Development of an ARINC 653 compatible Operating System
Technical report

Luís Fernando Arcaro
luis.arcaro@posgrad.ufsc.br

Rômulo Silva de Oliveira
romulo.deoliveira@ufsc.br

Systems and Automation Department (DAS),
Federal University of Santa Catarina (UFSC),
Florianópolis, Santa Catarina, Brazil

Abstract
This technical report is an extension of the article entitled “Design aspects of the development of ARINC 653 compatible Operating Systems”, which will be referred simply as “the main article”. The content of this report assumes that the main article, in which we describe the design of an experimental and educational ARINC 653 compatible OS that runs on the BeagleBone hardware platform, has already been read.

1 Introduction
In this technical report we describe the memory regions and the permission mapping we employed in the Operating System (OS) we developed in order to meet the spatial isolation requirements imposed by the ARINC 653 specification. We also present the test cases we created in order to provide evidences that the developed OS behaves as expected by the ARINC 653 specification and that it’s, therefore, useful for the purposes it’s aimed to.

2 Memory regions
We present below the memory regions which are generally found in ARINC 653 systems, associated to the acronyms by which they will be referenced next.

Module’s code (MOD.COD) Stores the code of the OS core and of the module’s execution contexts (MOD.DEF and MOD.HMC).

Module’s data (MOD.DAT) Stores the data used by the code stored in the module’s code region (MOD.COD).

Module’s stack (MOD.STK) Used as stack by the module’s default partition (MOD.DEF) and by the processor’s exception handling routines.

Module’s HM callback stack (MOD.HMC.STK) Used as stack by the module’s HM callback (MOD.HMC).

Partition’s code (PAR.COD) Stores the code of a given partition’s execution contexts (PAR.DEF, PAR.HMC, PAR.EH, and PRO).
Partition’s data (PAR.DAT) Stores the data used by the code stored in the associated partition’s code region (PAR.COD).

Partition’s default process stack (PAR.DEF.STK) Used as stack by the partition’s default process (PAR.DEF).

Partition’s error handler stack (PAR.EH.STK) Used as stack by the partition’s error handler (PAR.EH).

Partition’s HM callback stack (PAR.HMC.STK) Used as stack by the partition’s HM callback (PAR.HMC).

Process’s stack (PRO.STK) Used as stack by a partition’s process (PRO).

The following additional memory regions were defined in the developed OS, due both to characteristics of the employed hardware platform and design decisions we made throughout its development.

Exception vector table (VECTBL) Stores the processor’s exception handling routines branch instructions.

Peripherals registers (PERPHR) Region in which all of the processor’s peripherals memory-mapped registers are contained.

First and second level translation tables (FLTT and SLTT) Stores the processor’s Memory Management Unit (MMU) first and second level translation tables, respectively [1].

Module’s and partitions’ dynamic allocation (MOD.HP and PAR.HP) Used for variable-length memory segments dynamic allocation at module and at partition level, respectively.

Partition’s data image (PAR.DAT.IMG) Contains the copy of a partition’s data region, which is used to restore it when the partition is restarted.

3 Permission mapping

We present below the permission mapping used in the OS we developed for each of the system’s memory regions which were presented in the previous section.

Module’s code (MOD.COD) Contains code that must be executable from all of the execution contexts, since it’s where the services provided by the OS are located. In order to execute in privileged mode OS services are usually invoked as software interrupts, but in the OS we developed they are invoked as ordinary methods (in unprivileged mode) which, once started, require privilege to be raised by invoking a software interrupt. Therefore, we assign this region read-only permission at any privilege level for all of the execution contexts.

Module’s data (MOD.DAT) Since the OS code can be run from any execution context and considering that all of its portions that manipulate data run in privileged mode, we assign all the execution contexts at least permission for reading and writing in this region in privileged mode. The
MOD.DEF and MOD.HMC execution contexts run in unprivileged mode and their data is stored in this region, therefore they must also have read and write permission to it in unprivileged mode. Since it’s a data region, execution is forbidden.

Module’s stack (MOD.STK) Since exception handling routines use this region as stack, run in privileged mode and can be triggered from any task, all of the execution contexts are assigned at least read and write permission in privileged mode to this region. For the MOD.DEF execution context, which runs in unprivileged mode and uses the module’s stack, we also assign read and write permission in unprivileged mode. Since it’s a stack region, execution is forbidden.

Module’s HM callback stack (MOD.HMC.STK) Since it uses this region as stack, the MOD.HMC execution context has full access to it. Since it’s a stack region, execution is forbidden.

Partition’s code (PAR.COD) May only be executed (read) by the PAR.DEF, PRO, PAR.EH, and PAR.HMC execution contexts that belong to the partition to which the region belongs.

Partition’s data (PAR.DAT) The execution contexts of the partition to which the region is related (PAR.DEF, PRO, PAR.EH, and PAR.HMC) are allowed to read and write this region at any privilege level, since their data is stored in it. Since this region may be restored by the OS when the partition is restarted, which may be triggered during the execution of any task due to errors, we also provide read and write permission in privileged mode for all of the execution contexts. Since it’s a data region, execution is forbidden.

Partition’s default process stack (PAR.DEF.STK) Since it uses this region as stack, the PAR.DEF execution context of the partition to which the region is related has full access to it. Since it’s a stack region, execution is forbidden.

Partition’s error handler stack (PAR.EH.STK) Since it uses this region as stack, the PAR.EH execution context of the partition to which the region is related has full access to it. Since it’s a stack region, execution is forbidden.

Partition’s HM callback stack (PAR.HMC.STK) Since it uses this region as stack, the PAR.HMC execution context of the partition to which the region is related has full access to it. Since it’s a stack region, execution is forbidden.

Process’s stack (PRO.STK) Since it uses this region as stack, the PRO execution context of the process to which the region is related has full access to it. In order to enable optimizations employed by ARINC 653 intrapartition communication mechanisms, specifically buffers, in which messages may be copied directly between local variables of two processes of a same partition [2], the execution context PRO of the other processes of the partition to which the owner process belongs are also allowed to
read and write in this region, but only in privileged mode. Since it’s a stack region, execution is forbidden.

**Exception vector table (VECTBL)** In the ARMv7 architecture the exception vector table consists of instructions that divert the flow of execution to appropriate handling procedures, which run in privileged mode and may be triggered from any task. Therefore, all execution contexts must be allowed to execute (read) this region in privileged mode.

**Peripherals registers (PERPHR)** This region’s permissions define which execution contexts will be able to directly access the processor’s peripherals and, given the educational orientation of the OS we developed, we decided to allow access to these registers from any execution context, assigning them read and write permissions at any privilege level. We highlight that this approach does not meet the ARINC 653 spatial isolation requirements, since it allows execution contexts that belong to different partitions to perform write operations in this region [2], but we decided to use it in order to facilitate the processor’s peripherals’ access to the user. Since it’s not a code region, execution is forbidden.

**First and Second level translation tables (FLTT and SLTT)** The MMU translation tables are usually prepared by OS services which are invoked from the module’s default partition (MOD.DEF) during the module initialization or from the partitions’ default processes (PAR.DEF), but they can also be prepared from system partitions’ processes (PRO) if necessary. We therefore allow these execution contexts to read and write these regions, but only in privileged mode. Since these are not code regions, execution is forbidden.

**Module’s dynamic allocation (MOD.HP)** From the access permissions point of view this region is similar to the module’s data region, since it stores similar purpose information but which are dynamically allocated at runtime, but with no unprivileged mode permissions since it’s intended to be used exclusively by the OS core.

**Partition’s dynamic allocation (PAR.HP)** From the access permissions point of view this region is similar to the partition’s data region, since it stores similar purpose information but which are dynamically allocated at runtime, but with no unprivileged mode permissions since it’s intended to be used exclusively by the OS core. On top of that, this region is not restored when its related partition is restarted, reason for which we also revoke the permissions which would be necessary to perform this operation.

**Partition’s data image (PAR.DAT.IMG)** Since restarted partitions’ data region restoration is performed by the OS core and may be triggered during the execution of any task, we assign this region read and write permission in privileged mode for all of the execution contexts. Since it’s not a code region, execution is forbidden.

We present in Table [1] the complete permission mapping employed by the ARINC 653 compatible OS we developed. These permissions are expressed
in the PRV/NPR[*] format, where PRV and NPR are, respectively, the permissions for privileged (used by the OS) and unprivileged (used by the application software) execution modes, and can assume the values NO (no permission), RO (read-only), or RW (read and write). The optional indicator * denotes that code execution is not allowed in the region. Three different permissions are used in each mapping (default, partition, and process), and they are applied as follows:

1. If the execution context belongs to the same process as the memory region and the process permission is not empty, the process permission is applied.

2. If the previous condition is not satisfied, the execution context and the memory region are related to the same partition and the partition permission is not empty, the partition permission is applied.

3. If none of the previous conditions are satisfied and the default permission is not empty, the default permission is applied.

4. If none of these conditions is satisfied, the access to the region is forbidden.

System partitions need a permission assignment which allows maximum operation flexibility from them, but still keeps their spatial isolation. Therefore, system partitions executed in the OS we developed are subject to basically the same permission mapping as application partitions, but system ones have the following special characteristics:

**Execution in privileged mode** Their execution contexts always run in privileged mode, having therefore full control over the processor.

**Access to data regions** System partitions have permission to read and write in all system memory regions that contain data, and can therefore exchange information between application partitions.

**Access to peripherals registers** System partitions are the correct way of interacting with hardware elements in ARINC 653 systems [2], and they therefore have full access to the processor’s peripherals registers.
<table>
<thead>
<tr>
<th>Memory region</th>
<th>Execution context</th>
<th>Default</th>
<th>Partition</th>
<th>Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOD.COD</td>
<td>– ALL –</td>
<td>RO/RO</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>MOD.DAT</td>
<td>MOD.DEF RW/RW*</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>MOD.HMC RW/RW*</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>PAR.DEF RW/NO*</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>PAR.EH RW/NO*</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>PAR.HMC RW/NO*</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>PRO RW/NO*</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>MOD.STK</td>
<td>MOD.DEF RW/RW*</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>MOD.HMC RW/NO*</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>PAR.DEF RW/NO*</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>PAR.EH RW/NO*</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>PAR.HMC RW/NO*</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>PRO RW/NO*</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>MOD.HMC.STK</td>
<td>MOD.HMC RW/RW*</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>PAR.COD</td>
<td>PAR.DEF -</td>
<td>RO/RO</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>PAR.EH -</td>
<td>RO/RO</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>PAR.HMC -</td>
<td>RO/RO</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>PRO -</td>
<td>RO/RO</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>PAR.DAT</td>
<td>MOD.DEF RW/NO*</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>MOD.HMC RW/NO*</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>PAR.DEF RW/NO*</td>
<td>RW/RW*</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>PAR.EH RW/NO*</td>
<td>RW/RW*</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>PAR.HMC RW/NO*</td>
<td>RW/RW*</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>PRO RW/NO*</td>
<td>RW/RW*</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>PAR.DEF.STK</td>
<td>PAR.DEF -</td>
<td>RW/RW*</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>PAR.EH.STK</td>
<td>PAR.EH -</td>
<td>RW/RW*</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>PAR.HMC.STK</td>
<td>PAR.HMC -</td>
<td>RW/RW*</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>PRO.STK</td>
<td>PRO -</td>
<td>RW/NO*</td>
<td>RW/RW*</td>
<td>–</td>
</tr>
<tr>
<td>VECTBL</td>
<td>– ALL –</td>
<td>RO/NO</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>PERPHR</td>
<td>– ALL –</td>
<td>RW/RW*</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>FLTT</td>
<td>MOD.DEF RW/NO*</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>PAR.DEF RW/NO*</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>PRO RW/NO*</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>SLTT</td>
<td>MOD.DEF RW/NO*</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>PAR.DEF RW/NO*</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>PRO RW/NO*</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>MOD.HP</td>
<td>– ALL –</td>
<td>RW/NO*</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>PAR.HP</td>
<td>PAR.DEF -</td>
<td>RW/NO*</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>PAR.EH -</td>
<td>RW/NO*</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>PAR.HMC -</td>
<td>RW/NO*</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>PRO -</td>
<td>RW/NO*</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>PAR.DAT.IMG</td>
<td>– ALL –</td>
<td>RW/NO*</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

Table 1: Complete permission mapping of the developed OS
4 Test cases

In this section we present the complete list of the test cases we developed in order to provide evidences that the developed OS behaves as expected by the ARINC 653 specification and that it’s, therefore, useful with experimental and educational objectives. The tests are categorized according to the kind of system behavior or feature they are aimed to analyze.

• Scheduling and execution:
  
  **SLOWPARTITIONSCHEDULING**  Slow partition scheduling test (with time windows in the order of seconds).
  
  **FASTPARTITIONSCHEDULING**  Fast partition scheduling test (with time windows in the order of milliseconds).
  
  **SLOWPROCESSSCHEDULING**  Slow periodic process scheduling test (with periods in the order of seconds).
  
  **FASTPROCESSSCHEDULING**  Fast periodic process scheduling test (with periods in the order of milliseconds).
  
  **SYSTEMPARTITION**  Test case that runs a system partition that accesses the OS core information.

• ARINC 653 basic services:
  
  **SETMODULEMODE_IDLE**  Module stopping test.
  
  **SETMODULEMODE_COLDSTART**  Module reinitializing test.
  
  **GETPARTITIONID**  Partition identifier reading test.
  
  **GETPARTITIONSTATUS**  Partition status reading test.
  
  **SETPARTITIONMODE_IDLE_PROCESS**  Partition stopping from application process test.
  
  **SETPARTITIONMODE_IDLE_PARTITIONERRORHANDLER**  Partition stopping from error handler process test.
  
  **SETPARTITIONMODE_IDLE_PARTITIONHMCALLBACK**  Partition stopping from HM callback test.
  
  **SETPARTITIONMODE_COLDSTART_PROCESS**  Partition reinitializing from application process test.
  
  **SETPARTITIONMODE_COLDSTART_PARTITIONERRORHANDLER**  Partition reinitializing from error handler process test.
  
  **SETPARTITIONMODE_COLDSTART_PARTITIONHMCALLBACK**  Partition reinitializing from HM callback test.
  
  **GETPROCESSID**  Process identifier reading test.
  
  **GETPROCESSSTATUS**  Process status reading test.
  
  **SETPRIORITY**  Process priority setting test.
  
  **SUSPEND_RESUME**  Process suspending and resuming test.
  
  **SUSPENDSELF_RESUME**  Process self-suspending and resuming test.
**SUSPENDSELF** **RESUME** **TIMEOUT**  Process timed out self-suspending and resuming test.

**TIMEDWAIT**  Process timed waiting test.

**TIMEDWAIT SUSPEND** **RESUME**  Process suspending and resuming during timed wait test.

**TIMEDWAIT COOPERATIVESCHEDULING**  Cooperative scheduling test using **TIMED WAIT** service.

**REPLENISH**  Process deadline postponement test.

**STOPSELF** **START**  Process self-stopping and restarting test.

**STOPSELF DELAYEDSTART**  Process self-stopping and delayed restarting test.

**STOP** **DELAYED** **START**  Process stopping and delayed restarting test.

**STOP** **START**  Process stopping and restarting test.

**STOP** **START** **PERIOD** **START**  Periodic process stopping and restarting test, aimed to verify its first release, which must be at the start of the next period of the partition.

**STOP DELAYEDSTART**  Process stopping and delayed restarting test.

**DELAYEDSTART**  Process delayed starting test.

**LOCKPREEMPTION UNLOCKPREEMPTION**  Preemption locking test.

**LOCKPREEMPTION STOPSELF PROCESS**  Process self-stopping with preemption locked test.

**LOCKPREEMPTION STOPSELF_PARTITIONERRORHANDLER**  Error handler process self-stopping with preemption locked test.

**GETMYID**  Current process identifier reading test.

- **ARINC 653 synchronization and communication services:**
  - **EVENT**  Event test.
  - **SEMAPHORE_FIFO**  First In First Out (FIFO) semaphore test.
  - **SEMAPHORE_PRIORITY**  Priority semaphore test.
  - **SEMAPHORE DEADLOCK**  Semaphore deadlock test.
  - **BLACKBOARD**  Blackboard test.
  - **BUFFER**  Buffer test.
  - **BUFFER_FIFO**  FIFO buffer test.
  - **BUFFER_PRIORITY**  Priority buffer test.
  - **BUFFER_TIMEOUT RECEIVE**  Buffer time-limited receiving test.
  - **BUFFER_TIMEOUT SEND**  Buffer time-limited sending test.
  - **BUFFER FULL EMPTY**  Buffer full and empty states test.
  - **SAMPLINGPORT STANDARD**  Intra-module sampling ports test.
  - **SAMPLINGPORT PSEUDO MODULE1 MODULE2**  Inter-module sampling ports test (requires two interconnected platforms).
  - **QUEUINGPORT STANDARD**  Intra-module queuing ports test.
**QUEUINGPORT**

PSEUDO

**MODULE1/MODULE2**

Inter-module queuing ports test (requires two interconnected platforms).

- **Health Monitoring:**
  - **HEALTHMONITORING_CURRENTSYSTEMSTATE** Health Monitoring system state detection test.
  - **HEALTHMONITORING_PROPAGATION_APPLICATIONERROR** Application-raised errors propagation test.
  - **HEALTHMONITORING_PROPAGATION_STACKOVERFLOW** Stack overflow errors propagation test.
  - **HEALTHMONITORING_PROPAGATION_MEMORYVIOLATION** Memory violation errors propagation test.

**References**
