An efficient and deterministic multi-tasking run-time environment for Ada and the Ravenscar profile on the Atmel AVR®32 UC3 microcontroller

Kristoffer Nyborg Gregertsen
Dept. of Engineering Cybernetics
NTNU, Norway
gregerts@itk.ntnu.no

Amund Skavhaug
Dept. of Engineering Cybernetics
NTNU, Norway
amund@itk.ntnu.no

Abstract

This paper describes how an efficient and deterministic multitasking run-time environment supporting the Ravenscar tasking model of Ada 2005 was implemented on the Atmel AVR32 UC3A microcontroller. The open source GNU Ada Compiler (GNAT GPL 2007) was also ported to AVR32 as a part of this work, making a working Ada development environment available on the architecture for the first time.

1. Introduction

Dependability is essential in many embedded and real-time systems. Failures can often result in financial losses, environmental damage and even the loss of lives. Ada is a programming language designed for high-integrity systems and has many safeguards against common programming errors. The language is an ISO standard [1], making programs portable within different compilers, run-time libraries and operating systems.

While Ada is much used within high-integrity systems, the concurrent constructs of the language have often been excluded as being non-deterministic and inefficient [2]. Instead such methods as the cyclic executive [3], [4] has been used. Advances in static analysis have made it possible to check hard deadlines when using preemptive fixed priority scheduling. This has led to development of the Ravenscar profile [5], a subset of the Ada tasking model designed to provide the static and deterministic environment needed to perform static analysis [2]. The simplicity of the tasking model also allows efficient run-time environments.

The Polytechnical University of Madrid developed the Open Ravenscar Kernel (ORK) [6] on contract to the European Space Agency for the ERC32 architecture. The Open Ravenscar Kernel was further developed and integrated into the GNU Ada Run-Time Library (GNARL) by José F. Ruiz at AdaCore [7].

The Atmel AVR32 [8] is a brand new architecture designed by Atmel Norway in cooperation with the Norwegian University of Science and Technology (NTNU), and is optimized for code density and high computational throughput with low power consumption [9]. By porting the GNU Ada Compiler (GNAT) to AVR32 and the bare-board Ravenscar run-time environment to the UC3A microcontroller [10] Ada is made available on this architecture for the first time.

It is shown how the simplicity and power of the AVR32 architecture and the UC3A microcontroller combined with the restricted Ravenscar tasking model allows the multi-tasking run-time environment to be deterministic and efficient, making it well suited for high-integrity embedded applications.

2. The Ravenscar profile

The Ravenscar profile is specified as a set of configuration pragmas [1], [5] defining restrictions to the Ada tasking model and the required dynamic semantics. The following features are supported [2]:

- Tasks types and objects defined at library level.
- Protected types and objects, defined at library level, limited to one entry having a simple guard and a queue length of one.
- Ceiling Locking policy with FIFO dispatching policy within priorities.
- The Ada.Real_Time package for high-precision timing and the **delay until** statement.
- Synchronous task control, including suspension objects for simple synchronization.
- Protected procedures as statically bounded interrupt handlers.

The sequential parts of Ada are not affected by the profile [2].

978-3-9810801-5-5/DATE09 © 2009 EDAA

3. GNARL

The Ravenscar version of the GNU Ada Run-Time Library (GNARL) is designed to take advantage of the simplifications allowed by the profile [7]. Task management is simplified since all tasks are at library level, cannot terminate and have fixed priority. All task data structures are statically allocated, thus memory requirements are determined at link time. Protected objects are simplified since there are no asynchronous operations, no time-out on entry calls and no varying queue length on entries. Evaluation on protected entries may be done by proxy, thereby improving performance by reducing the number of context switches [7].

The GNU Low-Level Library (GNULL) is a translation layer between generic and actual operating system calls on most systems, but in this case it fully implements the needed dynamic semantics of the Ravenscar profile by including a multitasking core based on the Open Ravenscar Kernel [6], [7].

The core implements preemptive fixed priority scheduling with ceiling locking, having 256 priorities including the interrupt priorities. The number of priorities can easily be changed if needed. Each interrupt priority has its own interrupt stack allowing interrupt nesting while avoiding priority inversion [7]. Interrupts are masked as long as there is a task with higher or equal priority to that interrupt and all interrupts are masked while modifying core data.

The timing services of the core provides as high precision as possible while supporting the needed 50 year time span. This is done by using a 64-bit value for time divided into two parts. The least significant part is present in the hardware timer, while the most significant part is stored in memory and is incremented every time the hardware timer overflows.

4. The AVR32 architecture

The Atmel AVR32 architecture [8] is a 32-bit RISC architecture designed for high computational throughput with low power consumption [9]. The architecture defines instruction lengths of both 16 and 32-bits for high code density and there is a rich set of load / store instructions for high efficiency, supporting byte, half-word, word and double word memory access. The register file of the AVR32 architecture is fairly small having only 13 general purpose registers (R0 to R12), the link register (LR) used for storing routine return addresses, the program counter (PC) and the system register (SR).

The UC3 core [10] is the second implementation of the AVR32 architecture and is primarily intended for embedded control applications where deterministic execution times is important. The UC3 has an internal SRAM integrated with the CPU pipeline in order to bypass the system bus. This allows deterministic, single-cycle read/write memory access. The UC3 fully implements the DSP instructions of the AVR32 ISA such as single-cycle multiply and accumulate instructions for both modular and saturated arithmetic. Atmel claims it to deliver 1.3 Dhrystone MIPS / MHz.

5. Porting to the AVR32 architecture

5.1. Hardware setup

The EVK1100 evaluation board with the UC3A0512 microcontroller [10] was used for developing and testing the run-time environment. The UC3A0512 has 64 KB of internal SRAM, 512 KB of internal flash and is clocked by a 12 MHz external oscillator. The Atmel JTAG ICE Mk II was used for programming and debugging the device.

5.2. Porting the GNAT front-end

The GNU Ada compiler (GNAT) is an Ada front-end for the GNU Compiler Collection (GCC) developed at the University of New York and is now maintained by AdaCore. The GCC back-end for AVR32 was developed at the Norwegian University of Science and Technology and is now maintained by Atmel Norway. Both the front-end and back-end are open source software licensed under the GNU Public License (GPL).

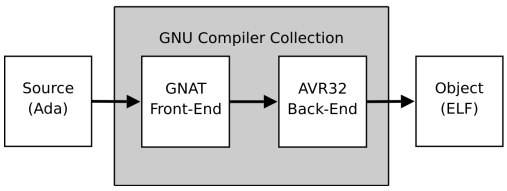

Figure 1. The GNU Compiler Collection.

Since both the front-end and back-end are open source components of GCC, porting GNAT to the AVR32 architecture was much matter of applying the GNAT GPL 2007 patches from AdaCore to the already patched GCC version 4.1.2 source code supplied by Atmel. There were however some incompatibilities caused by register promotion of return values from functions. A quick fix disabling register promotion was provided by Atmel, the problem should however be investigated further.

5.3. Porting the Ravenscar run-time

Only the code that needed to be changed due to differences between the ERC32 and the AVR32 was altered when porting the run-time environment.

5.3.1. Context switch. The context switch code consists of only 15 instructions with no branches:

```
/* Get address of running thread */
lda.w    r8, running_thread
ld.w     r9, r8[0]

/* Save context of running thread */
sub      r9, -48
stm      --r9, r0-r7, sp, lr
mfsr     r0, SYSREG_SR
st.w     --r9, r0
st.w     --r9, r12

/* Get address of first thread */
lda.w    r1, first_thread
ld.w     r9, r1[0]

/* First thread is now running thread */
st.w     r8[0], r9

/* Load context of first thread */
ld.w     r12, r9++
ld.w     r0, r9++
mtsr     SYSREG_SR, r0
sub      pc, -2
ldm      r9++, r0-r7, sp, pc
```

The addresses of the running and first thread are stored in memory instead of being passed as arguments for debugging purposes [7].

5.3.2. Interrupt handling. The AVR32 has a peripheral interrupt controller which groups different interrupt lines. Each interrupt group is assigned to one of the 4 interrupt levels by the software driver.

There is a low-level handler for each interrupt level. The interrupt ID is found by reading the interrupt cause register of the level to find the interrupt group, and then the interrupt request register of that group to find the interrupt line. The highest numbered asserted line is chosen if there are more than one.

The handler loads the interrupt stack for the given level and calls the interrupt wrapper with the interrupt ID. Prior to returning from the handler the task stack pointer is restored and a context switch is done if needed.

5.3.3. Peripheral drivers. The power manager unit is used to enable the external oscillator upon initialization of the system and setup the CPU and peripheral clocks relative to it. For simplicity, it was chosen to run both the CPU and the peripherals at the same clock rate as the external clock.

The interrupt controller provides the functionality to activate interrupts and read the interrupt ID. The package provides the mapping between interrupt identities and groups, and is specific to a given MCU series. The interrupt priorities are also defined in this package.

Two 16-bit counters are used by the timing services. One counter is used as the least significant part of the system clock, counting the whole 16-bit range and generating an interrupt on overflow. The other counter is used in one-shot mode for setting off alarms between the regular interrupts, allowing fine grained task release.

6. Metrics

6.1. Code size

The context switch for the AVR32 consists of only 15 assembler code lines. The number of assembler code lines for interrupt handling is only 18 compared to more than 100 for the ERC32. In total the number of assembler code lines is reduced from about 400 with the ERC32 to about 50 with the AVR32. The AVR32 implementation needs more peripheral drivers resulting in more Ada code lines as seen from Table 1. Most of the added lines are however register declarations for the peripherals and not executable statements.

Table 1. Comparison of Ada code metrics.

Metric	ERC32	AVR32
Statements	261	378
Declarations	528	896
Total	789	1274

6.2. Memory requirements

The memory requirements of the multitasking core are generally low, only the interrupt handling package uses a noticeable amount of SRAM memory due to the interrupt stacks in its BSS section. The size of the text section used by the run-time core is about 5.5 KB which is just above 1% of the total Flash memory available on the UC30512.

6.3. Performance

A simple test of the time needed to switch context was performed by having one task assert an external pin, unblock a second task and then go to sleep, when the second task started executing it negated the same pin. The time the external pin was asserted was measured to be about 15 μs when the system was running on 12 MHz, this equals approximately 180 clock cycles.

7. Discussion

7.1. Choice of hardware

The AVR32 is a brand new architecture with an instruction set created from scratch, making it interesting for research purposes. The architecture was created by Atmel Norway in Trondheim in collaboration with the NTNU located in the same city. The relationship between Atmel Norway and NTNU makes it possible to later design and test new hardware solutions supporting the run-time environment together with the code for the AVR32 core.

The UC3 core was chosen over the more powerful AP7 core out of several reasons. System implementation was easier on the UC3 since it requires less software drivers. However the deterministic one-cycle access time to internal SRAM was the primary reason for choosing the UC3. The AP7 has a higher average performance, but relies on external cached SDRAM resulting in a high worst-case time for memory access.

7.2. Multitasking core

The context switch is highly efficient and has a constant execution time, avoiding the problem of having a worst-case execution time that is significantly longer than the average. This should make it easier to preform accurate static analysis of applications.

The timer/counter module of the UC3 is only 16-bit causing frequent overflows. Division of the clock signal used by the timer reduces the frequency of timing interrupts but also the resolution of the system clock. Which is preferred depends on the application.

The programmable interrupt handling model of the AVR32 is very flexible. Different applications may assign different priorities to interrupt groups instead of having these decided by hardware. Only a modification of constants in the interrupt specification file and a recompilation is needed to change the priorities.

7.3. Memory requirements

The memory requirements are dominated by the task and interrupt stacks. The number of interrupt stacks is reduced from 15 on the ERC32 implementation to only 4 on the AVR32, reducing the amount of memory needed for these stacks with more than 73%. The size of the stacks may easily be altered. The secondary stack is used for returning objects of variable size from routines, and may be removed altogether for some systems further reducing the memory requirements.

8. Conclusion

The simplicity and power of the AVR32 architecture allows several enhancements compared to the original ERC32 implementation with regards to determinism, analyzability and efficiency. In particular the context switch and interrupt handling code are simplified. The context switch execution-time is deterministic easing schedulability analysis.

By porting the GNAT to AVR32 Ada is made available on this architecture for the first time. This could open new applications for AVR32 within high-integrity embedded real-time systems, and also open new markets for Ada 2005 and GNAT. The system may also be suitable for educational purposes.

Acknowledgment

Many thanks to José F. Ruiz and Arnauld Charlet at AdaCore for guidance on porting GNAT. Thanks to Atmel Norway for providing hardware and tools, and Ronny Pedersen for supporting the GCC back-end.

References

[1] ISO/IEC, *Ada Reference Manual - ISO/IEC 8652:1995(E) with Technical Corrigendum 1 and Amendment 1.*

[2] A. Burns, B. Dobbing, and T. Vardanega, "Guide for the use of the Ada Ravenscar profile in high integrity systems," *Ada Lett.*, vol. XXIV, no. 2, pp. 1–74, 2004.

[3] A. Burns and A. Wellings, *Real-Time Systems and Programming Languages*, 3rd ed. Pearson, 2001.

[4] T. Baker and A. Shaw, "The cyclic executive model and Ada," in *Proc. Real-Time Systems Symposium*, 6–8 Dec. 1988, pp. 120–129.

[5] A. Burns, "The Ravenscar profile," *Ada Lett.*, vol. XIX, no. 4, pp. 49–52, 1999.

[6] J. A. de la Puente, J. Zamorano, J. Ruiz, R. Fernández, and R. García, "The design and implementation of the Open Ravenscar Kernel," in *IRTAW '00: Proceedings of the 10th international workshop on Real-time Ada*. New York, NY, USA: ACM, 2001, pp. 85–90.

[7] J. F. Ruiz, "GNAT pro for on-board misson-critical space applications," *Ada-Europe*, 2005.

[8] Atmel Corporation, *AVR32 - Architecture Document*.

[9] J. Uthus and Ø. Strøm, "MCU architectures for computer-intensive embedded applications," Atmel Corporation, Tech. Rep., 2005.

[10] Atmel Corporation, *AT32UC3A Series - Preliminary Datasheet*.