Reconfigurable Linux for Spaceflight Applications

John Williams and Neil Bergmann
School of ITEE, The University of Queensland
Brisbane, Australia
{jwilliams;n.bergmann}@itee.uq.edu.au

Abstract

Future space missions will have significant requirements for on-board scientific computing beyond traditional computing services such as avionics and spacecraft management. Reconfigurable computing has been identified repeatedly as an enabling technology for such next-generation spaceflight computing systems. However, with this broadened deployment model comes the requirement for a fully-featured reconfigurable operating system, providing a development and deployment infrastructure. In this paper we discuss the requirements for such an operating system, and argue that embedded Linux running on FPGA-based soft or hard processor cores provides an excellent match for the role.

1. Introduction

The availability of high performance on-board computing has been identified as a key technological requirement of NASA's 21st century human spaceflight objectives. Reconfigurable computing is widely recognised as offering many compelling capabilities for this purpose, including high performance, fault-tolerance, design flexibility and reuse, and the capability for post-deployment functional retargeting, among others [1].

There are two primary challenges in applying reconfigurable computing to this domain. The first relates to the physical characteristics of programmable logic devices and the characterisation of their behaviour in space environments. We include in this characterisation, the study and practice of high-reliability logic design, device- and system-level redundancy and so on.

Assuming that the physical challenges can be met, the second challenge is to provide system architectures and development frameworks that support application development on reconfigurable platforms. This requirement is not unique to aerospace applications – significant research effort continues to be spent on making reconfigurable computing look more like software than hardware, as evidenced by research and development activities in the field of high-level synthesis. It is this development and deployment useability aspect that we consider in the current work.

Single-purpose space computing systems such as avionics, flight control and spacecraft management are typically built either on the "bare-metal" or on small footprint, real time operating systems such as vxWorks. This approach is suitable and indeed desirable because of the tight coupling between the hardware and software design. Similarly, the use of re-programmable logic devices in space missions has also tended to be mostly single-purpose – reconfigurablity is typically only exercised for post-launch update, or to work around unforeseen design issues. Thus, the reconfigurable logic design is tightly coupled with its surrounding systems and circuitry.

If the computational density advantages of reconfigurable computing [2] are to be leveraged for general-purpose onboard scientific computing, then a true reconfigurable operating system is needed. We outline the requirements of such an operating system in Section 3, and argue that embedded Linux, on FPGA-based hard or soft processor cores is the best approach to developing a general purpose reconfigurable space computing platform.

2. Background

Wigley and Kearney proposed the following axioms for reconfigurable operating systems [3]:

- the primary focus should be reducing complexity as seen by the user
- the operating system must form a contract between the systems software and the applications designer.

These axioms are a natural extension of operating system concepts to the reconfigurable computing domain, essentially advocating a logic management API. Until recently, most research on reconfigurable operating systems has assumed or implied a master/slave hardware architecture, whereby a conventional microprocessor based system (often a desktop PC) is interfaced to one or more FPGAs that provided the reconfigurable computing resources. This has led to a focus on tasks such as logic area partitioning and management (e.g. [3]) and dynamic module placement [4]. The master CPU systems execute whatever mainstream operating system is convenient, typically Linux or a Unix variant, or Windows. This split architecture can create a performance bottleneck on the system bus or interconnect, and limit the cohesion possible between the conventional and reconfigurable components of the system.

Two relatively recent developments change this model. Firstly, the advent of powerful soft processor cores, and embedded hard processor cores allows the FPGA itself to host a fully featured operating system. For example, Linux is

available on the Microblaze and NIOS soft processors, as well as the PPC and ARM hard cores present in various Xilinx and Altera FPGAs. Secondly, the introduction of self-reconfiguration capabilities in modern FPGAs, such as the Xilinx Internal Configuration Access Port (ICAP), allows self-reconfiguring systems without the need for an external host. Embedding the processor core within the logic fabric offers tighter integration and greater flexibility in the overall system architecture, and in many cases, higher performance by avoiding master-slave communication bottlenecks.

3. Requirements for a Reconfigurable Operating System

The role of the reconfigurable operating system in general purpose spaceflight computing applications is different to an RTOS upon which the avionics or navigation subsystem must execute. The system should support a wider variety of developers and applications for a wider range of scientific and technical computations. Specifically, a general purpose reconfigurable operating system must

1. **support sequential (processor-based) execution**, with a *familiar programming paradigm as a starting point for application development*. It is unreasonable to expect that all users will have hardware design capabilities, and therefore a graceful transition of application codes from conventional computing platforms must be offered;
2. **offer interoperability with existing general purpose computing infrastructure**, including networking, file storage and other I/O device interfacing;
3. **provide a process model that seamlessly supports hardware, software, and hybrid processes** within the same architecture, including *support for standard interprocess communication methodologies* across homogenous and heterogeneous application implementations;
4. **provide a logic management interface** that abstracts operations such as dynamic partial reconfiguration, in support of the hardware process model;
5. **support integration of hardware components developed in a variety of tool flows**, such as graphical tools like Simulink / System Generator, high-level synthesis tools such as Handel-C, as well as traditional VHDL and Verilog flows;
6. **be scaleable**, supporting single-chip, multi-chip and multi-board computing systems within the same operating system architecture.

Of these requirements, 1,2 and 6 are already supported by embedded Linux. One of the arguments for using Linux in embedded contexts is the ability to develop and prototype applications on standard desktop platforms. Interoperability with a broad range of existing systems is also a strong motivator. On scaleability, Linux is finding wide deployment in both multiprocessor (SMP) and multinode (cluster) configurations, and these are available to embedded Linux development.

Requirement 3 is partially supported, in that Linux has a mature interprocess communication model inherited from its Unix origins. Pipes and FIFO-based communication, shared memory, semaphores and network-based message passing are fundamental components of the existing operating system. Hardware analogues of these primitives are well established, and previous work has demonstrated how such structures can be naturally integrated into the Linux architecture [5].

Requirement 4 has frequently been addressed by previous reconfigurable operating systems research. We are looking at integrating these ideas into embedded Linux. For example, we have already demonstrated partial dynamic self-reconfiguring linux systems [6].

Requirement 5 can be met by creating bindings for the various tool flows that map onto the hardware mamangement/process/IPC architecture. Adequate support for this diversity of tools is essential for a reconfigurable operating system to gain widespread support.

4. References

[1] N. W. Bergmann and J. A. Williams, "Avionics Upgrade Management using Reconfigurable Logic," in Proc. Australian International Aerospace Congress, Brisbane, Australia, 2003.

[2] A. De Hon, "The Density Advantage of Configurable Computing," in *IEEE Computer*, 2000, pp. 41-49.

[3] G. Wigley and D. Kearney, "The Development of an Operating System for Reconfigurable Computing," in Proc. IEEE Symposium on FCCM, 2001.

[4] C. Patterson, "A Dynamic Module Server for Embedded Platform FPGAs," in Proc. ERSA, pp. 31-40, Las Vegas, USA, 2003.

[5] J. A. Williams and N. W. Bergmann, "Programmable Parallel Coprocessor Architectures for Reconfigurable System-on-Chip," in Proc., IEEE International Conference on Field Programmable Technologies (FPT04) (to appear), 2004.

[6] J. A. Williams and N. W. Bergmann, "Embedded Linux as a platform for dynamically self-reconfiguring systems-on-chip," in Proc. International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA '04), Las Vegas, Nevada, 2004.

If you are interested to know, why do we use Nginx in front of an application such as Flask, Django, Ruby on Rails, NodeJS, etc? When you have an Ubuntu or any Linux server and want to set up Flaskâ€¦Â Next, weâ€™ll set up a virtual environment in order to isolate our Flask application from the other Python files on the system. Start by installing the python3-venv package, which will install the venv module: pip3 install virtualenv. The goal of Mariner is to be used as an internal Linux distribution for Microsoftâ€™s engineering teams to build cloud infrastructure and edge products and services. Of course Mariner is open source and it has its own repo under Microsoftâ€™s GitHub organization. No ISOs or images of Mariner are provided, however the repo has instructions to build them on Ubuntu 18.04. There are a series of prerequisites listed in this GitHub page that roughly include Docker, RPM tools, ISO build tools and Golang, amongst others. The build process for an ISO is very straightforward, it relays on pre-compiled RPM packages from CBL-Mariner package repository. Since I wanted to install Mariner on my vSphere 7 homelab I choose to create the ISO. Reconfigurable computing has been identified repeatedly as an enabling technology for such next-generation spaceflight computing systems. However, with this broadened deployment model comes the requirement for a fully-featured reconfigurable operating system, providing a development and deployment infrastructure. In this paper we discuss the requirements for such an operating system, and argue that embedded Linux running on FPGA-based soft or hard processor cores provides an excellent match for the role. Discover the world's research. 20+ million members. The role of the reconfigurable operating system in general purpose spaceflight computing applications is different to an RTOS upon which the avionics or navigation subsystem must execute. The system should support a wider variety of developers and applications for a wider range of scientific and technical computations. Specifically, a general purpose reconfigurable operating system must. 1. support sequential (processor-based) execution, with a familiar programming paradigm as a starting point for application development.Â One of the arguments for using Linux in embedded contexts is the ability to develop and prototype applications on standard desktop platforms. Interoperability with a broad range of existing systems is also a strong motivator. */\1/p' /usr/include/linux/capability.h | tr A-Z a-z. // don't take cap_last_cap which is the same as the last cap_syslog capability. // const char *cap_name[CAP_LAST_CAP+1] =.Â "cap_linux_immutable", "cap_net_bind_service", "cap_net_broadcast"