Demonstrate an Economical Swarm of 8 CubeSats Useful for Space Science Investigations," presented at Small Satellite Conference, Utah State University, Logan, UT, USA, 2013. [Online]. Available: http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20140009202.pdf

[8] C. Cordeiro, K. Challapali, D. Birru, and N. Sai Shankar, "IEEE 802.22: the first worldwide wireless standard based on cognitive radios," in New Frontiers in Dynamic Spectrum Access Networks, 2005. DySPAN 2005. 2005 First IEEE International Symposium on, 2005, pp. 328-337.

[9] T. De Cola, and M. Marchese, "Performance analysis of data transfer protocols over space communications," *Aerospace and Electronic Systems, IEEE Transactions on*, vol. 41, no. 4, pp. 1200-1223, Oct, 2005.

[10] M. J. Marcus, "Spectrum issues in FCC'S national broadband plan [Spectrum Policy and Regulatory Issues]," *Wireless Communications, IEEE*, vol. 17, no. 2, pp. 6-6, April, 2010.

[11] J. Mitola, "Software radio architecture: a mathematical perspective," *Selected Areas in Communications, IEEE Journal on*, vol. 17, no. 4, pp. 514-538, April, 1999.

[12] K. Leveque, J. Puig-Suari, and C. Turner, "Global Educational Network for Satellite Operations (GENSO)," presented at Small Satellite Conference, Logan, UT, USA, 2007. [Online] Available: http://digitalcommons.usu.edu/cgi/viewcontent.cgi?article=1506&context=smallsat

[13] (2015) SatNOGS: Satellite Networked Open Ground Station, Athens, Greece. [Online]. Available: https://satnogs.org/

[14] (2015) Avnet – Zedboard: A Development board for Xilinx Zynq-7020, Phoenix, AZ, USA. [Online]. Available: http://zedboard.org/

[15] (2015) Lime Microsystems: Field programmable RF ICs LMS6002D, Guildford, Surrey, United Kingdom. [Online]. Available: http://www.limemicro.com/products/field-programmable-rf-ics-lms6002d/

[16] (2015) Analog Devices: AD-FMCOMMS3-EBZ: AD9361 Software development Kit, Norwood, MA, USA. [Online]. Available: http://www.analog.com/en/evaluation/eval-ad-fmcomms3-ebz/eb.html

[17] (2015) Lime Microsystems: Field programmable RF ICs LMS7002M, Guildford, Surrey, United Kingdom. [Online]. Available: http://www.limemicro.com/products/field-programmable-rf-ics-lms7002m/

[18] (2015) FUNcube: UK Amateur Radio Educational Satellite, Blandford, Dorset, UK [Online]. Available: http://funcube.org.uk/overview/contact-us/

[19] P. Karn (2002, Jan 7) *Proposed Coded AO-40 Telemetry Format (v1.2)*. [Online]. Available: http://www.ka9q.net/papers/ao40tlm.html

[20] S. W. Shaker, S. H. Elramly, and K. A. Shehata, "FPGA implementation of a configurable viterbi decoder for software radio receiver," in *Proc. AUTOTESTCON, 2009 IEEE*, 2009, pp. 140-144.

[21] J.Miller (2003) *Oscar-40 FEC Telemetry*. [Online]. Available: http://www.amsat.org/amsat/articles/g3ruh/125.html

[22] (2015) Analog Devices: AD-FMCOMMS3-EBZ User Guide, Norwood, USA. [Online]. Available: http://wiki.analog.com/resources/eval/user-guides/ad-fmcomms3-ebz

[23] (2015) Analog Devices: GitHub-Library for interfacing with IIO devices. [Online]. Available: https://github.com/analogdevicesinc/libiio

[24] (2015) FUNcube Dongle: FUNcube Dongle Pro+, Blandford, U.K. [Online]. Available: http://www.funcubedongle.com/?page_id=1073

[25] (2015) Rohde & Schwarz: SMBV100A Vector Signal Generator, Berkshire, U.K. [Online]. Available: http://www.rohde-schwarz.co.uk/en/product/smbv100a-productstartpage_63493-10220.html

[26] A.Csete (2015) *Fcdec-GitHub: FUNcube telemetry decoder for Linux*. [Online]. Available: https://github.com/csete/fcdec

[27] (2015) Hardkernel: Odroid-XU Lite, [Online] Available: http://www.hardkernel.com/main/products/prdt_info.php?g_code=G138503207322

[28] (2015) Analog Devices: GitHub-Library for IIO devices. [Online]. Available: https://github.com/analogdevicesinc/libiio/tree/master/iiod

[29] (2015) GNU gprof: Profiling tool. [Online] Available: https://sourceware.org/binutils/docs/gprof/

[30] L. Codrescu (2013) *Qualcomm Hexagon DSP: An architecture optimized for mobile multimedia and communications* [Online]. Available: http://pages.cs.wisc.edu/~danav/pubs/qcom/hexagon_hotchips2013.pdf

[31] P. Wang, J. McAllister, and Y. Wu, "Soft-core stream processing on FPGA: An FFT case study," in Proc. Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on, 2013, pp. 2756-2760.

[32] (2015) Zynq-7000 All programmable SoC overview. [Online]. Available: http://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf

[33] M. R. Maheshwarappa and C. P. Bridges, "Software defined radios for small satellites," Adaptive Hardware and Systems (AHS), 2014 NASA/ESA Conference on, Leicester, 2014, pp. 172-179.

[34] M. R. Maheshwarappa, M. Bowyer and C. P. Bridges, "Software Defined Radio (SDR) architecture to support multi-satellite communications," Aerospace Conference, 2015 IEEE, Big Sky, MT, 2015, pp. 1-10

[35] GNURadio. *Free and Open Source Toolkit for Software Radio*. Available: http://gnuradio.org

[36] P. Angeletti, M. Lisi, and P. Tognolatti, "Software Defined Radio: A key technology for flexibility and reconfigurability in space applications," in *Metrology for Aerospace (MetroAeroSpace), 2014 IEEE*, 2014.

[37] M. Cheffena and L. E. Bråten, "Low-cost digital beacon receiver based on software-defined radio," *IEEE Antennas and Propagation Magazine*, vol. 53, pp. 50-55, 2011.

[38] I. Lucresi, A. Di Carlofelice, and P. Tognolatti, "SDR-based system for satellite ranging measurements," *IEEE Aerospace and Electronic Systems Magazine*, vol. 31, pp. 8-13, 2016.

[39] L. L. Presti, P. di Torino, E. Falletti, M. Nicola, and M. T. Gamba, "Software defined radio technology for GNSS receivers," in *Metrology for Aerospace (MetroAeroSpace), 2014 IEEE*, 2014, pp. 314-319.

[40] F. Pinto, F. Afghah, R. Radhakrishnan, and W. Edmonson, "Software Defined Radio implementation of DS-CDMA in inter-satellite communications for small satellites," in *Wireless for Space and Extreme Environments (WiSEE), 2015 IEEE International Conference on*, 2015, pp. 1-6.

[41] E. L. Valles, K. Tarasov, J. Roberson, E. Grayver, and K. King, "An EMWIN and LRIT software receiver using GNU radio," in *2009 IEEE Aerospace conference*, 2009, pp. 1-11.

BIOGRAPHY

Mamatha R. Maheshwarappa (B.Eng, 2009; MSc 2012) received her BEng in Electronics and Communication Engineering from Nitte Meenakshi Institute of Technology, Bangalore in 2009. She was the system engineer for STUDSAT-1- India's first pico-satellite, designed and developed by undergraduate students. Later she received MSc in Space Technology and Planetary Exploration from the University of Surrey in 2012 and currently a PhD student at Surrey Space Centre, University of Surrey. Her research is in Software Defined Radios for multi-satellite links.

Dr Mark Bowyer (BEng, 1987; PhD, 1993) is a Senior Expert in Secure Communications at Airbus defence and Space working in the Communications, Intelligence and Security business unit. He has over 20 years work experience in Satellite Communications (SATCOMS) Systems. He has led activities in SATCOM Software Defined Radio R&D since its inception in 1997. This has included development of three generations of hardware platform and software applications. Today, he consults on all aspects of secure SATCOM including RF, signal processing and waveform design for ground, air, and space-based modem systems. Mark holds a BEng (1st) from Birmingham University and PhD from the University of Kent at Canterbury.

Dr Christopher P. Bridges (BEng, 2005; PhD 2009) leads the On-Board Data Handling (OBDH) research group within Surrey Space Centre (SSC). He researches software defined radios, real-time embedded systems, agent computing, Java processing, multi-core processing in FPGAs, and astrodynamics computing methods in many spaceflight payloads. In 2013, he designed, built and still operates the UK's first CubeSat (STRaND-1) with SSTL and now contributes towards computing hardware and software in missions with SSTL, on ESA's ESEO mission and also the NASA-JPL/CalTech AaRREST mission.