Mapping Concurrent Real-Time Software Languages to FPGA

Neil C. Audsley, Iain J. Bate and Michael Ward
Real-Time Systems Research Group,
Department of Computer Science, University of York, York, UK
[neil,ijb,mward]@cs.york.ac.uk

Abstract

The development of long lifetime hard real-time systems raises is becoming increasingly difficult, due to increased system complexity and pressure to reduce development times. Currently under consideration is the re-hosting of the software of such systems, designed in a mixture of new and legacy Ada, onto FPGAs. This paper provides an overview of the achievements, status and future directions of this work.

1 Introduction

Real-time embedded systems are becoming increasingly complex, in terms of their functional and non-functional properties, so making their design and implementation evermore difficult. Also, systems need to be developed in shorter times, due to business requirements to reduce time-to-market. For hard real-time (often safety-critical) systems, as typified by aerospace and automobile applications, this combination of complexity and shorter development times is extremely demanding. A further constraint on hard real-time development is the need to show that the system is fit-for-purpose [1], meeting its functional and non-functional (ie. timing and safety) requirements prior to it actually running [1] – often failure can have catastrophic consequences, eg. a timing failure in a hard real-time system.

Pressures of increased complexity (particularly in software) and reduced development times are difficult to realise for hard real-time system (ie. aerosopace) developments. Often, these pressures are addressed by:

- increasing the automation within the development process (as seen by the current trend towards the increased use of high-level modelling tools, eg. UML, Matlab, MatrixX, which contain automatic software generation facilities software production);
- increasing the use of legacy (and commercial-off-the-shelf) code, designed for previous systems.

Such approaches enable shorter development times, but do not reduce the complexity of determining whether the implemented system will meet its real-time requirements – establishing these timing properties rises rapidly as overall system complexity increases.

This paper overviews current work that is investigating the direct compilation of Ada software from aerospace systems direct to FPGA. The main motivation for this implementation approach is the better timing properties of FPGAs over conventional processors: processor scheduling is not required on FP-GAS, which reduces the jitter in the resultant system, and limits the variability in execution time of the implemented function [6, 7].

[1]Often such systems are not permitted to become operational until some regulatory authority has been assured as to their safety or fitness for purpose. Regulatory authorities in the aerospace domain impose stringent requirements upon developers, as detailed in standards such as DO-178B (civil aircraft)[2], Defence Standards 00-54, 00-55, 00-56 (for UK military hardware, process and software respectively)[3, 4, 5].

2 Ada for Hard Real-Time Systems

The system functions are in Ada, either generated by high level tools (eg. Matlab) or as legacy code. The Ada language [8] facilitates the programming of real-time systems. It contains facilities for programming-in-the-small (ie. sequential programming), facilities for programming-in-the-large (ie. data abstraction and packages), together with facilities for concurrent programming (ie. tasks and inter-task communication). In addition, subsets of Ada have been developed that effect restrictions upon Ada that enable conforming programs to be statically analysed for timing, resource and functional properties.

The SPARK subset of Ada [9] restricts the sequential part of the language. Conformant programs can be proved (partially) correct. SPARK does not contain any dynamic constructs, including concurrency (and synchronisation), the access (pointer) type, variant records (hence no object-oriented capabilities). Subprograms are no longer allowed to recurse, nor can procedure pointers be used. These restrictions make all subprogram call trees known at compile-time, and all variable references resolve to only one instance. The SPARK Ada subset is consistent with the requirements for real-time system timing analysis in that all conforming programs are statically analysable for their worst-case properties.

The Ravenscar tasking profile[10] is a statically analysable tasking subset. Unlike full Ada, Ravenscar compliant code is predictable in its timing behaviour and resource usage. The Ravenscar profile makes no comment on the sequential part of the language. The definition of Ravenscar is effectively included in the Ada standard, being part of Annex H (Safety and Security) which comments on applicability of Ada language features for use in safety-related systems.

A SPARK / Ravenscar conformant Ada program consists of a number of concurrent tasks, that interact via protected objects. These objects enforce mutual exclusion over some procedures and associated data within the object. Interaction with other devices is achieved by representation clauses, which associate a specific memory location with a program variable, so achieving a memory mapped programming model. Also, conformant programs are analysable for timing (and other statically determinable) properties.

3 Compilation of Ada to FPGA

The compilation of Ada to FPGA utilises the syntactic and semantic phases of the GNAT Ada compiler [11], bolting on an FPGA (effectively EDIF) code generation phase via the standard ASIS Ada compiler interface. The result is an ada to FPGA compiler [6, 7, 12].

3.1 Target Architecture

The physical target architecture assumed is that of a single Field Programmable Gate Array (FPGA) [13], coupled to a number of RAM banks. Clearly, limiting the target architecture to a single FPGA restricts the size of source program that can be implemented. However, the physical size of current high-end FPGAs is large, ensuring that substantial functionality can be achieved on a single device. Also, the presence of the RAM banks ensures that (parts of) the FPGA can be used for softcore CPUs, further extending the size of the functionality that can be implemented upon the target. This is discussed further in section 4.

3.2 Hardware Ada Compilation

SPARK / Ravenscar conformant Ada programs are ideal for direct compilation to hardware circuit. In[6, 7, 12] an Ada compilation process is described for such programs. Essentially, concurrency within Ada can be represented on hardware as truely parallel tasks. In terms of the Ravenscar tasking subset, the main implication is that task scheduling is no longer required – indeed, no run-time is required at all. The sequential language used within a task is relatively straightforward to compile to hardware, as the restrictions of the SPARK subset ensure that no dynamic statements are present in a task. Note that the concurrent features of Ada require a run-time (or kernel) to be present at run-time. One function of the run-time is to provide scheduling between the different application tasks. Given the restricted concurrency model of Ravenscar conformant programs, the run-time required for such programs is simple – indeed, a simplistic run-time was one of the prime motivations for the Ravenscar subset.

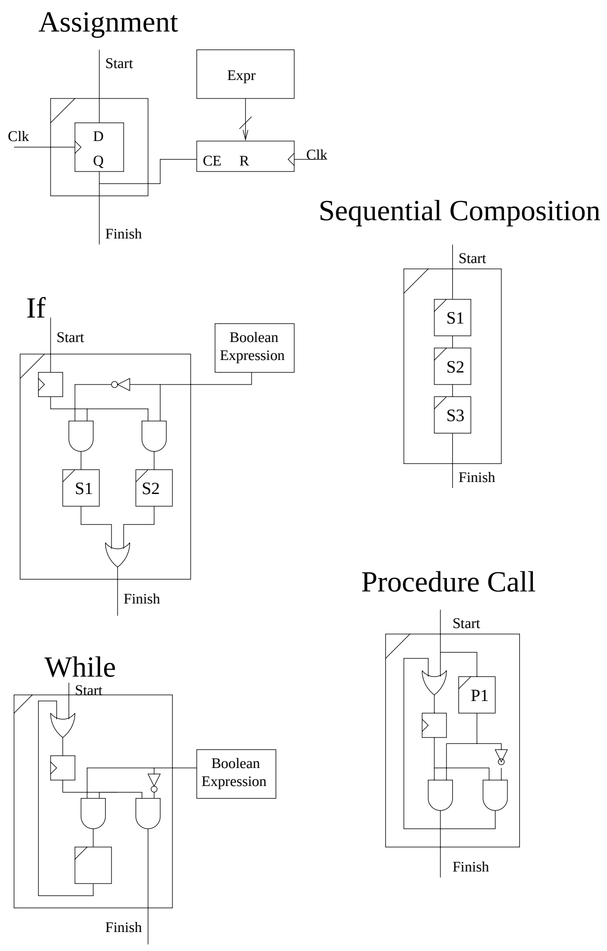

Figure 1: Circuit Templates.

Protected objects enforce mutual exclusion over some procedures and associated data. Hardware compilation does not remove the need for mutual exclusion, so protected objects remain. When contention exists over access to a protected object, the default locking policy of Ada is used, that is ceiling protocol [14], where ceiling priorities are defined in terms of the priorities of the tasks that access the object.

Examples of the hardware circuit templates that are used within the compiler are given in Figure 1.

4 Future Work

Currently, extensive case-studies are being undertaken in co-operation with our industrial partners. This work is highlighting some deficiencies in the breadth of the compiler (eg. not all binary operations are supported).

Further work is based around a number of related themes:

Integration with VHDL: Work is proceeding to allow close integration between Ada and VHDL components. This uses the VHDL signal concept to communicate with Ada components (along with shared memory). The communication is two-way, and does not restrict the Ada component to be master, as suggested by Ada standards.

Use of Softcores: The current approach follows an idealistic path, where all the Ada source is compiled to digital logic (ie. FPGA). This approach is not scalable as the size of FPGA is finite. Therefore, current work is investigating the use of softcores to implement parts of the Ada source that do not map efficiently to FPGA, or for when FPGA resources are exhausted. This raises many source language and compilation issues, including the granularity of Ada construct that is compiled to softcore or hardware, together with the communication between the differently compiled components.

Hardware-Software Codesign: The general approach of increased automation and higher-level development is seen in much hardware-software codesign research. Codesign enables automatic derivation of a hardware architecture and application software from a high-level specification [15, 16, 17]. Such approaches are limited in terms of the scale of the system that can be developed (usually small uniprocessor or multiprocessor based systems rather than large distributed systems); have limited traceability from specification to final design (due to automation); have limited ability to change / update parts of the system with ease at some later date (rather the entire modified system has to be re-generated with no guarantee that the new hardware architecture will be identical to the original); are largely aimed at short lifetime products, eg. consumer electronics; assume a complete specification is initially available.

Current work is investigating codesign principles to allow automatic trade-off of design choices, particularly in the non-functional domain, eg. time and the assignment of Ada to either softcore or circuit.

References

[1] Y. C. Yeh, "Dependability of the 777 primary flight control system," *Proceedings 5th IFIP Working Conference on Dependable Computing for Critical Applications*, 1995.

[2] European Organisation for Civil Aviation Electronics, *DO-178B: Software Considerations in Airborne Systems and Equipment Certification*, December 1992.

[3] *00-54 - Ministry of Defence, Interim Defence Standard 00-54 Requirements of Safety Related Electronic Hardware in Defence Equipment*, 1999.

[4] Ministry of Defence, *Defence Standard 00-55: Requirements for Safety-Related Software in Defence Equipment (Part 1: Requirements, Part 2: Guidance)*, August 1997.

[5] Ministry of Defence, *Defence Standard 00-56: Requirements for Safety Management Requirements for Defence Systems (Part 1: Requirements, Part 2: Guidance)*, December 1996.

[6] M. Ward and N. C. Audsley, "Hardware Compilation of Sequential Ada," in *Proceedings of CASES 2001*, pp. 99–107, 2001.

[7] M. Ward and N. C. Audsley, "Language Issues of Compiling Ada to Hardware," in *Proceedings of Ada Europe 2002*, 2002.

[8] S. Taft and R. Duff, eds., *Ada 95 Reference Manual: Language and Standard Libraries, International Standard ISO/IEC 8652:1996(E)*, vol. Lecture Notes in Computer Science 1246. Springer-Verlag, 1997.

[9] J. Barnes, *High Integrity Ada: The SPARK Approach*. Addison-Wesley, 1997.

[10] A. Burns, B. Dobbing, and G. Romanski, "The Ravenscar Tasking Profile for High Integrity Real-Time Programs," in *Reliable Software Technologies, Proceedings of the Ada Europe Conference, Uppsala*, vol. 1411, pp. 263–275, LNCS, Springer-Verlag, 1998.

[11] Ada Core Technologies, *GNAT Ada Compiler : http://www.gnat.com*, 2001.

[12] M. Ward and N. C. Audsley, "Hardware implementation of the ravenscar ada tasking profile," in *Proceeedings of CASES 2002*, pp. 59–68, 2002.

[13] Xilinx Corporation, *Xilinx Product Information : http://www.xilinx.com/products*, 2003.

[14] L. Sha, R. Rajkumar, and J. Lehoczky, "Priority Inheritance Protocols: An approach to real-time synchronisation," *IEEE Transactions on Computers*, vol. 39, no. 9, pp. 1175–1185, 1986.

[15] K. Suzuki and A. Sangiovanni-Vincentelli, "Efficeint Software Performance Esimation Methods for Hardware / Software Codesign," in *Proc. Design Automation Conference*, 1996.

[16] S. N. D. Gajski, F. Vahid and J. Chong, "System-Level Exploration with SpecSyn," in *Proc. Design Automation Conference*, pp. 812–817, 1998.

[17] J. Henkel and R. Ernst, "A Hardware/Software Paritioner Using a Dynamically Determined Granularity," in *Proc. Design Automation Conference*, pp. 691–696, 1997.