

C.E.R.N., EP Division, AID Group

RMKI, RFFO Division, Detector Building Group

# ALICE DETECTOR DATA LINK

*ALICE-DDL*

## DDL-RORC Library Version 4.3 User's Manual

Issue: Final  
Revision: 7

Reference: DDL-RORC Lib.4.3  
Created: 23/04/01  
Last modified: 10/08/04

**Prepared by: Ervin Dénes** (erwin.denes@cern.ch)

# 1 DOCUMENT INFORMATION

## 1.1 Abstract

The present document describes the routines and programs, which are necessary to use the RORC (pRORC or D-RORC) card as a data-collecting interface or as a data source unit in DATE [3] or test environments

## 1.2 Document Status Sheet

<b>1. Document Title:</b> DDL-RORC Library Version 4.3			
<b>2. Document Reference Number:</b> DDL-RORC Lib.4.3			
<b>3. Issue</b>	<b>4. Revision</b>	<b>5. Data</b>	<b>6. Reason for change</b>
Draft	1	23 April 2001	Original document
	2	10 May 2001	New naming conventions
Final	3	15 November 2001	New macros and in-line functions. Description of the test programs added.
	4	10 March 2002	Routines changed for physmem
	5	02 May 2002	Minor changes and new test programs added.
	6	21 May 2003	New library for final DDL cards Common routines for pRORC rev. 1 and 2 Inclusion of JTAG routines
	7	10 August 2004	Common library for pRORC and D-RORC New routine and program names

Table 1 Document Status Sheet

## 1.3 Document Change Record

<b>Document Change Record</b>		<b>DCR No.</b>	
		<b>Date</b>	
		<b>Originator</b>	
		<b>Approved by</b>	
<b>1. Document Title</b>		DDL-RORC Library Version 4.3	
<b>2. Document Reference Number</b>		EDMS ID= 340457	
<b>3. Document Issue/Revision Number</b>		Final/7	
<b>4. Page</b>	<b>5. Paragraph</b>	<b>6. Reason for Change</b>	

Table 2 Document Change Record (of changes made since issue ... )

# TABLE OF CONTENTS

<b>1</b>	<b>DOCUMENT INFORMATION</b>	<b>3</b>
1.1	ABSTRACT .....	3
1.2	DOCUMENT STATUS SHEET .....	3
1.3	DOCUMENT CHANGE RECORD .....	3
	<b>TABLE OF CONTENTS</b>	<b>4</b>
	<b>TABLE OF FIGURES</b>	<b>6</b>
<b>2</b>	<b>PREFACE</b>	<b>7</b>
	REFERENCES .....	9
<b>3</b>	<b>DESCRIPTION OF RORC HEADERS AND ROUTINES</b>	<b>11</b>
3.1	.H HEADER FILES.....	11
3.2	THE RORC_DRIVER.....	11
3.3	RORCFINDALL .....	12
3.4	RORCFIND .....	14
3.5	RORCOPENCHANNEL .....	15
3.6	RORCMAPCHANNEL .....	16
3.7	RORCCLOSE .....	17
3.8	RORCARMDATAGENERATOR .....	18
3.9	RORCARMDDL .....	20
3.10	RORCPUSHFREEFIFO .....	21
3.11	RORCCHECKFREEFIFO .....	22
3.12	SETTING RORC PARAMETERS ON/OFF .....	23
3.12.1	RORCLOOPBACKON.....	23
3.12.2	RORCLOOPBACKOFF .....	23
3.12.3	RORCHLTSPLITON .....	24
3.12.4	RORCHLTSPLITOFF .....	24
3.12.5	RORCHLTLCTLOFF.....	25
3.12.6	RORCHLTLCTLON.....	25
3.13	RORCSTARTDATAGENERATOR .....	26
3.14	RORCSTOPDATAGENERATOR.....	27
3.15	RORCSTARTDATA RECEIVER.....	28
3.16	RORCSTOPDATA RECEIVER .....	29
3.17	DDLWRITEDATABLOCK .....	30
3.18	DDLWRITEJTAGBLOCK.....	32
3.19	RORCARMFEIC .....	34
3.20	RORCSTARTTRIGGER.....	36
3.21	RORCSTOPTRIGGER.....	37
3.22	RORCSERIAL .....	38
3.23	RORCREADFW .....	39
3.24	RORCREADRORCSTATUS.....	40
3.25	DDL SERIAL .....	41
3.26	RORCHASDATA.....	42
3.27	RORCCHECKLINK .....	43

---

<b>4</b>	<b>TEST PROGRAMS</b>	<b>45</b>
4.1	RORC_FIND .....	46
4.2	MAILBOX READ AND WRITE PROGRAMS .....	47
4.2.1	PRORC_MBREAD .....	47
4.2.2	PRORC_MBWRITE.....	47
4.2.3	PRORC_MBRESET .....	48
4.2.4	PRORC_EMPTY_MB .....	48
4.3	RORC_RESET .....	49
4.4	SIU_RESET .....	50
4.5	RORC_ID .....	51
4.6	DIU_ID .....	52
4.7	SIU_ID.....	53
4.8	RORC_PUSH_FIFO.....	54
4.9	RORC_POP_FIFO.....	55
4.10	RORC_STATUS .....	56
4.11	RORC_REG.....	57
4.12	DIU_STATUS .....	58
4.13	SIU_STATUS .....	59
4.14	DDL_INIT_LINK .....	60
4.15	RORC_SEND_COMMAND .....	61
4.16	RORC_SEND .....	64
4.17	RORC_SEND_JTAG.....	67
4.18	RORC_RECEIVE .....	68
4.19	FEC2 .....	71
	INSTRUCTIONS USED IN THE SCRIPT FILE:.....	72
4.20	FEIC.MENU .....	76
<b>5</b>	<b>INSTALLATION</b>	<b>79</b>
	<b>NOTES</b>	<b>81</b>

---

## TABLE OF FIGURES

FIGURE 1: THE FREE FIFO – READY FIFO CONCEPT.....	8
FIGURE 2: USE OF THE PRORC LIBRARY ROUTINES.....	8

## 2 PREFACE

Following the request of several detectors to have a PCI-based Read-out Receiver Card (RORC) the development of a simple 33 MHz 32 bit PCI-based RORC (pRORC) has been proposed [1]. For the technical details of the pRORC card see [4]. Later the development of 66 MHz, 64-bit PCI RORC (named D-RORC) has been done aiming higher bandwidth and being capable to send data to the ALICE DAQ system and to the High Level Trigger (HLT) farm. For the technical details of the D-RORC card see [5].

In this paper we describe the C library routines and stand-alone programs necessary for steering and testing both types of RORC cards, during the DAQ data flow, not dealing with HLT functions.

Both RORC cards can be used as a PCI master during data collection and as a stand-alone PCI data source. In both working modes it is necessary to initialize the card and continuously fill the so-called Free FIFO with the addresses of memory blocks (so called pages) where events can be loaded. (Figure 1 shows the data-loading concept.) The Free FIFO is a 64 bit wide FIFO located on the RORC card containing the following information:

- Start address of the memory page where the next data page can be loaded (32 bits).
- Size of the given page (24 bits).
- Index of the Ready FIFO where information should be loaded at the end of the transfer (8 bits).

The filling of the Free FIFO can be done by the inline function `rorcPushFreeFifo()` described in chapter 3.10

The Ready FIFO is a 64-bit wide software area. Its address must be at 2KB boundary. After a successful data transfer it contains:

- The length of the data loaded into the corresponding memory page (32 bits).
- The transfer status (32 bits). If the given page was the last page of an event block then this field contains the Data Transmission Status Word (DTSW), which is added to the event block according the DDL protocol [2]. For other (not event block ending) pages this field is loaded with 0. The DTSW can contain a continuation bit, which signals that the next event block belongs to the same event.

The user of the pRORC card should fill the Ready FIFO record with `-1` before pushing the corresponding Free FIFO entry. The definition of the Ready FIFO can be found in the `rorc_lib.h` header file, while the polling inline function `rorcHasData()` is described in chapter 3.26

The RORC can be used for sending data blocks (e.g. pedestal data) to the detector Front-end Electronics. The corresponding routine is described in chapter 3.17.

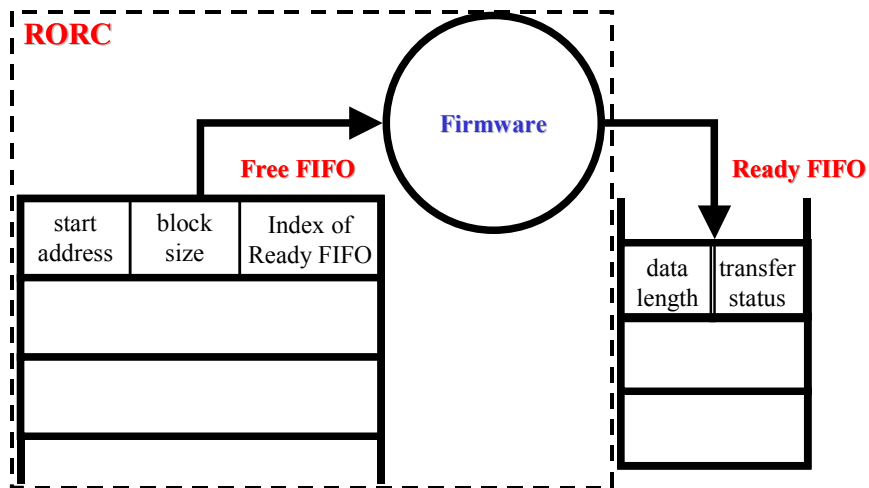


Figure 1: The Free FIFO – Ready FIFO concept

Figure 2 shows a flow chart presenting the use and calling order of the most important routines of the library.

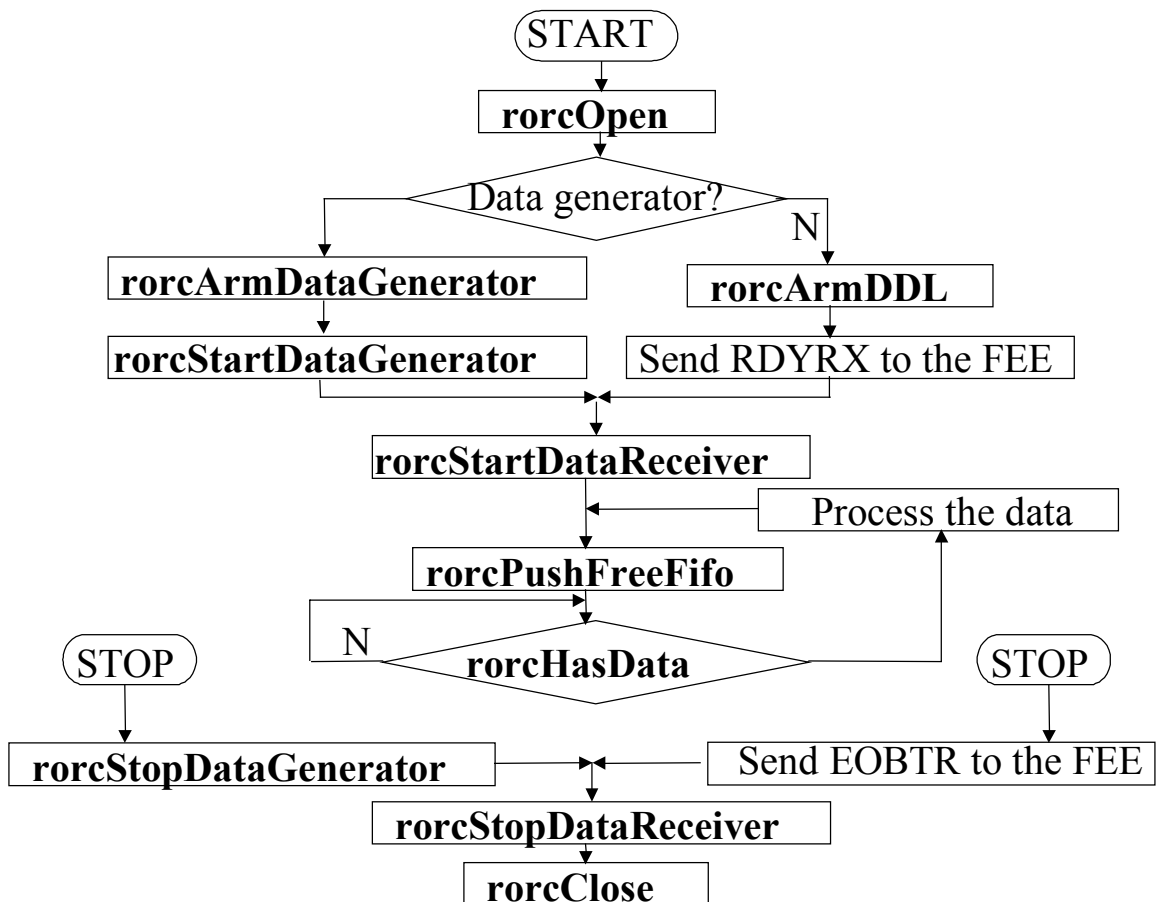


Figure 2: Use of the RORC library routines

## References

1. Proposal for a simple PCI-based RORC  
Presentation on the ALICE week October 2000  
ALICE-AW-2000-13-03  
EDMS id: 115308
2. ALICE DDL Interface Control Document:  
Alice internal note 96-43, CERN  
EDMS id: 112131
3. ALICE DATE V4 User's Guide  
ALICE Internal Note 2002-036, CERN  
EDMS id: 362104
4. PCI-based Readout Receiver Card in the ALICE DAQ System  
Paper presented in the LECC02 Workshop, Colmar, France, September 2002  
<http://cern.ch/ddl/doc/prorcLECC02.pdf>
5. The ALICE Data-Acquisition Read-out Receiver Card  
Paper presented in the LECC04 Workshop, Boston, USA, September 2004
6. The PHYSMEM memory manager module. Part of the ALICE DATE [3]  
software package
7. Front-end Electronics Emulator Interface Card:  
[http://cern.ch/ddl/doc/femu\\_urd.ps](http://cern.ch/ddl/doc/femu_urd.ps)
8. S5935 AMCC chip's specifications:  
<http://www.amcc.com/>



## 3 DESCRIPTION OF RORC HEADERS AND ROUTINES

### 3.1 *.h* header files

Before calling any of the following routines the user must include a header file:

```
#include "rorc_ddl.h"
```

This file contains the necessary definitions for the use of DDL. It has a reference to another header file, which contains the definitions special to the RORC cards:

```
#include "rorc_lib.h"
```

The header files contain the type definition of the structures referred in further descriptions. Also they contain definition of the macros described later on.

### 3.2 The rorc\_driver

The programs and routines described in this documents work under LINUX operating system. Currently we have RORC driver for LINUX kernel version 2.4. Before using the described routines and programs the RORC driver must be loaded. See chapter 5 for details.

### 3.3 rorcFindAll

Find all pRORC cards

#### SYNOPSIS

```
#include <rorc_lib.h>

int rorcFindAll(rorcHwSerial_t *hw,
               rorcHwSerial_t *diu_hw,
               rorcChannelId_t *channel,
               int *rorc_revision,
               int *diu_vers,
               int max_dev)
```

#### DESCRIPTION

The **rorcFindAll()** routine returns the version, serial, revision, minor and channel numbers of all RORC cards plugged in the PC together with the same information regarding the plugged in or embedded DIUs. The routine tries to open all RORC devices and reads from their configuration EPROM the hardware version and serial numbers. It also sends a DDL command to the DIU to find out the DIU version and serial number.

#### Parameters:

<i>hw</i>	pointer to an array of <i>rorcHwSerial_t</i> type structures. The routine loads into these structures the version and serial numbers of RORC cards found. <i>rorcHwSerial_t</i> is defined in the header file <i>rorc_lib.h</i> . Besides the major and minor version and the serial numbers it contains the full string found in the configuration EPROM of the RORC card. If there is no information in the EPROM about the hw version and serial numbers then the routine puts -1 into the structure as version and serial numbers.
<i>diu_hw</i>	pointer to an array of <i>rorcHwSerial_t</i> type structures. The routine loads into these structures the version and serial numbers of the DIU card found. If no DIU is plugged or there is no information in the DIU's EPROM about the hw version and serial numbers then the routine puts -1 into the structure as version and serial numbers.
<i>channel</i>	pointer to an array of <i>rorcChannel_t</i> structures. The routine supplies here the corresponding minor device numbers of the RORC cards and the channel numbers of the DIUs.
<i>rorc_revision</i>	pointer to an array of integers. The routine supplies here the corresponding device revision numbers (1 for pRORC, 2 for D-RORC with connector for DIU and 3 for D-RORC with embedded DIUs) of the RORC cards.
<i>diu_vers</i>	pointer to an array of integers. The routine supplies here the corresponding DIU version number (0 if no DIU, 1 if prototype DIU, 2 if final DIU plugged in version, and 3 if

*max\_dev* embedded DIU) of the DIU on the corresponding DDL channel.  
the size of *hw*, *diu\_hw*, *channel*, *rorc\_revision* and *diu\_vers* arrays.

Return value:

*number of DDL channels found* the number of the DDL channels found on PCI bus or 0.

SEE ALSO

*rorcFind()*, *rorcSerial()*, *rorcOpenChannel()*

## 3.4 rorcFind

Find the specified RORC card

### SYNOPSIS

```
#include <rorc_lib.h>

int rorcFind(int revision, int serial)
```

### DESCRIPTION

The **rorcFind()** routine returns the minor number of a RORC card with the specified revision and serial numbers. The routine tries to open all RORC devices plugged in PC and reads the revision number from their PCI configuration space and the hardware serial number from their configuration EPROM. If it finds the card, it returns its minor number, which can be used for opening and using the card. If several cards have the same specified revision and serial numbers then the routine returns the first one.

### Parameters:

<i>revision</i>	device revision number (1 for pRORC, 2 for D-RORC with connector for DIU and 3 for D-RORC with embedded DIUs) of the RORC card to be found.
<i>serial</i>	the serial number (a 5 digit decimal number) of a RORC card to be found.

### Return value:

<i>minor number</i> $\geq 0$	the minor number of the specified RORC card.
<i>RORC_STATUS_ERROR</i> = -1	the specified RORC was not found. Either such card is not plugged or other process uses it.

### SEE ALSO

*rorcFindAll()*, *rorcSerial()*, *rorcOpenChannel()*

## 3.5 rorcOpenChannel

Arm and reset the DDL channel

### SYNOPSIS

```
#include <rorc_lib.h>

int rorcOpenChannel (rorcHandle_t handle,
                    int rorc_minor, int rorc_channel)
```

### DESCRIPTION

The **rorcOpenChannel()** routine should be called for every DDL channel at the start of a run. The routine checks the existence of the RORC channel. If it finds the channel, it opens it and fills a descriptor. The descriptor address can be used as a handle for every further use of the given channel. The **rorcOpenChannel()** routine resets the RORC device and sends a command the DIU to find out whether there is any DIU plugged and if so, what is the given DIU version (prototype or final). The above information is written into the *handle* structure. If one does not want the RORC be reset, use the **rorcMapChannel()** routine instead.

### Parameters:

<i>handle</i>	address of a RORC descriptor structure. The <i>rorc_handle_t</i> type is a pointer to a <i>rorcDescriptor_t</i> structure, which contains all information about the PCI-based RORC. The structure type is defined in the <i>rorc_lib.h</i> header file. The caller, before calling the <b>rorcOpenChannel()</b> routine has to allocate a descriptor and supply its address to the routine. The routine fills the structure with data necessary at the further calls.
<i>rorc_minor</i>	device file minor number of the RORC card. Multiple RORC cards can be supported (with device file name <code>"/dev/prorcN"</code> , where N is the minor number). The minor numbers start from 0.
<i>rorc_channel</i>	RORC channel number (0 or 1). For pRORC or D-RORC with not embedded DIUs only channel 0 can be used.

### Return value:

<i>RORC_STATUS_OK = 0</i>	no error, channel initialized and <i>handle</i> points to a valid RORC descriptor.
<i>RORC_STATUS_ERROR = -1</i>	the RORC channel couldn't be open. Either no card was found or other process uses it or its PCI memory cannot be mapped.

### SEE ALSO

*rorcMapChannel()*, *rorcClose()*

## 3.6 rorcMapChannel

Arm the pRORC card

### SYNOPSIS

```
#include <rorc_lib.h>

int rorcMapChannel (rorcHandle_t handle,
                   int rorc_minor, rorc_channel)
```

### DESCRIPTION

The **rorcMapChannel()** routine can be called instead of **rorcOpenChannel()** routine when one does not want to reset the open device. It should be called for every DDL channel at the start of a run. The routine checks the existence of the RORC channel. If it finds the channel, it opens it and fills a descriptor. The descriptor address can be used as a handle for every further use of the given channel. The routine does not initialize the RORC card and does not send any command via the DDL channel.

To initialize the DDL components (reset RORC, DIU, SIU and establish the DDL link) use the routine **rorcArmDDL()**.

### Parameters:

<i>handle</i>	address of a RORC descriptor structure. The <i>rorc_handle_t</i> type is a pointer to a <i>rorcDescriptor_t</i> structure, which contains all information about the PCI-based RORC. The structure type is defined in the <i>rorc_lib.h</i> header file. The caller, before calling the <b>rorcMapChannel()</b> routine has to allocate a descriptor and supply its address to the routine. The routine fills the structure with data necessary at the further calls.
<i>rorc_minor</i>	device file minor number of the RORC card. Multiple RORC cards can be supported (with device file name <code>"/dev/prorcN"</code> , where N is the minor number). The minor numbers start from 0.
<i>rorc_channel</i>	RORC channel number (0 or 1). For pRORC or D-RORC with not embedded DIUs only channel 0 can be used.

### Return value:

<i>RORC_STATUS_OK = 0</i>	no error, channel initialized and <i>handle</i> points to a valid RORC descriptor.
<i>RORC_STATUS_ERROR = -1</i>	the RORC channel couldn't be open. Either no card was found or other process uses it or its PCI memory cannot be mapped.

### SEE ALSO

*rorcOpenChannel()*, *rorcArmDDL()*, *rorcClose()*

## 3.7 rorcClose

Close the RORC channel.

### SYNOPSIS

```
#include <rorc_lib.h>

int rorcClose(rorcHandle_t handle)
```

### DESCRIPTION

The **rorcClose()** routine should be called for every DDL channel at the end of a run. The routine closes all resources set up by a previous call of routine **rorcOpenChannel()** or **rorcMapChannel()**.

### Parameters:

*handle* address of the RORC descriptor. When the routine returns the *handle* will point to an invalid descriptor.

### Return value:

*RORC\_STATUS\_OK = 0* no error, channel closed

### SEE ALSO

*rorcOpenChannel()*, *rorcMapChannel()*

## 3.8 rorcArmDataGenerator

Initialize RORC's Data generator.

### SYNOPSIS

```
#include <rorc_lib.h>

int
rorcArmDataGenerator(rorcHandle_t      handle,
                    __u32               initEventNumber,
                    __u32               initDataWord,
                    int                 dataPattern,
                    int                 eventLen,
                    int                 seed,
                    int                 *rounded_length)
```

### DESCRIPTION

The **rorcArmDataGenerator()** routine should be called for every DDL channel where the RORC card will be used as data generator. The routine can be called after the call of the **rorcOpenChannel()** and before the call of the **rorcStartDataGenerator()** routines. It defines all the parameters needed for data generation. If **rorcStartDataGenerator()** is called without calling **rorcArmDataGenerator()** then the data generator will use unpredictable values.

### Parameters:

<i>handle</i>	address of the RORC descriptor.
<i>initEventNumber</i>	each event starts with the serial number of the given event (event count). This parameter defines the starting value of it.
<i>initDataWord</i>	the first data word of the event (after the event count). It is used only for some of the test patterns. Note: for D-RORC, if <i>seed</i> is not RORC_DG_NO_RANDOM_LEN, the first data word of each event is 0.
<i>dataPattern</i>	an integer between 1 and 7: RORC_DG_CONST: all data words are <i>initDataWord</i> . RORC_DG_ALTER: alternating pattern, starting from <i>initDataWord</i> RORC_DG_FLY0: flying 0 starting from 0xffffffffe RORC_DG_FLY1: flying 1 starting from 1 RORC_DG_INCR: incrementing data starting from <i>initDataWord</i> RORC_DG_DECR: decrementing data starting from <i>initDataWord</i> RORC_DG_RANDOM: random data

<i>eventLen</i>	length (from 1 to $2^{19}-1$ ) of the generated events in 32 bit words, including the event count. Important: because of the special features of the random number generation if random length is used ( <i>seed</i> is not equal to RORC_DG_NO_RANDOM_LEN), the minimum generated event length is 1, and the maximum value of the length will be <i>eventLen</i> rounded down to the nearest integer of power of 2.
<i>seed</i>	defines the seed value for random data length. If given, the event lengths will vary between 1 and <i>eventLen</i> . Using the value RORC_DG_NO_RANDOM_LEN no random length will be generated.
<i>rounded_length</i>	it us an output parameter: in case of random length generation the maximum event length is rounded to the nearest integer of power of two. The routine transfers this value to RORC as the maximum length and returns it to the user in this variable.

Return value:

*RORC\_STATUS\_OK* = 0           no error, data generator initialised  
*RORC\_INVALID\_PARAM* = -2    error: some of the parameters out of range.

SEE ALSO

*rorcOpenChannel()*, *pRrorcStartDataGenerator()*

## 3.9 rorcArmDDL

Initialize the RORC card and the DDL link

### SYNOPSIS

```
#include <rorc_ddl.h>

int rorcArmDDL(rorcHandle_t handle,
               int options)
```

### DESCRIPTION

The **rorcArmDDL()** routine should be called for every DDL channel where the RORC card will be not used as data generator but the data should arrive via DDL from the Front-end Electronics (FEE). The routine (on user request) tries to establish the link between the two end-points of the DDL: the DIU and SIU (only necessary with prototype DDL cards). According to the user request the routine resets the Free FIFO, the other parts of the RORC, the DIU or the FEE units. The RORC, DIU and SIU reset will be done before establishing the link. If several reset requests are “OR-d”, the program first resets the SIU, then establishes the link, then resets the DIU and at last resets the RORC. Resetting the RORC card means to empty all its FIFOs, including the Free FIFO, and then put all programmable features to their reset values. Resetting the DIU or SIU means to cut the DDL link (if it was on before the call of the routine). In the case of prototype version of the DDL cards, after the link cut, the link has to be re-established by calling **rorcArmDDL()** with *RORC\_LINK\_UP* parameter. In case of the final DDL cards, the link sets up automatically.

### Parameters:

<i>handle</i>	address of the RORC descriptor.
<i>options</i>	the following values can be “OR-d”:
	<ul style="list-style-type: none"><li>• <i>RORC_RESET_FF</i> reset Free FIFO</li><li>• <i>RORC_RESET_RORC</i> reset RORC</li><li>• <i>RORC_RESET_DIU</i> reset DIU</li><li>• <i>RORC_RESET_SIU</i> reset SIU</li><li>• <i>RORC_LINK_UP</i> establish the DDL link</li></ul>

### Return value:

<i>RORC_STATUS_OK</i> = 0	no error, requested task done.
<i>RORC_LINK_NOT_ON</i> = -4	link initialization did not succeed
<i>RORC_CMD_NOT_ALLOWED</i> = -8	routine called with not permitted option
<i>RORC_NOT_ACCEPTED</i> = -16	unsuccessful SIU reset

### SEE ALSO

*rorcArmDataGenerator()*, *rorc\_reset*

## 3.10 rorcPushFreeFifo

Push one entry into RORC's Free FIFO.

### SYNOPSIS

```
#include <rorc_lib.h>

rorcPushFreeFifo(rorcHandle_t      handle,
                 rorcMemAddress_t  blockAddress,
                 __u32             blockLength,
                 int               readyFifoIndex)
```

### DESCRIPTION

The **rorcPushFreeFifo()** is an in-line function what should be called when the user has a free data page and wants to load its parameters into the Free FIFO. It loads the parameters directly into the RORC registers.

The function does not check the range of the parameters: it masks them for the given range. It neither checks the Free FIFO status. If the Free FIFO is overflowed then the new parameters will not be loaded. The caller can check this situation using **rorcCheckFreeFifo()**.

### Parameters:

<i>handle</i>	address of the RORC descriptor.
<i>blockAddress</i>	physical address of the next free page in <i>physmem</i> memory.
<i>blockLength</i>	length of the next free page in byte (24 bit).
<i>readyFifoIndex</i>	index of the ready FIFO, where the "data arrived" flag has to be put to (8 bit).

### SEE ALSO

*rorcCheckFreeFifo()*

### 3.11 rorcCheckFreeFifo

Returns the status of the pRORC's Free FIFO.

#### SYNOPSIS

```
#include <rorc_lib.h>

int rorcCheckFreeFifo(rorcHandle_t handle)
```

#### DESCRIPTION

The **rorcCheckFreeFifo()** should be called when the caller wants to know how many FIFO entries are in the Free FIFO. Using pRORC device, it returns the number of entries in "8 entry" units (i.e. 0 means 1 to 8 entries, 1 means 9 to 16 entries, etc.). FIFO full and FIFO empty statuses are signaled. In the case of D-RORC, the routine only signals if the Free FIFO is not empty

#### Parameters:

*handle* address of the RORC descriptor.

#### Return value:

*Value between 1 and 15 (=x)* the number of not empty Free FIFO entries are between.  $8x+1$  and  $8x+8$

Return value	# of not empty FF entries
0	Between 1 and 8 words
1	Between 9 and 16 words
2	Between 17 and 24 words
3	Between 25 and 32 words
.....	.....
13	Between 105 and 112 words
14	Between 113 and 120 words
15	Between 121 and 128 words

*RORC\_STATUS\_OK = 0*

*RORC\_FF\_EMPTY = -256*

*RORC\_FF\_FULL = -128*

Free FIFO is not empty (and not full).

Free FIFO is empty.

error: Free FIFO full.

#### SEE ALSO

*rorcPushFreeFifo()*

## 3.12 Setting RORC parameters on/off

The following routines can be used to set RORC internal control parameters on or off.

### 3.12.1 *rorcLoopBackOn*

#### SYNOPSIS

```
#include <rorc_lib.h>

int rorcLoopBackOn(rorcHandle_t handle)
```

#### DESCRIPTION

The **rorcLoopBackOn()** routine should to be called when the user wants to set the operational control parameter “Internal Loop-back” bit. If this control bit is on the data generated by the RORC’s Data Generator will be sent back to the RORC as if it had arrived from the link.

This conditions can be reset by the routine **rorcLoopBackOff()**.

#### Parameters:

*handle* address of the RORC descriptor.

#### Return value:

*RORC\_STATUS\_OK = 0* no error

### 3.12.2 *rorcLoopBackOff*

#### SYNOPSIS

```
#include <rorc_lib.h>

int rorcLoopBackOff(rorcHandle_t handle)
```

#### DESCRIPTION

The **rorcLoopBackOff()** routine should to be called when the user wants to reset the operational control parameter “Internal Loop-back” bit

This condition is automatically set after RORC reset.

#### Parameters:

*handle* address of the RORC descriptor.

#### Return value:

*RORC\_STATUS\_OK = 0* no error

### 3.12.3 *rorcHltSplitOn*

#### SYNOPSIS

```
#include <rorc_lib.h>

int rorcHltSplitOn(rorcHandle_t handle)
```

#### DESCRIPTION

The D-RORC card with 2 integrated DIU can be used in “split mode”. It means that the data arriving in one channel can be transferred to the other channel. The **rorcHltSplitOn()** routine should to be called when the user wants the give channel to be used as the output channel.

This conditions can be reset by the routine **rorcHltSplitOff()**.

#### Parameters:

*handle* address of the RORC descriptor.

#### Return value:

*RORC\_STATUS\_OK* = 0 no error  
*RORC\_CMD\_NOT\_ALLOWED* = -8 routine cannot be called for pRORC or D-RORC with not integrated DIU.

### 3.12.4 *rorcHltSplitOff*

#### SYNOPSIS

```
#include <rorc_lib.h>

int rorcHltSplitOff(rorcHandle_t handle)
```

#### DESCRIPTION

The **rorcHltSplitOff()** routine should to be called when the user wants to switch off the data sending for the given channel.

This condition is automatically set after RORC reset.

#### Parameters:

*handle* address of the RORC descriptor.

#### Return value:

*RORC\_STATUS\_OK* = 0 no error  
*RORC\_CMD\_NOT\_ALLOWED* = -8 routine cannot be called for pRORC or D-RORC with not integrated DIU.

### 3.12.5 *rorcHltFlctlOn*

#### SYNOPSIS

```
#include <rorc_lib.h>

int rorcHltFlctlOn(rorcHandle_t handle)
```

#### DESCRIPTION

The D-RORC card with 2 integrated DIU can be used in “split mode”. It means that the data arriving in one channel can be transferred to the other channel. The **rorcHltFlctlOn()** routine should to be called when the given channel is used as the output channel (**rorcHltSplitOn()** is called or will be called) and the user wants the flow control from the receiver side (probably the HLT farm) be taken into account.

This conditions can be reset by the routine **rorcHltFlctlOff()**.

#### Parameters:

*handle* address of the RORC descriptor.

#### Return value:

*RORC\_STATUS\_OK* = 0 no error  
*RORC\_CMD\_NOT\_ALLOWED* = -8 routine cannot be called for pRORC or D-RORC with not integrated DIU.

### 3.12.6 *rorcHltFlctlOff*

#### SYNOPSIS

```
#include <rorc_lib.h>

int rorcHltFlctlOff(rorcHandle_t handle)
```

#### DESCRIPTION

The **rorcHltFlctlOff()** routine should to be called when the given channel is used as the output channel (**rorcHltSplitOn()** is called or will be called) and the user wants the flow control from the receiver side (probably the HLT farm) NOT be taken into account.

This condition is automatically set after RORC reset.

#### Parameters:

*handle* address of the RORC descriptor.

#### Return value:

*RORC\_STATUS\_OK* = 0 no error  
*RORC\_CMD\_NOT\_ALLOWED* = -8 routine cannot be called for pRORC or D-RORC with not integrated DIU.

### 3.13 rorcStartDataGenerator

Set RORC to start sending generated data

#### SYNOPSIS

```
#include <rorc_lib.h>

int rorcStartDataGenerator(rorcHandle_t handle,
                           __u32 maxLoop)
```

#### DESCRIPTION

The **rorcStartDataGenerator()** routine should to be called when the user wants to receive generated data. Normally the Data Generator sends the data out to the DDL link. If the user wants the simulated data arriving into the PC then the RORC has to set to loop-back mode before starting the Generator. This can be done by the routine **rorcLoopBackOn()**.

Data will arrive only when data receiver is started by calling the **rorcStartDataReceiver()** routine, and RORC's Free FIFO is not empty. Features of the generated data (data pattern, event length, event frequency) can be defined by a previous call of **rorcArmDataGenerator()**. To stop the data generator (in the case of infinite number of events) call the routine **rorcStopDataGenerator()**.

#### Parameters:

<i>handle</i>	address of the RORC descriptor.
<i>maxLoop</i>	number of events to be generated. Possible values are from 1 to $2^{32}-1$ . or RORC_DG_INFINIT_EVENT (infinite number of events).

#### Return value:

<i>RORC_STATUS_OK = 0</i>	no error, data generator started
---------------------------	----------------------------------

#### SEE ALSO

*rorcArmDataGenerator()*, *rorcLoopBackOn()*, *rorcStartDataReceiver()*  
*rorcStopDataGenerator()*

## 3.14 rorcStopDataGenerator

Stop sending generated data

### SYNOPSIS

```
#include <rorc_lib.h>

int rorcStopDataGenerator(rorcHandle_t handle)
```

### DESCRIPTION

The **rorcStopDataGenerator()** routine should to be called when the user wants to stop receiving generated data. The data generator stops sending events when the number of events preset in **rorcStartDataGenerator()** is reached. However **rorcDataStopGenerator()** has to be called to set the RORC card into normal state. If data sending is going on when this routine is called then the current event will be finished and no more data will be sent. (If the transfer is stuck, one has to reset the RORC card.)

### Parameters:

*handle*                                      address of the RORC descriptor.

### Return value:

*RORC\_STATUS\_OK = 0*                                      no error, data generator stopped

### SEE ALSO

*rorcStartDataGenerator()*

### 3.15 rorcStartDataReceiver

Set the DDL channel to data collecting state

#### SYNOPSIS

```
#include <rorc_lib.h>

int rorcStartDataReceiver(rorcHandle_t    handle,
                          unsigned long    readyFifoBaseAddress)
```

#### DESCRIPTION

The **rorcStartDataReceiver()** routine should to be called when the user wants to receive data via the DDL channel.

#### Parameters:

*handle* address of the RORC descriptor.  
*readyFifoBaseAddress* the physical memory address of the Ready FIFO. It must be a multiple of 2K, i.e. the lower 11 bits of the Ready FIFO address must be 0.

#### Return value:

*RORC\_STATUS\_OK = 0* no error, data collection started

#### SEE ALSO

*rorcStopDataReceiver()*

## 3.16 rorcStopDataReceiver

Stop data collecting

### SYNOPSIS

```
#include <rorc_lib.h>

int rorcStopDataReceiver(rorcHandle_t handle)
```

### DESCRIPTION

The **rorcStopDataReceiver()** routine should to be called when the user wants to stop receiving data via the DDL channel.

### Parameters:

*handle*                                      address of the RORC descriptor.

### Return value

*RORC\_STATUS\_OK = 0*                                      no error, data collection stopped

### SEE ALSO

*rorcStartDataReceiver()*

## 3.17 ddiWriteDataBlock

Send a data block to the FEE

### SYNOPSIS

```
#include <rorc_ddl.h>

int ddiWriteDataBlock(rorcHandle_t    handle,
                     unsigned long    bufferPhysAddress,
                     unsigned long    bufferWordLength,
                     unsigned long    returnPhysAddress,
                     volatile unsigned long *returnAddr,
                     __u32            feeAddress,
                     int              timeout,
                     stword_t        *stw,
                     int              *n_reply,
                     int              *step)
```

### DESCRIPTION

The **ddiWriteDataBlock()** routine should to be called when the user wants to send a data block to the FEE via the DDL channel. The routine fulfils the following 3 steps:

1. Sends a Start Block Write (STBWR) command to the FEE, specifying the front-end address where the data is intended.
2. Sends the data block.
3. Sends an End Of Block Transfer (EOBTR) command to the SIU.

If error occurs in any of the above steps, the routine returns with the error code, step number and the received reply from the FEE or SIU.

### Parameters:

<i>handle</i>	address of the RORC descriptor.
<i>bufferPhysAddress</i>	the physical memory address of the data.
<i>bufferWordLength</i>	the length of the data block in 32 bit words. The maximum length is 512 K words – 1 word.
<i>returnPhysAddress</i>	the physical memory address of a word where the number of transferred word will be put when the transfer had finished. When using D-RORC the address must be 2K aligned, i.e. its lower 11 bits must be 0. The routine writes -1 on this address before sending the data and to polls this address while the transfer is done.
<i>returnAddress</i>	a pointer to the virtual address of the above physical memory.
<i>feeAddress</i>	a maximum 19 bit long value which will be sent to the FEE in the STBWR command.
<i>timeout</i>	the number of waiting cycles for receiving the SIU reply.
<i>stw</i>	pointer to an array of status word structures where the routine returns the received statuses.

*n\_reply* pointer to a variable where the routine returns the number of received statuses.  
*step* pointer to a variable where the routine returns the step number where the routine returned from.

Return value:

<i>RORC_STATUS_OK</i> = 0	no error, data collection started
<i>RORC_LINK_NOT_ON</i> = -4	error: the link is down
<i>RORC_TIMEOUT</i> = -64	error: command can not be sent in time <i>timeout</i>
<i>RORC_NOT_ABLE</i> = -32	error: the previous download was not finished in time <i>timeout</i>
<i>RORC_TOO_MANY_REPLY</i> = -512	error: too many replies arrived
<i>RORC_NOT_ENOUGH_REPLY</i> = -1024	error: less reply arrived then expected in time <i>timeout</i>

SEE ALSO

*ddlWriteJTAGBlock()*

## 3.18 ddiWriteJTAGBlock

Send a JTAG data block to the FEE and wait for the response

### SYNOPSIS

```
#include <rorc_ddl.h>

int ddiWriteJTAGBlock(rorcHandle_t    handle,
                     unsigned long    bufferPhysAddress,
                     unsigned long    bufferWordLength,
                     unsigned long    returnPhysAddress,
                     volatile unsigned long *returnAddr,
                     __u32            feeAddress,
                     int              timeout,
                     stword_t        *stw,
                     int              *n_reply,
                     int              *step)
```

### DESCRIPTION

The **ddiWriteJTAGBlock()** routine should to be called when the user wants to send a data block to the FEE via the DDL channel. The routine fulfils the following 4 steps:

1. Sends a Start JATG Write (JSTART) command to the SIU, specifying the SIU address where the data is intended.
2. Sends the data.
3. Waits for JTAG response (TDO bits).
4. Sends an End Of JTAG Transfer (EOJTR) command to the SIU.

If error occurs in any of the above steps, the routine returns with the error code, step number and the received reply from the SIU. For receiving the JTAG response the routine **rorcStartDataReceiver()** must be called before the present routine.

Note: The JTAG function is supported only by the pRORC card, not by the D-RORC card.

### Parameters:

<i>handle</i>	address of the RORC descriptor.
<i>bufferPhysAddress</i>	the physical memory address of the JTAG data.
<i>bufferWordLength</i>	the length of the data block in 32 bit words. The maximum length is 100 words.
<i>returnPhysAddress</i>	the physical memory address of a memory area where the response will be put when the transfer had finished. The routine writes <i>-1</i> on these addresses before sending the data and to polls the last address while the transfer is done.
<i>returnAddress</i>	a pointer to the virtual address of the above physical memory.
<i>feeAddress</i>	a maximum 19 bit long value which will be sent to the SIU in the JSTART command. It could be 0.
<i>timeout</i>	the number of waiting cycles for receiving the JTAG reply.

*stw* pointer to an array of status word structures where the routine returns the received statuses.

*n\_reply* pointer to a variable where the routine returns the number of received statuses.

*step* pointer to a variable where the routine returns the step number where the routine returned from.

Return value:

<i>RORC_STATUS_OK = 0</i>	no error, data collection started
<i>RORC_LINK_NOT_ON = -4</i>	error: the link is down
<i>RORC_TIMEOUT = -64</i>	error: command can not be sent in time <i>timeout</i>
<i>RORC_TOO_MANY_REPLY = -512</i>	error: too many replies arrived
<i>RORC_NOT_ENOUGH_REPLY -1024</i>	error: less reply arrived then expected in time <i>timeout</i>

SEE ALSO

*ddlWriteDataBlock()*, *rorcStartDataReceiver()*

## 3.19 rorcArmFeic

Initialize the Front-End Emulator Interface Card (FEIC)

### SYNOPSIS

```
#include <rorc_ddl.h>

int rorcArmFeic(rorcHandle_t    handle,
                int              pattern,
                int              evlen,
                int              trig,
                int              flctrl,
                int              trdis,
                int              seed,
                int              time,
                int              *rounded_length)
```

### DESCRIPTION

The **rorcArmFeic()** routine sends commands to the FEIC card [7] and sets its working modes. After setting, it reads back and checks the working modes. The DDL link must be initialised before the call of **rorcArmFeic()** routine.

### Parameters:

<i>handle</i>	address of the RORC descriptor
<i>pattern</i>	the pattern code of the events to be generated by the FEIC. 1: external pattern generator 2: alternating pattern 3: flying 0 4: flying 1 5: incrementing data 6: decrementing data
<i>evlen</i>	event length or in case of random length generation the maximum event length: The FEIC accepts only event length between 16 words and 256Mwords in power of 2 steps (16, 32, ....., 256M). The routine modifies the value received by the user to a lower value, which meets the above requirements. The modified value will be returned to the user in the <i>rounded_length</i> parameter.
<i>trig</i>	trigger mode: 1: external push button 2: external trigger 3: 16 clocks gap after each event 4: 128 clocks gap after each event 5: every 10 ms 6: every 100 ms
<i>flctrl</i>	flow control simulation: 1: flow control after each received word 2: flow control after receiving 128 words 3: flow control after receiving 16 Kword

<i>trdis</i>	transmission disable: 1: after each word sent 2: after transmitting 128 words 3: after transmitting 16 Kword.
<i>seed</i>	defines the seed value for random data length. If given, the event lengths will vary between <i>l</i> and <i>evlen</i> . Using the value RORC_DG_NO_RANDOM_LEN no random length will be generated.
<i>time</i>	number of mail status checks for SIU response. If > 0 then the routine tests if the commands can be sent and wait as many cycles as specified if necessary.
<i>rounded_length</i>	the FEIC card accepts only event length rounded to the nearest lower power of two integer. The routine returns this value to the user in this variable.

Return values:

<i>RORC_STATUS_OK</i> = 0	the initialising commands was sent successfully and FEIC is set.
<i>RORC_STATUS_ERROR</i> = -1	error: reading back the FEIC's working modes the program found a mode not matching the initialised value.
<i>RORC_INVALID_PARAM</i> = -2	error: some of the parameters out of range.
<i>RORC_LINK_NOT_ON</i> = -4	error: the RORC was not able to initialise the FEIC because the DDL link was not on.
<i>RORC_TIMEOUT</i> = -64	error: the RORC was not able to initialise the FEIC because the FIEC did not reply properly during time-out <i>time</i> .

SEE ALSO

*prorcArmDataGenerator() feic.menu*

## 3.20 rorcStartTrigger

Send a RDYRX command to the FEE

### SYNOPSIS

```
#include <rorc_ddl.h>
```

```
int rorcStartTrigger(rorcHandle_t    handle,  
                    int             timeout,  
                    stword_t       stword)
```

### DESCRIPTION

The **rorcStartTrigger()** routine sends a RDYRX command to the FEE.

### Parameters:

<i>handle</i>	address of the RORC descriptor
<i>timeout</i>	the number of waiting cycles for receiving the FEE reply
<i>stword</i>	the FEE reply: a DDL status word. <i>stword.stw</i> contains the full reply. For the details of a status word, see [2].

### Return values:

<i>RORC_STATUS_OK = 0</i>	the RDYRX command was sent successfully.
<i>RORC_STATUS_ERROR = -1</i>	the RORC was not able to send the command.
<i>RORC_LINK_NOT_ON = -4</i>	the link is down; the RORC is not able to send the command.
<i>RORC_NOT_ACCEPTED = -16</i>	No reply arrived from SIU in time-out.

### SEE ALSO

*rorcStopTrigger()*

## 3.21 rorcStopTrigger

Send an EOBTR command to the FEE

### SYNOPSIS

```
#include <rorc_ddl.h>
```

```
int rorcStopTrigger(rorcHandle_t    handle,  
                   int             timeout,  
                   stword_t       stword)
```

### DESCRIPTION

The **rorcStopTrigger()** routine sends an EOBTR command to the FEE.

### Parameters:

<i>handle</i>	address of the RORC descriptor
<i>timeout</i>	the number of waiting cycles for receiving the FEE reply
<i>stword</i>	the FEE reply: a DDL status word. <i>stword.stw</i> contains the full reply. For the details of a status word, see [2].

### Return values:

<i>RORC_STATUS_OK = 0</i>	the EOBTR command was sent successfully.
<i>RORC_STATUS_ERROR = -1</i>	the RORC was not able to send the command.
<i>RORC_LINK_NOT_ON = -4</i>	the link is down; the RORC is not able to send the command.
<i>RORC_NOT_ACCEPTED = -16</i>	no reply arrived from SIU in time-out.

### SEE ALSO

*rorcStartTrigger()*

## 3.22 rorcSerial

Reads RORC's version and serial numbers

### SYNOPSIS

```
#include <rorc_lib.h>

rorcHwSerial_t rorcSerial(rorcHandle_t handle)
```

### DESCRIPTION

The **rorcSerial()** routine reads from the card's configuration EPROM its hardware version and serial numbers.

### Parameters:

*handle*                                      address of the RORC descriptor

### Return value:

*structure rorcHwSerial*                      The routine loads into this structure the version and serial numbers of the RORC card. *rorcHwSerial\_t* is defined in the header file *rorc\_lib.h*. Besides the major and minor version and the serial numbers it contains the full string found in the configuration EPROM of the RORC card. If there is no information in the EPROM about the hw version and serial numbers then the routine puts -1 into the structure as version and serial numbers. The library routine **rorcInterpretSerial(hw)** interprets the relevant fields and print the interpretation to the standard output.

### SEE ALSO

*rorcFind(), rorcFindAll(), rorcReadFw(), dllSerial()*

### 3.23 rorcReadFw

Read RORC's software identification word

#### SYNOPSIS

```
#include <rorc_lib.h>

int rorcReadFw(rorcHandle_t handl)
```

#### DESCRIPTION

The **rorcReadFw()** function returns the RORC's firmware identification word.

#### Parameters:

<i>handle</i>	address of the RORC descriptor.
<i>hw</i>	hardware identifications word's address.
<i>fw</i>	firmware identifications word's address.

#### Return value

The returned word contains the RORC's firmware identification in the following format:

bits 0-4:	day
bits 5-8:	month
bits 9-12:	year form 2000
bits 13-24	version number of the pRORC card's firmware
bits 25-31	Free FIFO size of the card in 64 units.

The library routine **rorcInterpretFw(fw)** interprets the relevant fields and print the interpretation to the standard output. The inline function **rorcFFSize(fw)** returns the number of Free FIFO entries of the card while **rorcFWVersMajor(fw)** and **rorcFWVersMinor(fw)** returns the major and minor version numbers of the card's firmware.

#### SEE ALSO

*rorcSerial()*

## 3.24 rorcReadRorcStatus

Read pRORC status

### SYNOPSIS

```
#include <rorc_lib.h>

int rorcReadRorcStatus(rorcHandle_t      handle,
                      rorcStatus_t      *status)
```

### DESCRIPTION

The **rorcReadRorcStatus()** function fills a structure (defined in *rorc\_lib.h* header file) containing information about RORC status and errors, such as: working mode of the RORC, Free FIFO status, link status, flow control status, etc. Before calling the **rorcReadRorcStatus()** routine the caller has to allocate a *rorcStatus\_t* structure and supply its address to the routine. The routine fills this structure.

The *rorcStatus\_t* structure contains three members:

- *ccsr*, the copy of the RORC's Operation Control and Status Register,
- *cerr*, the copy of the RORC's Error Register,
- *cdgs*, the copy of the RORC's Data Generator Status Register,

The meaning of the status and error bits can be found in *rorc\_lib.h* header file. The library routines **rorcInterpretStatus(ccsr)** and **rorcInterpretError(cerr)** interpret the relevant register bits and print the interpretation to the standard output.

### Parameters:

<i>handle</i>	address of the pRORC descriptor.
<i>status</i>	address of a <i>rorcStatus_t</i> type structure. The routine fills into this structure the RORC status information.

### Return value

<i>RORC_STATUS_OK = 0</i>	no error, RORC status structure filled
---------------------------	--

## 3.25 ddlSerial

Reads the version and serial numbers of the DIU or SIU card

### SYNOPSIS

```
#include <rorc_ddl.h>

rorcHwSerial_t ddlSerial(rorcHandle_t handle,
                        int destination,
                        int timeout)
```

### DESCRIPTION

The **ddlSerial** routine send command to the DIU or SIU requesting its hardware version and serial numbers. The routine works only for plugged DIU and DDL cards of final version.

### Parameters:

<i>handle</i>	address of the RORC descriptor
<i>destination</i>	DIU or SIU
<i>timeout</i>	the number of waiting cycles for receiving the DDL card's reply

### Return value:

<i>structure rorcHwSerial</i>	The routine loads into this structure the version and serial numbers of the DDL (DIU or SIU) card. <i>rorcHwSerial_t</i> is defined in the header file <i>rorc_lib.h</i> . Besides the major and minor version and the serial numbers it contains the full string received from the card. If there is no information received then the routine puts -1 into the structure as version and serial numbers (this is the case for the prototype version DDL cards or integrated DIUs).
-------------------------------	--

### SEE ALSO

*rorcSerial()*

## 3.26 rorcHasData

Check the Ready FIFO for new data block.

### SYNOPSIS

```
#include <rorc_lib.h>

int rorcHasData(rorcReadyFifo_t readyFifoBaseAddr,
               int readyFifoIndex)
```

### DESCRIPTION

The calling program has to specify the Ready FIFO base address and index. The routine polls the Ready FIFO entry and returns:

*RORC\_DATA\_BLOCK\_NOT\_ARRIVED* or  
*RORC\_NOT\_END\_OF\_EVENT\_ARRIVED* or  
*RORC\_LAST\_BLOCK\_OF\_EVENT\_ARRIVED*.

Since this routine is an in-line function, it makes the minimum work possible. It does not return values from the Ready FIFO. The caller can read the block length and the status from the FIFO.

The routine only returns the information of block arrival. The memory address of the given block (and the other blocks of the same event) has to be known by the caller.

### Parameters:

*readyFifoBaseAddr* base address of the Ready FIFO  
*readyFifoIndex* index of the Ready FIFO where the checking has to be done

### Return value:

*RORC\_DATA\_BLOCK\_NOT\_ARRIVED* = 0 no data block (data page) arrived (Ready FIFO status = -1)  
*RORC\_NOT\_END\_OF\_EVENT\_ARRIVED* = 1 data block (data page) arrived but not end-of-event block (status 0)  
*RORC\_LAST\_BLOCK\_OF\_EVENT\_ARRIVED* = 2 end-of-event data block arrived (status = DTSTW)  
If the continuation bit (bit 8) is set, the event will continue.

## 3.27 rorcCheckLink

Check the DDL link status.

### SYNOPSIS

```
#include <rorc_lib.h>

int rorcCheckLink(rorcHandle_t handle)
```

### DESCRIPTION

The **rorcCheckLink()** macro function checks a status word of the RORC card which reflects the link status.

### Parameters:

*handle* address of the pRORC descriptor.

### Return value

*RORC\_STATUS\_OK* = 0 the DDL link is on  
*RORC\_LINK\_NOT\_ON* = -4 the DDL link is not on

### SEE ALSO

*rorcReadRorcStatus()*



## 4 TEST PROGRAMS

The following programs use the RORC library routines described in the previous chapter. The executable programs can be used for testing the RORC card: read and write its registers, establish the DDL link, test the Data Generator function and receive and check data sent by the Front-End Emulator [7] or other data source.

We use the following **notations** for program options (parameter list):

- Optional parameters are in square brackets “[ ]”
- Obligatory part of the parameters are in twirl brackets “{ }”
- Mutually exclusive parts are separated by a vertical line “|”
- Parts in < > brackets has to be replaced by an actual value.

## 4.1 rorc\_find

Find and list all RORC cards not under use

### SYNOPSIS

```
rorc_find
```

### DESCRIPTION

The **rorc\_find** program lists the type and hardware identification text of all RORC cards plugged in the PC, together with the same information for the eventually plugged or integrated DIUs. The program tries to open all RORC devices and reads from their configuration EPROM the hardware identification.

The type of the RORC device could be pRORC, D-RORC and integrated D-RORC.

The type of the DIU could be old (prototype), new (final), embedded (in case of integrated D-RORC) DIU or “no DIU plugged”.

### SEE ALSO

*rorcFind(), rorcFindAll(), rorcSerial()*

## 4.2 Mailbox read and write programs

The following 4 programs can be used only for pRORC devices.

### 4.2.1 *prorc\_mbread*

Read the pRORC card's mailboxes.

#### SYNOPSIS

```
mbread [-{M|m} <pRORC_minor>]
```

#### DESCRIPTION

The **prorc\_mbread** program reads the 4 incoming and 4 outgoing mailboxes of the specified pRORC card's PCI bridge chip. The program displays the values on the standard output together with the contents of the chip's Mailbox Empty-Full (MBEF) register [8]. The non-zero bits of MBEF shows which mailbox not empty before the read execution.

#### Parameters:

<i>pRORC_minor</i>	device file minor number of the pRORC card. Multiple pRORC cards can be supported (with device file name <code>"/dev/prorcN"</code> , where N is the minor number). The minor numbers start from 0. The default value is 0.
--------------------	---

### 4.2.2 *prorc\_mbwrite*

Write the pRORC card's mailboxes.

#### SYNOPSIS

```
mbread [-{M|m} <pRORC_minor>] -{N|n} <mailbox_number>  
-{V|v} <hex_value>
```

#### DESCRIPTION

The **prorc\_mbwrite** program writes one of the 4 outgoing mailboxes of the specified pRORC card's PCI bridge chip. After loading the new value it displays the values of all mailboxes, like the **mbread** program.

#### Parameters:

<i>pRORC_minor</i>	device file minor number of the pRORC card. Multiple pRORC cards can be supported (with device file name <code>"/dev/prorcN"</code> , where N is the minor number). The minor numbers start from 0. The default value is 0.
<i>mailbox_number</i>	the number of the outgoing mailbox to be loaded (a decimal number from 1 to 4).

*hex\_value* the new value of the mailbox (a 8 digit hexadecimal number).

### 4.2.3 *prorc\_mbreset*

Reset the pRORC card's mailboxes.

#### SYNOPSIS

```
mbreset [-{M|m} <pRORC_minor>]
```

#### DESCRIPTION

The **prorc\_mbreset** program resets the 4 incoming and 4 outgoing mailboxes of the specified pRORC card's PCI bridge chip. The program displays the contents of the chip's Mailbox Empty-Full (MBEF) register [8] before and after the resetting. The non-zero bits of MBEF shows which mailboxes are not empty.

#### Parameters:

*pRORC\_minor* device file minor number of the pRORC card. Multiple pRORC cards can be supported (with device file name `"/dev/prorcN"`, where N is the minor number). The minor numbers start from 0. The default value is 0.

### 4.2.4 *prorc\_empty\_mb*

Read one of the pRORC card's mailboxes till no new data arrives.

#### SYNOPSIS

```
prorc_empty_mb [-{M|m} <pRORC_minor>]  
                [-{N|n} <mailbox_number>]
```

#### DESCRIPTION

The **prorc\_empty\_mb** program reads one of the 4 incoming mailboxes of the specified pRORC card's PCI bridge chip until the chip's Mailbox Empty-Full (MBEF) register [8] shows the given register empty.

#### Parameters:

*pRORC\_minor* device file minor number of the pRORC card. Multiple pRORC cards can be supported (with device file name `"/dev/prorcN"`, where N is the minor number). The minor numbers start from 0. The default value is 0.

*mailbox\_number* the number of the incoming mailbox to be read (a decimal number from 1 to 4). Default value is 1.

## 4.3 rorc\_reset

Initialize the RORC card and the DDL link

### SYNOPSIS

```
rorc_reset [-{M|m} <RORC_minor>] [-{C|c} <DDL channel>]  
          [-D|d|B|b|S|s|F|f|O|o|E|e|N|n]
```

### DESCRIPTION

The **rorc\_reset** program initializes the RORC card and /or a DDL channel According to the user request the routine resets the Free FIFO, the other parts of the RORC, the DIU or the SIU. Resetting the RORC card means to empty all its FIFOs, including the Free FIFO and error bits, and then put all programmable features to their reset values. Resetting the DIU or the SIU means to cut the DDL link (if it was on before the call of the program).

### Parameters:

<i>RORC_minor</i>	device file minor number of the RORC card. Multiple RORC cards can be supported (with device file name "/dev/prorcN", where N is the minor number). The minor numbers start from 0. The default value is 0.
<i>DDL channel</i>	DDL channel number, default = 0
<i>-D or -d</i>	reset DIU
<i>-B or -b</i>	reset both RORC and DIU
<i>-S or -s</i>	reset SIU
<i>-F or -f</i>	clear Free FIFO
<i>-O or -o</i>	clear RORC's other FIFOs
<i>-E or -e</i>	clear RORC's error bits
<i>-N or -n</i>	clear RORC's byte counters
<i>Missing reset code</i>	reset RORC

### SEE ALSO

*siu\_reset*

## 4.4 siu\_reset

Reset SIU and check if link is on.

### SYNOPSIS

```
siu_reset [-{M|m} <RORC_minor>] [-{C|c} <DDL channel>]  
          [-{N|n} <cycle>] [-{T|t} <time-out>]
```

### DESCRIPTION

The **siu\_reset** program initializes the SIU card. Resetting the SIU means to cut the DDL link and bring the SIU into its working state. For the final version SIU card the link will be re-established automatically. The calling of the **siu\_reset** program differs from the **rorc\_reset -s** call that **siu\_reset** checks several times the re-establishment of the link by reading the DIU and SIU status. The program returns if the DIU and SIU are in normal state or the specified check number is over. Before each status check the program sleeps 10 ms.

### Parameters:

<i>RORC_minor</i>	device file minor number of the RORC card. Multiple RORC cards can be supported (with device file name <code>"/dev/prorcN"</code> , where N is the minor number). The minor numbers start from 0. The default value is 0.
<i>DDL channel</i>	DDL channel number, default = 0
<i>cycle</i>	number of DIU and SIU status check after SIU reset. The default value is 3.
<i>time-out</i>	microseconds to wait for DIU response. Default value is 1000 $\mu$ s.

### SEE ALSO

*rorc\_reset, diu\_status, siu\_status*

## 4.5 rorc\_id

Display RORC's hardware and software identification words

### SYNOPSIS

```
prorc_id [-{M|m} <RORC_minor>] [-{C|c} <DDL channel>]  
        [-V <major version> -v <minor version>  
        -{P|p} <PLD version> -{S|s} <serial#>]  
        -{N|n} <channels>]  
        [-{D|d}{S|s}] [-{T|t} <time-out>]
```

### DESCRIPTION

The **rorc\_id** program reads and displays the RORC's type and its hardware and firmware identification words. For user request it displays the DIU or SIU firmware id words as well. In the end the program writes out whether the program library version and the RORC firmware are compatible.

The RORC's type is written as "RORC revision number". It is a number read from the PCI configuration space. If its value is 1, the device is pRORC, if it is 2 then the device is a D-RORC having one DDL channel, while if it is 3 then the device is a two-channel integrated D-RORC.

The RORC's hardware identification word contains the hardware release date and version number. The RORC's firmware identification word contains the firmware release date, the firmware's version, and the size of the Free FIFO. The DIU or SIU firmware id words contain the firmware's release date and version.

For the DIU or SIU firmware id the program sends a command to the device asking the fw id. The SIU firmware id can be asked only if the link is on. The program waits as many microseconds for the answer as specified in *time-out* parameter. If the answer arrives, the program interprets and displays it. For the hardware identification number of the DIU or SIU use the **diu\_id** or **siu\_id** programs.

The **rorc\_id** program can be used for writing the RORC's hardware identification as well. This feature is planned for the RORC's producers, not for the RORC's users. When the hardware identification is written and the producer has soldered a resistor out, the identification cannot be changed. As a user, do not use the *-V*, *-v*, *-P*, *-S* and *-N* switches.

### Parameters:

<i>RORC_minor</i>	device file minor number of the RORC card. Multiple RORC cards can be supported (with device file name <code>"/dev/prorcN"</code> , where N is the minor number). The minor numbers start from 0. The default value is 0.
<i>DDL channel</i>	DDL channel number, default = 0
<i>-D</i> or <i>-d</i>	display DIU's firmware id as well
<i>-S</i> or <i>-s</i>	display SIU's firmware id as well.
<i>time-out</i>	time-out value in micro seconds for DIU and SIU id request. Default value is 1000 $\mu$ s.

### SEE ALSO

*diu\_id*, *siu\_id*, *diu\_status*, *siu\_status*

## 4.6 diu\_id

Display DIU's hardware and software identification words

### SYNOPSIS

```
diu_id [-{M|m} <RORC minor>] [-{C|c} <DDL channel>]  
        [-V <major version> -v <minor version>  
        -P <PLD version> -B <speed version>  
        -S <serial#>]
```

### DESCRIPTION

The **diu\_id** program reads and displays the DIU's hardware identification words. This program can be used for writing this information into the card's memory. For writing the hardware identification a special resistor must be soldered in the card. If this resistor is soldered out, the hardware identification cannot be changed.

The DIU's hardware identification word contains the card major's and minor version numbers (e.g. 2.0), the PLD version code (e.g. 20K60E), the card's speed version (e.g. 2125 Mbps) and the card's serial number. If the major version number is 1 then the card is a prototype (old) DDL card, if it is 2 then the card is the final (new) card.

For *embedded* DIUs, i.e. when the DIUs are integrated onto the D-RORC card, the DIU does not have separated hardware identification. It is identified as a channel of the integrated D-RORC card. For reading the hardware identification of the D-RORC card use the **rorc\_id** program.

### Parameters:

<i>RORC_minor</i>	device file minor number of the RORC card. Multiple RORC cards can be supported (with device file name <code>"/dev/prorcN"</code> , where N is the minor number). The minor numbers start from 0. The default value is 0.
<i>DDL_channel</i>	DDL channel number, default = 0

The other parameters are only used when one wants to write in the hardware identification.

### SEE ALSO

*rorc\_id, siu\_id, diu\_status, siu\_status*

## 4.7 siu\_id

Display SIU's hardware and software identification words

### SYNOPSIS

```
siu_id [-{M|m} <RORC minor>] [-{C|c} <DDL channel>]  
        [-V <major version> -v <minor version>  
        -P <PLD version> -B <speed version>  
        -S <serial#>]
```

### DESCRIPTION

The **siu\_id** program reads and displays the SIU's hardware identification words. This program can be used for writing this information into the card's memory. For writing the hardware identification a special resistor must be soldered in the card. If this resistor is soldered out, the hardware identification cannot be changed.

The SIU's hardware identification words can be read only if the link is up.

The SIU's hardware identification word contains the card's major and minor version numbers (e.g. 2.0), the PLD version code (e.g. 20K60E), the card's speed version (e.g. 2125 Mbps) and the card's serial number. If the major version number is 1 then the card is a prototype (old) DDL card, if it is 2 then the card is the final (new) card.

### Parameters:

<i>RORC_minor</i>	device file minor number of the RORC card. Multiple RORC cards can be supported (with device file name <code>"/dev/prorcN"</code> , where N is the minor number). The minor numbers start from 0. The default value is 0.
<i>DDL_channel</i>	DDL channel number, default = 0

The other parameters are only used when one wants to write in the hardware identification.

### SEE ALSO

*rorc\_id, diu\_id, diu\_status, siu\_status*

## 4.8 rorc\_push\_fifo

Push some entries into RORC's Free FIFO.

### SYNOPSIS

```
rorc_push_fifo [-{M|m} <RORC number>]
               [-{C|c} <DDL channel>]
               [-{A|a} <address>] [-{L|l} <length>]
               [-{I|i} <index>] [-{P|p} <pattern>]
               [-{R|r} <cycles>] [-{T|t}]
```

### DESCRIPTION

The **rorc\_push\_fifo** program loads Free FIFO parameters (block address, block length and Ready FIFO index) into the Free FIFO. This program can be used for testing the RORC's Free FIFO feature. The user can specify the first address/length/index entry then the program generates the next entries following a user specified pattern: constant or incrementing addresses and indices will be used. FreeFIFO content can read back and check by the **rorc\_push\_fifo** program.

The program does not check the range of the parameters: it masks them for the given range. It neither checks the Free FIFO status. If it is overflowed then the new parameters will not be loaded. The caller can check this situation using **rorc\_status** program.

### Parameters:

<i>RORC_number</i>	device file minor number of the RORC card. Multiple RORC cards can be supported (with device file name <code>"/dev/prorcN"</code> , where N is the minor number). The minor numbers start from 0. The default value is 0.
<i>DDL_channel</i>	DDL channel number, default = 0
<i>address</i>	address of the first free block. Default value is the start address of the physmem memory.
<i>length</i>	length of the free blocks in words (24 bit). Default value is 4096 words.
<i>index</i>	index of the first ready FIFO, where the "data arrived" flag has to be put to (8 bit). Default value is 0.
<i>pattern</i>	if more than one push is requested this option defines how to make the second, third, etc pushes. The value of this parameter could be <i>c</i> (constant) or <i>i</i> (increment). In the first case all address and index pushed will be the same. In the second case every address is incremented by the block length and every index by one. The default value is <i>i</i> .
<i>cycle</i>	number of pushes. Default: 1.
<i>-T</i>	pushes the transfer address FIFO instead of the FreeFIFO. This feature can be used only for D-RORC devices.

### SEE ALSO

*rorc\_pop\_fifo, rorc\_status*

## 4.9 rorc\_pop\_fifo

Pop one or more entries from RORC's Free FIFO and display them.

### SYNOPSIS

```
rorc_pop_fifo [-{M|m} <RORC minor>]
              [-{C|c} <DDL channel>]
              [-{A|a} <address>] [-{L|l} <length>]
              [-{I|i} <index>] [-{P|p} <pattern>]
              [-{R|r} <cycles>] [-{T|t}] [-{N|n}]
```

### DESCRIPTION

The **rorc\_pop\_fifo** program works only for pRORC, not for D-RORC. It pops Free FIFO parameters (block address, block length and Ready FIFO index) from the Free FIFO. This program can check the data found in the FIFO supposed the **rorc\_push\_fifo** program pushed them before. For this checking the user can specify the first address/length/index entry. The next entries will be checked according a user specified pattern: constant or incrementing addresses and indices will be supposed.

The program can be called for popping all entries in the FIFO and display them.

### Parameters:

<i>RORC_minor</i>	device file minor number of the pRORC card. Multiple pRORC cards can be supported (with device file name "/dev/prorcN", where N is the minor number). The minor numbers start from 0. The default value is 0.
<i>DDL_channel</i>	DDL channel number, default = 0
<i>address</i>	address of the first free block. Default value is the start address of the physmem memory.
<i>length</i>	length of the free blocks in words (24 bit). Default value is 4096 words.
<i>index</i>	index of the first ready FIFO, where the "data arrived" flag has to be put to (8 bit). Default value is 0.
<i>pattern</i>	if more than one entry is in the FIFO this option defines how to check the second, third, etc entries. The value of this parameter could be <i>c</i> (constant) or <i>i</i> (increment). In the first case all address and index popped will supposed to be the same. In the second case every address is incremented by the block length and every index by one. The default parameter value is <i>i</i> .
<i>cycle</i>	number of pops. 0 means pop until Free FIFO is empty. Default: 0.
<i>-T</i>	pops the transfer address FIFO instead of the FreeFIFO. This feature can be used only for D-RORC devices.
<i>-N</i>	do not display the popped values but check their content. Default: display the popped values without checking them.

### SEE ALSO

*rorc\_push\_fifo*

## 4.10 rorc\_status

Show RORC status

### SYNOPSIS

```
rorc_status [-{M|m} <RORC_minor>] [-{C|c} <DDL channel>]
```

### DESCRIPTION

The **rorc\_status** program reads RORC's type, the Control/Status and error registers and displays information about RORC status and errors, such as: RORC's working mode, Free FIFO status, link status, flow control status, etc. The RORC's type is written as "pRORC revision". It is a number read from the PCI configuration space. If its value is 1, the device is pRORC, if it is 2 then the device is a D-RORC with plugged DIU, and if it 3 then the device is a D-RORC with integrated DIUs.

### Parameters:

<i>RORC_minor</i>	device file minor number of the RORC card. Multiple RORC cards can be supported (with device file name <code>"/dev/prorcN"</code> , where N is the minor number). The minor numbers start from 0. The default value is 0.
<i>DDL channel</i>	DDL channel number, default = 0

### SEE ALSO

*rorcReadRorcStatus()*

## 4.11 rorc\_reg

Show RORC registers

### SYNOPSIS

```
rorc_reg [-{M|m} <RORC_minor>] [-{C|c} <DDL channel>]  
        [-{A|a} <register_address>] [-{V|v} <hex_value>]
```

### DESCRIPTION

The **rorc\_reg** program reads RORC's registers directly reachable via PCI and displays their content in binary, hexadecimal and decimal notation. For more information about these registers, see [8] and [5]. For D-RORC devices the routine can read or write a single register as well.

#### Parameters:

<i>RORC_minor</i>	device file minor number of the RORC card. Multiple RORC cards can be supported (with device file name <code>"/dev/prorcN"</code> , where N is the minor number). The minor numbers start from 0. The default value is 0.
<i>DDL channel</i>	DDL channel number, default = 0
<i>register_address</i>	hexadecimal address of a D-RORC register. The maximum register address value is <code>0x6c</code> . The default value is 0.
<i>hex_value</i>	the value to be written into the given D-RORC register.

#### SEE ALSO

*rorcReadRorcStatus()*

## 4.12 diu\_status

Display DIU's status

### SYNOPSIS

```
diu_status [-{M|m} <RORC_minor>] [-{C|c} <DDL channel>]  
          [-{T|t} <time-out>] [-{V|v} <diu_version>]
```

### DESCRIPTION

The **diu\_status** program sends a status asking command to the DIU, waits for its reply and displays the DIU's status. The user has to specify the DIU version (prototype or final) for the correct interpretation of the status. The program displays the DIU's hardware and firmware identifications as well.

### Parameters:

<i>RORC_minor</i>	device file minor number of the RORC card. Multiple RORC cards can be supported (with device file name <code>"/dev/prorcN"</code> , where N is the minor number). The minor numbers start from 0. The default value is 0.
<i>DDL channel</i>	DDL channel number, default = 0
<i>time-out</i>	microseconds to wait for DIU response. Default value is 1000 $\mu$ s.
<i>diu_version</i>	1 for prototype, 2 for final (plugged or embedded) version. Default value is 2.

### SEE ALSO

*siu\_status, rorc\_send\_command*

## 4.13 siu\_status

Display SIU's status

### SYNOPSIS

```
siu_status [-{M|m} <RORC_minor>] [-{C|c} <DDL channel>]  
          [-{T|t} <time-out>] [-{V|v} <diu_version>]
```

### DESCRIPTION

The **siu\_status** program sends a status asking command to the SIU, waits for its reply and displays the SIU's status. The SIU receives the commands and replies only if the link is on. The program displays the SIU's hardware and firmware identifications as well. The program can read the above information only if the link is up.

The user has to specify the DIU version (prototype or final) for the correct interpretation of the status.

### Parameters:

<i>RORC_minor</i>	device file minor number of the RORC card. Multiple RORC cards can be supported (with device file name <code>"/dev/prorcN"</code> , where N is the minor number). The minor numbers start from 0. The default value is 0.
<i>DDL channel</i>	DDL channel number, default = 0
<i>time-out</i>	microseconds to wait for SIU response. Default value is 1000 $\mu$ s.
<i>diu_version</i>	1 for prototype, 2 for final (plugged or embedded) version. Default value is 2.

### SEE ALSO

*siu\_status, rorc\_send\_command ddl\_init\_link*

## 4.14 ddl\_init\_link

Establish DDL link

### SYNOPSIS

```
ddl_init_link [-{M|m} <RORC_minor>]
              [-{C|c} <DDL_channel>]
              [-{P|p} <print>]
              [-{S|s}] [-{T|t}] [-{N|n}]
```

### DESCRIPTION

The **ddl\_init\_link** program tries to establish the DDL link. First it resets the RORC and the DIU (cuts the link). Then it sends commands to DIU to establish the link, and reads DIU status. The program repeats these actions until it finds the link active. It can be run as link monitor, i.e. the user can request not to reset the link and to continue checking the DIU status even when the link is active.

The final version DDL cards automatically establish the link. For prototype DIU version, the call of this routine is mandatory.

### Parameters:

<i>RORC_minor</i>	device file minor number of the RORC card. Multiple RORC cards can be supported (with device file name <code>"/dev/prorcN"</code> , where N is the minor number). The minor numbers start from 0. The default value is 0.
<i>DDL channel</i>	DDL channel number, default = 0
<i>print</i>	# of status prints (default: -1 means no limit). If the program loops many times the printout can be very large. Setting this option the printout can be minimized.
<i>-s</i>	slave (default: master). This option can be used when the link has to be established between two DIUs. The program can be started at one of the DIUs without this option, i.e. as a master. Optionally the program can be started at the other end as well with this option set. Then the program will not initialize the activation of the link but will display the link status.
<i>-t</i>	test: no stop (default: no test). Setting this option the program does not stop when the link became active but continues monitoring the link status. To stop the program, press ^C.
<i>-n</i>	no reset (default: reset). Normally the program resets the RORC and DIU cards before (re)establishing the link. Setting this option the program does not cut the link. If the link is active program write out the link status and does not rebuilt the link.

### SEE ALSO

*diu\_status, siu\_status*

## 4.15 rorc\_send\_command

Send DDL command via RORC

### SYNOPSIS

```
rorc_send_command [-{M|m} <RORC_minor>]
                  [-{C|c} <DDL_channel>]
                  -{W|w} <command>
                  [-{R|r} <reg_num>]
                  [-{T|t} <time-out>]
                  [-{V|v} <diu_version>] // first calling mode
```

OR

```
rorc_send_command [-{M|m} <RORC_minor>]
                  [-{C|c} <DDL_channel>]
                  -{W|w} <cmd_code>
                  -{D|d} <destination>
                  [-{I|i} <tr_id>]
                  [-{P|p} <parameter>]
                  [-{R|r} <reg_num>]
                  [-{T|t} <time-out>]
                  [-{V|v} <diu_version>] // second calling
mode
```

### DESCRIPTION

The **rorc\_send\_command** routine can be used for sending commands to the RORC, to the DIU, to the SIU or to the Front-End Electronics (FEE) via the DDL link. The program sends the specified command and waits a while for the reply.

The command to be sent can be specified several ways:

- as a hexadecimal number starting with “0x” (first calling mode),
- using the command ASCII mnemonics (first calling mode),
- using the DDL command code together with the destination code and other command fields (second calling mode).

A DDL command has the following format:

D31	D30	D12	D11	D8	D7	D6	D5	D4	D3	D2	D1	D0
don't use	PARAMETER FIELD		IDENTIFIER FIELD		CODE FIELD				DESTINATION FIELD			
X	FEE address		transaction ID		command code				JTAG	FEE	SIU	DIU
optional command parameter					pRORC command code				0			

The user has to specify the DIU type (prototype or final) for the correct sending mode. In the case of D-RORC device the register number can be specified as well.

Parameters:

*RORC\_minor* device file minor number of the RORC card. Multiple RORC cards can be supported (with device file name `"/dev/prorcN"`, where N is the minor number). The minor numbers start from 0. The default value is 0.

*DDL\_channel  
command* DDL channel number, default = 0  
the command to be sent.

- a hexadecimal number starting with "0x", or
- the ASCII mnemonic of a DLL or RORC command. The following commands mnemonics are accepted:
  - PRSTAT get RORC status
  - PRID get RORC id
  - LBON RORC loop-back on
  - LBOFF RORC loop-back off
  - HLT\_FLCON HLT flow control on (for D-RORC)
  - HLT\_FLCOFF HLT flow control off (for D-RORC)
  - HLTON HLT switch on (for D-RORC)
  - HLTOFF HLT switch off (for D-RORC)
  - STEON "stop on error" on (only pRORC)
  - STEOFF "stop on error" off (only pRORC)
  - STARTWDMA stat data collection (write DMA)
  - STOPWDMA stop data collection
  - STARTDG start data generator
  - STOPDG stop data generator
  - STARTDLOAD start data downloading (read DMA)
  - STOPDLOAD stop data downloading
  - STARTJTAG start sending JTAG
  - STOPJTAG stop JTAG sending
  - RDYRX ready to receive (SIU command)
  - EOBTR end of block transfer (SIU command)
  - DTCC data transmission control command
  - JSTART start JTAG command (for pRORC)
  - EOJTR end of JTAG command (for pRORC)
  - JTCC JTAG transmission control command
  - STBWR start block write command
  - STBRD start block read command
  - FECTRL FEE control command
  - FESTRD FEE status read command
  - SRST SIU reset command
  - SUSPEND suspend DIU functions
  - WAKEUP wake up the DIU
  - TXLOOP DIU loop-back
  - TSTOP stop DDL test
  - TSTART start DDL test
  - TSTART\_F0 start DDL test with flying 0 pattern
  - TSTART\_F1 start DDL test with flying 1 pattern
  - TSTART\_INC start DDL test with increment pattern

	<ul style="list-style-type: none"><li>• TSTART_DEC start DDL test with decrement pattern</li><li>• DIUHW read DIU hardware id</li><li>• SIUHW read SIU hardware id</li><li>• DIUFW read DIU firmware id</li><li>• SIUFW read SIU firmware id</li><li>• DIUST read DIU status</li><li>• SIUST read SIU status</li><li>• DIUPM read DIU power monitor</li><li>• SIUPM read SIU power monitor</li></ul>
<i>destination</i>	the command destination (4 bit):  0: RORC, 1: DIU, 2: SIU, 4: FEE, 8: JTAG.
<i>tr_id</i>	the transaction id (0-15). Default = 0.
<i>parameter</i>	the command parameter (19 bit hex number). Default = 0.
<i>reg_num</i>	the number (from 0 to 27) of D-RORC register where the command has to be loaded. Default: 6 (DDL command register) in case of DDL commands, and 4 (Control/status reg.) in case of RORC commands. -R is ignored in case of pRORC or when using symbolic command.
<i>time-out</i>	microseconds to wait for SIU response. Default value is 1000 $\mu$ s.
<i>diu_version</i>	1 for prototype, 2 for final version. Default value is 2.

## 4.16 rorc\_send

Download data from the PC to the Front-end via the RORC card

### SYNOPSIS

```
rorc_send [{-M|-m|--minor}          <RORC minor>]
           [{-C|-c|--channel}       <DDL channel>]
           [{-L|-l|--length}        <data length>]
           [{-B|-b|--rand_len}      <seed>]
           [-J|-j|--rand_data]
           [-F|-f|--front_end]
           [{-A|-a|--address}       <fee address>]
           [{-P|-p|--pattern}       {c|a|0|1|i|d}]
           [{-I|-i|--init_word}     <init word>]
           [{-T|-t|--time_out}      <time-out>]
           [{-U|-u|--physmem}       <utilizable memory>]
           [{-O|-o|--offset}        <memory offset>]
           [{-E|-e|--events}        <max event>]
           [{-N|-n|--init_count}    <initial event count>]
           [{-K|-k|--file}          <input file name>]
           [{-Q|-q|--byte_print}    <gigabytes>]
           [-G|-g|--generator]
           [-R|-r|--rorc_lback]
           [-D|-d|--diu_lback]
           [-Y|-y|--ext_lback]
           [-S|-s|--STBWR]
           [-W|-w|--inf_wait]
           [-X|-x|--nocheck]
```

### DESCRIPTION

The **rorc\_send** program is made to send data via the DDL link to the Front-end Electronics. It uses the *physmem* memory management package [6] for allocating memory blocks for the source data and for the returning data in the case of loop-back.

One can call the **rorc\_send** program together with other programs using *physmem* memory management package on other RORCs. In this case the *physmem* memory must be divided to as many parts as many jobs are running. The division can be done by specifying the size of *physmem* to be used by the **rorc\_send** job (options *-U* and *-O*).

#### Parameters:

<i>RORC_number</i>	device file minor number of the RORC card. Multiple RORC cards can be supported (with device file name <code>"/dev/prorcN"</code> , where N is the minor number). The minor numbers start from 0. The default value is 0.
<i>DDL channel</i>	DDL channel number, default = 0
<i>data length</i>	size in words of the block to send (from 1 to $2^{19}-1$ ). The default value is 2047 words. If RORC's data generator and random length are used (option <i>-G</i> and <i>seed</i> is not

equal to RORC\_DG\_NO\_RANDOM\_LEN), the minimum length is 1, and the maximum value of the length will be *data length* rounded down to the nearest integer of power of 2.

*seed* defines the seed value for random data length. If given, the event lengths will vary between 1 and *data length*. Using the value RORC\_DG\_NO\_RANDOM\_LEN no random length will be generated (this is the default).

-J generate random data. (Options -J and -G are mutually exclusive.)

-F the data is sent to Front-end Electronics and will be read back. After sending each data block the program sends a Start Block Read (STBRD) command and waits for data coming back. If -X option is not on the returned data will be checked.

*fee address* the front-end address value to be sent in the Start Block Write (STBWR) command.

*pattern* the event pattern. It could be:

- c constant data = *init\_word*
- a alternating data
- 0 flying 0 starting from 0xffffffffe
- 1 flying 1 starting from 0x00000001
- i incremental data (this one is the default)
- d decremented data

*init word* the first number of each block. The default value is 0.

*time-out* microseconds to wait for the ending of the previous transfer. Default value is 1000  $\mu$ s.

*utilizable memory* specify the useable part of phymem memory in Megabytes. The default value is 30 MB or the physical size of phymem memory, if less than 30 MB.

*memory offset* offset in Megabytes of the useable memory relative to the start of phymem memory. The default value is 0.

*max event* number of data blocks to send. The 0 value specifies infinite block number. The default value is 0.

*initial event count* the event count (first word) of the first event. The default value is 1.

*input file name* if given the program reads here the data to be sent as a text data. Each data block starts with the length of the block; then follow the data words in hexadecimal format, separated by space, tabulator or new line characters. Empty lines and lines started with '\*', '#' or ';' characters are skipped. The data is not checked (implies option -X).

*gigabytes* The program prints the number of Gigabytes sent after this quantity of sent data. The default is 1 GB.

-G use RORC's data generator.

-R loop-back data in the RORC card.

-D loop-back data in the DIU card.

-Y data is looped-back by an external optical cable.

- S send STWBR (Start Block Write) and EOBTR (End of Block Transfer) commands before the first and after the last data download.
- W wait infinite time if data can not be sent because of flow control. One can stop the program if necessary by pressing ^C.
- X do not check the received data (in case of loop-back).

SEE ALSO

*rorc\_send\_jtag, rorc\_receive*

## 4.17 rorc\_send\_jtag

Download JTAG data from the PC to the Front-end via the RORC card

### SYNOPSIS

```
rorc_send_jtag [-{M|m} <minor>] [-{C|c} <DDL channel>]
               -{I|i} <input_fn> -{O|o} <output_fn>
               [-{P|p} <pmem_offset>] [-{A|a} <j_addr>]
               [-{L|l} <length>] [-{T|t} <time-out>]
```

### DESCRIPTION

The **rorc\_send\_jtag** program is made to send JTAG data via the DDL link to the Front-end Electronics. It uses the *phymem* memory management package [5] for allocating memory blocks for the source data and for the returning data. The data is read from a file. 8 JTAG words are packed into a 32-bit word and *length* words are sent in a JTAG frame. The returned TDO bits are packed out and (together with the input bits) are written to the output file.

### Parameters:

<i>minor</i>	device file minor number of the RORC card. Multiple RORC cards can be supported (with device file name <code>"/dev/prorcN"</code> , where N is the minor number). The minor numbers start from 0. The default value is 0.
<i>DDL channel</i>	DDL channel number, default = 0
<i>input_fn</i>	( <i>obligatory parameter</i> ) the program reads here the data to be sent as a text data. Each JTAG data word must be written in a separate row. Empty lines are skipped. Each row contains 3 characters representing the TDI, TMS, and TRST bits.
<i>output_fn</i>	( <i>obligatory parameter</i> ) the program writes here the original TDI, TMS, TRST bits together with the received TDO bits. The format of the file is the same as the input file, with the exception that the TDO bits are separated by 1 space from the other JTAG bits.
<i>pmem_offset</i>	offset in bytes of the useable part of phymem memory
<i>j_addr</i>	address value to be sent in the Start JTAG (JSTART) command.
<i>length</i>	size in words of the JTAG block to send (from 1 to 100). A 32-bit word can contain 8 JTAG words. The data read from the input file are divided to blocks containing 8 times <i>length</i> JTAG words. The default value is 100 words (corresponding to 800 JTAG words).
<i>time-out</i>	microseconds to wait for the arrival of the TDO bits. Default value is 1000 $\mu$ .

### SEE ALSO

*rorc\_send, rorc\_receive*

## 4.18 rorc\_receive

Functional test of the RORC card and the DDL link

### SYNOPSIS

```
rorc_receive [{-M|-m|--minor}           <RORC minor>]
              [{-C|-c|--channel}        <DDL channel>]
              [-v|--verbose]
              [{-G|-g|--generator}       <loop back_mode>]
              [-D|-d|--no_scatter]
              [{-R|-r|--reset_lev}       <reset level>]
              [{-X|-x|--check}          <check level>]
              [-Z|-z|--no_RDYRX]
              [{-B|-b|--page}            <page length>]
              [{-U|-u|--physmem}         <utilizable memory>]
              [{-O|-o|--offset}          <memory offset>]
              [{-E|-e|--events}          <events>]
              [{-I|-i|--init_word}       <init word>]
              [{-P|-p|--pattern}         c|a|0|1|i|d}
              [{-S|-s|--stat_file}       <stat file>]
              [{-L|-l|--length}          <data length>]
              [{-N|-n|--init_count}      <initial count>]
              [{-J|-j|--rand_len}        <random seed>]
              [{-F|--max_fifo}           <max FIFO>]
              [{-f|--min_fifo}           <min FIFO>]
              [{-T|-t|--sleep_time}      <sleep time>]
              [{-W|-w|--DMA_wait}        <DMA wait>]
              [{-Q|--byte_print}         <GBs to print>]
              [{-q|--page_print}         <pages to print>]
              [{-K|-k|--output_file}     <output file>]
              [{-A|-a|--front_end}       <FEE address>]
```

### DESCRIPTION

The **rorc\_receive** program is made to receive and check data from the DDL link or Data Generator. It uses the *physmem* memory management package [6] for allocating memory blocks where the data is be loaded to. The program fills every word of these blocks with its own address then when the data arrives compares every word with its expected value. It also checks whether the event length in the DTSW word is the real event length.

One can call the **rorc\_receive** program for several RORCs. In this case the *physmem* memory must be divided to as many parts as many **rorc\_receive** jobs are running. The division can be done by specifying the size of *physmem* to be used by one **rorc\_receive** job (options *-U* and *-O*).

Parameters:

<i>RORC minor</i>	device file minor number of the RORC card. Multiple RORC cards can be supported (with device file name <code>"/dev/prorcN"</code> , where N is the minor number). The minor numbers start from 0. The default value is 0.
<i>DDL channel</i>	DDL channel number, default = 0
<i>-v</i>	verbose mode: print details for debugging. (Capital V is not accepted!)
<i>loopback mode</i>	use RORC's data generator with loop-back mode: 0 means: use Data Generator, but do not loop-back data. Use this option if you want to send data via the link. 1 means: set DIU loop-back 2 means: set SIU loop-back any other value set RORC loop-back
<i>-N</i>	do not scatter the received data in <i>phymem</i> memory: every event page will be written on the same physical address (pages will overwrite each other).
<i>reset level</i>	0 means: do not reset RORC, neither DIU nor SIU 1 means: reset RORC only 2 means: reset RORC and DIU, do not reset SIU 3 means: reset RORC, DIU and SIU before collecting data
<i>check level</i>	0 means: do not check the received data 1 means: check only the first word of events 2 means: do not check the first word of events 3 means: check the whole received event. The default value is 3.
<i>-Z</i>	do not send RDYRX and EOBTR commands This option is automatically set with <code>-G</code> or <code>-A</code> options.
<i>page length</i>	length in bytes of the memory blocks (pages) where the data are expected. The default value is 4096 bytes.
<i>utilizable memory</i>	specify the useable part of <i>phymem</i> memory in Megabytes. The default value is 30 MB or the physical size of <i>phymem</i> memory, if less than 30 MB.
<i>memory offset</i>	offset in Megabytes of the useable memory relative to the start of <i>phymem</i> memory. The default value is 0.
<i>events</i>	number of events to read (and to generate if <code>-G</code> is set). The 0 value specifies infinite event number. The default value is 0.
<i>init word</i>	the first number of each event's payload (the second word of the events). The default value is 0.
<i>pattern</i>	the event pattern. It could be: <ul style="list-style-type: none"><li>• c constant data = <i>init_word</i></li><li>• a alternating data</li><li>• 0 flying 0 starting from 0xffffffffe</li><li>• 1 flying 1 starting from 0x00000001</li><li>• i incremental data (this one is the default)</li><li>• d decremented data</li></ul>

<i>stat file</i>	name of the file where the program writes the number of bytes transferred. If given, and the file already exist, the program adds the number of transferred bytes to the value already in the file.
<i>data length</i>	size in words of the longest event expected and the size of generated events if <i>-G</i> is set. The default value is 524287 words. If RORC's data generator and random length are used (option <i>-G</i> and <i>random seed</i> is not equal to RORC_DG_NO_RANDOM_LEN), the minimum length is 1, and the maximum value of the length will be <i>data length</i> rounded down to the nearest integer of power of 2.
<i>initial count</i>	the event count (first word) of the first event. The default value is 1.
<i>random seed</i>	defines the seed value for random data length. If given, the event lengths will vary between 1 and <i>data length</i> . Using the value RORC_DG_NO_RANDOM_LEN no random length will be generated (this is the default).
<i>max FIFO</i>	The program fills the Free FIFO at the beginning of the run and after each page until the number of entries in the Free FIFO reaches this value. This filling is done only if the number of entries in the Free FIFO is less than or equal to <i>min FIFO</i> . The default value is 128.
<i>min FIFO</i>	The program fills the Free FIFO after each page if the entries in the Free FIFO is less than or equal to this value. The default value is 127.
<i>sleep time</i>	the time in ms that the program waits after each event, simulating the occupancy of PC during data taking. The default value is 0 ms.
<i>DMA wait</i>	number of cycles the RORC waits after every DMA write. The default value is 16.
<i>GBs to print</i>	The program prints the number of Gigabytes received after this quantity of received data. The default is 1 GB.
<i>pages to print</i>	The program prints the number of pages received when this quantity of pages received. The default: no page printout.
<i>output file name</i>	if given the program dumps here the received events as a text data. The file can contain comment lines starting with '#' character. Each event in the file starts with a comment line containing the event number, then follows the length of the event; then follow the event words, each in a separate row. The data is not checked (implies option <i>-X 0</i> ).
<i>FEE address</i>	the data is read using the Start Block Read (STBRD) DDL command. <i>FEE address</i> is the front-end address value to be sent in the STBRD command.

SEE ALSO  
*rorc\_send*

## 4.19 FeC2

### Front-End Control and Configuration

#### SYNOPSIS

```
FeC2 [-{M|m} <Rorc device>] [-{C|c} <DDL channel>]  
      [-{F|f} <FeC2 script file>] [-{L|l} <log file>]  
      [-{O|o} <mem offset>] [-{U|u} <mem size>]  
      [-{T|t} <DDL timeout>] [-S|-s] [-v] [-H|-h]
```

#### DESCRIPTION

The FeC2 program can be used for controlling and configuring the Front-end Electronics via the DDL. It can download FEE commands and data blocks and can read FEE status and data. The user can compose his task by a simple script file.

The data blocks to be downloaded have to be written into files beforehand. It can happen that the same file has to be downloaded several times during the same run of the program or at different times. To accelerate the file handling there is a possibility to store the files into shared memory segments. If one calls the program with the `-S` switch, the program tries to store each files into shared memory, so for the next use of the file it will be retrieved from memory instead of the file.

Limitations of shared memory usage:

- Each DDL channel has its own shared memory system.
- Maximum number of channels in one LDC is 16.
- Maximum number of files for one channel is 15420.
- Maximum length of a file name is 255 characters.
- Maximum number of shared memory segments used by the program for storing the file data is 127. (One segment is used for administration. So the maximum number of segments used in one LCD is  $128 * 16 = 2048$ . Other programs, e.g. DATE, can use another 2048 segments.)
- 8 bytes are used for administration in each segment; so 4MB-8 bytes can be used for data storage.  $127 * (4MB-8B)$  is slightly less than half GB (precisely 507 MB). The average length of a file is 33.7 KB. (The DDL protocol allows the maximum data block size of 2MB-4bytes.)

There is a small program for cleaning the shared memories in a channel-by-channel way:

```
clean_shm [-m minor] [-c channel] [-x]
```

In case of `-x`, the program scans the shared memories and does not remove them.

Parameters:

<i>RORC device</i>	device file minor number of the RORC card. Multiple RORC cards can be supported (with device file name <code>"/dev/prorcN"</code> , where N is the minor number). The minor numbers start from 0. The default value is 0.
<i>DDL channel</i>	DDL channel number, default = 0
<i>FeC2 script file</i>	name of the script file. The default is FeC2.scr
<i>log file</i>	the name of the log file. The default is FeC2.log
<i>mem offset</i>	offset in physmem memory in MBs, default: 0.
<i>mem size</i>	utilizable physmem memory in MBs, default: 8.
<i>DDL timeout</i>	time-out value for DDL commands. Default: 1000 $\mu$ s.
<i>-S</i>	use shared memory for file storage.
<i>-v</i>	verbose mode (capital V not accepted).
<i>-H</i>	prints a short help and stops.

## INSTRUCTIONS USED IN THE SCRIPT FILE:

The commands and parameters can be separated by space(s) or tabulator(s). Each command should be written in one line. Any number of empty lines is allowed. Lines starting with a '#' , '\*' or ';' character are considered as comment. After ';' or '//' characters the remaining part of any line is considered as in-line comment. Comment lines, in-line comments and empty lines can be used in data files as well.

All the commands will be executed sequentially up the end of the script file, or until reaching a return/stop command, or till the occurrence of an error.

### DDL related instructions:

#### **reset [RORC | DIU | SIU]**

##### ACTION:

Resets the given part of the DDL link. If no parameter is given then the RORC card will be reset.

#### **write\_RDYRX**

##### ACTION:

A RDYRX DDL command will be sent to the Front-end.

#### **write\_EOBTR**

##### ACTION:

An EOBTR DDL command will be sent to the Front-end.

#### **write\_command <command code>**

##### WHERE:

<command code> is a hexadecimal number of maximum 19 bits.

##### ACTION:

A DDL command will be sent to the Front-end.

**write\_block <address> <file name> [<format>]**

WHERE:

- <address> is the front-end address (of maximum 19 bits) where the block has to be sent.
- <file name> is the name of the file where the data is.
- <format> the C style format of reading a word from the file. If missing binary file is supposed.

ACTION:

First the address, then the block of data will be sent to the Front-end. The length of the file should correspond to the length expected for the given address. The maximum length allowed is ( $2^{19}-1$ ) words.

**write\_jtag <input file> <output file> [<jtag length>]**

WHERE:

- <input file> is the name of the file where is the JTAG data to sent. The file has text format: 3 characters in each line, representing the TDI, TMS and TRST bits.
- <output file> the name of the file where the return data has to be written. The format of the file is similar to the input file, 5 characters in each line: TDI, TMS, TRST, space, TDO.
- <jtag length> the length of a JTAG frame in words. If the input data exceeds this value then several frames will be sent. The maximum and default values are 100 words.

ACTION:

The given JTAG data will be sent to the Front-end using the DDL JTAG channel, and then the TDO bits will be read and written to the output file.

**read\_and\_print <address> "<format>" [<stream>]**

WHERE:

- <address> is the front-end address (of maximum 19 bits) where the status read request has to be sent.
- <format> the C style format for printing the read data.
- <stream> the file name where to append the print. If <stream> is missing the program does the following: if a log file was defined (see the calling sequence of FeC2) it will be used, if not the print will be sent to the standard output.

ACTION:

A status read command will be sent to the Front\_end and the received value will be printed according to the C style <format> into the <stream> stream.

**read\_and\_check <address> <status> <mask>**

WHERE:

- <address> is the front-end address (of maximum 19 bits) where the status read request has to be sent.
- <status> is the expected reply sent by the Front-end (a 19 bit hexadecimal number)
- <mask> will be AND-ed with the received value before comparing against <status>.

**ACTION:**

A status read command will be sent to the Front-end and the (*reply* & <mask>) will be compared with <status>. If the comparison fails the "check\_fail" flag is set (see command **stop\_if\_failed** below).

**read\_until <address> <status> <mask> <timeout>**

**WHERE:**

- <address> is the front-end address (of maximum 19 bits) where the status read request has to be sent.
- <status> is the expected reply sent by the Front\_end (a 19 bit hexadecimal number)
- <mask> will be AND-ed with the received value before comparing against <status>.
- <timeout> is the maximum time in microseconds while the repeated status read is going on.

**ACTION:**

A status read command will be sent to the Front-end and the (*reply* & <mask>) will be compared with <status>. This will be repeated until exact match happens or the timeout is over. In the latter case the "check\_fail" flag is set (see command **stop\_if\_failed** below).

**read\_block <address> <file\_name> [<format>]**

**WHERE:**

- <address> is the front-end address (of maximum 19 bits) where the block has to be read from.
- <file name> is the name of the file where the data has to be written to.
- <format> the C style format of writing a word into the file. If missing, binary file will be written.

**ACTION:**

First the address will be sent, then a block of data will be read and written to the file. The length of the block is under the control of the Front-end.

**read\_and\_check\_block <address> <file\_name> [<format>]**

**WHERE:**

- <address> is the front-end address (of maximum 19 bits) where the block has to be read from.
- <file name> is the name of the file which contains the data to compare with.
- <format> the C style format of reading a word from the file. If missing binary file is supposed.

**ACTION:**

First the address, will be sent, then a block of data will be read and compare with the data in the file. The length of the block is under the control of the Front-end. If the comparison fails the "check\_fail" flag is set (see command **stop\_if\_failed** below).

Program flow related commands:

**define** <name> <value>

ACTION:

Whenever the <name> occurs as a command parameter the <value> will be used instead. The definition of <name> must be before its first use. To distinguish <name> and numbers the <name> must start with a letter while hexadecimal constants must start with 0x.

**wait** <usecs>

ACTION:

The execution of the program is suspended for the given number of microseconds.

**call** <file\_name>

ACTION:

The execution will jump to another script file, if file is found, else: stop processing. Recursive calls are not allowed.

**return**

ACTION:

Terminate the processing of the current script and return to one level higher (or stop in the highest level).

**stop\_if\_failed** [<exit\_code>]

ACTION:

If a previous check (**read\_and\_check**, **read\_until**, or **read\_and\_check\_block** instructions) fails the command processing will be stopped with the given or with 1 exit code.

**stop** [<exit\_code>]

ACTION:

Terminate the command processing with the given or 0 exit code.

SEE ALSO

*rorc\_send\_command rorc\_send, rorc\_receive*

## 4.20 feic.menu

Set and check the Front-End Emulator Interface Card (FEIC)

### SYNOPSIS

```
feic.menu [-{M|m} <minor>] [-{C|c} <channel>]  
          [-{F|f} <config file>
```

### DESCRIPTION

The **feic.menu** command offers a menu for checking and setting the FEIC card [7] working modes. The DDL link must be initialised before the call of **feic.menu** command.

The following working modes can be set:

*PATGEN* the pattern code of the events to be generated by the FEIC.

- 1: external pattern generator
- 2: alternating pattern
- 3: flying 0
- 4: flying 1
- 5: incrementing data
- 6: decrementing data

*EVLEN* event length or in case of random length generation the maximum event length: The FEIC can generate events with lengths between 16 words and 256Mwords in power of 2 steps (16, 32, ....., 256M). Each length can be set by two codes, according to the following table. In the case of using the second code, events with random length will be generated. In this case the length of the events will be varied between 1 word and the given number of words. The following codes are allowed:

- 01 or 81: event length: 16 words
- 02 or 82: event length: 32 words
- .....
- 09 or 89 event length: 4K words
- 0A or 1A: event length: 8K words
- .....
- 0F or 8F event length: 256K words
- 10 or 90: event length: 512K words
- .....
- 19 or 99: event length: 256M words

*TRIG* trigger mode:

- 1: external push button
- 2: external trigger
- 3: 16 clocks gap after each event
- 4: 128 clocks gap after each event
- 5: every 10 ms
- 6: every 100 ms

*FLCTRL* flow control simulation:  
1: flow control after each received word  
2: flow control after receiving 128 words  
3: flow control after receiving 16 Kword

*TRDIS* transmission disable:  
1: after each word sent  
2: after transmitting 128 words  
3: after transmitting 16 Kword.

*SEED* defines the seed value for random data length.

After a successful setting of FEIC working modes the program saves the parameter values into a configuration file. At the next call of the program it offers the saved values as the default ones.

Parameters:

*minor* device file minor number of the RORC card. Multiple RORC cards can be supported (with device file name "/dev/prorcN", where N is the minor number). The minor numbers start from 0. The default value is 0.

*channel* DDL channel number, default = 0

*config file* name of the FEIC configuration file. The default is *feic.cfg*

SEE ALSO

*prorcArmFeic()*



## 5 INSTALLATION

The header, source, object and executable files of RORC and DDL test programs and library are in a common *afs* area:

*/afs/cern.ch/alice/daq/ddl/rorc/*

This directory contains the different versions of the software as separate sub directories. It also contains the different versions in compress formats.

The latest version of the software is version 4.3. The compressed file names show the version number and the time of archiving. Use always the latest date of the given version. E.g. the selected version can be the file named *vers.4.3/rorc\_vers.4.3.3\_2004.07.23.tgz*, which contains *rorc* directory and was archived on 23 July 2004. The latest distributed version can be found in the DDL home pages as well:

[http://cern.ch/ddl/rorc\\_support.html](http://cern.ch/ddl/rorc_support.html)

Copy the compressed file onto your area, uncompress it and extract all directories and files from it. Use the following command for extract files:

```
gtar -xvzf rorc_vers.4.3.3_2004.07.23.tgz rorc/
```

You will get a directory structure with the following subdirectories:

- *rorc/* source header and make files
- *rorc/Linux/* executables

Some test programs uses the *phymem* memory manager module. [6]. We suppose that it is installed on the given machine.

Run *make -f Makefile clean*, then run *make -f Makefile*.

As a super user run *load\_rorc*. It will insert the driver module. Currently we have RORC driver for LINUX kernel 2.4. It is useful to make this insertion automatic during system boot by. adding the following line into */etc/rc.d/rc.local* file:

```
/sbin/insmod <your_dir>/rorc/Linux/prorc_driver.o
```

If an older version of the RORC driver is already loaded then run *reload\_rorc*. In addition run *make -f Makefile dev* (only once) to create the device files.

Now you can use the RORC and DDL test programs and library.



## NOTES