ERC32 Tool Set Evaluation

CRI Doc. no.:	***CRI/ERC32/TN/001***
Date:	***27.06.97***
Issue:	***1***
Revision:	***-***
Distribution:	***ESTEC, CRI***
Prepared by:	***Christian Maegaard***
Approved by:	***Lars Behrendt - QA***
Authorised by:	***Carsten Jørgensen***

ERC32 Tool Set Evaluation

June 1997

Document Change Record

Issue	Date	Change
1	27.06.97	Initial issue of document

Document Status Sheet

Page	Issue
i–vi	1
1–22	1

Table of Contents

1 Introduction **1**
- 1.1 Scope . . . 1
- 1.2 Abbreviations and Acronyms . . . 2
- 1.3 Terms and Definitions . . . 2
- 1.4 Document Outline . . . 2

2 Document References **3**
- 2.1 Applicable Documents . . . 3
- 2.2 Reference Documents . . . 3

3 Evaluation Overview **4**

4 AdaWorld Compiler System for ERC32 **5**
- 4.1 Description . . . 5
- 4.2 Porting Experiences . . . 5
- 4.3 Delay_Until . . . 6
- 4.4 Passive Tasks . . . 7
  - 4.4.1 Immediate Priority Ceiling Inheritance . . . 8

5 WCETE for ACS **10**
- 5.1 Description . . . 10
- 5.2 Evaluation . . . 10

6 Schedulability Analyzer **13**
- 6.1 Description . . . 13
- 6.2 Evaluation . . . 13
  - 6.2.1 Describing the System Model . . . 14
  - 6.2.2 Performing the Analysis . . . 15

7 Scheduler Simulator **17**
- 7.1 Description . . . 17
- 7.2 Evaluation . . . 17

8 ERC32 Target Simulator **18**
- 8.1 Description . . . 18
- 8.2 Evaluation . . . 19
  - 8.2.1 Testing with the ERC32 Target Simulator . . . 19
  - 8.2.2 Performance . . . 19
  - 8.2.3 Timing Verification . . . 20

8.2.4 Debugging 21

9 Evaluation Summary 22

A Protected Object Test A.1

B Schedulability Analysis Results B.1
B.1 Input to CRI Schedule Program (null RTS) B.2
B.2 Output from CRI Schedule Program (null RTS) B.3
B.3 Input to SpaceBel Schedulability Analyzer (null RTS) B.5
B.4 Output from SpaceBel Schedulability Analyzer (null RTS) B.9
B.5 Input to CRI Schedule Program (AdaWorld RTS) B.10
B.6 Output from CRI Schedule Program (AdaWorld RTS) B.12
B.7 Input to SpaceBel Schedulability Analyzer (AdaWorld RTS) B.14
B.8 Output from SpaceBel Schedulability Analyzer (AdaWorld RTS) . . . B.14

C Scheduler Simulator Results C.1

1 Introduction

1.1 Scope

This report documents the evaluation of the ERC32 tool set performed by CRI for ESTEC as described in [Proposal].

The evaluation covers:

1. ERC32 Ada World Compilation System
   Version: V5.5.4
2. WCET Estimator
   Version: AdaWorld V5.5.4
3. Schedulability Analyzer
   Version: HRT tools V2.2
4. Scheduler Simulator
   Version: HRT tools V2.2
5. ERC32 Target Simulator
   Version: V2.2

The evaluation uses the ERA on-board software as reference - specifically the delivered ASW SL V0 software is used.

A debugger and a target H/W was not available for the evaluation activities. These are very crucial parts of an SDE. It is therefore outside the scope of this evaluation to assess if a *full* SDE for ERC32 based on these tools can be used professionally.

The evaluation is performed as a product evaluation and not as a project evaluation. This means that it only evaluates whether tools would be useful in developing real-time on-board software and not if the projects developing the tools have reached a reasonable level given their development constraints.

1.2 Abbreviations and Acronyms

ADS	Arbitrary Deadline Scheduling
ASW	Application Software
BSP	Board Support Package
DMS	Deadline Monotonic Scheduling
ERA	European Robotic Arm
ESF	Execution Skeleton File
HRT	Hard Real-Time
HRTAF	Hard Real-Time Attributes File
ICPI	Immediate Priority Ceiling Inheritance
LRM	Language Reference Manual
RMS	Rate Monotonic Scheduling
RTCF	Run Time Characteristics File
SDE	Software Development Environment
SL	Service Layer
UCF	User Configuration File
WCET	Worst Case Execution Time
WCETE	Worst Case Execution Time Estimator
WCCT	Worst Case Computation Time
WCRT	Worst Case Response Time

1.3 Terms and Definitions

There are no special terms and definitions used in this document.

1.4 Document Outline

Chapter 2 contains a list of reference documents.

Chapter 3 contains an overview of the activities performed during the evaluation.

Chapters 4 to 8 provide a detailed description of the experience obtained when using each of the tools.

Chapter 9 provides an overall summary of the evaluation.

Appendix A provides some example sources used in the AdaWorld evaluation.

Appendix B provides some examples of runs with the SpaceBel Schedulability Analyser and the equivalent CRI tool.

Appendix C provides some examples of runs with the SpaceBel Scheduler Simulator.

2 Document References

2.1 Applicable Documents

The document which is applicable for this document is:

[Proposal] Evaluation of ERC32 tool set, CRI/OBSD/97/ERA/FAX/048

2.2 Reference Documents

The documents, except for the applicable document, which are referenced in this document are:

[TS]	Target Simulator User's Manual, 32B-SBI-SUM-0189-003, Issue 2.1
[HRT]	Schedulability Analyser and Scheduler Simulator Software User Manual, 32B-SBI-SUM-0189-001/2, Issue 1.3
[VHDL]	ERC32 VHDL models user's manual, Version 1.0
[WCETE]	Worst Case Execution Time Processing Preliminary User's Guide, RAT-ALS-WP20-PUG-001, Issue 2.0
[RMA]	A Practitioners Handbook to Real-Time Analysis, Mark H. Klein et al, Kluwer Academic Publishers

3 Evaluation Overview

As defined in [Proposal] the evaluation covered by this report consists of the following activities

1. Porting relevant parts of the SL software and tests to the AdaWorld compiler and using the ERC32 Target Simulator to execute the tests.
2. Applying the WCET extractor to the relevant SL software to obtain WCETs for protected operations and cyclic threads.
3. Compare the WCET extractor to the method used previously by CRI.
4. Apply the Schedulability Analyser and the CRI developed scheduling analysis program to the HRT attributes of the ERA ASW, and compare the results.
5. Apply the scheduler simulator to obtain a graphical representation of the obtained scheduling and compare it with the results found by the analysis above.

The evaluation is structured as one chapter for each tool being evaluated and a summary of the complete evaluation in the end.

4 AdaWorld Compiler System for ERC32

4.1 Description

The AdaWorld compilation system is a full cross development environment. It consists of

1. Compiler
2. Binder/Linker
3. Debugger
4. Make facility
5. Cross reference tool
6. Math library
7. Real-time extensions
8. Additional tools of minor importance

It is out of the scope of this evaluation to have a full walk-through of the complete product. Here it should suffice to say that the product seems very complete, and the only tool missing would be a statistical profiler (like the one included in VADS).

Instead the activities required to port the ERA ASW SL V0 software, and a more detailed evaluation of the real-time extensions is described.

The real-time extensions comprise absolute delays (delay_until) and protected objects (passive tasks).

4.2 Porting Experiences

As it was to be expected it took some time to get acquainted with all the different tools of the compiler system, in order to build executables of the ERA ASW SL V0 acceptance tests.

Specifically the library manager is much more complex than really needed. It therefore took some time to set up an appropriate library structure. The problem is that both library to library links as well as unit to unit links are possible. But one library can only link to one other library (parent) so you often have to use both mechanisms. The solution used in e.g. VADS where each library can depend on several others (a network) seems more elegant and easy to use.

After solving the library problems, the main remaining problems related to the passive task implementation that was a little bit different from the VADS implementation (see also below). One passive task had to be active in order to allow timed entry call, and all other passive task specifications had to be moved from package specifications to package bodies.

The last problem is that the test driver typically was not assigned a priority so it would get assigned a default one. This lead to a ceiling priority violation and program_error in most tests.

After correcting these problems, it was possible to execute 5 out of 7 tests correctly except for file IO, which was not supported by the board support package used. The remaining 2 resulted in program_error apparently occurring during a memory allocation in the Ada run-time. This was not further investigated since no debugger was available.

4.3 Delay_Until

For cyclic activities with a fixed period it is required to be able to make absolute delays.

In Ada83 this is typically done by a scheme like

```
Next := Calendar.Clock + Period;
loop
   delay Next - Calendar.Clock;
   Some_Processing;
   Next := Next + Period;
end loop;
```

This way drifting is avoided (the reason for introducing Delay_Until according to the AdaWorld documentation). But one problem remains if interrupts are not disabled during the complete execution of the delay statement. In that case the cyclic task could be suspended right after calling Calendar.Clock but before executing the delay, then the relative delay would be wrong when the delay was eventually executed. This race condition is what is avoided by the Ada95 construct **delay until**.

Instead of extending the language, a new library package Real_Time is provided with AdaWorld. This package provides a new definition of time (corresponding to Calendar.Time) and Time_Span corresponding to duration and conversion routines for the new types. Further it provides a Delay_Until procedure that accepts

Real_Time.Time as well as Calendar.Time;

The only problem when comparing to Ada95 is that Delay_Until cannot be used as an alternative in a select statement.

Some Ada83 implementations actually disable interrupts during execution of delay statements and thus provide a functionality similar to Ada95.

Often the use of **delay until** in select alternatives can be avoided, and the chosen implementation is therefore fine for most purposes.

4.4 Passive Tasks

The passive tasks have been introduced to cover 2 functionalities:

- Data protection (critical region)
- Sporadic release of one task by another (similar to binary semaphore)

Both of these could be covered by the standard Ada rendezvous mechanism. This construct is however very time consuming and therefore to be avoided in real-time applications.

To solve the problem there were in principle 3 options:

- Extend the language with the Ada95 construct *protected object*.
- Stay within the Ada83 syntax and use a specific pragma (PASSIVE) to instruct the compiler not to generate a real thread for a task but instead a passive set of procedures guarded by semaphores.
- Stay within the Ada83 syntax and provide direct access to a semaphore type and semaphore operations, through an Ada package (equivalent to the package Synchronous_Task_Control defined in the real-time annex D.10 of Ada95 LRM).

In AdaWorld the pragma is chosen and the same pragma as used in VADS is used. This is consistent, since an extension with the Ada95 protected objects would mean a compiler that is neither Ada83 nor Ada95. However the possibility of a direct access to the semaphores would have been easy to provide and given added flexibility to the user.

Since the Ada83 tasking model with rendezvous is much more powerful than providing the 2 functionalities above, a set of restrictions apply when pragma passive is specified for a task. These restrictions are documented in Appendix F of the LRM.

When compiling and running the ERA ASW sources only 2 of these restrictions were a problem (the same restrictions did not apply for VADS, nor Ada95 protected objects):

1. Task specification of a passive task must only occur in a package body.
2. Timed entry calls are not allowed.

The first restriction is not mentioned in the appendix F and was quite difficult to identify since the compiler only gave the message “****MAN Compilation stopped in EXPANDER at line 584 due to a compiler limitation.” and then a message that this was an internal exception in the compiler (apparently occurring in a module called EXPANDER). In the ERA ASW the protected object specifications have generally been placed in the package specifications to allow renaming of the entries. The cure is of course straight forward - move all passive task specifications to the package bodies and replace the rename by an in-line procedure call. The only problem with this is that it creates some extra body dependencies in the compilation order.

The second restriction is problematic when handling of non-nominal cases are required. For example in the ERA ASW it has to be detected if no telecommand has been received for 200ms. Generally the TC handler is activated sporadically by the TC buffer when something has been entered. Therefore it calls the TC buffer with a timed entry call, and starts an action to bring the arm into a safe state, if a timeout occurs. Several similar applications of this construct are used in the ERA ASW. There is no easy workaround, but probably if the restriction was known in advance a different design could have been made - e.g. based on cyclic checking of task activities.

4.4.1 Immediate Priority Ceiling Inheritance

In order to have an upper bound on the blocking each task in a set of tasks can experience due to access to protected objects, the *Immediate Priority Ceiling Inheritance* (IPCI) protocol is used (equivalent to the locking policy *Ceiling_Locking* of Ada95).

The mechanism is that the passive task can be assigned a priority that all its operations will be executed with, independently of the priority of the caller. When this priority is always selected to be at least the maximum priority of any of the callers (restriction) and all tasks have unique priorities (restriction if using the deadline monotonic scheduling model) this will ensure that each task will only be blocked once after each activation.

Unfortunately this mechanism in connection with barriers is not well described in appendix F. To test this, a test program is provided in appendix A.

The program contains 2 tasks, one having a low priority and one having a high priority. The low priority task calls a wait operation (entry with barrier) in a protected object. Then the high priority task calls a signal operation (opening the barrier) 2 times.

The low priority task registers itself when the wait is being performed and after the wait. The high priority task registers itself when it calls signal and between the 2 calls. The registration means adding a task identification (the priority is used since it is unique) to the sequence of registrations. The output is:

Registration order: 2, 1, 2, 2, 1.

It can be seen that immediately after exiting the signal operation of the high priority task, the low priority task takes over. This means that the priority is raised even while the task is waiting on the barrier or at least as soon as the barrier becomes true. This also means, as it can be seen, an extra double context switch. The low priority task is switched in only to complete the wait operation and then the high priority task continues its operation. When the high priority task is completed, the low priority task is switched in again to complete its operation. It is obvious that the desired output would have been:

Registration order: 2, 2, 2, 1, 1.

This means that the low priority task is only switched in when the high priority task is suspended. This behavior would appear if the low priority task were only made ready by the end of the signal operation but not considered inside the protected object. In this case it could have kept its priority. With the current implementation there is not much gain in using sporadic release from a passive task over a normal Ada rendezvous, where the overhead will also be 2 context switches. Another maybe even more important point is that the priority inversion is not very well bounded (see section 6.2.2).

A possible workaround could be to delete the pragma priority from the passive task. It would then be expected that each operation would be executed with the priority of the calling task. The expected output would therefore be (assuming that the signal operation still lets any waiting task get immediate access to the protected object):

Registration order: 2, 2, 1, 2, 1.

This behavior would be acceptable for all situations where the processing performed after a wait was released must always complete before the next signal. This is the most usual way to use sporadic activation. Unfortunately the output remains the same as when a fixed priority is specified.

This shows an obvious need for a direct access to a simple semaphore type as found in other Ada environments (e.g. TLD, VADS, and DDC-I) and incorporated in Ada95 real-time annex.

5 WCETE for ACS

5.1 Description

To know whether a given application can satisfy its timing constraints the first step is independently of the design and/or scheduling model to determine the time spent by execution of the different fragments of the code.

The Worst Case Execution Time Estimator [WCETE] is intended to provide an upper bound on the execution time for each of the critical threads of an application.

This output is an essential input to the overall schedulability analysis - see section 6.

Though the WCET for the different code fragments are interesting for any real-time system, the WCETE is directed specifically towards systems based on one of the scheduling theories, DMS or ADS.

The WCETE is built as a part of the AW compiler (ref. section 4). This means that when a specific compile option is given, the compiler will generate an execution skeleton that can be read by the Schedulability Analyser (ref section 6).

The tool uses an instruction timing file that describes the number of cycles used by each instruction in the instruction set. In the current version this file cannot be altered. This means that instructions with a variable execution time will have a fixed execution time in the estimator. It would be reasonable to be able to select between worst case and typical numbers.

For the estimator to work, a number of restrictions must be obeyed by the source to be analyzed. Some of the restrictions are introduced to make the job easier for the WCETE (referred as WCET Ada Application Restrictions and WCET Ada HRT Restrictions) - these restrictions match quite well with the restrictions of the Schedulability Analyzer. The remaining restrictions are to exclude constructs that result in unbounded execution time.

5.2 Evaluation

In principle there are two ways to obtain the WCET of the different operations.

- Theoretical calculation based on WCET for the instructions used.
- Measurement of the execution time, using the input to select different branches.

The WCETE uses the first approach. The second approach is what is normally used since it is completely unrealistic to perform the theoretical calculation without tool support.

The plan for evaluating the WCETE was to take some fragments of the ERA ASW on-board software and apply both methods. Comparing the results would then give some idea on the usefulness of the WCETE.

However the restrictions in the current version (prototype) are so severe that any practical example is not allowed. The example that was selected was a routine to determine the pose of ERA from a pixel image produced by one of the ERA cameras. This was selected because it was relatively simple to separate from the remaining ERA software. However the restrictions of not using array/record assignment, bit level representation clauses, exponential operators, and integer arithmetic were all violated. There was no way of working around these restrictions. The work-arounds proposed in [WCETE] could have solved a few of the problems but only at the cost of extremely bad performance, others could not be solved. Instead of making up a toy example it was decided to discuss only how the tool could be used in a development situation if all the prototype restrictions were removed.

The remaining restrictions which would be applicable also for operational versions of the tool also seem a bit too severe. It is of course reasonable to exclude any operations resulting in unbounded execution time. However the list of these constructs includes operations on composite types as assignment, comparison, and relational, and logical operators. If it is really true that these operations are unbounded it is a serious problem in using the Ada World compilation system for real time systems. In any case this restriction must be removed for the tool to have any practical use. The “Application” and “HRT” restrictions also appear to be over restrictive. However, if all restrictions are known from the beginning of a project and the tool is used right from the start, it may be possible to cope with these restrictions. One should however keep in mind that when producing a HRT system, performance is often very important. Too many restrictions may lead to less performant implementations.

It is not described in [WCETE] how the estimation is performed. It must be assumed that the method is to make a recursive decent through the parse tree until the instruction level is reached and then for each branch point select the highest WCET for any of the possible branches. This will ensure an upper bound. This may however be unrealistically high. For example

```
if A > B then
    B:=A;
end if;
if A < B then
    A:=B;
end if;
```

In this situation the theoretical worst case is of course the worst case of one if statement plus the best case (condition evaluation only) of one if statement and not

the sum of the two worst cases. It would therefore be nice to know in more detail how the tool works.

It is in general doubtful if a WCET produced by theoretical calculations based on instruction execution time can be used to gain a realistic upper bound - i.e. a bound that is not too far from the WCET that can result by applying actual data. The true number would in principle be the maximum number obtained by applying all possible combinations of input data and measuring the execution time for each run. In practice you would probably have to use a combination of the 2 methods.

The implemented WCETE has a number of advantages. It is directly integrated with the Ada World compiler and provides output that is readable by the Schedulability Analyzer. Therefore even if the produced values cannot be used at least it can be used to generate a template for the Execution Skeleton. Even here it is however a problem that it will only do that if all the severe restrictions were met.

6 Schedulability Analyzer

6.1 Description

When developing a hard real-time system, based on several tasks, verification of the timing constraints is not trivial. Specific theories - *rate monotonic scheduling* (RMS), *deadline monotonic scheduling* (DMS) - have been developed to facilitate a static verification that all tasks meet their deadlines, i.e. the system is *schedulable*.

The analysis required to determine if a system is schedulable is to look at the properties of each task (WCET, deadline, potential blocking time, period/minimal inter-arrival time) and derive the the worst case response time (WCRT).

The theory for performing this analysis is well known but complex, and therefore it is helpful to have a tool with this capability.

The schedulability analyzer implements these algorithms, and also helps in determining the blocking time and run-time system overhead to take into account.

The input to the tool is a *user configuration file* (UCF), an *execution skeleton file* (ESF), and a *run time characteristics file* (RTCF). The UCF describes the overall properties (not expected to change during development) of each task: name, deadline, and period/minimal inter-arrival time. The ESF describes the dynamic properties of each task and each protected object in the design by giving WCET and list of protected calls for each task and WCET for each protected object. The RTCF describes the time used by the RTS for its different activities such as context switching.

The potential blocking for each task is determined by looking at each protected object it calls and the WCETs for each operation in this object.

The WCET for the task is then increased by the RTS overhead determined from the RTCF.

Now the basic input for the schedulability analysis is ready and all WCRTs are calculated and a schedulability report is generated including sensitivity margins.

6.2 Evaluation

The evaluation of the tool is performed by entering the data for the ERA ASW. Previously on the ERA project we have used a proprietary CRI tool, and we will

compare the two in this evaluation.

6.2.1 Describing the System Model

Before activating the tool we need to describe the system model. The first step is to describe the static properties of the model. The UCF does that for the Schedulability analyzer. The CRI schedule tool takes only one combined input file - *HRT Attributes File* (HRTAF). An example of equivalent files for the 2 tools can be found in appendix B. As it can bee seen most of the information in the UCF is identical to the information in the HRTAF. The main difference is that the HRTAF does not distinguish between minimal inter arrival time and period, since it is treated identically by the analysis theory.

Then we need to describe the dynamic properties of the now defined thread set. This is done by specifying WCET for all threads and potential blocking. For the SpaceBel tool this is done by specifying

1. WCET for all threads (excluding access to protected objects and RTS overhead)
2. Call sequence of protected operations for all threads
3. WCET for all protected operations and barrier evaluations

For the CRI tool this is specified by

1. WCET for all threads (excluding access to protected objects and RTS overhead)
2. Potential blocking time for each thread (must be determined based on Call sequence of protected operations for all threads and WCET for all protected operations).
3. the mode in which the numbers apply.

As it is seen, the main difference here is that the SpaceBel tool derives the potential blocking time, while the user must perform this analysis himself for the CRI tool. Further the user has to deduct from the thread WCET the different overheads when using the SpaceBel tool. This is no problem when using the WCETE of course, but is quite tedious if measurements were used.

Another difference is that the CRI tool allows different numbers to be specified for different modes of the system. For example when ERA makes a proximity move (using image processing) the task c_prepare_vis_tac is part of the thread set otherwise not. But the WCET of c_2hz_checks is highest during free motion due to an otherwise inactive singularity check. It is therefore not possible to say which of the 2

modes will be the worst case without performing the analysis. The tool then takes as input the mode for which the analysis shall be performed. A similar facility in the SpaceBel tool would be beneficial.

In the example we have entered in the HRTAF the blocking time that the SpaceBel tool reports after Analysis.

6.2.2 Performing the Analysis

To validate the analysis performed, the 2 tools where activated with these inputs. Their inputs and outputs can be found in appendix B. The CRI tool is based on the theory described in [RMA] for deadline monotonic analysis with arbitrary deadlines and blocking. The same seems to apply for the SpaceBel tool (the theory is called ADS in [HRT]) and the same WCRT can also be observed.

From the SpaceBel tool we further get a sensitivity analysis, telling how much room there is for extending the WCET of each task. This feature is quite useful.

Let's now look into the figures produced by the tool - ref. appendix B.4. In this example we have used a null RTCF and the analysis is therefore simple to follow.

For example we can see that the WCCT for task 1 is the WCET for the thread plus the WCET for the protected entries it calls. It can also be observed that when calling a protected object with a barrier expression, the barrier evaluation time is included twice for calls to the guarded entry, and not included in the call to the unguarded entry. However the implementation of the pragma passive makes one evaluation at the entry of the guarded action and one at the exit of the unguarded action (documented in LRM appendix F). Therefore the WCCTs are not correct.

Then we can look at the blocking time. Blocking of a thread A occurs if a thread B with lower priority is executing after the activation but before the completion of A. This can only happen when protected objects are being accessed. For example task 1 is accessing three "Synchro" protected objects and one "Resource" protected object. The worst case scenario is then that one task (13) has just entered the some_entry protected action, and that 3 tasks (3,5,11) are all waiting on guarded entries. That would lead to a blocking corresponding to the execution of each of these 4 entries - i.e. 1.0 ms + 1.0 ms + 0.2 ms + 0.9 ms = 3.1 ms. This follows from the AdaWorld implementation of the ICPI described previously (section 4.4.1). So it seems that also the blocking times are not correct since the SpaceBel tool produced 1.5 ms.

In the real world the RTS of course has an overhead. Therefore we now make a new run with the RTCF corresponding to a 10 MHz ERC32 using the AdaWorld RTS.

The tool now calculates the WCCT for each thread as the WCET + overheads. This value was copied to the HRTAF to see if the new analysis would still show equal WCRTs. It is observed that now the 2 tools show different WCRTs. Whether this is a problem or not is difficult to assess since it is not obvious how the RTS overheads have been included.

In assessing the validity of the results it is a problem that the translation from the system model in the UCF, ESF and RTCF to a model that can be treated by well known schedulability analysis theories (RMS, DMS, ADS) is not provided. The tool is presented as a magic black box that tells you if the system is schedulable or not.

The fact that RTS overheads are handled by the tool in general makes life easier for the user. In some situations it is however better to have the freedom to specify them manually. For example In the ERA ASW the task c_motion_execution is actually also performing the activities of c_prepare_vis_tac (which is not really a task).

This means that the WCET of c_motion_execution is toggling between 3.1 and 18.1. This could of course be modeled as one task with a period of 50 ms and an execution time of 18.1, but it would lead to a utilization that is too high and maybe even not schedulable threads. Instead it is modeled as 2 threads and RTS overhead is only included for the 50 ms thread. Such a model is not possible with the SpaceBel tool.

7 Scheduler Simulator

7.1 Description

The Scheduler Simulator is a tool that shows to the user the dynamic behavior of a thread set defined as for the Schedulability Analyser.

It simply schedules all tasks to start at time 0 and then continues using their WCCT and period/minimal inter arrival time. It reports the sequence of events where the RTS is involved.

It is possible to view the report in text or graphically as a Gantt chart. It is possible also to filter the threads and events that should be reported.

7.2 Evaluation

The Scheduler Simulator is probably mostly relevant as an educational tool to help the understanding of scheduling a thread set. To a person who is familiar with these problems it will not provide any new insight not available from the Schedulability Analyser.

It is always scheduling starting from the so called *critical instant* where all tasks are activated at time 0. However to show the worst case scenario for task 1 as discussed above, 4 of the lower priority tasks should already have reached protected entries.

To benefit from the simulator it should be possible to select a thread from the thread set and have the worst case scenario for this thread simulated. The critical instant in our example is actually only the worst case for the soft thread with the lowest priority and therefore of less interest.

In appendix C the output generated from the tool using the same input files as for the Schedulability Analyser (AdaWorld RTCF) can be found. It appears that the system that was reported schedulable by the Schedulability Analyser has missed deadlines according to the Scheduler Simulator!

In general the WCRTs does not seem to match the event log.

8 ERC32 Target Simulator

8.1 Description

A target simulator is in general used to replace the actual target hardware in the development process. The reasons for doing that can be several:

- Non-availability of target hardware.
- Added execution control.
- Better interface to the host development environment.

The use of the simulator is approximately the same as for the hardware:

- Testing
- Timing verification
- Debugging

The ERC32 target simulator [TS] supports all these areas. It is constructed as a simulation process that connects to other UNIX processes through pipes. This flexible design allows one to connect either the user interface that is part of the simulator product or the AdaProbe debugger part of the Ada World Compilation system. Further it is possible to connect it to a user interface developed by the user, though this is not well described.

The hardware can be described in a configuration file and the operation can be commanded through the selected user interface.

The commands include what will be expected by any debugger like: execution, breakpoints, single stepping, and memory/register read and write. Further it has more advanced controls named markers and watchpoints. Using these it is possible to log the time when a certain address is accessed or to break the execution at that moment. This can be valuable for timing measurements or debugging.

8.2 Evaluation

For the purpose of the evaluation of the target simulator, it was attempted to use it for testing and timing measurements. Debugging was not possible since the AdaProbe was not available for the evaluation.

8.2.1 Testing with the ERC32 Target Simulator

It was attempted to port all the SL V0 acceptance tests to the ERC32 target and run them under the simulator. The port and the problems involved are described in chapter 4.

The tests were first run by using the user interface. It was quite easy to operate the simulator.

A normal test setup is however based on the following

- An automated test execution
- An automated verification of the test output

The first problem encountered was to make an automated execution by a UNIX script. This very usual application was not described in [TS]. It was however possible to make a working setup, though it was quite complicated due to the many pipes that had to be kept open. In this situation the flexible design was not very user friendly and it would have been helpful if a script for executing an ERC32 program from the UNIX prompt was provided.

The next problem was to get the output. Normally these tests would write their output to a named file. This concept is however not handled by the combination of simulator and the IO packages (BSP) in the executable. The problem has no easy solution, one possibility could be to write the test results in a specific memory area and then use the debugger dump command. The problem is that the dump is done in hexadecimal and therefore not readable. A post processing is therefore needed. This solution can however only be used if the test harness is developed by the user. If a commercial test harness - e.g. AdaTEST - is used, IO must be performed through the normal Text_IO package. This means that all communication must be sent via the serial channel mapped to standard input and output. That excludes some usage of AdaTEST but the tool can be generally used even in this configuration. The serial communication has one problem - it is extremely slow on the target simulator.

8.2.2 Performance

When running a test it is of course of primary importance that the execution time is not too high. Unfortunately this is a very weak point for the Target Simulator.

A comparison to the ESA developed SIS 2.7.1 simulator showed a performance ratio of about 1:13. The simulated time/real time ratio was measured to approximately 1:100 when executed on a 170 MHz UltraSPARC. On a SPARCstation 4 (110 MHz microSPARC) which is a normal development station on the ERA ASW team, the ratio is about 1:500.

This means that the simulator needs a very high level of optimization before it can be used in a development environment.

On the contrary the SIS 2.7.1 simulator has a performance that makes it realistic to use for development, unfortunately its features and interface are quite primitive.

8.2.3 Timing Verification

For assuring the system's timing properties the basic values to determine are the WCET for the different activities in the software.

As discussed in section 5 there are in principle 2 ways of obtaining these numbers: Measurement and theoretic calculations. When measurements are used it is of course important that the simulator timing is correct compared to the real hardware.

The problem with ERC32 is that the FPU has different execution speed for different input for multiplication and divisions. It is not mentioned in [TS] how this problem is solved in the target simulator. It is mentioned that the tool is validated against the VHDL model - it is unclear whether this assures correct timing for these instructions. In [VHDL] it is stated that "In the real device, FPU instructions execute in a varying number of cycles depending on the operands. In the model, the FPU instructions execute in the number of cycles as defined in the typical case in the FPU data sheet." When comparing to the SIS 2.7.1 (which uses always typical values) there is a difference that could indicate that the target simulator uses correct values. It is essential that the instruction timing is not too optimistic (it is for SIS 2.7.1).

For performing the timing verification you can use either coded traces where calls are made to Real_Time.Clock or use the time markers provided by the target simulator. The problem with the time markers is to find the correct addresses. The simulator does not know the symbolic names used in the Ada source.

In this evaluation we therefore used the approach where calls were inserted in the code to obtain the system time. In general the simulator proved useful for the purpose of timing verification.

8.2.4 Debugging

When debugging a real-time system using a target debugger, there are in principle 2 different implementations:

- Intrusive debugging
- Non-intrusive debugging

Intrusive debugging is characterized by the need to have special debugger software (e.g. ALSYMONITOR) loaded on the target together with the software under test. This monitor software then communicates with the debugger over a serial (or other) communication channel. Breakpoints are created by modifying the code to have special traps inserted.

Non-intrusive debugging is characterized by having a controller separated from the target, that can detect memory fetches execution address etc. Further it is able to halt the processor by stopping the clock. This controller then communicates with the debugger over a dedicated channel.

The clear advantage of the non-intrusive debugging is that the system behavior is not changed because it is being debugged. Problems with the intrusive debugger could be

1. Time spent by the monitor is not included in the schedulability analysis.
2. The clock ticks while the software is stopped e.g. at a breakpoint.
3. The available memory is different because parts are used for the monitor.
4. The monitor changes the code when inserting breakpoints - i.e. we are testing instrumented code.

Non-intrusive debugging on real hardware is always difficult because it requires special hardware solutions to allow the required level of external control of the target processor. Also the use of a pipeline makes breakpoint control difficult. However, the advantage of a simulator is that it is quite easy to get full control of the “target”. The target simulator provides a non-intrusive interface to AdaProbe, which is expected to be very helpful in a development process.

This is a clear advantage over the DEM32 boards and the SIS simulator that allows only the intrusive version using ALSYMONITOR. How well integrated AdaProbe and the target simulator (i.e. what is the level of control that can be obtained from AdaProbe) is, is not evaluated since AdaProbe was not available for the evaluation.

9 Evaluation Summary

As a general statement most of the evaluated tools show good potentials while still having some flaws that must be removed before they are fully ready for professional use.

The AdaWorld compiler is a very complete Ada compiler and the number of problems faced during the port was not alarming. The mode of operation is a bit cumbersome compared to other products (command interface and library structure). The real time extensions need some consideration as it was shown. Specifically a need for direct access to simple synchronization primitives like semaphores is required. This should not be a complex extension to implement.

The WCETE is very immature, it is difficult to say what the potential is. It will probably be difficult to find practical use for it.

The Schedulability Analyser is a very useful tool. It suffers a bit from the requirement to be compatible with the WCETE, which makes production of the input less obvious. A some alarming errors must be removed before it can be used in a project. For the use in the ERA ASW development the method and tool already used for WCET estimation and schedulability analysis is considered superior to the use of WCETE and/or Schedulability Analyser.

The Scheduler Simulator is found useful only for educational purposes. It is thus a less important tool in the suite. It seems to be erroneous and must be corrected before use.

The ERC32 target simulator is a very powerful simulator with great potential (unfortunately partly limited by AdaProbe). However, a performance improvement is a must before it can be used for real.

Appendix A

Protected Object Test

```
package Protected_Example is

   type Ids_T is array (1..10) of Integer;
   Ids: Ids_T := (others =>0);
   Id_Count: Integer := 0;
   procedure Signal
     ( Id : in Integer );
   procedure Wait
     ( Id : in Integer );
   procedure Register
     ( Id : in Integer );

end Protected_Example;

package body Protected_Example is

   Signalled : Boolean := False;

   procedure Register
     ( Id : in Integer ) is
   begin
      Id_Count := Id_Count+1;
      Ids(Id_Count) := Id;
   end Register;

   task Protected_Object is

      pragma Passive;

      pragma Priority(4);

      entry Signal
        ( Id : in Integer );

      entry Wait
        ( Id : in Integer );

   end Protected_Object;

   task body Protected_Object is

   begin
      loop
         select
            accept Signal
              ( Id : in Integer ) do
               Signalled := True;
               Register(Id);
            end Signal;

         or
            when Signalled =>
            accept Wait ( Id : in Integer ) do
               Register(Id);
            end Wait;
         or
            terminate ;
         end select ;
      end loop ;
   end Protected_Object;

   procedure Signal
     ( Id : in Integer ) is
   begin
      Protected_Object.Signal(Id);
   end ;

   procedure Wait
     ( Id : in Integer ) is
```

```
   begin
      Protected_Object.Wait(Id);
   end ;

end Protected_Example;
with Text_IO;
use Text_IO;
with Protected_Example;

procedure Protected_Test is

   package Int_IO is new Text_IO.Integer_IO(Integer);
   use Int_IO;

   task Low is
      pragma Priority(1);
      entry Quit;
   end Low;

   task body Low is

   begin
      Protected_Example.Wait(1);
      Protected_Example.Register(1);
      accept Quit;
   end ;

   task High is
      pragma Priority(2);
      entry Quit;
   end High;

   task body High is

   begin
      delay 0.5;
      Protected_Example.Signal(2);
      Protected_Example.Register(2);
      Protected_Example.Signal(2);
      accept Quit;
   end High;

begin
   Low.Quit;
   High.Quit;

   Put("Registration order: ");
   Put(Protected_Example.Ids(1));
   Put(", ");
   Put(Protected_Example.Ids(2));
   Put(", ");
   Put(Protected_Example.Ids(3));
   Put(", ");
   Put(Protected_Example.Ids(4));
   Put(", ");
   Put(Protected_Example.Ids(5));
end ;
```

Appendix B

Schedulability Analysis Results

B.1 Input to CRI Schedule Program (null RTS)

```
Task : c_subsystem_bus_scheduler
           period   :  50.0
           WCET     :   7.7
           Blocking :   1.5
           deadline :  41.00

Task : c_issa_bus_scheduler
           period   : 100.0
           WCET     :   2.8
           Blocking :   1.5
           deadline :  42.0

Task : c_motion_execution
           period   :  50.0
           WCET     :   3.1 -- T_Prox = 1.257*1.164*T_proto + 1.2
           Blocking :   1.5
           deadline :  32.0
   mode     : proximity

Task : c_motion_execution
           period   :  50.0
           WCET     :   4.1 -- T_Compl = 1.257*1.164*T_proto + 1.2 + 1.0
           Blocking :   1.5
           deadline :  32.0
   mode     : compliant

Task : c_motion_execution
           period   :  50.0
           WCET     :   1.9 -- T_Free = 1.164*T_proto + 1.2
           Blocking :   1.5
           deadline :  32.0
   mode     : free

Task : c_prepare_vis_tac
           period   : 100.0
           WCET     :  15.0
           Blocking :   1.5
           Deadline :  32.0
   Mode     : proximity

Task : s_event_handler
           period   :1000.0
           WCET     :   2.0
           Blocking :   1.0
           deadline :  50.0

Task : s_tc_handler
           period   : 200.0
           WCET     :   8.9
           Blocking :   1.0
           deadline :  50.0

Task : c_vhf_check
           period   :  50.0
           WCET     :   2.5
           Blocking :   1.0
           deadline :  50.0

Task : c_usd_ecc_collection
           period   :  50.0
           WCET     :   1.5
           Blocking :   1.0
           deadline :  50.0

Task : c_mf_check
           period   : 200.0
```

```
           WCET     :   2.0
           Blocking :   1.0
           deadline : 200.0

Task : c_telemetry_generation
           period   : 500.0
           WCET     :  21.0
           Blocking :   1.0
           deadline : 500.0

Task : c_2hz_check
           period   : 500.0
           WCET     :   7.5 -- 4.5 + 2 * 1.164 * T_proto
           Blocking :   1.0
           deadline : 500.0
   mode     : free

Task : c_2hz_check
           period   : 500.0
           WCET     :   4.5 -- 4.5 (Wouters measurement)
           Blocking :   1.0
           deadline : 500.0
   mode     : proximity

Task : c_2hz_check
           period   : 500.0
           WCET     :   4.5 -- 4.5 (Wouters measurement)
           Blocking :   1.0
           deadline : 500.0
   mode     : compliant

Task : s_action_ack_handler
           period   :1000.0
           WCET     :   2.0
           Blocking :   1.0
           deadline :1000.0

Task : c_lf_checks
           period   :1000.0
           WCET     :  12.0
           Blocking :   0.0
           deadline :1000.0
```

B.2 Output from CRI Schedule Program (null RTS)

```
Schedule - Deadline Monotonic Analysis, (C) CRI A/S 1995.

Object name              : c_subsystem_bus_scheduler
Period                   :   50.0 ms.
Worst-case execution time :    7.7 ms.
Blocking delays          :    1.5 ms.
Deadline                 :   41.0 ms.

Max response time        :    9.2 ms.


Object name              : c_issa_bus_scheduler
Period                   :  100.0 ms.
Worst-case execution time :    2.8 ms.
Blocking delays          :    1.5 ms.
Deadline                 :   42.0 ms.

Max response time        :   12.0 ms.
```

```
Object name               : c_motion_execution
Period                    :   50.0 ms.
Worst-case execution time :    3.1 ms.
Blocking delays           :    1.5 ms.
Deadline                  :   32.0 ms.

Max response time         :   15.1 ms.


Object name               : c_prepare_vis_tac
Period                    :  100.0 ms.
Worst-case execution time :   15.0 ms.
Blocking delays           :    1.5 ms.
Deadline                  :   32.0 ms.

Max response time         :   30.1 ms.


Object name               : s_event_handler
Period                    : 1000.0 ms.
Worst-case execution time :    2.0 ms.
Blocking delays           :    1.0 ms.
Deadline                  :   50.0 ms.

Max response time         :   31.6 ms.


Object name               : s_tc_handler
Period                    :  200.0 ms.
Worst-case execution time :    8.9 ms.
Blocking delays           :    1.0 ms.
Deadline                  :   50.0 ms.

Max response time         :   40.5 ms.


Object name               : c_vhf_check
Period                    :   50.0 ms.
Worst-case execution time :    2.5 ms.
Blocking delays           :    1.0 ms.
Deadline                  :   50.0 ms.

Max response time         :   43.0 ms.


Object name               : c_usd_ecc_collection
Period                    :   50.0 ms.
Worst-case execution time :    1.5 ms.
Blocking delays           :    1.0 ms.
Deadline                  :   50.0 ms.

Max response time         :   44.5 ms.


Object name               : c_mf_check
Period                    :  200.0 ms.
Worst-case execution time :    2.0 ms.
Blocking delays           :    1.0 ms.
Deadline                  :  200.0 ms.

Max response time         :   46.5 ms.


Object name               : c_telemetry_generation
Period                    :  500.0 ms.
Worst-case execution time :   21.0 ms.
Blocking delays           :    1.0 ms.
Deadline                  :  500.0 ms.
```

```
Max response time         :   82.3 ms.


Object name               : c_2hz_check
Period                    :  500.0 ms.
Worst-case execution time :    4.5 ms.
Blocking delays           :    1.0 ms.
Deadline                  :  500.0 ms.

Max response time         :   86.8 ms.


Object name               : s_action_ack_handler
Period                    : 1000.0 ms.
Worst-case execution time :    2.0 ms.
Blocking delays           :    1.0 ms.
Deadline                  : 1000.0 ms.

Max response time         :   88.8 ms.


Object name               : c_lf_checks
Period                    : 1000.0 ms.
Worst-case execution time :   12.0 ms.
Blocking delays           :    0.0 ms.
Deadline                  : 1000.0 ms.

Max response time         :   99.8 ms.



Total CPU load            :   59.5 %
The system is schedulable.
```

B.3 Input to SpaceBel Schedulability Analyzer (null RTS)

User Configuration File:

```
-- This file defines the characteristics of the ASW realtime constraints


PREFERENCES                              -- THE SCHEDULING PREFERENCES
DMS -- DEADLINES MUST BE BEFORE NEXT RELEASE
BLOCKING PROTOCOL IPCI -- IMMEDIATE CEILING PRIORITY INHERITENCE
END

TARGET CHARACTERISTICS
NO ATAC -- THE ATAC IS NOT PRESENT
PRIORITY LOW  1       -- NON-INTERRUPT PRIORITY ASSIGNED LEAST IMPORTANT FIRST
PRIORITY HIGH 128 -- INTERRUPT PRIORITIES START HERE AND DOWNWARDS
CPU CLOCK FREQUENCY 10 MHz -- CLOCK FREQUENCY CHOSEN AT 0.1 Microsecond
WAIT STATES READ 0 WRITE 1 -- NUMBER OF READ WRITE STATES
END


THREAD DEFINITION

  THREAD c_subsystem_bus_scheduler -- Cyclic THREAD
   CRITICALITY HARD
   PERIOD  500000
```

```
 DEADLINE 250000
END

THREAD c_issa_bus_scheduler
 CRITICALITY HARD
 PERIOD 1000000
 DEADLINE  300000
END

THREAD c_motion_execution
 CRITICALITY HARD
 MINIMUM   500000
 DEADLINE  320000
END

THREAD c_prepare_vis_tac
 CRITICALITY HARD
 PERIOD 1000000
 DEADLINE  320000
END

THREAD s_event_handler
 CRITICALITY HARD
 MINIMUM 10000000
 DEADLINE  500000
END

THREAD s_tc_handler
 CRITICALITY HARD
 MINIMUM 2000000
 DEADLINE  500000
END

THREAD c_vhf_check
 CRITICALITY HARD
 PERIOD  500000
 DEADLINE  500000
END

THREAD c_usd_ecc_collection
 CRITICALITY HARD
 PERIOD  500000
 DEADLINE  500000
END

THREAD c_mf_check
 CRITICALITY HARD
 PERIOD 2000000
 DEADLINE 2000000
END

THREAD c_2hz_check
 CRITICALITY HARD
 PERIOD 5000000
 DEADLINE 5000000
END

THREAD c_telemetry_generation
 CRITICALITY HARD
 MINIMUM  5000000
 DEADLINE 4000000
END

THREAD s_action_ack_handler
 CRITICALITY HARD
 MINIMUM 10000000
 DEADLINE 10000000
END
```

```
  THREAD c_lf_checks
   CRITICALITY SOFT
   PERIOD 10000000
   DEADLINE 10000000
  END

END

WCET DATA

END
```

Execution Skeleton File:

```
-- This file defines the execution characteristics of the ASW.

PROGRAM ASW

  THREAD c_subsystem_bus_scheduler -- Cyclic THREAD
   TYPE CYCLIC
   WCET 40000,0,0
   CALL_PO pr_event_buffer raise_event
   CALL_PO pr_synch2 release_motion
   CALL_PO pr_synch release_tm
   CALL_PO some_object long_entry
  END

  THREAD c_issa_bus_scheduler
   TYPE CYCLIC
   WCET 23000,0,0
   CALL_PO pr_tc_handler signal_new_issa_tc
  END

  THREAD c_motion_execution
   TYPE SPORADIC
   WCET 27000,0,0
   CALL_PO pr_synch2 wait_motion
   CALL_PO pr_acknowledge send_action_response
  END

  THREAD c_prepare_vis_tac
   TYPE CYCLIC
   WCET 150000,0,0
  END

  THREAD s_event_handler
   TYPE SPORADIC
   WCET 10000,0,0
   CALL_PO pr_event_buffer get_event
  END

  THREAD s_tc_handler
   TYPE SPORADIC
   WCET 79000,0,0
   CALL_PO pr_tc_handler get_tc
  END

  THREAD c_vhf_check
   TYPE CYCLIC
   WCET 15000,0,0
   CALL_PO some_object some_entry
  END

  THREAD c_usd_ecc_collection
   TYPE CYCLIC
   WCET 5000,0,0
```

```
 CALL_PO some_object some_entry
END

THREAD c_mf_check
 TYPE CYCLIC
 WCET 10000,0,0
 CALL_PO some_object some_entry
END

THREAD c_telemetry_generation
 TYPE SPORADIC
 WCET 200000,0,0
 CALL_PO pr_synch wait_for_tm
END

THREAD c_2hz_check
 TYPE CYCLIC
 WCET 40000,0,0
 CALL_PO pr_event_buffer raise_event
END

THREAD s_action_ack_handler
 TYPE SPORADIC
 WCET 17000,0,0
 CALL_PO pr_acknowledge get_action_response
END

THREAD c_lf_checks
 TYPE CYCLIC
 WCET 110000,0,0
 CALL_PO some_object some_entry
END

PROTECTED pr_synch
 TYPE SYNCHRO
 ENTRY release_tm
   WCET 1000,0,0
 BARRIER WCET 1000,0,0
 ENTRY wait_for_tm
   WCET 8000,0,0
END

PROTECTED pr_synch2
 TYPE SYNCHRO
 ENTRY release_motion
   WCET 1000,0,0
 BARRIER WCET 1000,0,0
 ENTRY wait_motion
   WCET 1000,0,0
END

PROTECTED pr_event_buffer
 TYPE SYNCHRO
 ENTRY raise_event
   WCET 5000,0,0
 BARRIER WCET 0,0,0
 ENTRY get_event
   WCET 10000,0,0
END

PROTECTED pr_acknowledge
 TYPE SYNCHRO
 ENTRY send_action_response
   WCET 1000,0,0
 BARRIER WCET 1000,0,0
 ENTRY get_action_response
   WCET 1000,0,0
END
```

```
  PROTECTED pr_tc_handler
   TYPE SYNCHRO
   ENTRY signal_new_issa_tc
     WCET 5000,0,0
   BARRIER WCET 1000,0,0
   ENTRY get_tc
     WCET 8000,0,0
  END

  PROTECTED some_object
   TYPE RESOURCE
   ENTRY some_entry
     WCET 10000,0,0
   ENTRY long_entry
     WCET 30000,0,0
  END

END
```

Null runtime characteristics file:

```
CONTEXT   SWITCH    TIME                           0,0,0
SCHEDULER SELECT    TIME                         (0,0,0)
TIMER     INTERRUPT OVERHEAD                       0,0,0
READY     AFTER     DELAY                          0,0,0
ENTER     DELAY     UNTIL    OVERHEAD     AT HEAD (0,0,0)
ENTER     DELAY     UNTIL    OVERHEAD NOT AT HEAD (0,0,0)
EXIT      DELAY     UNTIL    OVERHEAD               0,0,0
ENTER     PASSIVE   TASK                           0,0,0
EXIT      PASSIVE   TASK                           0,0,0
ENTER     SOFTWARE  SPORADIC WAIT                 (0,0,0)
EXIT      SOFTWARE  SPORADIC WAIT                  0,0,0
ENTRY     QUEUE     SERVICING                     (0,0,0)
INTERRUPT HANDLING  OVERHEAD                       0,0,0
ENTER     INTERRUPT SPORADIC WAIT                 (0,0,0)
EXIT      INTERRUPT SPORADIC WAIT                  0,0,0

MAXIMUM NON PREEMPTION                             0,0,0
```

B.4 Output from SpaceBel Schedulability Analyzer (null RTS)

```
Schedulability Analyser
***********************

Beginning of the Offline Analyser Report

Current date and time : 19/6/1997::15:36:42

Name of the input files :
UCF Files : /home/cma/ERC32/analyzer/asw.ucf
ESF Files : /home/cma/ERC32/analyzer/asw.esf
RTS Files : /home/cma/ERC32/analyzer/NULL.rts

Scheduling Policy : DMS   Blocking Protocol : IPCI   ATAC Mode : Not selected

List of defined threads
_______________________

In the following report, all threads are defined by a numeric identifier :

   Identifier   Thread
   1............C_SUBSYSTEM_BUS_SCHEDULER
   2............C_ISSA_BUS_SCHEDULER
   3............C_MOTION_EXECUTION
   4............C_PREPARE_VIS_TAC
   5............S_EVENT_HANDLER
```

```
   6...........S_TC_HANDLER
   7...........C_VHF_CHECK
   8...........C_USD_ECC_COLLECTION
   9...........C_MF_CHECK
   10..........C_2HZ_CHECK
   11..........C_TELEMETRY_GENERATION
   12..........S_ACTION_ACK_HANDLER
   13..........C_LF_CHECKS

List of defined Protected Objects
_________________________________

In the following report, all Protected Objects are defined by a numeric identifier :

   Identifier   Protected Object
   1...........PR_EVENT_BUFFER
   2...........PR_SYNCH2
   3...........PR_SYNCH
   4...........SOME_OBJECT
   5...........PR_TC_HANDLER
   6...........PR_ACKNOWLEDGE

List of all schedulable and non schedulable threads :
*****************************************************
```

Sch	Th#	Criticality	Deadline	Prio	WCCT FR	WCCT SR	Period MIAT	Response Time	Blocking Time	Blocking Origin	Utilisation Factor	Margin Analysis
yes	1	HARD	25000	15	7700	N/A	50000	9200	1500	PO # 1	1.54000E-01	2.47E+01
yes	2	HARD	30000	13	2800	N/A	100000	12000	1500	PO # 1	1.82000E-01	6.79E+01
yes	3	HARD	32000	11	3100	N/A	50000	15100	1500	PO # 1	2.44000E-01	6.13E+01
yes	4	HARD	32000	10	15000	N/A	100000	30100	1500	PO # 1	3.94000E-01	1.27E+01
yes	5	HARD	50000	9	2000	N/A	1000000	31600	1000	PO # 3	3.96000E-01	2.75E+02
yes	6	HARD	50000	8	8900	N/A	200000	40500	1000	PO # 3	4.40500E-01	6.18E+01
yes	7	HARD	50000	7	2500	N/A	50000	43000	1000	PO # 3	4.90500E-01	2.20E+02
yes	8	HARD	50000	6	1500	N/A	50000	44500	1000	PO # 3	5.20500E-01	3.67E+02
yes	9	HARD	200000	5	2000	N/A	200000	46500	1000	PO # 3	5.30500E-01	4.00E+03
yes	11	HARD	400000	4	21000	N/A	500000	82300	1000	PO # 4	5.72500E-01	7.84E+02
yes	10	HARD	500000	3	4500	N/A	500000	86800	1000	PO # 4	5.81500E-01	4.48E+03
yes	12	HARD	1000000	2	2000	N/A	1000000	88800	1000	PO # 4	5.83500E-01	2.02E+04
yes	13	SOFT	1000000	1	12000	N/A	1000000	99800	0	RTS	5.95500E-01	3.37E+03

```
List of all protected objects
*****************************
```

PO #	Prio	Protected Type	PO User List
1	19	SYNCHRONISATION	T#1, T#5, T#10,
2	18	SYNCHRONISATION	T#1, T#3,
3	17	SYNCHRONISATION	T#1, T#11,
4	16	RESOURCE	T#1, T#7, T#8, T#9, T#13,
5	14	SYNCHRONISATION	T#2, T#6,
6	12	SYNCHRONISATION	T#3, T#12,

```
Thread Set Utilisation Factor
*****************************

The Utilisation factor for the whole thread set is :  5.95500E-01
```

B.5 Input to CRI Schedule Program (AdaWorld RTS)

```
Task : c_subsystem_bus_scheduler
           period   :  50.0
           WCET     :   7.956
           Blocking :   1.562
           deadline :  41.00

Task : c_issa_bus_scheduler
           period   : 100.0
           WCET     :   2.941
           Blocking :   1.562
           deadline :  42.0

Task : c_motion_execution
           period   :  50.0
           WCET     :   3.216 -- T_Prox = 1.257*1.164*T_proto + 1.2
           Blocking :   1.562
```

```
           deadline :  32.0
   mode     : proximity

Task : c_motion_execution
           period   :  50.0
           WCET     :   4.1 -- T_Compl = 1.257*1.164*T_proto + 1.2 + 1.0
           Blocking :   0.3
           deadline :  32.0
   mode     : compliant

Task : c_motion_execution
           period   :  50.0
           WCET     :   1.9 -- T_Free = 1.164*T_proto + 1.2
           Blocking :   0.3
           deadline :  32.0
   mode     : free

Task : c_prepare_vis_tac
           period   : 100.0
           WCET     :  15.093
           Blocking :   1.562
           Deadline :  32.0
   Mode     : proximity

Task : s_event_handler
           period   :1000.0
           WCET     :   2.068
           Blocking :   1.068
           deadline :  50.0

Task : s_tc_handler
           period   : 200.0
           WCET     :   8.968
           Blocking :   1.068
           deadline :  50.0

Task : c_vhf_check
           period   :  50.0
           WCET     :   2.612
           Blocking :   1.068
           deadline :  50.0

Task : c_usd_ecc_collection
           period   :  50.0
           WCET     :   1.612
           Blocking :   1.068
           deadline :  50.0

Task : c_mf_check
           period   : 200.0
           WCET     :   2.112
           Blocking :   1.068
           deadline : 200.0

Task : c_telemetry_generation
           period   : 500.0
           WCET     :  21.068
           Blocking :   1.019
           deadline : 500.0

Task : c_2hz_check
           period   : 500.0
           WCET     :   7.5 -- 4.5 + 2 * 1.164 * T_proto
           Blocking :   1.0
           deadline : 500.0
   mode     : free

Task : c_2hz_check
           period   : 500.0
```

```
           WCET     :   4.641 -- 4.5 (Wouters measurement)
           Blocking :   1.019
           deadline : 500.0
   mode     : proximity

Task : c_2hz_check
           period   : 500.0
           WCET     :   4.5 -- 4.5 (Wouters measurement)
           Blocking :   1.0
           deadline : 500.0
   mode     : compliant

Task : s_action_ack_handler
           period   :1000.0
           WCET     :   2.068
           Blocking :   1.019
           deadline :1000.0

Task : c_lf_checks
           period   :1000.0
           WCET     :  12.112
           Blocking :   0.13
           deadline :1000.0
```

B.6 Output from CRI Schedule Program (AdaWorld RTS)

```
Schedule - Deadline Monotonic Analysis, (C) CRI A/S 1995.

Object name               : c_subsystem_bus_scheduler
Period                    :   50.0 ms.
Worst-case execution time :    8.0 ms.
Blocking delays           :    1.6 ms.
Deadline                  :   41.0 ms.

Max response time         :    9.5 ms.


Object name               : c_issa_bus_scheduler
Period                    :  100.0 ms.
Worst-case execution time :    2.9 ms.
Blocking delays           :    1.6 ms.
Deadline                  :   42.0 ms.

Max response time         :   12.5 ms.


Object name               : c_motion_execution
Period                    :   50.0 ms.
Worst-case execution time :    3.2 ms.
Blocking delays           :    1.6 ms.
Deadline                  :   32.0 ms.

Max response time         :   15.7 ms.


Object name               : c_prepare_vis_tac
Period                    :  100.0 ms.
Worst-case execution time :   15.1 ms.
Blocking delays           :    1.6 ms.
Deadline                  :   32.0 ms.

Max response time         :   30.8 ms.
```

```
Object name               : s_event_handler
Period                    : 1000.0 ms.
Worst-case execution time :    2.1 ms.
Blocking delays           :    1.1 ms.
Deadline                  :   50.0 ms.

Max response time         :   32.3 ms.


Object name               : s_tc_handler
Period                    :  200.0 ms.
Worst-case execution time :    9.0 ms.
Blocking delays           :    1.1 ms.
Deadline                  :   50.0 ms.

Max response time         :   41.3 ms.


Object name               : c_vhf_check
Period                    :   50.0 ms.
Worst-case execution time :    2.6 ms.
Blocking delays           :    1.1 ms.
Deadline                  :   50.0 ms.

Max response time         :   43.9 ms.


Object name               : c_usd_ecc_collection
Period                    :   50.0 ms.
Worst-case execution time :    1.6 ms.
Blocking delays           :    1.1 ms.
Deadline                  :   50.0 ms.

Max response time         :   45.5 ms.


Object name               : c_mf_check
Period                    :  200.0 ms.
Worst-case execution time :    2.1 ms.
Blocking delays           :    1.1 ms.
Deadline                  :  200.0 ms.

Max response time         :   47.6 ms.


Object name               : c_telemetry_generation
Period                    :  500.0 ms.
Worst-case execution time :   21.1 ms.
Blocking delays           :    1.0 ms.
Deadline                  :  500.0 ms.

Max response time         :   84.1 ms.


Object name               : c_2hz_check
Period                    :  500.0 ms.
Worst-case execution time :    4.6 ms.
Blocking delays           :    1.0 ms.
Deadline                  :  500.0 ms.

Max response time         :   88.7 ms.


Object name               : s_action_ack_handler
Period                    : 1000.0 ms.
Worst-case execution time :    2.1 ms.
Blocking delays           :    1.0 ms.
Deadline                  : 1000.0 ms.
```

```
Max response time         :   90.8 ms.


Object name               : c_lf_checks
Period                    : 1000.0 ms.
Worst-case execution time :   12.1 ms.
Blocking delays           :    0.1 ms.
Deadline                  : 1000.0 ms.

Max response time         :  135.4 ms.



Total CPU load            :   61.1 %
The system is schedulable.
```

B.7 Input to SpaceBel Schedulability Analyzer (AdaWorld RTS)

AdaWorld runtime characteristics file:

```
-- RTS Characteristics for DEM32 at 10 MHz (non-ATAC runtime)
-- Characteristics are in Processor Cycles

ENTER PASSIVE TASK 80,0,0
EXIT PASSIVE TASK 110,0,0
ENTER SOFTWARE SPORADIC WAIT (70,0,0)
EXIT SOFTWARE SPORADIC WAIT 30,0,0
ENTRY QUEUE SERVICING (60,0,0)
CONTEXT SWITCH TIME 340,0,0
SCHEDULER SELECT TIME (50,0,0)
ENTER DELAY UNTIL OVERHEAD AT HEAD (390,0,0)
ENTER DELAY UNTIL OVERHEAD NOT AT HEAD (220,0,0) + (30,0,0) * C
EXIT DELAY UNTIL OVERHEAD 80,0,0
TIMER INTERRUPT OVERHEAD 210,0,0
READY AFTER DELAY 120,0,0
INTERRUPT HANDLING OVERHEAD 670,0,0
ENTER INTERRUPT SPORADIC WAIT (30,0,0)
EXIT INTERRUPT SPORADIC WAIT 30,0,0
MAXIMUM NON PREEMPTION 1300,0,0
```

B.8 Output from SpaceBel Schedulability Analyzer (AdaWorld RTS)

```
Schedulability Analyser
***********************

Beginning of the Offline Analyser Report

Current date and time : 19/6/1997::15:37:12

Name of the input files :
UCF Files : /home/cma/ERC32/analyzer/asw.ucf
ESF Files : /home/cma/ERC32/analyzer/asw.esf
RTS Files : /home/cma/ERC32/analyzer/adaworld.rts

Scheduling Policy : DMS   Blocking Protocol : IPCI   ATAC Mode : Not selected

List of defined threads
_______________________
```

In the following report, all threads are defined by a numeric identifier :

```
   Identifier   Thread
   1...........C_SUBSYSTEM_BUS_SCHEDULER
   2...........C_ISSA_BUS_SCHEDULER
   3...........C_MOTION_EXECUTION
   4...........C_PREPARE_VIS_TAC
   5...........S_EVENT_HANDLER
   6...........S_TC_HANDLER
   7...........C_VHF_CHECK
   8...........C_USD_ECC_COLLECTION
   9...........C_MF_CHECK
   10..........C_2HZ_CHECK
   11..........C_TELEMETRY_GENERATION
   12..........S_ACTION_ACK_HANDLER
   13..........C_LF_CHECKS
```

List of defined Protected Objects

In the following report, all Protected Objects are defined by a numeric identifier :

```
   Identifier   Protected Object
   1...........PR_EVENT_BUFFER
   2...........PR_SYNCH2
   3...........PR_SYNCH
   4...........SOME_OBJECT
   5...........PR_TC_HANDLER
   6...........PR_ACKNOWLEDGE
```

List of all schedulable and non schedulable threads :

Sch	Th#	Criticality	Deadline	Prio	WCCT FR	WCCT SR	Period MIAT	Response Time	Blocking Time	Blocking Origin	Utilisation Factor	Margin Analysis
yes	1	HARD	25000	15	7956	N/A	50000	9821	1562	PO # 1	1.62024E-01	1.07E+01
yes	2	HARD	30000	13	2941	N/A	100000	12801	1562	PO # 1	1.91434E-01	2.89E+01
yes	3	HARD	32000	11	3216	N/A	50000	16089	1562	PO # 1	2.55754E-01	2.65E+01
yes	4	HARD	32000	10	15093	N/A	100000	31149	1562	PO # 1	4.06684E-01	5.64E+00
yes	5	HARD	50000	9	2068	N/A	1000000	32795	1068	PO # 3	4.08752E-01	1.94E+02
yes	6	HARD	50000	8	8968	N/A	200000	41763	1068	PO # 3	4.53592E-01	4.47E+01
yes	7	HARD	50000	7	2612	N/A	50000	44342	1068	PO # 3	5.05832E-01	1.53E+02
yes	8	HARD	50000	6	1612	N/A	50000	45993	1068	PO # 3	5.38072E-01	2.49E+02
yes	9	HARD	200000	5	2112	N/A	200000	48144	1068	PO # 3	5.48632E-01	3.58E+03
yes	11	HARD	400000	4	21068	N/A	500000	84847	1019	PO # 4	5.90768E-01	7.39E+02
yes	10	HARD	500000	3	4641	N/A	500000	89455	1019	PO # 4	6.00050E-01	4.11E+03
yes	12	HARD	1000000	2	2068	N/A	1000000	91595	1019	PO # 4	6.02118E-01	1.85E+04
yes	13	SOFT	1000000	1	12112	N/A	1000000	136575	130	RTS	6.14230E-01	3.16E+03

List of all protected objects

PO #	Prio	Protected Type	PO User List
1	19	SYNCHRONISATION	T#1, T#5, T#10,
2	18	SYNCHRONISATION	T#1, T#3,
3	17	SYNCHRONISATION	T#1, T#11,
4	16	RESOURCE	T#1, T#7, T#8, T#9, T#13,
5	14	SYNCHRONISATION	T#2, T#6,
6	12	SYNCHRONISATION	T#3, T#12,

Thread Set Utilisation Factor

The Utilisation factor for the whole thread set is : 6.14230E-01

Appendix C

Scheduler Simulator Results

TIME	EVENT	THREAD	PO	PRIORITY
50000	EXPIRED_DEADLINE	C_USD_ECC_COLLECTION		6
50000	EXPIRED_DEADLINE	C_VHF_CHECK		7
60500	EXPIRED_DEADLINE	S_TC_HANDLER		8

THREAD	TIME				ACHIEVED DEADLINE		MISSED DEADLINE	
	CPU		RTS		min	max	min	max
	SUM	%	SUM	%				
C_SUBSYSTEM_BUS_SCHE	15400	22.19%	0	0.00%	16100	17300	----	----
C_ISSA_BUS_SCHEDULER	2800	4.03%	0	0.00%	19500	19500	----	----
C_MOTION_EXECUTION	8900	12.82%	0	0.00%	15700	25800	----	----
C_PREPARE_VIS_TAC	15000	21.61%	0	0.00%	700	700	----	----
S_EVENT_HANDLER	5000	7.20%	0	0.00%	15700	41700	----	----
S_TC_HANDLER	16800	24.21%	0	0.00%	----	----	10500	10500
C_VHF_CHECK	5000	7.20%	0	0.00%	----	----	----	----
C_USD_ECC_COLLECTION	500	0.72%	0	0.00%	----	----	----	----
C_MF_CHECK	0	0.00%	0	0.00%	----	----	----	----
C_2HZ_CHECK	0	0.00%	0	0.00%	----	----	----	----
C_TELEMETRY_GENERATI	0	0.00%	0	0.00%	----	----	----	----
S_ACTION_ACK_HANDLER	0	0.00%	0	0.00%	----	----	----	----
C_LF_CHECKS	0	0.00%	0	0.00%	----	----	----	----

```
Simulation_Time :  69400
Processing_Time :  69400
Overall CPU     : 100.00
Overall RTS     :   0.00
```

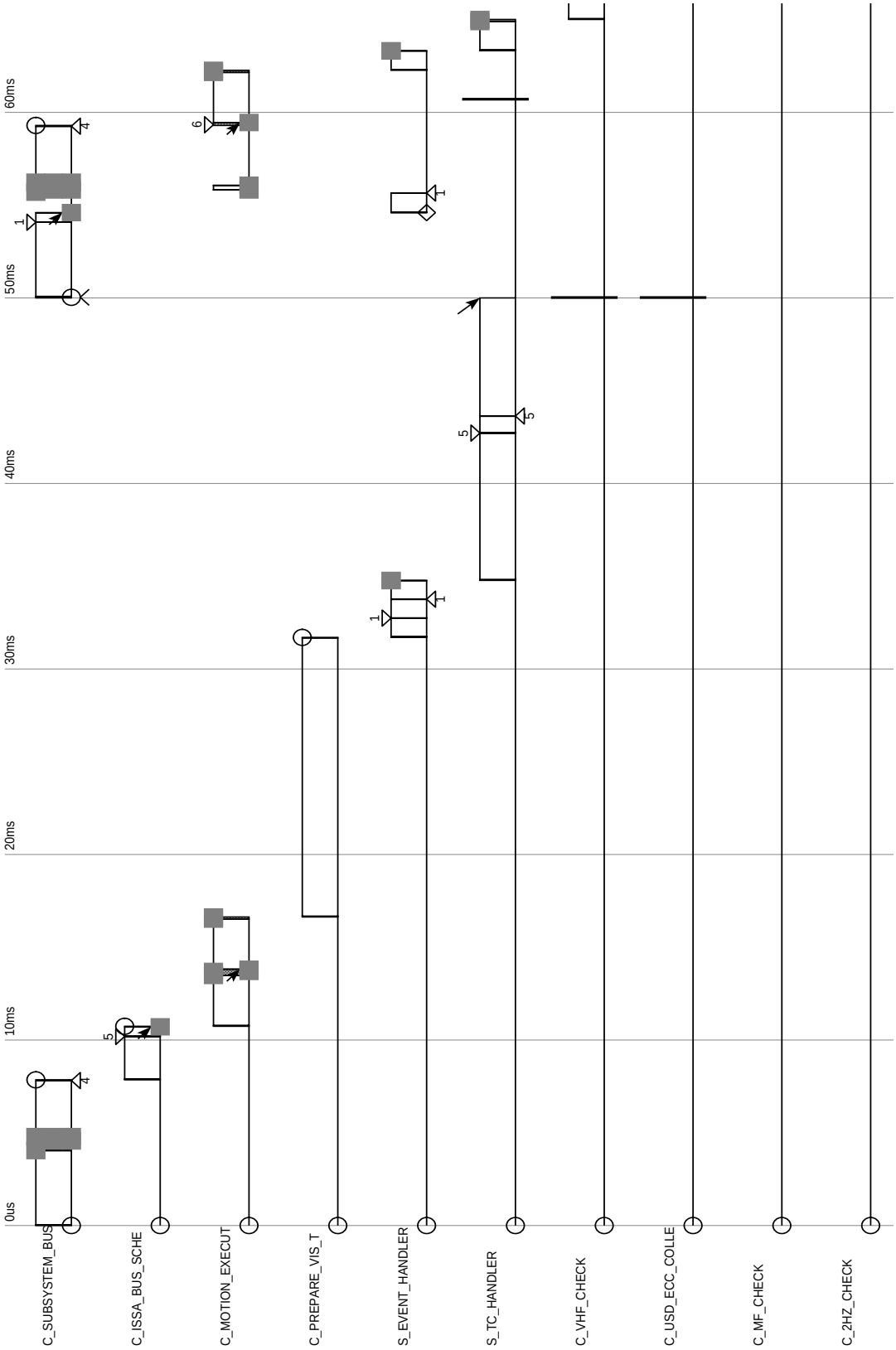

Figure 1

Gantt chart