CHAPTER 3

# Ethernet Communications

The ASIST/Ethernet software component is a set of application development tools that assists VCO/4K customers in developing host-controlled applications. This tool kit offers programmers a means to provide data communications services to a call processing application.

The ASIST/Ethernet component is application-layer software that allows a UNIX-based host to communicate with a VCO/4K. This product implements the Ethernet protocol, and allows the programmer to concentrate on the application rather than on low-level, host-to-switch communication issues. ASIST/Ethernet is ideal for the application developer who needs to prototype applications quickly in a laboratory environment.

ASIST/Ethernet is written specifically for the Sun Microsystems SunOS operating system.

## ASIST/Ethernet Features

The ASIST/Ethernet component was designed with the following requirements:

- Communication with a VCO/4K over Ethernet via a single application-layer process
- Interprocess Communication Support
- Multiblocked Message Support
- Error Reporting
- Link Control
- Link Configuration
- Link Statistics
- Application access to the Ethernet process via a function call interface using a first-in, first-out (FIFO) queue mechanism.

ASIST/Ethernet is written in the C programming language.

## ASIST/Ethernet Interface Format

The Ethernet Link Manager and the Ethernet Communication Driver communicate information to each other via a standard message format. This message format consists of Commands, Acknowledgments, Ethernet Packet Data, Error Reports and Link Statistics.

# Message Format—Commands

The application can send the following commands to the driver:

- Initialization command—The application issues this command to initialize a host port which it intends to use for communicating with the VCO/4K. This command must supply the UNIX socket via the configuration file. The Initialization Command initializes the port for raw input/output, but does not start polling.
- Activate/Deactivate command—These commands start or stop polling on the link. Only a previously initialized link can be activated.
- Deinitialize command—This command turns off a given link and restores the previous state of the socket.
- Send Statistics command—This command requests link statistics from the Ethernet Communications Driver. The application receives a Statistics message in response to this command.
- Set FIFO Debug command—This command turns on verbose debugging to the standard output for all FIFO actions performed by the Ethernet Communications driver.
- Send Link Debug command—This command turns on verbose debugging to the standard output for all link actions performed by the Ethernet Communications driver.

# Message Format—Acknowledgments

These message types are sent by the Ethernet Driver to the application process in response to a successfully executed command.

# Message Format—Data

Data packets contain Ethernet message data. Data packets are sent in both directions—Application to Ethernet Communications Driver and Ethernet Communications Driver to Application.

# Message Format—Errors

Error packets are sent by the Ethernet Communictions driver to the application to report faulty conditions on a link or the state of the read and write queues. Error packets are also sent if a command could not be executed successfully.

# Message Format—Statistics

The Ethernet Driver responds to a Send Statistics command by sending a Statistics packet. This packet contains various counter values for a given link.

## Link Manager—Communications Driver Interface

The following structure is the C language representation of the Link Manager-Communication Driver interface format. This data structure is located in the enet_if.h file.

```
typedef struct
}
    ushort bytecount;
    byte  pkt_type;
    union {
        DATA_PKT          qdata_pkt;
        LINK_STATS        qstat_pkt;
        ERROR_PKT         qerr_pkt;
        ACK_PKT           qack_pkt;
        CMD_PKT           qcmd_pkt;
        PARAMS            qpara_pkt;
        q_data;
}
Q_PKT;
```

# ASIST/Ethernet Configuration File

The following is a sample configuration file for the Ethernet Communications Driver:

```
/*Sample configuration file for VCO/4K system #1*/
log file: test.log
log level: 1

network a name: j_jetson
network a local: 2121
network a remote: 2122

network b name: g_jetson
betwork b local: 2021
network b remote: 2022
```

The definitions of the configuration file parameters are as follows:

log file—The name of the log file you want to create (or append to) for each log message function call. The log file name is a standard UNIX file name and may be up to 256 characters in length. If a log file name is not specified, the system uses the default, which is ./enetdrvr.log.

log level—The level of logging you want to output to the file. The level is from 0 to 4, where level 0 logs the least information and level 4 logs the most information.

network a name—The Ethernet node name assigned to the ACTIVE side of the VCO/4K system to which you are communicating.

network a local—The local port number for the Ethernet host configuration on the ACTIVE side of the VCO/4K system. The value for this parameter is specified in the Local Port field on the VCO/4K Host Configuration screen.

network a remote—The remote number associated with “network a local” for the ACTIVE side of the VCO/4K system. The value for this parameter is specified in the Rem Port field on the VCO/4K Host Configuration screen.

network b name—The Ethernet node name assigned to the standby side of the VCO/4K system to which you are communicating.

network b local—The local port number for the Ethernet host configuration on the STANDBY side of the VCO/4K system. The value for this parameter is specified in the Local Port field on the VCO/4K Host Configuration screen.

network b remote—The remote number associated with “network b local” for the ACTIVE side of the VCO/4K system. The value for this parameter is specified in the Rem Port field on the VCO/4K Host Configuration screen.

# ASIST/Ethernet Installation

The ASIST products were developed on a Sun Microsystems SPARCstation. While the ASIST product is independent of any particular operating system, the ASIST/Ethernet product requires a UNIX operating system environment.

This section contains a list of the media contents of the ASIST/Ethernet disk and a description of the installation and compilation procedure.

## ASIST/Ethernet Media

All ASIST/Ethernet files reside on a 3.5-inch diskette (1.44 Mb). The directory of this diskette is as follows:

```
/asist/enet/control.c
    datapath.c
    enet.h
    enetdrvr.c
    enetdrvr.make
    enet_if.h
    enet_mgr.c
    enet_mgr.make
    enet_util.c
    host.c
    network.c
    types.h
```

## Installing and Compiling

To copy the ASIST files from the supplied media, enter the appropriate UNIX `tar` command as shown below.

For systems running SunOS:

tar xvf /dev/rfd0 ./asist/enet (SunOS)

For systems running System V:

tar xvf /dev/f0t ./asist/enet (system V)

Each ASIST product includes a makefile for all the source modules. To compile the ASIST/Ethernet product, use the enet_mgr.make file.

The C language flag -DBSD is used in each makefile. When present, this flag indicates that the target operating system is SunOS; its absence indicates a System V environment.

# Function Description

The ASIST/Ethernet product contains the following functional areas:

- ENET Link Manager—This module facilitates the interface of the customer application code to the Ethernet Communications Driver.
- ENET Communication Driver—This module executes commands received from the ENET Link Manager, transmits and receives ENET packets, and manages the link between the host and the VCO/4K.
- ENET Utilities—This module contains functions that are used by the ENET Link Manager and ENET Communications Driver modules.

**Note** The ENET Utilities module uses the same data structures and constants as the ENET Link Manager.

The C language source files and functions provided by each of the functional areas are as follows.

## ENET Link Manager Source Files and Functions

Files: enet_if.h/enet_mgr.c

Functions:
Prepare Data Packet
Prepare Queue Packet
Check Receive Queue
Send Queue Packet to Driver
Spawn Another Driver Process
Initialize ENET Driver
Stop ENET driver
Initialize Link Manager
Process ENET Driver Commands
Send ENET Driver Commands
Create ENET Driver Process
Display ENET Attributes
Read Data From ENET Link
Write Data To Enet Link
Create FIFO
Name FIFO
Terminate ENET Communications Driver

# ENET Communication Driver Source Files and Functions

Files: enet.h/control.c/datapath.c/enetdrvr.c/host.c/network.c

Functions:
Create FIFO Name
Read Configuration File
Get Token
Print Driver Structures
Process Report
Control Data Path
Stop Data
Main
Handle Application Output Queue
Create Q_PKT
Send ACK_PKT
Send ERROR_PKT
Send Status
Open Read/Write FIFOs
Read Packet From Host Application Queue
Close Driver Connection
Display Network Statistics
Log Socket Information
Open Driver Socket
Open TCP Link
Reopen TCP Link
Close Network Side
Close Network Link
Read TCP Link
Write TCP Link
Send Report To Buffer
Send Message To VCO
Check For Report

# ENET Utilities Source Files and Functions

File: enet_if.h/enet_util.c

Functions:
Open And Write To File
Convert Hex to ASCII
Convert ASCII to Hex
Convert ASCII String Into Hexadecimal Byte Stream
Convert Hexadecimal Byte Stream Into ASCII String
Display Q_PKT Contents
Output ASCII Representation of Hexadecimal Byte Stream

# ENET Link Manager Data Structures and Constants

File Name: enet_if.h

```
*/


#define MAX_CFG_NAME           30     /* Max characters in config file name */
#define MAX_BLKS               10     /* Max blocking factor */
#define DONE                   1
#define FAILED                 0
#define MAX_MSG_LENGTH         256    /* max length of link message */

/*
```

Data packets contain single link message. Data packets are sent by the applicaton to the driver,to be transmitted . Similarly driver after deblocking the received link packet, sends it to the application.

```
*/

typedef struct

                {

                        long       unused;
                        byte       link_no          /* for/from which link */

                        unsigned short bytecount;   /* no of data characters */
                        unsigned short  bytecount;
                        unsigned short  msg_no;     /* for future use */

                        unchar  data[MAX_MSG_LENGTH];

                }

DATA_PKT;

#define DATAPKTSIZE                sizeof (DATA_PKT)

typedef struct

                {
                long    unused     /* type of command */
                byte    cmd;       /* for this link */
                byte    link_no;
                char    cfg_name   [MAX_CFG_NAME];
                }

CMD_PKT;

#define CMDPKTSIZE                 sizeof (CMD_PKT)

/*
```

```
     command packets are sent only by the application to the driver.
     The following commands are supported.

*/



#define PORT_DEINIT           0      /* turn off the initialized port */
#define PORT_INIT             1      /* initialize only, no polling */
#define PORT_ACTIVE           2      /* start polling */
#define PORT_DEACTIVE         3      /* stop polling */
#define SEND_STATS            4      /* get statistics */
#define SET_FIFO_DEBUG        5      /* Set SHOW_DRVR_FIFO debug option */
#define SET_LINK_DEBUG        6      /* Set SHOW_DATA_LINK debug option */



typedef struct

      {

          long    unused;                      /* info pertains to this link */
          byte    link_no                      /* equals cmd that is being acknowledged
          char    cfg_name[MAX_CFG_NAME];      */

      }



#define ACKPKTSIZE

sizeof (ACK_PKT)

typedef
struct

          {

                 long    unused;
                 byte    error                          /* type of error */
                 byte    link_no;                       /* info pertains to this link
                 char    cfg_name{[MAX_CFG_NAME];       */
                                                        /* only for init fail*/

          }

ERROR_PKT;

#define ERRPKTSIZE       sizeof (ERROR_PKT)

/*
error packets are sent by the driver to the application in case of faulty
conditions on any of the link or the states of the FIFO.
*/
```

```
#define INIT_FAIL              0      /* Port initialization failed */
#define PORT_INACTIVE          1      /* port is not in active state */
#define POLLING_FAIL           2      /* No response in max attempts */
#define RNR                    3      /* Secondary not ready */
#define TX_FAIL                4      /* No ack till max xmt attempts */
#define NACKS_2MANY            5      /* primary sent max nacks in a row */
#defineMSG_2LONG               6      /* Number of characters > MAX_MSG_LENGTH */



typedef struct

    {

         ulong        unused;
         ulong        Npolled;                  /* Total No of times tty polled */
         ulong        Nno_eots;                 /* Total no of times no EOTS */
         ulong        Ntxed_msgs;               /* No. of msgs txed successfully */
         ulong        Ntx_attempts;             /* No. of transmissioins */
         ulong        Nnacks;                   /* No of NACKS sent */
         ulong        Ntx_blk_msgs[MAX_BLKS];   /* No. of multiblocked msgs txed */
         ulong        Nrxed_msgs;               /* No of msgs rxed */
         ulong        Nlrc_errs;                /* No of LRC errors */
         ulong        Nrx_blk_msgs[MAX_BLKS];   /* No. of multiblocked msgs rxed */
         ulong        Nno_response;             /* No of times secondary failed to
                                                */

    }

LINK_STATS;

#define STATPKTSIZE    sizeof (LINK_STATS)

/*
structure containing the configurable parameter values. On getting this structure the
driver updates its own copy. For the changes concerning baud rate, parity and stopbits
to become effective, PORT_INIT command needs to be issued.
*/




#define MAXPARACHARS8              /* Maximum chars for any parameter name
                                   */

      typedef struct

      {

      long       unused;

      char       cfg_name[MAX_CFG_  /* tty device name */
                 NAME];

      }
```

```
PARAMS
#define PARAPKTSIZEsizeof (PARAMS)
typedef struct

      {

      unsigned short  bytecount;

      byte       pkt_type;         /*type of this queue packet */

      union

         {

         DATA_PKT qdata_pkt;
         LINK_STATS qstat_pkt;
         ERROR_PKT qerr_pkt;
         ACK_PKT qack_pkt;
         CMD_PKT qcmd_pkt;
         PARAMS  qpara_pkt;


      q_data;
      }

Q_PKT;



/*
Type of packets to be exchanged between the driver and the application of the FIFO.
/*

      #define DATA_TYPE                 1
      #define STATS_TYPE                2
      #define EROR_TYPE                 3
      #define CMD_TYPE                  4
      #define ACK_TYPE                  5
      #define PARAM_TYPE                6

/*    used as an index into the enet_mngr [] table */

      #define LOGICAL_LINK_0            0
      #define LOGICAL_LINK_1            1
      #define LOGICAL_LINK_2            2
      #define LOGICAL_LINK_3            3
      #define LOGICAL_LINK_4            4
      #define LOGICAL_LINK_5            5
      #defineLOGICAL_LINK_6             6
      #define LOGICAL_LINK_7            7


      #define MAX_LINKS            8     /*Maximum possible communication links */
      #define MAX_FIFO_NAME        20    /* Maximum characters for FIFO name */
```

```
/* Commands supported by the driver interface */


      #defineCREATE_DRV                 1
      #define TERM_DRVR                 2
      #define LINK_INIT                 3
      #define LINK_DEINIT               4
      #define LINK_ACTIVATE             5
      #define LINK_DEACTIVATE           6
      #define SEND_PARAMS               8
      #define GET_STATS                 9
      #define SHOW_LINKS                10



typedef struct

    {

        int          drvr_pid;
        char         destn_fifo [MAX_FIFO_NAME];
        char         source_fifo [MAX_FIFO_NAME];
        int          destn_fd;
        int          source_fd;
        char         cfg_name [MAX_CFG_NAME];
        byte         link_state;

    }

APP_ENET;
```

# ENET Link Manager Functions

## Names

make_datapkt, get_qpkt, check_qstate, send_toq, create_comm,init_enet, stop_enet, init_linkmngr, process_cmd, send_drvr_cmd, create_drvr, show_process, enet_read, enet_write, makefifo, fifoname, terminate_drvr

## Synopsis

```
#include <stdio.h>
#include <malloc.h>
#include <string.h>
#include <ctype.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <errno.h>
#include <signal.h>
```

```
#if defined(BSD)
#include <sys/wait.h>
#endif
#include "types.h"
#include "enet_if.h"
DATA_PKT *make_datapkt();
int get_qpkt();
uchar check_qstate();
int send_toq();
int create_comm();
int init_enet();
int stop_enet();
void init_linkmngr();
void process_cmd();
void send_drvr_cmd();
void create_drvr();
void show_process();
int enet_read();

int enet_write();
int makefifo();
void terminate_drvr();
```

# Global Data

```
APP_ENET enet_mngr[MAX_LINKS];
Q_PKT ipdata, *ipptr;
DATA_PKT thisdata, *dataptr;
CMD_PKT thiscmd, *cmdptr;
ACK_PKT thisack, *ackptr;
ERROR_PKT thiserror, *errorptr;
LINK_STATS thisstatus, *statusptr;
```

# Prepare Data Packet

### Name

DATA_PKT *make_datapkt(uchar *buf, ushort bytecount)

### Data Structure(s)

DATA_PKT thisdata

| Parameter | Type | Use | Description |
|---|---|---|---|
| buf | uchar* | input | Data from application |
| bytecount | ushort | input | Buffer size |

### Description

This function copies “bytecount” number of bytes from “buf” into the DATA_PKT “thisdata.” It returns a pointer to “thisdata.”

# Prepare Queue Packet

### Name

int get_qpkt(ushort type)

### Data Structure(s)

Q_PKT *ipptr;
DATA_PKT *dataptr;
CMD_PKT *cmdptr;

| Parameter | Type | Use | Description |
|---|---|---|---|
| type | ushort | input | Q_PKT identifier |

### Description

This function prepares the Q_PKT structure, pointed to by "ipptr," to be sent to the ENET Driver process. Valid "type" identifiers for this function are as follows:

```
DATA_TYPE
CMD_TYPE
PARAM_TYPE
```

This function copies the appropriate information into the correct location of the Q_PKT. For PARAM_TYPE it calls the "make_parapkt()" function. If successful, it returns a value greater than zero; otherwise it returns FAILED.

# Check Receive Queue

### Name

uchar check_qstate(ushort linkid)

### Data Structure(s)

APP_ENET enet_mngr[];
uchar process_stat[];
Q_PKT *ipptr;

| Parameter | Type | Use | Description |
|---|---|---|---|
| linkid | ushort | input | Link identifier |

### Description

This function checks if the ENET Driver process has sent any Q_PKT structures to the ENET Link Manager. The queue is identified in "enet_mngr[linkid].source_fd." It copies available data into the appropriate location in the Q_PKT structure, pointed to by "ipptr" and calls the "read()" system call. It returns the type of Q_PKT received; otherwise NULL.

## Send Queue Packet To Driver

### Name

int send_toq(ushort linkid)

### Data Structure(s)

APP_ENET enet_mngr[];
uchar process_stat[];
Q_PKT *ipptr;

| Parameter | Type | Use | Description |
|---|---|---|---|
| linkid | ushort | input | Link identifier |

### Description

This function writes the Q_PKT structure to the ENET Driver process. The queue is identified in "enet_mngr[linkid].destn_fd". It calls the "write()" system call. This function returns one of the following values:

- The number of the byte written to the queue if the function completes successfully.
- 0 if the driver/link identified by "linkid" is not active
- –1 if there is an error during the "write()" system call. The global variable "errno" will contain more information about the error.

## Spawn Another Driver Process

### Name

int create_comm(char path[], char name[], ushort linkid)

| Parameter | Type | Use | Description |
|---|---|---|---|
| path | char | input | "./enetdrvr" for ENET (or file name and directory path) |
| name | char | input | "enetdrvr" for ENET |
| linkid | ushort | input | Link identifier |

### Description

This function spawns another ENET Communications Driver Process. It sets and opens the FIFO files and returns one of the following values:

- 1 (DONE) if the function completes successfully
- 0 (FAILED) if the function fails

## Initialize ENET Driver

### Name

int init_enet(ushort linkid, char *config, char *path, char *program)

### Data Structure(s)

Q_PKT ipdata *ipptr;
DATA_PKT thisdata *dataptr;
CMD_PKT thiscmd *cmdptr;
ACK_PKT thisack, *ackptr;
ERROR_PKT thiserror, *errorptr;
LINK_STATUS thisstatus, *statusptr;

| Parameter | Type | Use | Description |
|---|---|---|---|
| linkid | ushort | input | Link ID |
| *config | char pointer | input | ENET configuration file name |
| *path | char pointer | input | ENET driver pathname |
| *program | char pointer | input | ENET driver program name |

### Description

This function performs the following tasks:

1. Initializes the global pointers “ipptr”, “dataptr”, “cmdptr”, “ackptr”, “errorptr”, and “statusptr.”
2. Creates the ENET driver process.
3. Activates the ENET link.

## Stop ENET Driver

### Name

int stop_enet(ushort linkid)

| Parameter | Type | Use | Description |
|---|---|---|---|
| linkid | ushort | input | Link ID |

### Description

This function sends a deactivate, deinitialize, and terminate message to the ENET Driver process, which is indicated by linkid (the logical link number).

## Initialize Link Manager

### Name

void init_linkmngr(void)

### Data Structure(s)

APP_ENET enet_mngr[], *mngrprt

### Description

This function initializes the link manager structures. The process IDs are initialized to 0 indicating that the process has not been created. The link state is set to deinitialized (LINK_DEINIT).

## Process ENET Driver Commands

### Name

void process_cmd(int command, ushort linkid)

| Parameter | Type | Use | Description |
|---|---|---|---|
| command | int | input | A valid ENET Driver command |
| linkid | ushort | input | Link ID |

### Description

This function invokes a ENET Communications Driver "command." The commands used by this function are as follows:

```
CREATE_DRVR
TERM_DRVR
LINK_INIT
LINK_DEINIT
LINK_ACTIVATE
LINK_DEACTIVATE
GET_STATUS
SEND_PARAMS
SHOW_LINKS
```

This function calls "create_drvr()," "terminate_drvr()," "send_drvr_cmd()," and "show_process()." The Link ID should reflect the logical link you want the command to operate on.

## Send ENET Driver Commands

### Name

void send_drvr_cmd (byte command, ushort linkid)

| Parameter | Type | Use | Description |
|---|---|---|---|
| command | byte | input | A valid command packet identifier |
| linkid | ushort | input | Link ID of driver process |

### Description

This function sends "commands" to the ENET Communications Driver process. The commands sent by this function are as follows:

```
LINK_INIT
LINK_DEINIT
LINK_ACTIVATE
LINK_DEACTIVATE
GET_STATUS
SEND_PARAMS
```

This function calls "get_qpkt()" to prepare the command Q_PKT, and then send Q_PKT by calling "send_toq()."

## Create ENET Driver Process

### Name

void create_drvr(ushort linkid, char *path, char *name)

| Parameter | Type | Use | Description |
|---|---|---|---|
| linkid | ushort | input | Link ID of driver process |
| *path | charpointer | input | "./enetdrvr" for ENET (or file name and directory path) |
| *name | charpointer | input | "enetdrvr" for ENET |

### Description

This function creates the ENET Communications Driver process, sets up the communication queue, and updates the link manager data structure.

## Display ENET Attributes

### Name

void show_process(ushort linkid)

| Parameter | Type | Use | Description |
|---|---|---|---|
| linkid | ushort | input | Link ID of driver process |

### Description

Displays the attributes of the running process associated with the logical link number. The process ID of the ENET Communications Driver process, source and destination communication queue names, respective file descriptors, and the state of the link are shown.

## Read Data from ENET Link

### Name

int enet_read(int linkid, unchar *buf, int *len)

### Data Structure(s)

Q_PKT *ipptr

| Parameter | Type | Use | Description |
|---|---|---|---|
| linkid | int | input | ENET Driver/link identifier |
| buf | pointer, unsigned char | input | Receive buffer |

| Parameter | Type | Use | Description |
|---|---|---|---|
| len | pointer, int | input | Maximum length (in bytes) of “buf” |
| len | pointer, int | output | Actual length of received data |

### Description

This function performs the following functions:

- Reads the receive queue for messages from the ENET/link identified by “linkid.” If a DATA message is read, it copies the received message into “buf” and updates “len” with actual message length. If another message type is read, no copying takes place.
- Returns the packet type: NULL, DAT_TYPE, STATS_TYPE, ERROR_TYPE, or ACK_TYPE.
- Returns –1 if the length of the received message is longer than the buffer “len” (DATA_TYPE only).

## Write Data to ENET Link

### Name

int enet_write(int linkid, unchar *buf, int len)

### Data Structure(s)

Q_PKT *ipptr

| Parameter | Type | Use | Description |
|---|---|---|---|
| linkid | int | input | ENET Driver/link identifier |
| buf | pointer, unsigned char | input | Transmit buffer |
| len | int | input | Length (in bytes) of “buf” |

### Description

This function performs the following functions:

- Copies “len” bytes from “buf” into the correct Q_PKT structure and writes it to the ENET driver/link identified by “linkid.”
- Returns the number of bytes written to the driver upon successful completion.
- Returns 0 if the link identified by “linkid” is not active.
- Returns –1 if there is a write error.
- Returns –2 if the length of “buf” is greate than MAX_MSG_LENGTH.

# Create FIFO

### Name

int makefifo(char *path)

| Parameter | Type | Use | Description |
|---|---|---|---|
| *path | char | input | FIFO pathname |

### Description

Communication is facilitated between the host application and the ENET Communications Driver via a data packet passing scheme involving two FIFOs: a read FIFO and a write FIFO. This function creates the FIFO indicated by “path” and returns the value from the mknod (make node) call.

# Create FIFO Name

### Name

char *fifoname(char *prefix, long key)

| Parameter | Type | Use | Description |
|---|---|---|---|
| *prefix | char | input | Always “src_” for host’s read FIFO and “dst_” for its write FIFO. |
| key | long | input | The process ID of the associated ENET driver. |

### Description

This function creates a file name for a FIFO using prefix and key arguments. These arguments are preceded by “/tmp/” in the FIFO name. This function returns the file name of the desired FIFO.

# Terminate ENET Communications Driver Process

### Name

void terminate_drvr(ushort linkid)

| Parameter | Type | Use | Description |
|---|---|---|---|
| linkid | ushort | input | Link ID of driver process |

### Description

Used by the host application to terminate the communication driver process associated with the link ID. The driver process closes the socket and FIFO communications and then ends.

# ENET Communications Driver Data Structures and Constants

**Note** The ENET Communications Driver is accessed via the ENET Link Manager. The descriptions that follow are provided for informational purposes. This module also uses all the data structures found in "enet_if.h."

File Name: enet.h

```
/* structure for host side global daa */

typedef struct {

              char           read_fifo_name [256];
              char           write_fifo_name [256];
              int            read_fifo_fd;
              int            write_fifo_fd;
              int            link_no;
              int            connected

}HOST;

/* structure for Ethernet side global data */

typedef struct {

              char           name [256];
              char           official_name [256];
              int            local _port;
              int            remote_port;
              int            status;

} LINK;

/* structure for this application global data */

typedef struct {

              int            pid;
              char           LogFile [256];
              int            LogLevel;
              char           cfg_name [256];

} DRVR;

typedef char string 10 [11];
```

```
/*
```

host message structures

This is the set of possible host application data structures which may be used to transfer data to and from the interface manager.

```
/*


enum host_message_types {
```

SDS_RECORD, /* the host application uses SDS message record */

```
};


/*
```

link_type

This is the set of possible SDA interfaces, or links, which may be used to connect to the switch.

```
*/



enum link_type {
                CLOSED,                 /* the link is not open */
                TCP_LINK                /* TCP_IP socket */
};


#define BROADCAST_BCA 0xDF
#define MAX_MESSAGE_LEN 512
#defineMIN_ADLC_PACKET 6
#define MIN_MESSAGE_LEN 6


#define NET_SIDE_A                      0
#define NET_SIDE_B                      1
#define HOST_CMD_A2                     2
#define HOST_RPT_A                      3

/* received message states */
```

```
#define MSG_NOTOK0                      0
#define MSG_OK                          1
#define ACCEPT_MSG                      2
#define NO_SPACE                        3


/* the data structure of the transfer buffer */

typedef struct

               unsigned short length;   /* message length */
               unsigned char data       /* message date*/
               [MAX_MESSAGE_LEN];

}
tcp_xfr_type;



typedef unsigned char byte_array [MAX_MESSAGE_LEN];          /* the raw data */



typedef union xbfr {

         tcp_xfr_type tcp;            /* TCP message */

         byte _array raw;

{ msg_buffer_type;

#define A_SIDE_ACTIVE 0x01
#define B_SIDE_ACTIVE 0x03
#define A_SIDE_STANDBY 0x00
#define B_SIDE_STANDBY 0x02


/* network status values */
#define NOTOK                       0
#define OK                          1
#define LINK_IS_BROKEN              -1
#define LRC_CHECK_ERROR             -2
#define LINK_LOGIC_ERROR            -3
#define INVALID_LINK_TYPE           -4
#define TOO_MANY_RETRIES            -5
#define MESSAGE_CHECK_ERROR         -6
#define NO_HOST_APPLICATION         -7

#define MAX_MSG_SIZE                256

extern int Log_Level;               /* The amount of logging performed */

extern int errno;
```

# ENET Communications Driver Functions

The ENET Communications Driver Process, enetdrvr, consists of 28 functions in 5 source code modules: control.c, datapath.c, enetdrvr.c, host.c, and network.c.

The following are the function names, synopsis and global data in the control.c file.

## control.c Names

char *fifoname, int read_configuration, int get_token, print_structs

## control.c Synopsis

```
#include <stdio.h>
#include <fcntl.h>
#include <poll.h>
#include <malloc.h>
#include "types.h"
#include "enet.h"
int read_configuration();
int get_token();
void print_structs();
char *fifoname();
```

## control.c Global Data

```
extern HOST             host;
extern LINK             link [ ];
extern DRVR             drvr;
extern struct poll fd   connection [ ];
```

The following are the function names, synopsis and global data in the datapath.c file.

## datapath.c Names:

process_report, data_path_control, stop_data_path_control

## datapath.c Synopsis:

```
#include <stdio.h>
#include <errno.h>
#include <poll.h>
#include <sys/time.h>
#include "enet.h"
#include "types.h"
#include "enet_if.h"

#define POLL_TIMEOUT   1000
    void               process_report ( );
    int                data_path_control ( );
    void               stop_data_path_control ( );
    int                check (report);
```

## datapath.c Global Data

```
extern HOST  host;
extern LINK  link[];
extern DRVR  drvr;
extern char  LogStr[];
extern int  shut_down;
extern int  active_network;
extern struct pollfd  connection[4];
extern Q_PKT  ipdata, *ipptr;
```

The following are the function names, synopsis and global data in the enetdrvr.c file.

## enetdrvr.c Names

```
main
```

## enetdrvr.c Synopsis

```
#include <stdio.h>
#include <signal.h>
#include <poll.h>
#include <fcntl.h>
#include "enet.h"
#include "types.h"
#include "enet_if.h"

/* function prototypes */

     int main( );

/* external function prototypes */

     int               parse_arguments ( );
     int               data_path_control ( );
     void              stop_data_path_control ( );
```

## enetdrvr.c Global Data

```
HOST  host;
LINK  link[2];
DRVR  drvr;
Q_PKT  ipdata,  *ipptr;
DATA_PKT  thisdata, *dataptr;
CMD_PKT  thiscmd, *cmdptr;
ACK_PKT  thisack, *ackptr;
ERROR_PKT  thiserror, *errptr;
LINK_STATS  thisstatus, *statptr;
PARAMS  paras, *paraptr;
```

The following are the function names, synopsis and global data in the host.c file.

## host.c Names

handle_que, send_toque, ack_toque, error_toq, stat_toq, setup_que, read_que, close_host

## host.c Synopsis

```
#include <stdio.h>
#include <errno.h>
#include <fcntl.h>
#include <stropts.h>
#include <poll.h>
#include <sys/types.h>
#include "enet.h"
#include "types.h"
#include "enet_if.h"

#define MAX_QTRIES   10       /* max attempts to open fifos */



/* function prototypes */

       int        handle_que( );
       int        send_toque( );
       void       ack_toque( );
       void       error_toq( );
       void       stat_toq( );
       int        setup_que( );
       int        read_que( );
       int        close_host( );

/* external function prototypes */

       extern char  *fifoname();
```

## host.c Global Data

```
extern HOST           host;
extern LINK           link[];
extern DRVR           drvr;
extern Q_PKT          ipdata,*ipptr;
extern DATA_PKT       thisdata,*dataptr;
extern CMD_PKT        thiscmd,*cmdptr;
extern ACK_PKT        thisack,*ackptr;
extern ERROR_PKT      thiserror,*errptr;
extern LINK_STATS     thisstatus,*statptr;
extern PARAMS         paras, *paraptr;
```

The following are the function names, synopsis and global data in the network.c file.

## network.c Names

print_netstat, print_sockaddr, open_network, open_network_link,reopen_network_link, close_network, close_network_link,read_network_link, write_network_link, network_receive, pass_to_network, check_report

## network.c Synopsis

```
#include <stdio.h>
#include <errno.h>
#include <fcntl.h>
#include <termio.h>
#include <poll.h>
#include <signal.h>
#include <time.h>
#include <sys/types.h>
#include <sys/time.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include "types.h"
#include "enet_if.h"
#include "enet.h"

typedef unsigned char link_packet[258];

typedef struct sockaddr_in SOCK_INET;    /*just a typedef of sockaddr_in */



/* function prototypes */

        void  print_netstat( );
        void  print_sockaddr( );
        int  open_network_link( );
        void  reopen_network_link( );
        int  close_network_link( );
        int  read_network_link( );
        int  write_network_link( );
        int  open_network( );
        int  close_network( );
        int  network_receive( );
        int  pass_to_network( );
        int  check_report( );

/* external function prototypes */

        int  get_token ();
```

## network.c Global Data

```
extern HOST  host;
extern LINK  link[2];
extern DRVR  drvr;
extern char  LogStr[];
extern int  shut_down;
extern int  link_debug;
extern int  active_network;
```

# Create FIFO Name

### Module: control.c

### Name

char *fifoname(char *prefix, longkey)

| Parameter | Type | Use | Description |
|---|---|---|---|
| *prefix | char | input | A character string to prefix the file name |
| key | long | input | A unique ID for this file name |

### Description

This function returns a file name for the FIFO using the prefix and key arguments. The prefix argument is always either “src_” for the host applications read FIFO, or “dst_” for the host applications write FIFO. The key argument is the process ID number for the associated communications driver. These arguments are preceded by “/tmp” in the FIFO name. This function returns a pointer to the character string containing the file name.

# Read Configuration File

### Module: control.c

### Name

int read_configuration(char *name)

| Parameter | Type | Use | Description |
|---|---|---|---|
| *name | char | input | A valid UNIX file name |

### Description

This function reads the configuration file which is pointed to by “name.” The specified items in the configuration file are read in to the appropriate global variables contained in the driver program.

### Returns

This function returns 0 (TRUE) if the file exists and contains valid data, –1 (FALSE) if the file does not exist or the data contained in the file is invalid.

# Get Token

### Module: control.c

### Name

int get_token(FILE *fil, char *tok, int siz)

| Parameter | Type | Use | Description |
|---|---|---|---|
| *fil | char pointer | input | An opened file pointer |
| *tok | char pointer | input | Return value of the next token read |
| siz | int | input | Size of the token |

### Description

This function is used by “read_configuration” to read individual portions of the configuration file.

### Returns

This function returns the following values:

- 1 if tok contains a token
- 0 if the end of the file is reached
- –1 for any other error

# Print Driver Structures

### Module: control.c

### Name

void print_structs(void)

### Description

This function prints the contents of the three major structures (extern HOST, extern LINK, and extern DRVR) of the ENET Communications Driver process to standard output.

# Process Report

### Module: datapath.c

### Name

void process_report(int net, unsigned char *msg, int len)

| Parameter | Type | Use | Description |
|---|---|---|---|
| net | int | input | The network from which the report came |
| *msg | uchar | input | Pointer to the report data |
| len | int | input | Length of report data |

### Description

This function takes an incoming report from the VCO/4K and sends it to the application by placing it in the FIFO. This function also checks for redundant switchover and adjusts its global variables so future commands are sent to the correct TCP link.

## Control Data Path

### Module: datapath.c

### Name

int data_path_control(void)

### Description

This function polls the TCP and FIFO links for data awaiting processing. If there is data on the link, this function calls the appropriate functions to parse and send the data to its destination. Polling is continuous as long as the global “shut_down” variable is set to –1 (FALSE).

### Returns

This function always returns 0 (TRUE).

## Stop Data

### Module: datapath.c

### Name

void stop_data_path_control(int sig, int code, struct sigcontext *scp, char *addr)

| Parameter | Type | Use | Description |
|---|---|---|---|
| sig | int | input | Information sent by the signal handler |
| code | int | input | Information sent by the signal handler |

| Parameter | Type | Use | Description |
|---|---|---|---|
| *scp | struct sigcontext | input | Information sent by the signal handler |
| *addr | char | input | Information sent by the signal handler |

### Description

This function is used for the three UNIX signals that are caught and processed by the driver process (SIGTERM, SIGQUIT, and SIGINT). It sets the global “shut_down” variable to 0 (TRUE), which sets up a graceful shutdown of the driver.

## Main

### Module: enetdrvr.c

### Name

int main(int argc, char**argv)

| Parameter | Type | Use | Description |
|---|---|---|---|
| argc | int | input | Standard C command line argument |
| **argv | char | input | Standard C command line argument |

### Description

This function is the main function of the enetdrvr process. It is responsible for initializing global variables, setting up the signal handlers and starting the polling loop. When the polling is complete, it shuts down the network and host connections and returns to the operating system.

### Returns

This function returns the status of “data_path_control,” which is always 0 (TRUE).

## Handle Application Output Queue

### Module: host.c

### Name

int handle_que(void)

### Description

This function is called by the polling rouine when data is available on the host applications output queue. This packet is read and the proper actions are taken depending on the type of packet sent. In general, a data packet contains a command to the VCO/4K, a command packet indicates the host wants the driver to perform a specific action.

### Returns

This function returns either a valid pkt_type, or a NULL if nothing is read.

## Create Q_PKT

### Module: host.c

### Name

int send_toque(void)

### Description

This function writes the packet in the global Q_PKT ippr structure to the host applications FIFO queue.

### Returns

This function returns OK if the data is written. Otherwise, NOTOK.

## Send ACK_PKT

### Module: host.c

### Name

void ack_toque(byte ack)

| Parameter | Type | Use | Description |
|---|---|---|---|
| ack | byte | input | One of the valid commands (PORT_INT, PORT_DEINT, etc.) |

### Description

This function is used by the ENET Communications Driver software to send an ACK packet to the host application. ACK packets are sent to acknowledge to the host application that its command was carried out.

# Send ERROR_PKT

### Module: host.c

### Name

void error_toque(byte ack)

| Parameter | Type | Use | Description |
|---|---|---|---|
| error | ushort | input | A valid error code |

### Description

This function is used by the ENET Communications Driver software to send an error packet to the host application. An error packet is sent to notify the applications that a command it sent has failed. Valid error packet types can be found in enet_if.h.

# Send Status

### Module: host.c

### Name

void stat_toq(void)

### Description

This function is used by the ENET COmmunications Driver to send statistics about its links to the host applications process in response to a SEND_STATS command. Statistics are reset to zero (and begin accumulating again) after they are sent to the host applications.

# Open Read/Write FIFOs

### Module: host.c

### Name

int setup_que(void)

### Description

This function instructs the ENET Communications Driver to open the host applications FIFOs for reading and writing. The queues must have been created by the host application prior to calling this function.

### Returns

This function returns OK if the queue is opened successfully. Otherwise, NOTOK.

## Read Packet from Host Application Queue

### Module: host.c

### Name

int read_que(void)

### Description

This function is used by the ENET Communications Driver to read one packet from the host application's output queue. It is used internally by "handle_que."

### Returns

This function returns DONE if data has been read. Otherwise, NULL.

## Close Driver Connection

### Module: host.c

### Name

int close_host(void)

### Description

This function is used to close the enetdrvr process connection to the host applications FIFOs. A message is sent to the log file to indicate this function has been called.

### Returns

This function always returns 0.

## Display Network Statistics

### Module: network.c

### Name

void print_netstat(int network)

| Parameter | Type | Use | Description |
|---|---|---|---|
| network | int | input | The network side for which the statistics are being displayed. |

### Description

This function is used to print the log file various statistics about the network TCP connection to standard output. The statistics include connection type, packets received, number of faults, reports received, average size, and number of link responses. “network” should indicate either NET_SIDE_A or NET_SIDE_B.

## Log Socket Information

### Module: network.c

### Name

void print_sockaddr(SOCK_NET *sock, char *lab)

| Parameter | Type | Use | Description |
|---|---|---|---|
| *sock | SOCK_NET | input | The socket for which the data is being printed |
| *lab | char | input | Text label |

### Description

This function prints information about the specific socket into a log file. Information includes the family, port, and address of the socket. “lab” is a user-definable string, which adds information to the file output.

## Open Driver Socket

### Module: network.c

### Name

int open_network(int side)

| Parameter | Type | Use | Description |
|---|---|---|---|
| side | int | input | The network side to open |

### Description

This function is used to set up the TCP sockets for the ENET Communications Driver to talk to the VCO/4K system. Two sides can be opened for redundant systems. (this feature is set in the configuration file.) “side” should indicate either NET_SIDE_A or NET_SIDE_B.

### Returns

If this function is successful, a 0 is returned. Otherwise, a –1 is returned.

## Open TCP Links

### Module: network.c

### Name

int open_network_link(int side, struct pollfd *pfd)

| Parameter | Type | Use | Description |
|---|---|---|---|
| side | int | input | The network side to open |
| *pfd | struct pollfd | input | The poll structure of the UNIX system |

### Description

This function is used internally by the open_network() to open the TCP links to the VCO/4K system. “*pfd” structure should be set up as indicated in he UNIX poll function intsructions. “side” should indicate either NET_SIDE_A or NET_SIDE_B.

### Returns

If the function completes successfully, it returns the file ID. Otherwise, it returns a –1.

## Reopen TCP Link

### Module: network.c

### Name

void repoen_network_link(int sig, int code, struct sigcontext *scp, char *addr)

| Parameter | Type | Use | Description |
|---|---|---|---|
| *sig | int | input | Information sent by signal handler |
| code | int | input | Information sent by signal handler |

| Parameter | Type | Use | Description |
|---|---|---|---|
| *scp | struct | input | Information sent by signal handler |
| *addr | char | input | Information sent by signal handler |

### Description

This function is called when the VCO/4K system sends a signal indicating that the TCP link is down. A 90-second signal alarm is sent to allow the Ethernet link on the VCO/4K system to reset. When the alarm is raised, it calls this function to attempt to reopen the link. If the link cannot be reopened, the link variables are cleared and a POLLING_FAIL error is sent to the host application.

## Close Network Side

### Module: network.c

### Name

int close_network(int side)

| Parameter | Type | Use | Description |
|---|---|---|---|
| side | int | input | The network side to close |

### Description

This function is used to stop all communications from the ENET Communications Driver to the VCO/4K system. “side” should be either NET_SIDE_A or NET_SIDE_B.

### Returns

If the network side is closed successfully, this function returns a 0. Otherwise, it returns a –1.

## Close Network Link

### Module: network.c

### Name

int close_network_link(struct pollfd *pfd)

| Parameter | Type | Use | Description |
|---|---|---|---|
| *pfd | struct | input | The poll structure for the link |

### Description

This function is used internally by “close_network” to handle the mechanics of closing the network link.

### Returns

If the link is closed successfully, this function returns a 0. Otherwise, it returns a –1.

## Read TCP Link

### Module: network.c

### Name

int read_network_link(int fd, unsigned char *msg, int size)

| Parameter | Type | Use | Description |
|---|---|---|---|
| fd | int | input | The file ID of the link |
| *msg | uchar pointer | input | Pointer to the buffer which will receive the data |
| size | int | input | The size of the buffer message |

### Description

This function is used internally by “network_receive.” It handles the mechanics of reading data from the TCP link and storing it in a buffer for processing.

### Returns

If the function completes successfully, it returns the number of bytes stored. Otherwise, it returns a –1.

## Write TCP link

### Module: network.c

### Name

int write_network_link(int fd, unsigned char *msg, int size)

| Parameter | Type | Use | Description |
|---|---|---|---|
| fd | int | input | The file ID of the link |

| Parameter | Type | Use | Description |
|---|---|---|---|
| *msg | uchar pointer | input | Pointer to the buffer from which to write |
| size | int | input | The number of bytes from the buffer to write |

### Description

This function writes “size” number of bytes from the msg buffer to the VCO/4K system which is pointed to by the fd descriptor.

### Returns

If the function completes successfully, it returns 0. Otherwise, it returns a –1.

## Send VCO Report

### Module: network.c

### Name

int network_receive(int network, unsigned char *message, int size)

| Parameter | Type | Use | Description |
|---|---|---|---|
| network | int | input | The network side from which to read |
| *message | unsigned char | input | The buffer that will hold the data read |
| size | int | input | The size of the buffer |

### Description

This function transfers a single report from the VCO/4K system into the buffer pointed to by “*message.” “network” should indicate either NET_SIDE_A or NET_SIDE_B.

### Returns

This function returns one of the following values:

- The number of bytes read
- –1 (LINK_IS_BROKEN) if the read fails
- 0 if nothing is read

# Send Message to VCO

### Module: network.c

### Name

int pass_to_network(int network, unsigned char *message, int length)

| Parameter | Type | Use | Description |
|---|---|---|---|
| network | int | input | The network side to which to write |
| *message | unsigned char | input | The buffer containing the data to write to the VCO/4K system |
| length | int | input | The number of bytes to write to the message buffer |

### Description

This function writes a command to the VCO/4K system indicated by “network.” Before sending the message, this function checks the length of the message. If the message is too long for the buffer, the message is not sent. The function does not perform any other tasks on the message; it is sent unchanged. “side” should indicate either NET_SIDE_A or NET_SIDE_B.

### Returns

This function returns one of the following values:

- 0 if the write is successful
- “LINK_IS_BROKEN” if the write fails
- “MSG_2LONG” if the message is greater than “MAX_MSG_LEN”

# Check for Report

### Module: network.c

### Name

int check_report(int network, unsigned char *msg_data, int size)

| Parameter | Type | Use | Description |
|---|---|---|---|
| network | int | input | The network side from which the report data is collected |
| *msg_data | unsigned char | input | The buffer which will hold the report |
| size | int | input | The size of the msg_data buffer |

### Description

This function checks the VCO/4K system pointed to by “network” to see if data is available. “side” should indicate either NET_SIDE_A or NET_SIDE_B.

### Returns

If this function is successful, it returns the number of bytes received from the VCO/4K. If an error occurs, the function returns a 0 or a –1.

# ENET Utilities Data Structures and Constants

This module uses the same data structures and constants as the ENET Link Manager.

# Ethernet Utilities Functions

The utilities module, enet_util.c, contains seven functions which are used by both the ENET Link Manager and ENET communications driver.

## Open and Write to File

### Name

int logMsg(char *log_file, char *msg)

| Parameter | Type | Use | Description |
|---|---|---|---|
| *log_file | char | input | A valid UNIX file name |
| *msg | char | input | Text string to write to file |

### Description

This function attempts to open the file name passed in “*log_file” and then writes into the file the string pointed to by “*msg.” This function is included for compatibility with the ADLC product.

### Returns

If this function completes successfully, it returns 0. Otherwise, it returns –1.

## Convert Hex to ASCII

### Name

unchar hex2ascii(unchar hexval)

| Parameter | Type | Use | Description |
|---|---|---|---|
| hexval | unchar | input | A hexadecimal value |

### Description

This function returns the ASCII character equivalent to the value passed to it in “hexval.” This function is included for compatibility with the ADLC product.

### Returns

The ASCII equivalent of “hexval.”

## Convert ASCII to Hex

### Name

unchar ascii2hex(int asciival)

| Parameter | Type | Use | Description |
|---|---|---|---|
| asciival | int | input | An ASCII character |

### Description

This function returns the hexadecimal value that is equivalent to the character passed to it in “asciival.” This function is included for compatibility with the ADLC product.

### Returns

The hexadecimal value of “asciival.”

## Convert ASCII String into Hexadecimal Byte Stream

### Name

int str2hex(char *str, uchar *buf)

| Parameter | Type | Use | Description |
|---|---|---|---|
| *str | char | input | A string of ASCII characters |
| *buf | uchar | input | Returns the hexadecimal byte stream |

### Description

This function translates the ASCII string pointed to by "*str" into a hexadecimal byte stream. This function is included for compatibility with the ADLC product.

### Returns

The number of bytes in the "buf" array.

## Convert Hexadecimal Byte Stream into ASCII String

### Name

char *hex2str(uchar *buf, int buflen)

| Parameter | Type | Use | Description |
|---|---|---|---|
| *buf | uchar | input | The hexadecimal byte array |
| buflen | int | input | Number of bytes to translate from "buf" |

### Description

This function translates the hexadecimal byte stream located in the buffer pointed to by "*buf" into an ASCII string. "buflen" specifies how many bytes in "buf" to translate. It inserts a blank character between every two digits in the output string. It calls the "malloc()" function to allocate space for the ASCII string.

### Returns

This function returns a pointer to the ASCII string.

## Display Q_PKT Contents

### Name

void show_qpkt(Q_PKT *qpktp)

| Parameter | Type | Use | Description |
|---|---|---|---|
| *qpktp | Q_PKT | input | Packet to display |

### Description

This function displays the contents of the Q_PKT packet (pointed to by "*qpktp" and identified by the pkt_type member). It displays a detailed breakdown of the packet on the standard output.

# Output ASCII Representation of Hexadecimal Byte Stream

## Name

void displayHex(uchar buf, int buflen)

| Parameter | Type | Use | Description |
|---|---|---|---|
| buf | uchar | input | Buffer of byte stream to display |
| buflen | int | input | Number of bytes to display |

## Description

This function displays on the standard output the ASCII representation of the hexadecimal byte stream pointed to by “buf” up to “buflen” bytes. It inserts a blank character after each displayed byte. This function is standard output to check its contents.