EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH

CERN - SL Division

SL/Note 95-108 (BT)

BROADCASTING MACHINE TIMING EVENTS ON LYNXOS

A. Burton* and J. H. Dieperink

Abstract

Recently, the process controllers of the LEP separators have been migrated from XENIX PCs and OS-9 VME systems to LynxOS PCs, now used in all SL/BT applications. A new TG3-PC interface is being used to receive general machine timing events. These are interpreted and broadcast over the MIL-1553B bus to the G-64 Equipment Control Assemblies (ECAs) by means of a new server program which is the subject of this document. This facility is used in parallel to the existing timing distribution system in the G-64 ECAs thus improving the reliability of the separator system and simplifying fault diagnostics. This work has been performed in the framework of the technical student program.

Geneva
24 October, 1995

(SL/BT/Notes.'95)

* Now at Bournemouth University, Dorset, UK.

Table of Contents

1. Introduction

Machine timing events are used to synchronously start voltage changes on the LEP separator electrodes in a selected number of collision points. They are generated on operator request by the General Machine Timing (GMT) system and distributed in LEP in the form of messages over a twisted pair multi-drop timing line.

At present in each collision point a TG3 interface card located in a G-64 Equipment Control Assembly (ECA) receives and decodes these messages. The decoded messages are then broadcast to all other ECAs on the MIL-1553B bus which control the operation of the separator equipment.

Until the shutdown 1994/95 the local host in each collision point was an IBM compatible 386 PC using the XENIX operating system. This has been replaced by an industrial 486 PC running the LynxOS operating system. This combination allows the use of the newly developed TG3-PC interface card [1]. It is physically located in the LynxOS PC and has the same functionality as the original TG3-G64 interface card.

The advantages of using, in addition, the TG3-PC interface for the distribution of the timing events to the separator equipment are:

- The possibility to improve the diagnostics in the day-to-day running of the separator systems. For instance the post-mortem table in an ECA must contain two entries for the same event, one coming from the TG3-PC and the other from the TG3-G64. Local timing problems can be isolated using the entries in the post-mortem table.

- The possibility of logging of status and error messages to a log file provides a history of the received events and the hardware status of the equipment.

- Alarms can be sent to the SL alarm system when errors occur, giving an early indication of possible timing problems.

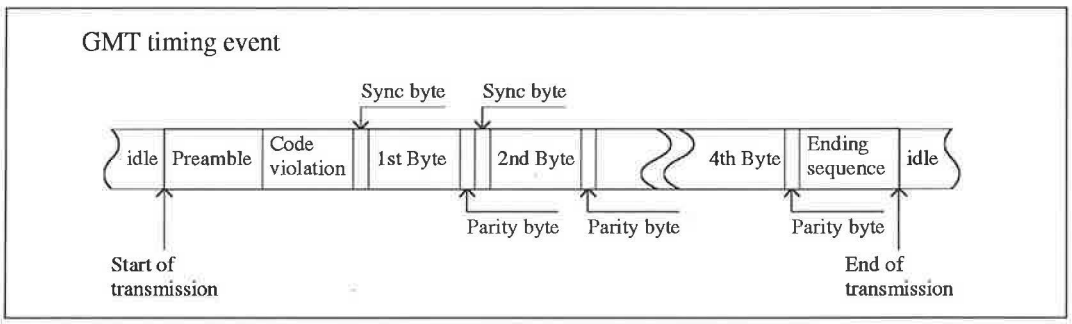
In order to profit fully from these new diagnostic tools a server application for the TG3-PC has been developed. This document describes the design and operation of the server application 'zltg3'. Also it describes the installation of both the TG3-PC interface driver and the 'zltg3' program.

2. Overview of the timing system

2.1. GMT system

The GMT distributes machine timing events and 1 ms clock timing events to all equipment in LEP and SPS. Machine timing events are used to synchronise operations in LEP i.e. beam injection, start ramp, stop ramp. The 1 ms clock timing events are used to synchronise the GMT system and to validate the reception of the machine timing events.

These events are distributed as data packets to all buildings in LEP. Time compensation is used in the GMT system to guarantee that the data packets arrive at all buildings simultaneously.



2.2. TG3-PC interface card

The TG3-PC interface card consists of two parts, a Receiver part and a Process part.

2.2.1. Receiver part

The Receiver part of the TG3-PC interface card receives timing messages from the multi-drop timing line of the GMT system. It checks the Manchester encoded data packets for transmission errors, which are flagged in the Status register and Receiver error register.

The Receiver then converts and strips the data packet down to an Event Frame which is passed to the Process part. The Event Frame contains four bytes:

Header	Machine Event Name	Cycle Type	Cycle Type Number

- **Byte 1:** The Header byte which identifies the machine type SPS or LEP.
- **Byte 2:** The Machine Event Name is an 8 bit event code to represent machine timing events in SPS or LEP.
- **Byte 3:** The Cycle Type labels the particle type i.e. electron, proton, positron.
- **Byte 4:** The Cycle Type Number identifies an individual cycle of a particular Cycle Type in a supercycle.

2.2.2. Process part

The Process part of the TG3-PC interface card compares each incoming Event Frame with a set of pre-loaded parameters in the event table. When a comparison is valid the requested action, relevant to that particular condition, is initiated i.e. a hardware interrupt of the PC and/or the transmission of a trigger pulse to some external equipment.

The Event table is an area of memory 245 bytes in size. It is partitioned into 49 lines and each line is partitioned into 5 columns. The 49th line is called the Start Super-Cycle (SSC) line and is reserved for actions that are triggered by the SSC frame.

The 48 remaining lines of the Event table are divided into 6 blocks of 8 lines The first 5 blocks are programmable, with each block representing a different Event Type i.e. **proton, positron, electron, antiproton** and **heavy ion** and the sixth block has been reserved for triggering on other types of frames i.e. Super-Cycle Event, Calendar frame.

The 5 columns in each line are:

- **Column 1:** The Machine Event Number
- **Column 2:** The Cycle Type Number
- **Column 3:** The Control Word - Specifying the action carried out by the TG3-PC interface card on valid comparison i.e.

Control Word	Action
$81	Bus interrupt and Timer One with millisecond clock delay.
$D9	Bus interrupt, Front panel output, and Timer One with microsecond clock delay.

- **Column 4:** Delay Word - MSB
- **Column 5:** Delay Word - LSB

  The Delay Word consists of two bytes and specifies the length of time in milliseconds between reception of the machine timing event and the initiation of the required action.

On reception of an Event Frame only the block of 8 lines is checked, this ensures a fast response by the hardware in the TG3-PC interface card. In LEP operation only the first block is used, as the particle type is not relevant for LEP.

2.2.3. TG3-PC constraints

The TG3-PC interface card hardware is constrained in two main areas. The first is a design limitation. The MTG system can send up to 3 machine timing events in a one millisecond interval. The TG3-PC will only service the first machine timing event which gives a valid comparison with the event table. All later machine timing events in the same one millisecond interval are ignored.

One practical example of this constraint is when the TG3-PC is used in the SPS. Then the SSC event and the start SPS p+ injection cycle event (1c) occur in the same one millisecond interval. One of both gives a valid comparison and triggers an interrupt, the other event is lost. In LEP this constraint is not applicable, machine timing events are generated on operator request, which means that they occur less often and the time between them is considerably greater than one millisecond.

The second constraint is the possible degraded performance of the host platform. The complete cycle of reception, hardware interrupt, reading, processing the data and then broadcasting it, must be completed before the reception of the next machine timing event. If the host platform performance is degraded by other processes then machine timing events will be lost (see also section 3.3.2).

3. TG3-PC server program

3.1. Program overview

3.1.1. Environment

The TG3-PC server program uses the SL Control Group library of pre-written routines to communicate with the TG3-PC driver program [2, 3]. For the creation of the error logging tags the functions of the TS Tool Kit [4, 5] are used.

This driver controls all access to the TG3-PC interface card. Application programs use the system interface routines **open, read, write, select, ioctl,** and **close** to access the TG3-PC interface card. The TG3-PC interface driver is the only supported driver for the TG3-PC interface card.

The library files used by the zltg3 program are **TG3lib.a, libEquip.a** and **libprint.a** and they are located in **/usr/local/lib.** The header files used are **TG3.h, TG3driv.h, TG3tim.h, TSCglobal** and **print.h** and they are located in **/usr/local/include** and **/user/spsabt/btsoft/TS/include**.

The location of the source programs, configuration files and header file are on the fileserver in the following directories:

Source file	**/user /spsabt/btsoft/prod/zltg3/zltg3.c**
Header file	**/user /spsabt/btsoft/prod/zltg3/tg3_tim.h**
Makefile	**/user /spsabt/btsoft/prod/zltg3/Makefile**
Executable file	**/user /spsabt/btsoft/prod/bin/LynxOS/zltg3**
Configuration files	**/user /spsabt/btsoft/prod/data/leptg3.cfg**
	/user /spsabt/btsoft/prod/data/spstg3.cfg

The TG3-PC server program is loaded from a central file server during the system boot (see section 3.2), it is designed to run as a background daemon process and requires no user interface during operation.

During loading operator configurable program parameters specify :

-a path\filename	: The path and file name for the event configuration table.
-b path\filename	: The path and file name of the error log file.
-c hostname	: The name of the TS server host.
-v	: Verbose mode. The status and event messages are printed to the screen of the host.
-l	: LEP operation.
-s	: SPS operation.

Status and error logging is provided to aid problem solving in the installation and operation of the TG3-PC server program. A user interface is displayed to help the user when incorrect program parameters are entered.

3.1.2. A brief description

The TG3-PC server program is organised into two main parts i.e. **Initialisation** and **Event handling.** These are further divided into functional modules of code which call the various library routines to access the TG3-PC interface card.

The methodology of the server program is first to initialise the interface. This includes testing for correct functionality and loading the event table with the required event data. Next the Event handling is carried out in three phases, **select, read,** and **broadcast.** A brief description follows.

The program enters a forever loop and uses the **select** system call to wait on events. When a valid event comparison occurs, the **select** call returns a positive integer. This then initiates the **read** call to get the event number from the TG3-PC interface. The broadcast phase repackages the event number and sends it to the MIL-1553B controller.

A time-out in the **select** call allows the program to check the hardware status of the TG3-PC interface and if an error is present at this time an error message will be written to a log file. The **select** call is then re-initialised and called again. This forever loop terminates correctly when the a **safe_exit** routine is called.

When the server program detects error conditions, an entry is made in the log file and an alarm is sent to the SL alarm system. The event number is also written to the log file.

3.1.3. TG3 data structures

The data structures and definitions for the TG3 can be found in the LynxOS driver functional description and user's guide. This outlines the main data structure for the driver which is the union **iob,** and which is defined in the header **tg3.h**. The interface definitions are defined in the header **tg3driv.h**.

The header file **tg3_tim.h** defines the main TG3 server program data structure **tg3_descriptors** as seen below.

```
struct alarm                    /* TSserver alarm */
        {
         unsigned char   ServerHostName[32];
         unsigned char   Message[32];
        };

struct tg3_descriptors          /* file descriptors */
        {
         unsigned        char fd;
         unsigned        char maxfd;
         unsigned        char tg3nb;
         char            config_file[128];
         char            error_file[128];
         char            host_system_name[32];
         struct alarm    TSalarm;
        };
```

3.2. Initialisation

The TG3-PC interface is initialised by the TG3-PC server program and is as follows:

- The error logging tags in the TSserver are created. These are the **SystemAlarmName,** the **ErrorValue**, the **SystemAlarmMessage,** and the **AlarmMessage**.
- The print daemon is initialised for the error logging file.
- The file descriptors referencing the TG3-PC interface card in the LynxOS are initialised.
- The TG3-PC interface card is disabled synchronously i.e. the disable action is performed immediately upon receiving the disable command. The process part will now ignore all Event Frames passed by the receiver part.
- The TG3-PC interface card is tested for correct operation. This action also clears the event table, Receiver Error register, and Interrupt Vector register.

- The Event Table is then loaded with event parameters, i.e. Machine header (LEP or SPS), event name, cycle type, and cycle type number, from the Event Configuration file on the file server . The Event Configuration files for LEP and SPS can be found in appendix B.

- The TG3-PC interface card is enabled synchronously i.e. the enable action is performed on reception of the next Super cycle frame after receiving the enable command. The process part now accepts Event Frames passed by the receiver part.

3.3. Event handling

Event handling in the host PC is carried out in three phases, **Select, Read,** and **Broadcast.**

3.3.1. Select phase

The Event handling is initiated by using the **select** call. The **select** call allows programs to wait for I/O on a number of devices (file descriptors) at the same time.

The **select** call is only implemented for a read in the LynxOS driver.

The file descriptor for reading the TG3-PC interface card is declared by the **typedef struct fd_set**, which is in the header file **types.h.** The structure is initialised in the server program by the declaration **fd_set tg3fd_set.**

if((nfound = select(tg3_desc.maxfd, &tg3fd_set, 0, 0, &tg3_select_tmo)) == MINUS)

The **select** call in the TG3 server program is initialised using the following parameters:

- **tg3_desc.maxfd** The maximum number of file descriptors in the system. This gives the maximum number of file descriptors that are examined by the **select** call.

- **&tg3fd_set** The read file descriptor. The address of the I/O descriptor is set.

- **0,0** The write and except file descriptors are set to null. They are not used in this application.

- **&tg3_select_tmo** The address of the time out value held in the structure, **struct timeval tg3_select_tmo.**

The TG3 server program is placed on the wait queue until either an event occurs or the time out value is reached.

The **select** system call returns the following values:

- **MINUS** (-1) The **select** call failed.

- **ZERO** (0) The **select** call time-out. A time-out value is set in case of no event within a specified time, i.e. indicating a problem in the GMT. The **tg3_select_tmo** is assigned a value of 20 seconds. This is larger than the expected super cycle time for SPS, and provides a convenient period for error checking in LEP.

- **PLUS** (>0) The **select** call **tg3select()** in the LynxOS Driver provides the system with a value that is different from 0 when an event is available for read.

When a machine timing event is received and a valid comparison is made with the pre-loaded parameters in the event table, the TG3-PC interface card generates a hardware interrupt in the LynxOS PC. The LynxOS operating system calls the TG3-PC interface driver to handle the interrupt.

The TG3-PC interface driver uses an interrupt handler, **tg3intr()**, to check for errors and updates the error values accordingly. The interrupt handler **tg3intr()** uses the Interrupt Vector register and Interrupt Source register to locate the row number in the event table where the event name is stored.

The event buffer is loaded with the Event Frame and the routine **new_event()** is called. This routine checks the device table for waiting client processes, i.e. **zltg3**, which is waiting for specific machine timing events. The client process **zltg3** is then taken off the wait queue.

3.3.2. Read Phase

When the **select** call returns a positive non-zero value the read procedure is called.

The read procedure uses the function **FD_ISSET()** to check that the contents of the returned file descriptor pointer **tg3_desc->fd**, are in the file descriptor set **&tg3fd_set**. This ensures that this procedure does not block the waiting for the event. The system call **read** is used to access the TG3-PC interface card.

```
if(read(tg3_desc->fd,&iob.i_rbevent, sizeof(struct rb_event)) != sizeof(struct rb_event))
```

The **read** call passes the pointer **&iob.i_rbevent** of the data structure **rb_event**, which is contained in the union **iob**, to the TG3-PC interface driver. The TG3-PC interface driver's read routine reads the event table in the TG3-PC interface card and returns the data structure rb_event which now contains the last machine timing event.

The returned data structure **struct rb_event** is checked using **sizeof(struct rb_event)**. If the comparison fails the error is logged and the program is terminated using the routine **safe_exit**.

The data structures **rb_event** and **frame** and the union **iob** contain:

```
struct rb_event
        {
         struct frame     r_event;
         long             r_ssc_num;
         long             r_cylen ;
         int              r_time;
         int              r_flags;
        };

struct frame
        {
         unsigned char    e_head;
         unsigned char    e_name;
         unsigned char    e_type;
         unsigned char    e_type_no;
        };

union iob
        {
         struct hw_stat          i_stat;
         struct dr_event         i_decl;
         struct decl_event       i_evdecl;
         struct gs_event         i_event;
         struct rb_event         i_ rbevent;
         struct tab_dump         i_dump;
         struct tstamp           i_times;
         short                   i_modes;
         long                    i_value;
         u_char                  i_date[6];
        };
```

The union **iob** is declared as a global union. The machine event number is referenced by **iob.i_rbevent.r_event.e_name.**

The event buffer only contains the last received machine timing event. If the event buffer is not read before the next machine timing event it will be over-written and the previous machine timing event will be lost. The **iob.i_rbevent.r_flags** are checked with **GS_MISSED** for missed machine timing events. This situation can occur if the host LynxOS PC is heavily loaded and the response to TG3-PC interface card interrupts are delayed.

if(iob.i_rbevent.r_flags & GS_MISSED) print(log_all,"Missed event flag set");

3.3.3. Broadcast phase

The machine event number is now held in the data structure **struct rb_event** which is passed to the broadcast function. The machine event number (**iob.i_rbevent.r_event.e_name)** is then repacked in a new data packet for two main reasons:

- To be compatible with the format of the timing broadcast messages on the MIL-1553B interface.
- To swap the byte order of the integer which is different in the Motorola processor on the ECAs compared with the processor in the host PC.

The library routine **m_broad** [6] sends the data packet to the MIL-1553B bus new bus controller (NBC) which broadcasts it to the ECAs on the MIL-1553B bus.

3.3.4. Data packet

Packet Length	Broadcast Identifier	Remote Terminal Interface Number	Machine Event Number	Time Delay	Error Checksum
byte	byte	integer	byte	integer	integer

The data packet contains:

- The Package Length is always **$09** bytes, this tells ECA the position of the Error checksum, two bytes less than the length of the package.
- The Broadcast Identifier byte is always **$AA**. It identifies that this data packet is a timing broadcast message and not another message broadcast.
- Remote Terminal Interface Number (RTI) tells the receiver ECA which device sent the broadcast. The code **$00BB** has been chosen for the new bus controller.
- The Machine Event Number identifies the current machine timing event.
- The Time Delay is not used by this application, the default value is $0000.
- The Error Checksum is used to verify the correct reception of the data packet by the ECAs.

The utility **m_broad** returns a non-zero value for broadcasting errors. This is recorded in the error log file. The return value does not influence the program operation.

3.4. Closing the program

The TG3-PC server program can be terminated in a controlled manner for the following conditions:

1. Error conditions

The occurrence of a non-recoverable hardware problem, i.e. a failure when trying to access the TG3-PC interface card or when trying to read the event configuration table. A list of error messages and possible causes is included in appendix C.

2. **Signal** calls

Signal calls are used to close the program correctly when the system is shutdown or when the process is manually killed. **Signal** calls which trap run-time hardware errors also close the program correctly. The **signal** calls declared in this application and the **signal** calls declared in the TSserver program cause conflicts. Therefore the position of these calls in the TG3-PC server program was changed, so that the **signal** calls are reinitialised after every alarm message call to the TSserver.

```
signal(SIGINT,    safe_exit);
signal(SIGQUIT,   safe_exit);
signal(SIGFPE,    safe_exit);
signal(SIGBUS,    safe_exit);
signal(SIGSEGV,   safe_exit);
signal(SIGTERM,   safe_exit);
signal(SIGKILL,   safe_exit);
```

The function **safe_exit** is used to close the server program **zltg3**, it carries out the following:

- The message "Program clean exit" is written in the error log file.
- The TG3-PC file descriptor is closed, this de-allocates system resources.
- The error log file is closed.
- The program exits.

The **safe_exit** function

```
safe_exit(struct tg3_descriptors *tg3_desc)
        {
         print(log_all,"Program clean exit");
         close(tg3_desc->fd);
         sleep(2);
         close_logfile();
         exit();
        }
```

3.5. GMT 1 ms clock

An additional feature of the TG3-PC server program is to check the reception of the GMT 1 ms clock data packets.

The Receiver part of the TG3-PC interface card monitors the 1 ms clock timing packets on the GMT. When an error is detected the Receiver error register is flagged, the flags are:

- Bit 6: The **1 ms clock missing flag** indicates that the clock is no longer present. The flag resets when the 1 ms clock is detected again.
- Bit 7: The **watch-dog flag** indicates that one or more 1 ms clock packets have been missed since last reset. This flag is reset by the function **tg3intr()** in the TG3-PC driver.

The Receiver error register cannot generate an hardware interrupt for an error condition. The Receiver error register is therefore checked when either of two following conditions occur:

1. Select call time-out.

The TG3-PC Receiver's error register is read by the **ioctl** call **TR_GETSTATUS** and the returned byte **iob.i_stat.s_flags** is checked for the **1 ms clock missing** flag **or** the **WATCH_DOG** flag. When either of these conditions is true an error message is logged and an alarm is sent to the TSserver.

```
/*
*   Check TG3-PC Receiver's error register
*   for one milli second  'WATCH_DOG' and 'MISS1MSCLK'
*   after select time-out
*   Note:   iob.i_stat.s_flags is reset after the test
*/

int rec_reg_chk(struct tg3_descriptors *tg3_desc)
        {
        if(ioctl(tg3_desc->fd, TR_GETSTATUS, &iob) < ZERO)
                {
                print(log_all,"Unable to read Tg3-PC Reciever's error reg");
                strcpy(tg3_desc->TSalarm.Message, "Cannot Read TG3");
                return(MINUS);
                }
        else
        .if((iob.i_stat.s_flags & WATCH_DOG)||(iob.i_stat.s_flags & MISS1MSCLK))
                {
                strcpy(tg3_desc->TSalarm.Message, "GMT Clock Fail");
                iob.i_stat.s_flags = 0x00;
                return(PLUS);
        }
        return(ZERO);
        }
```

2. On every SPS SSC event.

When an SPS SSC event occurs the hardware status of the TG3-PC interface card is read by **tg3intr()** into the temporary buffers, **ds->l_wdterr, ds->t_wdterr**, **ds->l_ckon** and **ds->t_ckon**. The TG3-PC Receiver error register is then cleared by **tg3intr()**. The **ioctl** call **TR_GETSTATUS** copies these temporary buffers into the **iob.i_stat.s_laperrs**[] and **iob.i_stat.s_triperrs**[] arrays.

The **iob.i_stat.s_laperrs[1]** and **iob.i_stat.s_laperrs[2]** arrays are then checked for a **NON-ZERO** condition, if this condition is true a 1 ms clock error has occurred since the last SSC event. The hardware error message is sent to the error log file and an alarm is sent to the TSserver.

```
/*
*   Check TG3-PC Receiver's error register
*   for one milli second 'WATCH_DOG' and 'MISS1MSCLK'
*   after SSC event
*   Note:  SPS operation only
*/


        {
        if(ioctl(tg3_desc->fd, TR_GETSTATUS, &iob) < ZERO)
                {
                print(log_all,"Unable to read TG3-PC Reciever's error reg");
                strcpy(tg3_desc->TSalarm.Message, "Cannot Read TG3");
                return(MINUS);
```

```
}
 else
.if((iob.i_stat.s_laperrs[1] != ZERO) || (iob.i_stat.s_laperrs[2] != ZERO))
          {
          print(log_all,"GMT one mill-sec clock fail");
          strcpy(tg3_desc->TSalarm.Message, "GMT Clock Fail");
          return(PLUS);
          }
 return(ZERO);
}
```

3.6. Error reporting and event logging

The TG3-PC event timing program is designed to run as a background daemon process. Status and error logging are provided to aid problem solving in the installation and operation of the TG3-PC program. The machine timing events are logged in the same file as the error and status messages.

The name and location of the log file is specified by the TG3-PC program parameters:

-b /user/spsabt/btsoft/prod/log/errlog

In LEP, the TG3-PC program will run simultaneously on several LynxOS platforms. A log file directory on the file server has been reserved for all of these LynxOS platforms. Each log file has a unique name consisting of the host system name concatenated with errlog, i.e. **errlog.zxls25.**

The log file directory is: **/spsabt/btsoft/prod/log/**

A list of error messages and possible causes are included in appendix C.

3.6.1 Print() utility

The print package for the Error Log file consists of a library of pre-written C calls in **libprint.a** [7] and a header file called **print.h.** The print package provides parameter for outputting to screen or logging to file of any server or daemon program in the SL controls infrastructure.

The initialisation of the Error Log file is carried out by the **init_print** function. The use of this is outlined below:

init_print(BOOLEAN screen_output, BOOLEAN file_output, char *filename,
char *product_name, int size);

Where:

BOOLEAN screen_output	= **(P_FALSE)**	(no screen output)
BOOLEAN file_output	= **(P_TRUE)**	(file output)
char *filename	= Directory path and file name of the error log file	
char *product_name	= Host system name	
int size	= Maximum size of error log file	

The message is time stamped and has an entry [----] to identify the host system.

The printing function is: **print(log_all," Error message");**

The print output in the 'Error Log file' for the above will be:

1995-01-26-15:13:03:[host system name]: Error message

The log file is closed by the function call: **close_logfile();**

Status messages will only be logged when the verbose (**-v**) option is used in the program parameters. Error logging however is automatic.

The log file is to be limited in length to 50,000 bytes by the **int size** entry in **init_print()**. When the log file reaches this length it will be renamed with the extension **.old** and the error log file will be restarted with the original name.

3.6.2. TSserver alarm

The TG3-PC server program sends an alarm to the SL Alarm system via the Tsserver [8], when a fatal error occurs in the TG3-PC timing system. This to notify the LEP operators, who monitor the operating status of the LEP accelerator, that an error has occurred in the TG3-PC timing system.

The function call used to send an alarm is **send_tg3alarm()**. It uses the pre-written routines in **TSCglobal.h**. The procedure to send an alarm to the SL alarm system is outlined below.

The declarations for the TSserver:

seq_RTAG_DEF	**TagSet;**	Data structure defined in **TSCglobal.h**
ERR_MSG	**ErrMsg;**	TSserver returns error/status messages
int	**sid;**	TSserver identifier is not used in this application. It is used to identify individual TSservers on multi-TSserver systems as foreseen for the future [9].
long int	**ErrNr;**	Error value returned by TSserver

The example procedure to declare tag names in the TSserver is as follows:

1. First, allocate memory for the 2 tags.

```
TagSet.sequence = (RTAG_DEF *)malloc(2 * sizeof(RTAG_DEF));
TagSet.length  = 0;
```

2. Next, assign the alarm names for TagSet array (example for zxls25).

```
ProcName            zltg3_ zxls25
HostName            modena
SystemAlarmName     ZL_ zxls25_TG3_ERROR
ErrorValue          0
SystemAlarmMessage  ZL_ zxls25_TG3_MESSAGE
AlarmMessage  0
```

3. Next, test the connection to TSserver.

```
if((ErrNr = TSconnect(HostName, ProcName, &sid, ErrMsg))!= TS_NO_ERROR)
print(log_all,"Unable to connect to send alarm to TSserver: Error msg is %s",ErrMsg);
```

4. Next, copy the names to the TagSet array.

```
FCSTRCPY(TagSet.sequence[TagSet.length].Name, SystemAlarmName);
FCSTRCPY(TagSet.sequence[TagSet.length].Value, ErrorValue);
TagSet.length++;
FCSTRCPY(TagSet.sequence[TagSet.length].Name, SystemAlarmMessage);
FCSTRCPY(TagSet.sequence[TagSet.length].Value, AlarmMessage);
TagSet.length++;
```

5. Next, create the tags in the TSserver.

```
if((ErrNr = TScreTagsByName(&TagSet, ErrMsg)) != TS_NO_ERROR)
  print(log_all,"Unable to create tags in TSserver: Error msg is %s",ErrMsg);
```

6. Last, disconnect and return memory allocated.

```
TSdisconnect(ErrMsg)
free(TagSet.sequence)
```

The example procedure to send an alarm to the TSserver is as follows:

1. First, allocate memory for the 2 tags.

```
TagSet.sequence = (RTAG_DEF *)malloc(2 * sizeof(RTAG_DEF));
TagSet.length   = 0;
```

2. Next, assign the alarm value and message for TagSet array (example for zxls25).

ProcName	**zltg3_ zxls25**
HostName	**modena**
SystemAlarmName	**ZL_ zxls25_TG3_ERROR**
ErrorValue	**1**
SystemAlarmMessage	**ZL_ zxls25_TG3_MESSAGE**
AlarmMessage	**Failed TG3 Init**

3. Next, test the connection to the TSserver.

```
if((ErrNr = TSconnect(HostName, ProcName, &sid, ErrMsg)) != TS_NO_ERROR)
print(log_all,"Unable to connect to send alarm to TSserver: Error msg is %s",ErrMsg);
```

4. Next, copy the names to the TagSet array.

```
FCSTRCPY(TagSet.sequence[TagSet.length].Name, SystemAlarmName);
FCSTRCPY(TagSet.sequence[TagSet.length].Value, ErrorValue);
TagSet.length++;
FCSTRCPY(TagSet.sequence[TagSet.length].Name, SystemAlarmMessage);
FCSTRCPY(TagSet.sequence[TagSet.length].Value, AlarmMessage);
TagSet.length++;
```

5. Next, change the Tag values and message with TagsSetByName.

```
if((ErrNr = TSsetTagsByName(TagSet, ErrMsg))!= TS_NO_ERROR)
  print(log_all,"Unable to send alarm name to TSserver: Error msg is %s",ErrMsg);
```

6. Last, disconnect and return memory allocated.

```
TSdisconnect(ErrMsg)
free(TagSet.sequence)
```

The above example can be checked by using the following command at the LynxOS prompt:

```
TSreadTags modena -p ZL_z
```

- Examples:

After TScreTagsByName:	**ZL_ zxls25_TG3_ERROR**	**0**
	ZL_ zxls25_TG3_MESSAGE	**0**
After TSsetTagsByName:	**ZL_ zxls25_TG3_ERROR**	**1**
	ZL_ zxls25_TG3_MESSAGE	**Failed TG3 Init**

4. Installation

4.1. TG3 driver

The TG3 driver can be installed or de-installed dynamically. To check which driver is running on the LynxOS system the following steps must be carried out.

- Type from any directory on the host system used: **grep inst /etc/transfer.ref**

This should give a list of installed drivers i.e.

#% /usr/local/bin/instdrvr -i TG3sdesc.320.i7
#% /usr/local/bin/instdrvr -i NBC_PCdesc.bt
#% /usr/local/bin/instdrvr -i BBdesc

The TG3 Driver can have two configurations:

for the SPS: **TG3sdesc.320.i7**
or, for the LEP: **TG3ldesc.320.i7**

Note that the address of the TG3-PC interface on the PC internal bus is 320 and the number of the hardware interrupt is 7.

- Type the following to **install** the TG3 driver: for the SPS: **instdrvr -i TG3sdesc.320.i7**
  or, for the LEP: **instdrvr -i TG3ldesc.320.i7**

- Type the following to **de-install** the TG3 driver: for the SPS: **instdrvr -u TG3sdesc.320.i7**
  or, for the LEP: **instdrvr -u TG3ldesc.320.i7**

- To confirm the that the driver is installed or de-installed use the command **drivers**. This will display all current installed drivers, as the example below shows:

	id	type	major devs.	start	size	name
	0	char	1	0	0	null
	1	char	1	0	0	mem
	2	char	1	0	0	ctrl drvr
	3	char	1	0	0	Raw HD
	4	block	1	0	0	HD
	5	char	1	0	0	Raw floppy
	6	block	1	0	0	Floppy
	7	char	1	0	0	kdconsole
	8	char	2	0	0	serial
	9	char	16	0	0	pty
	10	char	1	0	0	RRD
	11	block	1	0	0	RD
	12	char	1	0	0	hbtcpip
	13	char	1	0	0	wd3e
	14	char	1	0	0	unfs
⇒	**15**	**char**	**1**	**db311070**	**13356**	**TG3driv**
	16	char	1	db2ff9c0	4940	BBdriv

The number in the 'major devs' column should be either 0 (de-installed) or 1 (installed) for **TG3driv**.

4.2. TG3-PC server program

4.2.1. Automatic installation at system boot

The following lines are included in the **transfer.ref** file used in the automatic system boot.

/user/spsabt/btsoft/prod/bin/LynxOS zltg3
/user/spsabt/btsoft/prod/bin/LynxOS zltg3 btexpert spsabt 0644 server -
%./zltg3 -l -a /user/spsabt/btsoft/prod/data/leptg3.cfg
-b /user/spsabt/btsoft/prod/log/errlog -c modena%

4.2.2. Manual installation

The location of the executable file, configuration files and the error log file are:

Executable file	**/user /spsabt/btsoft/prod/bin/LynxOS/zltg3**
Configuration files	**/user /spsabt/btsoft/prod/data/leptg3.cfg**
	/user /spsabt/btsoft/prod/data/spstg3.cfg
Error log file	**/user /spsabt/btsoft/prod/log/**

The user interface is displayed to aid the user when incorrect program parameters are entered.

Command line arguments for program parameters.

Valid options:

[-v] : Verbose
-a : Path/filename for Event configuration file
-b : Path/filename for error log file
-c : TSserver host name
-s : SPS TG3 interface card descriptor
-l : LEP TG3 interface card descriptor

Example for SPS:

zltg3 -s -a /user/spsabt/btsoft/prod/data/spstg3.cfg -b /user/spsabt/btsoft/prod/log/errlog -c modena

Example for LEP:

zltg3 -l -a /user/spsabt/btsoft/prod/data/leptg3.cfg -b /user/spsabt/btsoft/prod/log/errlog -c modena

The error log name is concatenated with the host system name to provide a unique log name. Only the on-line LEP systems listed should use the log directory:

/user /spsabt /btsoft/prod/log.

A description of the error messages is include in the appendix C.

The host names of the operational LynxOS systems used for the LEP separators are: ztls15, zxls25, ztlj33, zxls45, ztlj56, zxls65, ztlj76, and zxls85.

5. Timing error reporting

The objective is to inform the operators of LEP of a possible problem in the machine timing system by means of the early detection of missing events. The initially proposed method was to check for the absence of the 1 ms clock. Since this absence can not generate an interrupt to the host system, the TG3-PC interface card has to be polled by the server program. This can be achieved by using the **select** call time-out, which is 20 seconds, for checking the 1 ms clock missing flag and the watchdog flag. This means that if one of them is true a timing error has occurred.

However, this method causes a problem because the TG3-PC interface card generates it's own 1 ms clock when the externally received 1 ms clock is absent. Unfortunately this state is not correctly reported to the Receiver's error register. This renders the above test method invalid and timing errors could occur without being detected by the present method.

Alternative solutions for solving this problem are:

- Reading the one 1 ms clock counter and comparing the value with the length of the SSC.
- Comparing consecutive SSC numbers.
- Comparing the date and time recorded in the auxiliary table.

All the above solutions have been tested. There respective strengths and weaknesses are listed below.

For normal operation **without** missing 1 ms events the following information can be obtained:

1. The **readlastssc(tg3_desc->fd)** call [4] returns the last SSC number, so after every SSC the function should return consecutive numbers (HEX).
2. The **readcylen(tg3_desc->fd)** call returns the number of milliseconds in the previous SSC.
3. The **ioctl(tg3_desc->fd, TG3_DATE, &iob)** call returns the date and time in 6 bytes. This shows the date and time incrementing after every read.

For normal operation **with** missing 1 ms events the following fault indications were obtained :

1. After a transitory error the **readcylen(tg3_desc->fd)** call returns a number that differs from the number of 1 ms ticks in the currently used SSC. This error will be trapped.
   However, if the duration of the absence of 1ms ticks is a multiple of the SSC length then the error will not be trapped.

2. If the error is permanent the **readcylen(tg3_desc->fd)** call returns the number of 1 ms clock ticks in the last complete SSC before the error occurred. This number will be the same as the expected SSC length. This error will thus not be trapped.

3. After a transitory error the **readlastssc(tg3_desc->fd)** call returns the last SSC number before the error. However, if the duration of the absence of 1 ms ticks is less than length of the SSC then the SSC number will be incremented. This error will not be trapped.

4. If the error is permanent the **readlastssc(tg3_desc->fd)** call returns the last SSC number before the error. The SSC number will not be incremented. This error will be trapped.

5. After a transitory error the **ioctl(tg3_desc->fd, TG3_DATE, &iob)** call returns the date and time after the error. This will be different from the old time and the error will not be trapped.
   However, if the duration of the absence of 1 ms ticks is greater than length of the **select** call time-out the date and time will not be incremented. This error will be trapped.
6. If the error is permanent the **ioctl(tg3_desc->fd, TG3_DATE, &iob)** call returns the date and time before the error occurred, the time will be the same as the old time. This error will be trapped.

These results lead to the conclusion that each method can detect either fatal errors or transitory errors, but not both. A combination of the tests on the SSC length and the SSC number allow also the detection of fatal non-recoverable errors.

However, the weakness of using the SSC event tests in LEP is that, once in colliding mode, the SSC events are absent. The generation of a synchronous SSC event in LEP is impossible because just this absence of the SSC event determines the operating mode of LEP. Thus, it is not possible to capture timing system problems with tests on the SSC events.

Therefore the date and time check in the TG3-PC interface card is only option left for early detection of machine timing errors in LEP. This check has thus been implemented in the TG3-PC server program.

6. Conclusion

The TG3-PC server program **zltg3** is designed to be easily maintainable and robust during normal operation. However due to the wide variety of possible system conditions that can occur outside the control of the program, it remains possible that an external processes can interfere with the timing distribution system i.e. **TG3DIAL**.

The server program is designed to detect conditions that could interfere with its normal operation. The hardware is monitored for errors, each access to the TG3-PC interface card is validated on return. Each time data is transferred to or from the hardware or a file, the **sizeof()** routine checks that the size of the transfer is correct. The server program logs errors and status as they occur in an error log file, which is outlined in Appendix C.

The server program was extensively tested with other concurrent processes in the LynxOS host. This confirmed that under normal system operating conditions, the TG3-PC timing system can capture machine timing events from the GMT and broadcast them to the ECAs on the MIL-1553B bus. Entries in the diagnostic post mortem tables of the ECAs confirmed the reception of the timing broadcasts.

Since the start of the LEP run in April 1995 the server program has been in operation and fulfils its task to great satisfaction.

7. Acknowledgements

The support provided during the development and test phases by many members of the CERN-SL Controls Group is gratefully acknowledged. We are indebted to E. Carlier and V. Mertens for their contributions and to P. Bobbio for his help with the tests.

8. References

[1] B. Puccio, G. Beetham, and P. Nouchi, "TG3-PC User Manual", January 1993, CERN SL/CO Note (Draft '93).

[2] H. P. Christiansen, "LynxOS Driver for the TG3 Card, Functional Description and User's Guide", November 1993, CERN SL/CO-Note 93-56 (REV), World-Wide Web URL http://genova.cern.ch /~hpchr/tg3_doc.html.

[3] D. J. Cameron, "Documentation on Software for the TG3 and PTS Modules", 12 August 1992, CERN SL/CO/TI.

[4] A. Aimar, E. Carlier and V. Mertens, An User-Friendly Approach to Process Control Software in the Framework of a Generic Tool Kit for Distributed Applications, Workshop on Workstation & Software Tools for Automatic Control, Prague, Czech Republic, October 26 - 27, 1993, CERN SL/93-50 (BT), World-Wide Web URL http://dxbt00.cern.ch/ts/ts_papers/prague_art.html.

[5] V. Mertens, A. Aimar and E. Carlier, A Simple Generic Software Tool Kit for Distributed Controls Applications, Proc. International Conference on Accelerators and Large Experimental Physics Control Systems, Berlin, Germany, October 18 -22, 1993, Nucl. Inst. Meth. A352(1994)427, CERN SL/93-49 (BT), World-Wide Web URL http://dxbt00.cern.ch/ts/ts_papers /berlin93.html.

[6] A. - K. Brignet, "NBC LynxOS Driver (draft)", 5 December 1994, SL/Note 93-58 Revision 3 (CO), World-Wide Web URL http://genova.cern.ch/~abl/lynx /NBC_driver.html.
[7] P. Charrue, "Making outputs and logging from Servers and daemon", 8 July 1994, World-Wide Web URL http://genova.cern.ch/~charrue/print.html.
[8] A. Aimar, E. Carlier, C. Cameron, A. Ferrari, B. Goddard, M. Laffin, V. Mertens and M. Tyrrell, Centralised Equipment Surveillance Using a Configurable Software Tool Kit as Front-End to the CERN Alarm System, CERN SL/Note 95-71 (BT).
[9] Carlier SL/BT, private communication.

Appendix A

TG3-PC server program "zltg3.c"

```
/*
 * Program name          zltg3.c
 *
 *              General Machine Timing Program
 *
 *               for  TG3-PC Interface Card
 *
 *                       Andrew Burton
 *
 *                         SL/BT
 *
 */


/*
 *
 *             Modification History
 *
 *             REL 01        APR 95
 *
 *             Modification History
 *
 *             REL 02        JUN 95       SPS mode, H/W status check at Start Super Cycle
 *                                        Completed error log and alarm messages
 *
 *             REL 03        OCT 95       Date/Time check introduced
 */

#include <errno.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/time.h>
#include <sys/sched.h>
#include <signal.h>
#include <string.h>
#include <print.h>
#include "TSCglobal.h"
#include "sysconfig.h"
#include "tg3.h"
#include "tg3_tim.h"



/*global declarations*/

union iob iob;                          /*data structure in tg3.h*/
fd_set tg3fd_set;                       /*structure in types.h */
unsigned int verbose;
 int old_date_time[6];
/*
 *
 *     disable tg3 interface card
 *
 */

int disable_tg3(struct tg3_descriptors *tg3_desc)
{
 if(en_disable(tg3_desc->tg3nb,TG3_ASYDIS) == MINUS)
   {
    print(log_all,"Unable to disable TG3-PC interface card ");
    return(MINUS);
   }
 return(ZERO);
}

/*
 *
 *     enable tg3 interface card
 *
 */

int enable_tg3(struct tg3_descriptors *tg3_desc)
```

```
{
 if(en_disable(tg3_desc->tg3nb,TG3_SYNENA) == MINUS)
   {
    print(log_all,"Failed to enable TG3-PC interface card ");
    return(MINUS);
   }
 return(ZERO);
}
/*
 *
 *    Check TG3-PC date and time
 *    after select timeout
 *
 */

int date_time_chk(struct tg3_descriptors *tg3_desc)
{
 int i;

 if(ioctl(tg3_desc->fd, TG3_DATE, &iob) < ZERO)
   {
    print(log_all,"Failed to get date and time from TG3-PC interface card ");
    return(MINUS);
  }
 else
   {
    if((iob.i_date[0] & 0xff) <= (old_date_time[0] & 0xff))
      if((iob.i_date[1] & 0xff) <= (old_date_time[1] & 0xff))
        if((iob.i_date[2] & 0xff) <= (old_date_time[2] & 0xff))
          if((iob.i_date[3] & 0xff) <= (old_date_time[3] & 0xff))
            if((iob.i_date[4] & 0xff) <= (old_date_time[4] & 0xff))
              if((iob.i_date[5] & 0xff) == (old_date_time[5] & 0xff))
                {
                 strcpy(tg3_desc->TSalarm.Message, "date/time error");
                 return(PLUS);
                }
   }

   for(i = 0; i < 6; i++)
     {
      old_date_time[i] = (iob.i_date[i] & 0xff);
     }
 return(ZERO);
}

/*     not used due to hardware problem
 *
 *    Check TG3-PC Reciever's error reg
 *    for one milli second  'WATCH_DOG' and 'MISS1MSCLK'
 *    after select timeout
 *
 *  note        iob.i_stat.s_flags and Reciever's error register
 *              is reset after test
 *
 *
 */

int rec_reg_chk(struct tg3_descriptors *tg3_desc)
{
 static int log_once;

 if(ioctl(tg3_desc->fd, TR_GETSTATUS, &iob) < ZERO)
   {
    print(log_all,"Unable to read TG3-PC Reciever's error reg");
    strcpy(tg3_desc->TSalarm.Message, "Cannot Read TG3");
    return(MINUS);
   }
 else
   {
    if((iob.i_stat.s_flags & WATCH_DOG)||(iob.i_stat.s_flags & MISS1MSCLK))
      {
       strcpy(tg3_desc->TSalarm.Message, "GMT Clock Fail");

       /*
        * Reset Receiver error register
        */

       if(ioctl(tg3_desc->fd, TG3_RESREC, &iob) < ZERO)
          if(log_once == ZERO)
            {
             print(log_all,"Unable to reset TG3-PC Reciever's error reg");
```

```
            log_once = PLUS;
          }

        iob.i_stat.s_flags = 0x00;
        return(PLUS);
       }
    }
 log_once = ZERO;
 return(ZERO);
}

/*
 *
 *    Check TG3-PC Reciever's error reg
 *    for one milli second 'WATCH_DOG' and 'MISS1MSCLK'
 *    after SSC event
 *
 *    note   SPS operation only
 */

int get_SPS_status(struct tg3_descriptors *tg3_desc)
{
 if(ioctl(tg3_desc->fd, TR_GETSTATUS, &iob) < ZERO)
   {
    print(log_all,"Unable to read TG3-PC Reciever's error reg");
    strcpy(tg3_desc->TSalarm.Message, "Cannot Read TG3");
    return(MINUS);
   }
 else
   if((iob.i_stat.s_laperrs[1] != ZERO) || (iob.i_stat.s_laperrs[2] != ZERO))
      {
       print(log_all,"GMT one mill-sec clock fail");
       strcpy(tg3_desc->TSalarm.Message, "GMT Clock Fail");
       return(PLUS);
      }
 return(ZERO);
}

/*
 *
 *     Test TG3-PC interface card
 *
 */

int auto_test(struct tg3_descriptors *tg3_desc)
{
 int status;
 if(self_test(tg3_desc->tg3nb, &status) == MINUS)
   {
    print(log_all,"Unable to test TG3-PC Interface Card");
    return(MINUS);
   }
 if(status & BAD_STATUS )
   {
    print(log_all,"TG3-PC Interface Card failed self test ");
    return(MINUS);
   }
 return(ZERO);
}

/*
 *
 *     Initialise file descriptors
 *
 */

int get_tg3desc(struct tg3_descriptors *tg3_desc)
{
 if((tg3_desc->fd = get_tg3fd(tg3_desc->tg3nb)) < ZERO)
   {
    print(log_all,"Could Not Open TG3-PC Interface Card" );
    close(tg3_desc->fd);
    return(MINUS);
   }
 else
   {
    FD_SET(tg3_desc->fd, &tg3fd_set);
    tg3_desc->maxfd = tg3_desc->fd + FILE_DESCS;
    return(ZERO);
   }
}
```

```
/*
 *
 *     Write event table in tg3 interface card
 *
 */

int write_event_table(struct tg3_descriptors *tg3_desc)
{
 FILE *fp;
 char line[LINE_MAX], *cp;
 int n,ret;

 /*
  *
  *    Open and Read Event Configuration file
  *
  */

 if((fp = fopen(tg3_desc->config_file, "r")) == NULL)
   {
    if(verbose)
       printf("Unable to open event config file :-  %s \n ", tg3_desc->config_file);
    print(log_all,"Unable to open event config file :-  %s ", tg3_desc->config_file);
    return(MINUS);
   }
 for(;;)
   [
    cp=fgets(line,LINE_MAX , fp);
    if(cp==NULL)
      [
       if(verbose)
         printf("events from %s initialised\n", tg3_desc->config_file);
       fclose(fp);
       return(ZERO);
      ]

    line[24] = '\0';                          /*sscanf does not read comments*/

    /*
     *
     *       Assign values
     *
     */

    n=sscanf(line,"%x %x %x %x %x %x %x %x",
       &iob.i_evdecl.d_event.e_head,
       &iob.i_evdecl.d_event.e_name,
       &iob.i_evdecl.d_event.e_type,
       &iob.i_evdecl.d_event.e_typeno,
       &iob.i_evdecl.action,
       &iob.i_evdecl.delay,
       &iob.i_evdecl.row,
       &iob.i_evdecl.sig_no);

    if(n == ZERO)
      {
       print(log_all,"Unable to read Event Configuaration file :-  %s",tg3_desc-
>config_file);
       return(MINUS);
      ]

    if(((iob.i_evdecl.d_event.e_head & 0xf0) ==  tg3_desc->tg3nb) && (tg3_desc->fd  ==
MINUS))
      [
       print(log_all,"Compare failed when writing the Event Configuration file to the
TG3-PC Interface card");
       return(MINUS);
      ]
    else
      if(write(tg3_desc->fd, &iob.i_evdecl, sizeof(struct decl_event))
                                                        != sizeof(struct decl_event))
       [
         print(log_all,"Unable to write Event Configuration file to TG3-PC Event
table");
         return(MINUS);
       ]
    ]
]
```

```
/*
 *
 *      Switch byte order in integer
 *
 */

unsigned int change_pos(unsigned int tempbyte)
{
 unsigned int temp_byte,i;
 unsigned int temp_byte_00,temp_byte_01;
 unsigned int temp_byte_10,temp_byte_11;

 temp_byte = tempbyte;

 temp_byte_00 = temp_byte & 0x000f;
 temp_byte_01 = temp_byte & 0x00f0;
 temp_byte_10 = temp_byte & 0x0f00;
 temp_byte_11 = temp_byte & 0xf000;

 for(i=0; i<=7; i++)
   {
     temp_byte_00 = temp_byte_00 << 1;
     temp_byte_01 = temp_byte_01 << 1;
     temp_byte_10 = temp_byte_10 >> 1;
     temp_byte_11 = temp_byte_11 >> 1;

   }

 temp_byte = temp_byte_00 + temp_byte_01 + temp_byte_10 + temp_byte_11;

 return temp_byte;
}

/*
 *
 *      Broadcast Event to MIL-1553B Bus Controller
 *
 */

int send_event()
[
 struct m_broadcast
  [
    unsigned char  length;
    unsigned char  send_id;
    unsigned char  fifth_byte;
    unsigned char  rti;
    unsigned char  sixth_byte;
    unsigned char  event_num;
    unsigned short int delay;
    unsigned short int chksum;
  } broadcast ;

 int bc = MIL_BUS_CONT;
 int status;
 unsigned int temp_byte;
 unsigned int  addition;

 broadcast.length     = BROADCAST_LENGTH;
 broadcast.send_id    = BROADCAST_TX_ID;
 broadcast.fifth_byte = BROADCAST_PAD;
 broadcast.rti        = BROADCAST_RTI;
 broadcast.sixth_byte = BROADCAST_PAD;
 broadcast.event_num  = iob.i_rbevent.r_event.e_name;
 broadcast.delay      = BROADCAST_DELAY;
 if(verbose)
   printf(" \n Event number is %x \n",broadcast.event_num);

 addition  = broadcast.send_id
           + broadcast.rti
           + broadcast.event_num
           + broadcast.delay;

 broadcast.chksum = addition;
 temp_byte = broadcast.delay;
 broadcast.delay = change_pos(temp_byte);
 temp_byte = broadcast.chksum;
 broadcast.chksum = change_pos(temp_byte);
 print(log_all,"Event number is %x",broadcast.event_num);
 status = PLUS;                                          /*ensures status
changes*/
```

```
 if(m_broad(bc, &broadcast, sizeof(struct m_broadcast), &status) != 0)  /*Broadcast*/
   {
    print(log_all,"Unable to broadcast Event, status is %x",&status);
   ]
 return(ZERO);
}

/*
 *
 *     Read Event Table and Broadcast Event
 *
 */

int read_event(struct tg3_descriptors *tg3_desc)
{
 int ret;
 if(FD_ISSET(tg3_desc->fd, &tg3fd_set))
   {
    if(read(tg3_desc->fd,&iob.i_rbevent,sizeof(struct rb_event))!=sizeof(struct
rb_event))
      {
       print(log_all,"Unable to read Tg3 after an Event");
       return(MINUS);
      }

    if(send_event(&iob.i_rbevent) == MINUS)
      return(MINUS);

    if(iob.i_rbevent.r_flags & GS_MISSED)
      print(log_all,"Missed event flag set");

    if(tg3_desc->tg3nb == MACHINE_SPS)
        if(iob.i_rbevent.r_event.e_head == SPS_H)
           if((ret = get_SPS_status(tg3_desc)) == MINUS)
              return(MINUS);
           else
              if(ret == PLUS)
                return(PLUS);

  ]
 return(ZERO);
}

/*
 *
 *     close tg3 timing program
 *
 *     close all open files
 */

safe_exit(struct tg3_descriptors *tg3_desc)
{
 print(log_all,"Program clean exit");
 close(tg3_desc->fd);
 sleep(2);
 close_logfile();
 exit();
]

/*
 *
 *     initialise tg3 interface card
 *
 */

int initialise_tg3(struct tg3_descriptors *tg3_desc)
[

 if(verbose)
       printf("initialising\n ");
/*
 *    init TS server Tags
 */

 send_tg3alarm(tg3_desc);

 if(get_tg3desc(tg3_desc) != ZERO)
   return(MINUS);

 if(disable_tg3(tg3_desc) != ZERO)
   return(MINUS);
```

```
 if(auto_test(tg3_desc) != ZERO)
   return(MINUS);

 if(clear_table(tg3_desc->tg3nb) < ZERO)
   [
    print(log_all,"Failed to clear TG3 Event table");
    return(MINUS);
   ]

 if(write_event_table(tg3_desc) != ZERO)
   return(MINUS);

 if(enable_tg3(tg3_desc) != ZERO)
   return(MINUS);

 if(verbose)
       printf("end of initialising \n");
  return(ZERO);
]

user_mess()
{
 printf("                                                                    \n");
 printf(" Command line arguments for program parameters                      \n");
 printf("                                                                    \n");
 printf("      valid options                                                 \n");
 printf("                                                                    \n");
 printf("      [-v]  :  Verbose                                              \n");
 printf("       -a   :  Path/filename for Event configuration file           \n");
 printf("       -b   :  Path/filename for error log file                     \n");
 printf("       -c   :  TSserver host name                                   \n");
 printf("       -s   :  SPS TG3 interface card descriptor                    \n");
 printf("       -l   :  LEP TG3 interface card descriptor                    \n");
 printf("                                                                    \n");
 printf("example for SPS                                                     \n");
 printf("         zltg3 -s -a path/spstg3.cfg -b path/errlog -c modena & \n");
 printf("example for LEP                                                     \n");
 printf("         zltg3 -l -a path/leptg3.cfg -b path/errlog -c modena & \n");
 printf("                                                                    \n");
 printf("                                                                    \n");
]

int send_tg3alarm(struct tg3_descriptors *tg3_desc)
{
 seq_RTAG_DEF    TagSet;
 ERR_MSG         ErrMsg;
 int             sid;
 long int        ErrNr;

 static int      init_TSserver;

 unsigned  char  ProcName[32] = "zltg3_";
 unsigned  char  HostName[32];
 unsigned  char  ErrorValue[32];
 unsigned  char  SystemAlarmName[32] = "ZL_" ;
 unsigned  char  SystemAlarmMessage[32] = "ZL_" ;
 unsigned  char  AlarmMessage[16];

/*
 * build strings for Alarm names and message
 */

 int len = 32;
 unsigned char str1[32];
 unsigned char str2[32];
 int hostlen = 16;
 int messlen = 16;
 unsigned char charnull = '\0';

 strncat(ProcName, tg3_desc->host_system_name , len);

 strcpy(str1,  "_TG3_ERROR" );
 strcpy(str2,  "_TG3_MESSAGE" );

 strcpy(HostName, tg3_desc->TSalarm.ServerHostName);
 strncat(SystemAlarmName, tg3_desc->host_system_name , len);
 strncat(SystemAlarmName, str1, len);
 strncat(SystemAlarmMessage, tg3_desc->host_system_name , len);
 strncat(SystemAlarmMessage, str2, len);
```

```
/*
 * Connect to TSserver and Declare Tags in TSserver
 */

/*
 *     malloc
 */

 TagSet.sequence = (RTAG_DEF *)malloc(2 * sizeof(RTAG_DEF));
 TagSet.length   = 0;

/*
 *  Connect and declare Tags
 *
 *
 */

 if(init_TSserver == 0x0)
   {
    if((ErrNr = TSconnect(HostName, ProcName, &sid, ErrMsg)) != TS_NO_ERROR)
       print(log_all,"Unable to connect and declare Tags: Error msg is %s",ErrMsg);
    else
      {
       strcpy(AlarmMessage, "TG3_TAG_DEC");
       strcpy(ErrorValue, "0");
       FCSTRCPY(TagSet.sequence[TagSet.length].Name, SystemAlarmName);
       FCSTRCPY(TagSet.sequence[TagSet.length].Value, ErrorValue);
       TagSet.length++;
       FCSTRCPY(TagSet.sequence[TagSet.length].Name, SystemAlarmMessage);
       FCSTRCPY(TagSet.sequence[TagSet.length].Value, AlarmMessage);
       TagSet.length++;

       if((ErrNr = TScreTagsByName(&TagSet, ErrMsg)) != TS_NO_ERROR)
          print(log_all,"Unable to create tags in TSserver: Error msg is %s",ErrMsg);

      /*
       *  Disconnect from TSserver
       */

       if(ErrNr = TSdisconnect(ErrMsg))
          print(log_all,"Unable to TSdisconnect: Error ErrNr is %s",ErrNr);

      }
   }/* end of init tags */

 free(TagSet.sequence);

/*
 *     malloc
 */

 TagSet.sequence = (RTAG_DEF *)malloc(2 * sizeof(RTAG_DEF));
 TagSet.length   = 0;

/*
 *  Connect to TSserver and send Alarm
 */

    if((ErrNr = TSconnect(HostName, ProcName, &sid, ErrMsg)) != TS_NO_ERROR)
       print(log_all,"Unable to connect to send alarm to TSserver: Error msg is
%s",ErrMsg);
    else
     {
      tg3_desc->TSalarm.Message[15] = charnull;
      strcpy(AlarmMessage, tg3_desc->TSalarm.Message);
      strcpy(ErrorValue, "1");

      /*
       * The initialise OK message is written to the TSserver
       * on the first pass to over write any previous error message.
       */

      if(init_TSserver == 0x0)
       [
        strcpy(AlarmMessage, "TG3 init OK");
        strcpy(ErrorValue, "0");
        init_TSserver = 0x1;
       ]

      FCSTRCPY(TagSet.sequence[TagSet.length].Name, SystemAlarmName);
      FCSTRCPY(TagSet.sequence[TagSet.length].Value, ErrorValue);
```

```
      TagSet.length++;
      FCSTRCPY(TagSet.sequence[TagSet.length].Name, SystemAlarmMessage);
      FCSTRCPY(TagSet.sequence[TagSet.length].Value, AlarmMessage);
      TagSet.length++;

      if((ErrNr = TSsetTagsByName(TagSet, ErrMsg)) != TS_NO_ERROR)
         print(log_all,"Unable to send alarm name to TSserver: Error msg is
%s",ErrMsg);

     /*
      *  Disconnect from TSserver
      */

      if(ErrNr = TSdisconnect(ErrMsg))
         print(log_all,"Unable to TSdisconnect: Error ErrNr is %s",ErrNr);
    ]
 free(TagSet.sequence);

    /*
     *   Clean exit
     *
     *   note possible conflict with TSserver 'signal()', The signal()
     *        function is reinitialised after every alarm message.
     */

     signal(SIGINT, safe_exit);
     signal(SIGQUIT, safe_exit);
     signal(SIGFPE, safe_exit);
     signal(SIGBUS, safe_exit);
     signal(SIGSEGV, safe_exit);
     signal(SIGTERM, safe_exit);
     signal(SIGKILL, safe_exit);

} /* end of send_tg3alarm */


/*
 *
 *     main
 *
 */

main(int argc, char *argv[])
{
 /*
  *
  *
  * define variables
  *
  *
  */

 struct tg3_descriptors tg3_desc;                          /*tg3 descriptors*/
 struct timeval tg3_select_tmo;                        /*for time out on select */

 int nfound, c, result;
 int sel_tim_out_flag, SPS_tim_out_flag, log_once_flag;      /*limits error logging*/
 int cntl_flag, a_cntl_flag, b_cntl_flag, c_cntl_flag;  /*optarg parameter checking*/
 extern int    optind, opterr;                                      /*optarg*/
 extern char   *optarg;
 char          buffer[128];
 int           ch;                                             /*gethostname*/
 char          *pos;                                           /*gethostname*/
 char          tg3name[32];
 unsigned char charnull;                                 /*terminates string*/

 /*
  *
  *
  * init variables
  *
  *
  */

 tg3_select_tmo.tv_sec = TG3_SELECT_TMO;                    /* select timeout*/
 tg3_select_tmo.tv_usec = ZERO;
 verbose = ZERO;
 optind = ONE;                                               /*optargs*/
 opterr = ZERO;
 cntl_flag  = MINUS;
 a_cntl_flag  = MINUS;
```

```
 b_cntl_flag  = MINUS;
 c_cntl_flag =  MINUS;
 charnull = '\0';
 ch = '.';
 sel_tim_out_flag = ZERO;
 SPS_tim_out_flag = ZERO;
 log_once_flag = ZERO;
 FD_ZERO(&tg3fd_set);                                    /* empty file descriptor
set*/

 strcpy(tg3_desc.error_file, "Default_log");              /*default error file
name*/


 /*
  *     Gets command line arguments
  *
  *        for program parameters
  *
  *         valid options
  *
  *     -v   :  Verbose
  *     -a   :  Event configuration file
  *     -b   :  Event error log file
  *     -c   :  TSserver host name
  *     -s   :  SPS TG3 interface card descriptor
  *     -l   :  LEP TG3 interface card descriptor
  *
  *  default :  Illegal entries
  *
  *
  */


 while ((c = getopt(argc, argv, "a:b:c:lsv")) != EOF)
  [
  switch (c)
     {
      case 'v' :

           verbose = PLUS;
           break;

      case 'a' :

           a_cntl_flag = 1;
           sscanf(optarg, "%s", buffer);
           strcpy(tg3_desc.config_file, buffer);
           break;

      case 'b' :

           b_cntl_flag = 1;
           sscanf(optarg, "%s", buffer);
           strcpy(tg3_desc.error_file, buffer);
           break;

      case 'c' :

           c_cntl_flag = 1;
           sscanf(optarg, "%s", buffer);
           strcpy(tg3_desc.TSalarm.ServerHostName, buffer);
           break;

      case 's' :

           if(cntl_flag != 2)
             [
               tg3_desc.tg3nb = MACHINE_SPS;
               cntl_flag = 1;
             ]
           break;

      case 'l' :

           if(cntl_flag != 1)
             [
               tg3_desc.tg3nb = MACHINE_LEP;
               cntl_flag = 2;
             ]
           break;
```

```
        default  :
                printf("default argument: incorrect program arguments entered\n\n");

          }
    }

 /*
  *
  *         Get host name
  *
  */


 gethostname(tg3_desc.host_system_name, sizeof(tg3_desc.host_system_name));

 pos = strchr(tg3_desc.host_system_name,ch);
 tg3_desc.host_system_name[pos-tg3_desc.host_system_name] = '\0';

 strcat(tg3_desc.error_file, ".");
 strcat(tg3_desc.error_file, tg3_desc.host_system_name);

 /*
  *                init error logging
  */

 if(init_print(P_FALSE,P_TRUE,tg3_desc.error_file,tg3_desc.host_system_name,50000) ==
-1)
    {
     sprintf(stderr,"Cannot initialise errorlogging :- init_print failed \n");
    }

 /*
  *      Log start of program
  */

   print(log_all,"Start of ZLTG3 Program ");

 /*
  *   Check program options
  */

 if(cntl_flag == MINUS)
   {
   print(log_all,"Program arguments incorrect, enter s for SPS or L for LEP");
   user_mess();
   safe_exit(&tg3_desc);
   }

 if((a_cntl_flag || b_cntl_flag )  == MINUS)
   {
   print(log_all,"Program arguments incorrect, enter Configuration and/or Errorlog
file arguments");
   user_mess();
   safe_exit(&tg3_desc);
 }

 if(c_cntl_flag  == MINUS)
   {
   print(log_all,"Program argument incorrect, enter TSserver argument");
   user_mess();
   safe_exit(&tg3_desc);
 }

 /*
  *
  * Initialise TG3 Interface Card
  *
  */

 if(initialise_tg3(&tg3_desc) == MINUS)
   {
     strcpy(tg3_desc.TSalarm.Message, "Failed TG3 init");
     send_tg3alarm(&tg3_desc);
     safe_exit(&tg3_desc);
   }

 /*
  *
  *     Enter forever loop
  *
```

```
 */

for(;;)
   {

    /*
     *
     *    Select waits for event, until time out
     *
     */

    if((nfound = select(tg3_desc.maxfd,&tg3fd_set,0,0,&tg3_select_tmo)) == MINUS)
      {
       print(log_all," Select Failed " );
       if((result = initialise_tg3(&tg3_desc)) != ZERO)
           safe_exit(&tg3_desc);
      }

    if(nfound ==  ZERO)
      {
       if((result = date_time_chk(&tg3_desc)) < ZERO)
           safe_exit(&tg3_desc);
          else
            if(result == PLUS)
               if(log_once_flag == ZERO)
                {
                  print(log_all,"TG3-PC date time error");
                  send_tg3alarm(&tg3_desc);
                  log_once_flag = PLUS;
                }
/*
       if((result = rec_reg_chk(&tg3_desc)) < ZERO)
           safe_exit(&tg3_desc);
          else
            if(result == PLUS)
               if(log_once_flag == ZERO)
                {
                  print(log_all,"TG3-PC Reciever's Error Reg, 'WATCH_DOG' or
'MISS1MSCLK' is true");
                  send_tg3alarm(&tg3_desc);
                  log_once_flag = PLUS;
                }
*/
       /*
        * Restricts the logging of Select time out
        */

       if(sel_tim_out_flag == ZERO)
        {
         print(log_all,"Select Timed Out");
         sel_tim_out_flag = PLUS;
        }

       /*
        *   Log timeout error for SPS operation
        */

       if(tg3_desc.tg3nb == MACHINE_SPS)
          if(SPS_tim_out_flag == ZERO)
           {
            print(log_all,"Select Timed Out: SPS Timing Failed");
            strcpy(tg3_desc.TSalarm.Message, "SPS TimedOut");
            send_tg3alarm(&tg3_desc);
            SPS_tim_out_flag = PLUS;
           }

       /*
        *   Empty and reload file descriptor set
        */

        FD_ZERO(&tg3fd_set);
        FD_SET(tg3_desc.fd, &tg3fd_set);

      }                      /*end of nfound ==  ZERO) */

    if(nfound > ZERO)
      {
       if((result = read_event(&tg3_desc)) == MINUS)
        {
          send_tg3alarm(&tg3_desc);
          safe_exit(&tg3_desc);
```

```
            }
          else
            if(result == PLUS)
               send_tg3alarm(&tg3_desc);

          log_once_flag = ZERO;
          sel_tim_out_flag = ZERO;
          SPS_tim_out_flag = ZERO;

        }  /* End of nfound */
    }   /* End of for */
}         /* End of Program */
```

Appendix B

Header and configuration files

Header file "tg3_tim.h"

```
/*
 * tg3_tim.h - definitions for Timing Event Distribution program
 * Creation Andrew Burton CERN/SL/BT  13.12.94
 */


struct alarm                                          /* TSserver alarm */
  {
   unsigned char    ServerHostName[32];
   unsigned char    Message[32];
  };

struct tg3_descriptors                                /* file descriptors */
  {
   unsigned           char fd;
   unsigned           char maxfd;
   unsigned           char tg3nb;
   char               config_file[128];
   char               error_file[128];
   char               host_system_name[32];
   struct alarm       TSalarm;
 };

/* event configuration file */

#define MINUS                -1            /* return value */
#define ZERO                 0             /* return value */
#define PLUS                 1             /* return value */
#define ONE                  1             /* CONST value */
#define TG3_SELECT_TMO       20            /* longer than SPS supercycle */
#define BAD_STATUS           0x40          /* status check */
#define FILE_DESCS           1             /* number of additional file
                                              descriptors*/
#define LINE_MAX             100           /*length of line */
#define BROADCAST_LENGTH     0x09          /*length of data */
#define BROADCAST_TX_ID      0xAA          /*transmit identity */
#define BROADCAST_PAD        ZERO          /*padding message with zero */
#define BROADCAST_RTI        0xBB          /*RTI  */
#define BROADCAST_DELAY      ZERO          /*delay value zero */
/*#define MIL_BUS_CONT       2  */         /* original controller address*/
#define MIL_BUS_CONT         1             /* new controller address*/
#define MACHINE_SPS          SPS_H         /*TG3 program for SPS*/
#define MACHINE_LEP          LEP_H         /*TG3 program for LEP*/
```

LEP Configuration file **leptg3.cfg**

```
12 11 01 FF 81 00 01 00       # LEPZL stop squeeze
12 15 01 FF 81 00 01 00       # LEPZL start squeeze
12 16 01 FF 81 00 01 00       # LEPZL continue squeeze
12 40 01 FF C1 00 01 00       # LEPZL close sync switch
12 43 01 FF 81 00 01 00       # LEPZL validate default values(coast)
12 45 01 FF 81 00 01 00       # LEPZL next vector
12 46 01 FF 81 00 01 00       # LEPZL previous vector
12 4e 01 FF 81 00 01 00       # LEPZL test
```

SPS Configuration file **spstg3.cfg**

```
21 1c 03 01 81 00 01 00       # SPS e- injection.1 e-
21 1c 03 02 81 00 01 00       # SPS e- injection.2 e-
21 1c 04 01 81 00 01 00       # SPS e+ injection.1 e+
21 1c 04 02 81 00 01 00       # SPS e+ injection.2 e+
21 89 01 01 81 00 01 00       # SPS p+ beam.out.1  p+
21 89 03 01 81 00 01 00       # SPS e- beam out cycle 1
21 89 03 02 81 00 01 00       # SPS e- beam out cycle 2
21 89 04 01 81 00 01 00       # SPS e+ beam out cycle 1
21 89 04 02 81 00 01 00       # SPS e+ beam out cycle 2
2f 05 ff ff 81 00 01 00       # SPS p+ end flat top cycle 1
00 0f ff ff 81 00 01 00       # SPS End of supercycle -5 ms
                              (to be used for start of cycle)
20 ff ff ff 81 00 01 00       # SPS SSC TEST
```

Note: The next line over writes the SSC TEST

```
21 1c 01 01 81 00 01 00  # SPS p+ injection cycle 1
```

Appendix C

Error log messages

The description of the error log messages is split into 5 categories:

- Program parameters
- Initialisation phase
- Logging to alarm system
- Event handling
- During read calls

1. Program parameters

These are the errors reported when incorrect program parameters are passed:

- Local print to screen "Cannot initialise error logging :- init_print failed"

  The **init_print** call failed due to missing parameters which are error log name, the directory path, host system name and file size. The default parameters used ensure that the error should be external to the TG3-PC server program. The default error log filename is **default_log.host name** which is normally written to the local directory.

- "Program arguments incorrect, enter '**s**' for SPS or '**l**' for LEP"
- "Program arguments incorrect, enter Configuration and/or Error log file arguments"
- "Program arguments incorrect, enter TSserver argument"

  The prompts "Program arguments incorrect, enter ... " are user errors when the program checks the command line parameters. The help screen provided shows the correct format of the command line. The program only checks that the switches have been used. The contents of the parameters will be checked when they are used in the TG3-PC server program, i.e. path and file names.

2. Initialisation phase

Errors reported during the initialisation of the TG3-PC interface card. They are shown in the order they could occur. If the error reported is fatal to the operation of the TG3-PC server program the program halts with the **safe_exit** routine.

- "Could Not Open TG3-PC Interface Card"

  Hardware problem: TG3 interface card not installed or incorrectly jumpered.
  Software problem: Incorrect version of driver installed. Use procedure in installation section to check that the driver is configured correctly.

- "Unable to disable TG3-PC Interface Card"

  Hardware problem: TG3 interface card not installed or incorrectly jumpered.
  Software problem: Incorrect version of driver installed. Use procedure in installation section to check that the driver is configured correctly. The TG3 interface card may be in use by another server program. The TG3-PC server program is not designed for multiple usage on the same LynxOS host.

- "Unable to test TG3-PC Interface Card"
- "TG3-PC Interface Card failed self test"
- "Failed to clear TG3-PC Event table"

  Hardware problem: Errors reported when testing TG3-PC interface card. A test can only be run when the TG3-PC interface card is not being used by another server program.

- "Unable to open Event Configuration file :- 'path/filename'"

  Software problem: Incorrect path and/or configuration file name, re-enter program parameters.

- "Unable to read Event Configuration file:- 'path/filename'"

  Software problem: Event Configuration file may be corrupt or zero bytes long.

- "Compare failed when writing the Event Configuration file to the TG3 Interface Card"

  Configuration problem: Error caused when comparing declared header information and file descriptor. The program should not load a LEP configuration table into a SPS initialised TG3-PC interface card or an SPS configuration table into a LEP initialised TG3-PC interface card .

- "Unable to write Event Configuration file to TG3 Event table"

  Hardware problem: A hardware problem should have been isolated before this point i.e. unable to access the TG3-PC interface card.
  Software problem: Configuration problem: Failure when trying to write Event Configuration file to TG3-PC Event table. May be caused by contents of file being rejected by the TG3-PC interface card, i.e. an illegal entry in the action word.

- "Failed to enable TG3-PC Interface Card"

  Hardware problem: A hardware problem should have been isolated before this point.
  Software problem: Incorrect version of driver installed. Use procedure in installation section to check that the driver is configured correctly. The TG3 interface card may be in use by another server program. The TG3-PC Server program is not designed for multiple usage on the same LynxOS host.

3. Logging to Alarm System

Errors reported when connecting to TSserver, declaring or sending a tag and disconnecting from TSserver.

- "Unable to connect and declare Tags: Error msg is 'ErrMsg'"
- "Unable to create tags in TSserver: Error msg is 'ErrMsg'"
- "Unable to TSdisconnect: Error ErrNr is 'ErrNr'"
- "Unable to connect to send alarm to TSserver: Error msg is 'ErrMsg'"
- "Unable to send alarm name to TSserver: Error msg is 'ErrMsg'"

The TSserver can return "Unable to create tags in TSserver: Error msg is 'ErrMsg'" if the tag already exists. This is not an error message. Use the **TSreadTags modena -p ZL_z** command to check the TSserver tag names.

For all other TSserver error/status messages contact E. Carlier.

4. Event Handling

- "Select Failed"

  Hardware problem: Unable to make **select** call, the TG3-PC interface card is re-initialised. An hardware fault should be found during the re-initialisation and be reported by the failing test.

- "Select Timed Out"

  Status message: This is a normal occurrence in LEP, there are a small number of machine timing events per day in LEP. This test is used to check the hardware status of the TG3-PC interface card. The interval is every 20 seconds, only the first occurrence is recorded.

  Software problem: A possible cause is that the event table in the TG3-PC interface card is empty, so no interrupts from machine timing events occur. An external program can interfere with the configuration of the TG3-PC interface card. Use the TG3DIAL program to read the event table and confirm the correct configuration of declared machine timing events.

- "Select Timed Out: SPS Timing Failed"

  Hardware problem: No SPS machine timing events within an SPS supercycle, the hardware is checked by the reading the Receiver's error register, read below.

  Software problem: A possible cause is that the event table in the TG3-PC interface card is empty, so no machine timing events interrupts occur. An external program can interfere with the configuration of the TG3-PC interface card. Use the TG3DIAL program to read the event table and confirm the correct configuration of declared machine timing events. An external cause could be no machine timing events on GMT. Use TG3DIAL to wait for SSC event.

When a "Select Time Out" occurs the TG3-PC Receiver's error register is checked.

- "Unable to read TG3-PC Receiver's error reg"

  Hardware problem: Unable to access the TG3-PC interface card. Hardware fault in TG3-PC interface card. Use TG3DIAL and run self test option, this will read a error and status registers in the TG3-PC interface card.

- "TG3-PC Receiver's Error Reg 'WATCH_DOG' or '1 ms clock is missing is true'"

  Hardware problem: The one millisecond clock as failed. The fault may be in the GMT system. The first check is to see whether this condition is also reported by the other TG3-PC interface cards in LEP, thus showing a system wide fault. The post mortem table of the local TG3-G64 Interface card will show if the there is a mismatch in the reported machine timing events, showing a possible local timing problem in the LynxOS host or the local link in the GMT system.

5. During read call

- "Unable to read TG3 after an Event"

  Hardware problem: Unable to access the TG3-PC interface card, possible hardware fault. Use TG3DIAL to wait for an event.

- "Missed event flag set"

  Status message: The response of the LynxOS host PC was not quick enough and the machine timing event was overwritten in the TG3-PC interface card. This is not a fault. External processes could be slowing the system down sufficiently to delay the interrupt handling. Check system usage and errors reported by the other interface drivers and message handler. It is possible that another process could hang the system. Unable to access the TG3-PC interface card, possible hardware fault.

Note: SPS mode only.

When a machine timing event occurs the TG3-PC interface card hardware status is checked.

- "Unable to read TG3-PC Receiver's error reg"

  Hardware problem: Unable to access the TG3-PC interface card. Hardware fault in TG3-PC interface card. Use TG3DIAL and run self test option. This will read the error and status registers in the TG3-PC interface card.

- "GMT one mill-sec clock fail"

  Hardware problem: The one 1 ms clock has failed since the last read. The fault may be in the GMT system. The first check is to see whether this condition is also reported by the other TG3-PC interface cards in LEP, thus showing a system wide fault. The post mortem table of the local TG3-G64 Interface card will show if the there is a mismatch in the reported machine timing events, showing a possible local timing problem in the LynxOS host or the local link in the GMT system.

- "Unable to broadcast Event, status is 'status message'"

  The utility **m_broad** returned a status message when sending the broadcast data packet to the MIL-1553B Bus Controller. This is only a status message. It as no influence on the operation of the TG3-PC Server program.

Distribution:

SL/BI
A. Burns

SL/BT
B. Balhan
P. Bobbio
A. Burton
E. Carlier
J. P. Deluen
J. H. Dieperink
N. Garrel
B. Goddard
W. Kalbreier
R. L. Keizer
M. Laffin
A. Marchand
V. Mertens
H. Verhagen
E. Weisse

SL/CO
G. Beetham
A. Bland
P. Charrue
R. Lauckner
P. Nouchi
V. Paris
B. Puccio
P. Ribeiro
M Tyrrell
M Vanden Eynden

SL/DI
K. H. Kissler
S. Myers

SL/OP
R. Bailey
A. Faugier
M. Jonker
M. Lamont

SL/PC
J. Pett

SL/RF
E. Ciapala