

Intra ROB API's

Authors : G.Crone, M.Huet

Keywords : ROB ROB-in ROB-out API

DRAFT
Abstract

This note introduces the software interfaces between the components of a ROB complex.

NoteNumber :

Version : 0.03

Date : 10-Dec-1998

Reference :

1 Introduction

1.1 Purpose of the document

This document forms part of the specification of the software within a ROB complex in the Level-2 Pilot Project. This is both in order to allow the groups working on ROB prototypes to use a common design and to lead toward the specification of software required in the final ROB complex.

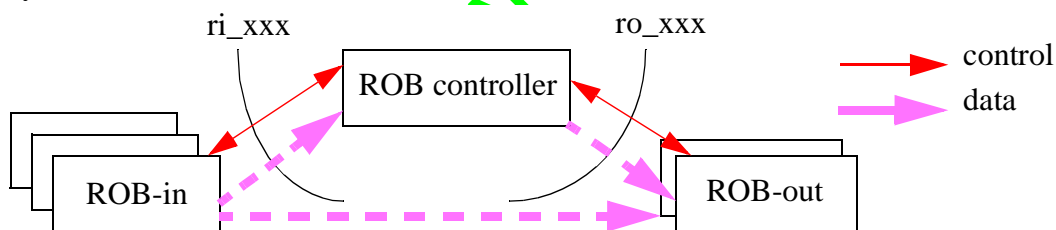
1.2 Boundaries

This note presents the internal API's of a ROB complex i.e. the API's between elements of the ROB complex. The API presented by the ROB complex to other parts of the level 2 system are described elsewhere.

Also the internals (including data structures) of the ROB-in and ROB-out are left up to the implementors of the various prototypes.

2 General description

The ROB Complex consists of at least 3 types of component, the input module (ROB-in), the network interface (ROB-out) and a controller. The controller is interfaced to both the input module(s) and the network interface(s). The communication among the components is described by the API's defined here.



Data requests (from both the level 2 trigger and DAQ/EF) and event deletion requests arrive through the ROB-out(s). The ROB controller fields the input messages from the ROB-out. In the case of data requests it determines which of the ROB-ins have relevant data, collects the data from them and prepares an output message to send to the requestor via the ROB-out. In the case of the deletion requests it fans out the request to each of the ROB-ins.

In some implementations it may be that data are sent by a ROB-out directly from the memory in a ROB-in or that the data is copied to an intermediate buffer in the ROB-out by the ROB-in but whatever the data path the transaction will be controlled by the ROB controller

In the software described here, the ROB controller is assumed to be implemented essentially as a polling loop (see flowchart in Appendix B).

The components of the ROB complex may be implemented in a variety of hardware which gives rise to the usual concerns of byte ordering (eg. big endian RIO Vs. little endian i960) and word size (eg. 64bit alpha Vs. 32bit SHARC).

3 The API's

There are 2 intra ROB API's.

- The API presented by the ROB-in to the ROB controller.
- The API presented by the ROB-out to the ROB controller.

4 The ROB-in API

4.1 Requirements

- The ROB-controller has to be able to acquire data for selected events.
- The ROB-controller has to be able to release unwanted events.
- The ROB-controller has to be able to control several ROB-in modules.

4.2 Functions

The functions of the ROB-in API fall into 3 categories.

- **Control:** ri_open, ri_close, ri_configure, ri_start & ri_reset.
- **Data handling:** ri_request_data, ri_poll_data, ri_delete_fragments, ri_load_emulation_data & ri_load_test_data.
- **Monitoring:** ri_get_statistics.

4.2.1 Control functions

Before data taking starts, each ROB-in must be opened by a call to ri_open, configured by a call to ri_configure and started by a call to ri_start.

4.2.2 Data handling functions

Data are requested by calls to ri_request_data which is passed a pointer to a data structure defining which data are wanted and where they should be put. The data are not necessarily copied to the ROB controller but may be copied to any location on the PCI bus.

After each request has been satisfied the ROB controller will be informed by the ROB-in concerned. The 'request completed' notifications should be polled by calls to the function ri_poll_data which will find the data structure of the next completed request if any are available.

ri_open

Synopsis

```
#include "rob-in.h"
RiReturnCode ri_open (RobInId id, RobIn *handle)
```

Parameters

RobInId	id	I	Identifier of ROB-in to open.
RobIn	*handle	O	Handle of the ROB-in opened to be used in all further calls to the API referring to this ROB-in

Returns

RI_SUCCESS	Normal successful completion
RI_OPEN	Specified ROB-in has already been opened
RI_NOTFOUND	Specified ROB-in could not be found

Description

Open a communication channel to the ROB-in and allocate any necessary resources. This involves the following 3 operations.

- Locate the device and map it into the virtual address space of the caller.
- Initialise message passing
- Allocate a ROB-in parameter structure.

Unresolved questions:

How do we specify which ROB-in we want to open and how do ROB-ins get mapped to the detector? For an emulator it is OK to arbitrarily allocate a ROB-in to an $\eta\phi$ region but in a real system the mapping is fixed by which cable/fibre is plugged in where. Do the individual ROB-ins know what $\eta\phi$ region they are attached to? Does the ROB controller ask the ROB-in for this information?

I am assuming here that the ROB controller has looked up an $\eta\phi$ region in a lookup table to obtain the ROB-in ID and that each ROB-in has some unique ID which can be read through the PCI bus and used by the code in the open function.

ri_close

Synopsis

```
#include "rob-in.h"

RiReturnCode ri_close (RobIn handle)
```

Parameters

RobIn	handle	I	handle of ROB-in to be closed.
-------	--------	---	--------------------------------

Returns

RI_SUCCESS	Normal successful completion
RI_OPEN	Specified ROB-in has not been opened

Description

Close communication channel to ROB-in, de-allocate associated resources and reset the ROB-in hardware.

DRAFT

ri_configure

Synopsis

```
#include "rob-in.h"
```

```
RiReturnCode ri_configure (RobIn handle, RiMode mode)
```

Parameters

RobIn	handle	I	Handle of ROB-in to be configured.
RiMode	mode	I	Mode of operation for this run.

Returns

RI_SUCCESS	Normal successful completion
RI_OPEN	Specified ROB-in has not been opened
RI_BAD_MODE	Specified mode is not available

Description

Set the mode of operation for this ROB-in. Valid modes include Normal, Test and Emulate.

DRAFT

ri_start

Synopsis

```
#include "rob-in.h"

RiReturnCode ri_start (RobIn handle)
```

Parameters

RobIn	handle	I	Handle of ROB-in to be configured.
-------	--------	---	------------------------------------

Returns

RI_SUCCESS	Normal successful completion
RI_OPEN	Specified ROB-in has not been opened

Description

Start the ROB-in, enable external inputs as appropriate to the configured mode.

DRAFT

ri_reset

Synopsis

```
#include "rob-in.h"

RiReturnCode ri_reset (RobIn handle)
```

Parameters

RobIn	handle	I	Handle of ROB-in to be reset.
-------	--------	---	-------------------------------

Returns

RI_SUCCESS	Normal successful completion
RI_OPEN	Specified ROB-in has not been opened

Description

Reset the specified ROB-in back to the state it was in after the open but before the configure.

DRAFT

ri_delete_fragments

Synopsis

```
#include "rob-in.h"
```

```
RiReturnCode ri_delete_fragments (int nfragments, EventID*list)
```

Parameters

int	nfragments	I	Number of items in list.
EventID	*list	I	List of event ID's to be deleted.

Returns

RI_SUCCESS	Normal successful completion
RI_OPEN	No ROB-in has been opened

Description

Inform all currently open ROB-ins that the event fragments given by list are no longer required and their buffer space may be reused. The ROB-ins will process this information asynchronously. This message does not need an acknowledgement from the ROB-ins.

ri_request_data

Synopsis

```
#include "rob-in.h"

RiReturnCode ri_request_data (RobIn handle,
                             RiRequest *request)
```

Parameters

RobIn	handle	I	Handle of the ROB-in to which the request should be sent
RiRequest	*request	I	Data structure defining the data being requested and where it is to be written.

Returns

RI_SUCCESS	Normal successful completion
RI_OPEN	Specified ROB-in has not been opened
RI_BUSY	The request could not be sent to the ROB-in because it has too many requests outstanding.

Description

Send a data request to ROB-in identified by handle. This is a non-blocking function which returns as soon as the request has been sent without waiting for the response from the ROB-in. The caller should call `ri_poll_data` to find out if the data has been transferred.

The selection criteria (e.g. event ID) and the destination of the data are defined by the `RiRequest` structure request. The destination does not have to be in the ROB controller. It could, for example, be in the ROB-out but the ROB controller will receive notification of the completion of the data transfer.

The request structure also defines whether the data itself or a descriptor of the data should be copied to the address given as the destination.

ri_poll_data

Synopsis

```
#include "rob-in.h"

RiReturnCode ri_poll_data (RiRequestId *request_id)
```

Parameters

RiRequestId	*request_id	O	Identifier of a completed data request.
-------------	-------------	---	---

Returns

RI_SUCCESS	A new request has completed (*request_id is valid)
RI_PENDING	No request has completed (*request_id is invalid)
RI_OPEN	No ROB-in has been opened
RI_NO_REQUEST	There are no outstanding data requests

Description

Check all outstanding data requests to see whether any request has been fulfilled. When a request has been satisfied, its actual length will be stored in the appropriate field of the RiRequest structure.

ri_get_statistics

Synopsis

```
#include "rob-in.h"
```

```
RiReturnCode ri_get_statistics (RobIn handle,  
                               RiStatistics *statistics)
```

Parameters

RobIn	handle	I	Handle of ROB-in to query
RiStatistics	*statistics	O	Buffer to receive statistics

Returns

RI_SUCCESS	Normal successful completion
RI_OPEN	Specified ROB-in has not been opened

Description

Retrieve statistics from a ROB-in.

DRAFT

ri_load_emulation_data

Synopsis

```
#include "rob-in.h"
```

```
RiReturnCode ri_load_emulation_data (RobIn handle,  
                                     int data_length,  
                                     int *emulation_data)
```

Parameters

RobIn	handle	I	Handle of ROB-in to load data into
int	data_length	I	Length of data to be downloaded in words.
int	*emulation_data	I	Data to be loaded

Returns

RI_SUCCESS	Normal successful completion
RI_OPEN	Specified ROB-in has not been opened
RI_NOFIT	Data does not fit

Description

Load ROB-in with data for use in emulation mode.

ri_load_test_data

Synopsis

```
#include "rob-in.h"
```

```
RiReturnCode ri_load_test_data (RobIn handle,  
                                int data_length,  
                                int *test_data)
```

Parameters

RobIn	handle	I	Handle of ROB-in to load data into
int	data_length	I	Length of data to be downloaded in words.
int	*test_data	I	Data to be loaded

Returns

RI_SUCCESS	Normal successful completion
RI_OPEN	Specified ROB-in has not been opened
RI_NOFIT	Data does not fit
RI_NOTEST	Test input facility not available

Description

Load ROB-in with data via test input for use in test mode only.

5 The ROB-out API

It is possible to define the role of the ROB-out in the following way:

It receives requests from the outside of the ROB and stores them according to their types. It provides the ROB controller with the external requests on demand. It sends replies to the requesters after reply building by the ROB controller.

The input part is made in such a way that the ROB controller can schedule its activity by choosing the moment it gets a new request to be satisfied and the type of this request (data request or delete request).

5.1 Requirements

- The ROB controller must be able to obtain data and fragment deletion requests from the ROB-out
- The ROB controller must be able to send out data in response to the data requests.
- The ROB controller must be able to handle multiple ROB-outs.

5.2 Functions

- **Control:** ro_open, ro_close, ro_reset.
- **Data handling:** ro_get_data_request, ro_get_delete_request, ro_send_data.
- **Monitoring:**

DRAFT

ro_open

Synopsis

```
#include "rob-out.h"
```

```
RoReturnCode ro_open (???, RobOut *handle)
```

Parameters

RobOut	*handle	O	Handle of the opened ROB-out
--------	---------	---	------------------------------

Returns

RO_SUCCESS	Normal successfull completion
RO_OPEN	Already opened
RO_NOTFOUND	ROB-out could not be found

Description

Open a communications channel to the ROB-out.

ro_close

Synopsis

```
#include "rob-out.h"

RoReturnCode ro_close (RobOut handle)
```

Parameters

RobOut	handle	I	Handle of the ROB-out to close
--------	--------	---	--------------------------------

Returns

RO_SUCCESS	Normal successfull completion
RO_OPEN	Not opened

Description

Close communications channel to the ROB-out.

DRAFT

ro_get_data_request

Synopsis

```
#include "rob-out.h"
```

```
RoReturnCode ro_get_data_request (RobOut handle,  
                                  RoDataRequest *request)
```

Parameters

RobOut	handle	I	Handle of the ROB-out
RoRequest	*request	O	Buffer to receive request data structure.

Returns

RO_SUCCESS	Normal successful completion
RO_OPEN	ROB-out not opened
RO_NONEWAITING	No request available

Description

Poll for a data request from the ROB-out and read it if available. This call includes an implicit copy of the request message and any necessary byte reordering is performed during the copy.

ro_get_delete_request

Synopsis

```
#include "rob-out.h"

RoReturnCode ro_get_delete_request (RobOut handle,
                                   RoDeleteRequest *request)
```

Parameters

RobOut	handle	I	Handle of the ROB-out
RoDeleteRequest	*request	O	Buffer to receive request data structure.

Returns

RO_SUCCESS	Normal successfull completion
RO_OPEN	ROB-out not opened
RO_NONEWAITING	No request available

Description

Poll for the next delete request from the ROB-out and read it if available. This call includes an implicit copy of the request message and any necessary byte reordering.

DRAFT

ro_send_data

Synopsis

```
#include "rob-out.h"
```

```
RoReturnCode ro_send_data (RobOut handle, RoData *output_data)
```

Parameters

RobOut	handle	I	Handle of the ROB-out
RoData	*output_data	I	Pointer to a descriptor of the data to be sent

Returns

RO_SUCCESS	Normal successfull completion
RO_OPEN	Already opened
RO_NOTFOUND	ROB-out could not be found

Description

Send out the data described by the RoData descriptor.

DRAFT

Appendix A: The header files

A very preliminary guess at the contents of rob-in.h and rob-out.h.

rob.h

```
#ifndef ROB_H
#define ROB_H

typedef struct {
    int nblocks ;
    struct {
        PciAddress address ;
        int length ;
    } block[MAX_BLOCKS] ;
} RobDataDescriptor ;

#endif
```

rob-in.h

```
#ifndef ROB_IN_H
#define ROB_IN_H

#include "rob.h"

typedef int RobIn ;          /* Handle of a ROB-in */
typedef int RobInId ;      /* Unique ROB-in identifier */

typedef enum {
    RI_SUCCESS = 0,
    RI_OPEN,              /* ROB-in open error */
    RI_NOTFOUND,         /* ROB-in not found */
    RI_BUSY,
    RI_BAD_MODE,
    RI_NOFIT,
    RI_NOTEST} RiReturnCode ;

typedef enum {RI_SLINK, RI_EMULATE, RI_TEST} RiMode ;
typedef enum {RI_COPY, RI_DESCRIPTOR} RiRequestMode ;
typedef enum {RI_ROI, RI_FRAGMENT} RiSelection ;

typedef int RiRequestId ;

typedef struct {
    RiRequestIdrequest_id ;
    EventID event ;
    RiSelection selection ;
    int eta_min ;
```

```

    int eta_max ;
    int phi_min ;
    int phi_max ;
    RiRequestMode request_mode ;
    PciAddress destination_address ; /* PCI address */
    int max_length ;
    int actual_length ;
} RiRequest ;

typedef struct {
    int fragments_indexed ;
    int fragments_released ;
    int fragments_requested ;
} RiStatistics ;

/* Function prototypes */
#ifdef __cplusplus
extern "C" {
#endif
RiReturnCode ri_open (RobinId id, RobIn *handle) ;
RiReturnCode ri_close (RobIn handle) ;
RiReturnCode ri_configure (RobIn handle, RiMode mode) ;
RiReturnCode ri_start (RobIn handle) ;
RiReturnCode ri_reset (RobIn handle) ;
RiReturnCode ri_delete_fragments (int nfragments,
                                   EventId *list) ;
RiReturnCode ri_request_data (RobIn handle,
                               RiRequest *request) ;
RiReturnCode ri_poll_data (RiRequestId *request_id) ;
RiReturnCode ri_get_statistics (RobIn handle,
                                RiStatistics *statistics) ;
RiReturnCode ri_load_emulation_data (RobIn handle,
                                     int data_length,
                                     int *emulation_data) ;
RiReturnCode ri_load_test_data (RobIn handle,
                                int data_length,
                                int *test_data) ;

#ifdef __cplusplus
}
#endif
#endif

```

rob-out.h

```

#ifndef ROB_OUT_H
#define ROB_OUT_H

#include "rob.h"

```

```

#define RO_MAX_DELETES 100

typedef enum {RO_DATA, RO_DELETE} RoRequestType ;
typedef int RobOut ;

typedef enum {
    RO_SUCCESS,
    RO_OPEN,
    RO_NOTFOUND,
    RO_NONEWAITING} RoReturnCode ;

typedef struct {
    RoRequester    requester ;
    EventID        event ;

} RoDataRequest ;

typedef struct {
    RoRequester    requester ;
    RobDataDescriptor  data ;
} RoData ;

typedef struct {
    int nevents ;
    EventId event[RO_MAX_DELETES] ;
} RoDeleteRequest ;

/* Function prototypes */
#ifdef __cplusplus
extern "C" {
#endif
RoReturnCode ro_open (???, RobOut *handle) ;
RoReturnCode ro_close (RobOut handle) ;
RoReturnCode ro_get_data_request (RobOut handle,
                                   RoDataRequest *request) ;
RoReturnCode ro_get_delete_request (RobOut handle,
                                     RoDeleteRequest *request) ;
RoReturnCode ro_send_data (RobOut handle, RoData *data) ;
#ifdef __cplusplus
}
#endif
#endif

```

DRAFT

Appendix B: Example ROB controller flow chart

