EDISOFT
DEFENCE & AEROSPACE TECHNOLOGIES

embedded brains

CISTER
Research Centre in
Real-Time & Embedded
Computing Systems

jenaoptronik

Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

Lero THE IRISH SOFTWARE RESEARCH CENTRE

# RTEMS SMP
## Ready to Fly

# FV3-202 Formal Verification Report
## Release 1

# CONTENTS



 

## Identification, Copyrights and License



| Release | Git hash | Date | Approved | Changes |
|---|---|---|---|---|
| 01 | ed3cda081 | Nov. 29th 2021 | Approved | For FR/AR |

| | Name | Company | Signature |
|---|---|---|---|
| Written by | Andrew Butterfield | Trinity College Dublin | Andrew Butterfield |
| Verified by | Andrew Butterfield | Trinity College Dublin | Andrew Butterfield |
| Approved by | Nuno Ramos | EDISOFT | RAMOS Nuno-Vicente Digitally signed by RAMOS Nuno-Vicente Date: 2021.12.16 11:31:06 Z |

| | Name | Signature (if necessary) |
|---|---|---|
| Approved by customer (if necessary) | | |

 

CHAPTER

# ONE

# INTRODUCTION

The aim of the RTEMS SMP project is to pre-qualify RTEMS of the community (rtems.org) and to decrease the manual work required for future pre-qualifications. For this purpose the project is split into two main tasks:

- RTEMS pre-qualification
- Qualification Toolchain

The RTEMS pre-qualification involves the creation of the set of deliverables required by the relevant ECSS standards. RTEMS is an open-source RTOS created in the 1980s by the USA Defense Department that has since been made public. Today it is maintained by OAR. Hence, modifications to it (along with the new deliverables produced in this project) require authorization by OAR in order to be merged into the RTEMS mainline.

The aim of the Qualification toolchain is to generate outputs automatically (e.g. documentation) derived from a set of inputs. Some of these inputs are derived from the RTEMS ticket system while others are derived from the RTEMS repository.

In relation to the testing of RTEMS, there are two connected PCs running at ESTEC, one capable of building, loading and running executables on the GR712 and GR740 and another PC capable of building and running test executables on a SCOC3 machine with 1553 and UART connections to the boards. Access to some of this has been granted to consortium members.

In addition to the main two tasks above, there are two other tasks:

- Formal Verification
- Test Implementation

Test Implementation involves taking a real application running on a multiprocessing version of RTEMS, and performing pre-qualification on that, to assess the effectiveness of the proposed solutions.

This deliverable is a final report on Task 3 regarding Formal Verification.

We give an overall summary of the sub-tasks in 2. We then follow in 4 with a discussion of the three case-studeies done. Finally, we conclude and make future reccommendations in 5.

 

CHAPTER

TWO

# TASK 3 OVERVIEW

We present a quick overview of the three sub-tasks in Task 3.

## 2.1 Task 3.1

Task 3.1 was an initial survey task to identify the algorithms and tools to be used and how the results would be integrated into the qualification environment.

It was scheduled to run for 6 months from KO meeting until the PDR.

A key constraint was finding tools and verification approaches that would fit the RTEMS community principles. Key issues for RTEMS included maintenance, and the fact that some of their build/CI systems had limited memory/storage. This means that new tools with large memory footprints could be problematic.

The deliverable document [BH21] discusses these issues under the headings of Methods, Algorithms, Integration, and Planning.

The main outcome of this sub-task was to identify two possible approaches to follow:

1. Use formal methods behind the scenes to generate tests.
2. Use formal methods to analyse code, which would require some code annotations.

Option 1 was recommended, using Promela/SPIN as the formal technique.

A shortlist of algorithms/features was drawn up, including scheduling algorithms, context switching, synchronisation primitives, key data-structures, and selected RTEMS API specifications.

For various reasons, there was no definitive decision made about the algorithms to be verified at the end of this task. It was felt that more exploration of the issues was needed in the early part of Task 3.2

## 2.2 Task 3.2

The key technical work of Task 3 was done by Task 3.2. The goal was to develop the formal models, noting assumptions made, and providing traceability to requirements and code.

It was scheduled to run for 17 months from the PDR to the QR.

A key event in Task 3.2 was the recruitment of a post-doctoral research fellow, Dr Frédéric Tuong. He brought a considerable amount of expertise in theorem proving to the activity.

During Task 3.2, the formal verification approach was eventually settled as follows:

- We would use Promela/SPIN for modelling and test generation.
- We would address three RTEMS features in a series of case-studies to build up expertise and results:
    1. Use a part of the Chains API as a initial case-study to develop the modelling and test-generation methods.
    2. Take the technique developed and apply it to the Event Manager
    3. Finally, look at the verification of the thread queue code used to implement the MrsP SMP scheduler protocol ([BW13]).

Case-studies 1 and 2 are essentially complete. Case-study 3 on the thread-queues has provided an abstract top-level model of their expected behaviour, when accessed via the use or MrsP semaphores. Another planned model of thread-queue code behaviour is incomplete. This is due to circumstances which radically reduced the availability to Task 3 of crucial domain expertise from key players in the rest of the consortium.

The deliverable document [BT21] describes the Task 3.2 outcomes under the headings of Methodology, Tools, User Manual, and three sections for the Case Studies.

## 2.3 Task 3.3

Task 3.3 is the production of *this* report.

It was scheduled to run for 1 month from the QR to the AR/FR.

CHAPTER

# THREE

# METHODOLOGY

## 3.1 Promela/SPIN

Our chosen formal modelling is Promela/SPIN (`spinroot.com`). Promela is the modelling language, while SPIN is the model checking tool.

The modelling language is used to describe concurrent processes executing with global shared state, as well as messaging mechanisms. A key feature of the models is that they embody the non-determinism found in concurrent systems. The language also provides a variety of ways to specify desired properties, ranging from special statement labels, the `assert(...)` statement, up to temporal properties expressed in linear temporal logic.

SPIN runs in two modes. The first, *simulation*, runs the model from the start, making random choices whenever non-determinism occurs, and halting if the model terminates, or a certain class of errors occur (deadlock, failed `assert()`,...). The second, "*verification*", explores all possible paths through the model. If no errors are uncovered, SPIN reports success. If an error is discovered, the sequence of events leading to that failure (a *counterexample*) is issued in a so-called *trail* file. SPIN can be set to halt when one error is encountered, but can also be asked to search for all error paths. Each trail file can be re-played by SPIN to show the details of the erroneous scenario.

## 3.2 Scenario Notation

While SPIN produces plenty of output to allow a user see what is happening, it is not easy to parse. Promela has a `printf` statement that can be used by a user to produce tailored output. These `printf` statements are executed during a simulation run, or when a trail file is replayed. They are not output during verification.

We use this facility to output all behaviour of interest in a easy to parse format. All of these lines are flagged by a prefix marker ("@@@") that makes it easy for the test generation tools to filter them out from the regular SPIN output. We refer to these lines as model, feature or scenario *annotations*.

## 3.3 Test Generation

The key idea for test generation using model-checkers is the following:

> Phase 1: develop a correct model by using the model-checker as normally intended
>
> Phase 2: Pick a property (known be true when Phase 1 is completed), negate it, and run a verification. The model checker will establish its false and generate a counterexample. This is in fact a scenario showing correct model behaviour associated with that property.

With this process, we can obtain trail files that represent correct (i.e. predicted) behaviours of the modelled systems. We replay these with SPIN to obtain the resulting scenario annotations.

The test generation software takes these annotations and uses them to lookup a YAML dictionary that maps annotations to RTEMS test code. These test code fragments are stitched together with some boilerplate code to produce valid RTEMS test programs.

There are a number of configurations under which these tests can be run. We can use simulators provided as part of the RTEMS tools (e.g. `sis`), and we also had access to real Leon processor hardware. Tests can also be performed in settings where SMP is disable or enabled.

CHAPTER

FOUR

# SELECTED ALGORITHMS

## 4.1 The Chains API

The first case study was exploring how to automatically generate tests from Promela/SPIN models. This was initially done by looking at a part of the Chains API, with a focus on getting end-to-end results, from a Promela model, to real test code, actually run on real hardware.

The Chains API provides a doubly-linked list data-structure designed for efficient implementation of LIFO/FIFO queues and similar. It has been designed to support fast constant-time update operations with a small memory footprint. The API is quite large with many calls, but we focussed on just two: `rtems_chain_append` and `rtems_chain_get`.

We designed a Promela model (*chains_api_model*) that gave a model of the code for the append and get calls, as well as 6 concurrent processes, 3 of which performed appends, while the other 3 performed gets on non-empty chains (the get processes blocked when a chain was empty).

A processes interacted with a single chain object.

The result of running test-generation is that we got all feasible interleavings (21) of three appends and three gets.

Despite the concurrent nature of the model, all generated test scenarios resulted in the 6 API calls being done in a fixed order in each given test by a single RTEMS task. Consequently, these tests have always worked on all test configurations.

## 4.2 The Event Manager

The RTEMS Event Manager was chosen as the second case-study because it involved concurrency and communication, had a small number of API calls (just two), but also had somewhat complex requirements related to task priorities.

The Event Manager allows tasks to send “events” to, and receive “events” from, other tasks. From the perspective of the Event Manager, “events” are just uninterpreted numbers in the range 0..31, encoded as a 32-bit bitset.

**`rtems_event_send(id,event_in)`** allows a task to send a bitset to a designated task

**`rtems_event_receive(event_in,option_set,ticks,event_out)`** allows a task to specify a desired bitset with options on what to do if it is not present.

Most of the requirements are pretty straightforward, but two were a little more complex, and drove the more complex parts of the modelling.

1. If a task was blocked waiting to receive events, and a lower priority task then sent the events that would wake that blocked task, then the sending task would be immediately preempted by the receiver task.
2. There was a requirement that explicitly discussed the situation where the two tasks involved were running on different processors.

The main changes to the Promela modelling and the annotations were to add features that indicated when tasks got swapped in/out, and to provide semaphores needed to ensure that a specific model scenario produced a test with the same interleaving behaviour. A key part of that was that all annotations contained the Promela process id associated with that annotations. The `spin2test` tools then generates separate segments of C test code, one for each Promela process, and notes when once process is suspended in favour of another.

At present, we have chosen to restrict the applicable test configurations to those that have at least two processors and are compiled with `RTEMS_SMP` defined.

## 4.3 The MrsP Thread Queues

The third case study dealt with one of the most critical, and highly complex, parts of RTEMS, namely the updates done to the scheduler to support SMP. In particular, the implementation of the MrsP protocol [BW13] is very important. The adoption of this protocol in RTEMS has led to a number of necessary revisions and improvements [CBHM15] [GZBW17] [Gom19]

A particular area of concern are the thread queue implementations that are a critical part of this protocol.

A variety of Promela models were developed, as background reading was done, before we started to focus on the thread queues. There were mainly various forms of weak memory models, including a model of the TSO mechanism used by the SPARCv8 architecture.

We then turned our focus to the thread queues. After some discussion, we defined an end-to-end scenario that captured all the relevant setup and use of these queues. This scenario consists of a number of tasks of different priorities running on a number of cores, accessing a number of shared resources. The resource access is mediated by the use of RTEMS semaphores initialised to use the MrsP protocol.

When the started to look at modelling the MrsP algorithm as implemented in RTEMS. This basically required us to model the RTEMS code to create, initialise, obtain and release such semaphores. The idea was to use the model-checker to check for desired behaviour, and lack of undesired behaviour such as deadlock or livelock.

Considerable progress has been made in this regard, but has stalled. The issue is that it is very hard to determine the relevant parts of the thread queue state, and how these are initialised. This is because system initialisation in RTEMS is itself quite complex, and much of the MrsP stuff appears as “add-ons”. This has all been exacerbated by the fast that access to the relevant domain expert has been very highly curtailed due to workload issues.

As a workaround, we then decided to focus on a high-level abstract model of MrsP behaviour, that could be used in the future to produce a validation testsuite. This model is based on the

idea that from a high-level, the MrsP behaviour should look just like the observable behaviour of the single-core priority inheritance algorithm from which it was derived.

CHAPTER

FIVE

# CONCLUSIONS

## 5.1 Achievements

We have developed a framework for using Promela/SPIN to provide models of RTEMS behaviour, with tools to support the generation of tests from those models. We used the Chains API to prototype initial ideas, and then the Event Manager to explore issues pertaining to concurrency, priority, and multi-core. We have made progress in modelling the complex thread-queue system that is the core of a key multi-score scheduling algorithm in RTEMS.

### 5.1.1 Unfinished Work

The thread-queue needs more work. The simplest to achieve is getting tests generated from the high level model. More difficult is the code model. The issue here is how the MrsP thread queues are initialised, and will some help from an RTEMS scheduler domain expert. The process of modelling the algorithm code is straightforward, but checking it out and validating it is not possible until the initial state can be determined and modelled.

## 5.2 Next Steps

It has always been the intention that the results of this activity would be returned to the RTEMS community, and active development and maintenance would continue into the future. Indeed, there has already been work done here by masters-level students at Trinity College Dublin, and more are taking up ideas from this material.

We expect that we will extend the number of RTEMS Managers that we will model, and will continue to work towards completion of the thread queue manager work. Getting tests generated from the high-level model would be a start.

The code model can proceed, and would benefit from a high-level description of what an initialised MrsP system looks like.

### 5.2.1 Opportunities

As an unexpected side-effect of this activity, there is now a large comprehensive resource in RTEMS that could be used as input to a number of downstream formal analysis activities. These are the Specification Items developed for the activity, and described in Section 5.2 of the RTEMS Software Engineering manual [con21]. This provides a hierarchy of YAML files that is easy to navigate and contains a lot of information from which RTEMS semantics could be inferred. Of particular interest are the `action-refinement` specifications, that provide pre- and post-conditions for RTEMS code, with a mapping between them. They are described abstractly using names, but are also associated with corresponding RTEMS C code. They are used for the test code generation done in Task 2. There is high scope for tools that can take these specification items and use them to drive some kind of formal analysis.

# BIBLIOGRAPHY

[BW13] A. Burns and A. J. Wellings. A Schedulability Compatible Multiprocessor Resource Sharing Protocol - MrsP. In *Proceedings of the 25th Euromicro Conference on Real-Time Systems (ECRTS 2013)*. 2013. URL: http://www-users.cs.york.ac.uk/~burns/MRSPpaper.pdf.

[BH21] Andrew Butterfield and Mike Hinchey. *Formal Verification Plan*. 2021. URL: \protect\T1\textdollar\protect\T1\textbraceleft/variant:/deployment-directory\protect\T1\textbraceright/doc/fm/fvp/FV1-200.pdf.

[BT21] Andrew Butterfield and Frédéric Tuong. *Formal Verification Artefacts*. 2021. URL: \protect\T1\textdollar\protect\T1\textbraceleft/variant:/deployment-directory\protect\T1\textbraceright/doc/fm/fva/FV2-201.pdf.

[CBHM15] Sebastiano Catellani, Luca Bonato, Sebastian Huber, and Enrico Mezzetti. Challenges in the Implementation of MrsP. In *Reliable Software Technologies - Ada-Europe 2015*, 179–195. 2015.

[con21] The RTEMS Project contributors. RTEMS Software Engineering, Git Commit $/steps/clone-rtems-docs:/commit. 2021. URL: \protect\T1\textdollar\protect\T1\textbraceleft/variant:/deployment-directory\protect\T1\textbraceright/doc/rtems/eng/eng.pdf.

[GZBW17] Jorge Garrido, Shuai Zhao, Alan Burns, and Andy J. Wellings. Supporting nested resources in mrsp. In Johann Blieberger and Markus Bader, editors, *Reliable Software Technologies - Ada-Europe 2017 - 22nd Ada-Europe International Conference on Reliable Software Technologies, Vienna, Austria, June 12-16, 2017, Proceedings*, volume 10300 of Lecture Notes in Computer Science, 73–86. Springer, 2017. URL: https://doi.org/10.1007/978-3-319-60588-3\T1\textbackslash{}_5, doi:10.1007/978-3-319-60588-3_5.

[Gom19] Ricardo Gomes. Analysis of MrsP Protocol in RTEMS Operating System. Master's thesis, CISTER, Departmento de Engenharia Informática, Instituto Superior de Engenharia do Porto (ISEP), Portugal, 2019.

 