Dual Core System-on-a-Chip Design to Support Inter-Satellite Communications

Christopher P. Bridges and Tanya Vladimirova
Surrey Space Centre
Department of Electronic Engineering
University of Surrey
Guildford, UK GU2 7XH
Email: {C.P.Bridges, T.Vladimirova}@surrey.ac.uk

Abstract

The next generation of satellites for distributed satellite missions will exploit the latest computing and wireless technologies for intersatellite connectivity. These missions enable opportunities in multiple-point sensing, greater communications capabilities and spacecraft redundancy. Requirements for processing and network capabilities have risen dramatically to meet strict needs of the end user and overcome various space disturbances and perturbations once in orbit. One such problem lies with a 'cluster' of satellites that have been deployed from the same launcher where they will be close together so ad-hoc technologies allow satellite communication. This paper addresses the hardware and software requirements for distributed computing opportunities using intersatellite connectivity. A system-on-a-chip design is proposed; including a general purpose processor core and a dedicated Java processing core to adapt and reconfigure the topology using real-time software Agent applications. This will make the network resilient to various space perturbations and ensure mission longevity. Integration of these two non-heterogeneous cores in a picosatellite technology demonstrator testbed and network topology reconfigurability procedures are also outlined.

1. Introduction

Distributed satellite systems (DSS) aim at offering a number of unique mission advantages including redundancy, lower cost, flexibility, multi-point sensing and greater communication capabilities [1, 2]. There are many types of DSS including:

- Formation Flying where a very strict formation is required to perform a mission, such as that found in synthetic aperture radar (SAR);
- Clustering Missions where satellites are loosely coupled around each other to perform a mission;
- Virtual Satellite Missions (also called fractionated missions) where a satellite has its subsystems divided onto multiple craft to perform a mission. E.g. one craft for computing, one craft for imaging, etc.

But these advanced missions have some very challenging functional requirements including attitude and orbit control, intersatellite links and flexible on-board computing.

A distributed satellite mission in low Earth orbit (LEO) is considered where multiple very small satellites are deployed at the same time in multiple planes and orbits to form a cluster of satellites. These satellites can then form an ad-hoc network for multiple-point sensing of Earth's atmosphere; analogous to a 'wireless sensor network' (WSN) [3]. Like a WSN, there is typically one communications link or 'sink' to an end user, in this case, a groundstation. The need for a cluster of satellites here is to ensure data can be recorded from multiple payloads at a specific time at various non-specific local locations. An example scenario is found in Fig. 1 and can be expanded to include tens or even hundreds of satellites, as envisioned by NASA's ANTs Mission [4].

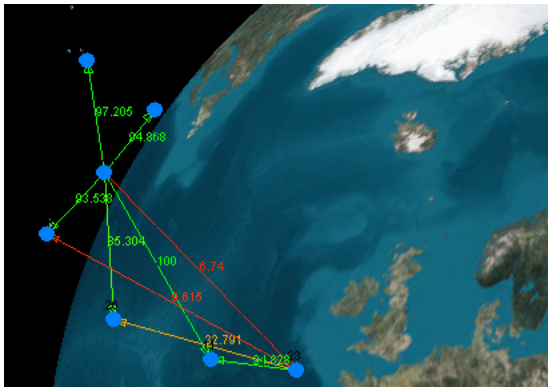

Figure 1. An example Cluster Scenario

Satellite drift along with other perturbations such as Earth's geophysical forces, variations in Earth's atmosphere and solar radiation pressure greatly affect the

978-0-7695-3166-3/08 $25.00 © 2008 IEEE
DOI 10.1109/AHS.2008.62

relative distances between the satellites and thus the network's connectivity and topology over time. When initially deployed from the launcher the satellite network will be close together with good connectivity. A 'server' satellite is typically predefined and other 'client' satellites can connect to it. But over many orbits, perturbations will affect the satellite network's connectivity by their relative distances and the network topology so that the server satellite may not be the most central to the network. At this point, connectivity to ground can be poor and intermittent with large periods in a 'disconnected' environment. If the server satellite is no longer the optimal central satellite or if the server satellite is damaged/ cannot perform its role, a new satellite or configuration is needed. To solve this problem, a dual-core processor with network reconfigurability functionality is proposed for distributed computing operations.

The network and distributed computing data requirements for the chosen cluster scenario can be summarized as follows:

Node Level Functionality Requirements (individual satellite level):

- Distributed computing platform
- Storing and forwarding of data using distributed computing paradigms; including:
  - 'High Data' or 'High Priority' Applications using a Client/ Server paradigm – Payload data through the network such as imaging data, larger science payloads & data aggregation.
  - 'Low Data' Applications using a Peer-to-Peer (P2P) paradigm Telemetry, location and velocity changes such as telemetry, "byte" size payload data (GPS, science payload measurements) & network management data.

Network Level Functionality Requirements (multiple satellite level):

- Ad-hoc intersatellite networking capabilities for initial formation.
- Adaptable and redundant ground-link communication schemes, i.e. main 'sink' to ground.
- Proactive and reactive topology schemes to mobility.

The development of an effective space network will also require other characteristics from mobile ad-hoc networks (MANETs) including network mobility and scalability.

The latest terrestrial distributed and networked systems are now using Agent systems to cope with large scale remotely located services and systems [5]. Relevant Agents systems are included in groundstations to aid in image signal processing [6] and on-board satellites for in-situ autonomous behaviours [7]. Agents are a higher abstraction of programming to deal with complex computing problems. By executing behaviorally and assigning an agent a 'role', communication interactions and autonomous actions become easier to realize. This allows them to work 'proactively' and 'reactively' to their environment and to any given task. They can be proactive when finding new communications routes in a networked environment and reactive to disconnections, low bandwidths or high latencies [8]. Unlike a typical distributed computing platform that typically has fixed network characteristics, agents could be employed to overcome and discover their given network situation. On a satellite, as few assumptions as possible should be made for the connection reliability over inter-satellite links (bandwidth, latencies, number of connections, etc). An agent could be designed to discover these characteristics and make the software and network operations more reliable and robust.

In practical experiments, the Java Agent DEvelopment framework and the Light Extensible Agent Platform (JADE-LEAP) [9] is adopted to communicate over an ad-hoc IEEE 802.11 wireless link using a number of protocols over multiple laptops. JADE-LEAP is a Java based Agent Development environment middleware for embedded devices and other resources constrained devices using wireless links to develop Agent systems and novel application areas such as web or service orientated computing. The JADE-LEAP platform is used here due to its light-weight footprint, its conformity to the Foundation of Intelligent and Physical Agents (FIPA) specifications [10] and a large community of users.

Agent systems are written in Java, which is unsuited to real-time mission-critical embedded systems due to problems with large standard libraries, a slow and undeterminable execution model, and dynamic class loading execution times; to name a few reasons. However, in recent years, improvements in library size and memory technologies have enabled Java on mobile phones and PDAs (without floating point support and other Java functionalities). But these tools are still not real-time or time predictable. To counter this problem, a deterministic Java processor is targeted to run Agents for real-time applications.

This paper, carried out under the ESPACENET project [11], presents a system-on-a-chip (SoC) design implemented in a field programmable gate array (FPGA) comprising a general purpose processor and a specific processor for real-time Java enabled computing; targeting a picosatellite demonstration mission. Section 2 describes the picosatellite demonstrator design. Section 3 presents

the dual-core processor design. Section 4 discusses the integration between the general purpose processor core and dedicated Java processing core. Section 5 introduces network topology reconfiguration issues. Section 6 concludes the paper.

2. Picosatellite Demonstrator Design

Embedded hardware technology is now available for designing, building and launching picosatellites. This project aims at using the standard picosatellite platform CubeSat [12]. For fast prototyping, commercial-off-the-shelf (COTS) components/boards are chosen, to develop a technology demonstrator testbed. Parts used in the design include:

- Flight OBC and satellite chassis from Pumpkin [13]
- Power module from Clyde-Space [14]
- SGR-05 GPS module from SSTL [15]
- MHX transceiver from Microhard Systems [16]
- PF5100 Virtex-4 FPGA FX60 Board for SoC [17]
- IEEE 802.11 PC/104 Board from Elcard [18]

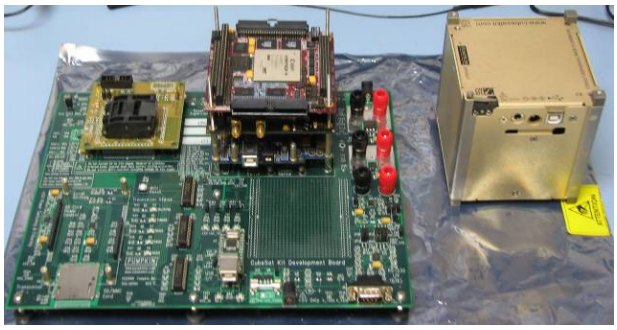

Figure 2. CubeSat Platform with Flight Module, IEEE 802.11 Board and FPGA Board

The CubeSat platform is a 10 x 10 x 10 cm standard bus structure weighing at 1 kg, which is compatible with the PC104 format. The CubeSat bus also comes in double (2U) and triple unit (3U) sizes to conform to the P-POD deployment mechanism. The CubeSat Kit platform provides a standard COTS solution to develop new technologies. Research into all current CubeSat missions shows that reliability and simplicity are key requirements to ensure success, whilst having more complex systems as separate payloads. This design follows trends to ensure that our satellite can achieve multiple objectives: from successful deployment, establishing communications and turning on/ off experimental payloads. The current demonstrator design can be seen in Fig. 2, with the Flight Module, IEEE 802.11 PC/104 board and FPGA Board attached on the left, for integration, and the structure on the right, depicting the final satellite.

The picosatellite nodes must be computationally able to run code on the satellite to optimize the network's ability to perform the mission (e.g. make decisions from complex algorithms to decide which satellite has the resources to communicate to ground) using intersatellite links. This will require additional hardware such as CPU, storage and RAM memory, along with added software resources such as an operating system, distributed computing environment and applications; all constrained in the CubeSat dimensions.

For a 'technology demonstration' mission, the payload design must include all the necessary components for the complete distributed computing platform, with communication capabilities, power systems and payloads. To ensure reliability in space, a new COTS based satellite bus architecture is proposed that treats the FPGA board, the IEEE 802.11 communications board and a camera as payloads, as shown in Fig. 3.

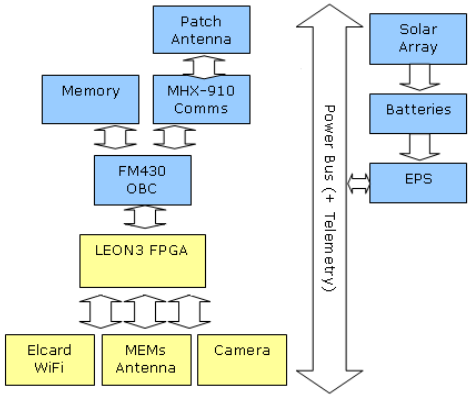

Figure 3. Demonstrator Satellite Architecture

This new COTS architecture is primarily controlled by the Flight Module board (FM430 OBC) and uses the SoC design to act as a hardware and software mediator for differing payload modes in the mission. These differing modes can include soft resets or various sleep modes but also hard resets and on/off switching for varying duty cycles in orbit. This will ensure that payloads can be precisely controlled. Due to the COTS nature of the design, the SoC board is also used to interface between various buses such as I2C, SPI, PCI and Ethernet for this demonstrator.

3. Dual-Core Processor Design

The proposed solution for this computing problem is a dual-core system to provide node level and network level functionality. The node level functionality is achieved using the LEON3 processor [19]; a general purpose soft-core processor from Gaisler Research which is a fully

compliant SPARC V8 architecture with various functionality through an extensive IP library. The LEON3 processor, adopted by ESA as the main CPU for future on-board computers since the end of 2006 [20], is used for typical processing and 'number crunching' capabilities such as image compression. As previously described, to accommodate an agent computing environment, a Java Virtual Machine (JVM) or Java Runtime Environment (JRE) is required to provide a heterogeneous platform on which to realise a communication medium between various platforms.

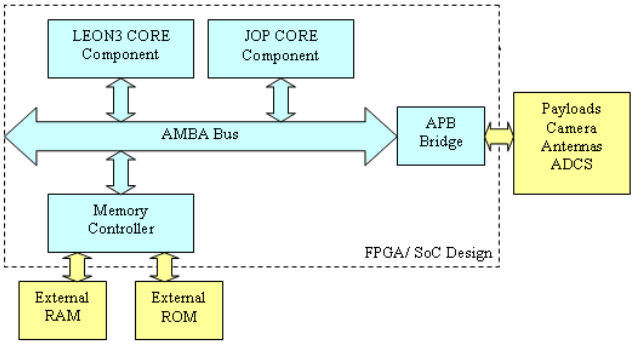

Figure 4. Overview of LEON3 and JOP Design

As 'Just in Time' (JIT) compilation at runtime is far from time deterministic for embedded real-time critical systems, this work also presents the use of an open-source Java specific processor called Java Optimised Processor (or JOP) [21] to enable real-time Java functionality on-board satellite systems. JOP is chosen as it is the smallest and fastest Java core to date. JOP is a RISC and stack based architecture used to execute Java bytecodes using microcode instructions. By utilizing JOP in an FPGA along with the LEON3 processor, the benefits include moving software to hardware (reducing the memory footprint with increased FPGA utilisation and increased speedup) and enabling Java applications, such as Agents, for real-time applications. The system architecture can be seen in Fig. 4 where JOP is integrated for a communications co-processor with the LEON3 using the AMBA bus.

3.1. Design Considerations

In the FPGA, there must be a memory sharing system between the two cores for access to external RAM. To reflect the multi-layered software design (see Fig. 5), a clearly defined inter process communication (IPC) or bus scheme is also required. In software, caches between the two cores must retain coherency and in the event of a core failure, the other must be able to act for a soft recovery or even reconfigure in the event of a single effect upsets or latch-ups (SEUs and SELs respectively) [3].

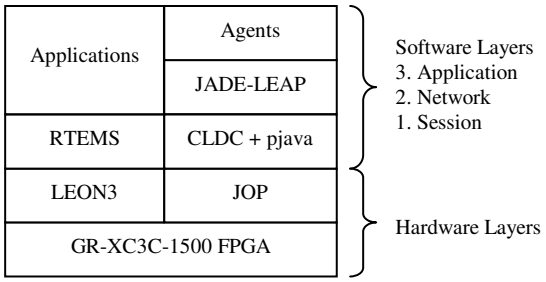

Figure 5. Hardware & Software Layer Design

The systems must have a low memory footprint (with operating environment and network stack) and still be real-time. A comparison of the memory footprint and functionality which looks at previous solutions to this problem can be found in Table 1, where there are three options considered:

1. A CORBA Middleware based implementation [22];
2. The standard Java libraries and software runtime used by PCs;
3. A new hardware/ software co-design where the standard Java runtime is replaced by hardware with CLDC and pjava.

Connection Limited Device configuration (CLDC) [23] and PersonalJava (pjava) are designed for the devices with intermittent network connections, slow processors and limited memory such as mobile phones, two-way pagers and PDAs – making them ideal to run in real-time on the JOP processor. These devices require either 16-or 32-bit CPUs and a minimum of 128 KB to 512 KB of RAM for the Java platform implementation and associated applications. The full JRE 1.4 requires over 15 MB alone and is a major deterrent for using Java on embedded devices but dynamic class parsers are now available to help minimize the application to a very small size. From Table 1, it can be seen that the third option offers the smallest memory footprint whilst retaining real-time functionality. Here, pjava, CLDC 1.0, JADE-LEAP and designed Agents has been reduced to 1.1 MB.

Table 1. Memory Footprint Comparison

OSI Software Layer Method	Size (MB)	Real-time
1. Full Software using CORBA (LEON3 + RTEMS, C++, ORB, 802.11 Driver, TCP/IP, Dyn. Lib.) [22]	1.739	Y
2. Full Software using Java (LEON3 + RTEMS, JRE 1.4 Std. Lib, CLDC 1.0)	>16.000	N
3. Hardware/ Software Co-Design (LEON3 + JOP + pjava, CLDC 1.0 + JADE-LEAP)	1.106	Y

3.2. Shared Memory and Caches

Multi-core system designs use shared memory for a fast form of IPC between the cores. Once the memory has been mapped, core synchronization is required between the processes for storing or fetching data to and from shared memory. The synchronization is implemented using the open-source AMBA2 Bus from ARM [24]; and more specifically, the Advanced High-Performance (AHB) Bus. The AHB Bus acts as the backbone in many SoC cores and is adopted here to provide connection between the cores, on-chip memories and off-chip external memory interfaces from various memory vendors. The AHB Bus operates bus arbitration to AHB Masters and Slaves where the core is first requested, addressed, granted access, and locked for use before finally being released to the arbiter.

The LEON3 system implements a standard data and instruction cache but JOP implements a 'stack' cache for data and 'method' cache for instructions which is designed for real-time worst case execution time (WCET) analysis. JOP's unique design, a hardware implementation of a JRE (at v 1.1) implements a simplified garbage collection (GC) model using the Real-Time Specification for Java [25] (RTSJ) which schedules a GC thread for automatic memory management. JOP's 512 KB cache size was determined by a previous analysis of JRE 1.1 method lengths being 98% < 512 KB [26]. The fault tolerant version of LEON3 has a configurable cache and memory system designed to be tolerable to SEUs or SELs in the space environment with protected on-chip memories using triple modular redundancy (TMR), parity checking or duplication [27]. As an enhancement to JOP's functionality, the existing JRE must be upgraded to a minimum of version 1.4 to run the JADE-LEAP Agent environment. To achieve this, new methods of the CLDC Stack have been added to JOP's instruction set at a simulation level to better support networking.

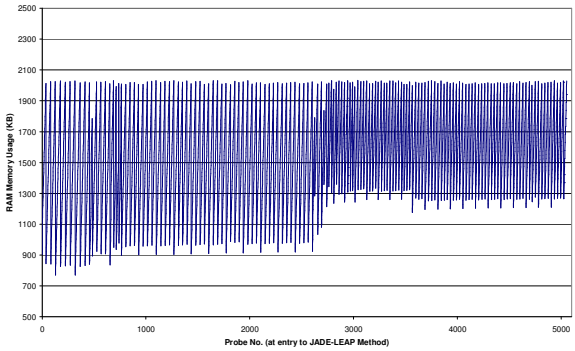

Figure 6. JADE-LEAP Memory Usage under CDC-1.0

An analysis of JADE-LEAP methods was performed to check the heap usage, specifically the DRAM allocated for the application, to show that JADE-LEAP requires between 750 KB to 2.1 MB of memory as shown in Fig. 6. As this memory size cannot be implemented on-chip, off-chip solutions are required. The current SoC solution can be seen in Fig. 7. where the cores implemented and memories have been determined. The Virtex-4 FPGA

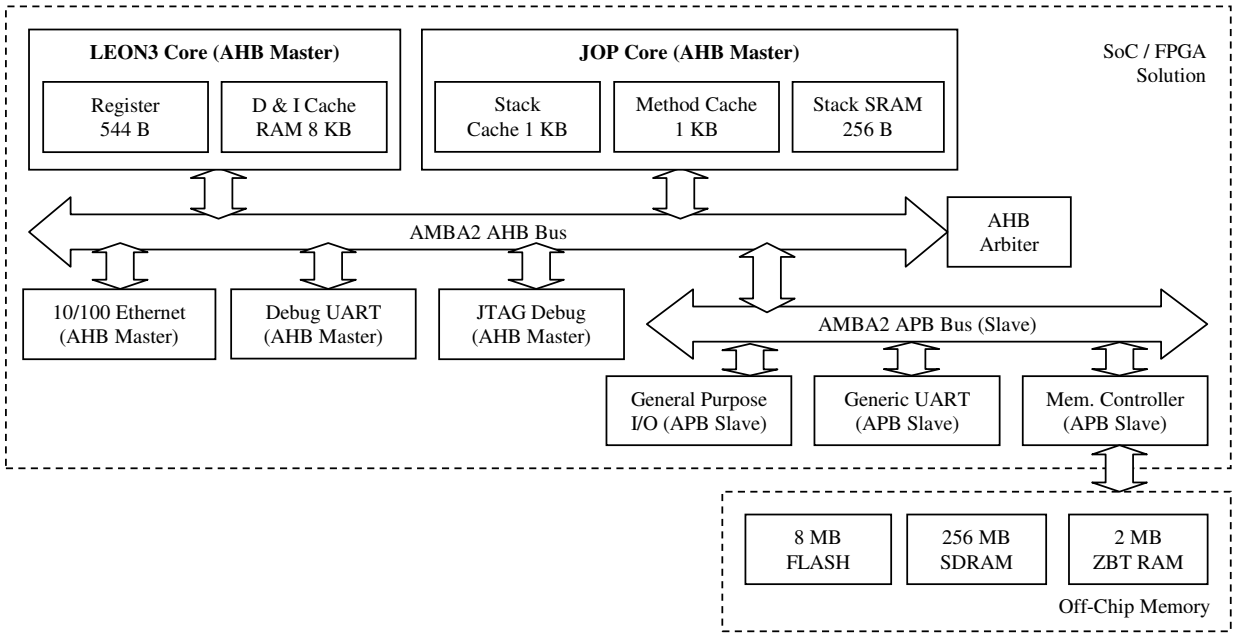

Figure 7. System-on-Chip Block Diagram

FX60 Board is targeted as it is used in the CubeSat design. Distributing the caches and memory can have two major benefits: 1) scaling the memory bandwidth and 2) a reduction in access time to local memory. This method is typically used to support larger processor counts such as with multiple LEON3 or Network-on-Chip (NoC) designs. This fully distributed shared memory approach will require higher IPC bandwidth where higher latencies will be found.

4. Dual-Core Processor Implementation

The advantage of using multiple cores on one configurable device is that a mix of technologies can be used, making the designs very versatile with differing memory systems and IP Cores but with high time and design costs. This is why the JOP core is integrated in such a way that it can be added to the Gaisler IP Core Library (or GRLIB) [28]. Integration thus far has included the JOP processor as an AHB Bus Master wrapped for interfacing purposes and connections to a reduced LEON3 with only necessary IP cores via the AHB Arbiter and AHB Bus.

Integration Configuration: The implemented SoC solution includes the following important cores implemented in the design:

- LEON3 Central Processing Unit (CPU)
- AMBA Bus: AHB (high speed) & APB (peripheral)
- Debug Support Unit (DSU) & JTAG Debug UART
- JOP Wrapper
- ESA Memory Controller

The LEON3 is used as the main controller in the FPGA. The AHB Bus provided high speed communication between internal cores. The APB Bus provides communication to external peripherals, such as memory and other I/O. The DSU and debug UART provide useful debugging information from the FPGA device. The JOP wrapper has the main JOP core, an AHB slave to communicate with the LEON3 processor and an APB slave for memory and I/O communication. The memory controller allows the synchronization between the SoC signals and external components.

Timing Results: Current synthesis results give a maximum frequency of 37.398 MHz, half the usual operating frequency of LEON3 and JOP. Multiple AHB clock signals (also called dynamic clock switching) can be used to allow the processor to reach higher performance.
The three WCET signals were also found in paths between:

1. LEON3 CPU and AMBA memory controller (11.392 ns through logic, 22.032 ns through routing)
2. LEON3 CPU and JOP AHB Master interface (5.128 ns through logic, 10.773 ns through routing)
3. JOP Cache and JOP Bytecode Address (2.255 ns through logic, 4.582 ns through routing)

This result highlights the need for optimizing for speed rather than area to decrease the heavy routing latencies in the design. A trade-off between the area and speed of the SoC design is usually required to control the communication systems. A system clock speed of 40 MHz (periodic time of 25 ns), when compared to the IEEE 802.11 MAC schemes measured in milliseconds, will be sufficient for on-board systems.

Table 2 presents the difference in on-chip logic cell (LC) utilization between a full LEON3 (with FPU & MMU), the JOP with an AHB interface and finally the combined reduced LEON3 and JOP with AHB interfaces (without FPU & MMU). By making the design application specific, we can utilize and ensure power efficiency in FPGAs.

Table 2. Summary of Integration Utilisation on GR-XC3C-1500 FPGA Board

Component	LCs Used
Full LEON3 CPU + AMBA Bus System	7546
JOP CPU + AMBA Interface	3252
Reduced LEON3 + JOP + AMBA Bus	8770

Memory Trade-off: There is a trade-off between on and off chip memory, specifically between speed and power requirements. SRAM although fast and on-chip will increase power consumption and logic blocks or cells required on the FPGA. DRAM will usually require an off-chip device and is typically 100 times slower that SRAM (not including bus arbitration latencies) but is a very dense memory technology. Power consumption of the complete dual-processor design and the current memory configuration shown in Fig. 7 is estimated in Xilinx's XPower at 2.33 W with 1.76 W used in memory interfacing. This, again, highlights the need for an efficient memory system.

Future work includes reading/ writing to the AHB bus from the JOP core, writing software for the LEON3 under RTEMS to implement network reconfigurability and testing the SoC design in the satellite cluster scenario using a digital interface with a desktop computer to emulate signals coming to and from the satellite.

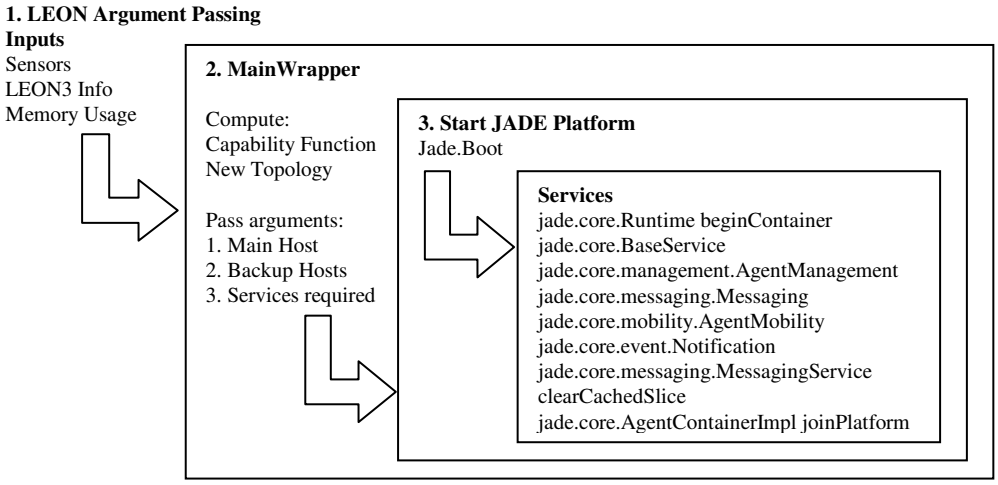

Figure 9. Reconfigurable Network Topology Algorithm Overview

5. Network Topology Reconfiguration Procedure

In this Section, a novel dual-core startup procedure is presented to deal with situations that require a hard reset, soft reset and network topology change. Fig. 9 shows an overview of the flow in which hardware and software resources are discovered so that network topology can be best reconfigured by better capable satellites. The stages are described below:

Stage 1: Startup FPGA Bus System & LEON3: Upon startup, each AMBA core sends its 'Plug & Play' signals to the AHB controller which then decodes the values and generates the correct select signals. The LEON3 is started where core identification and system memory along with starting tasks (or other existing loads) can be discovered.

Stage 2: Startup JOP & JADE-LEAP: JOP loads the main memory microcode from Flash and then the Boot() method is invoked to start running Java programs. The memory is then measured, GC is performed to clean the cache and initialize internal data structures. Class methods are then invoked and the Main() method is then invoked to start the Java application with argument passing to configure and optimize JADE-LEAP for satellite's role in the network topology and services.

Stage 3: Network Topology Refresh: To initialize, check or change the network topology, an IPC scheme is used that can soft-reset JADE-LEAP in differing configurations with differing services. An Agent can discover local data and identify services and existing interfaces or functionalities that the satellite has to offer the network. This discovery of local node level resources is defined as the 'Capability Function' (CF). It describes the relationship between total resources and current loads with a multiplication factor ω associated to the property and readings taken with regards to time t as follows:

$$CF = \frac{\omega 1 \times MEM(t) + \omega 2 \times PP(t) + \omega 3 \times BP(t)}{\omega 4 \times MR(t) + \omega 5 \times CL(t)} \quad (1)$$

where, MEM = Memory Available (RAM size), PP = Processing Power, BP = Battery Power, MR = Mobility Rating (a function of speed and acceleration), CL = Computing Load (existing on the satellite).

The "usefulness" value of a satellite node obtained from equation (1) can be used for a number of satellite management applications. These include routing applications, where only "useful" satellites are select when routing high priority data, or for when topology reconfiguration is required so that a "useful" and reliable satellite is selected as the sink to ground.

6. Conclusions

The unique problem presented in this paper is to overcome 'disconnected' and highly mobile space requirements for distributed computing in satellite networks for scenarios such as event monitoring, space vehicle inspection and even deep space exploration. To enable the latest Agent based distributed computing systems, a new dual core SoC design is proposed for real-time Java functionality on board the constrained picosatellite CubeSat platform. With strict system requirements of low memory footprint, Java functionality and hard real-time operation, preliminary results of the dual-core processor design are presented with general purpose functions using the LEON3 IP core and Java specific functions using the JOP IP core. An autonomous

Agent based network topology reconfigurability procedure is also presented which takes into account the current state of the satellite cluster at network level and any local resources or loads at node level for future distributed satellite management operations.

Acknowledgements

This work is carried out under the 'ESPACENET Project' funded by EPSRC (under grant EP/C546318/1).

References

[1] T. Vladimirova, X. Wu, K. Sidibeh, D. Barnhart and A. H. Jallad, "Enabling Technologies for Distributed Picosatellite Missions in LEO", *Proceedings of 1st NASA/ESA Conference on Adaptive Hardware and Systems* (AHS 2006), pp. 330-337, IEEE Computer Society.

[2] D. J. Barnhart, T. Vladimirova, "Very-Small-Satellite Design for Distributed Space Missions", AIAA Journal of Spacecraft and Rockets, Vol. 44, No. 6, November-December 2007.

[3] T. Vladimirova, X. Wu and C. P. Bridges, "Development of a Satellite Sensor Network for Future Space Missions", Proceedings of *IEEE Aerospace Conference 2008*, Big Sky, USA (IEEEAC'08).

[4] S. Curtis et al, "ANTS (Autonomous Nano Technology Swarm): An Artificial Intelligence Approach To Asteroid Belt Resource Exploration", Proceedings of *51st International Astronautical Congress*, 2-6 Oct 2000 in Rio de Janeiro, Brazil.

[5] JADE Website, Applications and Business, Website, http://jade.tilab.com/

[6] S. Chien et al, "The Techsat-21 Autonomous Space Science Agent", Proceedings of *International Conference on Autonomous Agents and Multiagent Systems* (AAMAS'02), July 15-19, 2002, Bologna, Italy

[7] Lloyd Wood et al, "Saratoga: a Delay-Tolerant Networking convergence layer with efficient link utilization", Proceedings of *Third International Workshop on Satellite and Space Communications*, September 2007.

[8] C. P. Bridges and T. Vladimirova, "Autonomous Software Agents in Wireless Embedded Systems", Proceedings of 3rd UK Embedded Forum (UKEF'07), p. 167, April 2007, Durham, IET

[9] F. Bellifemine, G. Caire, A. Poggi, G. Rimassa, "JADE White Paper", http://jade.tilab.com/papers/2003/WhitePaperJADEEXP.pdf

[10] Foundation for Physical Agents Specifications Website, http://www.fipa.org/

[11] N. Haridas, E. Yang, A. T. Erdogan, T. Arslan, N. Barton, A. J. Walton, J. S. Thompson, A. Stoica, T. Vladimirova, X. Wu, K. D. McDonald-Maier, W. G. J. Howells. "ESPACENET: A Joint Project for Evolvable and Reconfigurable Sensor Networks with Application toAerospace–Based Monitoring and Diagnostics",*Proceedings of 6th International Conference on Recent Advances in Soft Computing (RASC2006)*, pp. 410-415.

[12] CubeSatKit, http://www.cubesatkit.com

[13] Pumpkin Inc, http://www.pumpkininc.com

[14] Clyde Space, http://www.clydespace.com

[15] SSTL, SSTL SGR-05 series space GPS receivers, Datasheet, http://www.sstl.co.uk/documents/Subsys_SGR05_HQ%5B1%5D.pdf

[16] Microhard MHX-910, Datasheet, http://www.data-connect.com/Microhard_MHX-910.htm

[17] PF5100 Virtex-4 FPGA FX60, Datasheet, http://www.derivation.com/products/pf5100.html

[18] IEEE 802.11 PC/104 Board from Elcard, WIB250A67 Series PC/104+ 802.11ag WLAN Module, Datasheet, http://www.elcard.fi/pdfs/DSWIB250A67-02.pdf

[19] Gaisler Research – Leon 3 Processor, Datasheet, http://www.gaisler.com/doc/Leon3%20Grlib%20folder.pdf

[20] Andre-Louis Pouponnot, "High-performance LEON microprocessor prototypes released for evaluation", ESA, 30 June 2005, http://www.esa.int/techresources/ESTEC-Article-fullArticle_par-29_1120038112963.html, Last Accessed: 2/5/2008

[21] M. Schoeberl, "A Java Processor Architecture for Embedded Real-Time Systems", *Journal of Systems Architecture* (2007), Publish date unconfirmed, http://www.jopdesign.com/doc/rtarch.pdf

[22] T. Vladimirova, X. Wu, A. H. Jallad and C. P. Bridges, "Distributed Computing in Reconfigurable Picosatellite Networks", Proceedings of *NASA/ ESA Conference on Adaptive Hardware and Systems* (AHS-2007), pp. 682 – 692

[23] Java ME Website Connected Limited Device Configuration (CLDC), http://java.sun.com/products/cldc/

[24] "AMBA Specification Rev 2.0", ARM Website.

[25] Real-time Java Specification Main Website, http://www.rtsj.org/

[26] M. Schoeberl, "JOP Reference Handbook", Section 5.1.2. Method Types and Length, 2007, http://www.jopdesign.com/doc/handbook.pdf

[27] J. Gaisler and S. Habinc, "FPGA Design Using the LEON3 Fault Tolerant Processor Core", in Proceedings of 8th *Military and Aerospace Programmable Logic Devices* (MAPLD '05), 7-9 September 2005, Washington DC

[28] J.Gaisler, GRLIB IP Library User's Manual, Datasheet, http://gaisler.com/products/grlib/grip.pdf