ARINC 653 Migration Guide

LynxOS-178

DOC-2211-00

Product names mentioned in this document are trademarks of their respective manufacturers and are used here for identification purposes only.

Printed in the United States of America.

Table of Contents

Preface

Typographical Conventions

The typefaces used in this manual, summarized below, emphasize important concepts. All references to filenames and commands are case-sensitive and should be typed accurately.

Kind of Text	Examples
Body text; *italicized* for emphasis, new terms, and book titles	Refer to the *ARINC 653 Migration Guide*
Environment variables, filenames, functions, methods, options, parameter names, path names, commands, and computer data	`ls -l myprog.c /dev/null`
Commands that need to be highlighted within body text, or commands that must be typed as is by the user are **bolded**.	`login:` **`myname`** `#` **`cd /usr/home`**
Text that represents a variable, such as a filename or a value that must be entered by the user, is *italicized*.	`cat <filename>` `mv <file1> <file2>`

Blocks of text that appear on the display screen after entering instructions or commands	`Loading file /tftpboot/shell.kdi into 0x4000`
	`.....`
	`File loaded. Size is 1314816`
	`© 2015 Lynx Software Technologies, Inc. All rights reserved.`

Keyboard options, button names, and menu sequences	**`Enter`**, **`Ctrl-C`**

Technical Support

Lynx Software Technologies handles support requests from current support subscribers. For questions regarding Lynx Software Technologies products, evaluation CDs, or to become a support subscriber; our knowledgeable sales staff will be pleased to help you. Please visit us at:
http://www.lynx.com/training-support/contact-support/

How to Submit a Support Request

When you are ready to submit a support request, please include *all* of the following information:

- First name, last name, your job title
- Phone number, e-mail address
- Company name, address
- Product version number
- Target platform (for example, PowerPC)
- Board Support Package (BSP), Current Service Pack Revision, Development Host OS version
- Detailed description of the problem that you are experiencing:
- Is there a requirement for a US Citizen or Green Card holder to work on this issue?
- Priority of the problem - Critical, High, Medium, or Low?

Where to Submit a Support Request

Support, Europe	tech_europe@lynx.com +33 1 30 85 93 96
Support, worldwide except Europe	support@lynx.com +1 800-327-5969 or +1 408-979-3940 +81 33 449 3131 [for Japan]
Training and Courses	USA: training-usa@lynx.com Europe: training-europe@lynx.com USA: +1 408-979-4353 Europe: +33 1 30 85 06 00

Introduction

Beginning with LynxOS 2.2.4, each multi-process ARINC 653 application (partition) is built as single multi-threaded POSIX process which is intended to be run in a dedicated LynxOS-178 VM. In earlier versions of LynxOS-178, a multi-process ARINC 653 application was built using multiple single-threaded POSIX processes. This design did not conform to the ARINC 653 standard. As a result, a number of changes are required to ARINC 653 applications built using the earlier LynxOS-178 design.

Differences in implementation details and usage of the ARINC 653 library and driver in LynxOS-178 2.2.4 (and later releases) compared to earlier releases are explained below.

The reference to "earlier versions" includes both those releases that are numerically lower than 2.2.4 as well as releases on the 2.3.x branch (2.3.0, 2.3.1, etc.). The term "newer versions" will be used to refer to LynxOS-178 2.2.4 and later releases.

ARINC 653 Linking

In earlier versions of LynxOS-178 it was enough to link with the arinc653 library. In LynxOS-178 2.2.4 and later versions the user must specify the health monitor library as well.

ARINC 653 Linking with Libraries	
Earlier Versions of LynxOS-178	**LynxOS-178 2.2.4 and later versions**
The health monitor library was not required	The following libraries in the following order need to be specified for linker to build ARINC 653 applications:
`$ gcc -o example example.c -larinc653`	`$ gcc -o example example.c -larinc653 -lhm`

ARINC 653 Applications Starting Point

Newer versions of LynxOS-178 implement the "main()" function for all ARINC 653 applications in the libarinc653.a library. As such, user applications must not define this function, instead they must define "arinc653_main()" as their starting point. POSIX applications using the ARINC 653 inter-partition APIs do not need to define the `arinc653_main()` function.

ARINC 653 Application Starting Point	
Earlier Versions of LynxOS-178	**LynxOS-178 2.2.4 and later versions**
Applications start from the “main()” function.	Applications start from the “arinc653_main()” function. ARINC 653 applications MUST NOT define “main()”.

application.c:
int main(void)
{
...
}
	application.c:
	void arinc653_main(void)
	{
	...
	}

ARINC 653 APIs in POSIX Applications

It is possible to use the following ARINC APIs from POSIX applications (and partitions in case of inter-partition communication):

```
CREATE_ERROR_HANDLER()
GET_ERROR_STATUS()
REPORT_APPLICATION_MESSAGE()
RAISE_APPLICATION_ERROR()
CREATE_SAMPLING_PORT()
WRITE_SAMPLING_MESSAGE()
READ_SAMPLING_MESSAGE()
READ_SAMPLING_MESSAGE_CONDITIONAL()
READ_UPDATED_SAMPLING_MESSAGE()
GET_SAMPLING_PORT_ID()
GET_SAMPLING_PORT_STATUS()
GET_SAMPLING_PORT_CURRENT_STATUS()
CREATE_QUEUING_PORT()
SEND_QUEUING_MESSAGE()
RECEIVE_QUEUING_MESSAGE()
GET_QUEUING_PORT_ID()
GET_QUEUING_PORT_STATUS()
CLEAR_QUEUING_PORT()
```

ARINC 653 in POSIX Applications	
Earlier Versions of LynxOS-178	**LynxOS-178 2.2.4 and later versions**
It was necessary to call the `ARINC653_OPENDEV_FUNC` macro before using any other ARINC 653 interfaces.	The `ARINC653_OPENDEV_FUNC` macro is deprecated.

`if (ARINC653_OPENDEV_FUNC()) {`
`  /* ... handle failure ... */`
`}`
	`/* The ARINC653_OPENDEV_FUNC macro is no longer needed */`

ARINC 653 in POSIX Applications	
Earlier Versions of LynxOS-178	**LynxOS-178 2.2.4 and later versions**
It was necessary to invoke the `CREATE_PARTITION ioctl` of the ARINC 653 device driver before using any other ARINC 653 interfaces.	ARINC 653 APIs can be called without issuing the ARINC 653 driver `ioctl` command.

`if (ARINC653_IOCTL(CREATE_PARTITION, NULL) != NO ERROR){`
`  /* ... handle failure ... */`
`}`
	`/* the CREATE_PARTITION ioctl call is no longer needed */`

ARINC 653 in POSIX Applications	
Earlier Versions of LynxOS-178	**LynxOS-178 2.2.4 and later versions**
User-defined handlers for ARINC 653 APIs were installed as follows:	User defined handlers may be installed only for inter-partition communication APIs with the following `ioctl` command:

`typedef int (*FX)(struct arinc653_partition *, struct arinc653_statics *, char *);`
 	
`struct arinc653_io_install_handler ioh_arg;`
`FX fioh[2];`
 	
`ioh_arg.cmd = ARINC653_GET_MY_ID; /*get my id*/`
`ioh_arg.new_handler = fioh[1];`
`ioh_arg.old_handler_ptr = fioh;`
 	
`ioctl(arinc653_drv, ARINC653_INSTALL_HANDLER, &ioh_arg);`
	`#include <arinc/arinc653_private.h>`
	`/*`
	
	`typedef int (*arinc653_ioctl_inter_handler)(void*, void*, void*);`
	`This definition is given in arinc653_private.h */`
	
	`struct arinc653_io_install_handler ioh_arg;`
	`arinc653_ioctl_inter_handler old_handler;`
	
	`ioh_arg.cmd = ARINC653_IOCTL_INTER_GET_SAMPLING_PORT_ID; /*get sampling port id*/`
	`ioh_arg.new_handler = new_handler;`
	`ioh_arg.old_handler_ptr = old_handler;`
	`ioctl(arinc653 drv, ARINC653_IOCTL_INTER_INSTALL_HANDLER, &ioh_arg);`

Formerly, virtually all ARINC 653 APIs may have been called by issuing the corresponding driver `ioctl` command, this behavior is deprecated in newer versions of LynxOS-178.

POSIX applications using the ARINC 653 inter-partition communication APIs shall not be linked with the `libarinc653` library. For POSIX applications, these APIs are located in the C library, the arinc653 library is only intended to be used by ARINC 653 applications.

ARINC 653 Application Instantiation

In earlier version of LynxOS-178, the “arinc653_init” application was used to instantiate an ARINC 653 application. This is no longer necessary. Instead, the application can be called directly using the VCT command “CommandLine”

.

ARINC 653 Application Instantiation	
Earlier versions of LynxOS-178	**LynxOS-178 2.2.4 and later versions**
It was necessary to execute the "arinc653_init" application to instantiate an ARINC 653 application.	ARINC 653 applications are instantiated by calling the actual application. The choice of name and location is at the user's discretion.

`VCT:`
 	
`CommandLine=/arinc653_init;`
	`VCT:`
	
	`CommandLine=/usr/bin/arinc653_user_app;`

ARINC 653 Process Creation

Due to the design of the ARINC 653 implementation in previous versions of LynxOS-178, every process that would be called by an application had to be listed in the driver info file. The *`arinc653_info_proc[]`* array was used to list the ARINC 653 name of the process and the path to the external binary that implemented that process. In newer versions of LynxOS-178, the *`arinc653_info_proc[]`* is no longer used. Process creation follows the standard ARINC 653 model. The number of processes in the partition that was configured in the *`arinc653_info`* structure is also no longer needed. Instead, the VCT controls the number of ARINC processes (POSIX threads) that can be created in the partition.

ARINC 653 Error Handler Process Creation	
Earlier versions of LynxOS-178	**LynxOS-178 2.2.4 and later versions**
ARINC 653 process entry point was pre-configured in the driver info file. Each ARINC process was started as a separate POSIX process within a partition.	ARINC 653 processes (POSIX threads) are created by a master process of a partition (POSIX process). User needs to specify an ARINC 653 process entry point during process creation.

`PROCESS_ATTRIBUTE_TYPE AttrMaster = {"PR0_1", NULL, 101010, 2, INFINITE_TIME_VALUE, INFINITE_TIME_VALUE, SOFT};`
`PROCESS_ID_TYPE MasterTest;`
`RETURN_CODE_TYPE rc;`
 	
`CREATE_PROCESS(&AttrMaster, &MasterTest, &rc);`
	`PROCESS_ATTRIBUTE_TYPE AttrMaster = {"PR0_1", ep_Master, 101010, 2, INFINITE_TIME_VALUE, INFINITE_TIME_VALUE, SOFT};`
	`PROCESS_ID_TYPE MasterTest;`
	
	`RETURN_CODE_TYPE rc;`
	
	`CREATE_PROCESS(&AttrMaster, &MasterTest, &rc);`

ARINC 653 Error Handler Process Creation

ARINC 653 Error Handler Process Creation	
Earlier versions of LynxOS-178	**LynxOS-178 2.2.4 and later versions**
The error handler process was implemented as a separate POSIX process, which was started using the name configured in the driver info file. The `ENTRY_POINT` parameter passed to `CREATE_ERROR_HANDLER()` function was ignored.	The ARINC 653 error handler process is implemented as a POSIX thread within the partition. The user needs to specify the entry point function when creating an error handler process with `CREATE_ERROR_HANDLER()`

`info.c:`
`struct arinc653_info arinc653_info = {`
`  ...`
`  .ehpath = { "/bin/eh1", "/bin/eh2" }`
`};`
 	
`application.c:`
`void arinc653_main(void)`
`{`
`  RETURN_CODE_TYPE rc;`
`  ...`
`  CREATE_ERROR_HANDLER(NULL, 0, &rc);`
`  ...`
`}`
	`application.c:`
	
	`void errh_proc(void)`
	`{`
	`  /* handle errors here */`
	`}`
	
	`void arinc653_main(void)`
	`{`
	`  RETURN_CODE_TYPE rc;`
	`  ...`
	`  CREATE_ERROR_HANDLER(errh_proc, 0x4000, &rc);`
	`  ...`
	`}`

ARINC 653 Port Configuration

- The *arinc653_info_qport[]* and *arinc653_info_sport[]* arrays are now replaced by the *arinc653_info_ports[]* array which contain both sampling and queuing ports and the *arinc653_info_channels[]* array which defines the linkage between source and destination ports (for both queuing and sampling ports), please see detailed description below

ARINC 653 Port Configuration	
Earlier Versions of LynxOS-178	**LynxOS-178 2.2.4 and later versions**
Queuing and sampling ports were configured in the driver info file in separate arrays `arinc653_info_qport[]` and `arinc653_info_sport[]` respectively.	Queuing and sampling ports are configured in the driver info file using the `arinc653_info_ports[]` array and the `arinc653_info_channels[]` array. `arinc653_info_ports` holds information about port structure while `arinc653_info_channels` is used to define the underlying data channels for both queuing and sampling ports.

ARINC 653 Port Configuration

Earlier Versions of LynxOS-178

```
struct arinc653_info_qport
arinc653_info_qport[] = {
    { 0, 40, 5, SOURCE,
"QUEUING_PORT1" },
    { 1, 40, 5, DESTINATION,
"QUEUING_PORT2" },
    { 0, 40, 5, DESTINATION,
"QUEUING_PORT3" },
    { 1, 40, 5, SOURCE,
"QUEUING_PORT4" }
};
```

LynxOS-178 2.2.4 and later versions

```
#if !defined(DLDD) || defined(ARINC653_INFO_CHANNELS)
static_attribute_((used)) const struct
arinc653_info_channel arinc653_info_channels[] = {
  {
   .name = "CHANNEL-0",
   .max_message_size = 40,
   .max_nb_messages = 5,
   .mode = ARINC653_INFO_CHANNEL_QUEUING,
   }, {
   .name = "CHANNEL-1",
   .max_message_size = 40,
   .max_nb_messages = 5,
   .mode = ARINC653_INFO_CHANNEL_QUEUING,
  }
#ifdef DLDD
  , {}
#endif
};
#endif

#if !defined(DLDD) || defined(ARINC653_INFO_PORTS)
static_attribute_((used)) const struct
arinc653_info_port arinc653_info_ports[] = {
  {
   .vm = 0,
   .name = "QUEUING_PORT1",
   .direction = SOURCE,
   .max_nb_messages = 5,
   .channel = "CHANNEL-0"
  }, {
   .vm = 0,
   .name = "QUEUING_PORT2",
   .direction = DESTINATION,
   .max_nb_messages = 5,
   .channel = "CHANNEL-1"
  , {
   .vm = 1,
   .name = "QUEUING_PORT3",
   .direction = DESTINATION,
   .max_nb_messages = 5,
   .channel = "CHANNEL-0"
  }, {
   .vm = 1,
   .name = "QUEUING_PORT4",
   .direction = SOURCE,
   .max_nb_messages = 5,
   .channel = "CHANNEL-1"
  }
#ifdef DLDD
  , {}
#endif
};
#endif
```

- “static const struct arinc653_info_channel arinc653_info_channels;"

This data structure contains information about core module local inter-partition IPC channels used for communication between partitions. For each channel the following are specified: channel name, max message size for the channel, number of messages for the channel, and channel operation mode (queuing or sampling)

- *"static const struct arinc653_info_port arinc653_info_ports;"*

This data structure contains information about inter-partition IPC ports (channel access points). Each entry contains the following information: port name, partition (VM) id which this port is allocated to, max number of messages that port FIFO may contain (for queuing ports), port direction, channel name this port is connected to.

ARINC 653 Memory blocks configuration

In newer versions of LynxOS-178 the ARINC driver info file defines memory blocks for all partitions running on the system.

ARINC 653 Memory Block Description	
Earlier Versions of LynxOS-178	**LynxOS-178 2.2.4 and later versions**
It was not possible to specify address and access rules for a memory block. Memory block structure definition	Memory block structure definition

`struct arinc653_info_mblock {`
`  MEMORY_BLOCK_NAME_TYPE name;`
`  MEMORY_BLOCK_MODE_TYPE mode;`
`  MEMORY_BLOCK_SIZE_TYPE size;`
`};`
	`struct arinc653_info_mblock {`
	`  /* memory block name */`
	`  MEMORY_BLOCK_NAME_TYPE name;`
	`  /* physical memory address for this memory block */`
	`  /* this may be used to access memory mapped devices */`
	`  /* from partitions. If memory block is intended */`
	`  /* just for inter-partition memory sharing, */`
	`  /* use ARINC653_MBLOCK_ADDR_NONE here */`
	`  MEMORY_BLOCK_ADDR_TYPE addr;`
	`  /* size of memory block in bytes, however */`
	`  /* blocks are mapped on page basis */`
	`  MEMORY_BLOCK_SIZE_TYPE size;`
	`  /* mask of partitions which have read-only access */`
	`  /* to this memory block */`
	`  unsigned int romask;`
	`  /* mask of partitions which have read-write access */`
	`  /* to this memory block */`
	`  unsigned int rwmask;`
	`};`

It is possible to configure a memory block to be accessible only from specified partitions, this is done by setting `romask` and `rwmask` to appropriate values using `ARINC653_ACCESS_VM_ALLOWED(n)` macro.

The *`arinc653_info_mblocks`* array defined in the driver info file contains information about memory blocks available for partitions. Each memory block entry contains: memory block name, memory block physical address (NONE for inter-partition shared memory blocks), memory block size, memory block partition access masks (see table above).

ARINC 653 Driver Info Structure

In earlier versions of LynxOS-178 ARINC 653 driver info structure was responsible for configuring the number of available partitions, number of processes per partition, error handlers and numbers of intra- and inter-partition communication objects. In newer versions of LynxOS-178 this structure was simplified to hold information about inter-partition communication objects only.

ARINC 653 Application Instantiation	
Earlier Versions of LynxOS-178	**LynxOS-178 2.2.4 and later versions**
ARINC 653 driver info was used to configure entire module.	ARINC 653 driver info is only responsible for inter-partition communication primitives configuration.

`struct arinc653_info {`
`/* Number of partitions */`
`int npart;`
`/* Processes per each partition */`
`int nprocs[SYSTEM_LIMIT_NUMBER_OF_PARTITIONS];`
`/* Sampling ports per each partition */`
`int nsports[SYSTEM_LIMIT_NUMBER_OF_PARTITIONS];`
`/* Queuing ports per each partition */`
`int nqports[SYSTEM_LIMIT_NUMBER_OF_PARTITIONS];`
`/* Events in each partition */`
`int nevents[SYSTEM_LIMIT_NUMBER_OF_PARTITIONS];`
`/* Blackboards in each partition */`
`int nbbs[SYSTEM_LIMIT_NUMBER_OF_PARTITIONS];`
`/* Buffers in each partition */`
`int nbuffers[SYSTEM_LIMIT_NUMBER_OF_PARTITIONS];`
`/* Semaphores in each partition */`
`int nsems[SYSTEM_LIMIT_NUMBER_OF_PARTITIONS];`
`/* Memory blocks in each partition */`
`int nmblocks[SYSTEM_LIMIT_NUMBER_OF_PARTITIONS];`
`/* Maximum size of error handler queues */`
`int errqsz[SYSTEM_LIMIT_NUMBER_OF_PARTITIONS];`
`/* Error handlers */`
`char ehpath[SYSTEM_LIMIT_NUMBER_OF_PARTITIONS][MAXPATHLEN];`
`/* Process information array */`
`struct arinc653_info_proc *all_procs;`
`/* Sampling port information array */`
`struct arinc653_info_sport *all_sports;`
`/* Queuing port information array */`
`struct arinc653_info_qport *all_qports;`
`/* Memory block information array */`
`struct arinc653_info_mblock *all_mblocks;`
`};`
	`struct arinc653_info {`
	`/* pointer to memory blocks info table */`
	`const struct arinc653_info_mblock *mblocks;`
	`/* number of defined memory blocks */`
	`int nmblocks;`
	`/* pointer to channel info table */`
	`const struct arinc653_info_channel* channels;`
	`/* number of defined channels */`
	`int nchannels;`
	`/* pointer to ports info table */`
	`const struct arinc653_info_port* ports;`
	`/* number of defined ports */`
	`int nports;`
	`};`

ARINC 653 Intra-partition Communication Objects Configuration	
Earlier versions of LynxOS-178	**LynxOS-178 2.2.4 and later versions**
The maximum number of ARINC 653 blackboards, events, semaphores and buffers for a partition were configured in the driver info file.	Number of available blackboards for a partition is limited by `SYSTEM_LIMIT_NUMBER_OF_BLACKBOARDS` constant. Number of available events for a partition is limited by the `SYSTEM_LIMIT_NUMBER_OF_EVENTS` constant. Number of available semaphores for a partition is limited by the `SYSTEM_LIMIT_NUMBER_OF_SEMAPHORES` Number of available buffers for a partition is limited by the `SYSTEM_LIMIT_NUMBER_OF_BUFFERS` constant.

`struct arinc653_info {`
`…`
`/* Events in each partition */`
`int`
`nevents[SYSTEM_LIMIT_NUMBER_OF_PARTITIONS];`
`/* Blackboards in each partition */`
`int`
`nbbs[SYSTEM LIMIT NUMBER OF PARTITIONS];`
`/* Buffers in each partition */`
`int`
`nbuffers[SYSTEM LIMIT NUMBER OF PARTITIONS];`
`/* Semaphores in each partition */`
`int`
`nsems[SYSTEM_LIMIT_NUMBER_OF_PARTITIONS];`
`/* Memory blocks in each partition */`
`int`
`nmblocks[SYSTEM_LIMIT_NUMBER_OF_PARTITIONS];`
`…`
`};`
	These constants are defined in `arinc653.h`.

ARINC 653 Configuring Maximum Processes Number for a Partition	
Earlier versions of LynxOS-178	**LynxOS-178 2.2.4 and later versions**
The maximum number of ARINC 653 processes for a partition was configured in the driver info file.	The maximum number of processes for a partition is limited by the `SYSTEM_LIMIT_NUMBER_OF_PROCESSES` constant. The actual limit of processes which can be created in a partition is configured in VCT file.

`struct arinc653_info {`
`…`
`/* Processes per each partition */`
`int`
`nprocs[SYSTEM_LIMIT_NUMBER_OF_PARTITIONS];`
`…`
`};`
	The constant defining the maximum value is in `arinc653.h`. The actual number of processes allowed to be created in a partition is limited by the VCT configuration.

ARINC 653 Device driver file descriptor

In previous versions of the LynxOS-178 ARINC 653 library the ARINC 653 device file descriptor was used by the library macros to store the open file descriptor number. In newer versions of LynxOS-178 these macros were removed and the file descriptor is for internal use only.

ARINC 653 Device Driver File Descriptor	
Earlier versions of LynxOS-178	**LynxOS-178 2.2.4 and later versions**
Defined in `/usr/include/arinc653/arinc653_private.h`	The device driver file descriptor is not expected to be modified or used explicitly by the user.
`extern int arinc653_devfd;`	

Example of the ARINC 653 Driver Info

Below is an example of the ARINC 653 device info file. This file describes the following resources:

- memory block MBLOCK0 which has size of 4096 bytes and is available for read-write access by partition (VM) 0 and for read-only access by partition (VM) 1. Partition code may obtain the virtual address at which

the block is mapped into its address space by using the `GET_MEMORY_BLOCK_STATUS()` API call.

- module local inter-partition communication channel "CHANNEL-0" which operates in queuing mode and has a maximum message size of 64 bytes. Channels themselves are not visible to partitions.
- source queuing port QS1 available for partition (VM) 0 which has a FIFO size of 1 message.
- destination queuing port QD1 available for partition (VM) 1 which has a FIFO size of 1 message.

This example allows partition 0 to send messages to partition 1 via port QS1 and partition 1 may receive these messages via port QD1. Also, partition 0 may write data to the virtual address of MBLOCK0 in its address space obtained from the `GET_MEMORY_BLOCK_STATUS()` call and partition 1 may see this data at the virtual address in its address space also obtained from the `GET_MEMORY_BLOCK_STATUS()` API call.

```
#include <arinc653info.h>

#if !defined(DLDD) || defined(ARINC653_INFO_MBLOCKS)
static const struct arinc653_info_mblock arinc653_info_mblocks[] = {
 {
  .name = "MBLOCK0",
  .addr = ARINC653_MBLOCK_ADDR_NONE,
  .size = 4096,
  .rwmask = ARINC653_ACCESS_VM_ALLOWED(0),
  .romask = ARINC653_ACCESS_VM_ALLOWED(1),
 }
#if defined(DLDD)
 , {}
#endif
};
#endif

#if !defined(DLDD) || defined(ARINC653_INFO_CHANNELS)
static const struct arinc653_info_channel arinc653_info_channels[] = {
 {
  .name = "CHANNEL-0",
  .max_message_size = 64,
  .max_nb_messages = 1,
  .mode = ARINC653_INFO_CHANNEL_QUEUING,
 }
#if defined(DLDD)
 , {}
#endif
};
#endif

#if !defined(DLDD) || defined(ARINC653_INFO_PORTS)
static const struct arinc653_info_port arinc653_info_ports[] = {
 {
  .vm = 0,
  .name = "QS1",
```

```
  .direction = SOURCE,
  .max_nb_messages = 1,
  .channel = "CHANNEL-0",
 }, {
  .vm = 1,
  .name = "QD1",
  .direction = DESTINATION,
  .max_nb_messages = 1,
  .channel = "CHANNEL-0",
 }
#if defined(DLDD)
 , {}
#endif
};
#endif

#if !defined(DLDD) || defined(ARINC653_INFO_GENERAL)
const struct arinc653_info arinc653_info = {
#if !defined(DLDD)
 .mblocks = arinc653_info_mblocks,
 .nmblocks =
sizeof(arinc653_info_mblocks)/sizeof(arinc653_info_mblocks[0]),
 .channels = arinc653_info_channels,
 .nchannels =
sizeof(arinc653_info_channels)/sizeof(arinc653_info_channels[0]),
 .ports = arinc653_info_ports,
 .nports = sizeof(arinc653_info_ports)/sizeof(arinc653_info_ports[0]),
#endif
};
#endif
```