pGNAT: THE RAVENSCAR CROSS COMPILER FOR THE GURKH PROJECT

by

PEE SEEUMPORNROJ

S.B., Electrical Engineering and Computer Science
Massachusetts Institute of Technology, 2003

SUBMITTED TO THE DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING AND COMPUTER SCIENCE
AT THE
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
MAY 2004
[June 2004]

Signature of Author:_____
Department of Electrical Engineering and Computer Science
May 20, 2004

Certified by:_____
I. Kristina Lundqvist
Charles S. Draper Assistant Professor of Aeronautics and Astronautics
Thesis Supervisor

Accepted by:_____
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

pGNAT: The Ravenscar Cross Compiler for the Gurkh Project

by

Pee Seeumpornroj

Submitted to the Department of Electrical Engineering and Computer Science
May 20, 2004
in Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science

ABSTRACT

Concurrency has greatly simplified the design of embedded software, but the gain in design simplicity is offset by the complexity of system implementation. The Ravenscar profile of Ada95 defines safe tasking constructs that enable the use of deterministic concurrency. The translation of these high-level constructs by the compiler to deterministic object code is dependent on both the underlying operating system and the system operation platform.

The commonly available open-source development tools for compiling Ravenscar compliant Ada95 assume that the operating system is implemented as software. A hardware implemented run-time kernel requires a radical rethink of the execution architecture because operating system calls have to be routed from the host processor running the tasks to the hardware implemented kernel RavenHaRT.

The redesigned compiler pGNAT is based on the open-source GNAT compiler and uses the GCC back end to cross compile application code to PowerPC object code. The GNAT run-time library (GNARL) is modified to support the use of RavenHaRT. This thesis presents the technical challenges faced and the modifications carried out for generating RavenHaRT compatible, Ravenscar compliant object code.

Thesis Supervisor: I. Kristina Lundqvist
Title: Charles S. Draper Assistant Professor of Aeronautics and Astronautics

Acknowledgements

First of all, I would like to thank my advisor Professor Kristina Lundqvist for all her guidance and support. She always had time for a discussion as well as insightful recommendations. She was very understanding and always encouraged me, even while I stumbled on building the cross-compiler for two months. I also would like to thank Jayakanth Srinivasan who always brought an ample amount of energy and constructive discussions to the table. I would not have thought of many problems otherwise.

I would like to thank the ESL team: Carl Nehme, Anna Silbovitz, Sébastian Gorelov, Kathryn Fischer, Wayland Ni, and Gaston Fiore for their amiability and support. Carl and Seb, thanks for the movies, meals, and company late at night. I also would like to thank our team in Sweden: Professor Lars Asplund and Johan Furunäs. I would not have come to this point without their help and guidance.

The GNAT team in New York and around the world was also very supportive. The e-mail correspondence was informative and enabled me to find solutions to many problems. In addition, the members of the CrossGCC mailing list also assisted me while I was building the cross-compiler version of GNAT.

Finally, I would like to thank the members of my family, friends, and Jessica Yeh for always being there when I needed them the most. Finally, to the Thai government, I am thankful for the opportunity to further my study here at MIT.

Table of Contents

Chapter 1 Introduction 6
Chapter 2 Background 8
2.1 The Ada95 Language 8
2.1.1 The Ada95 Language and Real-Time Safety-Critical Systems 9
2.1.2 The Ravenscar Profile 10
2.2 The GNU NYU Ada Translator 13
2.2.1 The GNAT Compiler 14
2.2.2 GNAT Run-Time System and GNARL 15
2.3 The Gurkh Project 17
2.4 The RavenHaRT Kernel 19
2.5 Related Work 24
Chapter 3 pGNAT Design 27
3.1 pGNAT Overview 27
3.2 GNARL Dependency Analysis 28
3.2.1 Task Dependence 29
3.2.2 Protected Object Dependence 32
3.2.3 Delay_Until Dependence 34
Chapter 4 pGNAT Implementation 35
4.1 Task Construct 35
4.2 Protected Object Construct 38
4.2.1 Protected Entries 39
4.2.2 Protected Procedures/Functions 44
4.3 Delay_Until Statement 48
4.4 Ada Time Package 50
4.5 pGNAT Verification 50
Chapter 5 Conclusions 51
5.1 Limitations 51
5.2 Future Work 51
5.2.1 Interrupt Handling 52
5.2.2 Exception Handling 52
5.2.3 Linking 52
5.2.4 Optimizations 53
5.2.5 Validation 53
Chapter 6 References 54

Appendix A System.FPGA_Kernel_Interface 57
Appendix B Manual Examination of pGNAT 85

Chapter 1
Introduction

In recent years, the development of Field-Programmable Gate Arrays (FPGAs) has enabled a rapid prototyping of embedded systems. An embedded system consists of both hardware and software components, and has a dedicated function within a larger system. Embedded systems are widely used in a wide variety of applications, including those which are mission-critical or safety-critical. Both mission-critical and safety-critical systems typically have hard real-time constraints i.e., a missed deadline is likely to result in a system failure, causing a loss of equipment, properties, and even human lives. To prevent theses failures, major research effort has been put into making the systems deterministic and analyzable in order to uncover design and implementation flaws of the systems.

The use of concurrency has greatly simplified the design of mission-critical and safety-critical embedded software, but the gain in design simplicity is offset by the complexity of implementation. Traditional approaches, such as cyclic executives [BW97], are used to manage the complexity of concurrent embedded system implementation by simplifying the analysis necessary to prove the correctness of the system. Cyclic executives are simple to implement for moderate sized problems, but they cannot handle sporadic tasks and are inflexible in the face of code and schedule modifications [AL03]. The Gurkh project aims to provide true preemptive concurrency at the application level for high integrity systems, by exploiting programming language features, hardware/software codesign and formal verification [AL03].

The Gurkh system uses a specialized hardware implemented run-time kernel to provide core operating system services, as a result, a compiler needs to be built to support this unique operational platform. This requires a radical re-design of a traditional compiler and its run-time system. This thesis addresses the issues of re-designing an existing compiler, GNAT, for the Gurkh project as well as its implementation.

This thesis is organized as follows. Chapter 2 describes the Ada95 programming language and a deterministic subset of the Ada95 language called the Ravenscar Profile. In addition, the Gurkh project and its motivations are discussed along with the purpose and implementation of its hardware

run-time kernel, RavenHaRT. The chapter concludes with related work. Chapter 3 describes the design and implementation of pGNAT, a modified GNAT compiler. It also addresses the changes to the GNAT architecture needed for the compiler to work with the Gurkh operational platform. Chapter 4 provides the detailed implementation and verification of pGNAT. Chapter 5 contains limitations and a discussion of future work that could be carried out to complete the pGNAT compiler. The appendices contain supplementary information for the thesis. Appendix A contains the source code of the os_interface package, System.FPGA_Kernel_Interface. Appendix B exhibits the resultant object code produced by pGNAT in a readable format. This information is used for the manual inspection of pGNAT.

Chapter 2
Background

An embedded system contains both hardware and software components. The key environment factors influencing the capability of an embedded system are the programming language, operating system, operational platform and the development environment. The Gurkh project aims to provide a set of tools for formal verification and validation of safety-critical software systems. The Ravenscar subset [BDR98] of the Ada95 programming language [ARM95] is the programming language of choice within the Gurkh project. The Gurkh operational platform consists of a PowerPC processor, a Monitoring Chip (MC) and the RavenHaRT run-time kernel, both of which are synthesized on the Xilinx Virtex-II Pro board [XIL02]. This chapter will discuss each of these key environment factors in detail.

2.1 The Ada95 Language

The history of Ada can be dated back to 1974. The United States Department of Defense needed a new programming language which satisfied the requirements of competitive and parallel development [Bar96]. As a result, Ada83 was created, and its standard can be found in [ARM83]. As technology progressed, the United States Department of Defense established the Ada9X project in 1988 in order to add object-oriented programming capabilities, program libraries, interfacing, and tasking to Ada83, while maintaining its reliability and strong typing [Bar96]. This tasking and real-time annex addition makes Ada95 a suitable language for safety-critical systems. Ada 95 has been adopted by the industry, for example over 99 percent of the aviation software in the Boeing 777 is written in Ada95 [Ada]. All references to Ada in this thesis refer to Ada95 unless specifically stated otherwise.

2.1.1 The Ada95 Language and Real-Time Safety-Critical Systems

Real-time safety-critical systems often utilize embedded software. These applications are characterized by strict timing constraints, which can be categorized into deadlines with many levels of criticality. This notion of concurrency has greatly simplified the design of embedded software, but the gain in design simplicity leads to complexity in the implementation. Ada95 has tasking constructs as an intrinsic part of the language instead of relying directly on non real-time operating system services. The availability of tasking constructs not only make the implementation easy, but also provide greater reliability because each operating system provides different services for the control and timing needed by applications [Bar96]. With the real-time annex, a high-fidelity real-time clock can also be used in the systems. In addition to real-time constructs and the high-fidelity clock, Ada95 also provides many safety-enhancing features [Wic98]:

1. Strong Typing: Strong typing prevents many classes of errors. All values in Ada are associated with types that are appropriate to their domains. Each type definition covers precisely the set of applicable values. The derivation of types requires an explicit conversion, and the mechanisms for the derivation are complete and consistent, thus reducing errors in applications.
2. Exception Handling: Pre-defined exception handling in Ada allows programmers to detect erroneous conditions and to specify required behaviors under such conditions at run-time. Even though the use of pre-defined exception handling makes the verification more difficult, it allows residual errors to be detected and handled. This is very critical for safety-critical and real-time applications because it increases the availability of the applications [Wic98].
3. Package: Packages provide Ada with modularity. They allow the partitioning of a program into different parts which interact through well-defined interfaces. This quality facilitates the analysis of a program by decomposing a complex system into a finite number of components and restricting interactions between the components to their interfaces. Packages also provide encapsulation, which allows developers to design and build subsystems with the proper amount of visibility and security for the systems. This helps decrease the chance of failure of the system.

4. Early Error Discovery: The Ada language is designed such that more errors can be discovered at compile time. For example, Ada subprogram specifications allow the mode and subtype of each parameter to be specified, allowing compile-time type checks on parameters. Together with strong typing, most forms of incorrect invocation of a subprogram can be eliminated at the compile time. Moreover, all the arrays are checked if their ranges are correct during the compile time, resulting in an early bug discovery.

The above attributes make Ada a desirable language for use in safety-critical systems. Most importantly, Ada has been used successfully in many real-world projects, for example the Boeing 777 [Ada], the New York City Subway system [Ada], and space systems designed by NASA and the European Space Agency [Ada]. In conclusion, Ada is an ideal choice for safety-critical systems, and thus for the Gurkh project.

2.1.2 The Ravenscar Profile

The Ravenscar Profile [BDV03] is a subset of the Ada tasking model. The subset is designed to meet two goals:

- Satisfy the requirements for determinism, schedulability analysis, and memory-boundedness.
- Define a subset that maps to a small and efficient run-time system that supports task communication and synchronization.

The motivation and the specification of the profile are described below.

2.1.2.1 The Motivation of the Ravenscar Profile

Despite the advantages of Ada, hard real-time systems require that the applications used must be proved to be predictable. However, with the use of Ada run-time features, such as tasking rendezvous, select statements, and abort statement, the analysis of an Ada application is not currently feasible [BDV03]. Moreover, the non-determinism and potentially blocking behavior of

tasking or run-time calls makes it impossible to derive an upper bound on execution time necessary for schedulability analysis.

To allow for tasking and concurrency to be used in safety-critical system, the Ravenscar Profile was defined at the 8th International Real-Time Ada Workshop (IRTAW) in 1997. The complete definition of the Profile can be found in [BDV03] and [BDR98]. The important characteristics of the Ravenscar Profile are summarized in the next section.

2.1.2.2 The Description of the Ravenscar Profile Specification

The Ravenscar Profile requires a static task set in the system. Consequently, there are a fixed number of tasks that are created at the library level. Tasks cannot terminate, so they are structured as an infinite loop. The resultant task set in the system is always static. Task invocations can either be time-triggered (tasks executing in response to a time event, such as delays) or event-triggered (executing in response to an event external to tasks). Tasks communicate only through shared data called protected objects. Preemptive scheduling is chosen for the Ravenscar Profile; however, non-preemptive scheduling can be used as well. These restrictions allow the system to be analyzed for both functional and timing behavior [BDR98].

Protected objects are used for data sharing between tasks and task synchronization. Each protected object consists of protected entries, protected procedures, and protected functions. Protected entries are used for event-triggered task invocation. A task can be suspended when calling a protected entry of a protected object, depending on the Boolean value of the entry barrier. If the barrier value is false, the task is put on the entry queue of the corresponding protected entry. In Ravenscar, only one protected entry is allowed per protected object. In addition, the maximum length of the entry queue is limited to one, which means only one task is allowed to queue on an entry at a time. These two restrictions ensure determinism by avoiding the possibility of queues of tasks forming on an entry, which makes the length of the waiting time in the queue unpredictable. Furthermore, it avoids two or more entry barriers becoming open (value equals true) as a result of a protected action, which causes an non-determinism of which entry should be served first. Protected procedures have read and write access to the barrier value. When tasks call a protected procedure, the entry barrier is evaluated before exiting. If the barrier value is true, the remainder of the entry code is executed in

the context of the calling task, and the task suspended on the entry queue is released. Protected functions are similar to protected procedures; however, they have only a read access to the entry barrier.

All tasks and protected objects have priorities, and the scheduling of the tasks in the hardware run-time kernel is determined using a priority-based preemptive scheduling algorithm. For tasks with the same priority, *FIFO within priority* policy is used [Sil04]. Protected objects also have *ceiling locking* policy. This policy enforces that any task able to call a protected object must not have its priority higher than the protected object priority. When a task calls a protected object, the task priority will be temporarily raised to that of the protected object. This ensures that only one task can access a protected object at any point in time, and other tasks cannot interrupt the currently running task on the same protected object. This essentially eliminates the deadlock as well as the unbounded priority inversion problems [BDV03]. The unbounded priority inversion problem is defined as a lower priority task overtaking the computing time from a higher-priority task indefinitely. For example, in Figure 1, the low-priority task successfully acquires the lock L and is preempted by the high-priority task. The high-priority task cannot proceed because it cannot acquire the lock L, thus suspending itself. The medium-priority task becomes runnable and thus preempts the low-priority task. Now, the medium-priority task can run indefinitely, causing the unbounded priority inversion.

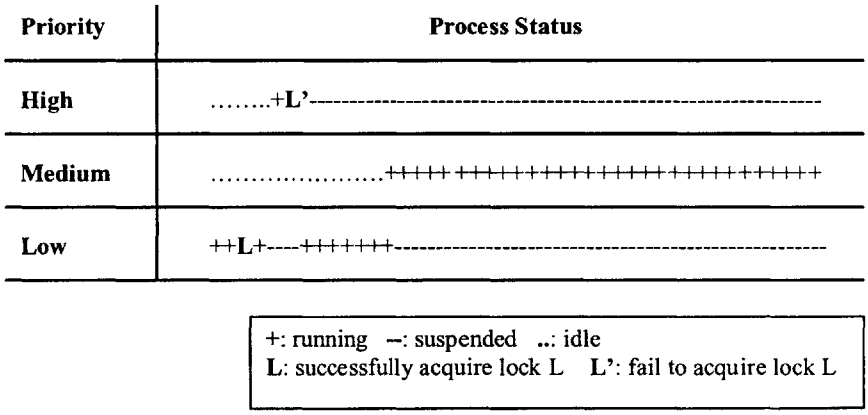

Figure 1: Unbounded Priority Inversion Problem

The Ravenscar Profile uses the Real-Time package defined in Annex D of the language reference manual [ARM95] instead of the Calendar package. This allows the use of a high-fidelity real-time clock. Tasks only use absolute delays for delaying tasks. Consequently, the *delay* statement becomes invalid, and only *delay_until* statement is used.

The Ravenscar Profile makes schedulability analysis feasible for concurrent applications, and thus can be used for mission-critical and safety-critical real-time systems. In addition, as mentioned above the Profile was designed to be suitable for mapping to a small and efficient run-time kernel that supports task synchronization and communication instead of a full Ada run-time system [BDV03]. The small run-time kernel is the hardware run-time kernel RavenHaRT used in this Gurkh project. The details of RavenHaRT will be described in Section 2.4.

2.2 The GNU NYU Ada Translator

Even with the strengths and success of Ada described in section 2.1.1, Ada was spreading very slowly through the software community in the early 1990s. The GNAT project was initiated by New York University to provide a free high-quality compiler that could run on various platforms and could be used in training as well as industrial applications. GNAT (GNU NYU Ada Translator) is both a front-end and a run-time system for Ada95 programs. It is written in Ada95 and uses the back-end GCC code generator. GNAT is part of the GNU software and is distributed according to the guidelines of the Free Software Foundation [SB94].

To prevent *reinventing the wheel* for the Gurkh project, GNAT acts as the foundation on which pGNAT is built. GNAT was chosen for three reasons: First, GNAT is open-source; therefore, all source code with documentations is available for downloads and can readily be used for necessary modifications. Second, it is widely used both in the industry and in research, hence has been tested through public usage. Finally, it utilizes the back-end GCC code generator, which provides an advantage of the GCC retargetable and rehostable capability.

2.2.1 The GNAT Compiler

GNAT is organized into the compiler, binder, and the run-time system. In this thesis, we only focus on the compiler and the run-time system. The compiler comprises the Ada implemented front-end and the back-end. The front-end consists of a parser and an expander. The parser parses the input source code and arranges it into an Abstract Syntax Tree (AST), which is used to communicate between different parts of the compiler. Then, the expander processes the high-level AST received from the parser and passes it to the back-end of the compiler. The back-end comprises Gigi (GNAT to GNU) and the GCC code generator as shown in Figure 2. The use of the GCC code generator allows GNAT to generate code for different platforms supported by GCC [SB94]. Each component is explained in greater detail below.

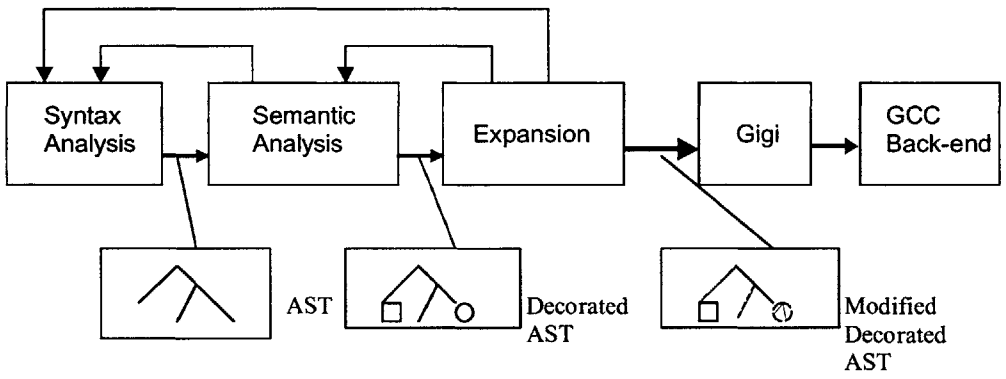

Figure 2: The GNAT Compiler Structure [SB94]

2.2.1.1 GNAT Parser/Expander

GNAT uses a hand-coded recursive descent parser [SB94]. The parser performs both syntax analysis and semantic analysis. The syntax analysis generates a high-level Abstract Syntax Tree and passes it to the semantic analyzer. The semantic analysis unit resolves the names and types in the AST, and decorates the AST with various semantic attributes and finally performs static legality checks for the program. The decorated tree is then passed to the expander. The expander modifies

the AST in order to simplify its translation into the GCC tree. It also transforms high-level AST nodes, such as nodes representing tasks and protected objects, into nodes which call Ada run-time library routines [Mir02]. The output of the expander is then passed to the back-end.

2.2.1.2 GNAT Gigi/Code Generation

Gigi exists for the purpose of interfacing the front-end with the GCC code generator. Gigi traverses the decorated and expanded AST in order to build a corresponding GCC tree. The tree is then passed to the GCC code generator. In order to bridge the semantic differences between Ada and C, several code generation routines in GCC have been extended or added [SB94].

2.2.2 GNAT Run-Time System and GNARL

The GNAT Run-Time System (RTS) provides run-time support for the compiler. To make GNAT portable, the RTS is written in Ada. The compiler communicates with the RTS through procedure and function calls. The RTS consists of three layers: GNARL, GNULL and POSIX Threads [Mir02]. An Ada program calls run-time services through GNARL subprogram calls. These calls then interface with GNULL calls which subsequently call the concurrent programming support provided by the POSIX Threads library. Figure 3 shows the hierarchy of the run-time system.

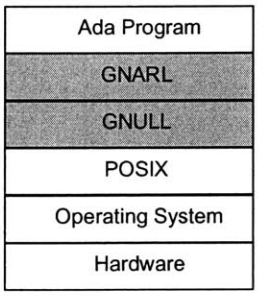

Figure 3: GNAT Ada Run-Time System

2.2.2.1 GNARL

GNARL (GNU Ada Run-Time Library) is an implementation of Ada95 tasking for GNAT. It implements high-level language constructs, such as tasks or protected objects to which the expander can make calls when it encounters these constructs during compilation [GB94]. The details on how GNAT implements tasks, protected objects, and delay statements will be described in Chapter 4. From a design point of view, GNARL also plays an important role because it delays the placement of information from the compile-time to run-time, thereby providing the users with the flexibility to modify the library to fit platform needs. GNARL is inherently complex because it implements Ada tasking semantics. As a result, for porting GNAT to a different platform, it is recommended to modify GNULL instead if possible.

2.2.2.2 GNULL

GNULL (GNU Lower Level Interface) serves as an interface from GNARL to services provided by an operating system or real-time kernel. This eliminates the dependence of GNARL on the underlying OS services, particularly POSIX Threads library [IEEE99] in this case. GNULL abstracts a subset of POSIX library, including POSIX Threads, for example thread creation/operations on mutexes, conditional variables, and signals [Mir02]. GNULL is designed to be very small in order to facilitate the process of porting GNAT to a different platform.

2.2.2.3 POSIX Threads

POSIX is an IEEE standard [IEEE99], providing an interface for applications to services supporting the creation and execution of multiple threads on a single process. These threads share address spaces and file descriptors. GNARL depends on two POSIX standards: 1003.4 and 1003.4a. The 1003.4 interface is an extension for Real-Time facilities providing common services needed by real-time applications. The 1003.4a interface is an extension for threads called Pthreads [IEEE99].

GNARL uses POSIX services to build tasking services and also uses Pthreads scheduler to schedule threads. However, Ada semantics are implemented in the GNARL layer.

The GNAT RTS is very different from the Gurkh run-time architecture. GNAT however serves as a good starting point for pGNAT because the majority of the front-end and back-end of the compiler can be reused. The design decisions for pGNAT are further elaborated in Chapter 3.

2.3 The Gurkh Project

The Gurkh project [AL03] is a joint research and development effort between the Massachusetts Institute of Technology and Mälardalen University in Sweden. The goal of this project is to provide a new framework and environment for the design, verification, and execution of embedded systems used in safety-critical applications. An overview of the Gurkh system is shown in Figure 4.

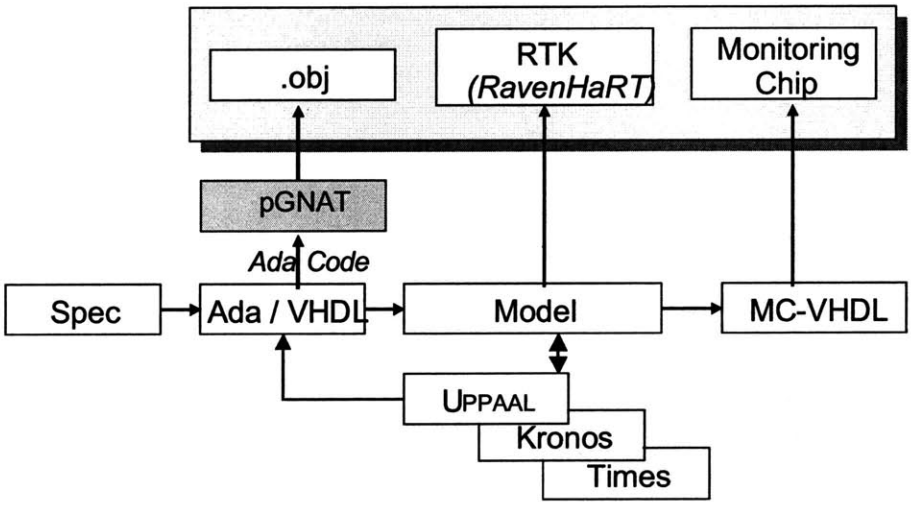

Figure 4: Overview of the Gurkh System

The Gurkh project, whose applications can be written using a combination of Ada and VHDL [VHDL00], is based on three foundational technologies: a predictable implementation of the run-time environment, formal verification, and a system for real-time monitoring for system safety.

- *Predictable implementation of the run-time environment*

  The determinism of the run-time environment attributes to the use of the Ravenscar subset of Ada and the hardware implemented run-time kernel. The Ravenscar Profile provides Gurkh applications with higher-level language safe tasking constructs; therefore, it allows an easier implementation and accurate analysis of the worst-case execution time (WCET) of the system. The hardware implemented Ravenscar-compliant run-time kernel RavenHaRT [Sil04] provides a one-to-one mapping from the automata in the formal model [AL03] to the finite state machines in the hardware [AL03]. This guarantees the predictability of the system.

- *Formal verification*

  Model checking is a formal verification technique which relies on an exhaustive state-space check whether a desired property holds in a finite model of a system [CW96]. The Gurkh project employs model checking to verify the system correctness at design time. A formal model of the hardware run-time kernel RavenHaRT, together with a formal model of the application, are taken through a number of steps to insure a complete verification. First, the Ada and VHDL code is translated to a readable intermediate format. Then, the output in the intermediate format is translated to different formal languages which can be used as an input for various existing verification tools, such as UPPAAL [Upp], Kronos [Yov97], and Times [AFM+02]. These tools then verify both the functional behavior and timing properties of the complete system.

- *System for real-time monitoring for system safety*

  The real-time monitoring system revolves around the Monitoring Chip. First, the information contained in the intermediate language, generated by the formal verification process, is used to create the monitoring chip operating in parallel with the execution of the system. This process reduces the information of tasks and processes in an application into nodes which contain a notation of a best-case and worst-case execution time (BCET and WCET). The Monitoring Chip keeps track of all tasks' states and raises an external interrupt if a BCET or WCET is violated. This enables real-time monitoring for system safety in the Gurkh project.

These three underlying methods enhance system safety. First, the run-time kernel and the application have been completely verified through an exhaustive state-space exploration in model

checking. Nevertheless, there is a possibility that there is a state or input that has not been considered in constructing the formal model of the kernel and the application. This can cause an unaccounted violation to the BCET and WCET, contributing to system failures. The Monitoring Chip however reacts to these occurrences by applying different fault models for each violation or error. Consequently, the system is able to continue delivering correct but degraded services in case of errors.

The physical system, the RTK and the Monitoring Chip, will be implemented on a Xilinx Virtex-II Pro FPGA [XIL02], which contains both FPGA logic and an IBM PowerPC 405 on-chip processor [PPC01]. The physical implementation is shown in Figure 5.

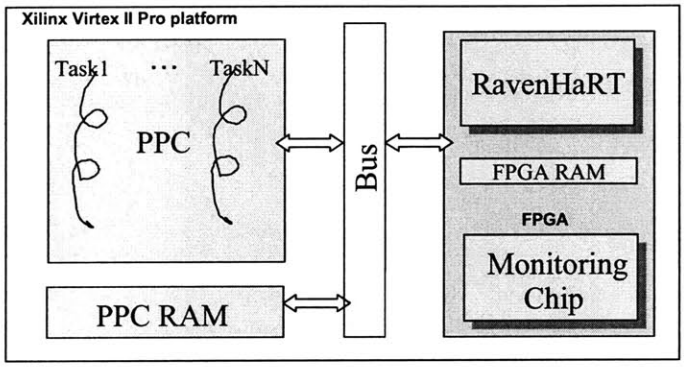

Figure 5: The Gurkh Operational Platform

The Memec Design Virtex-II Pro Development Kit [Mem03] is being used to implement the Gurkh system [Sil04]. This operation platform needs the specialized compiler described in this thesis, to compile the application code written in Ada and produce an object code for the run-time kernel RavenHaRT instead of a commonly available software operating system.

2.4 The RavenHaRT Kernel

The RavenHaRT kernel [Sil04] is a critical component because it provides a predictable run-time environment for the Gurkh system. It acts as an operating system, scheduling all tasks running on the system. RavenHaRT is designed to support Ravenscar-compliant Ada applications, with a

modified Ada compiler for the Gurkh run-time environment. Currently, it supports only a single processor system. RavenHaRT is implemented using Xilinx Virtex-II Pro which contains a PowerPC processor and an FPGA. This requires an interface for the communications between the PowerPC and RavenHaRT synthesized on the FPGA area.

RavenHaRT is composed of the ready queue, the delay queue, protected objects, task arbitration, and VCounter. The ready queue holds task information and schedule all tasks existing in the system according to their priorities. The delay queue keeps track of which tasks have made a *delay_until call* and when they should be set runnable again. RavenHaRT also has routines which implement protected objects (POs) comparable to those in GNARL for task synchronization and offer a delay_until service. Task arbitration solves race issues among tasks running in the system. For example, if a high-priority task wakes up from a delay, so the ready queue requests that this task be run. At the same time, the running may call a PO which changes its priority. This change in priority may not propagate immediately, thus the arbitration code is inserted to determine which task to run in a deterministic fashion [Sil04]. VCounter acts as a counter to keep track of time in the system. Finally, RavenHaRT can process external interrupts. Essentially, RavenHaRT provides thread creation/management as well as mutual exclusion facilities with the use of ceiling priority in POs. RavenHaRT components are shown in Figure 6.

The operational environment of RavenHaRT is different from the traditional run-time system described in 2.2.2. The execution of tasks is carried out on the PowerPC with RavenHaRT acting as the scheduler. In order to schedule tasks, RavenHaRT requires status updates of all tasks from the PowerPC. The status updates and task priorities are used to determine which task to run. RavenHaRT uses the pre-emptive highest-priority scheduling algorithm. Whenever there is any task preemption, the kernel sends an interrupt to the PowerPC. Then, a software interrupt handler would perform a task switch.

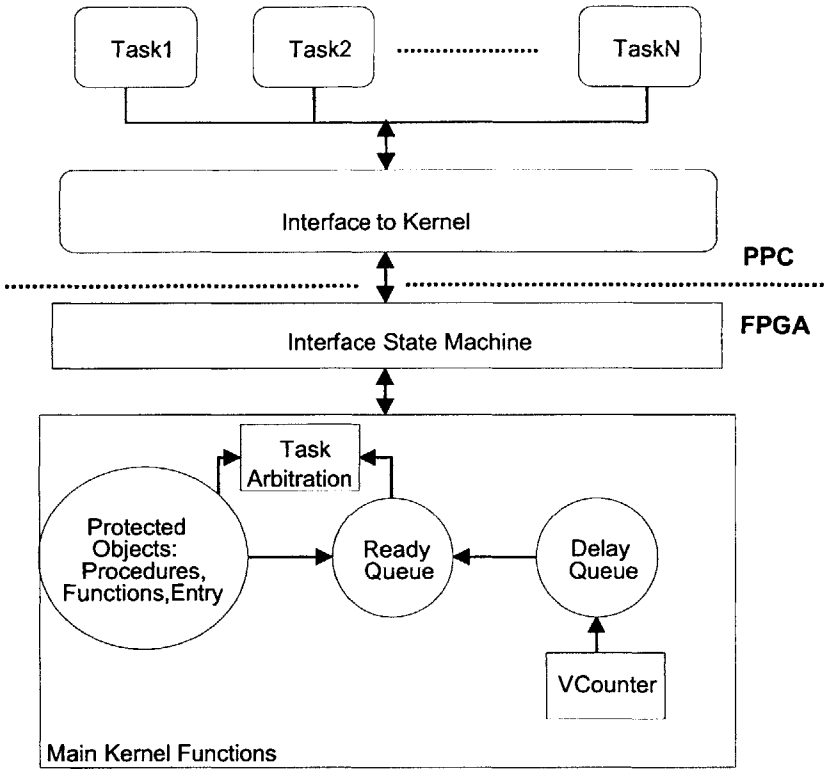

Figure 6: Architecture of the Kernel and Applications [Sil04]

RavenHaRT communicates with the PowerPC via the local bus interface. The bus has a 32-bit input (din) and output (dout) registers. The bus has only two states: wait_state and ack_state. When the PowerPC reads or writes to the bus, the write_n signal is set to high or low respectively. The PowerPC also specifies the address of registers which it will read from or write to. The chip select signal cs_n must be low at all times for reading and writing. After the read and write process is completed, the output ack_n is set to low, which the PowerPC will know that the read and write is done. All this communication protocol is taken care of by the Universal Asynchronous Receiver/Transmitter (UART). An interface package, *System.FPGA_Kernel_Interface*, is created to facilitate the communication between the PowerPC and RavenHaRT. All available RavenHaRT registers are listed in Figure 7.

Register Name	Address	Bit Used (32-bit Register)
Command (input)	00000100	27 | 5 Insignificant | Command
Lower Parameter/Barrier (input)	00001000	Following the Create command: 27 | 2 | 3 Insignificant | TaskID | Task Priority Following the SetFreq command: 24 | 8 Insignificant | VCounter Frequency Following the PO command: 27 | 3 | 2 Insignificant | Ceiling Prio | PO_ID Following the Delay Until or Barrier Request cmd: 32 Lower 32-bit of Delay Until time/ Barrier Values
Higher Parameter (input)	00001100	32 Higher 32-bit of Delay Until Time
Interrupt Acknowledge (input)	00010000	31 | 1 Insignificant | Ack
Status (output)	00010100	29 | 3 Insignificant | Status
New Task_ID (output)	00011000	30 | 2 Insignificant | Task_ID
Lower Time (output)	00011100	32 Lower 32 bits of the Delay Until Time
Higher Time (output)	00100000	32 Higher 32 bits of the Delay Until Time

Figure 7: RavenHaRT Register Addresses and Contents [Sil04]

RavenHaRT provides five services for software tasks running on the PowerPC: Protected Entry, Protected Procedure, Protected Function, Delay Until, and external interrupt handling. Tasks can call a Protected Entry, Protected Procedure, or Protected Function by writing a command into the Command register of RavenHaRT. RavenHaRT will then read the input from the Lower Parameter and the Higher Parameter registers if necessary and provide the services. The complete list of available commands for RavenHaRT is shown in Figure 8.

Command	Instruction Bits
Create	10000
DelayUntil	00001
GetTime	01101
SetFreq	01110
FindTask	01111
FPs	00010
UPe	00011
UFe	10001
FPe	00100
UFPx	00101
UPxe	00110
UFxe	10010
FPx	00111
Es	01000
UEb	01001
UEe	01010
UEx	01011
Ex	01100

Figure 8: Commands and Intruction Bits for RavenHaRT [Sil04]

After a command is processed, RavenHaRT responds to the PowerPC by writing the status to its Status register. The PowerPC then reads the status from the register and acts according to the output. All status signals used by RavenHaRT are shown in Figure 9.

Status	Register Bits
No Status	000
Bad Command	001
Command Done	010
Barrier Request	100

Figure 9: Status Signals Used by RavenHaRT [Sil04]

RavenHaRT is manually specified in VHDL and synthesized on the FPGA fabric of the Xilinx Virtex-II Pro board. Its minimalistic property provides advantages in terms of increased processor usage, better system level determinism, and simplified timing analysis. This makes RavenHaRT suitable for the Gurkh system. Additional information about RavenHaRT can be found in [Sil04].

2.5 Related Work

Previous work has been done in this field to provide a deterministic Ada run-time environment. One project, "An Experimental Testbed for Embedded Real Time Ada95," by Walker, Woolley, and Burns [WWB99], is very similar to pGNAT in the sense that it tries to realize the GNAT real-time programming potential by porting the GNAT public version to a bare processor to create a predictable and effective run-time system. Walker et al. addressed the problem of GNAT being designed to rely on the operating system services which are not deterministic, which makes it difficult to use GNAT to investigate the real-time behavior of Ada95 applications. This is also true for the case of distributed real-time systems due to the non-deterministic behavior of the communication protocols. We are only interested in the first case because the Gurkh system is currently a single processor stand-alone embedded system.

As a solution to this problem, Walker et al. chose to implement a software embedded kernel which provides services in a predictable manner. This kernel replaces the operating system layer of the traditional Ada run-time environment shown in Figure 3. It also interfaces with the Pthread layer to

provide an efficient and predictable implementation of thread management services. The new run-time system is shown in Figure 10.

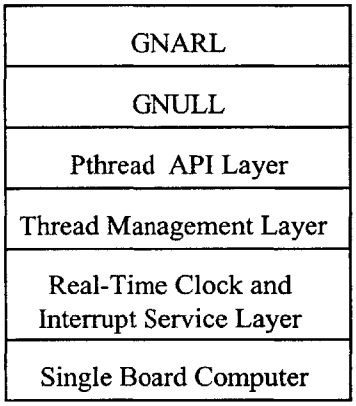

Figure 10: Architecture of the Embedded Software System.

The thread management layer provides only one function, kernel_suspend_and_ call (thread_manager). This kernel function is invoked by the Pthread layer at all scheduling decision points [WWB99]. It performs these steps:

1. Saves the context of the current thread in a thread descriptor maintained by the kernel.
2. Switches the system stack pointer to the thread manager stack.
3. Invokes the thread manager function.

This thread manager function then examines the queue of runnable threads and selects the task which has the highest priority to run. Then, it restores the context, and the selected thread resumes execution.

All the scheduling decisions can be dealt with by a single invocation of the thread manager with the exception of interrupts which would be resolved using the interrupt service layer [WWB99]. The use of the embedded software kernel in place of a commercially available operating system simplifies the thread management calls, resulting in a more efficient, deterministic, and easier to analyze implementation of the thread management services.

These changes are incorporated into GNAT by implementing an embedded kernel in one of the os_interface packages. Then, the cross-compiler version of GNAT is built with these selected

GNARL and GNULL packages. The GNAT linker is also used to produce an executable file for the target platform.

Besides the software embedded kernel implemented in [WWB99], other research has been conducted to make real-time systems more deterministic. Another specialized software kernel was implemented by de la Puente, Ruiz, and Gonzalez-Barahona to replace the POSIX threads because it was too costly for embedded systems and hard to analyze the time bound [PRG99]. Other research done by Ward and Audsley [WA01] took the hardware approach to overcome the indeterminism introduced by software as well as the CPU speed up features such as caches and pipelines. Ward and Audsley chose to implement hardware compilation on FPGAs to provide highly accurate worst-case execution time (WCET) computation. Calculating the WCET of a code block can then be carried out by simply adding together the execution times of the individual instructions.

These related work present solutions for real-time embedded systems to attain predictability and more accurate timing properties. However, they do not leverage the advantages provided by hardware/software co-design. The Gurkh system implements its operating system in hardware, thereby exploiting the inherent parallelism of hardware. RavenHaRT runs in parallel with the application software executing on the PowerPC. This helps RavenHaRT gain speed and efficiency. In addition, RavenHaRT implemented in hardware can be analyzed to obtain more accurate timing behaviors than software operating systems. The Gurkh system also gains the flexibility by using software compilation. In contrast to [WA01], pGNAT compilation process is done in software, thus it provides flexibility for code changes without having to re-synthesize the FPGAs whenever the application is modified. With these advantages of hardware/software co-design, the Gurkh system presents an improved solution to provide determinism in real-time embedded systems. Implementing pGNAT for the Gurkh unique platform introduces many design issues and challenges. This thesis addresses these problems and present solutions in the later chapters.

Chapter 3
pGNAT Design

The Gurkh system uses the Xilinx Virtex-II Pro board as the operating platform. pGNAT is designed to produce object code to run on the embedded PowerPC present on the operating platform. GNAT forms a foundation for pGNAT for three reasons: its open-source nature, stability, and its retargetable and rehostable GCC backend. This chapter addresses the design issues and the methodology used for implementing pGNAT. The GNAT program libraries can be divided into application libraries and the run-time libraries (RTS). The focus of pGNAT is on the run-time libraries.

3.1 pGNAT Overview

The architecture of the RTS can be partitioned into a layered architecture comprising of GNARL, GNULL and POSIX [Mir02] as shown in Figure 11. Application tasks make run-time calls through GNARL. The GNARL calls subsequently make service calls to the GNULL layer, which provides an abstraction to the POSIX library. GNULL is written to be used for many common platforms and the selection/implementation of the appropriate GNULL packages is left to the compiler designer. GNULL provides an interface between the thread services of the target OS to those required by GNARL [WWB99].

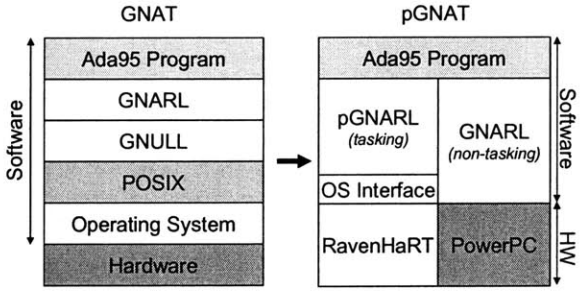

Figure 11: GNAT versus pGNAT

The GNAT POSIX layer provides two functions: task management facilities and mutual exclusion mechanisms. In Gurkh, RavenHaRT services essentially substitute the task management functionality of the POSIX library. In addition, inter-task communication, as defined by the Ravenscar Profile, is provided through the use of protected objects. Enforcing the ceiling priority of protected objects then implicitly provides mutual exclusion. For example, consider two tasks T1, T2 communicating through a protected object PO1, with T1 having higher priority than T2. If the lower-priority task T2 gets access to PO1 first, T2's priority is raised to the ceiling priority of PO1. T1 now has a lower priority than T2, and hence cannot preempt T2 to access PO1. If T1 gets access to PO1 first, the mutual exclusion also holds. As a result, pGNAT eliminates the POSIX Threads layer. Removing POSIX makes GNULL redundant, thus GNULL is also eliminated.

The only remaining RTS layer, GNARL, is partitioned into tasking and non-tasking using manual slicing as discussed in Section 3.2. pGNAT retains the non-tasking portion of GNARL and creates a new version of the tasking sections of GNARL, called pGNARL. The resultant architecture of pGNAT compresses the three-layered GNAT RTS into a single layer with two vertical partitions for handling tasking and non-tasking constructs as shown in Figure 11.

3.2 GNARL Dependency Analysis

The GNARL library consists of approximately 200 source files, each of which has embedded subprograms. A dependency model of the GNARL libraries is created to understand the couplings both internal to GNARL, as well as between GNARL, GNULL and POSIX. The dependency model is manually created using the *with* and *use* clauses within the libraries. The model creation is further guided by the dependencies created by tasks, protected objects and *delay_until* statements.

The GNAT compiler carries out task related operations located in the expander, shown in Figure 12.

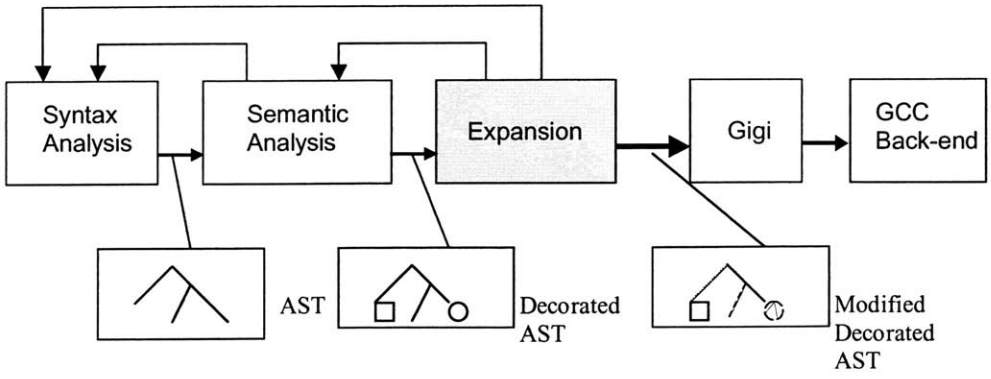

Figure 12: The Expander of the GNAT Compiler [SB94]

3.2.1 Task Dependence

After the front-end compilation has been completed, GNAT calls the RTSFIND package through a subprogram call in the EXP_CH9 package shown in Figure 13 to elaborate the tasking level constructs. The actual tasking level constructs are provided by the packages, *System.Tasking* (s-taskin), *System.Task_Primitives* (s-taspri), *System.Tasking.Stages* (s-tassta), *System.Tasking.Utilities* (s-tasuti) and *System.Tasking.Queuing* (s-tasque) within GNARL, GNULL provides the packages *System.OS_Interface* (s-osinte) and *System.Task.Primitives.Operations* to provide a mapping to the underlying operating system services. The manually obtained dependence graph is shown in Figure 13.

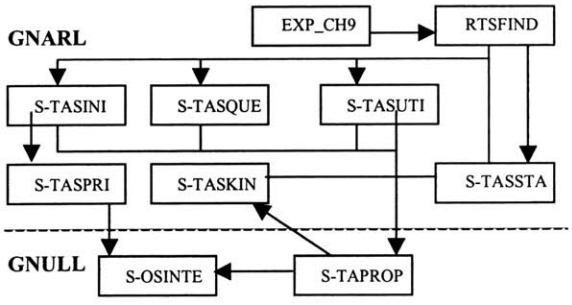

Figure 13: Dependency Graph for Task Creation

GNAT stores each task's information (task state, parent, activator etc) in the run-time system using a per-task record, called the Ada Task Control Block (ATCB). GNAT also uses another register to store task-specific information, which is linked to the POSIX level implementation of a Threads Control Block (TCB) in the POSIX services as shown in Figure 14.

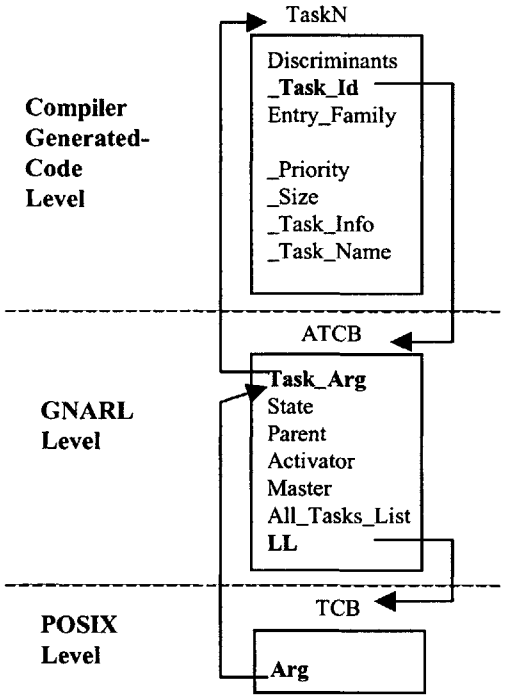

Figure 14: GNAT Tasking Implementation

The sequence of GNARL subprograms called over the life of a task is illustrated in Figure 15. The box on the left shows an activator, which is a task activating the new task. The box on the right is a new task created by the activator. All tasks go through these steps over its lifetime:

1. The *enter_Master* subprogram is called when the compiler encounters the declaration of a task.

2. The *Create_Task* subprogram is used to create the ATCB and insert it into the list of all tasks in the system.

3. The *Activate_Task* subprogram is called to create a new thread and associate it with the activation block. The task wrapper then executes, initializing the fields of the per-task-register's local data.

4. The *Complete_Activation* subprogram is called after all the initialization has been completed and before the first line of the tasks executable code has been executed.

5. The *Complete_Task* subprogram is called to terminate the task and its master. The *Complete_Task* subprogram also includes step 6, shown separately in Figure 15.

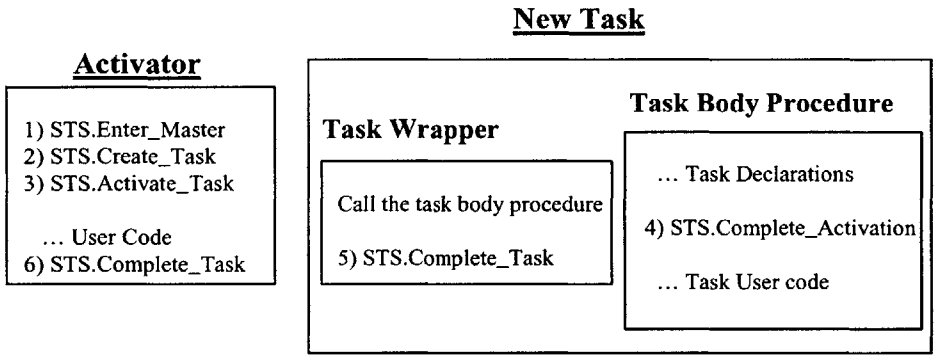

Figure 15: GNARL Subprograms Called During the Task Life-Cycle [Mir02]

The Ravenscar profile of Ada95 mandates that tasks have to exist at the library level and they should never terminate hence steps 5 and6 can safely be ignored. The absence of task hierarchy imposed by this constraint, allows pGNAT to eliminate the Master, Activator, and Parent information and thereby better utilize the scarce memory resources in the system. In addition, with the elimination of POSIX, the LL construct which links to TCB can also be eliminated. The resultant ATCB in *System.Tasking* has only the *State* and *All_Tasks_List* data structures.

3.2.2 Protected Object Dependence

In GNAT, each protected type specification gets translated into a record as shown in Figure 16. Every record contains an _object, *System.Tasking.Protected_Objects.Single_Entry. Protection_Entry*, which is the data structure of protected objects [Mir02].

```
type poV (discriminants) is record
      _object : aliased
System.Tasking.Protected_Objects.Single_Entry.Protection_Entry
      <private data fields>
end record;
```

Figure 16: Protected Type Specification

For protected objects shown in Figure 17, *System.Tasking.Protected_Object.Single_Entry* (s-tposen) defines its data structure *Protection_Entry.* This package depends on *System.Task_ Primitives.Operations* to access the kernel services for protected objects. Other packages, such as *System.Tasking.Protected.Objects* (s-taprob) and *System.Tasking.Entry_Calls* (s-taenca), also depend on *System.Tasking.Protected_Object.Entries* (s-tpoben); however, the use of the Ravenscar Profile, restricts the model to protected objects with single entries. The dependency graph obtained by tracing GNAT operations after front-end compilation is shown in Figure 17.

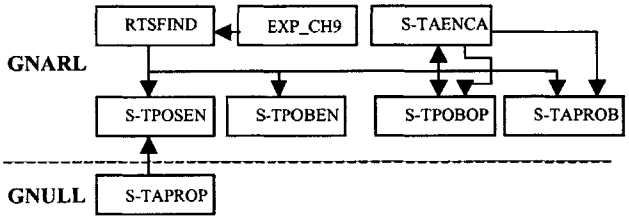

Figure 17: Dependency Graph for PO Construct

GNAT converts each entry procedure into a procedure including the user's code as shown in Figure 18. Every protected entry has an entry barrier, which is used to determine whether the entry is accessible. This entry barrier expression is translated by the compiler to a function that returns a

Boolean type [Mir02]. The access to this subprogram is stored in a table created by the compiler for GNARL to use when evaluating the entry barrier for an entry call.

```
procedure entE
   (O : System.Address;P : System.Address;E : Protected_Entry_Index)
 is
    type poVP is access poV;
    _Object : ptVP := ptVP!(O);
 begin
       <statement sequence>
       Complete_Entry_Body (_Object._Object);
    exception
       when all others =>
          Exceptional_Complete_Entry_Body (
            _Object._Object, Get_GNAT_Exception);
 end entE;
```

Figure 18: Translation of an Entry Body to a Procedure

Protected procedures and protected functions are treated similarly in GNAT as protected entries. Each protected subprogram is translated into two procedures: ProcN and ProcP. ProcN contains the user code while ProcP contains procedures to lock of the protected object, and handle exceptions that occur during the execution. ProcP calls ProcN after acquiring the lock to execute application code. Finally, ProcP calls the at end handler, expanded in exp_ch11.adb, to handle any exceptions occurred. The translated procedures are shown in Figure 19.

```
procedure ProcN (_object : in out poV) is
begin
   <sequence of statements>
end ProcN;

procedure ProcP (_object : in out poV) is
begin
   Abort_Defer.all;
   Lock (_object._object'Access);
   ProcN (_object;...);
at end --  start of at-end handler
   _clean;
end ProcP;
```

Figure 19: Translation of a Protected Subprogram

3.2.3 Delay_Until Dependence

Post front-end compilation, GNAT translates the delay until statement into a call to GNARL, RTE (*Ada.Real_Time.Delay.Delay_Until*). This procedure then calls the procedure *Timed_Delay* in *System.Task_Primitives.Operations.Timed_Delay* which then calls the POSIX function pthread_cond_timedwait to suspend the calling tasks until the specified time. The extracted dependency graph is shown in Figure 20.

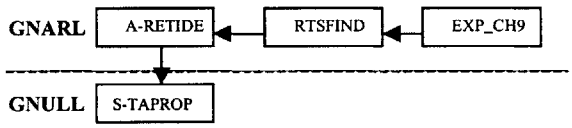

Figure 20: Dependency Graph of Delay_Until Statement

The dependency analysis provides insights about the couplings of GNARL, GNULL, and POSIX. This information is used to extract relevant tasking packages in GNARL. The pGNAT implementation uses the dependency graph to guild the modification of the GNARL library as discussed in Chapter 4.

Chapter 4
pGNAT Implementation

pGNAT implementation eliminates the POSIX and GNULL layer, and replaces the tasking portion of GNARL by pGNARL. RavenHaRT, as discussed in Section 2.4, acts as the scheduler and provides five kernel level services: protected entry (PE), protected procedure (PP), protected function (PF), Delay_Until, and external interrupt handling. Application tasks request these services through an interface package, called *System.FPGA_Kernel_Interface*. This chapter discusses the implementation of the interface package, and details the translation of individual constructs.

4.1 Task Construct

All tasks running on the PowerPC enter different stages over their life cycle by calling corresponding subprograms as shown in Figure 15. Tasks and their ATCBs are created in the second stage, *STS.Create_Task* and activated in the third stage, *STS.Activate_Task*. However, task creation and activation are done differently in hardware. RavenHaRT performs both task creation and task activation during the elaboration phase of the system. To create a task in RavenHaRT, the PowerPC writes the Create command, the task id, and its priority to appropriate RavenHaRT registers, then reads a status from the status register in order to see if the task creation in hardware is complete. Figure 21 shows pseudocode of what RavenHaRT expects the application running on the PowerPC to perform in order to create a task with ID = 2 and priority = 4 in RavenHaRT.

```
Write "Create", Command Register
Write 0x00000014, Lower Parameter Register
Read status, Status Register
While status == 'No Status"
   Read status, Status Register
```

Figure 21: Protocol for Creating a Task in RavenHaRT

RavenHaRT then processes the input, creates the task on the FPGA RAM, and writes back to the status register, informing the PowerPC that the task creation is done. To activate all tasks in RavenHaRT, the FindTask command is used. Once all tasks are created, RavenHaRT waits for the PowerPC to send the FindTask command prior to determining the next runnable task with the highest-priority. After identifying the next running task, RavenHaRT sends an interrupt to the PowerPC to inform what task needs to be run. The PowerPC then sends an interrupt acknowledgement back to RavenHaRT and read the next running task's ID from the New TaskID register. The communication protocol for activating tasks in RavenHaRT is detailed in Figure 22.

Write "FindTask", Command Register
Write 0x00000014, Lower Parameter Register
Read status, Status Register
While status == 'No Status"
  Read status, Status Register
--- Interrupt to PowerPC –
Write 0x00000001, Interrupt Acknowledge Register
Read NextTask, New TaskID Register

Figure 22: Protocol for Activating Tasks in RavenHaRT

pGNAT modifies *Create_Task* and *Activate_Tasks* in *System.Tasking.Stages* of the GNAT implementation. pGNAT puts all tasks in an activation chain (*System.Tasking.Activation_Chain*) prior to sending the FindTask command. The modified *Activate_Tasks* is shown in Figure 23. On receiving the FindTask command, RavenHaRT identifies the highest priority runnable task and uses an interrupt mechanism to carry out task activation.

```
procedure Activate_Tasks (Chain_Access : Activation_Chain_Access) is
      Self_ID         : constant Task_ID := STPO.Self;
      P               : Task_ID;
      C               : Task_ID;
      Next_C, Last_C  : Task_ID;
      Activate_Prio   : System.Any_Priority;
      Success         : Boolean;
      All_Elaborated  : Boolean := True;

   begin
      pragma Debug
        (Debug.Trace (Self_ID, "Activate_Tasks", 'C'));
      Initialization.Defer_Abort_Nestable (Self_ID);
      pragma Assert (Self_ID.Common.Wait_Count = 0);
```

```
--  Lock RTS_Lock, to prevent activated tasks
--  from racing ahead before we finish activating the chain.
--  Check that all task bodies have been elaborated.
C := Chain_Access.T_ID;
Last_C := null;

while C /= null loop
   if C.Common.Elaborated /= null
     and then not C.Common.Elaborated.all
   then
      All_Elaborated := False;
   end if;

   --  Reverse the activation chain so that tasks are
   --  activated in the same order they're declared.

   Next_C := C.Common.Activation_Link;
   C.Common.Activation_Link := Last_C;
   Last_C := C;
   C := Next_C;
end loop;

Chain_Access.T_ID := Last_C;

if not All_Elaborated then
   Initialization.Undefer_Abort_Nestable (Self_ID);
   Raise_Exception
     (Program_Error'Identity,
         "Some tasks have not been elaborated");
end if;

--  Activate all the tasks in the chain.
--  Creation of the thread of control was deferred until
--  activation. So create it now.

C := Chain_Access.T_ID;

while C /= null loop
   if C.Common.State /= Terminated then
      pragma Assert (C.Common.State = Unactivated);

      if C.Common.Base_Priority < Get_Priority (Self_ID) then
         Activate_Prio := Get_Priority (Self_ID);
      else
         Activate_Prio := C.Common.Base_Priority;
      end if;

      System.Task_Primitives.Operations.Create_Task
        (C, Task_Wrapper'Address,
         Parameters.Size_Type
           (C.Common.Compiler_Data.Pri_Stack_Info.Size),
         Activate_Prio, Success);
```

```
      --  There would be a race between the created task and the
      --  creator to do the following initialization, if we did not
      --  have a Lock/Unlock_RTS pair in the task wrapper to prevent
      --  it from racing ahead.

            if Success then
               C.Common.State := Runnable;
      ------------------------------------------------------
      --  Pee's  --
      --  if success, then send "Create" command to the command
      --  register and task priority to lower parameter register in
      --  the HW kernel
            Write_Create_Cmd
              (Interfaces.Unsigned_32 (C.Serial_Number - 100)
               , Interfaces.Unsigned_32 (Activate_Prio));
      ------------------------------------------------------
            else
      --  No need to set Awake_Count, State, etc. here since the loop
      --  below will do that for any Unactivated tasks.

               Self_ID.Common.Activation_Failed := True;
            end if;
         end if;

         C := C.Common.Activation_Link;
      end loop;

   ------------------------------------------------------
   -- Pee's  --
   -- All tasks are created, thus activate tasks now
   Write_FindTask_Cmd;
   ------------------------------------------------------

end Activate_Tasks;
```

Figure 23: Modified Activate_Tasks for pGNAT

4.2 Protected Object Construct

The protected object implementation diverges from traditional approaches. Unlike tasks which are created on the FPGA RAM, protected objects are pre-created on the PowerPC RAM, as state machines in hardware during the synthesis of RavenHaRT. As a consequence, the information of protected objects, such as states, barrier values, and user code, are stored on the PowerPC RAM, and the PowerPC has to update the kernel as it enters different stages of the PO execution.

4.2.1 Protected Entries

The protected entry state machine is shown in Figure 24. When the state machine begins executing, it expects a command Es be written to the command register, Es==1, between E0 and E1. It then requests an entry barrier value. The PowerPC then has to look up the value from the *Protection_Entry* of that protected object and pass the information to RavenHaRT by writing one or zero into one of the input kernel registers. RavenHaRT then evaluates the entry barrier and proceed to E2 if the value is one or E7 if otherwise. In order to keep the application and RavenHaRT synchronized, when a task has to wait for an entry to become true (from E7 to E8), RavenHaRT determines that the task should be preempted by other runnable tasks and sends an interrupt to the PowerPC to cause a task switch.

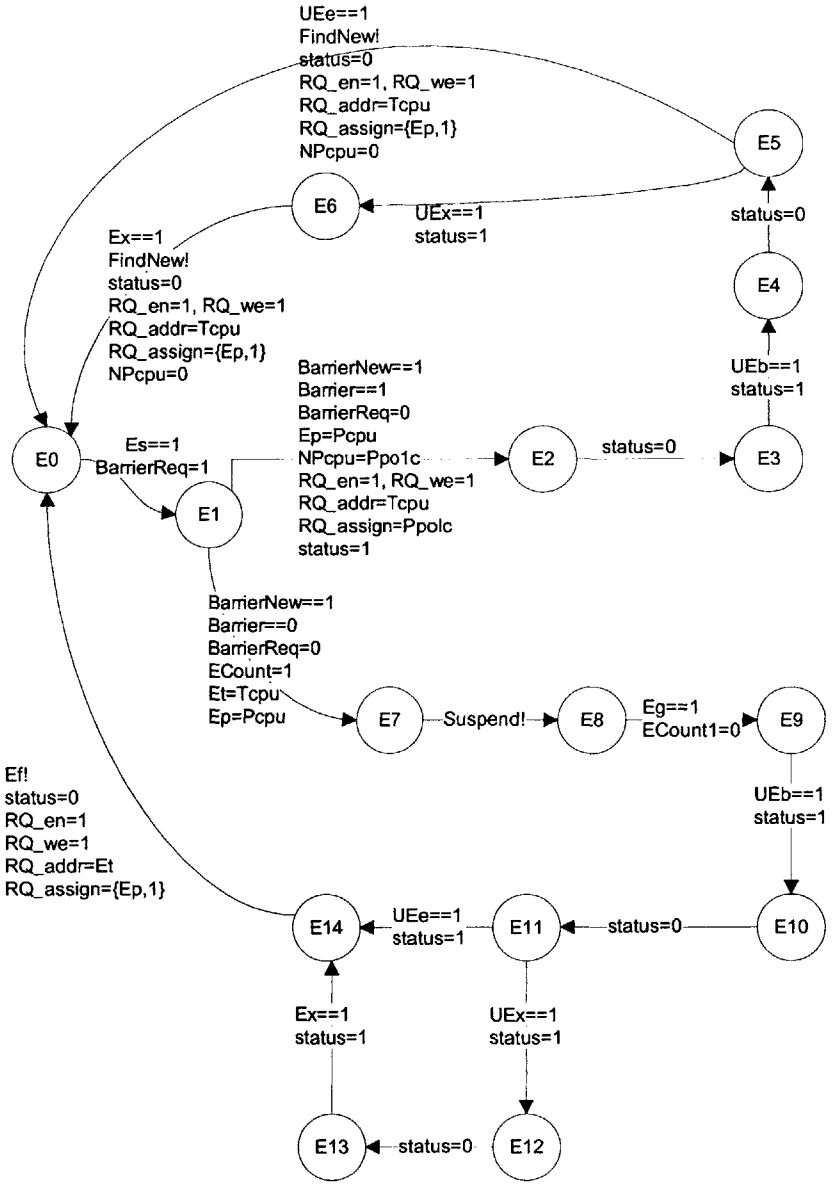

Figure 24: Protected Entry State Machine [Sil04]

This state machine can be presented in another form as a protocol for RavenHaRT protected entry. The pseudocode in Figure 25 shows the protocol of protected entry expected by RavenHaRT.

```
Write Es, Command Register
Write 0x00000012, Lower Parameter Register
Read status, Status Register
While status != 'Barrier Request'
Read status, Status Register
Write Barrier[31:0], Lower Parameter Register
Read status, Status Register
While status == 'No Status'
Read status, Status Register
If(Barrier[POId] == True)
        Write UEb, Command Register
        Write 0x00000002, Lower Parameter Register
        Read status, Status Register
While status == 'No Status'
Read status, Status Register
Set long jump (exception_E)
Jump Entry
Write UEe, Command Register
        Write 0x00000002, Lower Parameter Register
        Read status, Status Register
While status == 'No Status'
Read status, Status Register

exception_E:
Write UEx, Command Register
        Write 0x00000002, Lower Parameter Register
        Read status, Status Register
While status == 'No Status'
Read status, Status Register
Write Ex, Command Register
        Write 0x00000002, Lower Parameter Register
        Read status, Status Register
While status == 'No Status'
Read status, Status Register

If(Barrier[POId] == False)
        -- Interrupt to PPC here ---
Write 0x00000001, Interrupt Acknowledge Register
Read NextTask, New Task ID Register
```

Figure 25: RavenHaRT Protocol for Protected Entry [Sil04]

For each protected entry, the expander of the GNAT compiler generates an entry barrier function and an entry body procedure as shown in Figure 18. Whenever a task tries to access a protected

entry, the procedure *System.Tasking.Protected_Bojects.Operations.Protected_Entry_Call* is invoked. The procedure perform these steps [Mir02]:

1. Defer the abortion.
2. Lock the object for mutual exclusion.
3. Elaborate a new Entry Call Record in *System.Tasking*.
4. Call the GNARL procedure *PO_Do_Or_Queue* to issue the call or to enqueue it in the entry queue.
5. Call the GNARL *System.Tasking.Protected_Objects.Single_Entry.Service_Entry* to service the entry if open (PO can only have a maximum of one entry according to Ravenscar)
6. Unlock the object
7. Undefer the abortion
8. Check if some exception must be re-raised

The call to *PO_Do_Or_Queue* and *Service_Entry* in step 4 and 5 evaluates the barrier by calling the entry barrier function and executes the entry body procedure if the barrier value is true respectively. Both *Protected_Entry_Call* and *PO_Do_Or_Queue* as well as the entry body procedure are modified for pGNAT.

The GNARL modification in pGNAT lies mainly in *Build_Protected_Entry* procedure in the expander, exp_ch9.adb. The expander is modified to include commands which write necessary information to the kernel registers in the procedure *Build_Protected_Entry* which is called by *Expand_N_Protected_Body*. The procedure *Protected_Single_Entry_Call* and *PO_Do_Or_Queue* in *System.Tasking.Protected_Objects.Single_Entry* are also modified to initiate entry calls in RavenHaRT. Figure 26 shows the modified *Protected_Single_Entry_Call* and *PO_Do_Or_Queue*, and Figure 27 shows the modified entry body procedure.

Protected_Single_Entry_Call:

1. *Defer the abortion.*
2. *Call Write_Es_Cmd_Single_Entry to initiate the entry call in RavenHaRT*
3. *Elaborate a new Entry Call Record in System.Tasking.*
4. *Call the GNARL procedure PO_Do_Or_Queue to issue the call or to enqueue it in the entry queue.*
5. *Call the GNARL System.Tasking.Protected_Objects.Single_Entry.Service_Entry to service the entry if open (PO can only have a maximum of one entry according to Ravenscar)*
6. *Undefer the abortion*
7. *Check if some exception must be re-raised*
8. *Call Write_UEe_Cmd_Single_Entry to initiate an entry exit in RavenHaRT*

PO_Do_Or_Queue:

1. *Call the barrier function*
2. *Call Write_Entry_Barrier to service the barrier request by RavenHaRT*
3. *If the barrier is closed then enqueue the Entry Call Record and return*
4. *If the barrier is open then execute the entry body within the calling task context*

Figure 26: Modified Protected_Single_Entry_Call and PO_Do_Or_Queue

```
procedure entE
    (O : System.Address;P : System.Address;E : Protected_Entry_Index)
  is
     type poVP is access poV;
     _Object : ptVP := ptVP!(O);
  begin
        Write_Ueb_Cmd_Single_Entry;  --  RavenHaRT
        <statement sequence>
        Complete_Entry_Body (_Object._Object);
     exception
        Write_Uex_Cmd_Single_Entry;  --  RavenHaRT
        when all others =>
           Exceptional_Complete_Entry_Body (
             _Object._Object, Get_GNAT_Exception);
           Wirite_Ex_Cmd_Single_Entry;  --  RavenHaRT
  end entE;
```

Figure 27: Modified Entry Body Procedure

4.2.2 Protected Procedures/Functions

The routines provided by RavenHaRT to implemented protected subprograms can be modeled as a state machine to that in Figure 28. The state machine shows all states which RavenHaRT traverses and all inputs it expects from the PowerPC to keep RavenHaRT's states synchronized with those of the application. To start a protected subprogram, the task running on the PowerPC sends the command FPs and a parameter containing the ceiling priority and the PO's ID to RavenHaRT. Then, the protected subprogram would move from P0 to P1. The state machine determines whether the call is for protected procedure or protected function, then proceeds to P3 or P6 respectively. At P3, an entry barrier evaluation is required, thus RavenHaRT requests the PowerPC to send the entry barrier value to it.

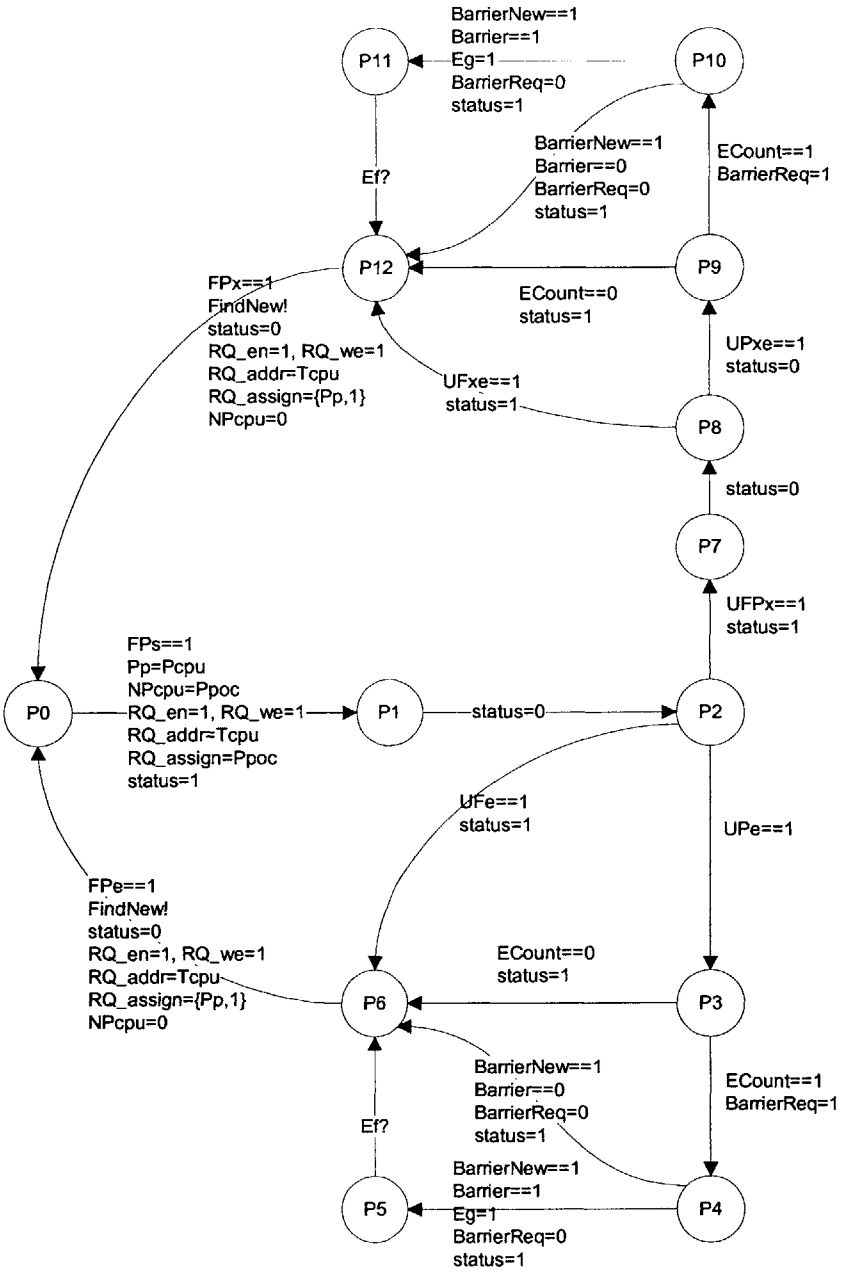

Figure 28: Protected Procedure/Function State Machine [Sil04]

The complete RavenHaRT protocol for protected procedure/function calls is derived from the state machine. Figure 29 shows the complete RavenHaRT protocol for protected procedure/function calls for a protected object with PO_ID = 2 and ceiling priority = 2.

Write FPs, Command Register
Write 0x00000012, Lower Parameter Register
Read status, Status Register
While status == 'No Status'
Read status, Status Register
Set long jump (exception_P)
Jump Procedure
Write UPe, Command Register
Write 0x00000002, Lower Parameter Register
Read status, Status Register
While status == 'No Status'
Read status, Status Register
If(status == 'Barrier Request')
Write Barrier[31:0], Lower Parameter Register
    Read status, Status Register
While status == 'No Status'
Read status, Status Register
If(Barrier[POId] == True)
    Write UEb, Command Register
    Write 0x00000002, Lower Parameter Register
Read status, Status Register
While status == 'No Status'
Read status, Status Register
Set long jump (exception_E)
Jump Entry
Write UEe, Command Register
Write 0x00000002, Lower Parameter Register
Read status, Status Register
While status == 'No Status'
Read status, Status Register

exception_E:
Write UEx, Command Register
Write 0x00000002, Lower Parameter Register
Read status, Status Register
While status == 'No Status'
Read status, Status Register
Write Ex, Command Register
Write 0x00000002, Lower Parameter Register
Read status, Status Register
While status == 'No Status'
Read status, Status Register

Write FPe, Command Register
Write 0x00000002, Lower Parameter Register
Read status, Status Register
While status == 'No Status'
Read status, Status Register

exception_P:

Write UFPx, Command Register
Write 0x00000002, Lower Parameter Register
Read status, Status Register
While status == 'No Status'
Read status, Status Register
Write UPxe, Command Register
Write 0x00000002, Lower Parameter Register
Read status, Status Register
While status == 'No Status'
Read status, Status Register
If(status == 'Barrier Request')
---REPEAT CODE FROM SAME 'IF' STATEMENT ABOVE---
Write FPx, Command Register
Write 0x00000002, Lower Parameter Register
Read status, Status Register
While status == 'No Status'
Read status, Status Register

Figure 29: RavenHaRT Protocol for Protected Procedure/Function Calls [Sil04]

The GNARL modification required to enable this interaction between the kernel and the PowerPC is made in the expander, exp_ch9.adb. For example, the FPs command and a parameter need to be sent at the beginning of ProcP by using a procedure from *System.FPGA_Kernel_Interface* to write the data to the kernel registers. The modified *EXP_CH9.Build_Protected_Subprogram_Body*, shown in Figure 30, incorporates these communications expected by RavenHaRT for protected procedures and protected functions.

```
procedure ProcN (_object : in out poV;...) is
  begin
     Write_UPe_Cmd_Single_Entry/ Write_UFe_Cmd_Single_Entry;
      -- RavenHaRT
     <sequence of statements>
  end ProcN;

  procedure ProcP (_object : in out poV;...) is
  begin
     Write_FPs_Cmd_Single_Entry;  -- RavenHaRT
     Abort_Defer.all;
     Lock (_object._object'Access);
     Write_UPe_Cmd_Single_Entry/ Write_UFe_Cmd_Single_Entry;
      -- RavenHaRT
     pprocN (_object;...);
     _clean;
     Write_FPe_Cmd_Single_Entry; -- RavenHaRT
```

```
exception
   when others =>
        declare
              E : Exception_Occurrence;
        begin
              Write_UFPx_Cmd_Single_Entry;  -- RavenHaRT
              GNARL.Save_Occurrence (E, GNARL.Get_Current_Exception);
              Write_UPxe_Cmd_Single_Entry/ Write_UFxe_Cmd_Single_Entry
               -- RavenHaRT
              _clean;
              GNARL.Reraise (E);
              Write_Fpx_Cmd_Single_Entry -- RavenHaRT
        end;
  end ProcP;
```

Figure 30: Modified Prtected Procedure/Function Calls

4.3 Delay_Until Statement

The Ravenscar Profile requires the delay statement to use absolute time. Therefore, only delay_until statements are used in the application software. The GNAT implementation of the delay_until statement is described in Section 3.2.3. A task calling *Delay_Until* eventually calls the procedure *Timed_Delay* in *System.Task_Primitives.Operations.Timed_Delay,* which then calls the POSIX function *pthread_cond_timedwait* to suspend the calling tasks until the specified time

Calls to delay_until service provided by RavenHaRT are implemented as a two-stage process: GetTime and DelayUnitl. The GetTime command calculates the delay time relative to the time maintained by the kernel. The DelayUntil command writes the delay time to corresponding registers. The complete protocol for RavenHaRT is shown in Figure 31.

Write GetTime, Command Register
Read status, Status Register
While status == 'No Status'
    Read status, Status Register
Read VCounter[31:0], Lower Time Register
Read VCounter[63:32], Higher Time Register
Write DelayUntil, Command Register
Write DelayTime[31:0], Lower Parameter Register
Write DelayTime[63:32], Higher Parameter Register
    Read status, Status Register
While status == 'No Status'
    Read status, Status Register
--- Interrupt to PPC here ---
Write 0x00000001, Interrupt Acknowledge Register
Read NextTask, New Task ID Register

Figure 31: RavenHaRT Protocol for the Delay_Until Statement [Sil04]

RavenHaRT then puts the task to sleep and schedules the next task on the Ready Queue. To modify GNARL to conform with the RavenHaRT protocol, *System.Task_Primitives.Operations.Timed_Delay* is modified to call *System.FPGA_Kernel_Interface.Write_DelayUntil_Cmd.* The modified procedure *Timed_Delay* is shown in Figure 32.

```
procedure Timed_Delay  (Self_ID  : Task_ID; Time : Duration;
                        Mode : ST.Delay_Modes) is
begin
    Write_DelayUntil_Cmd (Time);
end Timed_Delay;
```

Figure 32: Modified Delay_Until Statement

In addition to the *Timed_Delay* procedure, the *Ada.Real_Time* package is modified to create a mapping of the *Delay_Until* call provided by GNARL to the equivalent RavenHaRT services. The *Ada.Real_Time* package modification is described in Section 4.4.

4.4 Ada Time Package

RavenHaRT runs at a speed of 100 Mhz; therefore, to synchronize the FPGA clock VCounter and the speed of Ada application, the real time resolution has to be set equal to 10 nanoseconds. Currently, GNAT uses a resolution of one nanosecond, defined in *Ada.Real_Time* (a-reatim.ads). pGNAT sets *Ada.Real_Time.Time_Unit* and *Ada.Real_Time.Time_Span_Unit* are set to 10#1.0#EE-8 to synchronize the FPGA and Ada application clock.

4.5 pGNAT Verification

pGNAT has been used to successfully compile a Ravenscar compliant implementation of a generic navigation system. The –gnatG switch is used during compilation for the manual inspection of the translation process. The inspection showed that the communication between the PowerPC and RavenHaRT was established correctly. An example of an inspection is detailed in Appendix B. pGNAT has not yet been validated against the Ada compiler verification suite and is left for future work.

Chapter 5
Conclusions

The use of the RavenHaRT kernel in the Gurkh system requires a radical re-design of the compiler. This thesis presents a compiler called pGNAT, which exploit the benefits of hardware/software co-design, to meet the Gurkh system's requirements of RavenHaRT compatibility and Ravenscar compliance. The redesigned run-time system of pGNAT is simpler than that of GNAT, and produces deterministic, analyzable object code.

5.1 Limitations

Currently, pGNAT only addresses tasking and inter-task communication. pGNARL has been created to handle tasks, protected objects, and the delay_until statement. Applications running on the PPC can create tasks and have them communicate through protected objects. Certain critical capabilities such as interrupt handling, exception handling, and linker scripts need to be added to pGNAT before it can be used outside the academic environment.

pGNAT has only been unit tested and verified using manual inspection. pGNAT validation against the Ada Compiler Evaluation System [ACES] is left for future work.

5.2 Future Work

Future work entails additional implementations as well as testing. The additional functionality within pGNAT that need to be implemented are interrupt handling, exception handling, and linker scripts.

5.2.1 Interrupt Handling

The Gurkh system requires an interrupt handling mechanism in order to perform task switching and external input processing. When task preemption occurs, RavenHaRT sends an interrupt to the PowerPC. The processor needs to process this interrupt and wake up the preempting task. In addition, when the Gurkh system receives an external interrupt, a corresponding handler has to process the information. Currently, this functionality is absent from pGNAT and needs to be implemented in the future.

5.2.2 Exception Handling

Exception handling is necessary for a real-time safety-critical system. In an event of failures, a safety-critical system should be able to provide safe degraded services through the use of user-defined exception handling. Despite its necessity, both GNAT and pGNAT does not implement exception handling for the Gurkh operational platform.

Exception handler for other platforms is implemented in GNAT based on two POSIX calls: setjmp() and longjmp(). These two procedures are called to save and restore the stacks of the currently running task. To implement the exception handling facility in pGNAT, the procedures for saving and restoring stacks have to be implemented. These procedures can then be used in place of setjmp() and longjmp().

5.2.3 Linking

Exception handling and interrupt handling both require an understanding of the linking process. A linker script is necessary for pGNAT to produce an executable of an application. Writing the linker script requires an understanding of the memory and addresses of the Xilinx board and the libraries of newlib and libc.

5.2.4 Optimizations

pGNAT needs to be optimized in terms of space. Currently, the ATCB still maintains GNAT specific information that is not necessary for the Ravenscar Profile. The modification described in Chapter 4 only eliminates information such as masters, parents, and activators. The safe deletion of additional information in the ATCB has to be carried out after determining all the dependencies within the system.

5.2.5 Validation

pGNAT also has to be validated for industry use by verifying it against the Ada Compiler Validation Capability test suites. In addition, greater load testing needs to be carried out to ensure that pGNAT scales with increasing application software size.

Chapter 6

References

[Ada] www.adaic.org. (April 28, 2004)

[ACES] http://www.stsc.hill.af.mil/crosstalk/1994/05/xt94d05g.asp, (May 20, 2004)

[AFM+02] Amnell, T., Fersman, E., Mokrushin, L., Pettersson, P., Yi, W., "TIMES – A Tool for Modeling and Implementation of Embedded Systems," 8th Euromicro Workshop on Real-Time Systems, pp. 164-168, 1996.

[AL03] Asplund, L. Lundqvist, K., "The Gurkh Project: a Framework for Verification and Execution of Mission Critical Applications," Digital Avionics Systems Conference, 2003.

[ARM83] United States Department of Defense, "Reference Manual for the Ada Programming Language," Washington, DC, 1983.

[ARM95] Ada95 Reference Manual, Language and Standard Libraries, Version 6.0

[BDR98] Burns, A., Dobbing, B., Romanski, G., "The Ravenscar Tasking Profile for High Integrity Real-Time Programs," Ada-Europe 98, LNCS 1411, pp. 263-275, 1998.

[BDV03] Burns, A., Dobbing, B., Vardanega, T., "Guide for the Use of the Ada Ravenscar Profile in High Integrity Systems," University of York Technical Report YCS-2003-348, 2003.

[BW97] Burns, A., Wellings, A., "Real-Time Systems and Programming Languages," Addison-Wesley Longman Limited, Essex, England, pp. 400-403, 1997.

[Bar96] Barnes, J., "Programming in Ada95," Addison-Wesley Publishing Company Inc., 1996.

[CW96] Clarke, E. M., Wing, J. M., "Formal methods: State of the art and future directions," Tech. Rep. CMU-CS-96-178, Carnegie Mellon University, 1996.

[GB94] Giering, E. W., Baker, T. P., "The Gnu Ada Runtime Library (GNARL): Design and Implementation," Proceedings of the eleventh annual Washington Ada symposium & summer ACM SIGAda meeting on Ada, pp. 97-107, 1994.

[GRM02] GNAT Reference Manual, Ada Core Technologies, Inc., GNAT, The GNU Ada 95 Compiler version 3.15p, Document revision level 1.247.2.3, 2002.

[IEEE99] Information technology requirements and guidelines for test methods, specifications and test method implementations for measuring conformance to POSIX' standards, ISO/IEC 13210, 1999.

[Mem03] Virtex-II Pro (P4/P7) Development Board User's Guide, version 3.0, Memec Design, 2003.

[Mir02] Miranda, Javier, "A Detailed Description of the GNU Ada Run Time", Version 1.0, Applied Microelectronics Research Institute University of Las Palmas de Gran Canaria, Canary Islands, Spain, 2002.

[PPC01] "PowerPC 405 Embedded Processor Core User's Manual," Fifth Edition, 2001.

[PRG99] de la Puente, J. A., Ruiz, J. F., Gonzalez-Barahona, J. M., "Real-Time Programming with GNAT: Specialised Kernels versus POSIX Threads," Proceedings of the ninth international workshop on Real-time Ada, pp. 73-77, 1999.

[Rus02] Russell, J., "Program Slicing for Codesign," Proceedings of the tenth international symposium on Hardware/software codesign, pp. 91-96, 2002.

[SB94] Schonberg, E., Banner, B., "The GNAT project: a GNU-Ada 9X compiler," Proceedings of the conference on TRI-Ada '94, pp. 48 – 57, 1994.

[Sil04] Silbovitz, A., "The Ravenscar-Compliant Hardware Run-Time Kernel," Massachusetts Institute of Technology, 2004

[Upp] http://www.uppaal.com, April 22, 2003.

[VHDL00] IEEE Standard VHDL Language Reference Manual, IEEE Std 1076, The Institute of Electrical and Electronics Engineers, Inc., New York, NY, 2000.

[WA01] Ward, M., Audsley, N. C., "Synthesis and Design Tools: Hardware compilation of sequential Ada," Proceedings of the international conference on Compilers, architecture, and synthesis for embedded systems, Pages: 99 - 107, 2001.

[Wic98] Wichmann, B., "Guidance for the Use of the Ada Programming Language in High Integrity Systems", ACM SIGAda Ada Letters, Vol. 18, Issue 4, pp. 47-94, 1998.

[WWB99] Walker, W. M., Woolley P. T., Burns A., "An experimental testbed for embedded real time Ada 95," International Workshop on Real-time Ada Issues, Proceedings of the ninth international workshop on Real-time Ada , pp. 84-89, 1999.

[XIL02] "Virtex-II Pro Platform FPGA Handbook," v1.0, http://www.xilinx.com, April 22, 2003.

[Yov97] Yovine, S., "Kronos: A Verification tool for real-time systems," Springer International Journal of Software Tools for Technology Transfer 1(1/2), 1997.

Appendix A

System.FPGA_Kernel_Interface

This appendix shows the specification and the body of the System.FPGA _Kernel_Interface package. This package implements the services offered by RavenHaRT, which application tasks call upon. The specification contains RavenHaRT specific variables, such as the number of tasks and POs. If these variables are modified in RavenHaRT, the variables in the specification have to be modified accordingly. The body of the package shows every implementation detail of the interface package.

System.FPGA_Kernel_Interface Specification

```
------------------------------------------------------------------------------
--
--                GNU ADA RUN-TIME LIBRARY (GNARL) COMPONENTS
--         S Y S T E M . F P G A _ K E R N E L _ I N T E R F A C E
--                                                                          -
--                                  S p e c                                 -
--                                                                          -
--              Author       :  Pee Seeumpornroj                           -
--              Version      :  2/26/04                                    -
--              Supervisor   :  Kristina Lundqvist                         -
--                                                                          -
--              Embedded Systems Laboratory                                 -
--              Massachusetts Institute of Technology    --
--                                                                          -
-- GNARL is free software; you can  redistribute it  and/or modify it under -
-- terms of the  GNU General Public License as published  by the Free Soft- -
-- ware  Foundation;  either version 2,  or (at your option) any later ver- -
-- sion. GNARL is distributed in the hope that it will be useful, but WITH- -
-- OUT ANY WARRANTY;  without even the  implied warranty of MERCHANTABILITY -
-- or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License -
-- for  more details.  You should have  received  a copy of the GNU General -
-- Public License  distributed with GNARL; see file COPYING.  If not, write -
-- to  the Free Software Foundation,  59 Temple Place - Suite 330,  Boston, -
-- MA 02111-1307, USA.                                                      -
--                                                                          -
-- As a special exception,  if other files  instantiate  generics from this -
-- unit, or you link  this unit with other files  to produce an executable, -
-- this  unit  does not  by itself cause  the resulting  executable  to  be -
-- covered  by the  GNU  General  Public  License.  This exception does not -
-- however invalidate  any other reasons why  the executable file  might be -
-- covered by the  GNU Public License.                                      -
--                                                                          -
------------------------------------------------------------------------------
--  This is a package which provides an interface to RavenHaRT Kernel
--  It defines locations of registers, commands, and procedures to read/write
```

```
--  from the kernel's registers via the bus

with Interfaces;
with System.Storage_Elements;
with System.Tasking.Protected_Objects.Entries;
--  used for Protection_Entries_Access
with System.Tasking.Protected_Objects.Single_Entry;
--  used for Protection_Entry_Access

package System.FPGA_Kernel_Interface is

   ----------------------------
   --  RavenHaRT's variables  --
   ----------------------------

   --  The base address and total number of tasks and POs will be moved
   --  to RavenHaRT package

   --  Base Address for Registers
   UART_Base_Address : constant := 16#0000_0000#;

   --  Total number of tasks and POs
   Tot_Tasks : constant Integer := 2;
   Tot_POs   : constant Integer := 4;
   Highest_Prio : constant Integer := 7;

   --  Data for registers
   type Register_Data is new Interfaces.Unsigned_32;

   --  Encoding of commands
   --  commands are encoded with 8 bits
   subtype Command is Register_Data range 0 .. 2**8-1;
   Create     : constant Command := 16#10#;
   DelayUntil : constant Command := 16#01#;
   GetTime    : constant Command := 16#0D#;
   SetFreq    : constant Command := 16#0E#;
   FindTask   : constant Command := 16#0F#;
   FPs        : constant Command := 16#02#;
   UPe        : constant Command := 16#03#;
   UFe        : constant Command := 16#11#;
   FPe        : constant Command := 16#04#;
   UFPx       : constant Command := 16#05#;
   UPxe       : constant Command := 16#06#;
   UFxe       : constant Command := 16#12#;
   FPx        : constant Command := 16#07#;
   Es         : constant Command := 16#08#;
   UEb        : constant Command := 16#09#;
   UEe        : constant Command := 16#0A#;
   UEx        : constant Command := 16#0B#;
   Ex         : constant Command := 16#0C#;

   --  Register type stores 32-bit data
   type Register is new Interfaces.Unsigned_32;
   --  Address of each register in memory
   Command_Reg_Addr     : constant := 2#00000100#;
   Low_Param_Reg_Addr   : constant := 2#00001000#;
   Hig_Param_Reg_Addr   : constant := 2#00001100#;
   Intrrpt_Ack_Reg_Addr : constant := 2#00010000#;
   Stat_Reg_Addr        : constant := 2#00010100#;
   New_TaskID_Reg_Addr  : constant := 2#00011000#;
```

```
Low_Time_Reg_Addr     : constant := 2#00011100#;
Hig_Time_Reg_Addr     : constant := 2#00100000#;

--  Set address for each register
Command_Reg : Register;
for Command_Reg'Address use
   System.Storage_Elements.To_Address (Command_Reg_Addr);

Low_Param_Reg : Register;
for Low_Param_Reg'Address use
   System.Storage_Elements.To_Address (Low_Param_Reg_Addr);

Hig_Param_Reg : Register;
for Hig_Param_Reg'Address use
   System.Storage_Elements.To_Address (Hig_Param_Reg_Addr);

Intrrpt_Ack_Reg : Register;
for Intrrpt_Ack_Reg'Address use
   System.Storage_Elements.To_Address (Intrrpt_Ack_Reg_Addr);

Stat_Reg : Register;
for Stat_Reg'Address use
   System.Storage_Elements.To_Address (Stat_Reg_Addr);

New_TaskID_Reg : Register;
for New_TaskID_Reg'Address use
   System.Storage_Elements.To_Address (New_TaskID_Reg_Addr);

Low_Time_Reg : Register;
for Low_Time_Reg'Address use
   System.Storage_Elements.To_Address (Low_Time_Reg_Addr);

Hig_Time_Reg : Register;
for Hig_Time_Reg'Address use
   System.Storage_Elements.To_Address (Hig_Time_Reg_Addr);


--  Possible status signals from Status Register
subtype Status is Register_Data range 0 .. 2**3-1;
No_Status   : constant Status := 2#000#;
Bad_Command : constant Status := 2#001#;
Cmd_Done    : constant Status := 2#010#;
Barrier_Req : constant Status := 2#100#;


-------------------------------------------------------------
--  Procedures/Functions to communicate with the FPGA Kernel  --
-------------------------------------------------------------

--  Write_Create_Cmd  is used for creating task.  It Writes
--  create command to the Command_Reg and task_ID and task
--  priority to Low_Param_Reg
---------------------------------
--  Input : Id   := task ID
--          Prio := task priority
--  Output: Null
---------------------------------
procedure Write_Create_Cmd (Id, Prio : Interfaces.Unsigned_32);

--  Write_GetTime_Cmd reads time counter (VCounter) in the FPGA
```

```
--  and store the lower 32 bits in Low_Param_Reg and Hig_Time_Reg
---------------------------------
--  Input : Null
--  Output: Null (stored in Low/Hig_Time_Reg
---------------------------------
procedure Write_GetTime_Cmd;

--  Write_DelayUntil_Cmd is used by task to perform a delayuntil
--  command.  It writes Low_Time to Low_Param_Reg and Hig_Time
--  to Hig_Param_Reg
---------------------------------
--  Input : Time := absolute delay time from now (in Duration)
--  Output: Null
---------------------------------
procedure Write_DelayUntil_Cmd (Time : Duration);

--  Write_SetFreq_Cmd is used by task to perform a set frequency
--  of RavenHaRT during elaboration phase.  It writes 8-bit frequency
--  from Low_Param register to RavenHaRT.  For now, it should be set to
--  1 for frequency which means 1x the kernel clock speed, and the kernel
--  clock speed should be 100MHz.
---------------------------------
--  Input : Frequency (8-bit)
--  Output: Null
---------------------------------
procedure Write_SetFreq_Cmd (Freq : Integer);

--  Write_FindTask_Cmd is used by task to find the next task to run.
--  It takes no input and produces no output.  RavenHaRT however will
--  put the new task_ID in the New_TaskID_Reg.
---------------------------------
--  Input : Null
--  Output: Null
---------------------------------
procedure Write_FindTask_Cmd;

--  Write_Es_Cmd is used in System.Tasking.Protected_Objects.Operations.
--  Protected_Entry_Call and System.Tasking.Protected_Objects.Single_Entry.
--  Protected_Single_Entry_Call to insert a handshake protocol with
--  RavenHaRT when a task begins to execute an entry call.  The procedure
--  writes Es to Command_Reg and Ceiling_Prio and PO_Id to Low_Param_Reg.
--  This is a general version that works with multiple entries per PO
---------------------------------
--  Input : Object := Protection_Entries_Access (pointer to
--  data structure of PO)
--  Output: Null
---------------------------------
procedure Write_Es_Cmd
  (Object
     : System.Tasking.Protected_Objects.Entries.Protection_Entries_Access);

--  Write_Es_Cmd_* is used in System.Tasking.Protected_Objects.Operations.
--  Protected_Entry_Call and System.Tasking.Protected_Objects.Single_Entry.
--  Protected_Single_Entry_Call to insert a handshake protocol with
--  RavenHaRT when a task begins to execute an entry call.  The procedure
--  writes Es to Command_Reg and Ceiling_Prio and PO_Id to Low_Param_Reg.
--  This is a ravenscar version that works with one entry per PO
---------------------------------
--  Input : Object := Protection_Entry_Access (pointer to
--  data structure of PO)
```

```
--  Output: Null
---------------------------------
procedure Write_Es_Cmd_Single_Entry
  (Object :
     System.Tasking.Protected_Objects.Single_Entry.Protection_Entry_Access);

--  Write_UEb_Cmd is used by the expander (exp_ch9.adb) to insert
--  a handshake protocol with RavenHaRT when it begins to execute
--  entry code of the protected entry.  The procedure writes UEb
--  to Command_Reg and PO_Id (stored in Object) to Low_Param_Reg.
--  This is a general version that works with multiple
--  entries per PO
---------------------------------
--  Input : Object := Protection_Entries_Access (pointer to
--  data structure of PO)
--  Output: Null
---------------------------------
procedure Write_UEb_Cmd
  (Object
     : System.Tasking.Protected_Objects.Entries.Protection_Entries_Access);

--  Write_UEb_Cmd is used by the expander (exp_ch9.adb) to insert
--  a handshake protocol with RavenHaRT when it begins to execute
--  entry code of the protected entry.  The procedure writes UEb
--  to Command_Reg and PO_Id (stored in Object) to Low_Param_Reg.
--  This is a ravenscar version that works with one entry per PO
---------------------------------
--  Input : Object := Protection_Entry_Access (pointer to
--  data structure of PO)
--  Output: Null
---------------------------------
procedure Write_UEb_Cmd_Single_Entry
  (Object :
     System.Tasking.Protected_Objects.Single_Entry.Protection_Entry_Access);

--  Write_UEx_Cmd is used by the expander (exp_ch9.adb) to insert
--  a handshake protocol with RavenHaRT when an exception occurs while
--  executing the entry code of the protected entry.  The procedure
--  writes UEx to Command_Reg and PO_Id to Low_Param_Reg.
--  This is a general version that works with multiple entries per PO
---------------------------------
--  Input : Object := Protection_Entries_Access (pointer to
--  data structure of PO)
--  Output: Null
---------------------------------
procedure Write_UEx_Cmd
  (Object
     : System.Tasking.Protected_Objects.Entries.Protection_Entries_Access);

--  Write_UEx_Cmd is used by the expander (exp_ch9.adb) to insert
--  a handshake protocol with RavenHaRT when an exception occurs while
--  executing the entry code of the protected entry.  The procedure
--  writes UEx to Command_Reg and PO_Id to Low_Param_Reg.
--  This is a ravenscar version that works with one entry per PO
---------------------------------
--  Input : Object := Protection_Entry_Access (pointer to
--  data structure of PO)
--  Output: Null
---------------------------------
procedure Write_UEx_Cmd_Single_Entry
```

```
  (Object :
     System.Tasking.Protected_Objects.Single_Entry.Protection_Entry_Access);

--  Write_Ex_Cmd is used by the expander (exp_ch9.adb) to insert
--  a handshake protocol with RavenHaRT when the exception is dealt with
--  and exiting the entry body.  The procedure writes Ex to Command_Reg and
--  PO_Id to Low_Param_Reg.
--  This is a general version that works with multiple entries per PO
---------------------------------
--  Input : Object := Protection_Entries_Access (pointer to
--  data structure of PO)
--  Output: Null
---------------------------------
procedure Write_Ex_Cmd
  (Object
     : System.Tasking.Protected_Objects.Entries.Protection_Entries_Access);

--  Write_Ex_Cmd is used by the expander (exp_ch9.adb) to insert
--  a handshake protocol with RavenHaRT when the exception is dealt with
--  and exiting the entry body.  The procedure writes Ex to Command_Reg and
--  PO_Id to Low_Param_Reg.
--  This is a ravenscar version that works with one entry per PO
---------------------------------
--  Input : Object := Protection_Entry_Access (pointer to
--  data structure of PO)
--  Output: Null
---------------------------------
procedure Write_Ex_Cmd_Single_Entry
  (Object :
     System.Tasking.Protected_Objects.Single_Entry.Protection_Entry_Access);

--  Write_UEe_Cmd is used in System.Tasking.Protected_Objects.Operations.
--  Protected_Entry_Call and System.Tasking.Protected_Objects.Single_Entry.
--  Protected_Single_Entry_Call to insert a handshake protocol with
--  RavenHaRT when a task is about to exit an entry call.  The procedure
--  writes UEe to Command_Reg and PO_Id to Low_Param_Reg.
--  This is a general version that works with multiple entries per PO
---------------------------------
--  Input : Object := Protection_Entries_Access (pointer to
--  data structure of PO)
--  Output: Null
---------------------------------
procedure Write_UEe_Cmd
  (Object
     : System.Tasking.Protected_Objects.Entries.Protection_Entries_Access);

--  Write_UEe_Cmd is used in System.Tasking.Protected_Objects.Operations.
--  Protected_Entry_Call and System.Tasking.Protected_Objects.Single_Entry.
--  Protected_Single_Entry_Call to insert a handshake protocol with
--  RavenHaRT when a task is about to exit an entry call.  The procedure
--  writes UEe to Command_Reg and PO_Id to Low_Param_Reg.
--  This is a ravenscar version that works with one entry per PO
---------------------------------
--  Input : Object := Protection_Entry_Access (pointer to
--  data structure of PO)
--  Output: Null
---------------------------------
procedure Write_UEe_Cmd_Single_Entry
  (Object :
     System.Tasking.Protected_Objects.Single_Entry.Protection_Entry_Access);
```

```
--  Write_FPs_Cmd is used by the expander (exp_ch9.adb) to insert
--  a handshake protocol with RavenHaRT when it is about to start
--  a func/procedure call.  The procedure writes FPs to Command_Reg
--  and PO_Id to Low_Param_Reg.
--  This is a general version that works with multiple entries per PO
---------------------------------
--  Input : Object := Protection_Entries_Access (pointer to
--  data structure of PO)
--  Output: Null
---------------------------------
procedure Write_FPs_Cmd
  (Object
     : System.Tasking.Protected_Objects.Entries.Protection_Entries_Access);

--  Write_FPs_Cmd is used by the expander (exp_ch9.adb) to insert
--  a handshake protocol with RavenHaRT when it is about to start
--  a func/procedure call.  The procedure writes FPs to Command_Reg
--  and PO_Id to Low_Param_Reg.
--  This is a ravenscar version that works with one entry per PO
---------------------------------
--  Input : Object := Protection_Entry_Access (pointer to
--  data structure of PO)
--  Output: Null
---------------------------------
procedure Write_FPs_Cmd_Single_Entry
  (Object :
     System.Tasking.Protected_Objects.Single_Entry.Protection_Entry_Access);

--  Write_FPs_Cmd is used by the expander (exp_ch9.adb) to insert
--  a handshake protocol with RavenHaRT when it is about to start
--  a func/procedure call.  The procedure writes FPs to Command_Reg
--  and PO_Id to Low_Param_Reg.
--  This is a version that works with no entry in PO
---------------------------------
--  Input : Object := Protection (pointer to
--  data structure of PO --> in s-taprob)
--  Output: Null
---------------------------------
procedure Write_FPs_Cmd_No_Entry
  (Object :
     System.Tasking.Protected_Objects.Protection_Access);

--  Write_UPe_Cmd is used by the expander (exp_ch9.adb) to insert
--  a handshake protocol with RavenHaRT when it begins to execute
--  procedure body (procN).  The procedure writes UPe to Command_Reg
--  and PO_Id to Low_Param_Reg.
--  This is a general version that works with multiple entries per PO
---------------------------------
--  Input : Object := Protection_Entries_Access (pointer to
--  data structure of PO)
--  Output: Null
---------------------------------
procedure Write_UPe_Cmd
  (Object
     : System.Tasking.Protected_Objects.Entries.Protection_Entries_Access);

--  Write_UPe_Cmd is used by the expander (exp_ch9.adb) to insert
--  a handshake protocol with RavenHaRT when it begins to execute
--  execute procedure body.  The procedure writes UPe to Command_Reg
```

```
--  and PO_Id to Low_Param_Reg.
--  This is a ravenscar version that works with one entry per PO
---------------------------------
--  Input : Object := Protection_Entry_Access (pointer to
--  data structure of PO)
--  Output: Null
---------------------------------
procedure Write_UPe_Cmd_Single_Entry
  (Object :
     System.Tasking.Protected_Objects.Single_Entry.Protection_Entry_Access);

--  Write_UPe_Cmd is used by the expander (exp_ch9.adb) to insert
--  a handshake protocol with RavenHaRT when it begins to execute
--  execute procedure body.  The procedure writes UPe to Command_Reg
--  and PO_Id to Low_Param_Reg.
--  This is a  version that works with no entry in PO
---------------------------------
--  Input : Object := Protection (pointer to
--  data structure of PO --> in s-taprob)
--  Output: Null
---------------------------------
procedure Write_UPe_Cmd_No_Entry
  (Object :
     System.Tasking.Protected_Objects.Protection_Access);

--  Similar to Write_UPe_Cmd, but for protected functions
---------------------------------
--  Input : Object := Protection_Entries_Access (pointer to
--  data structure of PO)
--  Output: Null
---------------------------------
procedure Write_UFe_Cmd
  (Object
     : System.Tasking.Protected_Objects.Entries.Protection_Entries_Access);

--  Similar to Write_UPe_Cmd, but for protected functions
--  This is a ravenscar version that works with one entry per PO
---------------------------------
--  Input : Object := Protection_Entry_Access (pointer to
--  data structure of PO)
--  Output: Null
---------------------------------
procedure Write_UFe_Cmd_Single_Entry
  (Object :
     System.Tasking.Protected_Objects.Single_Entry.Protection_Entry_Access);

--  Similar to Write_UPe_Cmd, but for protected functions
--  This is a  version that works with no entry in PO
---------------------------------
--  Input : Object := Protection (pointer to
--  data structure of PO --> in s-taprob)
--  Output: Null
---------------------------------
procedure Write_UFe_Cmd_No_Entry
  (Object :
     System.Tasking.Protected_Objects.Protection_Access);

--  Write_UFPx_Cmd is used by the expander (exp_ch9.adb) to insert
--  a handshake protocol with RavenHaRT when it encouters an exception
--  while executing protected function/procedure. The procedure writes
```

```
--  UFPx to Command_Reg and PO_Id to Low_Param_Reg.
--  This is a general version that works with multiple entries per PO
---------------------------------
--  Input : Object := Protection_Entries_Access (pointer to
--  data structure of PO)
--  Output: Null
---------------------------------
procedure Write_UFPx_Cmd
  (Object
     : System.Tasking.Protected_Objects.Entries.Protection_Entries_Access);

--  Write_UFPx_Cmd is used by the expander (exp_ch9.adb) to insert
--  a handshake protocol with RavenHaRT when it encouters an exception
--  while executing protected function/procedure. The procedure writes
--  UFPx to Command_Reg and PO_Id to Low_Param_Reg.
--  This is a ravenscar version that works with one entry per PO
---------------------------------
--  Input : Object := Protection_Entry_Access (pointer to
--  data structure of PO)
--  Output: Null
---------------------------------
procedure Write_UFPx_Cmd_Single_Entry
  (Object :
     System.Tasking.Protected_Objects.Single_Entry.Protection_Entry_Access);

--  Write_UFPx_Cmd is used by the expander (exp_ch9.adb) to insert
--  a handshake protocol with RavenHaRT when it encouters an exception
--  while executing protected function/procedure. The procedure writes
--  UFPx to Command_Reg and PO_Id to Low_Param_Reg.
--  This is a version that works with no entry in PO
---------------------------------
--  Input : Object := Protection (pointer to
--  data structure of PO --> s-taprob)
--  Output: Null
---------------------------------
procedure Write_UFPx_Cmd_No_Entry
  (Object :
     System.Tasking.Protected_Objects.Protection_Access);

--  Write_UPxe_Cmd is used by the expander (exp_ch9.adb) to insert
--  a handshake protocol with RavenHaRT when it begins to process
--  an exception in protected function/procedure if any. The procedure
--  writes UPxe to Command_Reg and PO_Id to Low_Param_Reg.
--  This is a general version that works with multiple entries per PO
---------------------------------
--  Input : Object := Protection_Entries_Access (pointer to
--  data structure of PO)
--  Output: Null
---------------------------------
procedure Write_UPxe_Cmd
  (Object
     : System.Tasking.Protected_Objects.Entries.Protection_Entries_Access);

--  Write_UPxe_Cmd is used by the expander (exp_ch9.adb) to insert
--  a handshake protocol with RavenHaRT when it begins to process
--  an exception in protected function/procedure if any. The procedure
--  writes UPxe to Command_Reg and PO_Id to Low_Param_Reg.
--  This is a ravenscar version that works with one entry per PO
---------------------------------
--  Input : Object := Protection_Entry_Access (pointer to
```

```
--  data structure of PO)
--  Output: Null
---------------------------------
procedure Write_UPxe_Cmd_Single_Entry
  (Object :
     System.Tasking.Protected_Objects.Single_Entry.Protection_Entry_Access);

--  Write_UPxe_Cmd is used by the expander (exp_ch9.adb) to insert
--  a handshake protocol with RavenHaRT when it begins to process
--  an exception in protected function/procedure if any. The procedure
--  writes UPxe to Command_Reg and PO_Id to Low_Param_Reg.
--  This is a version that works with no entry in PO
---------------------------------
--  Input : Object := Protection (pointer to
--  data structure of PO --> in s-taprob)
--  Output: Null
---------------------------------
procedure Write_UPxe_Cmd_No_Entry
  (Object :
     System.Tasking.Protected_Objects.Protection_Access);

--  Similar to Write_UPxe_Cmd, but for protected function
---------------------------------
--  Input : Object := Protection_Entries_Access (pointer to
--  data structure of PO)
--  Output: Null
---------------------------------
procedure Write_UFxe_Cmd
  (Object
     : System.Tasking.Protected_Objects.Entries.Protection_Entries_Access);

--  Similar to Write_UPxe_Cmd, but for protected function
---------------------------------
--  Input : Object := Protection_Entry_Access (pointer to
--  data structure of PO)
--  Output: Null
---------------------------------
procedure Write_UFxe_Cmd_Single_Entry
  (Object :
     System.Tasking.Protected_Objects.Single_Entry.Protection_Entry_Access);

--  Similar to Write_UPxe_Cmd, but for protected function
---------------------------------
--  Input : Object := Protection (pointer to
--  data structure of PO)
--  Output: Null
---------------------------------
procedure Write_UFxe_Cmd_No_Entry
  (Object :
     System.Tasking.Protected_Objects.Protection_Access);

--  Write_FPx_Cmd is used by the expander (exp_ch9.adb) to insert
--  a handshake protocol with RavenHaRT when the exception is dealt with
--  and exiting the protected func/procedure body.  The procedure writes
--  FPx to Command_Reg and PO_Id to Low_Param_Reg.
--  This is a general version that works with multiple entries per PO
---------------------------------
--  Input : Object := Protection_Entries_Access (pointer to
--  data structure of PO)
--  Output: Null
```

```
---------------------------------
procedure Write_FPx_Cmd
  (Object
     : System.Tasking.Protected_Objects.Entries.Protection_Entries_Access);

--  Write_FPx_Cmd is used by the expander (exp_ch9.adb) to insert
--  a handshake protocol with RavenHaRT when the exception is dealt with
--  and exiting the protected func/procedure body.  The procedure writes
--  FPx to Command_Reg and PO_Id to Low_Param_Reg.
--  This is a ravenscar version that works with one entry per PO
---------------------------------
--  Input : Object := Protection_Entry_Access (pointer to
--  data structure of PO)
--  Output: Null
---------------------------------
procedure Write_FPx_Cmd_Single_Entry
  (Object :
     System.Tasking.Protected_Objects.Single_Entry.Protection_Entry_Access);

--  Write_FPx_Cmd is used by the expander (exp_ch9.adb) to insert
--  a handshake protocol with RavenHaRT when the exception is dealt with
--  and exiting the protected func/procedure body.  The procedure writes
--  FPx to Command_Reg and PO_Id to Low_Param_Reg.
--  This is a version that works with no entry in PO
---------------------------------
--  Input : Object := Protection (pointer to
--  data structure of PO)
--  Output: Null
---------------------------------
procedure Write_FPx_Cmd_No_Entry
  (Object :
     System.Tasking.Protected_Objects.Protection_Access);

--  Write_FPe_Cmd is used by the expander (exp_ch9.adb) to insert
--  a handshake protocol with RavenHaRT when exiting the protected
--  func/procedure body without any exception.  The procedure writes
--  FPe to Command_Reg and PO_Id to Low_Param_Reg.
--  This is a general version that works with multiple entries per PO
---------------------------------
--  Input : Object := Protection_Entries_Access (pointer to
--  data structure of PO)
--  Output: Null
---------------------------------
procedure Write_FPe_Cmd
  (Object
     : System.Tasking.Protected_Objects.Entries.Protection_Entries_Access);

--  Write_FPe_Cmd is used by the expander (exp_ch9.adb) to insert
--  a handshake protocol with RavenHaRT when exiting the protected
--  func/procedure body without any exception.  The procedure writes
--  FPe to Command_Reg and PO_Id to Low_Param_Reg.
--  This is a ravenscar version that works with one entry per PO
---------------------------------
--  Input : Object := Protection_Entry_Access (pointer to
--  data structure of PO)
--  Output: Null
---------------------------------
procedure Write_FPe_Cmd_Single_Entry
  (Object :
     System.Tasking.Protected_Objects.Single_Entry.Protection_Entry_Access);
```

```
   --  Write_FPe_Cmd is used by the expander (exp_ch9.adb) to insert
   --  a handshake protocol with RavenHaRT when exiting the protected
   --  func/procedure body without any exception.  The procedure writes
   --  FPe to Command_Reg and PO_Id to Low_Param_Reg.
   --  This is a ravenscar version that works with one entry per PO
   ---------------------------------
   --  Input : Object := Protection (pointer to
   --  data structure of PO)
   --  Output: Null
   ---------------------------------
   procedure Write_FPe_Cmd_No_Entry
     (Object :
        System.Tasking.Protected_Objects.Protection_Access);

   --  Write_Entry_Barrier is used in System.Tasking.Protected_Objects.
   --  PO_Do_Or_Queue and System.Tasking.Protected_Objects.Single_Entry.
   --  PO_Do_Or_Queue to write entry barrier value to RavenHaRT
   ---------------------------------
   --  Input : Barrier := 1 or 0 converted into Register_Data
   --  Output: Null
   ---------------------------------
   procedure Write_Entry_Barrier (Barrier : Boolean);

   --  Please use with caution.  If an interface is available above,
   --  please use the interface instead
   --  Read_Reg reads data from Register Reg, return null if no data
   ---------------------------------
   --  Input : Reg  := RavenHaRT register
   --  Output: Register_Data stored in Reg
   ---------------------------------
   function Read_Reg (Reg : in Register) return Register_Data;

   --  Get PO serial number for PO_Id
   function Get_PO_Serial_Number return Integer;

end System.FPGA_Kernel_Interface;
```

System.FPGA_Kernel_Interface Body

```
------------------------------------------------------------------------------
--
--                GNU ADA RUN-TIME LIBRARY (GNARL) COMPONENTS
--         S Y S T E M . F P G A _ K E R N E L _ I N T E R F A C E
--                                                                          -
--                                  B o d y                                 -
--                                                                          -
--             Author      :  Pee Seeumpornroj                             -
--             Version     :  2/26/04                                      -
--             Supervisor  :  Kristina Lundqvist                           -
--                                                                          -
--             Embedded Systems Laboratory                                 -
--             Massachusetts Institute of Technology    --
--                                                                          -
-- GNARL is free software; you can  redistribute it  and/or modify it under -
-- terms of the  GNU General Public License as published  by the Free Soft- -
-- ware  Foundation;  either version 2,  or (at your option) any later ver- -
-- sion. GNARL is distributed in the hope that it will be useful, but WITH- -
-- OUT ANY WARRANTY;  without even the  implied warranty of MERCHANTABILITY -
-- or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License -
-- for  more details.  You should have  received  a copy of the GNU General -
-- Public License  distributed with GNARL; see file COPYING.  If not, write -
-- to  the Free Software Foundation,  59 Temple Place - Suite 330,  Boston, -
-- MA 02111-1307, USA.                                                      -
--                                                                          -
-- As a special exception,  if other files  instantiate  generics from this -
-- unit, or you link  this unit with other files  to produce an executable, -
-- this  unit  does not  by itself cause  the resulting  executable  to  be -
-- covered  by the  GNU  General  Public  License.  This exception does not -
-- however invalidate  any other reasons why  the executable file  might be -
-- covered by the  GNU Public License.                                      -
--                                                                          -
------------------------------------------------------------------------------


--  This is a package which provides an interface to RavenHaRT Kernel
--  It defines locations of registers, commands, and procedures to read/write
--  from the kernel's registers via the bus

with Interfaces;
with System.Machine_Code;
with System.Tasking.Protected_Objects.Entries;
--  used for Protection_Entries_Access
with System.Tasking.Protected_Objects.Single_Entry;
--  used for Protection_Entry_Access
use Interfaces;

package body System.FPGA_Kernel_Interface is

   -------------------------
   --  Local Subprograms  --
   -------------------------
   --  Assume that I is a natural number
   function Bit_Encode (I : Integer) return Integer;

   function Bit_Encode (I : Integer) return Integer is
      Result : Integer := 0;
      Tmp : Integer;
```

```
begin
   Tmp := I mod 2;
   Tmp := Tmp + I;
   --  now Tmp should be divisible by 2
   --  Take care of corner case
   if (Tmp = 0) then
      return Result;
   end if;

   --  figure out how many bits we need
   while (Tmp /= 1) loop
      Tmp := Tmp / 2;
      Result := Result + 1;
   end loop;
   return Result;
end Bit_Encode;

-----------------------
--  Local Variables  --
-----------------------
--  number of bits need to encode Tasks, POs, and Priority
Bit_Task : Integer := Bit_Encode (Tot_Tasks);
Bit_PO   : Integer := Bit_Encode (Tot_POs);
Bit_Prio : Integer := Bit_Encode (Highest_Prio);

--  PO serial number used for PO_Id
Next_PO_Serial_Number : Integer := 0;


--------------------------------
--  Local Subprograms (Cont.) --
--------------------------------
--  Write_Reg writes Register_Data Data to Register Reg
--  Could be blocking because it needs to wait until the kernel
--  write a response to the status register
---------------------------------
--  Input : Reg  := RavenHaRT register
--          Data := data written to Reg
--  Output: Null
---------------------------------
procedure Write_Reg
  (Reg : in out Register; Data : in Register_Data);

--  Create register data for command Create
function Create_Cmd_Data
  (Id, Prio : Interfaces.Unsigned_32) return Register_Data;

--  Create register data for command Es and FPs
function Es_FPs_Cmd_Data
  (Id, Ceiling_Prio : Interfaces.Unsigned_32) return Register_Data;

procedure Write_Reg
  (Reg : in out Register; Data : in Register_Data) is
begin
   System.Machine_Code.Asm ("eieio");
   Reg := Register (Data);
   System.Machine_Code.Asm ("eieio");

end Write_Reg;
```

```
function Create_Cmd_Data
  (Id, Prio : Interfaces.Unsigned_32) return Register_Data is
   Result : Interfaces.Unsigned_32;
begin
   Result := Interfaces.Unsigned_32 ((2**(Bit_Task) - 1)) and Id;
   Result := Interfaces.Shift_Left (Result, Bit_Prio);
   Result := Result or (Interfaces.Unsigned_32
                        ((2**Bit_Prio - 1)) and Prio);
   return Register_Data (Result);
end Create_Cmd_Data;

function Es_FPs_Cmd_Data
  (Id, Ceiling_Prio : Interfaces.Unsigned_32) return Register_Data is
   Result : Interfaces.Unsigned_32;
begin
   Result := Interfaces.Unsigned_32 ((2**Bit_Prio - 1)) and Ceiling_Prio;
   Result := Interfaces.Shift_Left (Result, Bit_PO);
   Result := Result or (Interfaces.Unsigned_32 ((2**Bit_PO - 1)) and Id);
   return Register_Data (Result);
end Es_FPs_Cmd_Data;


-------------------
--  Subprograms  --
-------------------

procedure Write_Create_Cmd (Id, Prio : Interfaces.Unsigned_32) is
   Stat : Status := No_Status;
begin
   Write_Reg (Command_Reg, Create);
   Write_Reg (Low_Param_Reg, Create_Cmd_Data (Id, Prio));

   --  wait til we get command done
   Stat := Status (Read_Reg (Stat_Reg));
   while (Stat = No_Status) loop
      Stat := Status (Read_Reg (Stat_Reg));
   end loop;
end Write_Create_Cmd;

procedure Write_GetTime_Cmd is
   Stat : Status := No_Status;
begin
   --  Write command GetTime to Command_Reg
   Write_Reg (Command_Reg, GetTime);
   --  wait til we get command done
   Stat := Status (Read_Reg (Stat_Reg));
   while (Stat = No_Status) loop
      Stat := Status (Read_Reg (Stat_Reg));
   end loop;
end Write_GetTime_Cmd;

procedure Write_DelayUntil_Cmd (Time : Duration) is
   Stat : Status := No_Status;
   Low  : Register_Data;
   Hig  : Register_Data;
   Start_Time : Unsigned_64;
   Delay_Time : Unsigned_64;
begin
   --  first, get time from VCounter
   Write_GetTime_Cmd;
```

```
   --  Read 64-bit time from Low_Time_Reg and Hig_Time_Reg
   Low := Read_Reg (Low_Time_Reg);
   Hig := Read_Reg (Hig_Time_Reg);

   Start_Time := Shift_Left (Unsigned_64 (Hig), 32) or Unsigned_64 (Low);

   --  Calculate delay time (64 bits)
   Delay_Time := Start_Time + Unsigned_64 (Time);

   Low := Register_Data ((16#00000000FFFFFFFF# and Delay_Time));
   Hig := Register_Data (Shift_Right (Delay_Time, 32));

   --  Write Delay Until Command to Command_Reg
   Write_Reg (Command_Reg, DelayUntil);
   Write_Reg (Low_Param_Reg, Low);
   Write_Reg (Hig_Param_Reg, Hig);

   --  wait til we get command done
   Stat := Status (Read_Reg (Stat_Reg));
   while (Stat = No_Status) loop
      Stat := Status (Read_Reg (Stat_Reg));
   end loop;
end Write_DelayUntil_Cmd;

procedure Write_SetFreq_Cmd (Freq : Integer) is
   Stat : Status := No_Status;
begin
   --  Write SetFreq Command to Command_Reg and 1 to Low_Param_Reg
   Write_Reg (Command_Reg, SetFreq);
   Write_Reg (Low_Param_Reg, Register_Data (Freq));
   --  wait til we get command done
   Stat := Status (Read_Reg (Stat_Reg));
   while (Stat = No_Status) loop
      Stat := Status (Read_Reg (Stat_Reg));
   end loop;
end Write_SetFreq_Cmd;

procedure Write_FindTask_Cmd is
   Stat : Status := No_Status;
begin
   --  Write FindTask command to the Command_Reg
   Write_Reg (Command_Reg, FindTask);
   --  wait til we get command done
   Stat := Status (Read_Reg (Stat_Reg));
   while (Stat = No_Status) loop
      Stat := Status (Read_Reg (Stat_Reg));
   end loop;
end Write_FindTask_Cmd;


procedure Write_Es_Cmd
  (Object :
     System.Tasking.Protected_Objects.Entries.Protection_Entries_Access) is
   Stat : Status := No_Status;
begin
   Write_Reg (Command_Reg, Es);
   Write_Reg (Low_Param_Reg,
              Es_FPs_Cmd_Data
              (Interfaces.Unsigned_32 (Object.PO_Id)
```

```
                , Interfaces.Unsigned_32 (Object.Ceiling)));
   --  wait til we get command done
   Stat := Status (Read_Reg (Stat_Reg));
   while (Stat /= Barrier_Req) loop
      Stat := Status (Read_Reg (Stat_Reg));
   end loop;
end Write_Es_Cmd;

procedure Write_Es_Cmd_Single_Entry
  (Object :
     System.Tasking.Protected_Objects.Single_Entry.Protection_Entry_Access)
is
   Stat : Status := No_Status;
begin
   Write_Reg (Command_Reg, Es);
   Write_Reg (Low_Param_Reg,
              Es_FPs_Cmd_Data
              (Interfaces.Unsigned_32 (Object.PO_Id)
               , Interfaces.Unsigned_32 (Object.Ceiling)));
   --  wait til we get command done
   Stat := Status (Read_Reg (Stat_Reg));
   while (Stat /= Barrier_Req) loop
      Stat := Status (Read_Reg (Stat_Reg));
   end loop;
end Write_Es_Cmd_Single_Entry;

procedure Write_UEb_Cmd
  (Object :
     System.Tasking.Protected_Objects.Entries.Protection_Entries_Access) is
   Stat : Status := No_Status;
begin
   Write_Reg (Command_Reg, UEb);
   Write_Reg (Low_Param_Reg,
              Register_Data (Object.PO_Id));

   --  wait til we get command done
   Stat := Status (Read_Reg (Stat_Reg));
   while (Stat = No_Status) loop
      Stat := Status (Read_Reg (Stat_Reg));
   end loop;
end Write_UEb_Cmd;

procedure Write_UEb_Cmd_Single_Entry
  (Object :
     System.Tasking.Protected_Objects.Single_Entry.Protection_Entry_Access)
is
   Stat : Status := No_Status;
begin
   Write_Reg (Command_Reg, UEb);
   Write_Reg (Low_Param_Reg,
              Register_Data (Object.PO_Id));

   --  wait til we get command done
   Stat := Status (Read_Reg (Stat_Reg));
   while (Stat = No_Status) loop
      Stat := Status (Read_Reg (Stat_Reg));
   end loop;
end Write_UEb_Cmd_Single_Entry;

procedure Write_UEx_Cmd
```

```
  (Object :
     System.Tasking.Protected_Objects.Entries.Protection_Entries_Access) is
   Stat : Status := No_Status;
begin
   Write_Reg (Command_Reg, UEx);
   Write_Reg (Low_Param_Reg,
              Register_Data (Object.PO_Id));

   --  wait til we get command done
   Stat := Status (Read_Reg (Stat_Reg));
   while (Stat = No_Status) loop
      Stat := Status (Read_Reg (Stat_Reg));
   end loop;
end Write_UEx_Cmd;

procedure Write_UEx_Cmd_Single_Entry
  (Object :
     System.Tasking.Protected_Objects.Single_Entry.Protection_Entry_Access)
is
   Stat : Status := No_Status;
begin
   Write_Reg (Command_Reg, UEx);
   Write_Reg (Low_Param_Reg,
              Register_Data (Object.PO_Id));

   --  wait til we get command done
   Stat := Status (Read_Reg (Stat_Reg));
   while (Stat = No_Status) loop
      Stat := Status (Read_Reg (Stat_Reg));
   end loop;
end Write_UEx_Cmd_Single_Entry;

procedure Write_Ex_Cmd
  (Object :
     System.Tasking.Protected_Objects.Entries.Protection_Entries_Access) is
   Stat : Status := No_Status;
begin
   Write_Reg (Command_Reg, Ex);
   Write_Reg (Low_Param_Reg,
              Register_Data (Object.PO_Id));

   --  wait til we get command done
   Stat := Status (Read_Reg (Stat_Reg));
   while (Stat = No_Status) loop
      Stat := Status (Read_Reg (Stat_Reg));
   end loop;
end Write_Ex_Cmd;

procedure Write_Ex_Cmd_Single_Entry
  (Object :
     System.Tasking.Protected_Objects.Single_Entry.Protection_Entry_Access)
is
   Stat : Status := No_Status;
begin
   Write_Reg (Command_Reg, Ex);
   Write_Reg (Low_Param_Reg,
              Register_Data (Object.PO_Id));

   --  wait til we get command done
   Stat := Status (Read_Reg (Stat_Reg));
```

```
   while (Stat = No_Status) loop
      Stat := Status (Read_Reg (Stat_Reg));
   end loop;
end Write_Ex_Cmd_Single_Entry;

procedure Write_UEe_Cmd
  (Object :
     System.Tasking.Protected_Objects.Entries.Protection_Entries_Access) is
   Stat : Status := No_Status;
begin
   Write_Reg (Command_Reg, UEe);
   Write_Reg (Low_Param_Reg,
              Register_Data (Object.PO_Id));

   --  wait til we get command done
   Stat := Status (Read_Reg (Stat_Reg));
   while (Stat = No_Status) loop
      Stat := Status (Read_Reg (Stat_Reg));
   end loop;
end Write_UEe_Cmd;

procedure Write_UEe_Cmd_Single_Entry
  (Object :
     System.Tasking.Protected_Objects.Single_Entry.Protection_Entry_Access)
is
   Stat : Status := No_Status;
begin
   Write_Reg (Command_Reg, UEe);
   Write_Reg (Low_Param_Reg,
              Register_Data (Object.PO_Id));

   --  wait til we get command done
   Stat := Status (Read_Reg (Stat_Reg));
   while (Stat = No_Status) loop
      Stat := Status (Read_Reg (Stat_Reg));
   end loop;
end Write_UEe_Cmd_Single_Entry;

procedure Write_FPs_Cmd
  (Object :
     System.Tasking.Protected_Objects.Entries.Protection_Entries_Access)
is
   Stat : Status := No_Status;
begin
   Write_Reg (Command_Reg, FPs);
   Write_Reg (Low_Param_Reg,
              Es_FPs_Cmd_Data
              (Interfaces.Unsigned_32 (Object.PO_Id)
               , Interfaces.Unsigned_32 (Object.Ceiling)));

   --  wait til we get command done
   Stat := Status (Read_Reg (Stat_Reg));
   while (Stat = No_Status) loop
      Stat := Status (Read_Reg (Stat_Reg));
   end loop;
end Write_FPs_Cmd;

procedure Write_FPs_Cmd_Single_Entry
  (Object :
     System.Tasking.Protected_Objects.Single_Entry.Protection_Entry_Access)
```

```
is
   Stat : Status := No_Status;
begin
   Write_Reg (Command_Reg, FPs);
   Write_Reg (Low_Param_Reg,
              Es_FPs_Cmd_Data
              (Interfaces.Unsigned_32 (Object.PO_Id)
               , Interfaces.Unsigned_32 (Object.Ceiling)));

   --  wait til we get command done
   Stat := Status (Read_Reg (Stat_Reg));
   while (Stat = No_Status) loop
      Stat := Status (Read_Reg (Stat_Reg));
   end loop;
end Write_FPs_Cmd_Single_Entry;

procedure Write_FPs_Cmd_No_Entry
  (Object :
     System.Tasking.Protected_Objects.Protection_Access)
is
   Stat : Status := No_Status;
begin
   Write_Reg (Command_Reg, FPs);
   Write_Reg (Low_Param_Reg,
              Es_FPs_Cmd_Data
              (Interfaces.Unsigned_32 (Object.PO_Id)
               , Interfaces.Unsigned_32 (Object.Ceiling)));

   --  wait til we get command done
   Stat := Status (Read_Reg (Stat_Reg));
   while (Stat = No_Status) loop
      Stat := Status (Read_Reg (Stat_Reg));
   end loop;
end Write_FPs_Cmd_No_Entry;

procedure Write_UPe_Cmd
  (Object :
     System.Tasking.Protected_Objects.Entries.Protection_Entries_Access)
is
   Stat : Status := No_Status;
begin
   Write_Reg (Command_Reg, UPe);
   Write_Reg (Low_Param_Reg,
              Register_Data (Object.PO_Id));

   --  wait til we get command done
   Stat := Status (Read_Reg (Stat_Reg));
   while (Stat = No_Status) loop
      Stat := Status (Read_Reg (Stat_Reg));
   end loop;
end Write_UPe_Cmd;

procedure Write_UPe_Cmd_Single_Entry
  (Object :
     System.Tasking.Protected_Objects.Single_Entry.Protection_Entry_Access)
is
   Stat : Status := No_Status;
begin
   Write_Reg (Command_Reg, UPe);
   Write_Reg (Low_Param_Reg,
```

```
                Register_Data (Object.PO_Id));

   --  wait til we get command done
   Stat := Status (Read_Reg (Stat_Reg));
   while (Stat = No_Status) loop
      Stat := Status (Read_Reg (Stat_Reg));
   end loop;
end Write_UPe_Cmd_Single_Entry;

procedure Write_UPe_Cmd_No_Entry
  (Object :
     System.Tasking.Protected_Objects.Protection_Access)
is
   Stat : Status := No_Status;
begin
   Write_Reg (Command_Reg, UPe);
   Write_Reg (Low_Param_Reg,
              Register_Data (Object.PO_Id));

   --  wait til we get command done
   Stat := Status (Read_Reg (Stat_Reg));
   while (Stat = No_Status) loop
      Stat := Status (Read_Reg (Stat_Reg));
   end loop;
end Write_UPe_Cmd_No_Entry;

procedure Write_UFe_Cmd
  (Object :
     System.Tasking.Protected_Objects.Entries.Protection_Entries_Access)
is
   Stat : Status := No_Status;
begin
   Write_Reg (Command_Reg, UFe);
   Write_Reg (Low_Param_Reg,
              Register_Data (Object.PO_Id));

   --  wait til we get command done
   Stat := Status (Read_Reg (Stat_Reg));
   while (Stat = No_Status) loop
      Stat := Status (Read_Reg (Stat_Reg));
   end loop;
end Write_UFe_Cmd;

procedure Write_UFe_Cmd_Single_Entry
  (Object :
     System.Tasking.Protected_Objects.Single_Entry.Protection_Entry_Access)
is
   Stat : Status := No_Status;
begin
   Write_Reg (Command_Reg, UFe);
   Write_Reg (Low_Param_Reg,
              Register_Data (Object.PO_Id));

   --  wait til we get command done
   Stat := Status (Read_Reg (Stat_Reg));
   while (Stat = No_Status) loop
      Stat := Status (Read_Reg (Stat_Reg));
   end loop;
end Write_UFe_Cmd_Single_Entry;
```

```
procedure Write_UFe_Cmd_No_Entry
  (Object :
     System.Tasking.Protected_Objects.Protection_Access)
is
   Stat : Status := No_Status;
begin
   Write_Reg (Command_Reg, UFe);
   Write_Reg (Low_Param_Reg,
              Register_Data (Object.PO_Id));

   --  wait til we get command done
   Stat := Status (Read_Reg (Stat_Reg));
   while (Stat = No_Status) loop
      Stat := Status (Read_Reg (Stat_Reg));
   end loop;
end Write_UFe_Cmd_No_Entry;

procedure Write_UFPx_Cmd
  (Object :
     System.Tasking.Protected_Objects.Entries.Protection_Entries_Access)
is
   Stat : Status := No_Status;
begin
   Write_Reg (Command_Reg, UFPx);
   Write_Reg (Low_Param_Reg,
              Register_Data (Object.PO_Id));

   --  wait til we get command done
   Stat := Status (Read_Reg (Stat_Reg));
   while (Stat = No_Status) loop
      Stat := Status (Read_Reg (Stat_Reg));
   end loop;
end Write_UFPx_Cmd;

procedure Write_UFPx_Cmd_Single_Entry
  (Object :
     System.Tasking.Protected_Objects.Single_Entry.Protection_Entry_Access)
is
   Stat : Status := No_Status;
begin
   Write_Reg (Command_Reg, UFPx);
   Write_Reg (Low_Param_Reg,
              Register_Data (Object.PO_Id));

   --  wait til we get command done
   Stat := Status (Read_Reg (Stat_Reg));
   while (Stat = No_Status) loop
      Stat := Status (Read_Reg (Stat_Reg));
   end loop;
end Write_UFPx_Cmd_Single_Entry;

procedure Write_UFPx_Cmd_No_Entry
  (Object :
     System.Tasking.Protected_Objects.Protection_Access)
is
   Stat : Status := No_Status;
begin
   Write_Reg (Command_Reg, UFPx);
   Write_Reg (Low_Param_Reg,
              Register_Data (Object.PO_Id));
```

```
   --  wait til we get command done
   Stat := Status (Read_Reg (Stat_Reg));
   while (Stat = No_Status) loop
      Stat := Status (Read_Reg (Stat_Reg));
   end loop;
end Write_UFPx_Cmd_No_Entry;

procedure Write_UPxe_Cmd
  (Object :
     System.Tasking.Protected_Objects.Entries.Protection_Entries_Access)
is
   Stat : Status := No_Status;
begin
   Write_Reg (Command_Reg, UPxe);
   Write_Reg (Low_Param_Reg,
              Register_Data (Object.PO_Id));

   --  wait til we get command done
   Stat := Status (Read_Reg (Stat_Reg));
   while (Stat = No_Status) loop
      Stat := Status (Read_Reg (Stat_Reg));
   end loop;
end Write_UPxe_Cmd;

procedure Write_UPxe_Cmd_Single_Entry
  (Object :
     System.Tasking.Protected_Objects.Single_Entry.Protection_Entry_Access)
is
   Stat : Status := No_Status;
begin
   Write_Reg (Command_Reg, UPxe);
   Write_Reg (Low_Param_Reg,
              Register_Data (Object.PO_Id));

   --  wait til we get command done
   Stat := Status (Read_Reg (Stat_Reg));
   while (Stat = No_Status) loop
      Stat := Status (Read_Reg (Stat_Reg));
   end loop;
end Write_UPxe_Cmd_Single_Entry;

procedure Write_UPxe_Cmd_No_Entry
  (Object :
     System.Tasking.Protected_Objects.Protection_Access)
is
   Stat : Status := No_Status;
begin
   Write_Reg (Command_Reg, UPxe);
   Write_Reg (Low_Param_Reg,
              Register_Data (Object.PO_Id));

   --  wait til we get command done
   Stat := Status (Read_Reg (Stat_Reg));
   while (Stat = No_Status) loop
      Stat := Status (Read_Reg (Stat_Reg));
   end loop;
end Write_UPxe_Cmd_No_Entry;

procedure Write_UFxe_Cmd
```

```
  (Object :
     System.Tasking.Protected_Objects.Entries.Protection_Entries_Access)
is
   Stat : Status := No_Status;
begin
   Write_Reg (Command_Reg, UFxe);
   Write_Reg (Low_Param_Reg,
              Register_Data (Object.PO_Id));

   --  wait til we get command done
   Stat := Status (Read_Reg (Stat_Reg));
   while (Stat = No_Status) loop
      Stat := Status (Read_Reg (Stat_Reg));
   end loop;
end Write_UFxe_Cmd;

procedure Write_UFxe_Cmd_Single_Entry
  (Object :
     System.Tasking.Protected_Objects.Single_Entry.Protection_Entry_Access)
is
   Stat : Status := No_Status;
begin
   Write_Reg (Command_Reg, UFxe);
   Write_Reg (Low_Param_Reg,
              Register_Data (Object.PO_Id));

   --  wait til we get command done
   Stat := Status (Read_Reg (Stat_Reg));
   while (Stat = No_Status) loop
      Stat := Status (Read_Reg (Stat_Reg));
   end loop;
end Write_UFxe_Cmd_Single_Entry;

procedure Write_UFxe_Cmd_No_Entry
  (Object :
     System.Tasking.Protected_Objects.Protection_Access)
is
   Stat : Status := No_Status;
begin
   Write_Reg (Command_Reg, UFxe);
   Write_Reg (Low_Param_Reg,
              Register_Data (Object.PO_Id));

   --  wait til we get command done
   Stat := Status (Read_Reg (Stat_Reg));
   while (Stat = No_Status) loop
      Stat := Status (Read_Reg (Stat_Reg));
   end loop;
end Write_UFxe_Cmd_No_Entry;

procedure Write_FPx_Cmd
  (Object :
     System.Tasking.Protected_Objects.Entries.Protection_Entries_Access)
is
   Stat : Status := No_Status;
begin
   Write_Reg (Command_Reg, FPx);
   Write_Reg (Low_Param_Reg,
              Register_Data (Object.PO_Id));
```

```
   --  wait til we get command done
   Stat := Status (Read_Reg (Stat_Reg));
   while (Stat = No_Status) loop
      Stat := Status (Read_Reg (Stat_Reg));
   end loop;
end Write_FPx_Cmd;

procedure Write_FPx_Cmd_Single_Entry
  (Object :
     System.Tasking.Protected_Objects.Single_Entry.Protection_Entry_Access)
is
   Stat : Status := No_Status;
begin
   Write_Reg (Command_Reg, FPx);
   Write_Reg (Low_Param_Reg,
              Register_Data (Object.PO_Id));

   --  wait til we get command done
   Stat := Status (Read_Reg (Stat_Reg));
   while (Stat = No_Status) loop
      Stat := Status (Read_Reg (Stat_Reg));
   end loop;
end Write_FPx_Cmd_Single_Entry;

procedure Write_FPx_Cmd_No_Entry
  (Object :
     System.Tasking.Protected_Objects.Protection_Access)
is
   Stat : Status := No_Status;
begin
   Write_Reg (Command_Reg, FPx);
   Write_Reg (Low_Param_Reg,
              Register_Data (Object.PO_Id));

   --  wait til we get command done
   Stat := Status (Read_Reg (Stat_Reg));
   while (Stat = No_Status) loop
      Stat := Status (Read_Reg (Stat_Reg));
   end loop;
end Write_FPx_Cmd_No_Entry;

procedure Write_FPe_Cmd
  (Object :
     System.Tasking.Protected_Objects.Entries.Protection_Entries_Access)
is
   Stat : Status := No_Status;
begin
   Write_Reg (Command_Reg, FPe);
   Write_Reg (Low_Param_Reg,
              Register_Data (Object.PO_Id));

   --  wait til we get command done
   Stat := Status (Read_Reg (Stat_Reg));
   while (Stat = No_Status) loop
      Stat := Status (Read_Reg (Stat_Reg));
   end loop;
end Write_FPe_Cmd;

procedure Write_FPe_Cmd_Single_Entry
  (Object :
```

```
     System.Tasking.Protected_Objects.Single_Entry.Protection_Entry_Access)
is
   Stat : Status := No_Status;
begin
   Write_Reg (Command_Reg, FPe);
   Write_Reg (Low_Param_Reg,
              Register_Data (Object.PO_Id));

   --  wait til we get command done
   Stat := Status (Read_Reg (Stat_Reg));
   while (Stat = No_Status) loop
      Stat := Status (Read_Reg (Stat_Reg));
   end loop;
end Write_FPe_Cmd_Single_Entry;

procedure Write_FPe_Cmd_No_Entry
  (Object :
     System.Tasking.Protected_Objects.Protection_Access)
is
   Stat : Status := No_Status;
begin
   Write_Reg (Command_Reg, FPe);
   Write_Reg (Low_Param_Reg,
              Register_Data (Object.PO_Id));

   --  wait til we get command done
   Stat := Status (Read_Reg (Stat_Reg));
   while (Stat = No_Status) loop
      Stat := Status (Read_Reg (Stat_Reg));
   end loop;
end Write_FPe_Cmd_No_Entry;

procedure Write_Entry_Barrier (Barrier : Boolean) is
   Stat : Status := No_Status;
begin
   if Barrier then
      Write_Reg (Low_Param_Reg, Register_Data (1));
   else
      Write_Reg (Low_Param_Reg, Register_Data (0));
   end if;
   --  wait til we get command done
   Stat := Status (Read_Reg (Stat_Reg));
   while (Stat = No_Status) loop
      Stat := Status (Read_Reg (Stat_Reg));
   end loop;
end Write_Entry_Barrier;

function Read_Reg (Reg : in Register) return Register_Data is
   R : Register_Data;
begin
   System.Machine_Code.Asm ("eieio");
   R := Register_Data (Reg);
   System.Machine_Code.Asm ("eieio");
   return R;
end Read_Reg;

function Get_PO_Serial_Number return Integer is
   Result : Integer;
begin
   Result := Next_PO_Serial_Number;
```

```
      Next_PO_Serial_Number := Next_PO_Serial_Number + 1;
      return Result;
   end Get_PO_Serial_Number;

end System.FPGA_Kernel_Interface;
```

Appendix B

Manual Examination of pGNAT

To manually examine pGNAT's correctness, the –gnatG switch is used to compile an application in order to produce a readable output. We can inspect the output to see if the communications between RavenHaRT and the PowerPC are established correctly. These communications are subprogram calls to the System.FPGA_Kernel_Interface, thus we look for statements referring to System.FPGA_Kernel_Interface. Both the source code and the compilation output of the Generic Navigation System (nay_sys.adb) are shown below.

The source code for nav_sys.adb

```
-- ********************************************
-- Ravenscar Generic NAV System December 2003 design
-- Author:  Sébastien Gorelov
-- Advisor: Kristina Lundqvist
-- Program name: NAV_SYS.adb
-- First Created: December 12th 2003
-- Last modified: January 08th 2004
-- Change History: This design differs from the
--                 preliminary Nav System:
--                 1. The High Rate Task does not
--                 suspend anymore and the suspension
--                 object code has been removed.
--                 2. The High Rate Task does not
--                 verify the IMU buffer a second
--                 time in the same loop.
--                 (12-12-2003).
--                 Correct Bug: Add an unset_flag
--                 command just after the reset
--                 expiry date command.
--                 (15-12-2003).
-- Issues: Q.What happens when the clock overflows?
--         Should we be concerned with that?
--         A.The clock system cannot overflow before
--         July 16th, 2554, at 23:34:33.709551616 (EST).
-- Embedded Systems Lab
-- Massachusetts Institute of Technology
-- References:
-- (1).
-- Guide for the use of the Ada Ravenscar Profile
-- in high integrity systems
-- A. Burns, B. Dobbing, T. Vardanega
-- University of York January 2003
-- Technical Report YCS-2003-348
-- ********************************************
-- N O T E S:
-- This is the main system procedure.
```

```
-- It consists of three protected objects (P.O.)
-- and five tasks:
-- a. Epoch P.O.

-- b. Urgent P.O.

-- c. Event P.O.

-- d. External Trigger Task 1. Priority: Highest (is 4).

-- e. External Trigger Task 2. Priority: Highest. (is 4).

-- f. GPS Receiver. Priority: High. (is 3).

-- g. Low Rate Nav Task. Priority: Lowest. (is 1).

-- h. High Rate Nav Task. Priority: Low. (is 2).

pragma Restrictions(No_Entry_Queue);
pragma Restrictions(No_Abort_Statements);
with Ada.Real_Time;
with system;
with Rand;
with Gen_Buff;
with Ada.Text_Io;
with Ada.Integer_Text_Io;
use Ada.Real_Time;
use Ada.Text_Io;

procedure Nav_Sys is

-- Buffer Instantiation -------------------------
   package Kf_Nav_Buff is new Gen_Buff;
   package Imu_Buff is new Gen_Buff;
   package Los_Buff is new Gen_Buff;

-- GPS Receive event type declaration -----------
   type gps_message_datatype is range -100..100;

-- buffer and other variable types declarations -
   Kf_Data  : Kf_Nav_Buff.Data_Format;
   Imu_Data : Imu_Buff.Data_Format;
   Los_Data : Los_Buff.Data_Format;
   gps_data : gps_message_datatype;

-- Epoch P.O. -----------------------------------
--    This protected object enables the coordination
--    of tasks' start times. The P.O. reads the clock
--    on creation and then makes the clock value available
--    to all tasks. This code was written in (1).

   protected Epoch is
      procedure Get_Start_Time (
            T :    out Ada.Real_Time.Time );
   private
      pragma Priority(System.Priority'Last);
      Start : Ada.Real_Time.Time;
      First : Boolean := True;
   end Epoch;
```

```
   protected body Epoch is
      procedure Get_Start_Time (
            T :    out Ada.Real_Time.Time ) is
      begin
         if First then
            First := False;
            Start := Ada.Real_Time.Clock;
         end if;
         T := Start;
      end Get_Start_Time;
   end Epoch;
---------------------------------- Epoch P.O. --

-- Urgent P.O. ---------------------------------
--    This protected object serves as a flag that is
--    set by the High Rate task and read by the Low Rate
--    task. It contains the procedures check_timeout (which
--    sets the flag in case of time-out), unset flag,
--    reset timer, and the function check_flag_set, which
--    returns the flag status.

    protected Urgent is
      procedure check_timeout;
      procedure Unset_Flag;
      procedure reset_xpry_date;
      -- if check_flag_set is true then the urgent signal was set
      function check_flag_set return boolean;
   private
      Signal : Boolean := False;
      Expiry_Date :          Ada.Real_Time.Time;
   end Urgent;
   -- protected object body: urgent signal
   protected body Urgent is
      -- procedure check_timeout
      procedure check_timeout is
      begin
         if (ada.Real_Time.Clock >= Expiry_Date) then
            Signal := True;
         end if;
      end check_timeout;
      -- procedure Unset_Flag
      procedure Unset_Flag is
      begin
            Signal   := False;
      end Unset_Flag;
      -- procedure reset_xpry_date
      procedure Reset_Xpry_Date is
      begin
         Expiry_Date := Ada.Real_Time.Clock + Ada.Real_Time.Milliseconds(19_000);
      end Reset_Xpry_Date;
      -- function check_flag
      function Check_Flag_set return boolean is
      begin
         return Signal;
      end Check_Flag_set;
   end Urgent;
---------------------------------- Urgent P.O. --

-- Event P.O. ----------------------------------
--    This protected object is an event that is triggered
```

```
--    by the external triger task 1 and handled by the
--    GPS receive task. It is because the GPS receive task
--    is not cyclic but event-based that we employ this
--    scheme, which is inspired from (1).

protected type Event_Object(Ceiling : System.Priority) is
      entry Wait (
            D :    out gps_message_datatype );
      procedure Signal (
            D : in     gps_message_datatype );
   private
      pragma Priority(Ceiling);
      Current : gps_message_datatype;
      Signalled : Boolean := False;
   end Event_Object;
   -- event body
   protected body Event_Object is
      -- wait entry declaration
      entry Wait(D: out gps_message_datatype) when Signalled is
      begin
         D := Current;
         Signalled := False;
      end Wait;
      -- signal procedure declaration
      procedure Signal(D: in gps_message_datatype) is
      begin
         Current := D;
         Signalled := True;
      end Signal;
   end Event_Object;
---------------------------------- Event P.O. --

-- Event P.O. instantiation ---------------------
   GPS_RCV_EVENT : Event_Object(4);

-- External Trigger task variable declarations --
   -- random variable in the range -100..100
   Pouf1, Pouf2       : Rand.Rand_Time;
   -- variable to transform the random time into a proper offset
   rand_offset1, rand_offset2 : integer;

-- External Trigger Task 1 ----------------------
--    This "cyclic" task executes with random periods.
--    Its purpose is to simulate the random arrival of GPS
--    messages. The data it passes is the random number
--    generated. It passes them to the Event P.O.
--    This task is guaranteed to have a period between
--    18001 and 20001 msec, with a 1 msec resolution.

   task type phantom1(Pri : System.Priority; Offset : Natural) is
      pragma Priority(Pri);
   end Phantom1;
-- task body
   task body Phantom1 is
      Next_Period :          Ada.Real_Time.Time;
      begin
         Epoch.Get_Start_Time(Next_Period);
         Pouf1 := Rand.Get_Rand_Time;
         rand_offset1 := 5001 + 10 * integer(Pouf1);
```

```
         Next_Period := Next_Period + Ada.Real_Time.Milliseconds(Offset) +
Ada.Real_Time.Milliseconds(rand_offset1);
      loop
         delay until Next_Period;
           -- pass the rand_offset to GPS_RCV_EVENT
           Gps_Rcv_Event.Signal(Gps_Message_Datatype(Pouf1));

           -- schedule the next wake-up
           Pouf1 := Rand.Get_Rand_Time;
           rand_offset1 := 19001 + 10 * integer(Pouf1);
           Next_Period := Next_Period + Ada.Real_Time.Milliseconds(rand_offset1);
      end loop;
   end Phantom1;
---------------------- External Trigger Task 1 --

-- External Trigger Task 1 instantiation --------
   External_Trigger_Task_1 : Phantom1(4,0);

-- External Trigger Task 2 ---------------------
--    This task behaves similarly to the External Trigger Task 1
--    except for the period, which is now between 1 and
--    2001 msec, same resolution, and the destination of
--    the data, which is now the IMU_BUFF buffer.
   task type phantom2(Pri : System.Priority; Offset : Natural) is
      pragma Priority(Pri);
   end Phantom2;
-- task body
   task body Phantom2 is
      Next_Period :          Ada.Real_Time.Time;
      dont_care : boolean;
      begin
         Epoch.Get_Start_Time(Next_Period);
         Pouf2 := Rand.Get_Rand_Time;
         rand_offset2 := 1001 + 10 * integer(Pouf2);
         Next_Period := Next_Period + Ada.Real_Time.Milliseconds(Offset) +
Ada.Real_Time.Milliseconds(rand_offset2);
      loop
         delay until Next_Period;
           -- pass the rand_offset2 to IMU_BUFF buffer
           Imu_Buff.Place_Item(Imu_Buff.Data_Format(Pouf2), Dont_Care);

           -- schedule the next wake-up
           Pouf2 := Rand.Get_Rand_Time;
           rand_offset2 := 5001 + 10 * integer(Pouf2);
           Next_Period := Next_Period + Ada.Real_Time.Milliseconds(rand_offset2);
      end loop;
   end Phantom2;
---------------------- External Trigger Task 2 --

-- External Trigger Task 2 instantiation --------
   External_Trigger_Task_2 : Phantom2(4,0);

-- GPS Receiver ---------------------------------
--    This task is event-triggered and suspends on the
--    protected entry of P.O. event. This code is inspired
--    from (1). It reads the data issued by the External
--    Trigger Task 1. The data processing is simulated.
--    The result is passed to the LOS_BUFF buffer.

   task GPS_receiver is
```

```
      pragma Priority(3);
   end GPS_receiver;

   task body GPS_receiver is
      dont_care : boolean;
   begin
      loop
         GPS_RCV_EVENT.Wait(gps_data);
         -- GPS message processing goes here

         los_data := Los_Buff.Data_Format(gps_data);
         LOS_BUFF.Place_Item(los_data, dont_care);
      end loop;
   end GPS_receiver;
-------------------------------- GPS Receiver --

-- Low Rate Nav Task ---------------------------
--    This is a cyclic task with period 5000 msec.
--    It simulates a Kalman Filter. We model it in the
--    following fashion: it checks and inputs data from the
--    LOS_BUFF buffer.
--    If there is data in the buffer, the
--    processing is simulated and the output is placed in
--    the KF_NAV_BUFF buffer. We assume that overwritting data
--    is acceptable. Action is then taken to clear the flag
--    that was potentially raised since the last time the
--    High Rate Nav task emptied the buffer.
--    If there is no data in the LOS_BUFF buffer, the urgent
--    flag status is checked. If it is set, the task produced
--    an estimate and places it in the KF_NAV_BUFF buffer.
--    The flag is then unset.

   task type Lo(Cycle_Time : Positive; Pri : System.Priority; Offset : Natural)
is
      pragma Priority(Pri);
   end Lo;
   -- task body
   task body Lo is
      dont_care   :          Boolean;
--      Urge        :          Boolean;
      Success     :          Boolean;
      Next_Period :          Ada.Real_Time.Time;
      Period      : constant Ada.Real_Time.Time_Span :=
Ada.Real_Time.Milliseconds (Cycle_Time);

   begin
      -- here we start both processes using the offsets.
      Epoch.Get_Start_Time(Next_Period);
      Next_Period := Next_Period + Ada.Real_Time.Milliseconds(Offset);
      loop
         delay until Next_Period;
         Los_Buff.Extract_Item(Los_Data, Success);
         if Success then
            -- Kalman Filtering process goes here
            Kf_Data := Kf_Nav_Buff.Data_Format(Los_Data);
            KF_NAV_BUFF.Place_Item(Kf_Data,dont_care);
            Urgent.Unset_Flag;
         else
            if Urgent.Check_Flag_Set then  -- we check the urge signal
            -- fill the buffer and warn high rate that the data is delivered
```

```
            -- this simulates the KF estimation:
            KF_NAV_BUFF.Place_Item(50,dont_care);
            urgent.Unset_Flag;
            end if;
         end if;
         -- schedule the next wake-up
         Next_Period := Next_Period + Period;
      end loop;
   end Lo;
---------------------------- Low Rate Nav Task --

-- Low Rate Nav Task instantiation -------------
   Kalman_Filter : Lo (5000, 2, 0);

-- High Rate Nav Task --------------------------
--    This is a cyclic task with period 1000 msec.
--    It simulates a sequencer which takes input from two
--    buffers. We model it in the following fashion:
--    we start by extracting data from the IMU_BUFF
--    buffer, whether there is data or not. If there is,
--    it will then be processed. We then check if
--    there is a Urgent PO Time-Out, in which case the
--    the urgent flag is set within the Urgent PO.
--    Data is then extracted from the KF_NAV_BUFF buffer.
--    If there is data, then the Time-Out is rescheduled to
--    15 seconds later using reset_xpry_date in Urgent PO.
--    This 15 second period is the equivalent of three Low Rate Nav
--    task periods.

   task type Hi(Cycle_Time : Positive; Pri : System.Priority; Offset : Natural)
is
      pragma Priority(Pri);
   end Hi;
   task body Hi is
      Success1, Success2     :          Boolean; -- if false, it means buffer
empty
      Next_Period :          Ada.Real_Time.Time;

      Period      : constant Ada.Real_Time.Time_Span :=
Ada.Real_Time.Milliseconds (Cycle_Time);

   begin
      -- here we start both processes using the offsets.
      Epoch.Get_Start_Time(Next_Period);
      Next_Period := Next_Period + Ada.Real_Time.Milliseconds(Offset);
      Urgent.Reset_Xpry_Date;
      loop
         delay until Next_Period;
         Imu_Buff.Extract_Item(Imu_Data,Success1);
            if Success1 then
               Put("IMU data is being processed");
               New_Line;
            end if;
         Urgent.Check_Timeout;
         if Urgent.Check_Flag_Set then
            Put("Time's up.");
            New_Line;
         end if;
         -- extract from the buffer
         KF_NAV_BUFF.Extract_Item(Kf_Data, Success2);
```

```
         if Success2 then
            Put("Hi: success, we reset the expiry date and the data is ");
            Ada.Integer_Text_Io.Put(Integer(Kf_Data));      -- data was in the
buffer
            New_Line;                                       -- we print it
            Urgent.Reset_Xpry_Date;                         -- and reschedule the
expiry_date
            Urgent.Unset_Flag;
         else
            Put("Hi: no data in KF buffer");
            New_Line;
             -- data was not in the buffer
         end if;

-- schedule the next wake-up
            Next_Period := Next_Period + Period;
      end loop;
   end Hi;
--------------------------- High Rate Nav Task --

-- High Rate Nav Task Instantiation ------------
   Sequencer     : Hi (1000, 1, 0);

-- NAV_SYS body execution
begin
   Put("Main procedure body execution");
   New_Line;
end NAV_SYS;
```

The generated expanded output for nav_sys.adb

```
with ada;
with ada;
with ada;
pragma restrictions (no_entry_queue);
pragma restrictions (no_abort_statements);
with ada.ada__real_time;
with system;
with rand;
with gen_buff;
with ada.ada__text_io;
with ada.ada__integer_text_io;
use ada.ada__real_time;
use ada.ada__text_io;
with system.system__tasking.system__tasking__protected_objects;
with system.system__fpga_kernel_interface;
with system.system__soft_links;
with ada.ada__exceptions;
with system.system__tasking;
with system.system__tasking.system__tasking__protected_objects.
  system__tasking__protected_objects__entries;
with system;
with system.system__finalization_implementation;
with system.system__finalization_root;
with system.system__tasking.system__tasking__protected_objects.
  system__tasking__protected_objects__operations;
with system.system__storage_elements;
with system.system__standard_library;
with system.system__parameters;
with system.system__task_info;
with system.system__tasking.system__tasking__stages;
with ada.ada__real_time.ada__real_time__delays;

procedure nav_sys is
   F74b : system__finalization_root__finalizable_ptr := null;

   procedure nav_sys___clean is
   begin
      system__soft_links__abort_defer.all;
      system__finalization_implementation__finalize_list (F74b);
      system__soft_links__abort_undefer.all;
      return;
   end nav_sys___clean;
begin
   _chain : aliased system__tasking__activation_chain;
   system__tasking__activation_chainIP (_chain);

   package kf_nav_buff is
      package kf_nav_buff__gen_buff renames kf_nav_buff;
      package kf_nav_buff__gen_buffGH renames kf_nav_buff__gen_buff;
      type nav_sys__kf_nav_buff__data_format is range -100 .. 100;
      procedure nav_sys__kf_nav_buff__place_item (item :
        nav_sys__kf_nav_buff__data_format; overwrote : out boolean);
      procedure nav_sys__kf_nav_buff__extract_item (item : out
        nav_sys__kf_nav_buff__data_format; success : out boolean);
   end kf_nav_buff;

   package body kf_nav_buff is
```

```
protected type nav_sys__kf_nav_buff__buffT is
   procedure nav_sys__kf_nav_buff__buffT__place (item : in
     nav_sys__kf_nav_buff__data_format; success : out boolean);
   procedure nav_sys__kf_nav_buff__buffT__extract (item : out
     nav_sys__kf_nav_buff__data_format; success : out boolean);
private
   nav_sys__kf_nav_buff__buffT__buffer_full : boolean := false;
   nav_sys__kf_nav_buff__buffT__buffer_empty : boolean := true;
   nav_sys__kf_nav_buff__buffT__data :
     nav_sys__kf_nav_buff__data_format;
end nav_sys__kf_nav_buff__buffT;
type nav_sys__kf_nav_buff__buffTV is limited record
   buffer_full : boolean := false;
   buffer_empty : boolean := true;
   data : nav_sys__kf_nav_buff__data_format;
   _object : aliased
     system__tasking__protected_objects__protection;
end record;
procedure nav_sys__kf_nav_buff__buffPT__placeN (_object : in out
  nav_sys__kf_nav_buff__buffTV; item : in
  nav_sys__kf_nav_buff__data_format; success : out boolean);
procedure nav_sys__kf_nav_buff__buffPT__placeP (_object : in out
  nav_sys__kf_nav_buff__buffTV; item : in
  nav_sys__kf_nav_buff__data_format; success : out boolean);
procedure nav_sys__kf_nav_buff__buffPT__extractN (_object : in
  out nav_sys__kf_nav_buff__buffTV; item : out
  nav_sys__kf_nav_buff__data_format; success : out boolean);
procedure nav_sys__kf_nav_buff__buffPT__extractP (_object : in
  out nav_sys__kf_nav_buff__buffTV; item : out
  nav_sys__kf_nav_buff__data_format; success : out boolean);
freeze nav_sys__kf_nav_buff__buffTV [
   procedure nav_sys__kf_nav_buff__buffTVIP (_init : in out
     nav_sys__kf_nav_buff__buffTV) is
   begin
      _init.buffer_full := false;
      _init.buffer_empty := true;
      system__tasking__protected_objects__initialize_protection (
        _init._object'unchecked_access,
        system__tasking__unspecified_priority);
      return;
   end nav_sys__kf_nav_buff__buffTVIP;
]
kf_nav_buff__buff : nav_sys__kf_nav_buff__buffT;
nav_sys__kf_nav_buff__buffTVIP (nav_sys__kf_nav_buff__buffTV!(
  kf_nav_buff__buff));

procedure nav_sys__kf_nav_buff__place_item (item :
  nav_sys__kf_nav_buff__data_format; overwrote : out boolean) is
   ok : boolean;
begin
   nav_sys__kf_nav_buff__buffPT__placeP (
     nav_sys__kf_nav_buff__buffTV!(kf_nav_buff__buff), item, ok);
   overwrote := not ok;
   return;
end nav_sys__kf_nav_buff__place_item;

procedure nav_sys__kf_nav_buff__extract_item (item : out
  nav_sys__kf_nav_buff__data_format; success : out boolean) is
begin
   nav_sys__kf_nav_buff__buffPT__extractP (
```

```
      nav_sys__kf_nav_buff__buffTV!(kf_nav_buff__buff), item,
      success);
    if not success then
       item := 0;
    end if;
    return;
end nav_sys__kf_nav_buff__extract_item;


procedure nav_sys__kf_nav_buff__buffPT__placeN (_object : in out
  nav_sys__kf_nav_buff__buffTV; item : in
  nav_sys__kf_nav_buff__data_format; success : out boolean) is
    buffR : system__tasking__protected_objects__protection
      renames _object._object;
    dataP : nav_sys__kf_nav_buff__data_format renames _object.data;
    buffer_emptyP : boolean renames _object.buffer_empty;
    buffer_fullP : boolean renames _object.buffer_full;
begin
    success := not buffer_fullP;
    dataP := item;
    buffer_emptyP := false;
    buffer_fullP := true;
    return;
end nav_sys__kf_nav_buff__buffPT__placeN;

procedure nav_sys__kf_nav_buff__buffPT__extractN (_object : in
  out nav_sys__kf_nav_buff__buffTV; item : out
  nav_sys__kf_nav_buff__data_format; success : out boolean) is
    buffR : system__tasking__protected_objects__protection
      renames _object._object;
    dataP : nav_sys__kf_nav_buff__data_format renames _object.data;
    buffer_emptyP : boolean renames _object.buffer_empty;
    buffer_fullP : boolean renames _object.buffer_full;
begin
    success := not buffer_emptyP;
    if not buffer_emptyP then
       item := dataP;
       buffer_fullP := false;
       buffer_emptyP := true;
    end if;
    return;
end nav_sys__kf_nav_buff__buffPT__extractN;

procedure nav_sys__kf_nav_buff__buffPT__extractP (_object : in
  out nav_sys__kf_nav_buff__buffTV; item : out
  nav_sys__kf_nav_buff__data_format; success : out boolean) is

    procedure nav_sys__kf_nav_buff__buffPT__extractP___clean is
    begin
       system__tasking__protected_objects__unlock (_object._object'
         unchecked_access);
       system__soft_links__abort_undefer.all;
       return;
    end nav_sys__kf_nav_buff__buffPT__extractP___clean;
begin
    system__fpga_kernel_interface__write_fps_cmd_no_entry (_object.
      _object'unchecked_access);
    system__soft_links__abort_defer.all;
    system__tasking__protected_objects__lock (_object._object'
      unchecked_access);
```

```
   system__fpga_kernel_interface__write_upe_cmd_no_entry (_object.
     _object'unchecked_access);
   system__fpga_kernel_interface__write_fpe_cmd_no_entry (_object.
     _object'unchecked_access);
   B144b : begin
      nav_sys__kf_nav_buff__buffPT__extractN (_object, item,
        success);
   exception
      when all others =>
         B147b : declare
            E146b : ada__exceptions__exception_occurrence;
         begin
            ada__exceptions__save_occurrence (E146b,
              system__soft_links__get_current_excep.all.all);
            nav_sys__kf_nav_buff__buffPT__extractP___clean;
            ada__exceptions__reraise_occurrence_no_defer (E146b);
         end B147b;
   at end
      nav_sys__kf_nav_buff__buffPT__extractP___clean;
   end B144b;
   return;
end nav_sys__kf_nav_buff__buffPT__extractP;

procedure nav_sys__kf_nav_buff__buffPT__placeP (_object : in out
  nav_sys__kf_nav_buff__buffTV; item : in
  nav_sys__kf_nav_buff__data_format; success : out boolean) is

   procedure nav_sys__kf_nav_buff__buffPT__placeP___clean is
   begin
      system__tasking__protected_objects__unlock (_object._object'
        unchecked_access);
      system__soft_links__abort_undefer.all;
      return;
   end nav_sys__kf_nav_buff__buffPT__placeP___clean;
begin
   system__fpga_kernel_interface__write_fps_cmd_no_entry (_object.
     _object'unchecked_access);
   system__soft_links__abort_defer.all;
   system__tasking__protected_objects__lock (_object._object'
     unchecked_access);
   system__fpga_kernel_interface__write_upe_cmd_no_entry (_object.
     _object'unchecked_access);
   system__fpga_kernel_interface__write_fpe_cmd_no_entry (_object.
     _object'unchecked_access);
   B136b : begin
      nav_sys__kf_nav_buff__buffPT__placeN (_object, item,
        success);
   exception
      when all others =>
         B139b : declare
            E138b : ada__exceptions__exception_occurrence;
         begin
            ada__exceptions__save_occurrence (E138b,
              system__soft_links__get_current_excep.all.all);
            nav_sys__kf_nav_buff__buffPT__placeP___clean;
            ada__exceptions__reraise_occurrence_no_defer (E138b);
         end B139b;
   at end
      nav_sys__kf_nav_buff__buffPT__placeP___clean;
```

```
      end B136b;
      return;
   end nav_sys__kf_nav_buff__buffPT__placeP;
begin
   null;
end kf_nav_buff;

package kf_nav_buff is new gen_buff;

package imu_buff is
   package imu_buff__gen_buff renames imu_buff;
   package imu_buff__gen_buffGH renames imu_buff__gen_buff;
   type nav_sys__imu_buff__data_format is range -100 .. 100;
   procedure nav_sys__imu_buff__place_item (item :
     nav_sys__imu_buff__data_format; overwrote : out boolean);
   procedure nav_sys__imu_buff__extract_item (item : out
     nav_sys__imu_buff__data_format; success : out boolean);
end imu_buff;

package body imu_buff is
   protected type nav_sys__imu_buff__bufft is
      procedure nav_sys__imu_buff__bufft__place (item : in
        nav_sys__imu_buff__data_format; success : out boolean);
      procedure nav_sys__imu_buff__bufft__extract (item : out
        nav_sys__imu_buff__data_format; success : out boolean);
   private
      nav_sys__imu_buff__bufft__buffer_full : boolean := false;
      nav_sys__imu_buff__bufft__buffer_empty : boolean := true;
      nav_sys__imu_buff__bufft__data :
        nav_sys__imu_buff__data_format;
   end nav_sys__imu_buff__bufft;
   type nav_sys__imu_buff__buffTV is limited record
      buffer_full : boolean := false;
      buffer_empty : boolean := true;
      data : nav_sys__imu_buff__data_format;
      _object : aliased
        system__tasking__protected_objects__protection;
   end record;
   procedure nav_sys__imu_buff__buffPT__placeN (_object : in out
     nav_sys__imu_buff__buffTV; item : in
     nav_sys__imu_buff__data_format; success : out boolean);
   procedure nav_sys__imu_buff__buffPT__placeP (_object : in out
     nav_sys__imu_buff__buffTV; item : in
     nav_sys__imu_buff__data_format; success : out boolean);
   procedure nav_sys__imu_buff__buffPT__extractN (_object : in out
     nav_sys__imu_buff__buffTV; item : out
     nav_sys__imu_buff__data_format; success : out boolean);
   procedure nav_sys__imu_buff__buffPT__extractP (_object : in out
     nav_sys__imu_buff__buffTV; item : out
     nav_sys__imu_buff__data_format; success : out boolean);
   freeze nav_sys__imu_buff__buffTV [
      procedure nav_sys__imu_buff__buffTVIP (_init : in out
        nav_sys__imu_buff__buffTV) is
      begin
         _init.buffer_full := false;
         _init.buffer_empty := true;
         system__tasking__protected_objects__initialize_protection (
           _init._object'unchecked_access,
           system__tasking__unspecified_priority);
         return;
```

```
   end nav_sys__imu_buff__buffTVIP;
]
imu_buff__buff : nav_sys__imu_buff__buffT;
nav_sys__imu_buff__buffTVIP (nav_sys__imu_buff__buffTV!(
  imu_buff__buff));

procedure nav_sys__imu_buff__place_item (item :
  nav_sys__imu_buff__data_format; overwrote : out boolean) is
   ok : boolean;
begin
   nav_sys__imu_buff__buffPT__placeP (nav_sys__imu_buff__buffTV!(
     imu_buff__buff), item, ok);
   overwrote := not ok;
   return;
end nav_sys__imu_buff__place_item;

procedure nav_sys__imu_buff__extract_item (item : out
  nav_sys__imu_buff__data_format; success : out boolean) is
begin
   nav_sys__imu_buff__buffPT__extractP (nav_sys__imu_buff__buffTV!
     (imu_buff__buff), item, success);
   if not success then
      item := 0;
   end if;
   return;
end nav_sys__imu_buff__extract_item;


procedure nav_sys__imu_buff__buffPT__placeN (_object : in out
  nav_sys__imu_buff__buffTV; item : in
  nav_sys__imu_buff__data_format; success : out boolean) is
   buffR : system__tasking__protected_objects__protection
     renames _object._object;
   dataP : nav_sys__imu_buff__data_format renames _object.data;
   buffer_emptyP : boolean renames _object.buffer_empty;
   buffer_fullP : boolean renames _object.buffer_full;
begin
   success := not buffer_fullP;
   dataP := item;
   buffer_emptyP := false;
   buffer_fullP := true;
   return;
end nav_sys__imu_buff__buffPT__placeN;

procedure nav_sys__imu_buff__buffPT__extractN (_object : in out
  nav_sys__imu_buff__buffTV; item : out
  nav_sys__imu_buff__data_format; success : out boolean) is
   buffR : system__tasking__protected_objects__protection
     renames _object._object;
   dataP : nav_sys__imu_buff__data_format renames _object.data;
   buffer_emptyP : boolean renames _object.buffer_empty;
   buffer_fullP : boolean renames _object.buffer_full;
begin
   success := not buffer_emptyP;
   if not buffer_emptyP then
      item := dataP;
      buffer_fullP := false;
      buffer_emptyP := true;
   end if;
   return;
```

```
end nav_sys__imu_buff__buffPT__extractN;

procedure nav_sys__imu_buff__buffPT__extractP (_object : in out
  nav_sys__imu_buff__buffTV; item : out
  nav_sys__imu_buff__data_format; success : out boolean) is

   procedure nav_sys__imu_buff__buffPT__extractP___clean is
   begin
      system__tasking__protected_objects__unlock (_object._object'
        unchecked_access);
      system__soft_links__abort_undefer.all;
      return;
   end nav_sys__imu_buff__buffPT__extractP___clean;
begin
   system__fpga_kernel_interface__write_fps_cmd_no_entry (_object.
     _object'unchecked_access);
   system__soft_links__abort_defer.all;
   system__tasking__protected_objects__lock (_object._object'
     unchecked_access);
   system__fpga_kernel_interface__write_upe_cmd_no_entry (_object.
     _object'unchecked_access);
   system__fpga_kernel_interface__write_fpe_cmd_no_entry (_object.
     _object'unchecked_access);
   B161b : begin
      nav_sys__imu_buff__buffPT__extractN (_object, item, success);
   exception
      when all others =>
         B164b : declare
            E163b : ada__exceptions__exception_occurrence;
         begin
            ada__exceptions__save_occurrence (E163b,
              system__soft_links__get_current_excep.all.all);
            nav_sys__imu_buff__buffPT__extractP___clean;
            ada__exceptions__reraise_occurrence_no_defer (E163b);
         end B164b;
   at end
      nav_sys__imu_buff__buffPT__extractP___clean;
   end B161b;
   return;
end nav_sys__imu_buff__buffPT__extractP;

procedure nav_sys__imu_buff__buffPT__placeP (_object : in out
  nav_sys__imu_buff__buffTV; item : in
  nav_sys__imu_buff__data_format; success : out boolean) is

   procedure nav_sys__imu_buff__buffPT__placeP___clean is
   begin
      system__tasking__protected_objects__unlock (_object._object'
        unchecked_access);
      system__soft_links__abort_undefer.all;
      return;
   end nav_sys__imu_buff__buffPT__placeP___clean;
begin
   system__fpga_kernel_interface__write_fps_cmd_no_entry (_object.
     _object'unchecked_access);
   system__soft_links__abort_defer.all;
   system__tasking__protected_objects__lock (_object._object'
     unchecked_access);
   system__fpga_kernel_interface__write_upe_cmd_no_entry (_object.
```

```
      _object'unchecked_access);
    system__fpga_kernel_interface__write_fpe_cmd_no_entry (_object.
      _object'unchecked_access);
    B153b : begin
       nav_sys__imu_buff__buffPT__placeN (_object, item, success);
    exception
       when all others =>
          B156b : declare
             E155b : ada__exceptions__exception_occurrence;
          begin
             ada__exceptions__save_occurrence (E155b,
               system__soft_links__get_current_excep.all.all);
             nav_sys__imu_buff__buffPT__placeP___clean;
             ada__exceptions__reraise_occurrence_no_defer (E155b);
          end B156b;
    at end
       nav_sys__imu_buff__buffPT__placeP___clean;
    end B153b;
    return;
 end nav_sys__imu_buff__buffPT__placeP;
begin
 null;
end imu_buff;

package imu_buff is new gen_buff;

package los_buff is
 package los_buff__gen_buff renames los_buff;
 package los_buff__gen_buffGH renames los_buff__gen_buff;
 type nav_sys__los_buff__data_format is range -100 .. 100;
 procedure nav_sys__los_buff__place_item (item :
   nav_sys__los_buff__data_format; overwrote : out boolean);
 procedure nav_sys__los_buff__extract_item (item : out
   nav_sys__los_buff__data_format; success : out boolean);
end los_buff;

package body los_buff is
 protected type nav_sys__los_buff__bufft is
    procedure nav_sys__los_buff__bufft__place (item : in
      nav_sys__los_buff__data_format; success : out boolean);
    procedure nav_sys__los_buff__bufft__extract (item : out
      nav_sys__los_buff__data_format; success : out boolean);
 private
    nav_sys__los_buff__bufft__buffer_full : boolean := false;
    nav_sys__los_buff__bufft__buffer_empty : boolean := true;
    nav_sys__los_buff__bufft__data :
      nav_sys__los_buff__data_format;
 end nav_sys__los_buff__bufft;
 type nav_sys__los_buff__buffTV is limited record
    buffer_full : boolean := false;
    buffer_empty : boolean := true;
    data : nav_sys__los_buff__data_format;
    _object : aliased
      system__tasking__protected_objects__protection;
 end record;
 procedure nav_sys__los_buff__buffPT__placeN (_object : in out
   nav_sys__los_buff__buffTV; item : in
   nav_sys__los_buff__data_format; success : out boolean);
 procedure nav_sys__los_buff__buffPT__placeP (_object : in out
   nav_sys__los_buff__buffTV; item : in
```

```
  nav_sys__los_buff__data_format; success : out boolean);
procedure nav_sys__los_buff__buffPT__extractN (_object : in out
  nav_sys__los_buff__buffTV; item : out
  nav_sys__los_buff__data_format; success : out boolean);
procedure nav_sys__los_buff__buffPT__extractP (_object : in out
  nav_sys__los_buff__buffTV; item : out
  nav_sys__los_buff__data_format; success : out boolean);
freeze nav_sys__los_buff__buffTV [
   procedure nav_sys__los_buff__buffTVIP (_init : in out
     nav_sys__los_buff__buffTV) is
   begin
      _init.buffer_full := false;
      _init.buffer_empty := true;
      system__tasking__protected_objects__initialize_protection (
        _init._object'unchecked_access,
        system__tasking__unspecified_priority);
      return;
   end nav_sys__los_buff__buffTVIP;
]
los_buff__buff : nav_sys__los_buff__bufft;
nav_sys__los_buff__buffTVIP (nav_sys__los_buff__buffTV!(
  los_buff__buff));

procedure nav_sys__los_buff__place_item (item :
  nav_sys__los_buff__data_format; overwrote : out boolean) is
   ok : boolean;
begin
   nav_sys__los_buff__buffPT__placeP (nav_sys__los_buff__buffTV!(
     los_buff__buff), item, ok);
   overwrote := not ok;
   return;
end nav_sys__los_buff__place_item;

procedure nav_sys__los_buff__extract_item (item : out
  nav_sys__los_buff__data_format; success : out boolean) is
begin
   nav_sys__los_buff__buffPT__extractP (nav_sys__los_buff__buffTV!
     (los_buff__buff), item, success);
   if not success then
      item := 0;
   end if;
   return;
end nav_sys__los_buff__extract_item;


procedure nav_sys__los_buff__buffPT__placeN (_object : in out
  nav_sys__los_buff__buffTV; item : in
  nav_sys__los_buff__data_format; success : out boolean) is
   buffR : system__tasking__protected_objects__protection
     renames _object._object;
   dataP : nav_sys__los_buff__data_format renames _object.data;
   buffer_emptyP : boolean renames _object.buffer_empty;
   buffer_fullP : boolean renames _object.buffer_full;
begin
   success := not buffer_fullP;
   dataP := item;
   buffer_emptyP := false;
   buffer_fullP := true;
   return;
end nav_sys__los_buff__buffPT__placeN;
```

```
procedure nav_sys__los_buff__buffPT__extractN (_object : in out
  nav_sys__los_buff__buffTV; item : out
  nav_sys__los_buff__data_format; success : out boolean) is
   buffR : system__tasking__protected_objects__protection
     renames _object._object;
   dataP : nav_sys__los_buff__data_format renames _object.data;
   buffer_emptyP : boolean renames _object.buffer_empty;
   buffer_fullP : boolean renames _object.buffer_full;
begin
   success := not buffer_emptyP;
   if not buffer_emptyP then
      item := dataP;
      buffer_fullP := false;
      buffer_emptyP := true;
   end if;
   return;
end nav_sys__los_buff__buffPT__extractN;

procedure nav_sys__los_buff__buffPT__extractP (_object : in out
  nav_sys__los_buff__buffTV; item : out
  nav_sys__los_buff__data_format; success : out boolean) is

   procedure nav_sys__los_buff__buffPT__extractP___clean is
   begin
      system__tasking__protected_objects__unlock (_object._object'
        unchecked_access);
      system__soft_links__abort_undefer.all;
      return;
   end nav_sys__los_buff__buffPT__extractP___clean;
begin
   system__fpga_kernel_interface__write_fps_cmd_no_entry (_object.
     _object'unchecked_access);
   system__soft_links__abort_defer.all;
   system__tasking__protected_objects__lock (_object._object'
     unchecked_access);
   system__fpga_kernel_interface__write_upe_cmd_no_entry (_object.
     _object'unchecked_access);
   system__fpga_kernel_interface__write_fpe_cmd_no_entry (_object.
     _object'unchecked_access);
   B178b : begin
      nav_sys__los_buff__buffPT__extractN (_object, item, success);
   exception
      when all others =>
         B181b : declare
            E180b : ada__exceptions__exception_occurrence;
         begin
            ada__exceptions__save_occurrence (E180b,
              system__soft_links__get_current_excep.all.all);
            nav_sys__los_buff__buffPT__extractP___clean;
            ada__exceptions__reraise_occurrence_no_defer (E180b);
         end B181b;
   at end
      nav_sys__los_buff__buffPT__extractP___clean;
   end B178b;
   return;
end nav_sys__los_buff__buffPT__extractP;

procedure nav_sys__los_buff__buffPT__placeP (_object : in out
```

```
      nav_sys__los_buff__buffTV; item : in
      nav_sys__los_buff__data_format; success : out boolean) is

       procedure nav_sys__los_buff__buffPT__placeP___clean is
       begin
          system__tasking__protected_objects__unlock (_object._object'
            unchecked_access);
          system__soft_links__abort_undefer.all;
          return;
       end nav_sys__los_buff__buffPT__placeP___clean;
    begin
       system__fpga_kernel_interface__write_fps_cmd_no_entry (_object.
         _object'unchecked_access);
       system__soft_links__abort_defer.all;
       system__tasking__protected_objects__lock (_object._object'
         unchecked_access);
       system__fpga_kernel_interface__write_upe_cmd_no_entry (_object.
         _object'unchecked_access);
       system__fpga_kernel_interface__write_fpe_cmd_no_entry (_object.
         _object'unchecked_access);
       B170b : begin
          nav_sys__los_buff__buffPT__placeN (_object, item, success);
       exception
          when all others =>
             B173b : declare
                E172b : ada__exceptions__exception_occurrence;
             begin
                ada__exceptions__save_occurrence (E172b,
                  system__soft_links__get_current_excep.all.all);
                nav_sys__los_buff__buffPT__placeP___clean;
                ada__exceptions__reraise_occurrence_no_defer (E172b);
             end B173b;
       at end
          nav_sys__los_buff__buffPT__placeP___clean;
       end B170b;
       return;
    end nav_sys__los_buff__buffPT__placeP;
begin
    null;
end los_buff;

package los_buff is new gen_buff;
type nav_sys__gps_message_datatype is range -100 .. 100;
kf_data : kf_nav_buff.nav_sys__kf_nav_buff__data_format;
imu_data : imu_buff.nav_sys__imu_buff__data_format;
los_data : los_buff.nav_sys__los_buff__data_format;
gps_data : nav_sys__gps_message_datatype;
protected type nav_sys__epochT is
    procedure nav_sys__epochT__get_start_time (t : out ada.
      ada__real_time.ada__real_time__time);
private
    pragma priority (245);
    nav_sys__epochT__start : ada.ada__real_time.ada__real_time__time;
    nav_sys__epochT__first : boolean := true;
end nav_sys__epochT;
type nav_sys__epochTV is limited record
    start : ada.ada__real_time.ada__real_time__time;
    first : boolean := true;
    _object : aliased system__tasking__protected_objects__protection;
```

```
end record;
procedure nav_sys__epochPT__get_start_timeN (_object : in out
  nav_sys__epochTV; t : out ada__real_time__time);
procedure nav_sys__epochPT__get_start_timeP (_object : in out
  nav_sys__epochTV; t : out ada__real_time__time);
freeze nav_sys__epochTV [
   procedure nav_sys__epochTVIP (_init : in out nav_sys__epochTV) is
   begin
      _init.first := true;
      system__tasking__protected_objects__initialize_protection (
        _init._object'unchecked_access, 245);
      return;
   end nav_sys__epochTVIP;
]
epoch : nav_sys__epochT;
nav_sys__epochTVIP (nav_sys__epochTV!(epoch));

procedure nav_sys__epochPT__get_start_timeN (_object : in out
  nav_sys__epochTV; t : out ada__real_time__time) is
   epochR : system__tasking__protected_objects__protection renames
     _object._object;
   firstP : boolean renames _object.first;
   startP : ada__real_time__time renames _object.start;
begin
   if firstP then
      firstP := false;
      startP := ada.ada__real_time.clock;
   end if;
   t := startP;
   return;
end nav_sys__epochPT__get_start_timeN;

procedure nav_sys__epochPT__get_start_timeP (_object : in out
  nav_sys__epochTV; t : out ada__real_time__time) is

   procedure nav_sys__epochPT__get_start_timeP___clean is
   begin
      system__tasking__protected_objects__unlock (_object._object'
        unchecked_access);
      system__soft_links__abort_undefer.all;
      return;
   end nav_sys__epochPT__get_start_timeP___clean;
begin
   system__fpga_kernel_interface__write_fps_cmd_no_entry (_object.
     _object'unchecked_access);
   system__soft_links__abort_defer.all;
   system__tasking__protected_objects__lock (_object._object'
     unchecked_access);
   system__fpga_kernel_interface__write_upe_cmd_no_entry (_object.
     _object'unchecked_access);
   system__fpga_kernel_interface__write_fpe_cmd_no_entry (_object.
     _object'unchecked_access);
   B6b : begin
      nav_sys__epochPT__get_start_timeN (_object, t);
   exception
      when all others =>
         B9b : declare
            E8b : ada__exceptions__exception_occurrence;
         begin
```

```
            ada__exceptions__save_occurrence (E8b,
              system__soft_links__get_current_excep.all.all);
            nav_sys__epochPT__get_start_timeP___clean;
            ada__exceptions__reraise_occurrence_no_defer (E8b);
         end B9b;
   at end
      nav_sys__epochPT__get_start_timeP___clean;
   end B6b;
   return;
end nav_sys__epochPT__get_start_timeP;

protected type nav_sys__urgentT is
   procedure nav_sys__urgentT__check_timeout;
   procedure nav_sys__urgentT__unset_flag;
   procedure nav_sys__urgentT__reset_xpry_date;
   function nav_sys__urgentT__check_flag_set return boolean;
private
   nav_sys__urgentT__signal : boolean := false;
   nav_sys__urgentT__expiry_date : ada.ada__real_time.
     ada__real_time__time;
end nav_sys__urgentT;
type nav_sys__urgentTV is limited record
   signal : boolean := false;
   expiry_date : ada.ada__real_time.ada__real_time__time;
   _object : aliased system__tasking__protected_objects__protection;
end record;
procedure nav_sys__urgentPT__check_timeoutN (_object : in out
  nav_sys__urgentTV);
procedure nav_sys__urgentPT__check_timeoutP (_object : in out
  nav_sys__urgentTV);
procedure nav_sys__urgentPT__unset_flagN (_object : in out
  nav_sys__urgentTV);
procedure nav_sys__urgentPT__unset_flagP (_object : in out
  nav_sys__urgentTV);
procedure nav_sys__urgentPT__reset_xpry_dateN (_object : in out
  nav_sys__urgentTV);
procedure nav_sys__urgentPT__reset_xpry_dateP (_object : in out
  nav_sys__urgentTV);
function nav_sys__urgentPT__check_flag_setN (_object : in
  nav_sys__urgentTV) return boolean;
function nav_sys__urgentPT__check_flag_setP (_object : in
  nav_sys__urgentTV) return boolean;
freeze nav_sys__urgentTV [
   procedure nav_sys__urgentTVIP (_init : in out nav_sys__urgentTV) is
   begin
      _init.signal := false;
      system__tasking__protected_objects__initialize_protection (
        _init._object'unchecked_access,
        system__tasking__unspecified_priority);
      return;
   end nav_sys__urgentTVIP;
]
urgent : nav_sys__urgentT;
nav_sys__urgentTVIP (nav_sys__urgentTV!(urgent));

procedure nav_sys__urgentPT__check_timeoutN (_object : in out
  nav_sys__urgentTV) is
   urgentR : system__tasking__protected_objects__protection renames
     _object._object;
   expiry_dateP : ada__real_time__time renames _object.expiry_date;
```

```
   signalP : boolean renames _object.signal;
begin
   if (ada.ada__real_time.clock >= expiry_dateP) then
      signalP := true;
   end if;
   return;
end nav_sys__urgentPT__check_timeoutN;

procedure nav_sys__urgentPT__unset_flagN (_object : in out
  nav_sys__urgentTV) is
   urgentR : system__tasking__protected_objects__protection renames
     _object._object;
   expiry_dateP : ada__real_time__time renames _object.expiry_date;
   signalP : boolean renames _object.signal;
begin
   signalP := false;
   return;
end nav_sys__urgentPT__unset_flagN;

procedure nav_sys__urgentPT__reset_xpry_dateN (_object : in out
  nav_sys__urgentTV) is
   urgentR : system__tasking__protected_objects__protection renames
     _object._object;
   expiry_dateP : ada__real_time__time renames _object.expiry_date;
   signalP : boolean renames _object.signal;
begin
   expiry_dateP := ada__real_time__Oadd (ada.ada__real_time.clock,
     ada.ada__real_time.ada__real_time__milliseconds (19000));
   return;
end nav_sys__urgentPT__reset_xpry_dateN;

function nav_sys__urgentPT__check_flag_setN (_object : in
  nav_sys__urgentTV) return boolean is
   urgentR : system__tasking__protected_objects__protection renames
     _object._object;
   expiry_dateP : ada__real_time__time renames _object.expiry_date;
   signalP : boolean renames _object.signal;
begin
   return signalP;
end nav_sys__urgentPT__check_flag_setN;

function nav_sys__urgentPT__check_flag_setP (_object : in
  nav_sys__urgentTV) return boolean is

   procedure nav_sys__urgentPT__check_flag_setP___clean is
   begin
      system__tasking__protected_objects__unlock (_object._object'
        unchecked_access);
      system__soft_links__abort_undefer.all;
      return;
   end nav_sys__urgentPT__check_flag_setP___clean;
begin
   system__fpga_kernel_interface__write_fps_cmd_no_entry (_object.
     _object'unchecked_access);
   system__soft_links__abort_defer.all;
   system__tasking__protected_objects__lock (_object._object'
     unchecked_access);
   system__fpga_kernel_interface__write_ufe_cmd_no_entry (_object.
     _object'unchecked_access);
   system__fpga_kernel_interface__write_fpe_cmd_no_entry (_object.
```

```
      _object'unchecked_access);
   B39b : begin
      return nav_sys__urgentPT__check_flag_setN (_object);
   exception
      when all others =>
         B42b : declare
            E41b : ada__exceptions__exception_occurrence;
         begin
            ada__exceptions__save_occurrence (E41b,
              system__soft_links__get_current_excep.all.all);
            nav_sys__urgentPT__check_flag_setP___clean;
            ada__exceptions__reraise_occurrence_no_defer (E41b);
         end B42b;
   at end
      nav_sys__urgentPT__check_flag_setP___clean;
   end B39b;
end nav_sys__urgentPT__check_flag_setP;

procedure nav_sys__urgentPT__reset_xpry_dateP (_object : in out
  nav_sys__urgentTV) is

   procedure nav_sys__urgentPT__reset_xpry_dateP___clean is
   begin
      system__tasking__protected_objects__unlock (_object._object'
        unchecked_access);
      system__soft_links__abort_undefer.all;
      return;
   end nav_sys__urgentPT__reset_xpry_dateP___clean;
begin
   system__fpga_kernel_interface__write_fps_cmd_no_entry (_object.
     _object'unchecked_access);
   system__soft_links__abort_defer.all;
   system__tasking__protected_objects__lock (_object._object'
     unchecked_access);
   system__fpga_kernel_interface__write_upe_cmd_no_entry (_object.
     _object'unchecked_access);
   system__fpga_kernel_interface__write_fpe_cmd_no_entry (_object.
     _object'unchecked_access);
   B31b : begin
      nav_sys__urgentPT__reset_xpry_dateN (_object);
   exception
      when all others =>
         B34b : declare
            E33b : ada__exceptions__exception_occurrence;
         begin
            ada__exceptions__save_occurrence (E33b,
              system__soft_links__get_current_excep.all.all);
            nav_sys__urgentPT__reset_xpry_dateP___clean;
            ada__exceptions__reraise_occurrence_no_defer (E33b);
         end B34b;
   at end
      nav_sys__urgentPT__reset_xpry_dateP___clean;
   end B31b;
   return;
end nav_sys__urgentPT__reset_xpry_dateP;

procedure nav_sys__urgentPT__unset_flagP (_object : in out
  nav_sys__urgentTV) is
```

```
   procedure nav_sys__urgentPT__unset_flagP___clean is
   begin
      system__tasking__protected_objects__unlock (_object._object'
        unchecked_access);
      system__soft_links__abort_undefer.all;
      return;
   end nav_sys__urgentPT__unset_flagP___clean;
begin
   system__fpga_kernel_interface__write_fps_cmd_no_entry (_object.
     _object'unchecked_access);
   system__soft_links__abort_defer.all;
   system__tasking__protected_objects__lock (_object._object'
     unchecked_access);
   system__fpga_kernel_interface__write_upe_cmd_no_entry (_object.
     _object'unchecked_access);
   system__fpga_kernel_interface__write_fpe_cmd_no_entry (_object.
     _object'unchecked_access);
   B23b : begin
      nav_sys__urgentPT__unset_flagN (_object);
   exception
      when all others =>
         B26b : declare
            E25b : ada__exceptions__exception_occurrence;
         begin
            ada__exceptions__save_occurrence (E25b,
              system__soft_links__get_current_excep.all.all);
            nav_sys__urgentPT__unset_flagP___clean;
            ada__exceptions__reraise_occurrence_no_defer (E25b);
         end B26b;
   at end
      nav_sys__urgentPT__unset_flagP___clean;
   end B23b;
   return;
end nav_sys__urgentPT__unset_flagP;

procedure nav_sys__urgentPT__check_timeoutP (_object : in out
  nav_sys__urgentTV) is

   procedure nav_sys__urgentPT__check_timeoutP___clean is
   begin
      system__tasking__protected_objects__unlock (_object._object'
        unchecked_access);
      system__soft_links__abort_undefer.all;
      return;
   end nav_sys__urgentPT__check_timeoutP___clean;
begin
   system__fpga_kernel_interface__write_fps_cmd_no_entry (_object.
     _object'unchecked_access);
   system__soft_links__abort_defer.all;
   system__tasking__protected_objects__lock (_object._object'
     unchecked_access);
   system__fpga_kernel_interface__write_upe_cmd_no_entry (_object.
     _object'unchecked_access);
   system__fpga_kernel_interface__write_fpe_cmd_no_entry (_object.
     _object'unchecked_access);
   B15b : begin
      nav_sys__urgentPT__check_timeoutN (_object);
   exception
      when all others =>
```

```
         B18b : declare
            E17b : ada__exceptions__exception_occurrence;
         begin
            ada__exceptions__save_occurrence (E17b,
              system__soft_links__get_current_excep.all.all);
            nav_sys__urgentPT__check_timeoutP___clean;
            ada__exceptions__reraise_occurrence_no_defer (E17b);
         end B18b;
   at end
      nav_sys__urgentPT__check_timeoutP___clean;
   end B15b;
   return;
end nav_sys__urgentPT__check_timeoutP;

protected type nav_sys__event_object (nav_sys__event_object__ceiling :
  system.system__priority) is
   entry nav_sys__event_object__wait (d : out
     nav_sys__gps_message_datatype);
   type nav_sys__event_object__A43b is access all
     nav_sys__gps_message_datatype;
   type nav_sys__event_object__P44b is record
      d : nav_sys__event_object__A43b;
   end record;
   type nav_sys__event_object__A45b is access all
     nav_sys__event_object__P44b;
   procedure nav_sys__event_object__signal (d : in
     nav_sys__gps_message_datatype);
private
   pragma priority (ceiling);
   nav_sys__event_object__current : nav_sys__gps_message_datatype;
   nav_sys__event_object__signalled : boolean := false;
end nav_sys__event_object;
type nav_sys__event_objectV (ceiling : system__priority) is limited
  record
   _controller :
     system__finalization_implementation__limited_record_controller;
   current : nav_sys__gps_message_datatype;
   signalled : boolean := false;
   _object : aliased
     system__tasking__protected_objects__entries__protection_entries
     (1);
end record;
procedure nav_sys__event_objectPT__wait48bE (O : system__address; P :
  system__address; E :
  system__tasking__protected_objects__protected_entry_index);
function nav_sys__event_objectPT__wait49bB (O : system__address; E :
  system__tasking__protected_objects__protected_entry_index) return
  boolean;
procedure nav_sys__event_objectPT__signalN (_object : in out
  nav_sys__event_objectV; d : in nav_sys__gps_message_datatype);
procedure nav_sys__event_objectPT__signalP (_object : in out
  nav_sys__event_objectV; d : in nav_sys__gps_message_datatype);
subtype nav_sys__Tevent_objectAS is
  system__tasking__protected_objects__entries__protected_entry_body_array
  (1 .. 1);
reference nav_sys__TTevent_objectASP1
[storage_error when system__tasking__protected_objects__entry_body'
  size >= 16#8000_000# "object too large"]
event_objectA : aliased
  system__tasking__protected_objects__entries__protected_entry_body_array
```

```
      (1 .. 1) := ((
       barrier => nav_sys__event_objectPT__wait49bB'unrestricted_access,
       action => nav_sys__event_objectPT__wait48bE'unrestricted_access));
   function nav_sys__event_objectF (O : system__address; E :
     system__tasking__protected_objects__protected_entry_index) return
     system__tasking__protected_objects__protected_entry_index;
   freeze nav_sys__event_objectV [
      procedure nav_sys__event_objectVDI (1 : in out
        system__finalization_root__finalizable_ptr; v : in out
        nav_sys__event_objectV; b : short_short_integer) is
      begin
         system__finalization_implementation__initialize (v._controller);
         system__finalization_implementation__attach_to_final_list (1,
           system__finalization_root__finalizable?(v._controller), b);
         return;
      end nav_sys__event_objectVDI;
      procedure nav_sys__event_objectVDF (v : in out
        nav_sys__event_objectV; b : boolean) is
      begin
         if b then
            system__finalization_implementation__finalize_one (
              system__finalization_root__finalizable?(v._controller));
         else
            system__finalization_implementation__finalize (v.
              _controller);
         end if;
         return;
      exception
         when others =>
            system__soft_links__abort_undefer.all;
            [program_error "finalize raised exception"]
      end nav_sys__event_objectVDF;
      procedure nav_sys__event_objectVIP (_init : in out
        nav_sys__event_objectV; ceiling : system__priority) is
      begin
         _init.ceiling := ceiling;

           system__finalization_implementation__limited_record_controllerIP
           (_init._controller, P5s => true);
         _init.signalled := false;

           system__tasking__protected_objects__entries__protection_entriesIP
           (_init._object, 1, P14s => true);
         ada__finalization__initialize__2 (_init._object);
         system__finalization_implementation__attach_to_final_list (
           _init._controller.f, system__finalization_root__finalizable
           ?(_init._object), 1);


system__tasking__protected_objects__entries__initialize_protection_entries
           (_init._object'unchecked_access, ceiling, _init'address,
           event_objectA'unrestricted_access, nav_sys__event_objectF'
           unrestricted_access);
         return;
      end nav_sys__event_objectVIP;
   ]
   freeze nav_sys__event_object__P44b [
      procedure nav_sys__P44bIP (_init : in out
        nav_sys__event_object__P44b) is
      begin
```

```
      _init.d := null;
      return;
   end nav_sys__P44bIP;
]

function nav_sys__event_object__event_objectPT__wait49bB (O :
  system__address; E :
  system__tasking__protected_objects__protected_entry_index) return
  boolean is
   type nav_sys__event_objectPT__wait49bB__event_objectVP is access
     nav_sys__event_objectV;
   _object : nav_sys__event_objectPT__wait49bB__event_objectVP :=
     nav_sys__event_objectPT__wait49bB__event_objectVP!(O);
   [constraint_error when _object = null "access check failed"]
   event_objectR :
     system__tasking__protected_objects__entries__protection_entries
     renames _object.all._object;
   signalledP : boolean renames _object.all.signalled;
   currentP : nav_sys__gps_message_datatype renames _object.all.
     current;
   ceilingD : system__priority renames _object.all.ceiling;
begin
   return signalledP;
end nav_sys__event_object__event_objectPT__wait49bB;


procedure nav_sys__event_objectPT__wait48bE (O : system__address; P :
  system__address; E :
  system__tasking__protected_objects__protected_entry_index) is
   type nav_sys__event_objectPT__wait48bE__event_objectVP is access
     nav_sys__event_objectV;
   _object : nav_sys__event_objectPT__wait48bE__event_objectVP :=
     nav_sys__event_objectPT__wait48bE__event_objectVP!(O);
begin
   [constraint_error when _object = null "access check failed"]
   system__fpga_kernel_interface__write_ueb_cmd (_object.all._object'
     unchecked_access);
   B58b : declare
      event_objectR :
        system__tasking__protected_objects__entries__protection_entries
        renames _object.all._object;
      signalledP : boolean renames _object.all.signalled;
      currentP : nav_sys__gps_message_datatype renames _object.all.
        current;
      ceilingD : system__priority renames _object.all.ceiling;
   begin
      nav_sys__event_object__A45b!(P).d.all := currentP;
      signalledP := false;
   end B58b;
   [constraint_error when _object = null "access check failed"]

     system__tasking__protected_objects__operations__complete_entry_body
     (_object.all._object'unchecked_access);
   return;
exception
   when all others =>
      [constraint_error when _object = null "access check failed"]
      system__fpga_kernel_interface__write_uex_cmd (_object.all.
        _object'unchecked_access);
      [constraint_error when _object = null "access check failed"]
```

```
system__tasking__protected_objects__operations__exceptional_complete_entry_body
           (_object.all._object'unchecked_access,
           system__soft_links__get_gnat_exception);
         [constraint_error when _object = null "access check failed"]
         system__fpga_kernel_interface__write_ex_cmd (_object.all.
           _object'unchecked_access);
         return;
   end nav_sys__event_objectPT__wait48bE;

   procedure nav_sys__event_objectPT__signalN (_object : in out
     nav_sys__event_objectV; d : in nav_sys__gps_message_datatype) is
      event_objectR :
        system__tasking__protected_objects__entries__protection_entries
        renames _object._object;
      signalledP : boolean renames _object.signalled;
      currentP : nav_sys__gps_message_datatype renames _object.current;
      ceilingD : system__priority renames _object.ceiling;
   begin
      currentP := d;
      signalledP := true;
      return;
   end nav_sys__event_objectPT__signalN;

   function nav_sys__event_objectF (O : system__address; E :
     system__tasking__protected_objects__protected_entry_index) return
     system__tasking__protected_objects__protected_entry_index is
   begin
      return E;
   end nav_sys__event_objectF;

   procedure nav_sys__event_objectPT__signalP (_object : in out
     nav_sys__event_objectV; d : in nav_sys__gps_message_datatype) is

      procedure nav_sys__event_objectPT__signalP___clean is
      begin
         B70b : declare
            C69b : constant

system__tasking__protected_objects__entries__protection_entries_access :=
              _object._object'unchecked_access;
            self_id : constant system__tasking__task_id :=
              system__task_primitives__operations.self;
         begin

              system__tasking__protected_objects__operations__po_service_entries
              (self_id, C69b);
         end B70b;
         system__tasking__protected_objects__entries__unlock_entries (
           _object._object'unchecked_access);
         system__soft_links__abort_undefer.all;
         return;
      end nav_sys__event_objectPT__signalP___clean;
   begin
      system__fpga_kernel_interface__write_fps_cmd (_object._object'
        unchecked_access);
      system__soft_links__abort_defer.all;
      system__tasking__protected_objects__entries__lock_entries (
        _object._object'unchecked_access);
```

```
   system__fpga_kernel_interface__write_upe_cmd (_object._object'
     unchecked_access);
   system__fpga_kernel_interface__write_fpe_cmd (_object._object'
     unchecked_access);
   B67b : begin
      nav_sys__event_objectPT__signalN (_object, d);
   exception
      when all others =>
         B73b : declare
            E72b : ada__exceptions__exception_occurrence;
         begin
            ada__exceptions__save_occurrence (E72b,
              system__soft_links__get_current_excep.all.all);
            nav_sys__event_objectPT__signalP___clean;
            ada__exceptions__reraise_occurrence_no_defer (E72b);
         end B73b;
   at end
      nav_sys__event_objectPT__signalP___clean;
   end B67b;
   return;
end nav_sys__event_objectPT__signalP;

subtype nav_sys__Tgps_rcv_eventS is nav_sys__event_object (4);
gps_rcv_event : nav_sys__event_object (4);
nav_sys__event_objectVIP (nav_sys__TTgps_rcv_eventSV!(gps_rcv_event),
  4);
B75b : begin
   system__soft_links__abort_defer.all;
   nav_sys__event_objectVDI (F74b, nav_sys__event_objectV!(
     gps_rcv_event), 1);
exception
   when all others =>
      B77b : declare
         E76b : ada__exceptions__exception_occurrence;
      begin
         ada__exceptions__save_occurrence (E76b,
           system__soft_links__get_current_excep.all.all);
         system__standard_library__abort_undefer_direct;
         ada__exceptions__reraise_occurrence_no_defer (E76b);
      end B77b;
at end
   system__standard_library__abort_undefer_direct;
end B75b;
pouf1 : rand.rand__rand_time;
pouf2 : rand.rand__rand_time;
rand_offset1 : integer;
rand_offset2 : integer;
task type nav_sys__phantom1 (pri : system.system__priority; offset :
  natural) is
   pragma priority (pri);
end nav_sys__phantom1;
phantom1E : aliased boolean := false;
phantom1Z : system__parameters__size_type :=
  system__parameters__unspecified_size;
type nav_sys__phantom1V (pri : system__priority; offset : natural)
  is limited record
   _task_id : system__tasking__task_id;
   _priority : integer := pri;
end record;
procedure nav_sys__phantom1B (_task : access nav_sys__phantom1V);
```

```
freeze nav_sys__phantom1V [
   procedure nav_sys__phantom1VIP (_init : in out nav_sys__phantom1V;
     _chain : in out system__tasking__activation_chain; _task_name :
     in string; pri : system__priority; offset : natural) is
      subtype nav_sys__phantom1VIP__S79b is string (_task_name'first
        (1) .. _task_name'last(1));
   begin
      _init.pri := pri;
      _init.offset := offset;
      _init._task_id := null;
      _init._priority := pri;
      system__tasking__stages__create_task (_init._priority,
        phantom1Z, system__task_info__unspecified_task_info, 0,
        system__tasking__task_procedure_access!(nav_sys__phantom1B'
        address), _init'address, phantom1E'unchecked_access, _chain,
        _task_name, _init._task_id);
      return;
   end nav_sys__phantom1VIP;
]

procedure nav_sys__phantom1B (_task : access nav_sys__phantom1V) is
   offset : natural renames _task.all.offset;
   pri : system__priority renames _task.all.pri;

   procedure nav_sys__phantom1___clean is
   begin
      system__soft_links__abort_defer.all;
      system__tasking__stages__complete_task;
      system__soft_links__abort_undefer.all;
      return;
   end nav_sys__phantom1___clean;
begin
   system__soft_links__abort_undefer.all;
   next_period : ada.ada__real_time.ada__real_time__time;
   L_1 : label
   system__tasking__stages__complete_activation;
   nav_sys__epochPT__get_start_timeP (nav_sys__epochTV!(epoch),
     next_period);
   pouf1 := rand.get_rand_time;
   rand_offset1 := 5001 + 10 * integer(pouf1);
   next_period := ada__real_time__Oadd (ada__real_time__Oadd (
     next_period, ada.ada__real_time.ada__real_time__milliseconds (
     offset)), ada.ada__real_time.ada__real_time__milliseconds (
     rand_offset1));
   L_1 : loop
      ada__real_time__delays__delay_until (next_period);
      nav_sys__event_objectPT__signalP (nav_sys__TTgps_rcv_eventSV!(
        gps_rcv_event), nav_sys__gps_message_datatype(pouf1));
      pouf1 := rand.get_rand_time;
      rand_offset1 := 19001 + 10 * integer(pouf1);
      next_period := ada__real_time__Oadd (next_period, ada.
        ada__real_time.ada__real_time__milliseconds (rand_offset1));
   end loop L_1;
   return;
exception
   when all others =>
      B82b : declare
         E81b : ada__exceptions__exception_occurrence;
      begin
         ada__exceptions__save_occurrence (E81b,
```

```
            system__soft_links__get_current_excep.all.all);
          nav_sys__phantom1___clean;
          ada__exceptions__reraise_occurrence_no_defer (E81b);
       end B82b;
       return;
at end
   nav_sys__phantom1___clean;
end nav_sys__phantom1B;

phantom1E := true;
subtype nav_sys__Texternal_trigger_task_1S is nav_sys__phantom1 (
  4, 0);
external_trigger_task_1 : nav_sys__phantom1 (4, 0);
subtype nav_sys__TJ84bS is string (1 .. 0);
J84b : string (1 .. 0) := "";
nav_sys__phantom1VIP (nav_sys__TTexternal_trigger_task_1SV!(
  external_trigger_task_1), _chain, J84b, 4, 0);
task type nav_sys__phantom2 (pri : system.system__priority; offset :
  natural) is
   pragma priority (pri);
end nav_sys__phantom2;
phantom2E : aliased boolean := false;
phantom2Z : system__parameters__size_type :=
  system__parameters__unspecified_size;
type nav_sys__phantom2V (pri : system__priority; offset : natural)
  is limited record
   _task_id : system__tasking__task_id;
   _priority : integer := pri;
end record;
procedure nav_sys__phantom2B (_task : access nav_sys__phantom2V);
freeze nav_sys__phantom2V [
   procedure nav_sys__phantom2VIP (_init : in out nav_sys__phantom2V;
     _chain : in out system__tasking__activation_chain; _task_name :
     in string; pri : system__priority; offset : natural) is
      subtype nav_sys__phantom2VIP__S87b is string (_task_name'first
        (1) .. _task_name'last(1));
   begin
      _init.pri := pri;
      _init.offset := offset;
      _init._task_id := null;
      _init._priority := pri;
      system__tasking__stages__create_task (_init._priority,
        phantom2Z, system__task_info__unspecified_task_info, 0,
        system__tasking__task_procedure_access!(nav_sys__phantom2B'
        address), _init'address, phantom2E'unchecked_access, _chain,
        _task_name, _init._task_id);
      return;
   end nav_sys__phantom2VIP;
]

procedure nav_sys__phantom2B (_task : access nav_sys__phantom2V) is
   offset : natural renames _task.all.offset;
   pri : system__priority renames _task.all.pri;

   procedure nav_sys__phantom2___clean is
   begin
      system__soft_links__abort_defer.all;
      system__tasking__stages__complete_task;
      system__soft_links__abort_undefer.all;
      return;
```

```
   end nav_sys__phantom2___clean;
begin
   system__soft_links__abort_undefer.all;
   next_period : ada.ada__real_time.ada__real_time__time;
   dont_care : boolean;
   L_2 : label
   system__tasking__stages__complete_activation;
   nav_sys__epochPT__get_start_timeP (nav_sys__epochTV!(epoch),
     next_period);
   pouf2 := rand.get_rand_time;
   rand_offset2 := 1001 + 10 * integer(pouf2);
   next_period := ada__real_time__Oadd (ada__real_time__Oadd (
     next_period, ada.ada__real_time.ada__real_time__milliseconds (
     offset)), ada.ada__real_time.ada__real_time__milliseconds (
     rand_offset2));
   L_2 : loop
      ada__real_time__delays__delay_until (next_period);
      imu_buff.nav_sys__imu_buff__place_item (imu_buff.
        nav_sys__imu_buff__data_format(pouf2), dont_care);
      pouf2 := rand.get_rand_time;
      rand_offset2 := 5001 + 10 * integer(pouf2);
      next_period := ada__real_time__Oadd (next_period, ada.
        ada__real_time.ada__real_time__milliseconds (rand_offset2));
   end loop L_2;
   return;
exception
   when all others =>
      B90b : declare
         E89b : ada__exceptions__exception_occurrence;
      begin
         ada__exceptions__save_occurrence (E89b,
           system__soft_links__get_current_excep.all.all);
         nav_sys__phantom2___clean;
         ada__exceptions__reraise_occurrence_no_defer (E89b);
      end B90b;
      return;
at end
   nav_sys__phantom2___clean;
end nav_sys__phantom2B;

phantom2E := true;
subtype nav_sys__Texternal_trigger_task_2S is nav_sys__phantom2 (
  4, 0);
external_trigger_task_2 : nav_sys__phantom2 (4, 0);
subtype nav_sys__TJ92bS is string (1 .. 0);
J92b : string (1 .. 0) := "";
nav_sys__phantom2VIP (nav_sys__TTexternal_trigger_task_2SV!(
  external_trigger_task_2), _chain, J92b, 4, 0);
task type nav_sys__gps_receiverTK is
   pragma priority (3);
end nav_sys__gps_receiverTK;
gps_receiverTKE : aliased boolean := false;
gps_receiverTKZ : system__parameters__size_type :=
  system__parameters__unspecified_size;
type nav_sys__gps_receiverTKV is limited record
   _task_id : system__tasking__task_id;
   _priority : integer := 3;
end record;
procedure nav_sys__gps_receiverTKB (_task : access
  nav_sys__gps_receiverTKV);
```

```
   freeze nav_sys__gps_receiverTKV [
      procedure nav_sys__gps_receiverTKVIP (_init : in out
        nav_sys__gps_receiverTKV; _chain : in out
        system__tasking__activation_chain; _task_name : in string) is
         subtype nav_sys__gps_receiverTKVIP__S95b is string (_task_name'
           first(1) .. _task_name'last(1));
      begin
         _init._task_id := null;
         _init._priority := 3;
         system__tasking__stages__create_task (_init._priority,
           gps_receiverTKZ, system__task_info__unspecified_task_info,
           0, system__tasking__task_procedure_access!(
           nav_sys__gps_receiverTKB'address), _init'address,
           gps_receiverTKE'unchecked_access, _chain, _task_name, _init.
           _task_id);
         return;
      end nav_sys__gps_receiverTKVIP;
   ]
   gps_receiver : nav_sys__gps_receiverTK;
   subtype nav_sys__TJ97bS is string (1 .. 0);
   J97b : string (1 .. 0) := "";
   nav_sys__gps_receiverTKVIP (nav_sys__gps_receiverTKV!(gps_receiver),
     _chain, J97b);

   procedure nav_sys__gps_receiverTKB (_task : access
     nav_sys__gps_receiverTKV) is

      procedure nav_sys__gps_receiverTK___clean is
      begin
         system__soft_links__abort_defer.all;
         system__tasking__stages__complete_task;
         system__soft_links__abort_undefer.all;
         return;
      end nav_sys__gps_receiverTK___clean;
   begin
      system__soft_links__abort_undefer.all;
      dont_care : boolean;
      L_3 : label
      system__tasking__stages__complete_activation;
      L_3 : loop
         L_3__B101b : declare
            X :
              system__tasking__protected_objects__protected_entry_index :=
              1;
            J99b : aliased nav_sys__gps_message_datatype;
            P : nav_sys__event_object__P44b := (
               d => J99b'unchecked_access);
            B100b :

system__tasking__protected_objects__operations__communication_block;


system__tasking__protected_objects__operations__communication_blockIP
              (B100b);
         begin
            type nav_sys__gps_receiverTK__A104b is access all
              nav_sys__TTgps_rcv_eventSV;
            R105b : nav_sys__gps_receiverTK__A104b :=
              nav_sys__TTgps_rcv_eventSV!(gps_rcv_event)'reference;
```

```
system__tasking__protected_objects__operations__protected_entry_call
              (R105b.all._object'unchecked_access, X, P'address,
              system__tasking__simple_call, B100b);
            [constraint_error when P.d = null "access check failed"]
            gps_data := P.d.all;
         end L_3__B101b;
         los_data := los_buff.nav_sys__los_buff__data_format(gps_data);
         los_buff.nav_sys__los_buff__place_item (los_data, dont_care);
      end loop L_3;
      return;
   exception
      when all others =>
         B108b : declare
            E107b : ada__exceptions__exception_occurrence;
         begin
            ada__exceptions__save_occurrence (E107b,
              system__soft_links__get_current_excep.all.all);
            nav_sys__gps_receiverTK___clean;
            ada__exceptions__reraise_occurrence_no_defer (E107b);
         end B108b;
         return;
   at end
      nav_sys__gps_receiverTK___clean;
   end nav_sys__gps_receiverTKB;

   gps_receiverTKE := true;
   task type nav_sys__lo (cycle_time : positive; pri : system.
     system__priority; offset : natural) is
      pragma priority (pri);
   end nav_sys__lo;
   loE : aliased boolean := false;
   loZ : system__parameters__size_type :=
     system__parameters__unspecified_size;
   type nav_sys__loV (cycle_time : positive; pri : system__priority;
     offset : natural) is limited record
      _task_id : system__tasking__task_id;
      _priority : integer := pri;
   end record;
   procedure nav_sys__loB (_task : access nav_sys__loV);
   freeze nav_sys__loV [
      procedure nav_sys__loVIP (_init : in out nav_sys__loV; _chain :
        in out system__tasking__activation_chain; _task_name : in
        string; cycle_time : positive; pri : system__priority; offset :
        natural) is
         subtype nav_sys__loVIP__S111b is string (_task_name'first(
           1) .. _task_name'last(1));
      begin
         _init.cycle_time := cycle_time;
         _init.pri := pri;
         _init.offset := offset;
         _init._task_id := null;
         _init._priority := pri;
         system__tasking__stages__create_task (_init._priority, loZ,
           system__task_info__unspecified_task_info, 0,
           system__tasking__task_procedure_access!(nav_sys__loB'address),
           _init'address, loE'unchecked_access, _chain, _task_name,
           _init._task_id);
         return;
      end nav_sys__loVIP;
```

```
]

procedure nav_sys__loB (_task : access nav_sys__loV) is
   offset : natural renames _task.all.offset;
   pri : system__priority renames _task.all.pri;
   cycle_time : positive renames _task.all.cycle_time;

   procedure nav_sys__lo___clean is
   begin
      system__soft_links__abort_defer.all;
      system__tasking__stages__complete_task;
      system__soft_links__abort_undefer.all;
      return;
   end nav_sys__lo___clean;
begin
   system__soft_links__abort_undefer.all;
   dont_care : boolean;
   success : boolean;
   next_period : ada.ada__real_time.ada__real_time__time;
   period : constant ada.ada__real_time.ada__real_time__time_span :=
     ada.ada__real_time.ada__real_time__milliseconds (cycle_time);
   L_4 : label
   system__tasking__stages__complete_activation;
   nav_sys__epochPT__get_start_timeP (nav_sys__epochTV!(epoch),
     next_period);
   next_period := ada__real_time__Oadd (next_period, ada.
     ada__real_time.ada__real_time__milliseconds (offset));
   L_4 : loop
      ada__real_time__delays__delay_until (next_period);
      los_buff.nav_sys__los_buff__extract_item (los_data, success);
      if success then
         kf_data := kf_nav_buff.nav_sys__kf_nav_buff__data_format
           (los_data);
         kf_nav_buff.nav_sys__kf_nav_buff__place_item (kf_data,
           dont_care);
         nav_sys__urgentPT__unset_flagP (nav_sys__urgentTV!(urgent));
      else
         if nav_sys__urgentPT__check_flag_setP (nav_sys__urgentTV!(
           urgent)) then
            kf_nav_buff.nav_sys__kf_nav_buff__place_item (50,
              dont_care);
            nav_sys__urgentPT__unset_flagP (nav_sys__urgentTV!(
              urgent));
         end if;
      end if;
      next_period := ada__real_time__Oadd (next_period, period);
   end loop L_4;
   return;
exception
   when all others =>
      B114b : declare
         E113b : ada__exceptions__exception_occurrence;
      begin
         ada__exceptions__save_occurrence (E113b,
           system__soft_links__get_current_excep.all.all);
         nav_sys__lo___clean;
         ada__exceptions__reraise_occurrence_no_defer (E113b);
      end B114b;
      return;
at end
```

```
   nav_sys__lo___clean;
end nav_sys__loB;

loE := true;
subtype nav_sys__Tkalman_filterS is nav_sys__lo (5000, 2, 0);
kalman_filter : nav_sys__lo (5000, 2, 0);
subtype nav_sys__TJ116bS is string (1 .. 0);
J116b : string (1 .. 0) := "";
nav_sys__loVIP (nav_sys__TTkalman_filterSV!(kalman_filter), _chain,
  J116b, 5000, 2, 0);
task type nav_sys__hi (cycle_time : positive; pri : system.
  system__priority; offset : natural) is
   pragma priority (pri);
end nav_sys__hi;
hiE : aliased boolean := false;
hiZ : system__parameters__size_type :=
  system__parameters__unspecified_size;
type nav_sys__hiV (cycle_time : positive; pri : system__priority;
  offset : natural) is limited record
   _task_id : system__tasking__task_id;
   _priority : integer := pri;
end record;
procedure nav_sys__hiB (_task : access nav_sys__hiV);
freeze nav_sys__hiV [
   procedure nav_sys__hiVIP (_init : in out nav_sys__hiV; _chain :
     in out system__tasking__activation_chain; _task_name : in
     string; cycle_time : positive; pri : system__priority; offset :
     natural) is
      subtype nav_sys__hiVIP__S119b is string (_task_name'first(
        1) .. _task_name'last(1));
   begin
      _init.cycle_time := cycle_time;
      _init.pri := pri;
      _init.offset := offset;
      _init._task_id := null;
      _init._priority := pri;
      system__tasking__stages__create_task (_init._priority, hiZ,
        system__task_info__unspecified_task_info, 0,
        system__tasking__task_procedure_access!(nav_sys__hiB'address),
        _init'address, hiE'unchecked_access, _chain, _task_name,
        _init._task_id);
      return;
   end nav_sys__hiVIP;
]

procedure nav_sys__hiB (_task : access nav_sys__hiV) is
   offset : natural renames _task.all.offset;
   pri : system__priority renames _task.all.pri;
   cycle_time : positive renames _task.all.cycle_time;

   procedure nav_sys__hi___clean is
   begin
      system__soft_links__abort_defer.all;
      system__tasking__stages__complete_task;
      system__soft_links__abort_undefer.all;
      return;
   end nav_sys__hi___clean;
begin
   system__soft_links__abort_undefer.all;
   success1 : boolean;
```

```
   success2 : boolean;
   next_period : ada.ada__real_time.ada__real_time__time;
   period : constant ada.ada__real_time.ada__real_time__time_span :=
     ada.ada__real_time.ada__real_time__milliseconds (cycle_time);
   L_5 : label
   system__tasking__stages__complete_activation;
   nav_sys__epochPT__get_start_timeP (nav_sys__epochTV!(epoch),
     next_period);
   next_period := ada__real_time__Oadd (next_period, ada.
     ada__real_time.ada__real_time__milliseconds (offset));
   nav_sys__urgentPT__reset_xpry_dateP (nav_sys__urgentTV!(urgent));
   L_5 : loop
      ada__real_time__delays__delay_until (next_period);
      imu_buff.nav_sys__imu_buff__extract_item (imu_data, success1);
      if success1 then
         ada__text_io__put__4 ("IMU data is being processed");
         ada__text_io__new_line__2 (spacing => 1);
      end if;
      nav_sys__urgentPT__check_timeoutP (nav_sys__urgentTV!(urgent));
      if nav_sys__urgentPT__check_flag_setP (nav_sys__urgentTV!(
        urgent)) then
         ada__text_io__put__4 ("Time's up.");
         ada__text_io__new_line__2 (spacing => 1);
      end if;
      kf_nav_buff.nav_sys__kf_nav_buff__extract_item (kf_data,
        success2);
      if success2 then
         ada__text_io__put__4 (
           "Hi: success, we reset the expiry date and the data is ");
         ada.ada__integer_text_io.ada__integer_text_io__put__2 (
           integer(kf_data), width =>
           ada__integer_text_io__default_width, base =>
           ada__integer_text_io__default_base);
         ada__text_io__new_line__2 (spacing => 1);
         nav_sys__urgentPT__reset_xpry_dateP (nav_sys__urgentTV!(
           urgent));
         nav_sys__urgentPT__unset_flagP (nav_sys__urgentTV!(urgent));
      else
         ada__text_io__put__4 ("Hi: no data in KF buffer");
         ada__text_io__new_line__2 (spacing => 1);
      end if;
      next_period := ada__real_time__Oadd (next_period, period);
   end loop L_5;
   return;
exception
   when all others =>
      B126b : declare
         E125b : ada__exceptions__exception_occurrence;
      begin
         ada__exceptions__save_occurrence (E125b,
           system__soft_links__get_current_excep.all.all);
         nav_sys__hi___clean;
         ada__exceptions__reraise_occurrence_no_defer (E125b);
      end B126b;
      return;
at end
   nav_sys__hi___clean;
end nav_sys__hiB;

hiE := true;
```

```
   subtype nav_sys__TsequencerS is nav_sys__hi (1000, 1, 0);
   sequencer : nav_sys__hi (1000, 1, 0);
   subtype nav_sys__TJ128bS is string (1 .. 0);
   J128b : string (1 .. 0) := "";
   nav_sys__hiVIP (nav_sys__TTsequencerSV!(sequencer), _chain, J128b,
     1000, 1, 0);
   system__tasking__stages__activate_tasks (_chain'unchecked_access);
   ada__text_io__put__4 ("Main procedure body execution");
   ada__text_io__new_line__2 (spacing => 1);
   return;
exception
   when all others =>
      B184b : declare
         E183b : ada__exceptions__exception_occurrence;
      begin
         ada__exceptions__save_occurrence (E183b,
           system__soft_links__get_current_excep.all.all);
         nav_sys___clean;
         ada__exceptions__reraise_occurrence_no_defer (E183b);
      end B184b;
at end
   nav_sys___clean;
end nav_sys;
```