DYNAMIC ENGINEERING
150 DuBois St. Suite C Santa Cruz CA 95060
831-457-8891
https://www.dyneng.com
sales@dyneng.com
Est. 1988

Software User's Guide (Linux)

SpaceWire RMAP

SpaceRMAP

Dynamic Engineering
150 DuBois St Suite C
Santa Cruz, CA 95060
831-457-8891

Revised 05/26/2021

Dynamic Engineering has made every effort to ensure that this manual is accurate and complete. Still, the company reserves the right to make improvements or changes in the product described in this document at any time and without notice. Furthermore, Dynamic Engineering assumes no liability arising out of the application or use of the device described herein.

The electronic equipment described herein generates, uses, and can radiate radio frequency energy. Operation of this equipment in a residential area is likely to cause radio interference, in which case the user, at his own expense, will be required to take whatever measures may be required to correct the interference.

Dynamic Engineering's products are not authorized for use as critical components in life support devices or systems without the express written approval of the president of Dynamic Engineering.

Connection of incompatible hardware is likely to cause serious damage.

Table of Contents

Product Description

SpaceWire Remote Memory Access Protocol (RMAP) was developed to support reading and writing from/to memory in a remote SpaceWire.node. RMAP can be used to configure a SpaceWire network, control SpaceWire nodes, and transfer data between nodes. Software is written in C and currently executes on Linux.

Software Description

The Dynamic Engineering SpaceWire RMAP package was developed and implemented to configure/control our SpaceWire Router product. It supports both in-band (SpaceWire) and out-of-band interfaces including Ethernet, and serial/USB interfaces. The Dynamic Engineering SpaceWire router has both interfaces as well as 12 SpaceWire router ports.

The RMAP package is compliant with applicable SpaceWire specifications, specifically ECSS-ES-50-12C and ECSS-E-50-11C. The software package consists of 2 user space applications. A client RMAP application as well as a server application. Typically. the client application is run on a host computer. For in-band communication, a SpaceWire adapter card must be installed on the host. The server issues RMAP commands and processes corresponding response packets.

The server application (contained in de_RmapSrvr.c) nominally executes in an embedded SpaceWire node. Any packet with a destination address of 0 would be directed to this application. The server executes the specified memory read or write command upon receipt. It then formats the response packet to be returned on the SpaceWire port it was received.

The RMAP package supports the following commands and responses: read single address, read multiple (incrementing address), read/modify/write single address, and write single address. All commands executed may optionally be logged to a file by the server app assuming the node has a filesystem. This is useful for initial bring-up and any subsequent debug as required. The client application includes a menu for initiating command packets and displaying the corresponding responses. The menu routine for the client application demonstrates appropriate usage of APIs/functions contained in the package.

Most of the RMAP code base is shared/utilized in the Dynamic Engineering SpaceWire Router. Sections of code specific to the router are conditionally compiled via a compile time switch. Router registers are accessible from user space in this product, thus the RMAP package as delivered allocates memory for demonstration purposes. When installed, the end user merely points the pointer devMem (in de_RmapSrvr.c) to the RMAP memory space.

If RMAP memory is not accessible from user space, end user must modify access methods as appropriate such as ioctl access. Access methods are contained in de_SpwrCmnRmap.c, specifically wrData and rdData.

Further, mutual exclusion is supported via a compile time switch. One possible method is defined in de_SpwrCmn.h via the macros GET_LOCK and PUT_LOCK.

Four top level routines are invoked by server application. de_confMgmt configures the RMAP interface as program arguments specify. If in-band management (SpaceWire) is utilized, the Dynamic Engineering SpaceWire adapter is assumed for this example. However, any vendor's card should be compatible with minor modifications.

Next, de_rcvRMAP is invoked to receive a RMAP command packet, assuming the packet received is indeed addressed to RMAP command port, de_procRmapCmd is invoked. This function parses and validates the command packet. If command packet is proper, the command is executed, and an appropriate reply packet is formatted and returned. The reply packet is then returned to requestor via de_sndRmap.

The RMAP client application is contained in de_RmapClient.c It invokes 3 top level functions. de_confMgmtIf, the same function invoked by the server application. Of course, both applications must be configured to use same communication interface. de_fmtRmapCmd formats the specified command packet, transmits the command packet, and returns the transaction ID of the command. de_getRmapRply is invoked to receive and parse the expected response packet. These functions utilize the same send and receive functions as the server.

Application Programming model

APIs for top level functions described above:

```
/*****************************************************************
*  de_confMgmtIf
*
*  This function configures the specified management interface
*  for RMAP command processing
*
*  Parameters:
*   iface    - Management interface (0 = SpaceWire, 1 = USB,
*              2 = Ethernet
*   port     - device number for SpaceWire or USB, UDP port for
*              Ethernet
*   mgmtIp   - IP address in IPv4 format if interface is Ethernet
*              and code executing on management app, otherwise *
*              don't care.
*
*  Special Considerations:
*
*  Returns:
*   0 upon success, 0 <  failure
*/
int de_confMgmtIf (de_Iface_t iface, unsigned int port,
                                                  char* mgmtIp);

/*****************************************************************
*  de_rcvRmap
*
*  This function posts a read awaiting RMAP packet
*
*  Parameters:
*   pkt      - Pointer to RMAP packet
*   pktLen   - Length of RMAP packet
*
*  Special Considerations:
*
*  Returns:
*   Length of RMAP packet received, 0 <=  failure
*/
int de_rcvRmap (uint8_t *pkt, int pktLen);
```

```
/*****************************************************************
*
*  de_sndRmap
*
*  This function transmits the specified RMAP packet
*
*  Parameters:
*   pkt     - Pointer to RMAP packet
*   pktLen  - Length of RMAP packet
*
*  Special Considerations:
*
*  Returns:
*   Length of RMAP packet sent, 0 <  failure
*/
int de_sndRmap (uint8_t *pkt, int pktLen);

/*****************************************************************
*  de_procRmapCmd
*
*  This function processes RMAP commands, and formats a reply *
*  packet
*
*  Parameters:
*   pkt     -   SpaceWire packet
*   pktLen  -   Length of input command and length of reply
*               upon return.
*
*  Special Considerations:
*   Packet buffer is reused for reply, and it is assumed it will
*   be large enough for reply packet.
*
*  Returns:
*   Pointer to RMAP reply.
*/
uint8_t* de_procRmapCmd (uint8_t* pkt, int* pktLen);
```

```
/*****************************************************************
*  de_fmtRmapCmd
*
*  This function formats and sends the specified RMAP command
*
*  Parameters:
*   cmd     - RMAP command
*   regAddr - Address of register to access
*   dataLen - Data word count
*   data    - If a write or RMW, data to be written
*   mask    - If RMW, corresponding mask
*   pkt     - Pointer to RMAP command packet
*
*  Special Considerations:
*   It is assumed pkt is of at least the size of DE_MAX_RMAP_CMD
*
*  Returns:
*   Transaction Id of this command upon success, < 0 upon failure
*/
int de_fmtRmapCmd (uint8_t cmd, uint16_t regAddr,
    uint16_t dataLen, uint32_t data, uint32_t mask,
    uint8_t *pkt);

/*****************************************************************
*  de_getRmapRply
*
*  This function posts a read to specifed SpaceWire port awaiting
*  RMAP
*  reply.  Received reply is validated and any read data is
*  returned in buffer pointed to by parameter data.
*
*  Parameters:
*   transId - Expected transaction Id
*   data    - Pointer to data buffer for received data, NULL if
*             no data expected
*   dataLen - Length of buffer in 32 bit words
*
*  Special Considerations:
*
*  Returns:
*   data length of received data upon success, < 0 upon failure
*/
int de_getRmapRply (uint16_t transId, uint32_t *data,
                                              uint16_t dataLen);
```

Installation

Two makefiles are delivered with RMAP package namely Makefile.svr and Makefile.clnt. End user may require one or both. Makefiles are delivered with production compile defines enabled.
Common defines are -DLOG, which enables logging of errors to a file in directory executable is resident. Otherwise, errors will be displayed on stdout. Server log file is named serverLog.txt, client log is clientLog.txt
-DVERBOSE is the other common define, and outputs more debug information. This switch is disabled.
The server makefile has 2 additional defines. -DTRACE enables logging of all memory/register access to the file regTrace.txt in the same directory as executable and is enabled. -DSEM is disabled, this switch enables the mutual exclusion method.

RMAP server and client applications

The server application resides on the device where RMAP commands are executed and is named de_server. de_client is the client menu application issuing RMAP commands and receiving RMAP replies.

Invocation parameters

de_server

./de_server interface (0=SpaceWire,1=Serial,2=USB,3=Ethernet) port(devNum or UDP port for Ethernet) [optional security key (0-255)]

Examples:

SpaceWire, port 1

./de_server 0 1

Ethernet UDP port 50000, change security key to 55

./de_server 3 50000 55

Note: Security keys must be same on server and client or no reply will be returned by server. This is a menu option on client.

de_client

./de_client interface (0=SpaceWire,1=Serial,2=USB,3=Ethernet)

port(devNum or UDP port for Ethernet) IP address if Ethernet (IPv4 dot notation)

SpaceWire, port 0

./de_client 0 1

Ethernet UDP port 50000

./de_client 3 50000 192.168.1.75

Warranty and Repair

Please refer to the warranty page on our website for the current warranty offered and options.

http://www.dyneng.com/warranty.html

Service Policy

Before returning a product for repair, verify as well as possible that the suspected unit is at fault. Then call the Customer Service Department for a RETURN MATERIAL AUTHORIZATION (RMA) number. Carefully package the unit, in the original shipping carton if this is available, and ship prepaid and insured with the RMA number clearly written on the outside of the package. Include a return address and the telephone number of a technical contact. For out-of-warranty repairs, a purchase order for repair charges must accompany the return. Dynamic Engineering will not be responsible for damages due to improper packaging of returned items. For service on Dynamic Engineering Products not purchased directly from Dynamic Engineering contact your reseller. Products returned to Dynamic Engineering for repair by other than the original customer will be treated as out-of-warranty.

Out of Warranty Repairs

Out of warranty repairs will be billed on a material and labor basis. Customer approval will be obtained before repairing any item if the repair charges will exceed one half of the quantity one list price for that unit. Return transportation and insurance will be billed as part of the repair and is in addition to the minimum charge.

For Service Contact:

Customer Service Department
Dynamic Engineering
150 DuBois St. Suite C
Santa Cruz, CA 95060
831-457-8891
InterNet Address support@dyneng.com