EVALUATION OF REAL-TIME OPERATING SYSTEMS FOR SAFETY-CRITICAL SYSTEMS

Marcus Hedlund
Fredrik Aronson

EXAMENSARBETE 2002
ELEKTRONIK
EVALUATION OF REAL-TIME OPERATING SYSTEMS FOR SAFETY-CRITICAL SYSTEMS

Marcus Hedlund
Fredrik Aronson

This thesis is performed at The School of Engineering at Jönköping University, within the subject field of electronics and embedded systems. The work is a part of the 1 1/2 year long education to masters grade. The authors are responsible for their opinions, conclusions and results.

Supervisors: Alf Johansson (The School of Engineering)
Mikael Lundqvist (SAAB Avionics AB)
Lars Hansson (SAAB Avionics AB)

Extent: 20 p (D-level)

Date: Dec 2002
Abstract

Saab Avionics in Jönköping develops safety critical products for military applications. A Real-Time Operating System (RTOS) can be used to develop programs for hard real-time applications. When using an RTOS within Saab Avionics, several demands need to be fulfilled. One of the more important requirements is the ability to comply with the DO-178B level A safety standard. As Ada is a programming language with high reliability and is used at Saab Avionics, the development environment should also be compatible with an Ada compiler.

This work is divided into two parts. The first part is a theoretical study of different RTOS. Some chosen RTOS are compared with the demands mentioned above to see which OS is best suited for Saab Avionics applications. Only a few of the available vendors seemed to offer products that could comply with these requirements. These products were studied in more detail. A short list of some suitable RTOS is the end result of this work.

The second part is not as theoretical as the first one. A test suite was to be created, which could verify that real-time functions work in a system, containing a BSP and an RTOS. The main thought is that if the BSP is not correctly set up, some functions will not run properly. A section of an already existing test suite for Ada compilers was used to create the tests. A complementary test suite was also created to further improve the test possibilities.

Keywords

RTOS, BSP, Ada, VxWorks, AdaMulti, GNAT, Tornado, DO-178B, ACATS and Ravenscar
Sammanfattning


Nyckelord

RTOS, BSP, Ada, VxWorks, AdaMulti, GNAT, Tornado, DO-178B, ACATS och Ravenscar
Acknowledgements

During this work at Saab Avionics, a lot of help has been given. We would like to thank all people involved at the “mission systems” department. There are also some people we would like to direct a special thanks to, for their assistance in our work:

Our supervisor Mikael Lundqvist has been the most involved in our assignment. Mikael has spent a lot of his time, encouraging, giving us tips and answering our questions. Lars Hansson has also been very encouraging and supporting. For his help with the used hardware, we would like to thank Anders Johansson. We would also like to show appreciation to Ada Core Technologies for their technical support and for extending the evaluation licence of GNAT pro.
Table of contents

1 INTRODUCTION --------------------------------------------------------------- 1
  1.1 COMPANY BACKGROUND ------------------------------------------------------ 1
  1.2 PURPOSE ----------------------------------------------------------------- 1
  1.3 PROJECT GOALS ------------------------------------------------------------- 2
  1.4 LIMITATIONS --------------------------------------------------------------- 2
  1.5 THESIS STRUCTURE ---------------------------------------------------------- 2

Part I

2 THEORETICAL BACKGROUND ----------------------------------------------------- 5
  2.1 PROGRAMMING LANGUAGE ------------------------------------------------------ 5
  2.2 RTS/RTOS ----------------------------------------------------------------- 5
    2.2.1 What is an RTS? -------------------------------------------------------- 5
    2.2.2 What is an RTOS? ------------------------------------------------------- 5
  2.3 BASICS OF AN RTOS ---------------------------------------------------------- 7
    2.3.1 Client/server OS ------------------------------------------------------- 7
    2.3.2 Processes and threads --------------------------------------------------- 8
    2.3.3 Scheduling ------------------------------------------------------------- 8
    2.3.4 Critical sections and mutual exclusion ---------------------------------- 10
    2.3.5 How semaphores works --------------------------------------------------- 10
    2.3.6 Monitors --------------------------------------------------------------- 11
    2.3.7 Protected objects ------------------------------------------------------- 11
    2.3.8 Interrupts -------------------------------------------------------------- 12
  2.4 CRITERIA’S FOR A GOOD RTOS ----------------------------------------------- 13
    2.4.1 Safety features -------------------------------------------------------- 13
    2.4.2 Switching times -------------------------------------------------------- 13
    2.4.3 POSIX ----------------------------------------------------------------- 14
    2.4.4 Memory protection ------------------------------------------------------ 14
    2.4.5 Scalability ------------------------------------------------------------- 14
    2.4.6 Price ----------------------------------------------------------------- 14
    2.4.7 Scheduling ------------------------------------------------------------- 14
  2.5 BUY OR DESIGN AN RTOS? ---------------------------------------------------- 16

3 METHOD ---------------------------------------------------------------------- 17
  3.1 RTOS PRODUCTS ------------------------------------------------------------- 17
    3.1.1 Integrity/AdaMulti by Green Hill ---------------------------------------- 17
    3.1.2 VxWorks/Tornado II by WindRiver ---------------------------------------- 19
    3.1.3 VxWorks AE/Tornado III by WindRiver ------------------------------------ 19
    3.1.4 QNX Neutrino (6.2)/Moments Development suite by QNX Software Systems ---- 21
    3.1.5 LynxOS/CodeWarrior by Lynux Works ------------------------------------- 22

4 RESULT ---------------------------------------------------------------------- 23
  4.1 INTEGRITY ---------------------------------------------------------------- 23
  4.2 VxWORKS ------------------------------------------------------------------ 24
  4.3 VxWORKS AE --------------------------------------------------------------- 24
  4.4 QNX 6.2 ------------------------------------------------------------------- 25
  4.5 LYNXOS -------------------------------------------------------------------- 25
# Table of contents

## Part II

5 THEORETICAL BACKGROUND ................................................................. 29

5.1 ADA ................................................................. 29
  5.1.1 What is Ada .............................................. 29
  5.1.2 The advantages of using Ada 95 ...................... 30
  5.1.3 Ada run-time library ................................... 30
  5.1.4 GSTART ....................................................... 30
  5.1.5 Ravenscar .................................................... 31

5.2 RTOS OR RAVENSCAR ADA .............................................. 33

5.3 ADA TEST SUITE .................................................. 33

5.4 PURPOSE OF ACATS ............................................. 34

5.5 WHAT IS A BSP? .................................................... 35
  5.5.1 Structure of a BSP ....................................... 35
  5.5.2 How is a BSP developed .............................. 36
  5.5.3 How to test a BSP ....................................... 36

6 METHOD ......................................................................................... 37

6.1 WHAT WILL BE TESTED .............................................. 37

6.2 CHOSEN TEST METHOD ............................................. 37

6.3 GETTING STARTED .................................................. 37

6.4 THE TEST SUITE ...................................................... 39
  6.4.1 The ACATS test suite ................................... 39
  6.4.2 ACATS naming .............................................. 41
  6.4.3 Ada extensions .............................................. 41
  6.4.4 Running the tests .......................................... 42
  6.4.5 Running several programs ......................... 42
  6.4.6 Problems ..................................................... 43
  6.4.7 The Complementary Test Suite .................... 44

6.5 MODIFYING THE BSP ................................................. 45

7 RESULT ..................................................................................... 47

7.1 VERIFICATION OF THE TEST SUITES ......................... 47

7.2 THE CHANGES MADE .................................................. 47
  7.2.1 Floating point ................................................ 47
  7.2.2 Cache .......................................................... 47
  7.2.3 MMU ............................................................. 48
  7.2.4 Clock rate ..................................................... 48

8 CONCLUSION ................................................................................. 49

8.1 FUTURE PLANS FOR PROJECT ................................. 50

9 REFERENCES AND LITERATURE ........................................... 51

9.1 REFERENCES .......................................................... 51

9.2 LITERATURE ........................................................... 54

10 GLOSSARY OF TERMS AND ABBREVIATIONS .......... 55

11 APPENDICES .............................................................................. 59
# Table of figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Gripen - Cockpit view</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Client/Server architecture</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>Task states</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>Scheduling behaviour</td>
<td>9</td>
</tr>
<tr>
<td>5</td>
<td>Concurrency problems</td>
<td>10</td>
</tr>
<tr>
<td>6</td>
<td>Semaphore example</td>
<td>11</td>
</tr>
<tr>
<td>7</td>
<td>Priority inversion example</td>
<td>14</td>
</tr>
<tr>
<td>8</td>
<td>Illustration showing the protection domains architecture</td>
<td>20</td>
</tr>
<tr>
<td>9</td>
<td>Augusta Ada Byron</td>
<td>29</td>
</tr>
<tr>
<td>10</td>
<td>The BSP in the structure of an embedded system</td>
<td>35</td>
</tr>
<tr>
<td>11</td>
<td>Ada - main program example</td>
<td>43</td>
</tr>
<tr>
<td>12</td>
<td>Example of how changes in the BSP can be made</td>
<td>45</td>
</tr>
</tbody>
</table>
1 INTRODUCTION

1.1 Company background

Saab Avionics is a leading supplier of avionics and electronic warfare systems on the international market and a principal supplier to JAS-39 Gripen. The company’s main operations cover electronic warfare systems, display systems, tactical reconnaissance systems, modular avionics, advanced mechanical equipment, and electromagnetic technology services.

Figure 1: Gripen - Cockpit view

1.2 Purpose

Saab Avionics uses Real Time Operating Systems (RTOS) to develop time-critical applications. This is the reason why this work focuses on real-time capabilities and RTOS.

This work is made up of two different investigations, where the purpose of the first investigation is to evaluate and find different RTOS that can be used with the Motorola PowerPC processor family. Since the RTOS will be used in military products, the demand for reliability is extremely high. Therefore the RTOS must also be compatible with the Ada 95 language, as it is a very well suited programming language for safety critical applications.

At the moment, Saab Avionics uses a number of software products from different vendors. For instance, Green Hill’s AdaMulti 2000 is used to develop applications written in Ada 95. These applications can be downloaded to hardware and then be run together with a run-time system such as GSTART. When using a run-time system or an RTOS together with hardware, a Board Support Package (BSP) is needed. The main purpose of the second investigation is to create a test suite, which verifies the real-time features of a system, consisting of a BSP and a run-time. The test suite will be used to see if it is possible to identify errors in the run-time or the BSP set up, which could affect the behaviour of the real-time features. The possibilities of just using this Ada run-time (GSTART) on its own without an RTOS as a complement will also be investigated.
1.3 Project goals

By making a market research find some RTOS that will fulfill certain safety standards and other requirements needed for Saab Avionics applications. Also develop a way of testing real-time functions in a system, consisting of a BSP and an RTOS.

1.4 Limitations

Five RTOS have been chosen as a limit to be able to make a thorough investigation within project time. The investigation will be theoretical, as it will require licenses for each RTOS if a more practical evaluation is done. The second investigation will only focus on the real-time features in a system and will not verify any other part in a BSP. This has been decided to prevent the assignment from becoming too large. Licenses are available for VxWorks, GSTART and GNAT Pro.

1.5 Thesis structure

This thesis begins with a comprehensive introduction, describing the purpose of the two investigations. Both parts begin with a theoretical background describing necessary information, making the investigations more understandable. The first part begins by describing what a Real-time operating system is and its features. It then leads to a more thorough study of the different RTOS on the market. The second part also begins by describing necessary background information. It then explains how a test suite can be developed and used. The work is summed up with a conclusion covering both investigations, explaining the results and an evaluation of what have been achieved. Hardware/software installation, test descriptions etc. are presented in the appendices.
Part I

Evaluation of Real-Time Operating Systems

Goal: Make a market research and find some RTOS that will fulfill certain safety standards and other requirements needed for Saab Avionics applications.
Part I - Theoretical background

2 THEORETICAL BACKGROUND

2.1 Programming language

Ada 95 is a program language that will be used to create the applications for different projects within the company. It is designed for applications where correctness, safety, and reliability are the prime goals. Ada is well suited for developing reusable components, real time, and parallel processing systems. More information about Ada is given in part II.

2.2 RTS/RTOS

2.2.1 What is an RTS?

There are different definitions of Real Time Systems (RTS). Here are a few of them.

- Real time computing is where system correctness depends not only on the correctness of the logical result of the computation, but also on the result delivery time.
- A real time system responds in a predictable way to unpredictable external stimuli arrivals.
- A real time system is an interactive system that maintains an ongoing relationship with an asynchronous environment, i.e. an environment that progress irrespective of the RTS, in an uncooperative manner.

As stated above, the system must be able to cope with errors and still have the functionality required to complete its task before its deadline.

There are various types of RTS:
- Hard real-time: Missing a deadline has catastrophic result for the system.
- Firm real-time: Missing a deadline entails an unacceptable quality reduction.
- Soft real-time: Missing a deadline gives an acceptable quality reduction.
- Non real-time: No deadlines have to be met.

A system can usually be made up of a mixture of these.

2.2.2 What is an RTOS?

A Real Time Operating System differs from a regular operating system such as Windows. A regular OS is optimised to reduce response time while an RTOS is optimised to finish tasks within their deadlines.

There are two kinds of real time operating systems. The most common type is called event driven. This type gives each task a priority and lets the task with the highest priority run first. The other type is called time driven. This type differs from event driven as it has a predefined running scheme.

The four most common types of task scheduling are:

1. Preemptive – a high priority task can immediately interrupt a low priority task.
2. Non-preemptive – a task with high priority cannot start its execution until the one running is finished.
4. Cyclic scheduling – tasks are scheduled to run at certain times.
Services that an RTOS provides are:

- Definition and activation of tasks
- Time management (in general and for tasks)
- Communication and synchronisation
- Error handling
- Input and output
- Memory handling
- Scheduling
2.3 Basics of an RTOS

The main ideas will now be explained to give a better understanding of how an RTOS works. The topics covered will be:

- Client/server model
- Processes and threads
- Scheduling
- Critical sections and mutual exclusions
- Interrupts

2.3.1 Client/server OS

Operating systems usually use a client/server approach, which means that applications act as clients requesting services from the server through system calls. The client/server architecture is a message based and modular infrastructure that is intended to improve:

- **Usability** – The ease that which a user can learn to operate, prepare inputs for, and interoperate outputs of a component.
- **Flexibility** – The ease with which a component can be modified for use in applications.
- **Scalability** – The ease with which a component can be modified to fit the problem area.
- **Interoperability** – The ability of two or more components to exchange information and to use the information.

![Client/Server Architecture](image)

**Figure 2: Client/Server architecture**

The major problem with this model is the overhead due to memory protection. Every time a service is requested, the system has to switch from the application’s memory space to the server’s memory space.
2.3.2 Processes and threads

An application, which runs on a system, can be subdivided into multiple tasks, also known as processes. When these processes are executed concurrently, it is called “multitasking”. This is essential when developing real time systems. The application must be able to respond in a predictable way to multiple, simultaneous external events. Switching from one process to another is time-consuming. Therefore the concepts of “threads” were invented. It is a sub process or a lightweight process that inherits the context of a process but uses only a subset of it so that switching between threads can be done more rapidly.

Tasks have 3 different states:
1) Running - this means that the microprocessor is executing the instructions of this task.
2) Ready - this means that some other task is in the running state but that this task has things that it could do if the processor becomes available.
3) Blocked- the task has nothing to do, even if the processor becomes available. The task gets into this state because it waits for an external event to happen.

![Task states diagram](image)

Figure 3: Task states

2.3.3 Scheduling

The scheduler is one of the basic parts in an RTOS. Its function is to switch from one task, thread or process to another. This is essential for a multitasking system. If more than one thread wants to use the processor simultaneously, an algorithm is needed to decide which thread will run first. There are many different kinds of algorithms that can be used for this purpose. Rate Monotonic Scheduling (RMS) is one of the most used and gives a decision rule concerning which priority level to assign to each thread. With this scheduling, a high priority process may be allowed to run during the execution of a lower priority one. In a pre-emptive scheme, there will be an immediate switch to a higher priority process. This method can increase chances of completing tasks before their deadlines.
Alternatively, with non-preemption, the lower priority process will be allowed to complete before the other executes.
Part I - Theoretical background

Ex: A system has three processes, A, B and C. A has the highest priority as it has the shortest period.

<table>
<thead>
<tr>
<th>Process</th>
<th>Period, T</th>
<th>Computation time, C</th>
<th>Priority, P</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>5</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>10</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td>20</td>
<td>7</td>
<td>3</td>
</tr>
</tbody>
</table>

Preemptive behaviour

Non-preemptive behaviour

Figure 4: Scheduling behaviour

As can be seen above, the preemptive behaviour fulfils the deadlines. However in the non-preemptive case, the task C cannot be interrupted by the higher priority tasks A and B. This causes Task A to miss its deadline.
2.3.4 Critical sections and mutual exclusion

When different tasks are being run, the scheduler will change the tasks being executed, depending on task priorities and interrupts. Sometimes the Operating System (OS) needs to disable the interrupts from time to time to execute critical code that should not be interrupted. This piece of code is known as “critical section”. The synchronization that is needed to protect a “critical section” is known as “mutual exclusion”.

Disabling the interrupts can be a solution to this problem as it prevents the OS from switching tasks. This is not a good way to handle the problem because it blocks higher prioritised tasks, which might not even affect the information in the critical section. Semaphores were created to handle these types of situations.

For example, semaphores can be used when two processes share memory, such as global variables. The semaphores will prevent both processes that share the same memory from interacting with the memory at the same time, but still enable a high priority task to be executed.

![Diagram](image)

**Figure 5: Concurrency problems**

The above example show how both cars will get a green signal and start driving, which causes a collision. This is what can happen without mutual exclusion.

2.3.5 How semaphores works

There are two types of semaphores: binary semaphore and counting semaphore. As its name implies, a binary semaphore can only take two values: 0 or 1. A counting semaphore allows values between 0 and 255, 65535 or 4294967295, depending on whether the semaphore mechanism is implemented using 8, 16 or 32 bits, respectively. The semaphores are atomic, which means that they cannot be interrupted. There are two different operations when using semaphores. These are called wait and signal.
Part I - Theoretical background

- The wait operation decreases the semaphore with 1, if the semaphore > 0. Otherwise it delays the process until the semaphores value becomes > 0.
- Signal operation increases the semaphore value by 1.

Before a semaphore can be used, it needs to be initialised. In the example below, the semaphore is initialised to a value of 1. When the program starts, it will only be possible to enter one of the processes. This is because the semaphore value is decreased to zero and prevents the other process from entering its critical section. When the critical section has been completed, the value is increased, enabling the other process to execute its code. In other words, the requesting task says: “Give me the key. If someone else is using it, I am willing to wait for it!”

![Semaphore example](image)

**Figure 6: Semaphore example**

Semaphores can increase the probability of errors occurring in the code, as they are difficult to use correctly. It is in other words easy to make mistakes. This is why alternative methods were created instead of semaphores.

### 2.3.6 Monitors

The monitor solution is a way to protect data instead of using semaphores. In a language that supports monitors, a collection of procedures is declared to be in the monitor, and mutual exclusion is enforced on those procedures – only one process can be executing with its program counter in monitor code.

### 2.3.7 Protected objects

The criticism of monitors centres on their use of condition variables. By replacing this approach to synchronisation by the use of guards, protected objects were created. ADA is the only major language providing this mechanism. A protected object encapsulates data and allows access to them only via protected subprograms or protected entries. The language guarantees that the data is updated under mutual exclusion. At the entry of a protected object, there are guards, termed as barriers, which must be evaluated true before a task is allowed entry. If not, the process attempting to enter the protected object will sleep until the running task is finished. An example of this is when writing to a shared memory [28].
Part I - Theoretical background

2.3.8 Interrupts

An interrupt is a hardware mechanism used to inform the CPU that an asynchronous event has occurred. When an interrupt is recognised, the CPU saves part (or all) of its context and jumps to a special subroutine called an interrupt service routine (ISR).

Interrupts allow a microprocessor to handle events when they occur. This is a better solution than polling. When using a polling method, the microprocessor continuously checks for an event, which uses up a lot of processing time.

As stated earlier, disabling interrupts during execution is usually a bad solution as this can prevent high priority tasks to finish within their deadline, if they are triggered during the time the interrupts are deactivated. However, in some cases when the running task is very important, a polling method is required instead of interrupts, to create a deterministic and safe behaviour.
2.4 Criteria’s for a good RTOS

2.4.1 Safety features

The RTOS should be capable to meet the commercial avionics safety critical standards, FAA’s DO-178B.

DO-178B defines “criticality” levels that are based upon the contribution of software to potential failure conditions as determined by the system safety assessment process. The software level implies that the level of effort required to show compliance with certification requirements varies with the failure condition category. Software level definitions are: [6]

**Level A:** Software with anomalous behaviour, as shown by the system safety assessment process, would cause or contribute to a failure of system function resulting in a catastrophic failure condition for the aircraft.

**Level B:** Software with anomalous behaviour, as shown by the system safety assessment process, would cause or contribute to a failure of system function resulting in a hazardous/severe-major failure condition for the aircraft.

**Level C:** Software with anomalous behaviour, as shown by the system safety assessment process, would cause or contribute to a failure of system function resulting in a major failure condition for the aircraft.

**Level D:** Software with anomalous behaviour, as shown by the system safety assessment process, would cause or contribute to a failure of system function resulting in a minor failure condition for the aircraft.

**Level E:** Software with anomalous behaviour, as shown by the system safety assessment process, would cause or contribute to a failure of system function with no effect on aircraft operational capability or pilot workload.

The table gives an idea of the cost involved when trying to achieve different levels [30].

<table>
<thead>
<tr>
<th>Level</th>
<th>Failure-consequence.</th>
<th>Cost.</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Catastrophic</td>
<td>2.8 * X</td>
</tr>
<tr>
<td>B</td>
<td>Hazardous</td>
<td>2.6 * X</td>
</tr>
<tr>
<td>C</td>
<td>Major</td>
<td>2 * X</td>
</tr>
<tr>
<td>D</td>
<td>Minor</td>
<td>X</td>
</tr>
<tr>
<td>E</td>
<td>No effect</td>
<td>Not included in standard</td>
</tr>
</tbody>
</table>

The table above shows the rule of thumb when calculating the development cost for DO-178B certification.

Note. Just because the RTOS can meet a safety standard, it does not mean that the finished product will. This depends on the application code, which has to be tested along with the RTOS to see if it can meet the standards mentioned above.

2.4.2 Switching times

When a multitasking kernel decides to run a different task, the current task’s context (CPU registers) is saved in a storage area (stack). Once this operation is performed, the new task’s context is restored from its storage area and then resumes its execution of the new task’s code. This process is called context switch or a task switch. The number of registers that have to be
Part I - Theoretical background

saved and restored by the CPU determines the time required to perform a context switch. The context switch time should be minimized as much as possible.

2.4.3 POSIX

Portable Operating System Interface (POSIX) is a set of operating system interfaces based on the UNIX operating system. The need for standardization arose because programs needed to be moved among different systems, without having to be recoded. UNIX was selected for this purpose. POSIX compatibility can become important in a real time operating system.

2.4.4 Memory protection

When two or more processes share the same physical memory, the OS must protect them from writing into each other’s memory. This is called memory protection, and it is made more difficult because access is such a basic attribute of CPUs.

2.4.5 Scalability

Scalability is the ability of a computer application or product (hardware or software) to continue to function well as it is changed in size or volume in order to meet a user need. Today’s RTOS are not scalable enough to deal with both small and complex systems. A particular RTOS is aimed at a particular application size. However, an RTOS that is scalable is a better choice than one that is not. Changing the size of the RTOS will change the amount of ROM and RAM needed to run the application. This can be an important factor when deciding what RTOS to choose.

2.4.6 Price

There are many different RTOS, varying widely in costs. It is not necessarily the most expensive one that is the best. Some RTOS has royalties, which is a payment to a company for each items sold under a patent by another company. A RTOS without royalties can become a cheaper alternative than one with this type of payment.

2.4.7 Scheduling

Priority inversion is a problem in a real time operative system that should be avoided. Priority inversion is when a lower priority task blocks a higher priority task.

![Figure 7: Priority inversion example](image-url)
Part I - Theoretical background

In the example above, priority inversion occurs. Task A has the highest priority and C has the lowest. As can be seen, these two tasks share a critical section that prevents the non-executing task to begin until the task executing is finished.

In this example process C begins at time 0 and enters its critical section until Task B, which has a higher priority, takes control of the processor. When Task A takes over control, it will not be able to finish as Task C has locked its semaphore. Task A will have to wait until task B is finished and until Task C leaves its critical section, unlocking its semaphore.

There are several ways to prevent this from happening. RTOS needs to cope with this kind of problem.
2.5 Buy or design an RTOS?

When the decision has been made that an RTOS is needed in a project, a choice has to be made, should we buy or produce our own?

As stated earlier, there are a few important characteristics that need to be taken in consideration when producing an effective RTOS:

- **Availability** – The RTOS should run on target processor and work well with the associated tools
- **Efficiency** – It must require little overhead and be extremely fast
- **Reliability** – The RTOS should be tested and proven with over hundreds of designs
- **Functionality** – It should offer the functions required for the application now and in the foreseeable future
- **Sustainability** – The RTOS should be maintained to fix current and future problems or needs. Products for avionic applications have very long life cycles and therefore needs maintenance and support during this period.
- **Economy** – The RTOS should not be a burdensome cost on the project
- **Quality** - It should be well documented and fully supported

A self-made RTOS will rarely live up to all of these expectations. Only a commercially produced and supported RTOS can offer these features.

Designing your own tools is never a good solution as vendors have spent several years producing and refining tools to many different clients. Millions of dollars have been invested into producing the software, and buying one of these RTOS enables you to share the knowledge these vendors possess. It is impossible to design, develop, implement, test, and support a self-made RTOS for even close to the price of a commercial software system.

If the design and development team does not have access to exceptional tools, the entire organisation will suffer from reduced competitiveness. An increase in productivity and faster time-to-market will easily justify the price of a commercial RTOS.

Even though the present needs are known, the future requirements are not. Software vendors devote a significant portion of its budget to ongoing research and development. Vendors have to do this as their competitive market depends upon it. The homemade RTOS will never be as rich in functionality, ease of use, flexibility, comprehensiveness, etc., and will not keep the pace as the years progress.

A typical problem with homemade software is that the documentation is far from adequate and there are only a few individuals that possess the knowledge of how the program works. If this person or persons would for some reason become unavailable, new users will rarely comprehend the program as well as the creator/creators. Vendors can provide well-written documentation, technical support throughout their use of the system, and even education of how the software works and how it can be used efficiently.

There is no use re-inventing the wheel, as it is a lot safer and easier to take advantage of the expertise the vendors possess. In other words, buying a commercial off-the-shelf RTOS is the most logical solution for a company.
Part I - Method

3 METHOD

3.1 RTOS products

The Internet was used to get an overview of all the RTOS existing on the market. From this overview a handful of the most promising RTOS were selected and studied more closely. Information from homepages, customers, test- and evaluation reports, and even contacts within the different companies were used to gather information about the RTOS in question. Non-biased information was prioritised as the companies’ sites do not mention the negative points of their products and like to brag about non-relevant issues.

The RTOS that seemed to be most popular and well known were chosen. These are the RTOS that will be described in this paper:

- **Integrity** by Greenhill Software,
- **VxWorks** and **VxWorks AE** by Windriver,
- **QNX Neutrino** by QNX Software Systems, and
- **LynxOS** by Lynx Real Time Systems.

Information of the different RTOS will be presented below.

3.1.1 **Integrity /AdaMulti by Green Hill**

**Product description**

The Integrity Real-Time Operating System is a secure, high reliability, royalty free RTOS intended for use in mission critical embedded systems [1]. Its object-oriented design allows verification of the secure communication, individual components, and the system as a whole. The Integrity RTOS together with Green Hills AdaMulti IDE, and Ada95/C/C++ compilers provides a complete single vendor RTOS and development solution for developing real time mission and safety critical software systems capable of meeting the security and safety standards of ISO/IEC 15408 and RTCA DO-178B. Integrity uses the hardware memory protection capabilities of MMUs to isolate the kernel and user processes into their own separate memory spaces, thereby preventing tasks with errors from affecting kernel operations or other processes. The kernel, which can be stored in ROM, uses a pre-emptive, priority-based multitasking scheduler that the company says permits developers to assign address spaces with a fixed amount of physical memory, thereby preventing any single task from depleting resources and depriving others. Tasks also may be assigned fixed percentages of CPU time, as defined in the standard ARINC 653, to guarantee deterministic behaviour [2, 40].

**The ARINC 653 protection domain scheduler**

Time management is a major characteristic of integrated modular avionics (IMA) systems. In order to meet this market need, ARINC 653 partition scheduler was offered as an option for real time operating systems. The ARINC 653 protection domain scheduler allows one to independently schedule several applications. Each application runs in a different application domain. It is possible to specify a number of clock ticks during which tasks from an application domain will execute. This period is called an instance. When this instance is finished, tasks from the next protection domain instance will be scheduled. When all protection domains instances have been scheduled, then the first protection domain instance is scheduled again. The protection domain scheduler has no impact on kernel tasks. Priority-based preemptive scheduling is used within a domain.
Part I - Method

- Guaranteed time to run for each application
- Priority-based scheduling within domains
- Space partitioning preserved
- ARINC 653 compliant

Tools

Integrity comes with a wide variety of tools. These tools make it easy to debug and understand how the whole design will function. A few of the tools that Green Hill provides will be explained below.

- **Resource Analyser**
  Development and run time analysis tool that provides extensive visibility into CPU execution at the task and address space level. Graphical interface to critical system information, including task usage, address space memory use, CPU time history, and memory use history.

- **Event Analyser**
  Enables programmers to understand the complex real-time interactions of their system. Important events like semaphore calls, task context switches, and interrupts are logged on the target in real time. The data is transferred to the host where it is displayed graphically. The data can be gathered in a circular buffer on the target, with a minimal effect on the system. User can upload information upon request, alternatively data can be sent continuously to the host via TCP/IP.

- **ISIM**
  Enables programmers to develop and test applications when target hardware is not yet manufactured.

- **Trace Points**
  Enables non-instructive data collection in running system. When a task hits a trace point, the specified registers and memory are probed and their values stored for later view. Ideal for trouble shooting or monitoring fielded systems.

**Users of Integrity**

<table>
<thead>
<tr>
<th>Company</th>
<th>Product</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goodrich [12]</td>
<td>EMC-100 Engine control</td>
<td>Follows standard DO-178B</td>
</tr>
<tr>
<td>MotoTron [13]</td>
<td>Engine control – Allstar 708 engine</td>
<td>Uses MPC555 (same as SAAB uses)</td>
</tr>
<tr>
<td>Lockheed Martin [14]</td>
<td>F-35 Joint Strike Fighter</td>
<td>Uses AdaMulti as development kit, DO-178B</td>
</tr>
</tbody>
</table>


Part 1 - Method

3.1.2 VxWorks/Tornado II by WindRiver

Product description

VxWorks 5.4 is developed by Wind River and is a part of the Tornado II development platform. Tornado uses C++ to develop applications for VxWorks, but it is also possible to develop applications in Java and Ada [4]. Ada development is done using either APEX (Rational Rose) or GNAT (Ada Core Technologies). Green Hill’s Multi Development suite could also be used.

VxWorks is designed for scalability enabling developers to allocate memory to the application, instead of the operating system.

Tools

Tornado 2 includes powerful development and debugging tools to handle problems in embedded applications. These tools address such issues as remote source-level debugging of both C and C++, target and tool management, system object tracking, memory usage profiling and automated configuration. [4]

Users of VxWorks

<table>
<thead>
<tr>
<th>Company</th>
<th>Product</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>NASA [15]</td>
<td>X-38 International Space Station “lifeboat”</td>
<td>-</td>
</tr>
<tr>
<td>Honda [16]</td>
<td>ASIMO Robot</td>
<td>-</td>
</tr>
</tbody>
</table>

3.1.3 VxWorks AE/Tornado III by WindRiver

Product Description

The newest RTOS from Wind River is VxWorks AE [8]. VxWorks AE is intended for critical embedded applications with demanding reliability, availability, and serviceability (RAS) requirements. It is an expansion of the original VxWorks, mentioned above.

VxWorks AE delivers:

- Unique protection domains that permit developers to create logical containers that isolates and protects applications running in separate domains (fig. 7). This type of protection is more powerful, easier to understand, and easier to work with.

- “Application serviceability” features that include automated reclamation. This prevents resource leaks that can occur over time in a dynamic system. All resources allocated within a VxWorks AE protection domain are, by default, owned by and associated with that domain. This means that if the domain is destroyed, all the resources are reclaimed and returned to the system pool.

- Powerful new loader facilities that automate linkages between software modules, thus improving developers’ productivity by permitting them to move in one domain into any other domain without recompiling code.
- Ability to upgrade applications “live”, online.
- Advanced support for memory management, resource management, distributed messaging, and POSIX features.

![Diagram of protection domains architecture]

**Figure 8: Illustration showing the protection domains architecture**

VxWorks AE is designed to work with microprocessors that have full MMUs. Processors with full MMU hardware support include most modern members of the Intel® Architecture, MPC (Motorola and IBM), ARM®, and MIPS® processor families.

**Tools**

*Tornado tools III* is the development environment for VxWorks AE. It enables developers to accelerate the configuration, development, and debugging of VxWorks AE applications. The toolset is bundled with the VxWorks AE RTOS and has been optimised to allow developers to take advantage of the new ‘protection domain’ model employed by VxWorks AE and other RAS features.

**Users of VxWorks AE**

Unfortunately no users were found.
Part I - Method

3.1.4 QNX Neutrino (6.2)/Momentics Development suite by QNX Software Systems

Product description

QNX Neutrino is the latest RTOS from QNX Software Systems [10]. It is essentially an upgrade of the QNX RTOS 6.1. It is designed to support multiple platforms such as MIPS, PPC, StrongARM, x86 and SH4. QNX Neutrino features a modular, user configurable kernel. Features such as hardware protection and a scheduler are optional.

QNX Neutrino is engineered from the ground up for the latest POSIX 1003.1 standards and drafts, including real-time and thread options. Neutrino provides an extensible framework for the creation of reliable, scalable, and high-performance embedded products.

Tools

QNX Momentics Development suite offers tightly integrated development tools and a rich complement of timesaving source kits and board support packages [11]. It targets the most popular processors (x86, MIPS, PowerPC, ARM, StrongARM, XScale, SH-4), using your choice of language (C, C++, Embedded C++, Java) and development host (Windows, Solaris, or self-hosted on QNX Neutrino).

There are two different packages to choose from:

- **QNX® Momentics® Professional Edition**
  The most comprehensive suite, including an integrated development environment (IDE) and BSP source for a large selection of target environments. QNX Momentics Professional Edition is ideal for commercial developers working with large teams on multiple projects.

- **QNX® Momentics® Standard Edition**
  Designed to get you started with basic commercial development for specific reference platforms. It allows you to complete smaller, simpler projects cost-effectively by means of a standard, command-line GNU tool chain.

Users of QNX

<table>
<thead>
<tr>
<th>Company</th>
<th>Product</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cogent Real-time Systems [17]</td>
<td>AirNav navigation system</td>
<td>Used in helicopters</td>
</tr>
<tr>
<td>Scheidt and Bachmann [18]</td>
<td>Multimedia Gas Pump</td>
<td>-</td>
</tr>
</tbody>
</table>
Part I - Method

3.1.5 LynxOS/CodeWarrior by Lynux Works

Product Description:

LynxOS is a UNIX-compatible POSIX-conformant multi-process and multi-threaded real-time embedded operating system [3]. Because of its open API, LynxOS embedded application developers achieve the fastest time to market. LynxOS has hard real-time response while supporting a variety of services and system loading without performance degradation. LynxOS is scalable, pre-emptive, and Romable. Integrated MMU process protection and virtual memory support promotes reliable, high availability embedded applications.

Tools

LynuxWorks provides a set of powerful tools designed to speed development time and bring products to market more quickly with better performance and quality.

- **VisualLynx**

- **CodeWarrior IDE edition**
  A powerful tool for embedded systems. Consists of an editor, code browser, compiler, linker, debugger, and intuitive GUI.

- **Spyker**
  A kernel trace analyser. Kernels can be dynamically instrumented at runtime.

- **Aphelion Java Toolkit**
  Aphelion eliminates the obstacles common for the use of Java in an embedded environment.

- **TotalView**
  A powerful debugger.

- **LynxInsure++**
  Improves software reliability by verifying memory references and programming implementation. Bugs can be pinpointed quickly during the development and integration phase.

- **Messenger**
  Implements a message based communication protocol for intelligent devices to cooperatively process information.

Users of LynxOS

<table>
<thead>
<tr>
<th>Company</th>
<th>Product</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>Royal Danish Navy</td>
<td>Command, control and communication system</td>
<td>-</td>
</tr>
<tr>
<td>Innovate Concepts</td>
<td>Networking Communication System</td>
<td>For US Army Aviation Helicopters</td>
</tr>
<tr>
<td>Hewlett-Packard</td>
<td>Laser Jet printer</td>
<td>-</td>
</tr>
</tbody>
</table>
Part I - Result

4 RESULT

Using the information gathered, some conclusions can be drawn for each RTOS. A final evaluation of the products is made, showing whether they are suitable for Saab Avionics applications or not.

4.1 Integrity

Advantages

- Saab Avionics is already in close contact with Green Hill
- Possibility to get DO-178B, level A certification
- Development tools that supports Ada which also meets the demands can also be provided by Green Hill
- Memory protection
- Royalty free
- ARINC 653

Disadvantages

- No negative points have been found among the information gathered

Difficulties in finding non-biased information have prevented a thorough research of Integrity RTOS. However Integrity can meet all demands set by Saab Avionics. Lockheed Martin, who is known for building military aircraft, chose Integrity for their Joint Strike Fighter. Their demands are the same as Saab Avionics and they are very satisfied with the product [24]. Also worth mentioning is that Green Hill has grown rapidly under 2001 and has become the second largest RTOS provider in the world.

Green Hill is one of the few vendors providing their own Ada compiler that also is compliant to other RTOS such as VxWorks.

This is a good solution as all software and support can be obtained from the same company. The Integrity RTOS with its AdaMulti toolkit is very well suited for avionics applications.
Part I - Result

4.2 VxWorks

Advantages
- Good development environment with an extensive set of tools
- Extensive application program interface, API
- Predictable performance
- Professional technical support
- Supports a lot of different platforms
- Several ADA compilers available
- Possibility to reach DO-178B, Level A
- VxWorks has already been used in projects within the company

Disadvantages
- Manuals not clear
- Not fully POSIX compliant
- Poor support for communication between processes located on different processors
  - Message queues can only be used if there is a shared memory area for these processors to access
- No memory protection domains

VxWorks is one of the most widely used RTOS and is compatible with a wide variety of hardware [22]. VxWorks is a good candidate as it provides Ada support and can also satisfy the safety critical standards. The Ada compiler however, must be obtained from a different source, such as Green Hill or Ada Core Technologies.

The companies that have used VxWorks in their projects have been satisfied with the product, except for one. According to LynuxWorks homepage, Innovative Concepts Inc. changed RTOS from VxWorks to LynxOS because of difficulties in porting to VxWorks. This was a problem as VxWorks is not fully POSIX compliant [20]. This is perhaps not an important issue for Saab Avionics, but might be worth mentioning. WindRiver has been in the RTOS business since 1987 and have developed knowledge within this area. This product is well suited for Saab Avionics requirements.

4.3 VxWorks AE

Advantages
- Provides memory protection domains

Disadvantages
- Documentation quality is far from acceptable
- Poor TCP/IP performance
- Some difficulties with x86 platforms
- Interrupt handling is 4 times slower than QNX
- How to use protection domains is not documented at all

It has been difficult to obtain information of current users of AE and therefore harder to make a reliable assumption about the product. However AE is the latest RTOS from Windriver and may be assumed that improvements have been made to the original VxWorks.

As AE is only an extension of VxWorks, the pros and cons are similar except from the points mentioned above [23].

Due to lack of information it is impossible to decide if VxWorks AE is the most suitable RTOS for Saab Avionics, even if it is supposed to meet the DO-178B standards.

Note: A new version of VxWorks AE called VxWorks AE 653 is due to be released in Autumn 2002. This version will be using the ARINC 653 protection domain scheduler.
4.4 QNX 6.2

Advantages

• Modern client-server architecture, message based
• Fault tolerant distributed system
• Robust, memory protection between kernel, device drivers and applications
• Good platform support
• Fast
• Technical support

Disadvantages

• No ADA compatibility at the moment
• Only 63 task priority levels
• Documentation
• Has no DO-178B kit

QNX is a fast and reliable RTOS that seems to have got a lot of positive response from users. It was recently in a competition held by dedicated systems, where it was compared with VxWorks AE and Windows CE. QNX outperformed the others easily [29]. However, since it has no Ada compilers available and no DO-178B certification kit, it does not fulfil the requirements set by Saab Avionics.

4.5 LynxOS

Advantages

• The only OS fully conformant with POSIX interfaces [25]
• Possibility to get DO-178B, level A certification
• ADA compiler available
• Memory protection
• Fast boot times

Disadvantages

• No negative points have been found among the information gathered

Information from non-biased Internet and testing sites were difficult to get hold off, but most questions were answered after a presentation held by Jan Brandberg, SESE AB. LynxOS is able to meet the demands set by Saab Avionics. One of Lynx greatest advantages seems to be its POSIX compliance, as it is the only RTOS that is fully conformant with POSIX. This enables applications to be easily ported without any greater difficulties. There are also Ada-compilers available, as well as a kit to achieve DO-178B level-A standard.
Goal: Develop a way of testing real-time functions in a system, consisting of a BSP and an RTOS.
Part II - Theoretical background

5 THEORETICAL BACKGROUND

5.1 ADA

Ada 95 is the language chosen by Saab Avionics to be used due to its real-time and safety critical properties. Some background information will be presented to give a clearer picture of what Ada is and how it was developed.

5.1.1 What is Ada

The story of Ada goes back to 1974 when the United States Department of Defence (DoD) realized that they were spending too much money on software, especially in the embedded systems area. In the early eighties, industry and universities were involved in a competition, sponsored by DoD, where the language of the future would be specified, based on certain requirements.

The first requirement was on the technical side, the language needed to meet the requirements of large-scale, reliability-critical applications (specifically embedded real-time systems). Second, it was intended to be a common DoD language making portability easy. The winning contribution, Ada, was refined to a standard 1983 and became ISO standard 8652 in 1987. This version became known as Ada 83.

The language was named after Augusta Ada Byron, Countess of Lovelace (1815-52). Ada, the daughter of Lord Byron, was the assistant of Charles Babbage and worked on his mechanical analytical engine. In some sense she can be seen as the world’s first programmer.

Figure 9: Augusta Ada Byron

After a few years it was decided that Ada 83 should be assessed and improved. This work was initiated in 1988 and the new standard became official in 1995, called Ada 95. The Ada 95 language has both increased the complexity of the tasking features and provided the means by which subsets of these features can be defined. To all of the Ada 83 features (dynamic task creation, rendezvous, abort), additions have been made, such as protected objects, Asynchronous Transfer of Control (ATC), task attributions, finalization, requeue, dynamic priorities and various low-level synchronization mechanisms.

Ada implementations, tools, and libraries are available on a wide variety of platforms through a number of vendors. The language is heavily used in industries such as transportation. Ada offers more security than languages such as C and C++, and better efficiency and a simpler run-time model than Java.
5.1.2 The advantages of using Ada 95

- **Portability**
  The language has become a standard and there exists an extensive test suite that can validate if the compiler is a real Ada-compiler. This test-suite used to be called ACVC (Ada Compiler Validation Capability), but has been replaced by ACATS (Ada Conformance Assessment Test Suite).

- **Reliability**
  The Ada language is designed in such a way that the compiler can perform a rigorous test on the code.

- **Readability**
  The program code is very easy to read and understand. This makes it easier for several programmers designing the same application to make changes in the code. Although Ada is easy to comprehend, it is still a very powerful language.

- **Abstraction**
  It is possible to design a very complex system and still maintain its simplicity.

- **Modularity**
  There are a wide variety of tools to create complicated systems, such as procedures, functions, packages, tasks, exceptions etc.

- **Object-orientation**
  Ada 95 has full support for object-oriented programming.

- **Real-time features**
  Concurrent processes (tasks) is used to simulate real-time applications.

5.1.3 Ada run-time library

When creating tasks in Ada, for concurrent purposes, some sort of scheduler is needed. All Ada compilers rely on some run-time library to implement this concurrency. There are Ada run-time libraries that map onto multithreading APIs such as Pthreads, in which case the resulting program will run as a task under an OS (e.g. Unix), but there are also Ada run-time libraries that run on bare hardware, i.e. no OS is required (in a way, the run-time library is the OS in this case). An example of a run-time which can run straight away on hardware is Green Hill’s GSTART.

5.1.4 GSTART

The main idea behind Green Hills' Safe-Tasking Ada Run-Time System, GSTART, is to provide a tasking model that supports determinism and schedulability analysis [31]. GSTART is defined by the Ravenscar Profile and contains support for a subset of the Ada 95 tasking model and prohibits dynamic memory allocation (including dynamic allocation of tasks and protected objects). This subset eliminates features, such as the rendezvous mechanism, and asynchronous transfer of control. GSTART is targeted to operate in a secure partition on Integrity RTOS-178B or as a bare target implementation.

The definition of GSTART:

- Task type and object declarations at the library level
- No unchecked deallocation of protected and task objects
- No dynamic allocation of task or protected objects
- Tasks are assumed to be non-terminating
Part II - Theoretical background

- Library level protected objects with no entries (to ensure atomic updates to shared data)
- Library level protected objects with a single entry. This entry has a barrier consisting of a single Boolean variable; moreover, only a single task may queue on this task entry
- No requeue
- No Task Aborts or Asynchronous Transfer of Control
- No rendezvous mechanism due to more efficient protected objects
- Real-time package, no reliance on the Calendar package
- Atomic and volatile pragmas
- ‘Delay until’ statements. Ada 83 relative delays are not allowed
- Count attribute for protected entries (but not within entry barriers)
- Task identifiers
- Task discriminants
- All priorities are static
- Protected procedures as interrupt handlers

5.1.5 Ravenscar

One of the goals of the 8th International Real-Time Ada Workshop, which was held in 1997 in Ravenscar, Yorkshire, England, was to define a safe tasking model for Ada. The outcome of this work is known as the “Ravenscar profile” [32].

As stated earlier, GSTART uses the Ravenscar profile, which defines a simple subset of the tasking features of Ada in order to support efficient, high integrity applications that need to be analysed for their timing properties [33]. It is restricted enough to allow a simple, efficient, and certifiable implementation, yet powerful enough to express common real-time idioms such as periodic and event-driven activities.

The Ravenscar profile can be supported by a relatively small run-time. It is reasonable to assume that the purpose-built run-time (supporting only the profile) would be efficient and “certifiable”.

With the profile, each task should be structured, as an infinite loop within which is a single invocation event. This is either a call to “delay-until” or a call to a protected entry.

It is necessary for the run-time system (RTS) to guarantee that the functional behaviour of the task will not be affected by interrupts or preemption.

The use of the Ravenscar profile allows timing analysis to be extended from just a prediction of the worst-case behaviour of an activity to an accurate estimate of the worst-case behaviour of the entire system. The computational model embodied by the Ravenscar profile is very simple and straightforward. The model does not include, for example, the rendezvous or the abort, and hence does not allow control flow between tasks (other than by the release of a task for execution in the event triggered model).
The Ravenscar profile is defined by the following:

- Task type and object declaration at the library level - No hierarchy of tasks and hence no exit protocols needed for blocks and sub programs
- No unchecked deallocation of protected and task objects - Removes the need for dynamic objects
- No dynamic allocation of task or protected object - Removes the need for dynamic objects
- Tasks are assumed to be non-terminating - This is because task termination is generally considered to be an error for a real-time program, which is long-running and defines all of its tasks at start-up
- Library level protected object with no entries – These provide atomic updates to shared data and can be implemented simply
- Library Level Protected Objects with a single entry – Used for invocation signalling, but removes the overheads of a complicated exit protocol
- Barrier consisting of a single Boolean variable – No side effects are possible and exit protocol becomes simple
- Only a single task may queue on entry – Hence no queue required, this is a static property that can easily be verified, or it can lead to a bounded error at run time
- No requeue – Leads to complicated protocols, significant overheads and is difficult to analyse (both functionally and temporally)
- No abort or ATC (Asynchronous Transfer of Control) – These features lead to the greatest overhead in the run time system due to the need to protect data structures against asynchronous task actions
- No use of the select statement – Non-deterministic behaviour is difficult to analyse, moreover the existence of protected objects has diminished the importance of the select statement to the tasking model
- No use of task entries – Not necessary to program systems that can be analysed, it follows that there is no need for the accept statement
- “Delay until” statement but no “delay” statement – The absolute form of delay is the correct one to use for constructing periodic tasks
- “Real-time” Package - To gain access to the real time clock
- No Calendar package – “Real-time” package is sufficient
- Atomic and volatile pragmas – Needed to enforce the correct use of shared data
- Count attribute (but not within entry barriers) – Can be useful for some algorithms and has low overhead
- Ada.Task_Identification - Can be useful for some algorithms and has low overhead, available in reduced form (no Abort_Task or task attribute functions Callable or Terminated)
- Task discriminants - Can be useful for some algorithms and has low overhead
- No user-defined – Introduces a dynamic feature into the run-time that has complexity and overhead
- No use of dynamic priorities – Ensures that the priority assigned at task creation is unchanged during the task’s execution, except when the task is executing a protected operation
- Protected procedures as interrupt handlers – Required if interrupts are to be handled
5.2 RTOS or Ravenscar Ada

After reading about Ravenscar and believing that this is the solution for all real-time applications, an RTOS might seem unnecessary. This is of course not entirely true as there are a lot of functions only available to the RTOS and not to a Ravenscar run-time such as GSTART. To make a comparison of the two, Green Hill’s Integrity RTOS and GSTART runtime will be used.

Here are a few examples why an RTOS should be considered instead of just using a Ravenscar product:

- Integrity supports memory protection using the MMU in the processor. This prevents a task from writing information in the same memory for another task, or even writing over the kernel memory. This will also protect against viruses and badly written code.
- When using the RTOS, it can guarantee CPU time for every task. A task cannot use the processor a longer time than it is allowed and hence corrupt the execution of a program.
- The RTOS supports Run Mode debugging using the Ethernet. This makes it possible to debug in real-time.
- Using the RTOS, it becomes a lot easier to add functionality such as TCP/IP, stack and other types of network components.
- Integrity supports distributed systems.
- With Integrity, Field Upgrade can be used, i.e. the software can be updated out on field using different protocols.
- The RTOS supports MULTI task debugging.

These are just a few examples but the list can become very long. Using an RTOS, a wide variety of tools can be used to simplify debugging and adding functionality to the program. It is important to have a clear picture of the features needed in the application, before choosing an RTOS or just a run-time such as GSTART.

It is also possible to combine GSTART with Integrity, which can be certified to safety standard DO-178B, level A.

5.3 Ada Test Suite

As mentioned earlier there exists a test suite, which is designed to test different Ada compilers. The Ada Conformity Assessment Test Suite (ACATS) is the official test method used to check conformity of an Ada implementation with the Ada programming languages standard (ANSI/ISO/IEC 8652:1995) [35]. This standard provides a framework for testing language processors, providing a stable and reproducible basis for testing. It is managed by the Ada Conformity Assessment Authority (ACAA). Before it became an ISO standard, the US DoD sponsored a similar assessment process, this test suite was known as the Ada Compiler Validation Capability (ACVC). When the ACAA took over the conformity assessment, it adopted the ACVC as the basis for its test suite. The ACAA determined to continue to use the name version numbering for the test suite in order to avoid confusion. The ACVC version 2.1 became ACATS 2.1.

ACATS 2.3 contains test programs to check for conformity to new language features defined in Ada 95, as well as tests program to check for conformity to language features shared between Ada 83 and Ada 95.
5.4 Purpose of ACATS

The intention of ACATS is to check whether an Ada compilation system is a conforming implementation, i.e. it produces an acceptable result for every test in the test suite. The fundamental goal of the validation is to ensure that the processing of the Ada language is the same in different compilers, making it portable. The ACATS test use language features that are expected to exist in production software. While the tests exercise a wide range of language features, they cannot include examples of all possible feature uses and interactions.

There are some things that ACATS cannot do:

- Does not guarantee that the compiler is correct, as it can only verify the features used in the test suite
- Does not check or report performance parameters such as compile-time capacities or run-time speed
- Does not check or report characteristics such as the presence and effectiveness of compiler optimisation
5.5 What is a BSP?

A Board Support Package (BSP) is a package containing drivers, used to configure an operating system for a given hardware. The BSP contains support for such board-level functionality as timers, Ethernet, serial, and other devices. Also included is an OS BSP, which is the partial set of boot code and drivers that operate the RTOS, and the test BSP, which is needed by hardware engineers to test a board without an operating system.

![BSP Diagram](image)

**Figure 10: The BSP in the structure of an embedded system**

5.5.1 Structure of a BSP

The general structure of the BSP includes the following:

- **Setting up the CPU**
  When the CPU comes out of a reset, its registers must be set to allow it to function properly.
- **Setting up memory**
  Identifying where different parts of memory are located, such as stack, data etc.
- **Setting up interrupt**
  Interrupt controller should be set to handle OS and I/O interrupts.
- **Setting up the operating system scheduler timer**
  The timer must be set to provide to provide time ticks for the scheduler to handle task switching.
- **Setting up the communication drivers for the OS debug channel**
  The debug channel is the connection between the host and the target. The debug channel is usually a serial port or an Ethernet channel.
- **Setting up the application specific drivers for the board**
  These drivers are not required for the OS to operate, but are required by the application. The drivers could be for timers or serial ports.
- **Running the OS**
  At some point, the OS is started and the first task is launched.
- **Running the application**
  When the OS is ready the application task is launched.
5.5.2 How is a BSP developed

Creating a BSP is a very time consuming task as it can take several months to construct. As the complexity of the hardware increases, the longer time it takes. To be able to develop the BSP it would be necessary for the developer to take part in the OS vendors’ BSP training courses, as the developer needs to fully comprehend all the important parts of the OS and the hardware. It is possible to purchase a BSP for a specific processor and then adapt it for the hardware. This can still be very time consuming.

However, a new way of creating BSPs has begun to emerge, called driver synthesis. Driver synthesis automatically creates BSPs for both evaluation- and target boards. A point- and click interface is used to specify the design requirements and parameters. A program called DriveWay is one of the first driver synthesizers. This program will allow a BSP to be created in one to three weeks instead of several months, when doing it without the help of the synthesis tool [26, 27].

In the beginning it was planned that a BSP would be set up in this work but now, after realizing the work needed to do this work, an already fully functional BSP will be used.

5.5.3 How to test a BSP

The execution of new BSP code on a processor can, and often does bring unexpected results, even if the code is expected to be error free. Hang-ups, mysterious interrupts, and data transfer errors are just a few examples of what can happen when using a newly designed BSP.

A new BSP should always be tested using a written and clearly identified test specification. The specification should cover all possible combinations in a system such as bus speeds, data size and transfer speeds, caches on and off, polling and interrupt modes, and error checking. Software, especially when developed for a new hardware platform, is rarely error-free and needs a good testing specification to identify and resolve problems.

A number of vendors and independent organizations provide validation service for BSPs. WindRiver supplies a Validation Test Suite, in source form, along with its Tornado BSP Developer’s Kit. Once the BSP has successfully passed the test suite in the toolkit, WindRiver will provide paid verification and compliance certification for BSPs written for VxWorks RTOS. In most cases, RTOS vendors provide at least a porting guide, as well as tools similar to those provided by WindRiver. However these Test Suites costs a lot of money and this is why a part of this project is to find a way of testing some basic functions, which are controlled by the BSP.
6 METHOD

A BSP can be very large and complex, and designing a test program to test all its features can take a very long time. This is probably why buying a test program from RTOS vendors with its BSP can become expensive. However a test that verifies some basic functions can be created, even though it will not be as advanced as the test suites provided by the RTOS vendors.

6.1 What will be tested

The main goal of this assignment has focused on RTOS and their real-time features. The test suite, that is going to be developed, will also focus on this area. The real-time features, when downloaded to hardware, will depend upon a correctly set up BSP and a fully functional RTOS. The aim of this test suite is to create a number of tests, verifying that different real-time features, such as timing, multitransfer, priorities, etc. works correctly with the existing BSP.

6.2 Chosen test method

One possible solution when trying to create a test suite is by using the already existing test suite for Ada compilers, ACATS. A lot of time and effort has been put in to producing this test suite. It contains thousands of tests, divided up into different test areas. They are quite complex, and examine the different features thoroughly. It requires a lot of expertise and experience to correctly set up the complete ACATS test suite and it is not recommended for anyone to try to set it up on their own [36]. However in this assignment only a handful of tests, focusing on real-time capabilities, have been picked. These tests are not as difficult to set up as the rest of the test suite. They only require a few extra programs to function properly. Hopefully these tests will be a good way of testing the real time features of the BSP and RTOS. 

There are a few different specialised annexes. These annexes are optional in the ACATS test and many implementations will not support them in full. The real-time annex, which will be focused on, covers priorities, scheduling etc. [34]

6.3 Getting started

The first idea for the assignment was to use ACATS for testing a homemade BSP together with GSTART. As explained above, the intentions are not to provide a test suite to test the whole system, as this would be too time consuming. However, a test suite that will test the real-time functions should, hopefully, be the final result of this project. 

Certain ACATS test should be able to be downloaded to hardware, and be run together with an RTOS. Although the initial thought with the ACATS test suite was to create a way to set a standard for Ada compilers, it is also believed that it will be possible to test certain parts of a BSP as well.

At first, AdaMulti 2000 was intended to be used with an Ada run time called GSTART, together with a board containing a Motorola processor, MPC 8245. However the target board became difficult to gain access to, as it was involved in another project. Therefore a new piece of hardware was provided. Another type of Motorola processor called MPC 604 was included in this board, which was already set up to use the VxWorks RTOS. AdaMulti 2000 was supposed to be the environment used to provide downloadable programs for the RTOS, but the documentation did not contain much useful information about how this should be done.
Part II - Method

This information might have been badly documented, as VxWorks is not Green Hills own RTOS.
After some time, it also became apparent that the ACATS test suite would not have been possible to use with GSTART. This was due to the fact that Ravenscar has strict rules about how code is written, preventing Ada programs being written in an “ordinary” way.
As the new hardware was set up to use VxWorks, and it was difficult to get AdaMulti to create and download programs to this hardware, another option seemed easier to implement.

This option was to use GNAT pro integrated with the Tornado environment, as an evaluation license already had been provided. GNAT pro is another type of Ada95 compiler, and it is especially well suited for integration with Tornado. This compiler does not use the Ravenscar model, making ACATS available again.
The software tools, programs and hardware that will be used are:
- MPC 604 Motorola processor
- GNAT pro Ada 95 compiler integrated with VxWorks
- VxWorks RTOS
- Tornado II development environment

Note: Installation of GNAT Pro is given in Appendix E and the connection set up between the hardware and Tornado environment is described in Appendix F
6.4 The Test Suite

6.4.1 The ACATS test suite

After searching for the ACATS test suite, the version 2.3 was downloaded [35]. The folder containing the tests for the real-time annex was studied closely and several tests were selected. All the tests in the test suite have names that describe what chapter in the Language Reference Manual they cover. This will be explained later.

To make it easier and faster to verify a certain feature in the suite, the tests were divided up into small groups. The tests are shown below:

<table>
<thead>
<tr>
<th>Test num.</th>
<th>Synchronous transfer</th>
<th>Clock</th>
<th>Daisy</th>
<th>Exception</th>
<th>Floating points</th>
<th>Interrupt</th>
<th>pragma</th>
<th>Preemption</th>
<th>Priorities</th>
<th>Procedures</th>
<th>Protected object</th>
<th>Queuing</th>
<th>Rendezvous</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>C940005</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C940006</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C940007</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C940011</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C940012</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C940013</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Test name: C94 (Protected units and protected objects)

<table>
<thead>
<tr>
<th>Test num.</th>
<th>Synchronous transfer</th>
<th>Clock</th>
<th>Daisy</th>
<th>Exception</th>
<th>Floating points</th>
<th>Interrupt</th>
<th>pragma</th>
<th>Preemption</th>
<th>Priorities</th>
<th>Procedures</th>
<th>Protected object</th>
<th>Queuing</th>
<th>Rendezvous</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>C940005</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C940006</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C940007</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C940011</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C940012</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C940013</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Test name: CXD1 (Task Priority)

<table>
<thead>
<tr>
<th>Test num.</th>
<th>Synchronous transfer</th>
<th>Clock</th>
<th>Daisy</th>
<th>Exception</th>
<th>Floating points</th>
<th>Interrupt</th>
<th>pragma</th>
<th>Preemption</th>
<th>Priorities</th>
<th>Procedures</th>
<th>Protected object</th>
<th>Queuing</th>
<th>Rendezvous</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>CXD1001</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CXD1002</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CXD1003</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CXD1004</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CXD1005</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CXD1006</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CXD1007</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CXD1008</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Test name: CXD2 (Priority Scheduling)

<table>
<thead>
<tr>
<th>Test num.</th>
<th>Synchronous transfer</th>
<th>Clock</th>
<th>Daisy</th>
<th>Exception</th>
<th>Floating points</th>
<th>Interrupt</th>
<th>pragma</th>
<th>Preemption</th>
<th>Priorities</th>
<th>Procedures</th>
<th>Protected object</th>
<th>Queuing</th>
<th>Rendezvous</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>CXD2001</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CXD2002</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CXD2003</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CXD2004</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CXD2006</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CXD2008</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Test name: CXD3 (Priority Ceiling Locking)
### Tests

<table>
<thead>
<tr>
<th>Test num.</th>
<th>Synchronous transfer</th>
<th>Clock</th>
<th>Daisy</th>
<th>Exception</th>
<th>Floating points</th>
<th>Interrupt</th>
<th>P精美</th>
<th>Preemption</th>
<th>Priority</th>
<th>Procedure</th>
<th>Protected object</th>
<th>Queuing</th>
<th>Rendezvous</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>CXD4001</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CXD4002</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CXD4003</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CXD4004</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CXD4005</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CXD4006</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CXD4007</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CXD4008</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CXD4009</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CXD4010</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Test name: CXD4 (Entry Queuing Policies)**

<table>
<thead>
<tr>
<th>Test num.</th>
<th>Synchronous transfer</th>
<th>Clock</th>
<th>Daisy</th>
<th>Exception</th>
<th>Floating points</th>
<th>Interrupt</th>
<th>P精美</th>
<th>Preemption</th>
<th>Priority</th>
<th>Procedure</th>
<th>Protected object</th>
<th>Queuing</th>
<th>Rendezvous</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>CXD5001</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Test name: CXD5 (Dynamic Priorities)**

<table>
<thead>
<tr>
<th>Test name: CXD6 (Preemptive abort)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test num.</td>
</tr>
<tr>
<td>-----------</td>
</tr>
<tr>
<td>CXD6001</td>
</tr>
<tr>
<td>CXD6002</td>
</tr>
</tbody>
</table>

**Test name: CXD8 (Monotonic time)**

<table>
<thead>
<tr>
<th>Test name: CXD9 (Delay Accuracy)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test num.</td>
</tr>
<tr>
<td>-----------</td>
</tr>
<tr>
<td>CXD9001</td>
</tr>
</tbody>
</table>

**Test name: CXD10 (Synchronous Task Control)**

<table>
<thead>
<tr>
<th>Test num.</th>
<th>Synchronous transfer</th>
<th>Clock</th>
<th>Daisy</th>
<th>Exception</th>
<th>Floating points</th>
<th>Interrupt</th>
<th>P精美</th>
<th>Preemption</th>
<th>Priority</th>
<th>Procedure</th>
<th>Protected object</th>
<th>Queuing</th>
<th>Rendezvous</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>CXDA001</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CXDA002</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CXDA003</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Shows selected ACATS tests and features covered

A more detailed description of the tests is given in *Appendix G.*
**Part II - Method**

### 6.4.2 ACATS naming

The naming in the ACATS test suite indicates what part of the Language Reference Manual they cover. The name of an Ada 95 test is composed of seven or eight characters. This naming scheme was invented when some operating systems had short file-name limits. The use of each character is shown below:

![Diagram showing the structure of an ACATS test name](image)

**Position (from the left)**

1. **Letter**
   - Test class: foundation are marked ‘f’
2. **Alphanumeric**
   - An ‘x’ indicates that the test includes one or more features from an annex
3. **Alphanumeric**
   - Core clause or annex letter identifier
4. **Hexadecimal**
   - Sub-clause (if a core test), or clause (if an annex test)
5. **Alphanumeric**
   - Foundation identifier (alphabetic, unless no foundation is required, in which case a ‘0’)
6-7. **Decimal**
   - Sequence number of this test in a series of tests for the same clause foundation code will have “00”
8. **Alphanumeric**
   - Optional – compilation sequence identifier. This position is used only if the consists of multiple files

**Example:**

CXD1001.a: The first position means that the test is a C test (execution time test)
   The second and third position indicates that the test is in annex D
   Fourth position describes which chapter in [Ada 95] library reference manual the test covers
   Sixth and seventh position represents the sequence number of the test series

### 6.4.3 Ada extensions

Ada code can be written in three different forms, these extensions can be *.adb, *.ads and *.ada. The extension types are described below:

- ***.ads** – Specification, shows the programs that calls on a package, information about which procedures that exists in the package.
- ***.adb** – Body, Contains the execution code.
- ***.ada** – A combination of the two above

All tests in the ACATS test suite are written in the form *.ada.
Part II - Method

Unlike AdaMulti 2000, GNAT pro cannot compile files in this form and needs to be split up into two pieces (*.adb and *.ads), when using packages. At first this was done manually, but after realising that this took a lot of time and became tedious, an alternative way of doing this was necessary. A helpful tool was found in the GNAT compiler, called “gnatchop”. This tool could split the Ada file to *.ads and *.adb on its own, greatly simplifying this procedure.

6.4.4 Running the tests

To be able to run the tests, some extra source files were needed to allow the program to compile, as these programs contained some information required by the tests. The files needed are called:

- Repspec.ada and Repbody.ada – Was renamed to REPport.ads and REPport.adb.
  All the tests use this support package. It contains routines to automate test result reporting as well as routines designed to prevent optimisers from removing key sections of the test code.
- IMPDEF.ada - Was renamed to Impdef.adb.
  The ACATS tests use the entities in Impdef to control test execution. Much of the information relates to the timing of running code.
- Impdef_c.ada - Was renamed to Impdef.annex_c.adb and Impdef.annex_c.ads.
  Contains instructions for certain real-time tests.
- Impdef_d.ada - Was renamed to Impdef.annex_d.ads.
  Contains instructions for certain real-time tests.

These files could be found under the “support” folder in the ACATS test. The values for these programs were left at their default value.

After these files were included and the *.ada files had been changed to *.adb and *.ads, single tests could be compiled and built. If the hardware and Tornado was set up correctly, according to appendix F, the program could now be downloaded to hardware.

6.4.5 Running several programs

To allow more than one test to run automatically, a main program was created. This program was written in such manner that it could call for several tests during its execution. This required however that all the tests needed to be rewritten so that they became packages, as only one main procedure was allowed to run.
The main program was set up according to the structure below:

```ada
with Ada.Text_IO; use Ada.Text_IO; -- Enables PUT_LINE function
with test1; use test1; -- What package to use
with test2; use test2; -- "-

procedure Start_file is
begin
   put_line("*************** - Test 1 starting - ****************");
   put_line("");
   first_test, -- Call for test package 1
   put_line("");
   put_line("*************** - Test 2 starting - ****************");
   put_line("");
   second_test;
   put_line("");
   ...
   put_line("---");
   put_line("*************** - Test suite finished - ****************");
   put_line("");
end;
```

Figure 11: Ada - main program example

6.4.6 Problems

The Ada programs, when downloaded to the hardware, will only be able to be run once and then need to be reloaded. This is because the program changes, as it runs and will not be the same when the program runs again.

Some of these tests need to be run separately (marked with “RS”). They set up different values, which can only be done in the main program. Some of these tests require that these settings are set up differently, and cannot be run in the same program. The compiler did not find any errors during compilation. When this problem became apparent to the technical support at GNAT pro, they decided to add a warning message to the future versions of GNAT pro.
Part II - Method

6.4.7 The Complementary Test Suite

Several tests were created written in C as a complement to the ACATS test suite. These tests make it possible to verify the correctness of the following functions:

**Priority inversion** - The example illustrates a typical situation in which priority inversion takes place

**Multi threading** - This program creates 10 different tasks and then prints out the id for each task

**Semaphores** - This program creates 2 tasks, which share a semaphore. One task increases a variable, whilst the other decreases it

**Signalling** - Shows how signals can be sent and received in tasks

**Round-Robin scheduling** - Creates 3 tasks, which are scheduled according to time slices. The tasks are given a certain amount of time to run, before another task takes over and runs the same amount of time

**Message Queues** - A couple of messages are sent between two tasks, using message queues

**Priority Scheduling** - Creates 3 different task with different priorities

**Timing** - This test reveals the system clock rate and allows the user to find out how many ticks are needed to delay a certain amount of time

**Memory (Ada)** - Verifies that the RAM parameters have been set up correctly. This is done by viewing the address contents, at the beginning and the end of the defined memory area

These codes with their result can be seen in Appendix H.

Each test returns true or false, depending on the actual result. The outcome of the tests can be observed in the hyper terminal when connected to the hardware. All of these tests can be run separately, but there also exists a program called “C_testsuite.cpp”. This program calls on all the above-mentioned tests (except Memory). Memory needs to be run on its own.
6.5 Modifying the BSP

To verify the created test suites, changes can be done to the BSP to simulate errors. How to make these changes will now be explained.

When the hardware is switched on, the kernel with its BSP is downloaded from a predefined location. In this case the kernel is located at the host computer at: c:\Tornado\target\config\mv2604\VxWorks.

The BSP contents can be changed in the Tornado II environment, by creating a bootable project and choosing BSP in one of the options. This will allow the user to make specific changes in the BSP, through a point- and click interface.

![Image of example changes in BSP](image.png)

Figure 12: Example of how changes in the BSP can be made

After the changes have been built, the old kernel in the directory above needs to be replaced with the modified version. The hardware is then rebooted, to download the new BSP.
7  RESULT

7.1  Verification of the test suites

The part of the test suite that tests the RTOS cannot really be proven, as it is not possible to change the contents of the OS kernel. However by changing the BSP and then by running the test programs with this BSP, errors should hopefully be detected.

7.2  The changes made

The CPU runs all computations, but uses some direct helpers to work more efficiently. These units could be changed or excluded from the BSP without interfering with vital settings to the RTOS.

These units are:
- Floating Point Unit – Used for floating-point calculations
- Caches – holds instructions or data close to the CPU
- Memory Management Unit – Controls memory accesses
- Timers – CPU timing

Unfortunately more modifications could not be done without causing system failure.

The changes made and the names of the tests that failed due to this change are stated below.

7.2.1  Floating point

There exists a register, which makes floating-point calculations possible. This register was disabled and the modified BSP was tested.

The CXD1008 ACATS test failed due to this modification, which was not surprising as it tests floating-point calculations.

7.2.2  Cache

There are two different options that can be modified that affect instruction- and data caches. These were the following:
- “USER_I_CACHE_SUPPORT” - instruction cache mode
- “USER_D_CACHE_SUPPORT” - data cache mode

These were initially enabled.

By disabling the instruction cache support, a few unacceptable results became present in the test suite. These tests were the following:
- Priority inversion  - C_TestSuite
- Priority Scheduling - C_TestSuite
- Signalling - C_TestSuite
- Round Robin - C_TestSuite

Although the tests were able to produce the correct response, the execution time was significantly increased (x10). Each of the above mentioned tests took over thirty seconds to complete.

The ACATS tests were not affected by these changes.

The data cache support was also disabled. The outcome of the tests, were exactly the same as above.
Part II - Result

7.2.3 MMU

There are two registers, which can be enabled or disabled. These are:
“USER_D_MMU_ENABLE” – Data MMU
“USER_I_MMU_ENABLE” – Instruction MMU
These were initially enabled.

The data MMU was disabled and the following tests were affected:
Priority inversion - C_TestSuite
Priority Scheduling - C_TestSuite
Signalling - C_TestSuite
Round Robin - C_TestSuite
These tests were affected in the same manner as in the cache test, when the MMU was disabled. It was later found that the cache support uses the MMU and by disabling this function, cache also became disabled [39].

Disabling the instruction MMU did not cause any errors. The only conclusion that can be drawn from this is that the instruction MMU does not affect the cache in the same way as stated above.

7.2.4 Clock rate

The value of the system clock can be changed in the BSP. The test program “Time_testing” returns the value of system clock rate, so that the user can see it. As the test program is written in C, the delay function is written in terms of ticks per second. By knowing the system clock rate, a certain time delay can be set and observed. This test can identify any faults with the timing properties.
8 CONCLUSION

The first investigation has given a good overview of what the RTOS market has to offer. We found a few RTOS that can pass the demands required by Saab Avionics. VxWorks seemed to be the most widespread and well-known OS and we believe that this product is a good choice for the company. During this thesis, we got the opportunity to work with this product. Tornado II was very easy to use and worked very well together with Gnat Pro.

Green Hills software, AdaMulti and Integrity, also seems to be very good products. Unfortunately, Green Hill was unwilling to provide an evaluation license for their RTOS (Integrity). We did, however, get a chance to use AdaMulti together with GSTART. AdaMulti was not as user-friendly as VxWorks but contained good debugging tools. Green Hill can provide all tools necessary to comply with Saab Avionics requirements. This makes it possible to get all the technical support from the same vendor.

Lynx OS can also fulfill the requirements. One of its best characteristics is its POSIX compliance, which can facilitate portability.

QNX seems able to provide a very good RTOS. It was involved in a comparison test, with VxWorks and Windows CE. It outperformed both of these and has got a good reputation among people using it. Unfortunately, the vendor could not offer an Ada compiler or a DO-178B certification kit. Due to this, QNX is not able to meet the demands required by Saab Avionics, at this time.

There are probably a few more RTOS, which might be suitable for the company. However, the products we have chosen to investigate are well known and possess a lot of experience. We believe that this makes them the ideal choice, as they have been tested and verified over several years.

The second investigation resulted in two test suites. These test suites were created to verify a system, containing a BSP and an RTOS. They were named: the ACATS test suite and the Complementary Test Suite.

The ACATS test suite was the part that was given the most attention. It became an interesting idea to see if it was possible to use ACATS to verify a system. It was clear after this assignment that it was an ineffective way of verifying errors that might occur in a BSP, but a good way of testing the RTOS.

A floating-point modification was the only BSP error that ACATS was able to discover during this investigation. This is because the test suite focuses on specific behaviour in the RTOS, which are not influenced by the BSP.

The complementary test suite was written with the aim to create some real-time specific programs. It was meant to verify that problems such as priority inversion would not occur with the chosen RTOS. These tests were more easily affected by changes made to the BSP. The execution time of certain tests became very long, if the cache or the MMU was not set up correctly. Even though the test suite can find some errors in the BSP, it needs additional tests to pinpoint where the errors are located.

We believe that the test suites created can be a useful way of checking that the real-time features work correctly.
8.1 Future plans for project

Several ideas for future projects:
- Further tests for cache and MMU can be developed and added to the existing test suite.
- Create a test suite to verify UART, serial communication, read/write to memory etc.
- An evaluation of driver synthesis tools could be an interesting assignment, as it seems to be a future market.
9 REFERENCES AND LITERATURE

9.1 References

Access date: 2002-11-07

Access date: 2002-11-07

www.realtime-info.be/encyc/buyersguide/rtos/Object98.html
Access date: 2002-11-07

www.recab.fi/Mappi/RTOS/VxWorks.htm
Access date: 2002-11-07

[5] Wind River, VxWorks 5.x,
www.windriver.com/products/html/vxwks5x_ds.html
Access date: 2002-11-07

[6] Rational, For Avionics and DO-178B Software Development,
www.rational.com/products/testrt/market/avionics.jsp
Access date: 2002-11-07

www.windriver.com/pdf/t_dol178b_ds.pdf
Access date: 2002-11-07

www.windriver.com/pdf/vxworks_5e.pdf
Access date: 2002-11-07

www.windriver.com/products/vxworks_5e/faq.html
Access date: 2002-11-07

[10] QNX Software, QNX Neutrino RTOS Features and Benefits
www.qnx.com/products/ps_neutrino/features.html
Access date: 2002-11-07

www.qnx.com/products/ps_momentics/
Access date: 2002-11-07
References and literature

[12] Green Hills software, Goodrich Selects Green Hills Software’s Integrity -178B RTOS For New Engine,
www.ghs.com/news/220305i.html
Access date: 2002-11-07

www.ghs.com/news/220122m.html
Access date: 2002-11-07

[14] Green Hills software, Green Hill’s Integrity Selected As Operating System For F-35 Joint Strike Fighter,
Access date: 2002-11-07

www.windriver.com/html/x38.html
Access date: 2002-11-07

[16] Wind River, Honda Motor Co.: ASIMO Humanoid Robot,
www.windriver.com/success/asimo.html
Access date: 2002-11-07

[17] McHale John, Technology Focus, Military & Aerospace Electronics, February 2001, Real-time operating system vendors rush to comply with DO-178B,
Access date: 2002-11-07

Access date: 2002-11-07

[19] LynuxWorks, June 2002, Royal Danish Navy Selects LynxOS 4.0,
Access date: 2002-11-07

[20] LynuxWorks, Innovative Concepts and LynxOS Team Up to Support U.S. Army Communications
Access date: 2002-11-07

[21] LynuxWorks, HP Selects LynxOS for its LaserJet Operating System
Access date: 2002-11-07

[22] Dedicated Systems, June 8, 2001, RTOS evaluation project, VxWorks 5.3.1 (x86),
www.dedicated-systems.com, doc no.: DSE-RTOS-EVA-007
Access date: 2002-11-07
References and literature

[23] Dedicated Systems, May 13, 2001, RTOS evaluation project, VxWorks AE 1.1 (x86),
www.dedicated-systems.com, doc no.: DSE-RTOS-EVA-015
Access date: 2002-11-07

selects integrated development environment and fault-tolerant RTOS for F-35
Joint Strike Fighter,
www.realtime-info.be/vpr/layout/display/pr.asp?PRID=2686
Access date: 2002-11-07

www.standards.ieee.org/regauth/posix/posix2.html
Access date: 2002-11-07

[26] Gal-Oz Shaul, 1999, Driver Synthesis: Automating the Creating of BSPs,
Access date: 2002-11-07

[27] Aisys inc.,
www.aisysinc.com
Access date: 2002-11-07


and the QNX RTOS v6.1.
www.dedicated-systems.com/Encyc/
Access date: 2002-11-07

[30] Omvärldsbeskrivning och Kravbild för Säkerhetskritiska System. MOAV-
2001:040

[31] Safety Critical Products, GSTART
www.ghs.com/products/safety_critical/gstart.html
Access date: 2002-11-05

Access date: 2002-11-05

[33] Aonix, The Ravenscar Tasking Profile for High Integrity Real-Time Programs,
June 1998,
www.aonix.com/content/downloads/objectada/ravenscar2.pdf
Access date: 2002-11-05
References and literature

Access date: 2002-11-05

Access date: 2002-11-06

Access date: 2002-11-06

[37] Wind River, Tornado 2.0 Users guide, April 1999 (Windows version), page 41

[38] Wind River, Tornado 2.0 Users guide, April 1999 (Windows version), page 461

Access date: 2002-11-06

[40] ARINC specification 653, Avionics application software standard interface

9.2 Literature

Skansholm Jan, 1999, Ada från början. Studentlitteratur, Lund,


<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Programming Interface. A standard set of functions, provided by the operating system, that can be invoked from the software</td>
</tr>
<tr>
<td>ACAA</td>
<td>Ada Conformity Assessment Authority</td>
</tr>
<tr>
<td>ACATS</td>
<td>Ada Conformity Assessment Test Suite. A test suite created to verify an Ada compiler</td>
</tr>
<tr>
<td>ACVC</td>
<td>Ada Compiler Validation Capability. A test suite, now renamed to ACATS</td>
</tr>
<tr>
<td>Ada</td>
<td>Powerful programming language for safety critical applications</td>
</tr>
<tr>
<td>ARINC 653</td>
<td>Partition scheduler</td>
</tr>
<tr>
<td>BSP</td>
<td>Board Support Package. Contains drivers, used to configure an operating system for a given hardware</td>
</tr>
<tr>
<td>C</td>
<td>A programming language</td>
</tr>
<tr>
<td>C++</td>
<td>A programming language, an extension to C, that provides object-oriented features</td>
</tr>
<tr>
<td>Cache</td>
<td>A place to store data temporarily to improve computational performance</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
</tr>
<tr>
<td>DoD</td>
<td>Department of Defence</td>
</tr>
<tr>
<td>Driver synthesis</td>
<td>A way to create BSPs, using a point- and click interface</td>
</tr>
<tr>
<td>Ethernet</td>
<td>A LAN developed by XEROX in 1976</td>
</tr>
<tr>
<td>GNAT</td>
<td>Gnu New-York University Ada Translator. An Ada compiler</td>
</tr>
<tr>
<td>GNU</td>
<td>Gnu is Not Unix. GNU is a UNIX-like operating system that comes with source code that can be copied, modified, and redistributed</td>
</tr>
<tr>
<td>GSTART</td>
<td>Green Hills' Safe-Tasking Ada Run-Time System. A run-time system using the Ravenscar model</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface. A graphical representation of data</td>
</tr>
<tr>
<td>Term</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>HDLC</td>
<td>High-level Data Link Control, a group of protocols or rules for transmitting data between network points</td>
</tr>
<tr>
<td>Integrity</td>
<td>An RTOS developed by Green Hill</td>
</tr>
<tr>
<td>ISO</td>
<td>International Standards Organisation</td>
</tr>
<tr>
<td>Kernel</td>
<td>The centre of an operating system that provides basic services for all other parts of the operating system</td>
</tr>
<tr>
<td>LynxOS</td>
<td>An RTOS developed by Lynx Real-time Systems</td>
</tr>
<tr>
<td>MMU</td>
<td>Memory Management Unit. Controls memory accesses</td>
</tr>
<tr>
<td>MPC</td>
<td>Motorola Power PC. A powerful processor family created by Motorola</td>
</tr>
<tr>
<td>OS</td>
<td>Operating System</td>
</tr>
<tr>
<td>QNX</td>
<td>An RTOS developed by QNX Software Systems</td>
</tr>
<tr>
<td>Ravenscar</td>
<td>Reliable Ada Verifiable Executive Needed for Scheduling Critical Applications in Real-time. A programming profile to create deterministic code in Ada</td>
</tr>
<tr>
<td>Rendezvous</td>
<td>Synchronized communication between sender and receiver</td>
</tr>
<tr>
<td>RTCA</td>
<td>Radio Technical Commission for Aeronautics</td>
</tr>
<tr>
<td>RTOS</td>
<td>Real-Time Operating System</td>
</tr>
<tr>
<td>SAR</td>
<td>Segmentation And Reassembly, used for telecommunication networks</td>
</tr>
<tr>
<td>Stack</td>
<td>Data area or buffer used for storing requests that need to be handled</td>
</tr>
<tr>
<td>TCP/IP</td>
<td>Transmission Control Protocol/ Internet Protocol, a set of standard protocols for communicating across a single network or interconnected set of networks</td>
</tr>
<tr>
<td>UART</td>
<td>Universal Asynchronous Receiver/Transmitter, a way of communicating serially</td>
</tr>
<tr>
<td>UNIX</td>
<td>An operating system</td>
</tr>
<tr>
<td>VxWorks</td>
<td>An RTOS developed by WindRiver</td>
</tr>
</tbody>
</table>
Appendices
Appendices

11 APPENDICES

Appendix A: Integrity/Multi 2000.......................................................... A
Appendix B: VxWorks/Tornado II ....................................................... B
Appendix C: QNX/Momentics ........................................................... C
Appendix D: LynxOS/CodeWarrior.................................................... D
Appendix E: Installation of GNAT Pro.............................................. E
Appendix F: Connection to hardware .............................................. F
Appendix G: Chosen ACATS tests ................................................... G
Appendix H: The complementary test suite................................. H
APPENDIX A    Integrity/MULTI 2000

Features

- Royalty free
- Uses hard ware memory protection
- POSIX support
- Fast interrupt handling
- The kernel never masks or blocks interrupt while manipulating critical data structures.
- Real time event analyser
- DO-178B, level A certification

Development information

- Development Hosts
  Windows, Solaris, UNIX
- Supported Target Packages
  PowerPC
- Board Support Package
  MBX860/821, MVME2300, MCP750, MVME2600, SP103, MVME2700, ADS-8260, FADS860/821, ipEngine 823, CES RIOC4064, Radstone PPC4, Force Power
- Supported Compilers
  Green Hills Software’s C, C++ and Ada 95.
- Supported Tools
  Green Hills Software’s MULTI and AdaMulti Integrated Development Environments
- Supported Networks
  TCP/IP, Web Server
- Supported Standards
  POSIX
- Development Methodology
  Cross
- RTOS Supplied as
  Object, Source
- Available Components
  Floating Point, Communication, Math Library, File Support

Technical Information

- Kernel ROM (min, max):
  70 K
- Kernel RAM (min, max):
  20 K
- Minimum RAM per process:
  1 Page
- Minimum RAM per thread:
  1 Page
Appendix A

- **Minimum RAM per queue:**
  128 Bytes

- **Number of thread priority levels:**
  255 but configurable

- **Typical Thread Switch Latency:**
  50-100 clock cycles

- **Priority Inversion Avoidance Mechanism:**
  Yes, Priority Inheritance Mutex, Highest Locker Semaphore

- **Multiprocess Support:**
  Yes

- **Multiprocessor Support:**
  Yes

- **MMU Support:**
  Yes

- **Scheduling Policies:**
  Prioritized Round-Robin, ARINC 653

**Commercial Information:**

- **Royalty Free:**
  Yes

- **Standard Phone Support:**
  Paid

- **Preferred Phone Support:**
  Paid
Appendix B

APPENDIX B  VxWorks/Tornado II

Features [7]

- POSIX support
- RTCA/DO-178B and EUROCAE/ED-12B Level A Certification
- Efficient, real-time multitasking kernel
- Intertask communications mechanisms
- Resource protection mechanism
- Task management facilities
- Exception handling
- Software signals
- Floating point support
- Certifiable standard C library components including formatted I/O
- Optimised math libraries
- Error handling
- Utility functions
  - Buffer management libraries
  - Variable-length byte array manipulation
  - Doubly linked lists

Development information

- Development Hosts
  Windows NT, Solaris, SunOS, PH-UX, Win95
- Supported Target Processors
  x86, 68k, PPC, CPU 32, i960, SPARC, SPARCLite, SH, ColdFire, R3000, R4000, C16X, ARM, MIPS
- Board Support Package
  Supports a wide variety of boards and embedded computer platforms through board-support packages. BSPs are available to many manufacturers including Motorola, VMIC, Force, Intel/Cyclone and Digital.
- Supported Compilers
  Gnu C & C++, Ada, Assembler, Java.
- Supported Tools
  Tornado remote source-level debugger, browser, WindSh shell interface, incremental loader, target tool, WindConfig configuration for Board Support Packages.
- Supported Networks
  TCP/IP, NFS Client and server, No-copy TCP/IP, SNMP, STREAMS, OSI, SS7, ATM, Frame Relay, CORBA, ISDN, x.25, CMIP/GDMO, PPP, http server CSLIP, RPC, FTP, rlogin, telnet, bsmtp, SNMP.
- Supported Standards
  POSIX.1b, UNIX
- Development Methodology
  Cross
- RTOS Supplied as
  Object
- GUI Support
Appendix B

X-Windows, Motif

- **Integrated Java Support**
  Browser, applets, byte codes

- **Available Components**
  Floating Point Support, Communication Protocols, Cache Support, Network Support, Math Library, File Support

**Technical Information**

- **Kernel ROM (min, max):**
  16K, 488K

- **Kernel RAM (min, max):**
  620 bytes, 29.3 K

- **Minimum RAM per thread:**
  1000 bytes

- **Minimum RAM per queue:**
  200 bytes

- **Multithreading scheduling policy**
  Fixed priority, Round robin, Dynamic

- **Priority Inversion Avoidance Mechanism:**
  Yes, Priority Inheritance

- **Multiprocess Support:**
  No

- **Multiprocessor Support:**
  Yes

- **Multithread Support:**
  Yes

- **MMU Support:**
  Yes

- **Autocoder**
  No

- **RMA**
  Yes

**Commercial Information:**

- **Royalty Free:**
  No

- **Standard Phone Support:**
  Paid
Appendix B

- **Preferred Phone Support:**
  Paid

- **Product Updates**
  Paid

- **Newsletters and technical updates**
  Unlimited free
APPENDIX C QNX/Momentics

Features

- **Unparalleled fault flexibility**
  Every module in QNX Neutrino system runs in its own memory, protected from each other.

- **High Availability**
  QNX Neutrino provides a high availability (HA) manager, which means that the system can be designed to repair and restart failed processes automatically.
  HA Features include:
  - heart beating
  - check pointing
  - fault isolation
  - automatic restart and recovery
  - post-mortem file analysis

- **Increased serviceability**
  Due to the memory protected design, new services can safely be added to the system, even after it is deployed.

- **POSIX compliance**
  QNX Neutrino was designed from the ground up to comply with POSIX real-time standards.

- **Real-time performance**
  Designed for mission critical performance. Neutrino provides:
  - Automatic synchronisation of component interactions
  - Low context switch times
  - Nested interrupts
  - Priority inversion avoidance
  - Four scheduling methods: fixed priority preemptive scheduling, run-to-block (FIFO), round robin, or sporadic

- **Full-featured GUI**
  QNX Neutrino includes the Photon® microGUI®, a highly modular graphical environment.

Development information

- **Development Hosts**
  Windows, Solaris, Self-Hosted, QNX4, Linux

- **Supported Target Processors**
  x86, PowerPC, MIPS, StrongARM, SH4

- **Board Support Package**
  see www.qnx.com

- **Supported Compilers**
  GCC

- **Supported Tools**
  GNU, QNX Real-Time Platform

- **Supported Networks**
  TCP/IP, FTP, SMTP, SNMP, NFS, PPP, ATM, ISDN, RPC, Telnet, Bootp, tiny TCP/IP
• **Supported Standards**
  POSIX.1 (a,b,c,d)

• **Development Methodology**
  Cross, Native

• **RTOS Supplied as**
  Object

• **GUI Support**
  Photon, X in Photon, Citrix ICA

• **Available Components**
  Floating Point, Communication, Math Library, File Support, see also www.qnx.com

### Technical Information

- **Kernel ROM (min, max):**
  64K, 64K

- **Kernel RAM (min, max):**
  -

- **Minimum RAM per thread:**
  -

- **Minimum RAM per queue:**
  -

- **Scheduling policy**
  Fixed priority preemptive scheduling, run-to-block (FIFO), round robin, or sporadic

- **Priority Inversion Avoidance Mechanism:**
  Yes, Priority Inheritance

- **Multiprocess Support:**
  Yes

- **Multiprocessor Support:**
  Yes

- **Multithread Support:**
  Yes

- **MMU Support:**
  Yes

- **Number of task priority levels**
  64

- **Typical thread switch latency**
  Depends

- **Guaranteed Maximum Interrupt Latency**
  Depends
Appendix C

- System clock resolution
  Depends

Commercial Information:

- Royalty Free:
  No

- Cost development seat:
  Free for non-commercial use

- Preferred Phone Support:
  Paid

- Standard Phone Support:
  Limited free
APPENDIX D  LynxOS/CodeWarrior

Development Information:

- **Development Hosts:**
  Sun Solaris, SunOS, RS6000, LynxOS Native/Hosted

- **Supported Target Processors:**
  x86, 68k, PPC, microSPARC, microSPARC II, PA-RISC

- **Board Support Packages:**
  PowerPC Family: Motorola MVME1600 and MVME2600 VME Boards, Motorola PowerStack Series E Systems, Motorola Ultra Boards and Ultra-based PowerStack RISC PC Systems, FORCE CPU-6604, FORCE CPU-60xRT, CES RTPC 8067; Motorola MPC860 PowrQUICC: Motorola ADS

- **Supported Compilers:**
  Included in LynuxWorks Open Development Environment: gcc, G++; Via third-parties: FORTRAN 77/90, C++, Ada83, Ada95, Pascal, Modula-2

- **Supported Tools:**
  Included in LynuxWorks Open Development Environment: gdb, X/Motif client/server development tools; Included in LynuxWorks PosixWorks Environment: TotalView - Multi-process, Multi-thread, multi-processor source-level debugger, TimeScan - Posix-aware m

- **Supported Networks:**
  TCP/IP, SNMP, NFS, Numerous network interface cards and devices, Other protocols and hardware through third-parties

- **Supported Standards:**
  POSIX.1/.1b/.1c, Unix BSD 4.3

- **Development Methodology:**
  Cross, Native

- **RTOS Supplied as:**
  Object, Source

- **Supported GUI:**
  X-Windows, Motif, others

- **Available Components:**
  Floating Point, Communication, Math Library, File Support, Cache Support, Network Support

Technical Information:

- **Kernel ROM (min, max):**
  280K, 4M
Appendix D

- **Kernel RAM (min, max):**
  500K, 4G

- **Minimum RAM per process:**
  1073 bytes

- **Minimum RAM per thread:**
  1073 bytes

- **Minimum RAM per queue:**
  80 bytes

- **Number of priority levels:**
  256 (512)

- **Typical Thread Switch Latency:**
  4us to 19us

- **Guaranteed Maximum Interrupt Latency:**
  14us

- **System Clock Resolution:**
  20ns

- **Priority Inversion Avoidance Mechanism:**
  Yes, priority inheritance

- **Multiprocess Support:**
  Yes

- **Multithread Support:**
  Yes

- **Multiprocessor Support:**
  Yes

- **MMU Support:**
  Yes

- **Autocoder:**
  Yes

- **RMA:**
  Yes

- **Scheduling Policies:**
  Prioritized FIFO, Prioritized Round-Robin, Fixed Priority, Time Slice, Dynamic, deadline monotonic scheduling, Edynamic, run to completion

**Commercial Information:**

- **Royalty Free:**
  No
Appendix D

- **Standard Phone Support**:  
  Paid

- **Preferred Phone Support**:  
  Paid
Appendix E

Installation of GNAT Pro

Why GNAT instead of AdaMulti?

For this assignment, a fully functional BSP and a Motorola MPC 604 processor was given. At first GreenHills AdaMulti 2000 was going to be used to enable programming in Ada95. However using this in combination with VxWorks became quite cumbersome due to inadequate documentation. Therefore another option was investigated. The Ada95 compiler, GNAT Pro, was also available. This compiler is especially suited to work in combination with VxWorks and seemed easier to use instead of AdaMulti. AdaMulti could probably have worked just as well, but GNAT was the program that was chosen. GNAT Pro was also chosen because VxWorks at this stage had already been setup and worked with the hardware.

Installation

An evaluation licence of Tornado II had been installed on the host computer, and to this development environment an Ada-compiler was needed. As mentioned above, GNAT pro was chosen for this task.

The GNAT software was downloaded from the Internet [36] after a temporary licence was acquired.

The two files that were downloaded were:

- gnat-3.15a1-ppc-vxworks.exe
- t2_integration.exe

At first an extra installation was made to the already existing GNAT native software. This complement was called GNAT pro for VxWorks and was installed in the same directory as GNAT native.

When the software was installed correctly, another program needed to be installed (t2_integration). This new program integrated the GNAT pro software with the Tornado II environment.

Two new buttons were created in the Tornado II toolbar. These buttons provided quick access to the Ada environment.

![Figure E-1: Tornado II environment overview](image)
The tool that follows the GNAT pro software is an integrated development environment developed by Ada Core Technologies. The system includes the Emacs editor, the GNAT toolchain and a graphical debugger. Ada Glide was pretty easy and pleasant to work with. Thanks to its well-made interface and especially the help templates, the programming gets simplified significantly.

One advantage of GNAT is that when the GNAT software have been integrated in Tornado, both C/C++ and Ada files can be mixed, implemented and used together in the same project.

**Problems**

At first, the standard GNAT pro was installed. However, it did not work with the Tornado environment straight away. As described above, two more programs needed to be downloaded and installed in the right order.

Due to the great amount of tests, having all source files in the same directory made it quite hard to get a decent overview of the files. When compiling, the compiler must be able to locate source files needed by the main program. Search paths are used to guide this process. To find these source files it was necessary to list all directories in the “ADA_INCLUDE_PATH” environment variable. This could be done as shown in figure E-3.
Since the path name for the tests were quite long, it became necessary to divide the ADA_INCLUDE_PATH section into two pieces. To be able to run chapters C94, CXD1, CXD2 and CXD3, ADA_INCLUDE_PATH 1 will need to be copied to the environment variable. To run the rest of the test, ADA_INCLUDE_PATH 2 need to be copied instead.

The ADA_INCLUDE_PATH 1 variable:

T:\USERS\la-ex\ACATS_test;T:\USERS\la-ex\ACATS_test\c940005;T:\USERS\la-ex\ACATS_test\c940006;T:\USERS\la-ex\ACATS_test\c940007;T:\USERS\la-ex\ACATS_test\c940011;T:\USERS\la-ex\ACATS_test\c940012;T:\USERS\la-ex\ACATS_test\c940013;T:\USERS\la-ex\ACATS_test\ccd6003;T:\USERS\la-ex\ACATS_test\ccd1001;T:\USERS\la-ex\ACATS_test\ccd1002;T:\USERS\la-ex\ACATS_test\ccd1003;T:\USERS\la-ex\ACATS_test\ccd1004;T:\USERS\la-ex\ACATS_test\ccd1005;T:\USERS\la-ex\ACATS_test\ccd1006;T:\USERS\la-ex\ACATS_test\ccd1007;T:\USERS\la-ex\ACATS_test\ccd1008;T:\USERS\la-ex\ACATS_test\ccd2001;T:\USERS\la-ex\ACATS_test\ccd2002;T:\USERS\la-ex\ACATS_test\ccd2003;T:\USERS\la-ex\ACATS_test\ccd2004;T:\USERS\la-ex\ACATS_test\ccd2006;T:\USERS\la-ex\ACATS_test\ccd2007;T:\USERS\la-ex\ACATS_test\ccd2008;T:\USERS\la-ex\ACATS_test\ccd3001;T:\USERS\la-ex\ACATS_test\ccd3002;T:\USERS\la-ex\ACATS_test\ccd3003
The ADA_INCLUDE_PATH 2 variable:
T:\USERS\la-ex\ACATS_test;T:\USERS\la-ex\ACATS_test\cx44001;T:\USERS\la-ex\ACATS_test\cx44002;T:\USERS\la-ex\ACATS_test\cx44003;T:\USERS\la-ex\ACATS_test\cx44004;T:\USERS\la-ex\ACATS_test\cx44005;T:\USERS\la-ex\ACATS_test\cx44006;T:\USERS\la-ex\ACATS_test\cx44007;T:\USERS\la-ex\ACATS_test\cx44008;T:\USERS\la-ex\ACATS_test\cx44009;T:\USERS\la-ex\ACATS_test\cx44010;T:\USERS\la-ex\ACATS_test\cx45001;T:\USERS\la-ex\ACATS_test\cx45002;T:\USERS\la-ex\ACATS_test\cx45003;T:\USERS\la-ex\ACATS_test\cx45004;T:\USERS\la-ex\ACATS_test\cx45005;T:\USERS\la-ex\ACATS_test\cx45006;T:\USERS\la-ex\ACATS_test\cx45007;T:\USERS\la-ex\ACATS_test\cx45008;T:\USERS\la-ex\ACATS_test\cx45009;T:\USERS\la-ex\ACATS_test\cx45010;T:\USERS\la-ex\ACATS_test\cx80001;T:\USERS\la-ex\ACATS_test\cx80002;T:\USERS\la-ex\ACATS_test\cx80003;T:\USERS\la-ex\ACATS_test\cx80004;T:\USERS\la-ex\ACATS_test\cx80005;T:\USERS\la-ex\ACATS_test\cx80006;T:\USERS\la-ex\ACATS_test\cx80007;T:\USERS\la-ex\ACATS_test\cx80008;T:\USERS\la-ex\ACATS_test\cx80009;T:\USERS\la-ex\ACATS_test\cx80010;T:\USERS\la-ex\ACATS_test\cxda001;T:\USERS\la-ex\ACATS_test\cxda002;T:\USERS\la-ex\ACATS_test\cxda003;T:\USERS\la-ex\ACATS_test\cxda004

Note: The test paths will differ depending on where the tests are stored. To use the new environment variable, the computer needs to be restarted.
Appendix F

Connection to hardware

Installation of hardware

The information below will describe how programs and hardware was setup to work.

1. The hardware (Target) was connected to the PC (Host) using a serial cable. The VxWorks hyper terminal was used to communicate between the host and target. The connection was setup as below:

![Connection setup](image)

Figure F-1: Connection setup

2. By using the information shown in the hyper terminal, the correct values for the hardware can be setup [37]. The information from the hardware will look something like this:

![Hardware setup](image)

Figure F-2: Hardware setup

3. Some information needed to be changed so that it would be possible to communicate with the target using the Ethernet. The hardware needs the Ethernet when downloading the VxWorks core and applications. The information that needed to be changed is described below:
Host name – The name of the host machine to boot from.
File name – The full path name of the VxWorks object module to be booted.
Inet on Ethernet – The Internet address of a target system with an Ethernet interface.
Host inet – The Internet address of the host to boot from. This value was obtained by typing “Ping hostname” in the command prompt in windows.
Gateway inet – The Internet address of the gateway node if the host is not on the same network as the target.
User – The user name that VxWorks uses to access the host; that user must have permission to read the VxWorks boot image file. VxWorks must have access to this user’s ftp-sign on, with the ftp password provided below.
Ftp password – The “user” password. The boot program does not require this field, but it must be supplied to boot over the network from a windows host.

4. When the hardware is turned on, it tries to download the VxWorks core and BSP. To enable this, an FTP (File Transfer Protocol) server is required as the download mechanism depends on this protocol. Tornado includes its own FTP server called WFTPD. This program needs to be set up with the same username and password as the hardware [38]. If the information is set up correctly, this message will be shown in the WFTPD window when the hardware is booting:

![FTP server window](image)

Figure F-3: FTP server window

5. To connect the host and target, VxWorks also needs to be configured. By choosing configure under tools, as in the picture, the connection can be set up.
6. The values were set up as in the picture:

7. When all these adjustments have been made, the installation should be complete. When turning on the hardware, the VxWorks core and BSP will be downloaded using Ethernet. The hardware is now setup and applications can be downloaded from VxWorks. The information from the hyper terminal will look like this when everything has been properly setup.
Problems

At first, the hardware was connected so that it would communicate over the Ethernet. For some reason, the hardware was unable to download the core for VxWorks. A hub was used to solve this. The hub connected the host and target together, and it worked straight away.
REAL TIME TEST SUITE

These tests contain code that tests the real-time functions in Ada. The table shows which types of tests that are covered. If a more accurate description is needed, click the link in the table.

The tests also have a link to the corresponding chapter in the Language Reference Manual (LRM).

**RS:** The chosen test will need to be run separately, as it does not work together with the other tests within the same family.

**LRM - Table of Contents**
## Appendix G

### Tests

<table>
<thead>
<tr>
<th>Test num.</th>
<th>Synchronous transfer</th>
<th>Clock</th>
<th>Delay</th>
<th>Exception</th>
<th>Floating points</th>
<th>Interrupt</th>
<th>Preemption</th>
<th>Priorities</th>
<th>Procedures</th>
<th>Protected object</th>
<th>Queuing</th>
<th>Rendezvous</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test name: C94 (Protected units and protected objects) (LRM)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C940005</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C940006</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C940007</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C940011</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C940012</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C940013</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| Test name: CXD1 (Task Priority) (LRM) |
| CXD1001  | X                     |       |       |           |                 |           |            |            |            |                  |         |            |      |
| CXD1002  |                       | X     |       |           |                 |           |            |            |            |                  |         |            |      |
| RS       |                       |       |       |           |                 |           |            |            |            |                  |         |            |      |
| CXD1003  |                       | X     |       |           |                 |           |            |            |            |                  |         |            |      |
| CXD1004  |                       |       |       |           |                 |           |            |            |            |                  |         |            |      |
| CXD1005  |                       |       |       |           |                 |           |            |            |            |                  |         |            |      |
| CXD1006  |                       |       |       |           |                 |           |            |            |            |                  |         |            |      |
| RS       |                       |       |       |           |                 |           |            |            |            |                  |         |            |      |
| CXD1007  |                       |       |       |           |                 |           |            |            |            |                  |         |            |      |
| RS       |                       |       |       |           |                 |           |            |            |            |                  |         |            |      |
| CXD1008  |                       |       |       |           |                 |           |            |            |            |                  |         |            |      |
|           |                       |       |       |           |                 |           |            |            |            |                  |         |            |      |

| Test name: CXD2 (Priority Scheduling) (LRM) |
| CXD2001  | X                     |       |       |           |                 |           |            |            |            |                  |         |            |      |
| CXD2002  |                       |       |       |           |                 |           |            |            |            |                  |         |            |      |
| CXD2003  |                       | X     |       |           |                 |           |            |            |            |                  |         |            |      |
| CXD2004  |                       |       |       |           |                 |           |            |            |            |                  |         |            |      |
| CXD2006  |                       |       |       |           |                 |           |            |            |            |                  |         |            |      |
| CXD2008  |                       |       |       |           |                 |           |            |            |            |                  |         |            |      |
|           |                       |       |       |           |                 |           |            |            |            |                  |         |            |      |

<p>| Test name: CXD3 (Priority Ceiling Locking) (LRM) |
| CXD3001  | X                     |       |       |           |                 |           |            |            |            |                  |         |            |      |
| CXD3002  |                       | X     |       |           |                 |           |            |            |            |                  |         |            |      |
| CXD3003  |                       |       |       |           |                 |           |            |            |            |                  |         |            |      |</p>
<table>
<thead>
<tr>
<th>Test num.</th>
<th>Synchronous transfer</th>
<th>Clock</th>
<th>Delay</th>
<th>Exception</th>
<th>Floating points</th>
<th>Interrupt</th>
<th>Precise</th>
<th>Preemption</th>
<th>Priorities</th>
<th>Procedures</th>
<th>Protected object</th>
<th>Queuing</th>
<th>Rendezvous</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>CXD4001</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CXD4002</td>
<td>RS</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CXD4003</td>
<td>RS</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CXD4004</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CXD4005</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CXD4006</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CXD4007</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CXD4008</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CXD4009</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CXD4010</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Test name: CXD4 (Entry Queuing Policies) (LRM)

Test name: CXD5 (Dynamic Priorities) (LRM)

Test name: CXD6 (Preemptive abort) (LRM)

Test name: CXD8 (Monotonic time) (LRM)

Test name: CXD9 (Delay Accuracy) (LRM)

Test name: CXD10 (Synchronous Task Control) (LRM)
Appendix G

C940005.A

OBJECTIVE:
Check that the body of a protected function can have internal calls to other protected functions and that the body of a protected procedure can have internal calls to protected procedures and to protected functions.

TEST DESCRIPTION:
Simulate a meter at a freeway on-ramp which, when real-time sensors determine that the freeway is becoming saturated, triggers stop lights which control the access of vehicles to prevent further saturation. A protected object represents each on-ramp - in this case only one is shown (Test_Ramp). The routines to sample and alter the states of the various sensors, to queue the vehicles on the meter and to release them are all part of the protected object and can be shared by various tasks. Apart from the function/procedure tests this example has a mix of other tasking features.

--------------------------------- Test result ---------------------------------
--- C940005 Check internal calls of protected functions and procedures.
==== C940005 PASSED ==============

C940006.A

OBJECTIVE:
Check that the body of a protected function can have external calls to other protected functions and that the body of a protected procedure can have external calls to protected procedures and to protected functions.

TEST DESCRIPTION:
Use a subset of the simulation of the freeway on-ramp described in C940005. In this case two protected objects are used but only a minimum of routines are shown in each. Both objects are hard coded and detail two adjacent on-ramps (Ramp_31 & Ramp_32) with routines in each, which use external calls to the other.

--------------------------------- Test result ---------------------------------
--- C940006 Check external calls of protected functions and procedures.
==== C940006 PASSED ==============

G:4
C940007.A

OBJECTIVE:
Check that the body of a protected function declared as an object of a given type can have internal calls to other protected functions and that a protected procedure in such an object can have internal calls to protected procedures and to protected functions.

TEST DESCRIPTION:
Simulate a meter at a freeway on-ramp which, when real-time sensors determine that the freeway is becoming saturated, triggers stop lights which control the access of vehicles to prevent further saturation. A protected object of the type Ramp represents each on-ramp. The routines to sample and alter the states of the various sensors, to queue the vehicles on the meter and to release them are all part of the protected object and can be shared by various tasks. Apart from the function/procedure tests this example has a mix of other tasking features. In this test two objects representing two adjacent ramps are created from the same type. The same "traffic" is simulated for each ramp. The results should be identical.

------------------------ Test result ------------------------
---- C940007 Check internal calls of protected functions and procedures
     in objects declared as a type.
===== C940007 PASSED ===================================.

C940011.A

OBJECTIVE:
Check that, in the body of a protected object created by the execution of an allocator, external calls to other protected objects via the access type are correctly performed

TEST DESCRIPTION:
Use a subset of the simulation of the freeway on-ramp described in c940005. In this case an array of access types is built with pointers to successive ramps. The external calls within the protected objects are made via the index into the array. Routines, which refer to the “previous” ramp and the “next” ramp, are exercised. (Note: The first and last ramps are assumed to be dummies and no first/last condition code is included)

------------------------ Test result ------------------------
---- C940011 Protected Objects created by allocators: external calls via access types.
===== C940011 PASSED ===================================.
Appendix G

C940012.A

OBJECTIVE:
Check that a protected object can have discriminants

TEST DESCRIPTION:
Use a subset of the simulation of the freeway on-ramp described in c940005. In this case an array of access types is built with pointers to successive ramps. Each ramp has its Ramp_Number specified by discriminant and this corresponds to the index in the array. The test checks that the ramp numbers are assigned as expected then uses calls to procedures within the objects (ramps) to verify external calls to ensure the structures are valid. The external references within the protected objects are made via the index into the array. Routines which refer to the "previous" ramp and the "next" ramp are exercised.

---------------------------------- Test result ----------------------------------
---- C940012 Check that a protected object can have discriminants.
     ==== C940012 PASSED ================================

C940013.A

OBJECTIVE:
Check that items queued on a protected entry are handled FIFO and that the 'count attribute of that entry reflects the length of the queue.

TEST DESCRIPTION:
Use a small subset of the freeway ramp simulation shown in other tests. With the timing pulse off (which prevents items from being removed from the queue) queue up a small number of calls. Start the timing pulse and, at the first execution of the entry code, check the 'count attribute. Empty the queue. Pass the items being removed from the queue to the Ramp_Sensor_01 task; there check that the items are arriving in FIFO order. Check the final 'count value

Send another batch of items at a rate which will, if the delay timing of the implementation is reasonable, cause the queue length to fluctuate in both directions. Again check that all items arrive FIFO. At the end check that the 'count returned to zero reflecting the empty queue.

---------------------------------- Test result ----------------------------------
---- C940013 Check that queues on protected entries are handled FIFO and that 'count is correct.
     ==== C940013 PASSED ================================

G:6
Appendix G

CXD1001.A

OBJECTIVE:
Check that the range of System.Priority is at least 30 values; that System.Interrupt_Priority has at least one value and is higher than System.Priority and the System.Default_Priority is at the center of the range of System.Priority.
Check the behavior of Ada.Dynamic_Priorities.Set_Priority and Get_Priority; specifically that Set_Priority will set a value that can later be confirmed with Get_Priority.
Check that, in the absence of Pragma Priority, the main subprogram has a base priority of Default_Priority.

TEST DESCRIPTION:
Verify the priority ranges with simple declarations and assignments. Include one additional runtime calculation. Check the manipulation of Ada.Dynamic_Priorities by getting the current value, setting it to something different then getting it again to check that the new value has been set.
Note: This test does not verify that the priority modification has any effect on the object program.

An incorrect range of system priorities will cause this to fail. If the range of system priorities is < 30 the declaration of the subtype App_Priority should result in a compile time warning or fatal error. If the fatal error does not occur then a Constraint_Error raised during the elaboration of the same declaration is a test failure. If the compiler itself is incorrectly implemented the assignment to App_Priority_1 should raise Constraint_Error resulting in test failure. The test is designed in such a way that the exception is unhandled. As a further check (which does not rely on the implementation correctly dealing with ranges) a "non-user" type check is used to verify the actual range.

-------------------------- Test result --------------------------
--- CXD1001 Check Priority range, set and get Ada.Dynamic_Priorities.
==== CXD1001 PASSED ================================.

CXD1002.A

OBJECTIVE:
Check that the base priority of the main subprogram can be set by means of pragma priority.
Check that a task's base priority is the priority of the parent at the time the task is created when the priority of the parent has been set by means of pragma priority.
Check that a task's base priority is the priority of the parent at the time the task is created when the priority of the grandfather has been set by means of pragma priority.

TEST DESCRIPTION:
The main subprogram sets its priority to something other than the default by means of pragma priority. It then spawns a child, and the child in turn spawns another child. In each case check the base priority with Ada.Dynamic_Priorities.Get_Priority.
Appendix G

---------------------------- Test result ---------------------------

---- CXD1002 Check setting of base priority of main with pragma priority.
==== CXD1002 PASSED ===============================================.

CXD1003.A

OBJECTIVE:
Check that during rendezvous, the task accepting the entry call inherits the active priority of the caller. Specifically, check when the caller has a higher priority than the receiver.

TEST DESCRIPTION:
The rules of inheritance are such that the rendezvous will be performed at the maximum of all the inherited priorities. The base priority of the receiver is a source of inheritance and is low but the caller is high so the rendezvous must be performed at the priority of the caller.

We use the fact that a call to a Protected object whose Ceiling Priority is lower than the caller's will result in a Program Error being raised. We arrange to call an entry in a task, which has a low priority from a task with high priority. We have a Protected Object whose Ceiling is between the priority of the low and high priority tasks. We call the PO from the lower priority task to make sure that the call can be made. We then call the low priority task from the high and in the rendezvous call the PO again. This time, as the rendezvous should inherit the High priority of the caller the Ceiling priority of the PO will be violated raising program error.

Since implementations are permitted to round up ceilings to the highest priority in the range we use System.Priority for the priorities of the low priority task and the PO and we use Interrupt_Priority for the calling task.

---------------------------- Test result ---------------------------

---- CXD1003 Priority: Rendezvous inherits active priority of caller.
==== CXD1003 PASSED ===============================================.

CXD1004.A

OBJECTIVE:
Check that during activation, a task being activated inherits the active priority of its activator (in this case the activator's base priority). Check that, if this priority is higher than the base priority of the activated task, this base priority remains unchanged.

TEST DESCRIPTION:
Three separate instances of a task type T1 are generated, one by the Main with the Default Base priority, one by a task (Task_3Q) with a base priority high in the normal priority range and one by a task who's base priority is in the Interrupt range. T1 is designed to trap its activation priority and store the same in the Resulting_Priorities array when the body of the task is executed. The instances of the task are distinguished by discriminants.
Two levels of activation priority can be verified – Interrupt priority and less-than-interrupt priority. These are called Hi_Low_Priority High/Low. We can determine which of these applies by calling a Protected Object who’s Ceiling priority has been set to System.Priority'Last. If the current activation priority is not greater than the ceiling (Low) the call will be honored. If it is greater than the ceiling (High) this is a violation of the Ceiling Priority of the Protected Object and Program_Error will be raised.

------------------------ Test result ------------------------

----- CXD1004 During Activation, a task inherits the active priority of its parent.

==== CXD1004 PASSED ==============.

CXD1005.A

OBJECTIVE:
Check that, during activation, a task being activated inherits the active priority of its activator. Specifically, check when the active priority of the activator is higher than the activator's Base Priority. Check that if the priority of the activated task is higher than its base priority, the base priority remains unchanged.

TEST DESCRIPTION:
A task with Base Priority in the interrupt range enters a rendezvous with one whose base is in the normal priority range. During the rendezvous, which will be performed at the interrupt priority, a task is created (Activating_Task). This task should then be activated at interrupt priority. During activation the task notes its current active priority (High or Low - see below) and preserves this for later check in the body. Also the body of the task verifies that it is running at its own base priority rather than the priority of its activation.

Two levels of activation priority can be verified – Interrupt priority and less than interrupt priority. These are Hi_Low_Priority High/Low. We can determine which of these applies by calling a Protected Object who’s Ceiling priority has been set to System.Priority'Last. If the current activation priority is not greater than the ceiling (Low) the call will be honored. If it is greater than the ceiling (High) this is a violation of the Ceiling Priority of the Protected Object and Program_Error will be raised.

------------------------ Test result ------------------------

----- CXD1005 Check that during Activation a task inherits the active priority of its parent.

==== CXD1005 PASSED ==============.
CXD1006.A

TEST DESCRIPTION:
This test declares a task that contains a pragma Interrupt_Priority. The package Ada.Dynamic_Priorities is used to query the priority of the task.

APPLICABILITY CRITERIA:
This test applies only to implementations supporting the Real-Time Systems Annex, and which allow tasks to have priorities in the interrupt priority range (D.2.1(10)).

----------------------------- Test result -----------------------------
---- CXD1006 Check the priority of a task containing an
     Interrupt_Priority pragma without the optional
     expression.
---- CXD1006 PASSED ================================.

CXD1007.A

OBJECTIVE:
Check that a priority pragma has no effect if it occurs in the declarative_part of a subprogram_body other than the main subprogram. Check that the priority specified for the main subprogram sets the priority of the environment task. Check that dynamic values can be specified in the interrupt_priority and priority pragmas.

TEST DESCRIPTION:
This test declares two subprograms, both of which contain pragmas priority declarations. A check is made that only the real main procedure's priority is used for the environment task. Note that the check is performed in the non-main subprogram so the priority expected is not the priority specified in the pragma priority declaration in that subprogram. Tasks are declared where the priorities for the tasks are dynamic values. Each task's priority is checked. The package Ada.Dynamic_Priorities is used to query the priority of the tasks (including the environment task).

----------------------------- Test result -----------------------------
---- CXD1007 Check the priority of the environment task and priorities
     set with dynamic expressions.
---- CXD1007 PASSED ================================.

CXD1008.A

OBJECTIVE:
Check that task scheduling, floating-point operations, and exceptions work properly together.

TEST DESCRIPTION:
The first package in this test is a generic cyclic task package. This package is instantiated in the main procedure multiple times to produce a number of tasks, all at different priorities. The second package in this test declares the operations that are carried out by the tasks. Since the
main purpose of the test is to check out the interaction of features when preemption occurs, the operations each inform the other operations as to when they have been preempted. Each operation is performed until it has been preempted a desired number of times.

------------------------------------- Test result -------------------------------------
---- CXD1008 Check that preemptive scheduling, floating point, and exceptions all work together.
==== CXD1008 PASSED ===============================.

CXD2001.A

OBJECTIVE:
Check that when Task.Dispatching_Policy is FIFO.Within_Priorities and the setting of the base priority of a task takes effect, the task is added to the tail of the ready queue of its active priority.

TEST DESCRIPTION:
Two tasks of low priority are activated by a task of medium priority. While they are on the low priority ready queue the medium priority task changes its own priority to be the same as the children. It should go to the tail of the low priority queue. Each task registers its subsequent action with a protected object, which checks the order of the registration. If the medium priority task is not suspended but continues it will register before the low priority tasks and the test will fail.

------------------------------------- Test result -------------------------------------
---- CXD2001 Setting base priority moves task to the end of the ready queue of that priority.
==== CXD2001 PASSED ===============================.

CXD2002.A

OBJECTIVE:
Check that when Task.Dispatching_Policy is FIFO.Within_Priorities and a task executes a delay statement that does not result in blocking, it is added to the tail of the ready queue of its active priority.

TEST DESCRIPTION:
Driver task is high priority. Delay_Task and type Sub_Task are medium priority
Low_Priority_Task is low priority

All the tasks get spawned (three of type Sub_Task) and each waits at a rendezvous with Driver. Driver makes the rendezvous with each. The rendezvous is completed at the priority of Driver and each task then gets placed on the ready queue of its priority (as they are lower than Driver). Careful handshaking ensures that the order on the medium queue is Delay_Task, then all the Sub_Tasks.
Delay_Task should run first. It is designed to rendezvous with Driver do a delay 0.0 then Register. After the rendezvous it should delay, get put on the back of the queue then Register
when it reaches the front of the queue. Sub_Task performs the rendezvous and then all it does is Register; Low_Priority_Task does the same.
The order of Registration must be: All the Sub_Tasks, Delay_Task, Low_Priority_Task. We repeat the whole process with delay_until a time in the past.

------------------------------------------ Test result ------------------------------------------
---- CXD2002 Default Task Dispatching - a non-blocking delay puts the
task at the tail of the ready queue for that priority.
===== CXD2002 PASSED ========================

CXD2003.A

OBJECTIVE:
Check that when TaskDispatching_Policy is FIFO_Within_Priorities and and a task's priority is lowered due to the loss of inherited priority it is added to the head of the ready queue for its priority

TEST DESCRIPTION:
Three tasks are created for each of three priorities. They are activated and each becomes blocked awaiting rendezvous with the Test Driver (Prime_Task). The Prime_Task then calls each one of the sub tasks in a predetermined order. As each sub task completes its rendezvous it should be placed on the head of the ready queue for its priority. Since the Prime_Task has the highest priority, the sub tasks will inherit the higher priority during the rendezvous and then lose this inherited priority at the end of the rendezvous. This will result in the sub task being put at the head of the ready queue for its priority.
The queues should be serviced in priority sequence and as each task reaches the head of the queue it registers with a protected object which checks against the expected sequence. This check is by means of an ID which was presented to the task at the time of the rendezvous. The tasks are called in a pseudo random (but predefined) order to ensure that there is no residual sequencing imposed by the initial activation.

------------------------------------------ Test result ------------------------------------------
---- CXD2003 Default Task Dispatching - when a task drops in priority due to loss of inheritance it is added to the head of the ready queue for its priority.
===== CXD2003 PASSED ========================

CXD2004.A

OBJECTIVE:
Check that when TaskDispatching_Policy is FIFO_Within_Priorities and the active priority of a running task is lowered due to loss of its inherited priority and there is a ready task of the same priority that is not running, the running task continues to run.

TEST DESCRIPTION:
A high priority Driver task calls two default priority Sub_Tasks ensuring they get placed on the default priority ready queue in a known order. The Driver task then terminates leaving the default priority queue as the only active ready queue. Sub_Task_A is the first on the queue so
Appendix G

it is restarted and calls a very high priority Protected Object. While it is running the protected object its inherited priority is high. As it completes the protected object it loses this inherited priority. But, as there is only another task of the same priority ready to run it should not get suspended but should continue to run; it immediately registers the fact that it is running. When Sub_Task_A terminates, Sub_Task_B is at the head of the ready queue and is restarted. It registers this action. The routine with which they both register determines the order of registration. If Sub_Task_B registers before Sub_task_A it means that the latter was suspended and placed on the tail of the queue when it lost its inherited priority and the test fails.

------------------------------------------- Test result -------------------------------------------

---- CXD2004 Default Task Dispatching. Loss of inherited priority of active task does not cause suspension if no higher priority task is ready.
==== CXD2004 PASSED ===========================

CXD2006.A

OBJECTIVE:
Check that priority ceases to be inherited as soon as the condition calling for the inheritance no longer exists.

TEST DESCRIPTION:
This test has one task that, by default, will run at the default priority. A second task that runs at an interrupt priority is used to cause the first task to inherit a higher priority. The active priority of the first task is checked around events that change the priority inheritance.

------------------------------------------- Test result -------------------------------------------

---- CXD2006 Check that priority ceases to be inherited as soon as the conditions for inheritance no longer exist.
==== CXD2006 PASSED ===========================

CXD2008.A

OBJECTIVE:
Check that if the Task_Dispatching_Policy is FIFO_Within_Priorities and a blocked task becomes ready then it is added to the tail of the ready queue for its priority.

TEST DESCRIPTION:
This test creates a number of tasks with a priority that is lower than the main task. The tasks first block on a suspension object and then on a protected object. In each case the tasks are unblocked in a particular order. Once unblocked the tasks register with a protected object that records the order in which the tasks arrived. This order is checked to see that they arrived in the order they were unblocked.
Appendix G

------------------- Test result -------------------

--- CXD2008 Check that when a blocked task becomes ready, it is added
to the tail of the ready queue for its priority.
==CXD2008 PASSED=============================

CXD3001.A

OBJECTIVE:
Check that Program_Error is raised if a task calls a protected operation who's ceiling is lower
than the task's active priority. Check this for Function, Procedure and Entry. Check that the
exception is not raised if the ceiling is equal to or higher than the priority of the calling task.

TEST DESCRIPTION:
Create a protected object with a mid-range priority and which contains a procedure, a function
and an entry. Create two tasks, one having a priority lower than the PO and one having a
priority equal to it; each of the tasks calls each of the subprograms and entry in the PO
checking that Program_Error is NOT raised.

Create three tasks with priority higher than the PO. The tasks in turn call one of the
subprograms and the entry. In each case check that Program_Error is raised.

------------------- Test result -------------------

--- CXD3001 Locking_Policy: Ceiling Locking. Calling protected
operations.
==CXD3001 PASSED=============================

CXD3002.A

OBJECTIVE:
Check that when Locking_Policy is Ceiling_Locking and no pragmas Priority,
Interrupt_Priority, Interrupt_Handler or Attach_Handler is specified in a protected definition
the Ceiling Priority of the protected object is System.Priority_Last

TEST DESCRIPTION:
Define a Protected Object, which has no Priority Pragmas. This should be given a ceiling of
Priority_Last. Create three tasks each of which call the P.O. Give the first a priority lower than
Priority_Last just to check reasonable access; the call should be successful. Give the second a
priority of Priority_Last. Since this is the same as the priority ceiling of the PO the call should
be successful. Give the third task the priority, which is precisely one higher than the expected
ceiling (in this case Interrupt'First). The call should fail the ceiling check and Program_Error
should be raised. Verification that the ceiling check is being performed correctly is carried out
elsewhere in the suite)

------------------- Test result -------------------

--- CXD3002 Ceiling_Locking: Default Priority Ceiling.
==CXD3002 PASSED=============================.
C XD3003.A

OBJECTIVE:
Check that when Locking_Policy is Ceiling_Locking and no pragma Priority or Interrupt_Priority is specified in a protected definition but a pragma Interrupt_Handler is specified, the ceiling priority is in the range of System.Interrupt_Priority.

TEST DESCRIPTION:
Define a Protected Object which has no Priority Pragma but does have an Interrupt_Handler pragma. Create two tasks both of which call the P.O.
Give the first a priority of Priority_Last just to check reasonable access; the call should be successful. Give the second task a priority of Interrupt_Priority_First.
This task should also be successful in accessing the P.O.

---------------------- Test result ----------------------
--- CXD3003 Ceiling_Locking: default priority ceiling for an interrupt
   handler protected object.
==== CXD3003 PASSED ==================================

C XD4001.A

OBJECTIVE:
Check that when Priority Queuing is in effect and the base priority of a task is set (changed), the priorities of any queued calls from that task are updated and that the ordering is modified accordingly.

TEST DESCRIPTION:
Three tasks are generated at the default priority. These tasks each call an entry in the distributor. Handshaking between the driver (Main), the tasks and the distributor ensures that the tasks arrive in the queue in order. Once all tasks are queued the base priority of the third task is raised. Further handshaking allows the Distributor to now service the queue and the order of the items being taken from the queue is checked. The call from the third task should be the one first handled.

---------------------- Test result ----------------------
CXD4001 Reordering entry queues on Priority change.
==== CXD4001 PASSED ==================================

C XD4002.A

OBJECTIVE:
Check that if no Queuing_Policy is specified, the policy for the partition is FIFO_Queueing and that the priorities of the calling tasks have no effect.

TEST DESCRIPTION:
Generate three tasks. The test driver calls each task in succession presenting it with a unique identification (Message_Number) and a priority level. The tasks use calls to Ada.Dynamic_Priorities to set their individual base priorities to that requested by the driver.
Each task calls the Distributor task and is queued. Handshaking between the driver and the Distributor ensures that the arrival order (by unique identification) is known. The higher priority calls are queued last. Once all the calls are queued the driver (by handshaking) allows the Distributor to proceed to the rendezvous where the tasks are queued. The distributor verifies that the tasks are processed in FIFO rather than priority order.

-------------------------- Test result --------------------------
--- CXD4002 Lack of Queuing_Policy gives FIFO queuing.
==== CXD4002 PASSED =========================================

CXD4003.A

OBJECTIVE:
Check that if Queuing_Policy FIFO_Queuing is specified for a partition the task entry queues are handled in FIFO order and that the priorities of the calling tasks have no effect.

TEST DESCRIPTION:
Generate three tasks. The test driver calls each task in succession presenting it with a unique identification (Message_Number) and a priority level. The tasks use calls to Ada.DYNAMIC_PRIORITIES to set their individual base priorities to that requested by the driver. Each task calls the Distributor task and is queued. Handshaking between the driver and the Distributor ensures that the arrival order (by unique identification) is known. The higher priority calls are queued last. Once all the calls are queued the driver (by handshaking) allows the Distributor to proceed to the rendezvous where the tasks are queued. The distributor verifies that the tasks are processed in FIFO rather than priority order.

-------------------------- Test result --------------------------
--- CXD4003 Check that if Queuing_Policy is FIFO_queuing the queues are handled FIFO and priorities have no effect.
==== CXD4003 PASSED =========================================

CXD4004.A

OBJECTIVE:
Check that changes to the active priority of the caller do not affect the priority of a call after it is first queued when the queuing policy is priority queuing.

TEST DESCRIPTION:
Three calls of equal priority are made to the Input queue of the Distributor. Before the calls are serviced, the priority of the task making the last call is raised and held high; while it is high the Distributor services the queue and checks that the last call is not processed out of turn. The task whose priority is modified (Modified_Task) uses an Asynchronous Select to place a call on the Input queue and, in parallel, use Ada.DYNAMIC_PRIORITIES.Set_Priority to raise its own priority.
----------------------------------- Test result -----------------------------------
----- CXD4004 Change in Active priority of a caller does not affect priority of a queued call.
===== CXD4004 PASSED ===============================.

CXD4005.A

OBJECTIVE:
Check that when Priority Queuing is in effect and the base priority of a task is set (changed), the priorities of any queued calls from that task to entries in a Protected Object are updated and that the ordering is modified accordingly.

TEST DESCRIPTION:
Generate three tasks at the default priority. These tasks each call the distributor. There is a guard in effect so the calls are all queued. Once all tasks are queued the base priority of the third task is raised. The guard is then released and the order of the items being taken from the queue is checked. The call from the third task should be the one first handled.

----------------------------------- Test result -----------------------------------
----- CXD4005 Reordering entry queues of Protected Objects on Priority change.
===== CXD4005 PASSED ===============================.

CXD4006.A

OBJECTIVE:
Check that if Queuing_Policy is Priority_Queueing, the calls to an entry are queued in an order consistent with the priority of the calls and that if an entry is removed and then reinserted it is added behind any other calls with equal priority in that queue.

TEST DESCRIPTION:
Nine test message tasks are generated, three each of different base priorities. Each is given a unique message number, which identifies it. Each message task queues itself on the Input queue of the Distributor. Handshaking ensures that the order of arrival at the Input queue is known. Once all the messages have been queued we know that they will be ordered first by priority and then FIFO order within that priority. We then set the base priority of the first message within the midrange priority (message number 201) but we do not alter that priority but set it to the same value as its original base priority. It should be pulled and reinserted in the queue after the other two tasks of the same priority. The distributor is then allowed to proceed to the Input rendezvous where it services the calls and checks the proper sequence.

----------------------------------- Test result -----------------------------------
----- CXD4006 Priority_Queueing: reinserted item is queued behind those of same priority.
===== CXD4006 PASSED ===============================.
Appendix G

**CXD4007.A**

**OBJECTIVE:**
Check that when multiple entry_barriers of a protected object become True and more than one of the respective queues are nonempty, the call with the highest priority is selected. Check that a minimum of 30 different priorities can be specified and that the priorities make a difference in the task scheduling.

**TEST DESCRIPTION:**
This test contains declarations for 30 tasks plus the environment task. All of these tasks enqueue themselves on entry queues of a single protected record. Once all the tasks have had a chance to become enqueued, the main procedure unblocks the entry queues. The order in which the entries are serviced is checked.

-------------------------------- Test result --------------------------------
---- CXD4007 Check that protected object entries are serviced based upon
the priority of the tasks waiting.
==== CXD4007 PASSED ==============

**CXD4008.A**

**OBJECTIVE:**
Check that when:
multiple entry_barriers of a protected object become True, more than one of the respective queues are nonempty, and the callers are all of the same priority then the entries are taken in textual order.
Check that when:
multiple alternatives of a selective_accept have queued callers and the callers are all of the same priority then the accept_alternative that is textually first in the selective_accept is selected.

**TEST DESCRIPTION:**
This test contains declarations for several tasks plus the environment task. All of these tasks enqueue themselves on entry queues of a single protected record. Once all the tasks have had a chance to become enqueued, the main procedure unblocks the entry queues. The order in which the entries are serviced is checked. A similar check is made for the selective_accept check.

-------------------------------- Test result --------------------------------
---- CXD4008 Check that protected object and task entries are serviced
in textual order when all waiting tasks are of equal priority.
==== CXD4008 PASSED ==============.
CXD4009.A

OBJECTIVE:
Check that when multiple alternatives of a selective_accept have queued callers and the callers are all of different priority then the accept_alternative that has the highest priority task waiting is selected.

TEST DESCRIPTION:
This test contains declarations for several tasks, each of which is at a different priority. All of these tasks enqueue themselves on a task entry of a single task. This single task has a selective_accept that will block the tasks until all the tasks are enqueued on the various entries. Once all the tasks are enqueued, the main procedure unblocks the entry queues. The order in which the entries are serviced is checked.

------------------------ Test result ------------------------
--- CXD4009 Check that task entries are serviced in priority order when all waiting tasks are of different priority.
==== CXD4009 PASSED ================================

CXD4010.A

OBJECTIVE:
Check that if the expiration time of two open delay_alternatives is the same and no other accept_alternatives are open then the sequence_of_statements of the delay_alternative that is first in textual order in the selective_accept is executed.

TEST DESCRIPTION:
This test contains a single task that checks the various combinations of delay expiration time in a select statement where none of the accept_alternatives are open.

------------------------ Test result ------------------------
--- CXD4010 Check that when two delay alternatives have the same expiration time that the one that occurs textualy first is selected.
==== CXD4010 PASSED ================================

CXD5001.A

OBJECTIVE:
Check that for Get_Priority, Tasking_Error is raised if the specified task has terminated.
Check that for Get & Set Priority, Program_Error is raised if the task has a null Task_Identification.

TEST DESCRIPTION:
Target_Task is brought up and allowed to terminate; Get_Priority is issued specifying Target_Task. Get and Set priority calls are then issued with an uninitialized Task_Id as the parameter.
CXD6002.A

OBJECTIVE:
Check that in an asynchronous transfer of control an aborted construct is completed immediately at the first point that is outside the execution of an abort-deferred operation.

TEST DESCRIPTION:
This test uses asynchronous transfer of control as the aborting mechanism and checks that the abort is performed at the first point outside an abort-deferred operation. This test relies upon the ability to unblock an interrupt priority task that in turn aborts a normal priority task. This is accomplished by having the interrupt priority task waiting for a Set_True on a Suspension_Object. The normal priority tasks that are aborted note their state in global variables. These variables are named starting with Check_Point_. This state allows us to determine if the abort occurred at the right time. In particular, one of the state variables is set while in an abort-deferred region that should be executing when the abort occurs and is used to check that abort deferral is properly performed. The setting of the final state variable occurs outside the abort-deferred area and should not occur since aborts are supposed to be immediate.
Test result

--- CXD6002 Check that an asynchronous transfer of control takes place as soon as the aborted statement is no longer in an abort-deferred region.

==== CXD6002 PASSED ==================

CXD8001.A

OBJECTIVE:
Check the basic functions in the Package Ada.Real_Time.

TEST DESCRIPTION:
Simple calls to the basic functions are made using the private constants.

Test result

--- CXD8001 Monotonic Time: Check the basic functions.
- CXD8001 test 01a.
- CXD8001 test 01b.
- CXD8001 test 01c.
- CXD8001 test 01d.
- CXD8001 Subtest 01 complete.
- CXD8001 test 02a.
- CXD8001 test 02b.
- CXD8001 test 02c.
- CXD8001 test 02d.
- CXD8001 Subtest 02 complete.
- CXD8001 test 03a.
- CXD8001 test 03d.
- CXD8001 Subtest 03 complete.
- CXD8001 test 04a.
- CXD8001 test 04b.
- CXD8001 test 04c.
- CXD8001 test 04d.
- CXD8001 test 04e.
- CXD8001 test 04f.
- CXD8001 test 04g.
- CXD8001 Subtest 04 complete.
- CXD8001 test 05a.
- CXD8001 test 05b.
- CXD8001 test 05c.
- CXD8001 test 05d.
- CXD8001 Subtest 05 complete.

==== CXD8001 PASSED ==================.
Appendix G

CXD8002.A

OBJECTIVE:
Check that Ada.Real_Time.Time can be used in a delay_until_statement. Check that a delay_statement blocks the task for at least as long as the requested delay as measured by Real_Time.Clock.

TEST DESCRIPTION:
This test sets up a number of different tasks, each of a different priority, all doing delays. The delays are checked to make sure they are long enough.

------------------------------- Test result -----------------------------
--- CXD8002 Check the use of Real_Time.Clock in delay statements.
==== CXD8002 PASSED ==========================================

CXD8003.A

OBJECTIVE:
Check that the Ada.Real_Time package operations Split and Time_Of operations work properly. Check that the clock does not jump backwards.

TEST DESCRIPTION:
The checks of Split and Time_Of check a variety of samples to be sure that the correct value is returned. To check that there are no backward clock jumps, the clock is sampled as frequently as possible and compared to the previous sample.

------------------------------- Test result -----------------------------
--- CXD8003 Check the Clock, Split and Time_Of routines of the Ada.Real_Time package.
==== CXD8003 PASSED ==========================================

CXD9001.A

OBJECTIVE:
Check that when a delay_statement appears in a delay_alternative of a timed_entry_call the entry call is attempted regardless of the specified expiration time.

TEST DESCRIPTION:
The main program attempts entry calls to both protected objects and tasks using a select statement with a delay alternative. The delay alternatives used are both relative delay and delay until. In all cases the value specified for the delay is either negative or a time in the past.

------------------------------- Test result -----------------------------
--- CXD9001 Test that a timed entry call is attempted regardless of the specified expiration time.
==== CXD9001 PASSED ==========================================
CXDA001.A

OBJECTIVE:
Check that, in Ada.Synchronous_Task_Control, Set_True and Set.False alter the state of a Suspension_Object appropriately. Check that Current_State returns the expected state. Check that the initial value of a Suspension_Object is set to false.

TEST DESCRIPTION:
In the Main subprogram use Current_State to verify the state of a Suspension_Object initially and after the use of Set_True and Set.False.

------------------ Test result ------------------
--- CXDA001 Synchronous Task Control: basic functionality.
==== CXDA001 PASSED ==========================

CXDA002.A

OBJECTIVE:
Check that, in Ada.Synchronous_Task_Control, Suspend_Until_True does suspend the task until the Suspension_Object is Set.True. Check that a call on Suspend_Until_True will raise Program_Error if another task is waiting on the same Suspension_Object.

TEST DESCRIPTION:
One task calls Suspend_Until_True. A second task sets the suspension_object true and also a flag to say it has executed the code to do so. The first task checks the flag after the suspension. If the flag is not set then the second task did not execute showing that the first task did not suspend properly. Also in the second task call Suspend_Until_True specifying the same object and verify that Program_Error is raised.

------------------ Test result ------------------
--- CXDA002 Synchronous Task Control: Task suspension.
==== CXDA002 PASSED ==========================

CXDA003.A

OBJECTIVE:
Check that Set.False and Set.True can be called during a protected operation that has its ceiling priority in the Interrupt_Priority range.

TEST DESCRIPTION:
A task is blocked, waiting on a suspension object to become true. The task is unblocked by
1. the main procedure while executing in a protected operation that has its ceiling priority set to Interrupt_Priority'First
2. a task executing at interrupt priority.
Appendix G

-------------------------------- Test result --------------------------------

---- CXDA003 Check that a suspension_object can be changed while executing at an interrupt priority.

==== CXDA003 PASSED =============================================.
Appendix H  The complementary test suite

C_TestSuite

//
// C_TestSuite
//
// This program calls on all different tests and reports
// the result.
//
//
// --------------------------------------------------------------

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

/* include */
#include "vxWorks.h"
#include "taskLib.h"
#include "semLib.h"
#include "PriorityInversion.h"
#include "MultiThreading.h"
#include "PriorityScheduling.h"
#include "Semaphore.h"
#include "MessagePassing.h"
#include "RoundRobin.h"
#include "Signalling.h"
#include "Time_Testing.h"

int resultat = 0;
int testres = 1;

void start(void) /* function to create the three tasks */
{
    printf("Welcome to Test 2002\n");
    printf("Sit back and enjoy the show\n\nThe test results will be shown in the HyperTerminal\n");

    printf("-------- Priority inversion being tested --------\n");
    /*
     * if((resultat=PriorityInversion())==0)
     * {
     *     printf("Failed -\n");
     *     testres = 0;
     * }
     * else if (resultat == 1)
     *     printf("Completed successfully\n");
     *
     */
    prinft("\n-------- Multi threading being tested --------\n");
    if((resultat=MultiThreading())==0)
    {
        prinft("Failed -\n");
        testres = 0;
    }
    else if (resultat == 1)
        printf("Completed successfully\n");

    taskDelay(200); //Delay to allow test to finish properly

    printf("-------- Semaphores is being tested --------\n");
    if((resultat=Semaphore())==0)


Appendix H

```c
{  printf("\n\n\n- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 
");  printf("* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * 
");  }
else if(resultat==1)
  printf("\nCompleted successfully\n");

printf("\n- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 
");
printf("- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 
");
```

```c
else if (resultat==1)
  printf("\nCompleted successfully\n");

printf("\n- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 
");
```

```c
if (testres == 0)
  printf("\n\n\n- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 
");
```

```c
else if (resultat==1)
  printf("\nCompleted successfully\n");

printf("\n- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 
");
```
Appendix H

Result from C_TestSuite:

Welcome to Test 2002
Sit back and enjoy the show
The test results will be shown in the HyperTerminal

------------------------------- Priority inversion being tested ------------------------------

Completed successfully

------------------------------- Multi threading being tested -------------------------------

Completed successfully

------------------------------- Semaphores is being tested -------------------------------

Completed successfully

------------------------------- Signalling is being tested -------------------------------

Signal Generated
All signals caught successfully

------------------------------- Round-Robin scheduling is being tested with TimeSlice = 1 seconds -------------------------------

Completed successfully

------------------------------- MessageQueues are being tested -------------------------------

Completed successfully

------------------------------- PriorityScheduling is being tested -------------------------------

Completed successfully

------------------------------- Time is being tested -------------------------------

System Clock Rate is : 60 tics per sec
The delay was set to 500 ticks
Starting Delay
Delay Finished!!
Delay was 8.3 sec
Time test completed

------------------------------- Test finished successfully -------------------------------

H:3
Appendix H

Priority Inversion

// Priority Inversion
//
// The example below illustrates a typical situation in which priority inversion takes place.
// Here is the what happens:
//
// 1. A low priority task locks a semaphore.
// 2. LowPriority task gets preempted by MediumPriority which runs for a long time which
// results in the blocking of LowPriority.
// 3. HighPriority preempts MediumPriority and tries to lock the semaphore, which is
// currently locked by LowPriority.
//
//***********************************************
//#include "vxWorks.h"
#include "taskLib.h"
#include "semLib.h"

/* Function prototypes */
void HighPriority(void);
void MediumPriority(void);
void LowPriority(void);

/* Globals */
#define ITERATION 3
#define HIGH 102  // High priority
#define MEDIUM 103 // Medium priority
#define LOW 104   // Low priority
#define LONG_TIME 300000

SEM_ID semMutex;

int prio = 0;

int PriorityInversion(void) // Function to create the three tasks
{
    int i, low, medium, high;

    // Create priority based semaphore, preventing priority inversion (VxWorks)
    semMutex = semMCreate(SEM_Q_PRIORITY | SEM_INVERSION_SAFE);

    // Creates 3 different tasks
    if((low = taskSpawn("task1",LOW,0x100,20000,(FUNCPTR)LowPriority,0,0,0,0,0,0,0,0,0,0,0,0)) == ERROR)
        printf("taskSpawn HighPriority failed\n");
    if((medium = taskSpawn("task2",MEDIUM,0x100,20000,(FUNCPTR)MediumPriority,0,0,0,0,0,0,0,0,0,0)) == ERROR)
        printf("taskSpawn MediumPriority failed\n");
    if((high = taskSpawn("task3",HIGH,0x100,20000,(FUNCPTR)HighPriority,0,0,0,0,0,0,0,0,0,0,0,0)) == ERROR)
        printf("taskSpawn LowPriority failed\n");

    taskDelay(200); // Let the program finish before returning prio
    return prio;
}

//***********************************************
void LowPriority(void)
{
    int i,j;
    for (i=0; i < ITERATION; i++)
    {
        ...
    }
}
Appendix H

```c
semTake(semMutex, WAIT_FOREVER); // Wait indefinitely for semaphore, take semaphore
printf("Low priority locks semaphore\n");
for (j=0; j < LONG_TIME; j++) //Do nothing for a long time
    printf("Low priority unlocks semaphore\n");
semGive(semMutex); // Release semaphore
}

printf("********************* Low priority task finished **********************\n");
if (prio == 2)
    prio = 1;
else
    prio = 0;
}

//********************** Medium priority task finished **********************
void MediumPriority(void)
{
    int i;
taskDelay(10); // Allow time for task with lower priority to run a while
for (i=0; i < LONG_TIME*10; i++) // Run for a longer time
    {
        if ((i % LONG_TIME) == 0) // Prevent message from being written too often
            printf("Medium task running\n");
    }
printf("********************* Medium priority task finished **********************\n");
if (prio == 1)
    prio = 2;
else
    prio = 0;
}

//********************** High priority task finished **********************
void HighPriority(void)
{
    int i,j;
taskDelay(20); // Allow time for task with the lowest priority to take semaphore
for (i=0; i < ITERATION; i++)
    {
        printf("High priority tries to lock semaphore\n");
        semTake(semMutex, WAIT_FOREVER); // Wait indefinitely for semaphore, take semaphore
        printf("High priority locks semaphore\n");
        for (j=0; j < LONG_TIME; j++) // Do nothing for a long time
            printf("High priority unlocks semaphore\n");
        semGive(semMutex); // Release semaphore
    }
printf("********************* High priority task finished **********************\n");
if (prio == 0)
    prio = 1;
else
    prio = 0;
}

//******************************
```

H:5
Result from hyper terminal:

Low priority locks semaphore
Low priority unlocks semaphore
Low priority locks semaphore
Medium task running
High priority tries to lock semaphore
Low priority unlocks semaphore
High priority locks semaphore
High priority unlocks semaphore
High priority tries to lock semaphore
High priority locks semaphore
High priority unlocks semaphore
High priority tries to lock semaphore
High priority locks semaphore
High priority unlocks semaphore
High priority tries to lock semaphore
High priority locks semaphore
High priority unlocks semaphore
High priority tries to lock semaphore
High priority locks semaphore
High priority unlocks semaphore

************************** High priority task finished **************************
Medium task running
Medium task running
Medium task running
Medium task running
Medium task running
Medium task running
Medium task running
Medium task running

************************** Medium priority task finished **************************
Low priority locks semaphore
Low priority unlocks semaphore

************************** Low priority task finished **************************
### Multi Threading

```c
// MultiThreading
//
// This program creates 10 different tasks and then prints out the id of each of the tasks
//
//
//******************************************************************************************
#include "vxWorks.h"

#define ITERATIONS 10

void print(void);

int retval = 0;

int MultiThreading(void)  // Subroutine to perform the spawning
{   
    int i, taskId;
    for(i=0; i < ITERATIONS; i++)  // Creates ten tasks
        if(taskId = taskSpawn("print",90,0x100,2000,print,0,0,0,0,0,0,0,0,0,0,0)=ERROR)
            printf("taskSpawn taskOne failed\n");
        else
            retval = 1;
    return retval;
}

void print(void)
{   
    printf("I am task %d\n",taskIdSelf());  // Print task Id
}

**Result from hyper terminal:**

I am task 16775120
I am task 16772584
I am task 16770048
I am task 16767512
I am task 16764976
I am task 16762440
I am task 16759904
I am task 12127992
I am task 12125456
I am task 12122920
```
Semaphores

// Semaphore
// This program creates 2 tasks which share a semaphore. One task
// increases a variable, the other decreases. This global variable
// should not be increased/decreased by more than 2

#include "vxWorks.h"  // includes
#include "taskLib.h"
#include "semLib.h"
#include "stdio.h"

void SemTaskOne(void);  // function prototypes
void SemTaskTwo(void);

#define ITERATIONS 5     // globals
SEM_ID sem;
int global = 0, varde1 = 0;

int Semaphore(void)
{
  int taskIdOne, taskIdTwo;

  sem = semBCreate(SEM_Q_FIFO, SEM_FULL);  //Create semaphore with semaphore available
                                             //and queue tasks on FIFO basis.
  semTake(sem,WAIT_FOREVER);               //Lock the semaphore for scheduling purposes
  taskIdOne = taskSpawn("t1",90,0x100,2000,(FUNCPT)SemTaskOne,0,0,0,0,0,0,0,0,0,0);
  taskIdTwo = taskSpawn("t2",90,0x100,2000,(FUNCPT)SemTaskTwo,0,0,0,0,0,0,0,0,0,0);

  taskDelay(200);                        // Needed so that result can be calculated
  return varde1;
}

void SemTaskOne(void)
{
  int i;
  for (i=0; i < ITERATIONS; i++)
  {
    semTake(sem,WAIT_FOREVER);           //Wait indefinitely for semaphore, then take it
    printf("I am taskOne = %d \n", ++global);  //increases global
    semGive(sem);                        //Release semaphore
  }
}

void SemTaskTwo(void)
{
  int i;
  semGive(sem);                          //Give up semaphore(a scheduling fix)
  for (i=0; i < ITERATIONS; i++)
  {
    semTake(sem,WAIT_FOREVER);           //Wait for semaphore, then take it
    printf("I am taskTwo = %d \n", --global);  //decreases global
    semGive(sem);
  }
  if(global==0)                           // After iterations, global should be 0
    varde1 = 1;
}
Appendix H

Result from hyper terminal:

I am taskOne = 1
I am taskTwo = 0
I am taskOne = 1
I am taskTwo = 0
I am taskOne = 1
I am taskTwo = 0
I am taskOne = 1
I am taskTwo = 0
I am taskOne = 1
I am taskTwo = 0
Signalling

// Signalling
//
// In the example below, the "SignalGenerator" function generates the SIGINT signal,
// and directs the signal to the "SignalCatcher" task. When "SignalCatcher" receives the signal,
// it suspends its normal execution and branches to a signal handler that it has installed
// (CatchSignal function).
//
//******************************************************************************
/* includes */
#include "vxWorks.h"
#include "sigLib.h"
#include "taskLib.h"
#include "stdio.h"

/* function prototypes */
void CatchSignal(int);
void SignalCatcher(void);

/* globals */
#define NO_OPTIONS 0
#define ITERATIONS 100
#define LONG_TIME 1000000
#define HIGHTIME 100
#define LOWPRIORITY 101
int ownId;
int upp = 1;

/******************************************************************************

int SignalGenerator(void) // task to generate the SIGINT signal
{ 
    int i, j, taskId;
    STATUS taskAlive; // Creates the signal-catch function
    if((taskId = taskSpawn("signal",100,0x100,20000,(FUNC PTR)SignalCatcher,0,0,0,0,0,0,0,0)) == ERROR)
        printf("taskSpawn SignalCatcher failed\n");
    ownId = taskIdSelf(); // Get The Id Of The SignalGenerator's Task
    taskDelay(30); // 0.3 sec delay, to allow SignalCatcher to run
    for (i=0; i < ITERATIONS; i++)
    {
        if ((taskAlive = taskIdVerify(taskId)) == OK) // Check if SignalCatcher is alive
            {
                printf("Signal Generated\n");
                upp++;
                kill(taskId, SIGINT); // Generate signal
                // Lower sigGenerator priority to allow sigCatcher to run
                taskPrioritySet(ownId,LOWPRIORITY);
                }
        else // SignalCatcher is dead....
            break;
        }
    taskDelay(200);

*/
if (upp == 1)
    printf("All signals caught successfully\n");
else
    {
        printf("Signals not caught successfully\n");
        upp=0;
    }

return upp;

}/**
 ****************** Signal Catcher ********************/
void SignalCatcher(void)    // Task running until the SIGINT signal arrives
{
    struct sigaction newAction;   // Sigaction installs signal handler for a task
    int i, j;

    newAction.sa_handler = CatchSignal;  // Name of function to jump to
    sigemptyset(&newAction.sa_mask);     // No other signals blocked
    newAction.sa_flags = NO_OPTIONS;     // No special options = NULL

    if(sigaction(SIGINT, &newAction, NULL) == -1)    // Check to see if signal handler can be installed
        printf("Could not install signal handler\n");

    for (i=0; i < 100; i++)    // Normal execution, interrupted when signal arrives
        for (j=0; j < LONG_TIME; j++)
    }

/**
 ****************** CatchSignal ********************/
void CatchSignal(int signal) // Signal handler code
{
    printf("SIGNAL CAUGHT\n");
    upp--;
    taskPrioritySet(ownId,HIGH_PRIORITY);   // Increase sigGenerator priority to allow sigGenerator to run
}

Result from hyper terminal:
Signal Generated
Signal Caught
**Appendix H**

**Round-Robin**

```c
#include "vxWorks.h"
#include "taskLib.h"
#include "kernelLib.h"
#include "sysLib.h"
#include "logLib.h"

/* function prototypes */
void task1(void);
void task2(void);
void task3(void);

/* globals */
#define ITERATION1 5
#define ITERATION2 10
#define PRIORITY 101
#define TIMESLICE sysClkRateGet()
#define LONG_TIME 8000000
int a=0, b=0, c=0;

int RoundScheduling(void)

{
    int taskIdOne, taskIdTwo, taskIdThree;

    if(kernelTimeSlice(TIMESLICE) == OK) // Turn round-robin on
        printf("with TimeSlice = %d seconds ---\n\n", TIMESLICE/60); //Prints slice time

    // Creates 3 tasks with same priority
    if((taskIdOne = taskSpawn("task1",PRIORITY,0x100,20000,(FUNCPTR)task1,0,0,0,0,0,0,0,0,0,0,0)) == ERROR)
        printf("taskSpawn taskOne failed\n");
    if((taskIdTwo = taskSpawn("task2",PRIORITY,0x100,20000,(FUNCPTR)task2,0,0,0,0,0,0,0,0,0,0,0) == ERROR)
        printf("taskSpawn taskTwo failed\n");
    if((taskIdThree = taskSpawn("task3",PRIORITY,0x100,20000,(FUNCPTR)task3,0,0,0,0,0,0,0,0,0,0,0) == ERROR)
        printf("taskSpawn taskThree failed\n");

    taskDelay(500);

    if (a==b && b==c)
        return 1;
    else
        return 0;
}
```

H:12
Appendix H

//******************************************* task1 *******************************************
void task1(void)
{
    int i,j;
    for (i=0; i < ITERATION1; i++)
    {
        for (j=0; j < ITERATION2; j++)
        {
            for ((j)=0; j < LONG_TIME; j++) // Allow time for context switch
                logMsg("ln",0,0,0,0,0,0);
            // Log messages
            a++;
        }
    }
}

//******************************************* task2 *******************************************
void task2(void)
{
    int i,j;
    for (i=0; i < ITERATION1; i++)
    {
        for (j=0; j < ITERATION2; j++)
        {
            for ((j)=0; j < LONG_TIME; j++) // Allow time for context switch
                logMsg("ln",0,0,0,0,0,0);
            // Log messages
            b++;
        }
    }
}

//******************************************* task3 *******************************************
void task3(void)
{
    int i,j;
    for (i=0; i < ITERATION1; i++)
    {
        for (j=0; j < ITERATION2; j++)
        {
            for ((j)=0; j < LONG_TIME; j++) // Allow time for context switch
                logMsg("ln",0,0,0,0,0,0);
            // Log messages
            c++;
        }
    }

//*******************************************

Result from hyper terminal:

0xb90ef8 (task1):
0xb90ef8 (task1):
0xb90ef8 (task1):
0xb8b8e90 (task2):
0xb8b8e90 (task2):
0xb8b8e90 (task2):
0xb86e88 (task3):
0xb86e88 (task3):
0xb86e88 (task3):
0xb90ef8 (task1):
0xb90ef8 (task1):
0xb8b8e90 (task2):
0xb8b8e90 (task2):
0xb86e88 (task3):
0xb86e88 (task3):
Appendix H

Message passing

// Message Passing
//
// A Task creates a message queue. Task 1 sends a message through
// a queue to task2. The second task prints out that
// a message has been received and sends a message back to
// task 1 through another queue. Task 1 now receives this message
// and prints out a receipt that everything has been acceptable
//
//******************************************************************************

// includes
#include "vxWorks.h"
#include "msgQLib.h"

// function prototypes
void MessTaskOne(void);
void MessTaskTwo(void);

// defines
#define MAX_MESSAGES 100
#define MAX_MESSAGE_LENGTH 50

// globals
MSG_Q_ID msgQueuId;
MSG_Q_ID msgQueuId2;
int result;

//****************************************************************************** - MessagePassing -******************************************************************************

int MessagePassing(void) // Function to create the message queue and two tasks
{
    int taskIdOne, taskIdTwo;

    // Create message queue
    if ((msgQueuId = msgQCreate(MAX_MESSAGES,MAX_MESSAGE_LENGTH,MSG_Q_FIFO)) == NULL)
        printf("msgQCreate in failed\n");
    if ((msgQueuId2=msgQCreate(MAX_MESSAGES,MAX_MESSAGE_LENGTH,MSG_Q_FIFO)) == NULL)
        printf("msgQCreate in failed\n");

    // Create the two tasks that will use the message queue
    if((taskIdOne = taskSpawn("t1", 90, 0x100, 2000,(FUNCPTR)MessTaskOne,0,0,0,0,0,0,0,0,0,0)) == ERROR)
        printf("taskSpawn t1 failed\n");
    if((taskIdTwo = taskSpawn("t2", 90, 0x100, 2000,(FUNCPTR)MessTaskTwo,0,0,0,0,0,0,0,0,0,0)) == ERROR)
        printf("taskSpawn t2 failed\n");

    return result;
}

//****************************************************************************** - Task 1 -******************************************************************************

void MessTaskOne(void) // Task that writes to the message queue
{
    char msgBuf2[MAX_MESSAGE_LENGTH]; //Creates a variable that can receive a message
    char message[] = "\nReceived message from taskOne"; //Message to be sent

    //Send message to MessTaskTwo
    if((msgQSend(msgQueueId, message,MAX_MESSAGE_LENGTH, WAIT_FOREVER,MSG_PRI_NORMAL)) == ERROR)
Appendix H

```c
printf("msgQSend in taskOne failed\n");

//Receive message from MessTaskTwo
if(msgQReceive(msgQueueId2,msgBuf2,MAX_MESSAGE_LENGTH, WAIT_FOREVER)== ERROR)
{
    printf("msgQReceive in taskTwo failed\n");
    result=0;
}
else
{
    printf("%s\n",msgBuf2);   //Print message
    result = 1;         //Message passing works
}
msgQDelete(msgQueueId2);  //Delete message queue
}

//******************************************************************************
//******************************************************************************

/* Task 2 */

void MessTaskTwo(void) //Tasks that reads from the message queue
{
    char msgBuf[MAX_MESSAGE_LENGTH]; //Creates a variable that can receive a message
    char reply[] = "Received reply from taskTwo\n"; //Message to be sent

    //Receive message from MessTaskOne
    if(msgQReceive(msgQueueId,msgbuf,MAX_MESSAGE_LENGTH, WAIT_FOREVER)== ERROR)
        printf("msgQReceive in taskTwo failed\n");
    else
    {
        printf("%s\n",msgbuf); //Print message
        if((msgQSend(msgQueueId2,reply,MAX_MESSAGE_LENGTH,
                      WAIT_FOREVER,MSG_PRI_NORMAL)) == ERROR)
            printf("msgQSend in taskOne failed\n");
    }
    msgQDelete(msgQueueId);  //Delete message queue
}

Result from hyper terminal:

Received message from taskOne
Received reply from taskTwo
```
Appendix H

Priority Scheduling

// Priority Scheduling
//
// Creates 3 different task with different priorities. The higher priority tasks are run before the lower priority tasks.
//
//
//*******************************************************************************
// Includes
#include "vxWorks.h"
#include "taskLib.h"
#include "logLib.h"

// Function prototypes
void taskOne(void);
void taskTwo(void);
void taskThree(void);

// Globals
#define ITERATION1 30
#define ITERATION2 1
#define LONG_TIME 1000000
#define HIGH 100   // High priority
#define MID 101    // Medium priority
#define LOW 102    // Low priority
int retvalue = 0;

//******************************************************
int PriorityScheduling(void)   // Function that creates the three tasks
{
    int taskIdOne, taskIdTwo, taskIdThree;

    //printf("\n\n\n\n\n\n");                          // Create three tasks . Task1 with diff. prio
    if((taskIdOne = taskSpawn("task1",LOW,0x100,20000,(FUNC PTR)taskOne,0,0,0,0,0,0,0,0,0,0)) == ERROR)
        printf("taskSpawn taskOne failed\n");   //Task2 with diff. prio
    if((taskIdTwo = taskSpawn("task2",MID,0x100,20000,(FUNC PTR)taskTwo,0,0,0,0,0,0,0,0,0,0)) == ERROR)
        printf("taskSpawn taskTwo failed\n");   //Task3 with diff. prio
    if((taskIdThree=taskSpawn("task3",HIGH,0x100,20000,(FUNC PTR)taskThree,0,0,0,0,0,0,0,0,0,0)) == ERROR)
        printf("taskSpawn taskThree failed\n");

    taskDelay(200);
    return retvalue;
}

//******************************************************
void taskOne(void)
{
    int i,j;
    for (i=0; i < ITERATION1; i++)
    {
        for (j=0; j < ITERATION2; j++)
            //logMsg("\n",0,0,0,0,0,0);            //Log message
        for (j=0; j < LONG_TIME; j++)         //Wait for a long time
            logMsg("\n",0,0,0,0,0,0,0);
Appendix H

```c
if (retvalue==2)
    retvalue=1;
else
    retvalue=0;

void taskTwo(void)
{
    int i,j;
    for (i=0; i < ITERATION1; i++)
    {
        for (j=0; j < ITERATION2; j++)
        {
            for (j=0; j < LONG_TIME; j++) // Wait for a long time
                logMsg("n",0,0,0,0,0,0);
        }
        if (retvalue==1)
            retvalue=2;
        else
            retvalue=0;
    }

void taskThree(void)
{
    int i,j;
    for (i=0; i < ITERATION1; i++)
    {
        for (j=0; j < ITERATION2; j++)
        {
            for (j=0; j < LONG_TIME; j++) // Wait for a long time
                logMsg("n",0,0,0,0,0,0); // Log message
        }
        if (retvalue==0)
            retvalue=1;
        else
            retvalue=0;
    }
```

Result from hyper terminal:

0xb86e88 (task3);
0xb8bec0 (task2);
0xb90ef8 (task1);
Timing

// Time testing
//
// This test reveals the system clock rate and allows the user to
// find out how many ticks are needed to delay a certain amount of time.
//
//
//********************************************************************************
//
//* includes */
#include "vxWorks.h"
#include "stdio.h"
#include "taskLib.h"

#define TIME sysClkRateGet()  // Defines the amount of ticks the program is delayed
#define Any_Time 500.0

void time_test (void)
{
    printf("\n\tSystem Clock Rate is : %.1u tics per sec\n".TIME);
    printf("\n\tThe delay was set to %.0f ticks\n". Any_Time);
    printf("\n\tStarting Delay\n");
    taskDelay (Any_Time);
    printf("\n\tDelay Finished!!\n");
    printf("\n\tDelay was %.1f sec\n". Any_Time / TIME);
}

Result from Tornado Shell:

System Clock Rate is : 60 tics per sec

The delay was set to 500 ticks

Starting Delay

Delay Finished!!

Delay was 8.3 sec

Time test completed
Appendix H

Memory (Ada)

-- Memory testing
--
-- This program checks the parameters of the RAM memory. The code sets the wanted
-- RAM storage area. The program then reads at different addresses to confirm the
-- correctness of its contents. This is done by checking for zeros before the RAM starts,
-- and also just at the end of its memory space. It also checks the first memory space
-- in RAM, which should not be equal to zero.

with Ada.Text_IO;
use Ada.Text_IO;
with System;
with System.Storage_Elements;
use System.Storage_Elements;

procedure Memory is

    Memory_Start : constant := 16#100_000#-8;  -- Starts at 8 bits before beginning of RAM storage area
    Memory_End : constant := 16#200_000#-8;  -- Starts at 8 bits before end of RAM storage area
    Bool : Boolean := False;

-- This procedure will read the data stored on the given address. It also performs the check if the data is correct
procedure Read_Mem (Addr : System.Storage_Elements.Integer_Address ) is

    Test_Mem : Integer;
    Tst : Integer;
    Ad : System.Storage_Elements.Integer_Address;

    for Test_Mem use at System.Address
    (System.Storage_Elements.To_Address (Addr));
    begin
        Ad := Addr;
        Put (System.Storage_Elements.Integer_Address'Image(Addr));
        Put(" : ");
        Tst := Test_Mem;
        Put_Line(Integer'Image(Test_Mem));
        if (Ad >= Memory_Start and Ad < Memory_Start+8) or (Ad >= Memory_End and Ad < Memory_End+8) then
            if Tst /= 0 then
                Put_Line("******** Test failed! Should be 0 ********");
                Bool := True;
            end if;
        end if;
        if (Ad = Memory_Start+8) then
            if Tst = 0 then
                Put_Line("******** Test failed! Should not be 0 ********");
                Bool := True;
            end if;
        end if;
    end Read_Mem;

begin
-- The loop below increases the address by 4 and is repeated 16 times.
-- The address is sent to the procedure Read_Mem
for I in 0..16 loop
    Put(Integer'Image(I));
    Read_Mem(Memory_Start + System.Storage_Elements.Integer_Address(I*4));
end loop;
Appendix H

-- The loop below increases the address by 4 and is repeated 16 times.
-- The address is sent to the procedure Read_Mem
for I in 0..16 loop
    Put(Image('Integer')(I));
    Read_Mem(Memory_End + System.Storage.Elements.Integer_Address(I'4));
end loop;
Put_Line("************************************************************************");
if Bool = False then -- Checks the result of the test
    Put_Line(" Test finished successfully");
else
    Put_Line(" Test failed!!");
end if;

end Memory;

Result from Tornado Shell:

0 1048568 : 0
1 1048572 : 0
2 1048576 : 2086869624
3 1048580 : 2086666532
4 1048584 : 2080375416
5 1048588 : 2081440678
6 1048592 : 2081506214
7 1048596 : 2081571750
8 1048600 : 2081637286
9 1048604 : 1885536256
10 1048608 : 1617108992
11 1048612 : 2080375980
12 1048616 : 2086666532
13 1048620 : 1275068716
14 1048624 : -8388340
15 1048628 : -16776948
16 1048632 : -25165556
0 2097144 : 0
1 2097148 : 0
2 2097152 : 1207959605
3 2097156 : 1207959609
4 2097160 : 1131376761
5 2097164 : 1919510376
6 2097168 : 194826809
7 2097172 : 942943537
8 2097176 : 960051488
9 2097180 : 1466527332
10 2097184 : 542271862
11 2097188 : 1701978195
12 2097192 : 2037609573
13 2097196 : 1836264480
14 2097200 : 1231971118
15 2097204 : 962592770
16 2097208 : 1207959561
************************************************************************
Test finished successfully