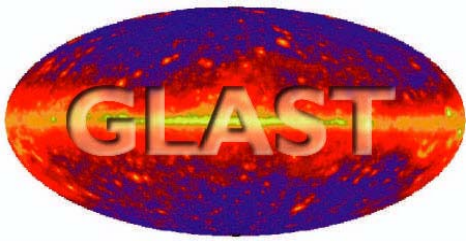


LAT Flight Software

CTDB 1553 Drivers

Type:	Design Description
Version:	V5-3-0
Author:	D.L. Wood
Created:	3 October 2002
Updated:	28 June 2005
Printed:	28 June 2005

A detailed description of the LAT SIU software drivers for interfacing to the GLAST 1553 data bus. The architecture of the LAT 1553 bus policy and hardware interface drivers is discussed. The external interface to the software drivers is defined.

Contents

Figures

Tables

0 Introduction

The GLAST LAT instrument xxx communicates with the Space Vehicle (SC) using a MIL_STD_1553B (1553) bus. The 1553 bus is the sole means for uplinking information to the LAT. The 1553 bus also allows the LAT instrument to output critical telemetry that may be of use to the SC. The 1553 telemetry output may also be employed when real-time contact capability is present for a ground station. The LAT has two hardware node interfaces. Each set of 1553 interface hardware for the LAT is managed by a single CPU (SIU). Only one SIU and associated 1553 interface, however, will be connected to the bus at a given time.

0.0 SIU 1553 Driver Overview

For the GLAST mission, the SC will act as the 1553 bus controller (BC) node. Each SIU acts as a remote terminal (RT) node. Thus, the bus protocol and schedule is under the control of the SC. The LAT and SC must externally coordinate what types of data are to be exchanged on the 1553 bus and with what bandwidth.

The 1553 bus specification allows for remote terminal nodes to be addressed by a terminal address and a subaddress. The LAT SIU will be configured to respond to a unique GLAST mission terminal address. The SIU remote terminal will further recognize a variety of subaddresses. The subaddress will specify the general type of data transaction.

For GLAST, the 1553 interface software has the particular challenge of transferring CCSDS packets across the bus. CCSDS packets are variable length structures transmitted and received asynchronously, while the 1553 bus model favors fixed length, synchronous data structures. The 1553 remote terminal software will perform all necessary packing and unpacking of the CCSDS packets into the 1553 data messages. The external user interface deals only with CCSDS packets, hiding many of the details about how the software inserts and extracts packet fragments.

Two flavors of 1553 driver are available. The interrupt mode drivers are intended for use with the SIU once the RTOS has booted and its services are available. The interrupt mode drivers provide a service task which responds to controller device interrupts. The polled mode drivers are intended for use before the SIU RTOS has booted. This boot environment does not provide services such as interrupt handlers, multitasking, or advanced memory management. Therefore, the polled drivers assume a more limited environment and are much less flexible for general use.

The primary focus of this document is on the remote terminal bus function since the LAT SIU will need to perform this duty during flight. A bus controller interface, however, is also presented as a means for quick, real-time testing and development. In addition, a simulator environment is presented which allows somewhat realistic CCSDS communication across Ethernet.

0.1 Definitions and Acronyms

1553 – MIL-STD-1553B serial data bus specification; in particular the serial data bus and data protocol implemented for the GLAST mission.

BC – The 1553 Bus Controller; the master bus node which directs the bus traffic.

CCSDS – Consultative Committee for Space Data Systems; a series of recommendations for space mission data formats and data handling procedures.

GBM – GLAST Gamma Burst Monitor; one of two science instruments for the GLAST mission.

HKP – Real-time housekeeping telemetry data; telemetry data which relates to the health and safety of the LAT instrument.

LAT – GLAST Large Area Telescope; one of two science instruments for the GLAST mission.

PCI – Peripheral Component Interconnect general purpose parallel data bus. For the purposes of the LAT, two variants are employed: CompactPCI (cPCI) and PCI Mezzanine (PMC).

RT – A 1553 Remote Terminal; the slave bus node that transact with the Bus Controller.

RTOS – Real Time Operating System; in particular the VxWorks 5.4 operating system used by the LAT.

SC – The GLAST Spacecraft; as built by Spectrum Astro. Refer to the *GLAST LAT Instrument – Spacecraft Interface Control Document* for the formal specifications of the SC as seen by the LAT.

SIB – Spacecraft Interface Board; the cPCI board in the SIU crates that contains the LAT 1553 remote terminal hardware.

SIU – The LAT Spacecraft Interface Unit; one of two computer crates which contain a processor and 1553 Remote Terminal interface hardware.

0.2 Reference Documents

Military Standard 1553B, Aircraft Internal Time Division Command/Response Multiplex Data Bus, United States Department of Defense, September 1978.

Military Standard 1553B Notice 2, United States Department of Defense, September 1986.

GLAST 1553 Bus Protocol Interface Control Document, Spectrum Astro, Inc.

GLAST LAT Instrument – Spacecraft Interface Requirements Document, 433-IRD-0001 Revision B, NASA Goddard Space Flight Center, April 2002.

Recommendation for Space Data System Standards, Advanced Orbiting Systems, Networks and Data Links: Architectural Specification, Blue Book 701.0-B-3, Consultative Committee for Space Data Systems, June 2001.

Recommendation for Space Data System Standards, Telecommand Part 3, Data Management Service Architectural Specification, Blue Book 203.0-B-1, Consultative Committee for Space Data Systems, January 1987.

GLAST Spacecraft Interface Board Hardware Specification, LAT Hardware Specification Document.

PMC-1553 Reference Manual, Rev 1.2, Alphi Technology Corporation, June 1998.

Enhanced Summit Family Product Handbook, UTMC Microelectronics Systems Inc., October 1999.

1553 Product Handbook, United Technologies Microelectronics Center Inc., 1992.

CCSDS Package User Manual, LAT Flight Software User Manual.

MSG User Manual, LAT Flight Software User Manual.

PBS Package Documentation, LAT Flight Software Code Documentation.

1 SIU 1553 Bus Interface

1.0 1553 Bus Commands

The 1553 bus controller usually executes a list of commands periodically . The list of commands and its internal timing characteristics is often called the *bus schedule*. The complete list of actions to be taken in each bus time slice is often called a *bus frame*. Each 1553 bus controller command will either specify a read of data from a remote terminal (a transmit), a write of data to a remote terminal (a receive), or a special link level command (a mode code command).

The mode code commands signal special actions to be taken by remote terminals. For instance, one of the mode commands will disable a remote terminal. Most of the mode commands are critical error actions take by the bus controller when a remote terminal is not behaving correctly. Many of the critical mode commands are handled automatically by the remote terminal hardware.

The bulk of the 1553 commands in a bus schedule are data transfer commands. The bus controller can either command a remote terminal to transmit a data message or send a remote terminal a data message. Each 1553 data message is rather small – 32 16-bit words or 64 bytes. A good rule to remember is that each data message allocated in a bus schedule uses 0.5 Kb of bandwidth.

1.1 1553 Subaddresses

The 1553 bus allows different types of data to be tagged according to *subaddress*. The subaddress concept is present in the 1553 specification at the link level. Each transaction on the bus must not only be directed to some remote terminal address, but also towards some subaddress within that remote terminal. The subaddress concept allows the system designer the ability to designate bandwidth on the bus not just according to node priority, but also according to function priority within the node.

The subaddress is specified in the bus commands by a value from ‘1’ to ‘31’. The subaddress value ‘0’ indicates that the command is a mode command. Also note that transmit (RT->BC) and receive (BC->RT) subaddress spaces are distinct. The LAT will use only a portion of the subaddress space available.

The SIU remote terminal must be configured beforehand to indicate the general nature of the 1553 bus schedule. The remote terminal must know which subaddresses will be used and how many 1553 data messages are allocated to each subaddress. The remote terminal, however, generally does not need to know the exact timing specifications of the bus schedule.

The subaddresses recognized by the SIU remote terminal are shown in Table 1 below.

Table 1- LAT SIU 1553 Subaddresses

Subaddress	Transfer Direction	Transfer Rate	Interface	Description
Command Receive (CmdRx)	BC->RT	3.5 – 10 Kbps (3.5 Kbps always for SC time, position, and attitude messages)	Asynchronous	Telecommand input for the LAT. • Ground commands • Ground uploads • SC real-time commands • SC stored commands • SC time, position, and ancillary messages • GBM alert messages
Command Transmit (CmdTx)	RT->BC	0 – 2.5 Kbps	Asynchronous	Telecommand output from the LAT. • SC repoint requests • GBM alert messages
Telemetry (Telem)	RT->BC	3.7 – 30.7 Kbps (3.7 Kbps always for HKP telemetry)	Asynchronous / Synchronous	Variable-length LAT telemetry packets. • Housekeeping telemetry • Diagnostic telemetry • Alert telemetry
Data Wraparound (Wrap)	N/A	N/A	N/A	SC test of RT basic functionality.

`

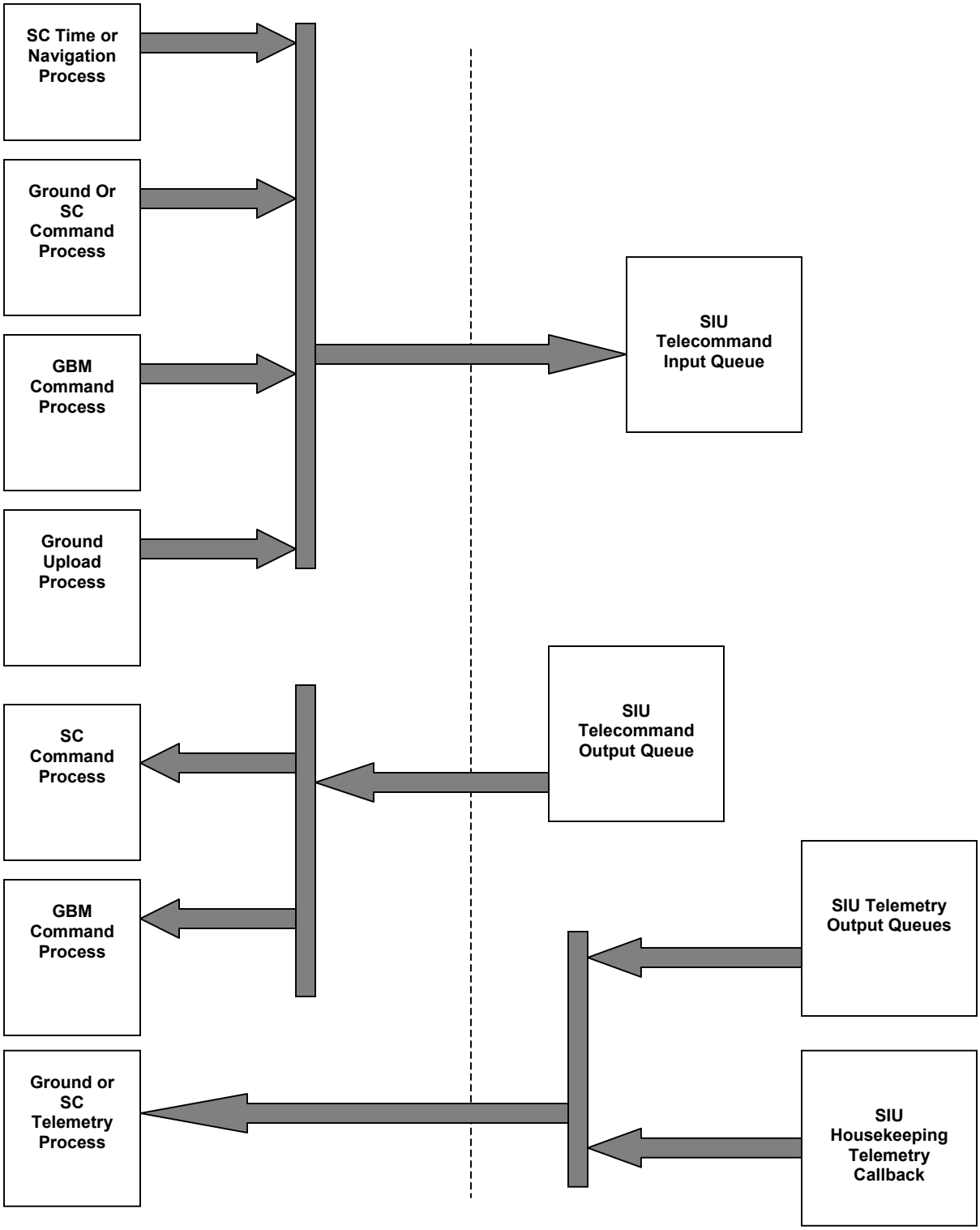
Each subaddress has a specified bandwidth and interface characteristics. Each subaddress is dedicated to transferring a particular type of data. The “Interface” column describes the user interface to the subaddress. Synchronous subaddresses will transact data at the time of the 1553 bus command. The asynchronous subaddresses place outgoing data in a queue for later sending or the place incoming data in a queue for later reception.

1.2 1553 CCSDS Packets

As mentioned before, the primary function of the SIU remote terminal interface will be to send and receive CCSDS packets. For asynchronous subaddress interfaces, the CCSDS packets will be placed on a queue. For asynchronous transmits, the packet is placed on an output queue and

the remote terminal will transmit the packet contents when bandwidth is available. For asynchronous receives, the remote terminal provides a pool of packet buffers for holding incoming data messages. The user then removes the packets from the receive queue when ready.

Figure 1 - LAT Conceptual 1553 Data Flow

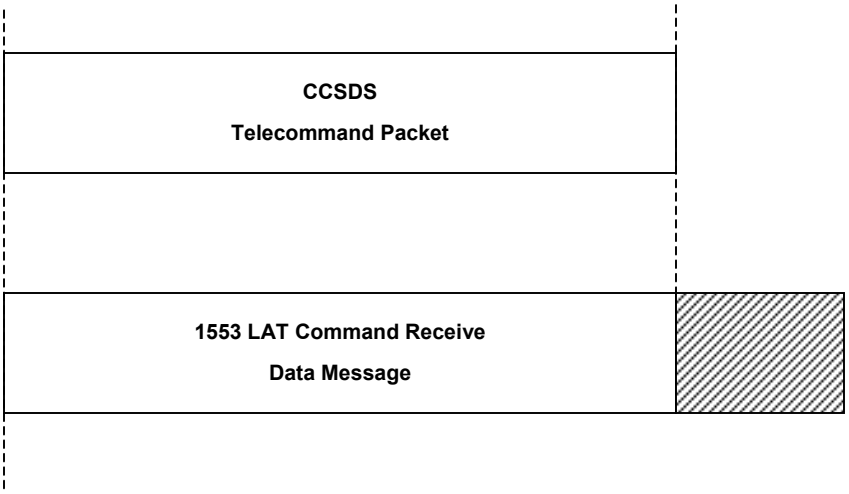


The diagram above shows a conceptual flow of data across the 1553 bus. Each subaddress is essentially independent, allowing different ground and flight applications to communicate simultaneously. Due to the constraints of the GLAST 1553 packet protocol, limits are placed on the maximum size of the CCSDS packets each subaddress may carry.

1.3 1553 Command Receive Subaddress

The LAT instrument is able to receive input telecommands across a dedicated 1553 command receive subaddress. Each command arrives as a single CCSDS telecommand packet. The input command packets are limited to a maximum rate of 20 Hz. The beginning of the CCSDS packet, where the header is located, is always aligned with the start of the first 1553 command message data word. The command packet state machine delivers the packets to the user after validation. The telecommand packets are placed on an input queue to decouple the application command processing tasks from the 1553 bus schedule.

Figure 2 - 1553 Command Receive Subaddress Packets



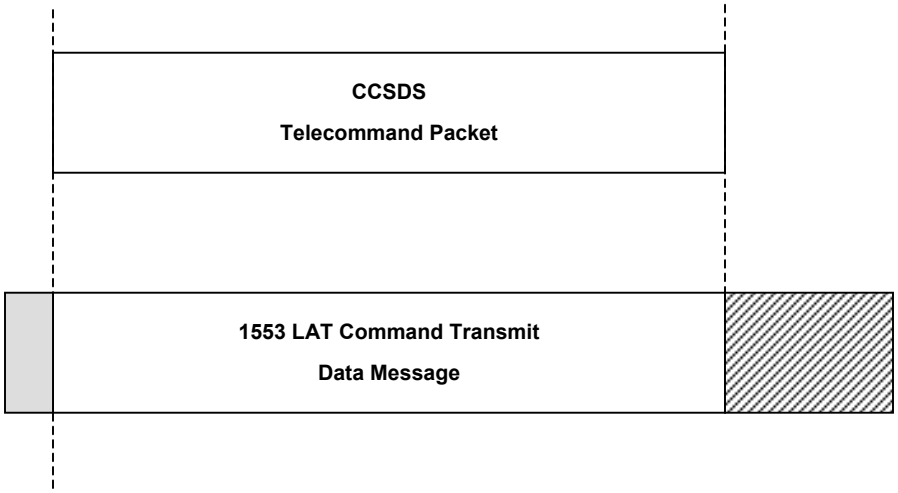
The CCSDS telecommand packet may not occupy the entire 1553 data message, in which case, a filling trailer of ‘0’’s is added to the end of the data message. The total size of each CCSDS telecommand input packet to the LAT is limited to 62 bytes, including the header and trailing checksum.

1.4 1553 Command Transmit Subaddress

The LAT instrument is able to send output telecommands across a dedicated 1553 command transmit subaddress. Each command is sent as a single CCSDS telecommand packet. The

output command packets are limited to a maximum rate of 5 Hz. The output telecommands are for consumption by the SC and include LAT repoint command requests related to gamma ray burst detection.

Figure 3 - 1553 Command Transmit Subaddress Packets

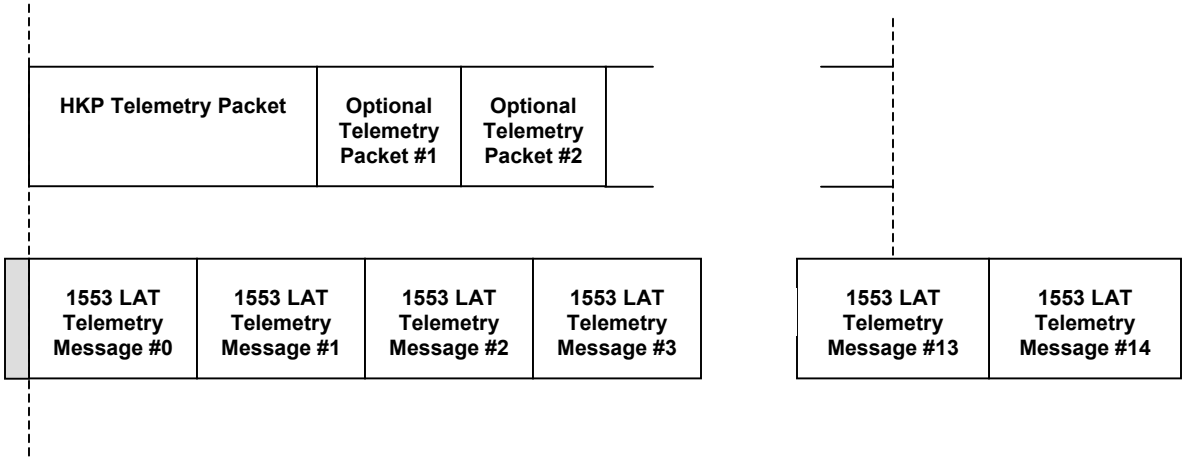


The first word of the 1553 command transmit data message is reserved by the GLAST protocol for flow control. The beginning of the CCSDS packet, where the header is located, is always aligned with the second 1553 command message data word. The command packet state machine delivers the packets to the bus from the user after validation. The telecommand packets are placed on an output queue to decouple the application command generation tasks from the 1553 bus schedule. The CCSDS telecommand packet may not occupy the entire 1553 data message, in which case, a filling trailer of ‘0’’s is added to the end of the data message. The total size of each CCSDS telecommand output packet from the LAT is limited to 62 bytes, including the header and trailing checksum.

1.5 1553 Telemetry Subaddress

The GLAST 1553 protocol allows the LAT to transmit variable length CCSDS packet telemetry across a set of dedicated subaddresses. The bus controller will poll the LAT remote terminal at a rate of 4 Hz to determine if any new telemetry packets are ready for transmission. The telemetry packets are bundled together into blocks of 15 1553 telemetry data messages.

Figure 4 - 1553 Telemetry Subaddress Packets



The first word of the first 1553 telemetry transmit data message is reserved by the GLAST protocol for flow control. The beginning of the first CCSDS packet, where the first header is located, is always aligned with the second data word of the first telemetry transmit data message. Thereafter, the telemetry packets are chained together, with the header length members indicating where the header of the following packet is to be located. The CCSDS telemetry packet chain may not occupy the entire set of 1553 data messages, in which case, a filling trailer of ‘0’’s is added to the end of the chain until all of the data messages are filled. The total size of each CCSDS telemetry output packet from the LAT is limited to 958 bytes, including the header. The telemetry packets are placed on an output queue to decouple the application telemetry generation tasks from the 1553 bus schedule.

The 1553 remote terminal interrupt mode drivers provide two queues for sending telemetry packets, a high priority queue and a low priority queue. When a new telemetry block is ready for preparation, the driver will first attempt to fill in the optional chain of CCSDS telemetry packets with packets taken from the high priority queue. Only when a telemetry block has remaining space and there are no outstanding packets waiting on the high priority queue will the driver examine the low priority queue for CCSDS telemetry packets to insert into the block.

The LAT 1553 remote terminal drivers provide a mechanism to guarantee the delivery of one real-time housekeeping telemetry packet for every telemetry block. A special HKP callback function will be invoked when the driver has transmitted the last telemetry block and is preparing a new one. The HKP callback will be provided the opportunity to insert the first packet in the list of packets in the block. This will allow the LAT to provide HKP telemetry packets at the recommended 4 Hz rate, regardless of how many other telemetry packets are pending for transmission.

1.6 1553 Wrap Subaddress

The LAT remote terminal will implement the data wraparound feature as specified in *MIL-STD-1553B Notice 2*. The RT simply allocates a buffer in the device shared memory and allows the BC node to write and then read back data test patterns sent and received in 1553 data messages on this subaddress. The RT drivers perform the necessary setup at initialization, but are not required to perform any further action.

1.7 1553 Mode Code Commands

The LAT remote terminal will implement the minimum mode code commands as specified in *MIL-STD-1553B Notice 2*. These mode codes relate primarily to the safety and reliability of the 1553 bus system. The required RT mode codes are listed in the table below.

Table 2 - LAT SIU 1553 Mode Code Commands

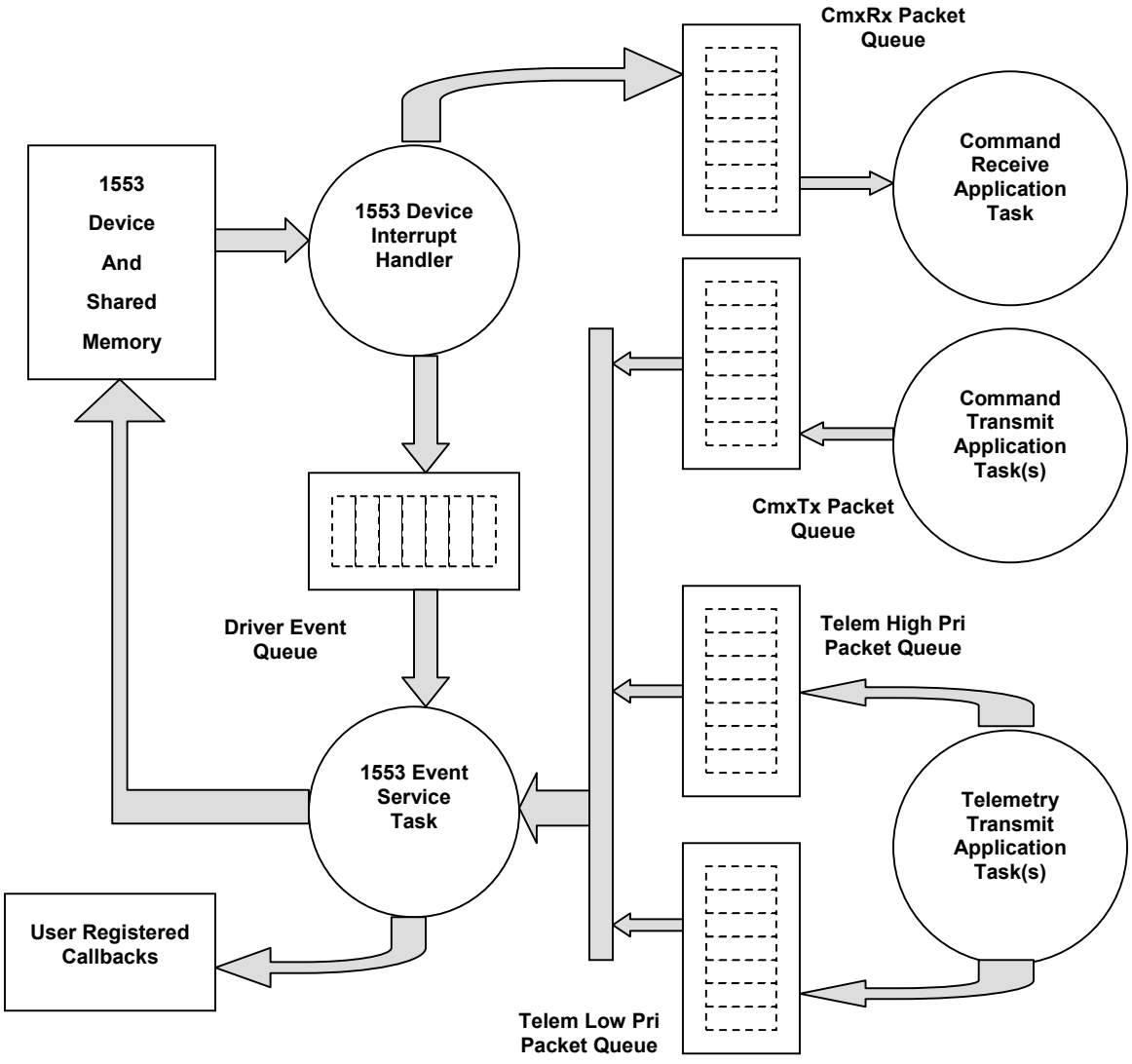
Mode Code Value	Description	Data Word
2	Transmit Status Word	No
4	Transmitter Shutdown	No
5	Override Transmitter Shutdown	No
8	Reset Remote Terminal	No

All of the 1553 controller devices relevant to the LAT flight software development automatically implement the required mode codes transparently in hardware. The user of the driver, however, is responsible for marking each of the required mode codes as legal (see Section 2.3.0.0.3).

2 Interrupt Mode Drivers

2.0 RT Driver Architecture

Figure 5 – RT Interrupt Mode Driver Architecture



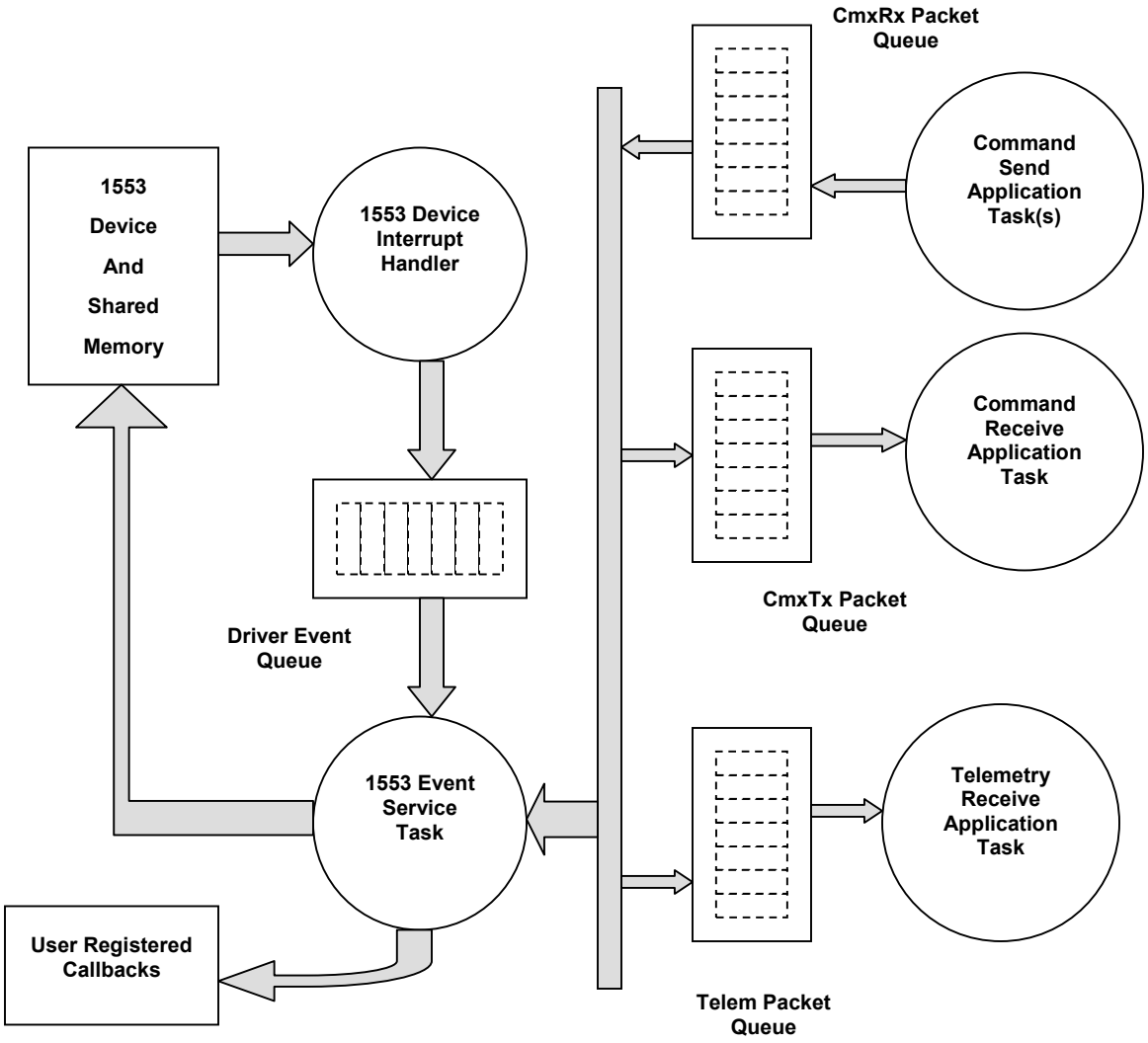
The driver configures the 1553 device to generate an interrupt whenever a valid subaddress message arrives on the bus or when the device reports an error. If the 1553 device indicates that a new telecommand receive data message has arrived, the incoming packet is placed directly on the CmdRx packet queue through a CCSDS interface library call. If the 1553 device indicates that it detected a device or bus error, a MSG error report is queued. Otherwise, if the 1553 device indicates that a data message has been sent from the remote terminal , the device interrupt handler will place a message into the driver event queue. Before exit, the 1553 interrupt handler clears the device interrupt signal. The 1553 event service task waits for new event messages on the queue and processes them. If the message indicates a 1553 data message bus transaction, the 1553 task will call the appropriate function from the CCSDS packet interface library to process the data. The packet interfaces are keyed by subaddress as provided by the 1553 device. The work of preparing a new set of outgoing data messages when the last set has been read is handled for the CmdTx and Telem packet queues within the 1553 service task.

The packet interfaces implement a policy to assemble incoming 1553 data messages into receive packets and fragment transmit packets into outgoing 1553 data messages. The device subaddress interrupt signal is given for receive subaddresses when a new message has arrived on that subaddress. The device subaddress interrupt signal is given for transmit subaddresses when the last message on that subaddress has been sent. A CCSDS packet from a receive interface is placed on one of the packet receive queues for reader applications to consume. A CCSDS transmit packet from sender applications is placed on an output queue, which is checked for any new available outgoing packets when the 1553 device has signaled that the last packets have been transmitted.

During certain phases of the bus schedule, user installed callbacks will be notified. It is important to note that the callbacks are executed in the context of the 1553 event service task and should be as efficient as possible.

2.1 BC Driver Architecture

Figure 6 – BC Interrupt Mode Driver Architecture



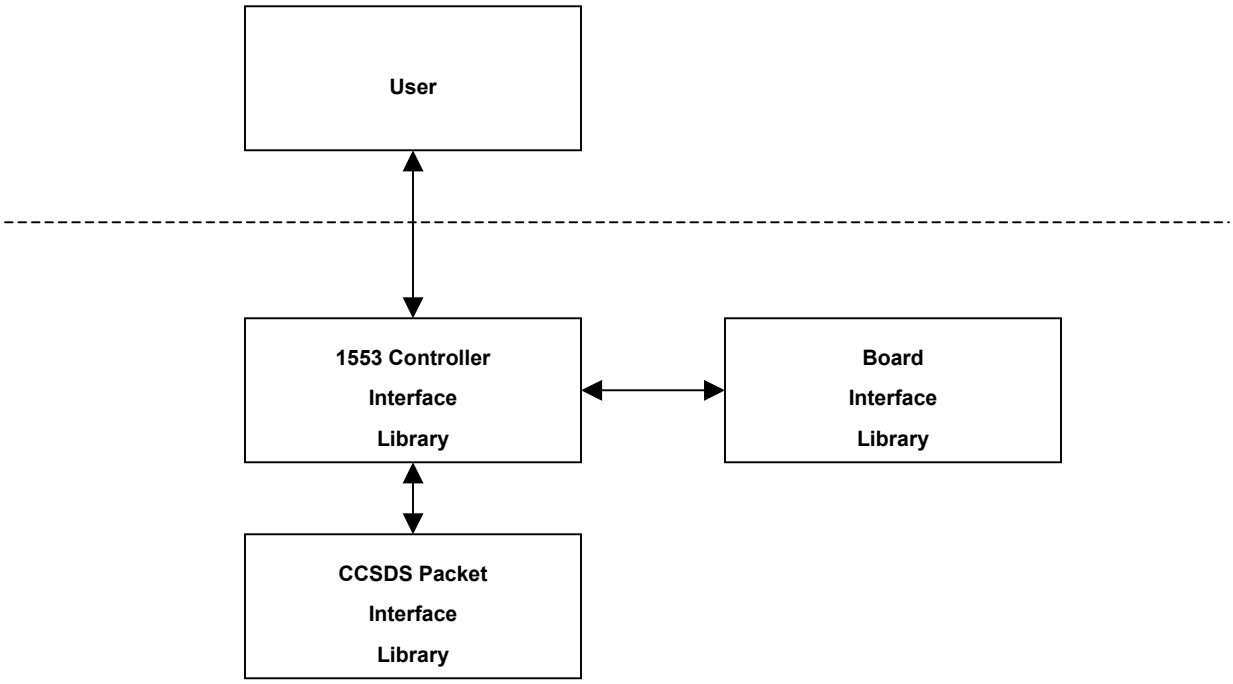
The driver configures the 1553 device to run a series of command lists, which implement the particular bus schedule. These command blocks are written into the device shared memory at startup. At regular intervals, the hardware controller is instructed to execute one of these command lists, termed a 'frame'. The hardware controller will execute the bus commands, each one initiating the transfer of a message on the bus. Once the device has completed execution of a frame command list, it generates an interrupt, which the driver services and schedules the service task for execution. The service task first looks to see if the upcomming frame holds a special sychronous telecommand slot, reserved for a SC attitude, ancillary, or time tone message. If such a telecommand slot is upcomming, a user callback is invoked to provide the telecommand packet for that slot. Next, if the upcomming frame holds a regular telecommand

slot, the CmdRx packet queue is checked for posted packets. If available, a packet is taken from the queue and inserted into the telecommand slot for the next frame. The event service task finally retrieves any CmdTx or Telem packets which may have arrived in the frame that just completed. If so, the packet is placed on the appropriate queue for reading by applications.

2.2 Driver Libraries

Each 1553 driver is split into multiple libraries in order to create a large base of code that is hardware independent. Each driver has three basic libraries as shown below.

Figure 7 – Interrupt Mode Driver Component Libraries



The user only interacts with the driver at the 1553 controller library level. The controller interface library contains all of the device specific knowledge of the target 1553 controller, but is written to a common interface specification; therefore, all controller interface functions present the same prototypes to the user, with just the name prefixes differing. The CCSDS packet interface is a device-independent library that is shared among all 1553 controller drivers. It contains the specifics for implementing the CCSDS packet interfaces and queues as dictated by the GLAST 1553 data protocol. The board interface library deals with any board-specific operations for the target platform that contains the 1553 controller device. It is used privately by the controller interface libraries.

2.2.0 Controller Interface Libraries

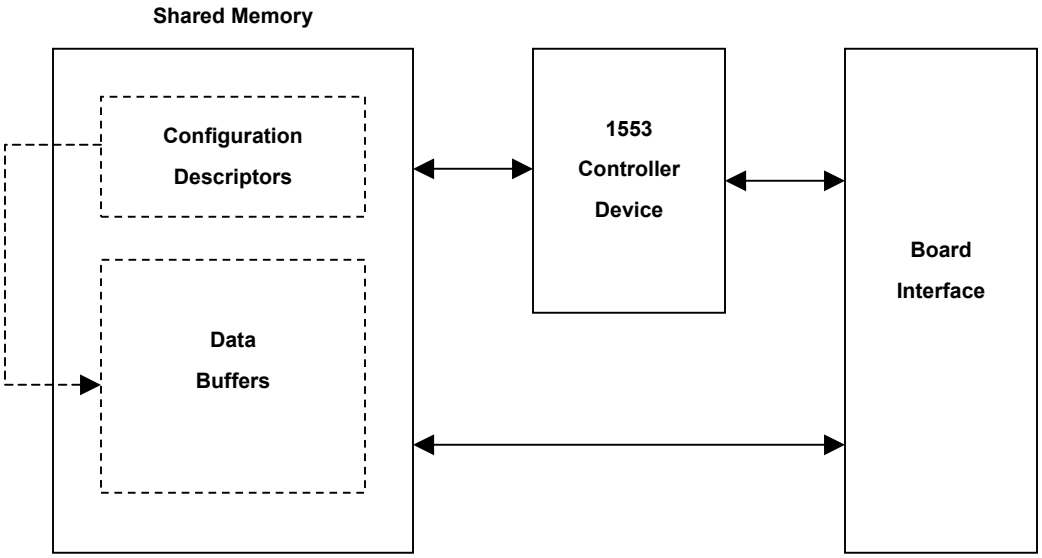
The controller libraries are the software responsible for interfacing to particular 1553 controller devices. For each controller supported, there are multiple versions, with each version supporting a different board interface. In addition, the controller libraries are divided according to the 1553 bus function that is required: bus controller, remote terminal, or bus monitor.

Each of the possible combinations supported is mapped to a different CMX constituent library. Each controller library is labeled:

lib<controller>_<bus function>_<board>.o

The controller interface libraries assume that the 1553 controller registers and associated shared memory can be accesses with standard programmed I/O methods. The figure below shows the basic layout of the hardware interface.

Figure 8 – Interrupt Mode Controller Hardware Interface



The shared memory is divided into two regions. The configuration descriptors define how the 1553 controller device should interface to the bus. For a RT node, the configuration is an array of subaddress descriptors which dictate which subaddresses are valid and which contain pointers to the data buffer region. For a BC node, the configuration area contains command lists detailing the bus schedule. The event log is an area where the controller device can list the details of bus activity. The data buffers region is the area where data messages are transmitted and received from the bus.

2.2.0.0 UTMC Summit

The UTMC Summit is the 1553 controller that is available to the LAT SIU during flight. In the flight configuration, the Summit controller is mounted on the SIB cPCI board. The table below lists the interrupt mode driver combinations supported for the Summit controller.

The SIB variants of the Summit drivers are designed to work with both the flight/EM model SIB board (PCI Device ID = 0x0844) and the evaluation model SIB board (PCI Device ID 0x0001). The differences between the software drivers is minimal.

Table 3- Interrupt Mode UTMC Summit Libraries

Package	Board	Bus Function	Tags	Constituent
CTDB	SIB	Remote Terminal	rad750 mcp750	sumt_rt_sib
		Bus Controller	rad750 mcp750	sumt_bc_sib
	Alphi PMC-1553B	Remote Terminal	mv2304 mcp750	sumt_rt_pmc1553
		Bus Controller	mv2304 mcp750	sumt_bc_pmc1553 vsbc_bc_pmc1553

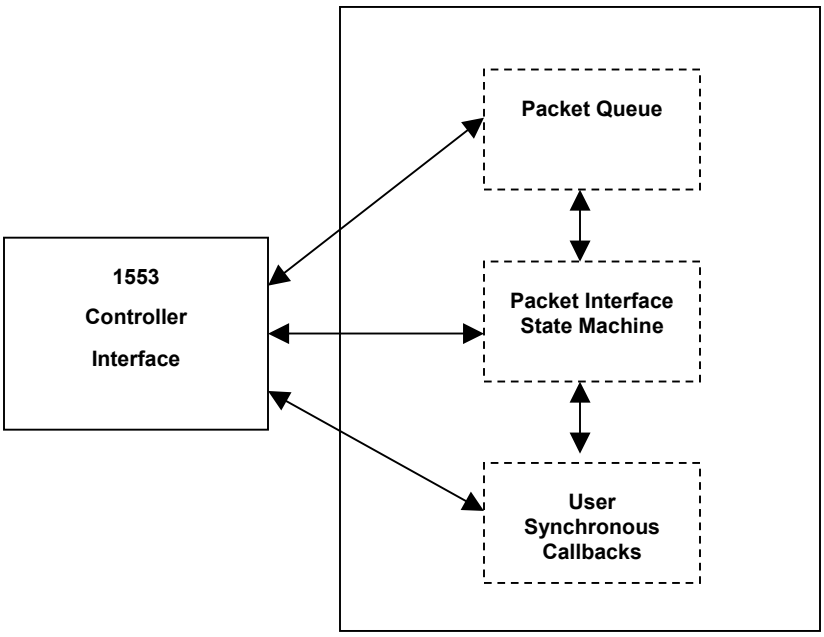
The SIB board, in addition to other LAT I/O functionality, provides a Summit DXE controller chip and 64 KB of shared 1553 RAM. The Alphi PMC-1553B board contains a Summit controller chip and 128 KB of shared 1553 RAM.

2.2.1 CCSDS Interface Libraries

The CCSDS Interface libraries are common to all 1553 drivers. The 1553 controller components will access an exported library of functions that initialize and maintain the packet buffers, queues, and state machines. There is one set of functions for each 1553 subaddress recognized. The CCSDS interface components are also divided according to the 1553 bus function that is required: bus controller, remote terminal, or bus monitor.

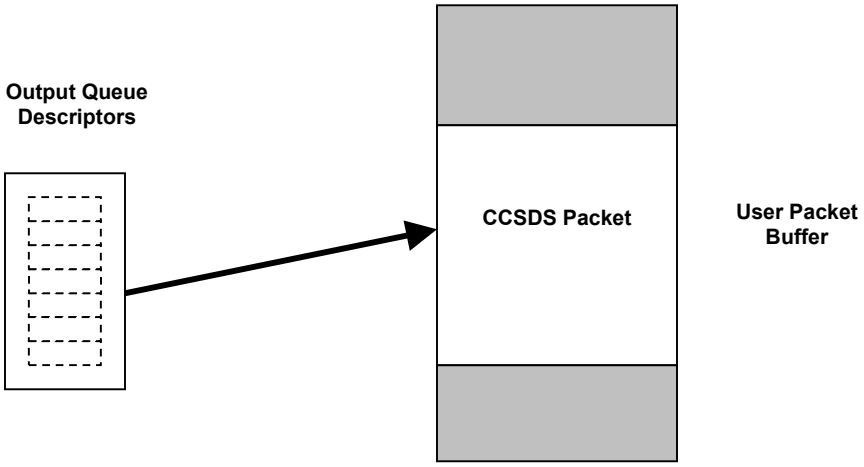
The figure below show the basic layout of the structures maintained by the CCSDS interface libraries.

Figure 9 - 1553 CCSDS Packet Interface Libraries



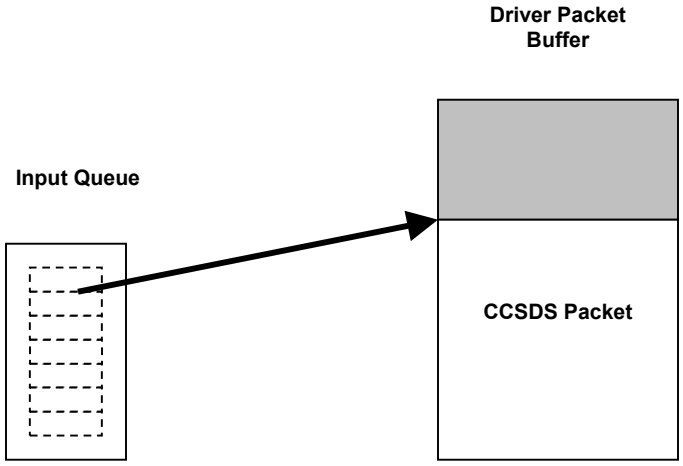
The primary responsibilities of the CCSDS interface libraries are to maintain the driver packet queues and buffer formatting. The figure below shows the basic setup for outgoing packets:

Figure 10 - Driver Output Queue



The driver maintains a queue of output descriptors, which contain a pointer to the user data packet buffer. The driver borrows the user packet buffer while it is on an output queue. The queue descriptors maintain a pointer to the CCSDS packet header, which is given to the driver when calling the driver send function. The driver does not touch the user packet buffer either before or after the actual CCSDS packet contents. The driver will invoke a user supplied callback function after the packet data has been sent and the driver is done borrowing the packet buffer. The figure below shows the basic setup for incoming packets:

Figure 11 – Driver Input Queue



The driver maintains a queue of input packet buffers, which contain a incoming packet data. The user borrows the driver packet buffer while it taken off of an input queue. The user is handed a pointer to the CCSDS packet header when calling the driver receive function. The RT drivers allow the user to reserve space before the start of the CCSDS packet header. The user should never assume the packet buffer extends beyond the end of the actual CCSDS packet.

2.2.2 External Library Requirements

The 1553 interrupt mode drivers require some additional libraries for needed services. A set of functions is required for parsing the CCSDS packet headers. In addition, a library for creating and manipulating queues and buffers is necessary. The MSG libraries are required for condition signaling.

Table 4 – Interrupt Mode Driver Required External Libraries

Package	Constituent
CCSDS	ccsds_pkt
PBS	pbs
MSG	msg_mt
SIB	sib (SIB versions of the drivers only)

Additional libraries may be needed on the target machine to satisfy the dependency requirements of the libraries listed above.

2.3 Driver Controller Interface

Each 1553 controller interrupt mode driver exports a set of functions for initializing and interacting with the 1553 bus interface. The interface varies slightly depending on the bus function the node is supporting.

In all of the function interface definitions, the name “DEV” is used to represent a specific 1553 controller driver.

2.3.0 Remote Terminal

Each function exported by a remote terminal interrupt mode controller driver has the following naming convention:

<DEV>_rt<Function>

DEV is the name of the 1553 controller device and Function is the detailed action to implement.

2.3.0.0 Device Control Functions

The 1553 remote terminal drivers provide a set of functions for initializing and controlling the bus interface.

2.3.0.0.1 DEV_rtSizeof

```
unsigned int DEV_rtSizeof(void)
```

This function returns the size of the device remote terminal descriptor. The user is responsible for allocating a block of memory of at least this size to hold the device descriptor. A pointer to the descriptor area is passed as a parameter to all of the other device interface functions. The drivers all now contain a static instance of the driver descriptor, making this function no longer necessary. Applications should no longer use it; instead use *DEV_rtGet()*.

2.3.0.0.2 DEV_rtGet

```
DEV_RT *DEV_rtGet()
```

Returns a pointer to the static instance of the driver descriptor. *DEV_rtInit()* must be called before use.

2.3.0.0.3 DEV_rtInit

```
unsigned int DEV_rtInit(DEV_RT *rt, const DEV_RT_Bus_Config *busConfig,
    const DEV_RT_Pkt_Config *pktConfig, const DEV_RT_Callback_Config
    *cbConfig, const DEV_RT_Task_Config *taskConfig)
```

This function initializes the 1553 remote terminal driver and device. The *rt* parameter is a pointer to an uninitialized device descriptor. Most of the specific device and bus configuration information is contained in the structure pointed to by the *busConfig* parameter. The remote terminal bus configuration structure is shown below.

```
typedef struct _DEV_RT_Bus_Config
{
 unsigned short rt_addr;
 unsigned short cmx_rx_subaddr;
 unsigned short cmd_tx_subaddr;
 unsigned int cmd_tx_timeout;
 unsigned short telem_subaddr;
 unsigned short telem_msg_count;
 unsigned int telem_timeout;
 unsigned short wrap_subaddr;
 unsigned short mode_flags[32];
} DEV_RT_Bus_Config;
```

The *rt_addr* member specifies the terminal address of the remote terminal. If the remote terminal address has been specified by locked hardware pins on the device, the *rt_addr* member is

ignored, and the driver will used the hardware specified value. The xxx_*subaddr* members specify the subaddress numerical values for each of the recognized subaddresses. The *xxx_timeout* members specify the maximum time in milliseconds between remote terminal transmit requests on the given subaddress before the remote terminal will raise a timeout error condition. The *xxx_msg_count* members specify the number of 1553 data messages each multiple message subaddress is assigned for a bus frame. The *mode_flags* member is an array with one entry for each of the possible mode commands. Each array entry is a word that is the logical OR of configuration bit flags for that mode code. The *DEV_RT_MODE_FLAG_VALID* mode flag specifies that the mode command is valid and accepted by the remote terminal. All information in the *DEV_RT_Bus_Config* structure must be complete and correct when calling *DEV_rtInit()*. Because the remote terminal driver uses this information both to configure the controller device shared memory structures and to configure the CCSDS packet state machines, incorrect or incomplete information will produce unpredictable results. Default values for the addressing members are provided in header *CTDB/CTDB_config.h*.

The *pktConfig* parameter points to an initialized structure of packet configuration information for the remote terminal driver.

```
typedef struct _DEV_RT_Pkt_Config
{
 unsigned int cmd_rx_queue_size;
 unsigned int cmd_rx_pkt_size;
 unsigned int cmd_rx_user_size;
 unsigned int cmd_tx_queue_size;
 unsigned int cmd_tx_pkt_size;
 unsigned int telem_high_queue_size;
 unsigned int telem_low_queue_size;
 unsigned int telem_pkt_size;
 unsigned int hkp_pkt_size;
} DEV_RT_Pkt_Config;
```

The *xxx_queue_size* members limit the maximum number of CCSDS packets that can be placed on each of the queues. The *xxx_pkt_size* members limit the maximum size, including the header, of packets transferred on the subaddresses. Default values for the *xxx_pkt_size* members are provided in header *CTDB/CTDB_config.h*. The *cmd_rx_user* size member allows users to reserve this number of bytes in the telecommand receive packet buffers. This number of bytes is set aside right before the CCSDS telecommand packet header.

The *cbConfig* parameter points to an initialized structure of synchronous callback function handlers for the remote terminal interface.

```
typedef struct _DEV_RT_Callback_Config
{
 DEV_RT_Hkp_Handler hkp_handler;
 void *hkp_arg;
} DEV_RT_Callback_Config;
```

The *hkp_handler* member installs a housekeeping telemetry handler for the synchronous 1553 HKP message slots. This subaddress interface is synchronous so that the HKP data may always be placed at a known location in the 1553 telemetry frame and so that the HKP data is updated regularly at the 4 Hz rate. The callback will be invoked by the driver when a new HKP message may be prepared for delivery on the 1553 bus. The prototype for the HKP callback is shown below.

```
unsigned int DEV_RT_Hkp_Handler(void *buf, void *arg)
```

The *buf* parameter is a memory buffer where the contents of the CCSDS HKP packet data should be placed. The *arg* parameter is user-defined and is the same value as the *hkp_arg* parameter in the *cbConfig* structure. The callback function should attempt to construct a HKP message for output when activated. The HKP packet, including header, must be no larger than the *hkp_pkt_size* value specified at startup. If the HKP handler can provide the message, the data should be copied to the provided buffer and the value *CTDB_SUCCESS* returned from the callback handler. If the HKP handler is for some reason unable to provide the requested data, then it should return value *CTDB_TIMEOUT*. If the *CTDB_SUCCESS* status code is returned, the driver expects to find a valid CCSDS header at the beginning of the buffer upon return from the user handler. The HKP callback function and parameter may be changed by calling *DEV_rtHkpSet()*.

The *taskConfig* parameter for the *DEV_rtInit()* function points to an initialized structure describing the RT service task attributes.

```
typedef struct _DEV_RT_Task_Config
{
   TASK_attr task_attr;
   MSG_Level task_level.
} DEV_RT_TASK_Config;
```

The *TASK_attr* structure is described fully in the *PBS* package documentation. The *MSG_Level* value determines at what severity level the RT service task signals conditions. See *MSG* user manual for full description.

Note that *DEV_rtStart()* must be called before the remote terminal device will start listening on the 1553 bus.

2.3.0.0.4 DEV_rtExit

```
unsigned int DEV_rtExit(DEV_RT *rt)
```

This function stops the 1553 remote terminal device from running and releases all resources acquired during the call to *DEV_rtInit()*.

2.3.0.0.5 DEV_rtStart

```
unsigned int DEV_rtStart(DEV_RT *rt)
```

This function starts the 1553 remote terminal device processing bus messages. Until this point, the remote terminal may not appear to respond to the bus controller. All of the diagnostics counters for the remote terminal are set to '0'.

2.3.0.0.6 DEV_rtStop

```
unsigned int DEV_rtStop(DEV_RT *rt)
```

This function stops the 1553 remote terminal device from processing bus messages.

2.3.0.0.7 DEV_rtHkpSet

```
unsigned int DEV_rtHkpSet(DEV_RT *rt, DEV_RT_Hkp_Handler *handler, void
    *arg)
```

This function changes the HKP callback function invoked in the driver service task when a new HKP packet slot is available in the outgoing telemetry blocks. This callback is set initially in the call to *DEV_rtInit()*. The old callback is unregistered, but users should allow for an old callback to be invoked at least once after making this call. This is because the *DEV_rtHkpSet()* function only makes a request to the RT driver service queue to perform the HKP switchover. This prevents the HKP callback being changed in the middle of processing a telemetry block.

2.3.0.0.8 DEV_rtHkpGet

```
unsigned int DEV_rtHkpGet(DEV_RT *rt, DEV_RT_Hkp_Handler **handler,
    void **arg)
```

This function retrieves the HKP callback function pointer and function argument value.

2.3.0.1 Packet Queue Functions

The 1553 remote terminal drivers provide a set of functions for sending and receiving CCSDS packets across the 1553 bus.

2.3.0.1.1 DEV_rtPktTelemSend

```
unsigned int DEV_rtPktTelemSend(DEV_RT *rt, void *pkt, DEV_RT_Pkt_Free
    *freeHandler, void *freeArg, DEV_RT_Priority priority, const TOC
    *timeout)
```

This function places a CCSDS telemetry packet from the user on one of the remote terminal telemetry output queues. When the telemetry subaddress bandwidth becomes available, the packet contents will be transmitted across the 1553 bus in the telemetry slots. The *pkt* parameter should point to the packet buffer with an already completed CCSDS packet header inserted at the top. The *freeHandler* parameter allows the driver to borrow the packet buffer memory while the packet is on a telemetry output queue. When the remote terminal has finished transmitting the packet, the *freeHandler* function will be called to notify the user and to allow the user to reclaim the packet memory. The format of the *DEV_RT_Pkt_Free* callback is shown below:

```
    void DEV_RT_Pkt_Free(void *pkt, void *arg)
```

The *pkt* parameter is a pointer to the original packet buffer passed to *DEV_rtPktTelemSend()*. The *arg* parameter is the same value as the *freeArg* parameter passed to *DEV_rtPktTelemSend()*. The *priority* parameter specifies whether the packet should go on the high priority or low priority telemetry output queue. Valid values are *DEV_RT_PRIORITY_HIGH* or *DEV_RT_PRIORITY_LOW*. The *timeout* parameter specifies how the caller should behave when the telemetry packet output queue is full. A value of *TOC_NOWAIT* will cause the function to return immediately with an error code of *CTDB_TIMEOUT*. A value of *TOC_FOREVER* or an actual timeout specification will cause the function to pend for available space on the output queue.

2.3.0.1.2 DEV_rtPktCmdTxSend

```
unsigned int DEV_rtPktCmdTxSend(DEV_RT *rt, void *pkt, DEV_RT_Pkt_Free
    *freeHandler, void *freeArg, const TOC *timeout)
```

This function places a CCSDS telecommand output packet from the user on the remote terminal telecommand output queue. When the LAT command transmit subaddress bandwidth becomes available, the packet contents will be transmitted across the 1553 bus in the LAT telecommand output slots. The *pkt* parameter should point to the packet buffer with an already completed CCSDS packet header inserted at the top. The *freeHandler* parameter allows the driver to borrow the packet buffer memory while the packet is on the command output queue. When the remote terminal has finished transmitting the packet, the *freeHandler* function will be called to allow the user to reclaim the packet memory. The format of the *DEV_RT_Pkt_Free* callback is shown below:

```
void DEV_RT_Pkt_Free(void *pkt, void *arg)
```

The *pkt* parameter is a pointer to the original packet buffer passed to *DEV_rtPktCmdTxSend()*. The *arg* parameter is the same value as the *freeArg* parameter passed to *DEV_rtPktCmdTxSend()*. The *timeout* parameter specifies how the caller should behave when the telecommand output packet queue is full. A value of *TOC_NOWAIT* will cause the function to return immediately with an error code of *CTDB_TIMEOUT*. A value of *TOC_FOREVER* or an actual timeout specification will cause the function to pend for available space on the output queue.

2.3.0.1.3 DEV_rtPktCmdRxRecv

```
unsigned int DEV_rtPktCmdRxRecv(DEV_RT *rt, void **pkt, const TOC
    *timeout)
```

This function waits for a new incoming telecommand packet to become available from the remote terminal. When the remote terminal event task has processed the data message containing a received command input packet, the packet will be placed on the command receive queue for a user application to get. The *pkt* parameter will contain a pointer to the packet buffer when the function returns successfully. The packet contents are stored in a buffer allocated from a driver internal pool. The user may borrow this packet memory, but should be careful since a limited number of packet buffers are available to hold the incoming command packets. The pointer returned in the *pkt* parameter points to the beginning of the CCSDS telecommand packet header. A number of bytes equal to the *DEV_RT_Pkt_Config.cmd_rx_user_size* initialization parameter is reserved just before this address. The user may use this memory however he pleased. The user should call *DEV_rtPktCmdRxFree()* when the packet buffer is no longer needed. The *timeout* parameter specifies how the caller should behave when the command input packet queue is empty. A value of *TOC_NOWAIT* will cause the function to return immediately with an error code of *CTDB_TIMEOUT*. A value of *TOC_FOREVER* or an actual timeout specification will cause the function to pend for incoming packets on the input queue.

2.3.0.1.4 DEV_rtPktCmdRxFree

```
unsigned int DEV_rtPktCmdRxFree(DEV_RT *rt, void *pkt)
```

This function returns a remote terminal input telecommand packet buffer to the driver packet buffer pool. It should be called when the user no longer needs the packet buffer provided by the call to *DEV_rtPktCmdRxRecv()*. The function expects to get back the same *pkt* pointer value as was provided by *DEV_rtPktCmdRxRecv()*.

2.3.0.2 Diagnostics Functions

The 1553 remote terminal drivers provide a set of functions for reading diagnostics information from the driver to indicate status and performance.

2.3.0.2.1 DEV_rtDiagGet

```
unsigned int DEV_rtDiagGet(DEV_RT *rt, DEV_RT_Diag *diag)
```

This function reads the diagnostics statistics from the driver and places the values in the structure pointed to by *diag*. See Section 4.2.0 for a description of the *DEV_RT_Diag* structure.

2.3.0.2.2 DEV_rtDiagClear

```
unsigned int DEV_rtDiagClear(DEV_RT *rt)
```

This function zeros all of the diagnostics statistics for the driver.

2.3.1 Bus Controller

Each function exported by a bus controller driver has the following naming convention:

<DEV>_bc<Function>

DEV is the name of the 1553 controller device and Function is the detailed action to implement.

2.3.1.0 Device Control Functions

The 1553 bus controller drivers provide a set of functions for initializing and controlling the bus interface.

2.3.1.0.1 DEV_bcSizeof

```
unsigned int DEV_bcSizeof(void)
```

This function returns the size of the device bus controller descriptor. The user is responsible for allocating a block of memory of at least this size to hold the device descriptor. A pointer to the descriptor area is passed as a parameter to all of the other device interface functions.

The drivers all now contain a static instance of the driver descriptor, making this function no longer necessary. Applications should no longer use it; instead use *DEV_rtGet()*.

2.3.1.0.2 DEV_bcGet

```
DEV_BC *DEV_bcGet()
```

Returns a pointer to the static instance of the driver descriptor. *DEV_bcInit()* must be called before use.

2.3.1.0.3 DEV_bcInit

```
unsigned int DEV_bcInit(DEV_BC *bc, const DEV_BC_Bus_Config *busConfig,
    const DEV_BC_Pkt_Config *pktConfig, const DEV_BC_Callback_Config
    *cbConfig, const DEV_BC_TASK_Config *taskConfig)
```

This function initializes the 1553 bus controller driver and device. The *bc* parameter is a pointer to an uninitialized device descriptor. Most of the specific device and bus configuration information is contained in the structure pointed to by the *busConfig* parameter. See Section 2.4 for the details of this structure's members.

The *pktConfig* parameter points to an initialized structure of packet configuration information for the remote terminal driver.

```
typedef struct _DEV_BC_Pkt_Config
{
 unsigned int cmd_rx_queue_size;
 unsigned int cmd_rx_pkt_size;
 unsigned int cmx_tx_queue_size;
 unsigned int cmd_tx_pkt_size;
 unsigned int telem_queue_size;
 unsigned int telem_pkt_size;
} DEV_BC_Pkt_Config;
```

The *xxx_queue_size* members limit the maximum number of CCSDS packets that can be placed on each of the queues. The *xxx_pkt_size* members limit the maximum size, including the header, of packets transferred on the subaddresses. Default values for the *xxx_pkt_size* members are provided in header *CTDB/CTDB_config.h*.

The *cbConfig* parameter points to an initialized structure of synchronous callback function handlers for the bus controller interface.

```
typedef struct _DEV_BC_Callback_Config
{
 DEV_BC_Sync_Cmd_Handler sync_cmd_handler;
 void *sync_cmd_arg;
} DEV_BC_Callback_Config;
```

The *sync_cmd_handler* member installs a synchronous telecommand handler for the 1553 SC time tone, attitude, and ancillary message slots. This subaddress interface is synchronous so that the SC data may always be placed at a known location in the 1553 telemetry frame and so that the data is updated regularly at the proper rate. The callback will be invoked by the driver when a new synchronous telecommand message may be prepared for delivery in the 1553 bus frame. The prototype for the callback is shown below.

```
unsigned int DEV_ BC_Sync_Cmd_Handler(DEV_BC_Subaddr subAddr, void
    *buf, void *arg)
```

The *subAddr* parameter indicates which type of SC telecommand the upcomming slot has scheduled. The possible values are *DEV_BC_SUBADDR_CMD_TT*, *DEV_BC_SUBADDR_CMD_ATT* , or *DEV_BC_SUBADDR_CMD_ANC* . The *buf* parameter is a memory buffer where the contents of the CCSDS telecommand packet data should be placed. The *arg* parameter is user-defined and is the same value as the *sync_cmd_arg* parameter in the *cbConfig* structure. The callback function should attempt to construct a telecommand message for output when activated. The telecommand packet, including header, must be no larger than the 62 byte GLAST limit. If the handler can provide the message, the data should be copied to the provided buffer and the value *CTDB_SUCCESS* returned from the callback handler. If the synchronous telecommand handler is for some reason unable to provide the requested data, then it should return value *CTDB_TIMEOUT*. If the *CTDB_SUCCESS* status code is returned, the driver expects to find a valid CCSDS header at the beginning of the buffer upon return from the user handler. The callback function and parameter may be changed by calling *DEV_bcSyncCmdSet()*.

The *taskConfig* parameter for the *DEV_bcInit()* function points to an initialized structure describing the BC service task attributes.

```
typedef struct _DEV_BC_Task_Config
{
   TASK_attr task_attr;
   MSG_Level task_level.
} DEV_BC_TASK_Config;
```

The *TASK_attr* structure is described fully in the *PBS* package documentation. The *MSG_Level* value determines at what severity level the BC service task signals conditions. See *MSG* user manual for full description.

Note that *DEV_bcStart()* must be called before the bus controller device will start commanding on the 1553 bus.

2.3.1.0.4 DEV_bcExit

```
unsigned int DEV_bcExit(DEV_BC *bc)
```

This function stops the 1553 bus controller device from running and releases all resources acquired during the call to *DEV_bcInit()*.

2.3.1.0.5 DEV_bcStart

```
unsigned int DEV_bcStart(DEV_BC *bc)
```

This function starts the 1553 bus controller device commanding bus messages. Until this point, the bus will remain idle. The driver maintains an internal timer that will restart the bus schedule on a new frame 25 times per second.

2.3.1.0.6 DEV_bcStop

```
unsigned int DEV_bcStop(DEV_BC *bc)
```

This function stops the 1553 remote terminal device from processing bus messages. Since this function really just disables the internal software frame timer, the bus controller will not actually stop until all data messages and mode commands for the current frame have completed.

2.3.1.0.7 DEV_bcSyncCmdSet

```
unsigned int DEV_bcSyncCmdSet(DEV_BC *bc, DEV_BC_Sync_Cmd_Handler
    *handler, void *arg)
```

This function changes the synchronous telecommand callback function invoked in the driver service task when a new SC telecommand packet slot is available in the bus schedule. This callback is set initially in the call to *DEV_bcInit()*. The old callback is unregistered, but users should allow for an old callback to be invoked at least once after making this call.

2.3.1.0.8 DEV_bcSyncCmdGet

```
unsigned int DEV_bcSyncCmdGet(DEV_BC *bc, DEV_BC_Sync_Cmd_Handler
    **handler, void **arg)
```

This function retrieves the synchronous telecommand callback function pointer and function argument value.

2.3.1.0.9 DEV_bcBusSwitch

```
unsigned int DEV_bcBusSwitch(DEV_BC *bc, DEV_BC_Bus_Side side)
```

This function changes whether the 1553 ‘A’ or ‘B’ busses is used as primary. All messages will first be attempted on the primary bus.

2.3.1.1 Packet Queue Functions

The 1553 bus controller drivers provide a set of functions for sending and receiving CCSDS packets across the 1553 bus.

2.3.1.1.1 DEV_bcPktTelemRecv

```
unsigned int DEV_bcPktTelemRecv(DEV_BC *bc, void **pkt, const TOC
      *timeout)
```

This function waits for a new telemetry packet to become available from the bus controller. When the bus controller event task has processed the last data message in a received telemetry packet, the packet will be placed on the telemetry receive queue for a user application to get. The *pkt* parameter will contain a pointer to the packet buffer when the function returns successfully. The packet contents are stored in a buffer allocated from a driver internal pool. The user may borrow this packet memory, but should be careful since a limited number of packet buffers are available to hold the incoming telemetry packets. The user should call *DEV_bcPktTelemFree()* when the packet buffer is no longer needed. The *timeout* parameter specifies how the caller should behave when the telemetry input packet queue is empty. A value of *TOC_NOWAIT* will cause the function to return immediately with an error code of *CTDB_TIMEOUT*. A value of *TOC_FOREVER* or an actual timeout specification will cause the function to pend for incoming packets on the input queue.

2.3.1.1.2 DEV_bcPktTelemFree

```
unsigned int DEV_bcPktTelemFree(DEV_BC *bc, void *pkt)
```

This function returns a bus controller telemetry packet to the driver packet buffer pool. It should be called when the user no longer needs the packet buffer provided by the call to *DEV_bcPktTelemRecv()*.

2.3.1.1.3 DEV_bcPktCmdTxRecv

```
unsigned int DEV_bcPktCmdTxRecv(DEV_BC *bc, void **pkt, const TOC
      *timeout)
```

This function waits for a new LAT telecommand output packet to become available from the bus controller. When the bus controller event task has processed the last data message in a received telecommand packet, the packet will be placed on the telecommand receive queue for a user application to get. The *pkt* parameter will contain a pointer to the packet buffer when the function returns successfully. The packet contents are stored in a buffer allocated from a driver internal pool. The user may borrow this packet memory, but should be careful since a limited number of packet buffers are available to hold the incoming telecommand packets. The user should call *DEV_bcPktCmdTxFree()* when the packet buffer is no longer needed. The *timeout* parameter specifies how the caller should behave when the telecommand input packet queue is empty. A value of *TOC_NOWAIT* will cause the function to return immediately with an error code of *CTDB_TIMEOUT*. A value of *TOC_FOREVER* or an actual timeout specification will cause the function to pend for incoming packets on the input queue.

2.3.1.1.4 DEV_bcPktCmdTxFree

```
unsigned int DEV_bcPktCmdTxFree(DEV_BC *bc, void *pkt)
```

This function returns a bus controller LAT telecommand output packet to the driver packet buffer pool. It should be called when the user no longer needs the packet buffer provided by the call to *DEV_bcPktCmdTxRecv()*.

2.3.1.1.5 DEV_bcPktCmdRxSend

```
unsigned int DEV_bcPktCmdRxSend(DEV_BC *bc, void *pkt, DEV_BC_Pkt_Free
    *freeHandler, void *freeArg, const TOC *timeout)
```

This function places a CCSDS command packet from the user on the bus controller command output queue. It will arrive at the remote terminal on the telecommand receive subaddress. When the command receive subaddress bandwidth becomes available, the packet contents will be transmitted across the 1553 bus in the command receive slots. The *pkt* parameter should point to the packet buffer with an already completed CCSDS packet header inserted at the top. The *freeHandler* parameter allows the driver to borrow the packet buffer memory while the packet is on the command output queue. When the bus controller event task has finished transmitting the packet, the *freeHandler* function will be called to allow the user to reclaim the packet memory. The format of the *DEV_BC_Pkt_Free* callback is shown below:

```
    void DEV_BC_Pkt_Free(void *pkt, void *arg)
```

The *pkt* parameter is a pointer to the original packet buffer passed to *DEV_bcPktCmdRxSend()*. The *arg* parameter is the same value as the *freeArg* parameter passed to *DEV_bcPktCmdRxSend()*. The *timeout* parameter specifies how the caller should behave when the command output packet queue is full. A value of *TOC_NOWAIT* will cause the function to return immediately with a MSG code of *CTDB_TIMEOUT*. A value of *TOC_FOREVER* or an actual timeout specification will cause the function to pend for incoming packets on the input queue.

2.4 Driver Bus Schedule Configuration

The bus controller driver provides some flexibility for the parameterization of the 1553 bus schedule. The schedule model, however, is dependent on the GLAST 1553 data protocol.

2.4.0 GLAST Bus Schedule Frame Format

The LAT bus controller drivers provide the capability of running 40 millisecond bus frames. The cycle of frames is repeated every 25 frames, which equals a one second period. The number of distinct frames in each cycle is defined by the macro *DEV_BC_NUM_BUS_FRAME* to be ‘25’. Each frame in the cycle can have up to *DEV_BC_NUM_BUS_CMD* bus slots allocated within the 40 millisecond bus frame. The bus controller driver will activate each new frame in the list at a rate of 25 Hz. At the end of the bus slot command list for each frame, the bus controller device will go idle until restarted by the frame timer.

2.4.1 Bus Controller Driver Configuration

The bus controller bus configuration structures are shown below.

```
typedef struct _DEV_BC_Bus_Frame
{
  unsigned int num_cmd;
  DEV_Bus_Cmd bus_cmd[DEV_BC_NUM_BUS_CMD];
} DEV_BC_Bus_Frame;

typedef struct _DEV_BC_Bus_Config
{
 unsigned short bus_side;
 unsigned short rt_addr;
  unsigned short cmd_rx_subaddr;
  unsigned short cmd_tx_subaddr;
  unsigned short telem_subaddr;
  unsigned short telem_msg_count;
  DEV_BC_Bus_Frame bus_frame[DEV_BC_NUM_BUS_FRAME];
} DEV_BC_Bus_Config;
```

The *bus_side* member specifies whether the bus controller uses 1553 bus side A or bus side B as the primary bus side. The values *DEV_BC_BUS_SIDE_A* or *DEV_BC_BUS_SIDE_B* may be used to set the primary bus. The *rt_addr* member is the terminal address of the LAT SIU node. Note that the bus controller drivers only support communications with a single remote terminal. The *xxx_subaddr* members specify the subaddress numerical values for each of the recognized subaddresses. The *xxx_msg_count* members specify the number of 1553 data messages each multiple message subaddress is assigned for a bus frame. Default values for the addressing members are provided in header *CTDB/CTDB_config.h*.

Each of the 25 *bus_frame* members describes the content of a single bus frame. The *num_cmd* member specifies how many slots are scheduled on the bus for each frame. It is limited to a maximum value of *DEV_BC_NUM_BUS_CMD*. The *bus_cmd* member array details the content and timing of the bus schedule for each 40-millisecond bus frame. Note that the general frame schedule is constant for a single invocation of the bus controller driver. Each *DEV_BC_Bus_Cmd* member of the array describes the particular bus activity for each possible slot. The *DEV_BC_Bus_Cmd* structure is shown below.

```
typedef struct _DEV_BC_Bus_Cmd
{
  unsigned short sub_addr;
  unsigned short time_delay;
  unsigned short flags;
} DEV_BC_Bus_Cmd;
```

The *sub_addr* member is the subaddress for the bus slot command. It should be one of the following values: *DEV_BC_SUBADDR_CMD_RX*, *DEV_BC_SUBADDR_CMD_TT*, *DEV_BC_SUBADDR_CMD_ATT*, *DEV_BC_SUBADDR_CMD_ANC*, *DEV_BC_SUBADDR_CMD_TX*, *DEV_BC_SUBADDR_TELEM_DATA*, or *DEV_BC_SUBADDR_TELEM_END*. *The DEV_BC_SUBADDR_TELEM_DATA* slots indicate the actual telemetry data transfer commands. The *DEV_BC_SUBADDR_TELEM_END* slots indicate

the telemetry block completion write command. The *DEV_BC_SUBADDR_CMD_TT*, *DEV_BC_SUBADDR_CMD_ATT*, and *DEV_BC_SUBADDR_CMD_ANC* slots indicate sychronous SC time tone, attitude, or ancillary telecommands. The *tx_flag* member indicates whether the command is a transmit action (RT to BC) or a receive action (BC to RT). The *time_delay* member gives the amount of time the bus controller device should pause after the current bus slot before the next bus command is executed. The delay value is specified in microseconds. Because the time delay values are short and should be precise, the bus slot timing configuration depends on the capability of the bus controller to provide a hardware timing mechanism. The flags member specify that a special action should take place for this bus slot. The *DEV_BC_BUS_FLAG_SKIP* indicates an idle slot on the 1553 bus. For these commands, only the time_delay value of the bus command structure is employed. The *DEV_BC_BUS_FLAG_BCAST* indicates that the bus command should be broadcast to all remote terminals. All bus commands transfer the maximum 32 16-bit word messages in every defined slot.

2.5 Driver Examples

These examples show how to setup and interact with the interrupt mode 1553 drivers.

2.5.0 Remote Terminal

This example shows how to setup a Summit 1553 interrupt mode driver for the remote terminal bus function.

```
  /* the remote terminal bus configuraion */


  #include "CTDB/CTDB_rt.h"
  #include "CTDB/SUMT_rt.h"
  #include "CTDB/CTDB_config.h"

  #include "PBS/MBA.h"
  #include "CCSDS/CCSDS_pkt.h"
  #include "MSG/MSG_pubdefs.h"


 0   SUMT_RT_Bus_Config sumt_bus_config =
     {
 1       CTDB_CONFIG_RT_ADDR,              /* rt_addr */
 2       CTDB_CONFIG_SUBADDR_CMD_RX,       /* cmd_rx_subaddr */
 3       CTDB_CONFIG_SUBADDR_CMD_TX,       /* cmd_tx_subaddr */
 4       500,                              /* cmd_tx_timeout */
 5       CTDB_CONFIG_SUBADDR_TELEM_FIRST,  /* telem_subaddr */
 6       CTDB_CONFIG_SUBADDR_TELEM_NUM,    /* telem_msg_count */
 7       500,                              /* telem_timeout */
 8       CTDB_CONFIG_SUBADDR_WRAP,         /* wrap_subaddr */
 9       {
             0, 0,
             SUMT_RT_MODE_FLAG_VALID,          /* mode code 2 */
             0,
             SUMT_RT_MODE_FLAG_VALID,          /* mode code 4 */
             SUMT_RT_MODE_FLAG_VALID,          /* mode code 5 */
             0, 0,
             SUMT_RT_MODE_FLAG_VALID,          /* mode code 8 */
             0, 0, 0, 0, 0, 0, 0,
             0, 0, 0, 0, 0, 0, 0, 0,
             0, 0, 0, 0, 0, 0, 0, 0,
         },
     };

     /* the device packet configuration */

10   SUMT_RT_Pkt_Config sumt_pkt_config =
     {
11         4,                                /* cmd_rx_queue_size */
12         CTDB_CONFIG_PKT_SIZE_CMD_RX,      /* cmd_rx_pkt_max_size  */
13         16,                               /* cmd_rx_user_size */
14         4,                                /* cmd_tx_queue_size */
15        CTDB_CONFIG_PKT_SIZE_CMD_TX,       /* cmd_tx_pkt_max_size */
16        32,                                /* telem_high_queue_size */
17        128,                               /* telem_low_queue_size */
18         CTDB_CONFIG_PKT_SIZE_TELEM,       /* telem_pkt_max_size */
19        CTDB_CONFIG_PKT_SIZE_HKP           /* hkp_pkt_max_size */
     };

     /* the device callback config */

20   SUMT_RT_Callback_Config sumt_cb_config =
     {
           rt_hkp_handle,                /* hkp_handler */
           NULL,                         /* hkp_arg */
```

```
      };

      /* the device service task config */

 21   SUMT_RT_Task_Config sumt_task_config =
      {
            {
                  "RT_Task",              /* the RT task name */
                  60,                     /* the RT task priority */
                  0,                      /* the RT task options */
                  NULL,                   /* the RT task stack address */
                  0,                      /* the RT task size */
            },

            MSG_LVL_ERROR,                /* the RT task MSG severity level */
      };


      void sumt_rt_go(void)
      {
        SUMT_RT *rt;
        unsigned int size
        unsigned int status;
        unsigned char *pkt;
        unsigned short telemSeq = 0, hkpSeq = 0;


      /*----------------------- initialize the device ----------------------*/

      /* get a pointer to the RT object */

      rt = SUMT_rtGet();

      /* initialize the RT */

 22   status = SUMT_rtInit(rt, &sumt_bus_config, &sumt_pkt_config, &sumt_cb_config,
&sumt_task_config);
      if(_msg_success(status) == 0)
      {
            /* error handler */
            return;
      }

      /* start the remote terminal */

 23   status = SUMT_rtStart(rt);
      if(_msg_success(status) == 0)
      {
            /* error handler */
            return;
      }

 24   /*------------------  send a telemetry packet  ----------------------*/

      /* allocate a packet buffer */

      size = (CCSDS_pktHdrSizeof(CCSDS_PKT_TYPE_TELEM) + TELEM_DATA_SIZE);
```

```
      pkt = MBA_alloc(size);
      if(pkt == NULL)
      {
            printf("cannot allocate pkt buffer\n");
            return;
      }

      /* fill in the CCSDS header */

      status = CCSDS_pktHdrCreate(pkt, CCSDS_PKT_TYPE_TELEM, RT_APID_TELEM,
            CCSDS_PKT_SEQ_NONE, telemSeq, TELEM_DATA_SIZE, time(NULL), 0);
      ++telemSeq;
      if(_msg_success(status) == 0)
      {
            /* error handler */
            MBA_free(pkt);
            return;
      }

      /* queue the packet */

      status = SUMT_rtPktTelemSend(rt, pkt, rt_pkt_free, NULL,
SUMT_RT_PRIORITY_HIGH, TOC_FOREVER);
      if(_msg_success(status) == 0)
      {
            /* error handler */
            MBA_free(pkt);
            return;
      }

 25   /*-------------------- receive a telecommand packet ----------------------*/

      status = SUMT_rtPktCmdRxRecv(rt, &pkt, TOC_FOREVER);
      if(_msg_success(status) == 0)
      {
            /* error handler */
            return;
      }

      /* after examining command contents, free the packet */

      SUMT_rtPktCmdRxFree(pkt);

    }

 26  void rt_hkp_handle(void *buf, void *arg)
    {
            /* fill in the HKP packet */

            CCSDS_pktHdrCreate(buf, CCSDS_PKT_TYPE_TELEM, RT_APID_HKP,
                 CCSDS_PKT_SEQ_NONE, hkpSeq, HKP_DATA_SIZE, time(NULL), 0);
            ++hkpSeq;

            return(CTDB_SUCCESS);
     }
```

```
27  void rt_pkt_free(void *pkt, void *arg)
    {
      MBA_free(pkt);
    }
```

Notes on Interrupt Mode Remote Terminal Driver Example

0. This instance of the *SUMT_RT_Bus_Config* structure defines the 1553 bus interface of the remote terminal.

1. The terminal address of the LAT SIU remote terminal is set to ‘3’.

2. The telecommand receive subaddress value for the remote terminal is set to ‘27’.

3. The telecommand transmit subaddress value for the remote terminal is set to ‘29’.

4. The telecommand transmit subaddress read attempt timeout for the remote terminal is set to ‘500’ milliseconds.

5. The base telemetry subaddress value for the remote terminal is set to ‘11’.

6. There are ‘15’ 1553 telemetry data messages for each frame. The remote terminal will transfer telemetry output blocks on subaddresses ’11’ through ‘25’.

7. The telemetry subaddress read attempt timeout for the remote terminal is set to ‘500’ milliseconds.

8. The data wraparound subaddress for the remote terminal is ‘30’.

9. There are 4 valid 1553 mode commands recognized, numbers 2, 4, 5, and 8.

10. This instance of the *SUMT_RT_Pkt_Config* structure sets up the CCSDS packet interface for the driver.

11. The CCSDS telecommand receive packet input queue is limited to ‘4’ packets.

12. Each packet on the CCSDS telecommand receive queue is limited to no more than ‘62’ bytes.

13. Each packet buffer for the CCSDS telecommand receive queue will reserve ‘16’ bytes right before the CCSDS telecommand header.

14. The CCSDS telecommand transmit packet output queue is limited to ‘4’ packets.

15. Each packet on the CCSDS telecommand transmit queue is limited to no more than ‘62’ bytes.

16. The CCSDS high priority telemetry packet output queue is limited to ‘32’ packets

17. The CCSDS low priority packet output queue is limited to ‘128’ packets,

18. Each telemetry packet is limited to no more that ‘842’ bytes.

19. CCSDS HKP telemetry packets are limited to no more than ‘116’ bytes.

20. This instance of the *SUMT_RT_Callback_Config* structure sets up the synchronous callback handler functions.

21. This instance of the *SUMT_RT_Task_Config* structure describes the RT service task attributes. In this example, defaults are used for the stack attributes.

22. Memory must be allocated for the RT driver descriptor object.

23. Here is where the *SUMT_RT* device descriptor is initialized. Pointers to the configuration structures are passed in as parameters. The remote terminal 1553 controller device is fully configured, but left disconnected from the 1553 bus.

24. The call to *SUMT_rtStart()* connects the remote terminal to the 1553 bus and starts processing bus events.

25. This section of the example shows the allocation, initialization, and sending of a CCSDS packet to the high priority telemetry subaddress packet queue.

26. This section of the example shows the user waiting to receive a CCSDS packet from the telecommand input subaddress queue. Once the packet is received, the user eventually returns the packet to the driver packet pool.

27. This function is installed as the HKP packet telemetry handler. It is called when the driver is ready to prepare a new HKP packet for transmission. This example fills in a packet header and returns *CTDB_SUCCESS* to indicate that a packet was created and copied to the buffer provided by the driver.

28. This is the function registered as the packet free handler for the telemetry packet send example. This simple example uses *MBA_alloc()* and *MBA_free()*. This is generally not recommended for actual remote terminal use with high traffic.

2.5.1 Bus Controller

This example shows how to setup a Summit 1553 interrupt mode driver for the bus controller bus function.

```
   #include "CTDB/SUMT_bc.h"
   #include "CTDB/CTDB_config.h"

   #include "PBS/MBA.h"
   #include "CCSDS/CCSDS_pkt.h"
   #include "MSG/MSG_pubdefs.h"

   /* the bus controller bus configuration */

 0 SUMT_BC_Bus_Config sumt_bus_config =
   {
 1     SUMT_BC_BUS_SIDE_A,                  /* bus_side */
 2     CTDB_CONFIG_RT_ADDR,                 /* rt_addr */
 3     CTDB_CONFIG_SUBADDR_CMD_RX,          /* cmd_rx_subaddr */
 4     CTDB_CONFIG_SUBADDR_CMD_TX,          /* cmd_tx_subaddr */
 5     CTDB_CONFIG_SUBADDR_TELEM_FIRST,     /* telem_subaddr */
 6     CTDB_CONFIG_SUBADDR_TELEM_NUM,      /* telem_msg_count */
 7     CTDB_CONFIG_SUBADDR_WRAP,          /* wrap_subaddr */


 8      {              /* frame 0 */
           18,
           {SUMT_BC_SUBADDR_CMD_RX,            0, 0},
           {0,                                 1472, SUMT_BC_BUS_FLAG_SKIP},
           {SUMT_BC_SUBADDR_TELEM_DATA,  700,  0},
           {SUMT_BC_SUBADDR_TELEM_DATA,  700,  0},
           {SUMT_BC_SUBADDR_TELEM_DATA,  700,  0},
           {SUMT_BC_SUBADDR_TELEM_DATA,  700,  0},
           {SUMT_BC_SUBADDR_TELEM_DATA,  700,  0},
           {SUMT_BC_SUBADDR_TELEM_DATA,  700,  0},
           {SUMT_BC_SUBADDR_TELEM_DATA,  700,  0},
           {SUMT_BC_SUBADDR_TELEM_DATA,  700,  0},
           {SUMT_BC_SUBADDR_TELEM_DATA,  700,  0},
           {SUMT_BC_SUBADDR_TELEM_DATA,  700,  0},
           {SUMT_BC_SUBADDR_TELEM_DATA,  700,  0},
           {SUMT_BC_SUBADDR_TELEM_DATA,  700,  0},
           {SUMT_BC_SUBADDR_TELEM_DATA,  700,  0},
           {SUMT_BC_SUBADDR_TELEM_DATA,  700,  0},
           {SUMT_BC_SUBADDR_TELEM_DATA,  700,  0},
           {SUMT_BC_SUBADDR_TELEM_END,         0,    0},
        },


 9      {              /* frame 1 */
           1,
           {SUMT_BC_SUBADDR_CMD_RX,      0,    0}
        },


10      {              /* frame 2 */
           1,
           {SUMT_BC_SUBADDR_CMD_RX,      0,    0}
        },


11      {              /* frame 3 */
```

```
            1,
            {SUMT_BC_SUBADDR_CMD_TX,      0,    0},
         },

12       {              /* frame 4 */
            1,
           {SUMT_BC_SUBADDR_CMX_ATT,      0,    0}
         },

13       {              /* frame 5 */
            18,
            {SUMT_BC_SUBADDR_CMD_TT,            0, 0},
            {0,                                 1472, SUMT_BC_BUS_FLAG_SKIP},
            {SUMT_BC_SUBADDR_TELEM_DATA,  700,  0},
            {SUMT_BC_SUBADDR_TELEM_DATA,  700,  0},
            {SUMT_BC_SUBADDR_TELEM_DATA,  700,  0},
            {SUMT_BC_SUBADDR_TELEM_DATA,  700,  0},
            {SUMT_BC_SUBADDR_TELEM_DATA,  700,  0},
            {SUMT_BC_SUBADDR_TELEM_DATA,  700,  0},
            {SUMT_BC_SUBADDR_TELEM_DATA,  700,  0},
            {SUMT_BC_SUBADDR_TELEM_DATA,  700,  0},
            {SUMT_BC_SUBADDR_TELEM_DATA,  700,  0},
            {SUMT_BC_SUBADDR_TELEM_DATA,  700,  0},
            {SUMT_BC_SUBADDR_TELEM_DATA,  700,  0},
            {SUMT_BC_SUBADDR_TELEM_DATA,  700,  0},
            {SUMT_BC_SUBADDR_TELEM_DATA,  700,  0},
            {SUMT_BC_SUBADDR_TELEM_DATA,  700,  0},
            {SUMT_BC_SUBADDR_TELEM_DATA,  700,  0},
            {SUMT_BC_SUBADDR_TELEM_END,         0,    0},
         },


14       {              /* frame 6 */
            1,
            {SUMT_BC_SUBADDR_CMD_RX,      0,    0}
         },


15       {              /* frame 7 */
            1,
            {SUMT_BC_SUBADDR_CMD_RX,      0,    0}
         },


16       {              /* frame 8 */
            1,
            {SUMT_BC_SUBADDR_CMD_TX,      0,    0},
         },

17       {              /* frame 9 */
            1,
           {SUMT_BC_SUBADDR_CMX_ATT,      0,    0}
         },

18       {              /* frame 10 */
            18,
            {SUMT_BC_SUBADDR_CMD_RX,            0, 0},
```

```
            {0,                                 1472, SUMT_BC_BUS_FLAG_SKIP},
            {SUMT_BC_SUBADDR_TELEM_DATA,  700,  0},
            {SUMT_BC_SUBADDR_TELEM_DATA,  700,  0},
            {SUMT_BC_SUBADDR_TELEM_DATA,  700,  0},
            {SUMT_BC_SUBADDR_TELEM_DATA,  700,  0},
            {SUMT_BC_SUBADDR_TELEM_DATA,  700,  0},
            {SUMT_BC_SUBADDR_TELEM_DATA,  700,  0},
            {SUMT_BC_SUBADDR_TELEM_DATA,  700,  0},
            {SUMT_BC_SUBADDR_TELEM_DATA,  700,  0},
            {SUMT_BC_SUBADDR_TELEM_DATA,  700,  0},
            {SUMT_BC_SUBADDR_TELEM_DATA,  700,  0},
            {SUMT_BC_SUBADDR_TELEM_DATA,  700,  0},
            {SUMT_BC_SUBADDR_TELEM_DATA,  700,  0},
            {SUMT_BC_SUBADDR_TELEM_DATA,  700,  0},
            {SUMT_BC_SUBADDR_TELEM_DATA,  700,  0},
            {SUMT_BC_SUBADDR_TELEM_DATA,  700,  0},
            {SUMT_BC_SUBADDR_TELEM_END,         0,    0},
        },


19      {               /* frame 11 */
            1,
            {SUMT_BC_SUBADDR_CMD_RX,      0,    0}
        },


20      {               /* frame 12 */
            1,
            {SUMT_BC_SUBADDR_CMD_RX,      0,    0}
        },


21      {               /* frame 13 */
            1,
            {SUMT_BC_SUBADDR_CMD_TX,      0,    0},
        },

22      {               /* frame 14 */
            1,
           {SUMT_BC_SUBADDR_CMX_ATT,      0,    0}
        },

23      {               /* frame 15 */
            18,
            {SUMT_BC_SUBADDR_CMD_RX,            0, 0},
            {0,                                 1472, SUMT_BC_BUS_FLAG_SKIP},
            {SUMT_BC_SUBADDR_TELEM_DATA,  700,  0},
            {SUMT_BC_SUBADDR_TELEM_DATA,  700,  0},
            {SUMT_BC_SUBADDR_TELEM_DATA,  700,  0},
            {SUMT_BC_SUBADDR_TELEM_DATA,  700,  0},
            {SUMT_BC_SUBADDR_TELEM_DATA,  700,  0},
            {SUMT_BC_SUBADDR_TELEM_DATA,  700,  0},
            {SUMT_BC_SUBADDR_TELEM_DATA,  700,  0},
            {SUMT_BC_SUBADDR_TELEM_DATA,  700,  0},
            {SUMT_BC_SUBADDR_TELEM_DATA,  700,  0},
            {SUMT_BC_SUBADDR_TELEM_DATA,  700,  0},
            {SUMT_BC_SUBADDR_TELEM_DATA,  700,  0},
```

```
            {SUMT_BC_SUBADDR_TELEM_DATA,  700,  0},
            {SUMT_BC_SUBADDR_TELEM_DATA,  700,  0},
            {SUMT_BC_SUBADDR_TELEM_DATA,  700,  0},
            {SUMT_BC_SUBADDR_TELEM_DATA,  700,  0},
            {SUMT_BC_SUBADDR_TELEM_END,         0,    0},
         },


24       {              /* frame 16 */
            1,
            {SUMT_BC_SUBADDR_CMD_RX,      0,    0}
         },


25       {              /* frame 17 */
            1,
            {SUMT_BC_SUBADDR_CMD_RX,      0,    0}
         },


26       {              /* frame 18 */
            1,
            {SUMT_BC_SUBADDR_CMD_TX,      0,    0},
         },

27       {              /* frame 19 */
            1,
           {SUMT_BC_SUBADDR_CMX_ATT,      0,    0}
         },

28       {              /* frame 20 */
            1,
            {SUMT_BC_SUBADDR_CMD_ANC,           0, 0},
         },


29       {              /* frame 21 */
            1,
            {SUMT_BC_SUBADDR_CMD_RX,      0,    0}
         },


30       {              /* frame 22 */
            1,
            {SUMT_BC_SUBADDR_CMD_RX,      0,    0}
         },


31       {              /* frame 23 */
            1,
            {SUMT_BC_SUBADDR_CMD_TX,      0,    0},
         },

32       {              /* frame 24 */
            1,
           {SUMT_BC_SUBADDR_CMX_ATT,      0,    0}
         },
```

```
    /* the driver packet configuration */

 33 SUMT_BC_Pkt_Config sumt_pkt_config =
    {
 34  4,                                  /* cmd_rx_queue_size */
 35  CTDB_CONFIG_PKT_SIZE_CMD_RX,        /* cmd_rx_pkt_size */
 36  4,                                  /* cmd_tx_queue_size */
 37  CTDB_CONFIG_PKT_SIZE_CMD_TX,        /* cmd_tx_pkt_size */
 38  32,                                 /* telem_queue_size */
 39  CTDB_CONFIG_PKT_SIZE_TELEM          /* telem_pkt_size */
    };

 40  SUMT_BC_Callback_Config sumt_cb_config =
     {
           bc_sync_cmd,                  /* the sync telecommand callback */
           NULL,                         /* the callback argument */
     };

 41  SUMT_BC_Task_Config sumt_task_config =
     {
           {
                 "BC_Task",              /* the BC task name */
                 70,                     /* the BC task priority */
                 0,                      /* the BC task options */
                 NULL,                   /* the BC task stack address */
                 0,                      /* the BC task size */
           },

           MSG_LVL_ERROR,                /* the BC task MSG severity level */
    };


    void sumt_bc_go(void)
    {
       unsigned int size;
      unsigned int status;
       SUMT_BC *bc;
      unsigned char *pkt;
      unsigned short uplSeq = 0;

    /*----------------------- initialize the device ----------------------*/

      /* get a pointer to the BC object */

      bc = SUMT_bcGet();

      /* initialize the BC */

 42   status = SUMT_bcInit(bc, &sumt_bus_config, &sumt_pkt_config, &sumt_cb_config,
&sumt_task_config);
      if(_msg_success(status) == 0)
      {
           /* error handler */
```

```
            return;
      }

       /* start the bus controller */

43    status = SUMT_bcStart(bc);
      if(_msg_success(status) == 0)
      {
            /* error handler */
            return;
      }

44    /*-------------------  receive a telemetry packet  -----------------------*/

      /* wait for packets at the queue */

      status = SUMT_bcPktTelemRecv(bc, (void**) &pkt, TOC_FOREVER);
      iif(_msg_success(status) == 0)
      {
            /* error handler */
            return;
      }

      /* after examining the telemetry data, free the packet */

      SUMT_bcPktTelemFree(pkt);

45    /*-------------------- send a telecommand packet ---------------------*/

      /* allocate a packet buffer */

      size = (CCSDS_pktHdrSizeof(CCSDS_PKT_TYPE_CMD) + CMD_DATA_SIZE);
      pkt = MBA_alloc(size);
      if(pkt == NULL)
      {
       printf("Cannot allocate pkt buffer\n");
       return;
      }

      /* fill in the CCSDS header */

       status = CCSDS_pktHdrCreate(pkt, CCSDS_PKT_TYPE_CMD, RT_APID_CMD,
            CCSDS_PKT_SEQ_NONE, cmdSeq, CMD_PKT_SIZE, 0, RT_FC_CMD);
      ++cmdSeq;
      if(_msg_success(status) == 0)
      {
       /* error handler */
       MBA_free(pkt);
       return;
      }

      /* insert the packet checksum */

      CCSDS_pktChksumInsert(pkt, size);


      /* queue the packet */
```

```
     status = SUMT_bcPktCmdRxSend(bc, pkt, bc_pkt_free, NULL, TOC_FOREVER);
     if(_msg_success(status) == 0)
     {
       /* error handler */
       MBA_free(pkt);
       return;
     }

    }


 46  void bc_pkt_free(void *pkt, void *arg)
    {
      MBA_free(pkt);
    }


 47  void bc_sync_cmd(SUMT_BC_Subaddr subAddr, void *buf, void *arg)
    {
      if(subAddr == SUMT_BC_SUBADDR_CMD_ATT)
       {
            /* fill in the attitude packet */

            CCSDS_pktHdrCreate(buf, CCSDS_PKT_TYPE_CMD, BC_APID_SC_SYNC,
CCSDS_PKT_SEQ_NONE, 0, ATT_DATA_SIZE, 0, BC_FCODE_SC_ATT);

            /* fill in the packet checksum */

            CCSDS_pktChksumInsert(buf, pktSize);
      }
    }
```

Notes on Interrupt Mode Bus Controller Driver Example

0. This instance of the *SUMT_BC_Bus_Config* structure defines the 1553 bus interface of the bus controller.

1. The bus controller will use 1553 bus side A as the primary.

2. The terminal address of the SIU remote terminal that the bus controller is to communicate with is set to ‘3’.

3. The telecommand receive subaddress value for the remote terminal is set to ‘27’.

4. The telecommand transmit subaddress value for the remote terminal is set to ‘29’.

5. The base telemetry subaddress value for the remote terminal is set to ‘11’.

6. There are ‘15’ 1553 telemetry data messages for each frame. The bus controller will read telemetry output blocks on subaddresses ’11’ through ‘25’.

7. The data wraparound subaddress for the remote terminal is ‘30’.

8. This instance of a *SUMT_BC_Bus_Frame* structure defines the bus slots for frame ‘0’. This frame contains 18 commands:

1) A telecommand receive slot.

2) A series of 15 telemetry data block slots. The telemetry data block slots are followed by the telemetry completion slot. This completes one telemetry packet block transfer.

9. This instance of a *SUMT_BC_Bus_Frame* structure defines the bus slots for frame '1'. This frame contains 1 command:

1) A telecommand receive slot.

10. This instance of a *SUMT_BC_Bus_Frame* structure defines the bus slots for frame '2'. This frame contains 1 command:

1) A telecommand receive slot.

11. This instance of a *SUMT_BC_Bus_Frame* structure defines the bus slots for frame '3'. This frame contains 1 command:

1) A telecommand transmit slot.

12. This instance of a *SUMT_BC_Bus_Frame* structure defines the bus slots for frame '4'. This frame contains 1 command:

1) A synchronous attitude telecommand slot.

13. This instance of a *SUMT_BC_Bus_Frame* structure defines the bus slots for frame '5'. This frame contains 18 commands:

1) A synchronous time tone telecommand slot.

2) A series of 15 telemetry data block slots. The telemetry data block slots are followed by the telemetry completion slot. This completes one telemetry packet block transfer.

14. This instance of a *SUMT_BC_Bus_Frame* structure defines the bus slots for frame '6'. This frame contains 1 command:

1) A telecommand receive slot.

15. This instance of a *SUMT_BC_Bus_Frame* structure defines the bus slots for frame '7'. This frame contains 1 command:

1) A telecommand receive slot.

16. This instance of a *SUMT_BC_Bus_Frame* structure defines the bus slots for frame '8'. This frame contains 1 command:

1) A telecommand transmit slot.

17. This instance of a *SUMT_BC_Bus_Frame* structure defines the bus slots for frame '9'. This frame contains 1 command:

1) A synchronous attitude telecommand slot.

18. This instance of a *SUMT_BC_Bus_Frame* structure defines the bus slots for frame '10'. This frame contains 18 commands:

1) A telecommand receive slot.

2) A series of 15 telemetry data block slots. The telemetry data block slots are followed by the telemetry completion slot. This completes one telemetry packet block transfer.

19. This instance of a *SUMT_BC_Bus_Frame* structure defines the bus slots for frame '11'. This frame contains 1 command:

1) A telecommand receive slot.

20. This instance of a *SUMT_BC_Bus_Frame* structure defines the bus slots for frame '12'. This frame contains 1 command:

1) A telecommand receive slot.

21. This instance of a *SUMT_BC_Bus_Frame* structure defines the bus slots for frame '13'. This frame contains 1 command:

1) A telecommand transmit slot.

22. This instance of a *SUMT_BC_Bus_Frame* structure defines the bus slots for frame '14'. This frame contains 1 command:

1) A synchronous attitude telecommand slot.

23. This instance of a *SUMT_BC_Bus_Frame* structure defines the bus slots for frame '15'. This frame contains 18 commands:

1) A telecommand receive slot.

2) A series of 15 telemetry data block slots. The telemetry data block slots are followed by the telemetry completion slot. This completes one telemetry packet block transfer.

24. This instance of a *SUMT_BC_Bus_Frame* structure defines the bus slots for frame '16'. This frame contains 1 command:

1) A telecommand receive slot.

25. This instance of a *SUMT_BC_Bus_Frame* structure defines the bus slots for frame '17'. This frame contains 1 command:

1) A telecommand receive slot.

26. This instance of a *SUMT_BC_Bus_Frame* structure defines the bus slots for frame '18'. This frame contains 1 command:

1) A telecommand transmit slot.

27. This instance of a *SUMT_BC_Bus_Frame* structure defines the bus slots for frame '19'. This frame contains 1 command:

1) A synchronous attitude telecommand slot.

28. This instance of a *SUMT_BC_Bus_Frame* structure defines the bus slots for frame '20'. This frame contains 1 command:

1) A synchronous ancillary telecommand slot.

29. This instance of a *SUMT_BC_Bus_Frame* structure defines the bus slots for frame '1'. This frame contains 1 command:

1) A telecommand receive slot.

30. This instance of a *SUMT_BC_Bus_Frame* structure defines the bus slots for frame '2'. This frame contains 1 command:

1) A telecommand receive slot.

31. This instance of a *SUMT_BC_Bus_Frame* structure defines the bus slots for frame '3'. This frame contains 1 command:

1) A telecommand transmit slot.

32. This instance of a *SUMT_BC_Bus_Frame* structure defines the bus slots for frame '4'. This frame contains 1 command:

1) A synchronous attitude telecommand slot.

33. This instance of the *SUMT_BC_Pkt_Config* structure sets up the CCSDS packet interface for the driver.

34. The CCSDS telecommand receive packet output queue is limited to ‘4’ packets.

35. Each packet on the telecommand receive output queue is limited to no more than ‘62’ bytes.

36. The CCSDS telecommand transmit packet input queue is limited to ‘4’ packets.

37. Each packet on the telecommand transmit input queue is limited to no more than ‘62’ bytes.

38. The CCSDS telemetry packet input queue is limited to ‘32’ packets.

39. Each packet on the telemetry input queue is limited to no more than ‘958’ bytes.

40. This instance of the *SUMT_RT_Callback_Config* structure sets up the synchronous callback handler functions.

41. This instance of the *SUMT_BC_Task_Config* structure describes the BC service task attributes. In this example, defaults are used for the stack attributes.

42. Here is where the *SUMT_BC* device descriptor is initialized. Pointers to the configuration structures are passed in as parameters. The bus controller 1553 controller device is fully configured, but left disconnected from the 1553 bus.

43. The call to *SUMT_bcStart()* connects the bus controller to the 1553 bus and starts processing bus commands. The bus controller device command list is automatically restarted every frame to drive the bus.

44. This section of the example shows the user waiting to receive a CCSDS packet from the telemetry subaddress queue. Once the packet is received, the user eventually returns the packet to the driver packet pool.

45. This section of the example shows the allocation, initialization, and sending of a CCSDS packet to the telecommand receive subaddress packet queue.

46. This is the function registered as the packet free handler for the upload packet send example. This simple example uses *MBA_malloc()* and *MBA_free()*. This is generally not recommended for actual bus controller use with high traffic.

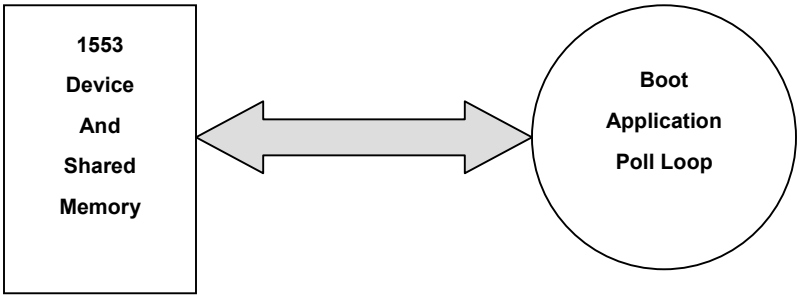
47. This is an example of a synchronous telecommand callback. This case handles the attitude telecommands by filling in the driver provided buffer with a valid CCSDS telecommand.

3 Polled Mode Drivers

3.0 Driver Architecture

The polled mode 1553 drivers are intended for use in boot situations where the services of a RTOS are not available. For the purposes of the LAT, only the remote terminal bus functions will be supported for polled mode drivers. It is also assumed that the 1553 remote terminal is the sole means of communications. This is important to note because the user must constantly poll the interface in order for the driver to work properly.

Figure 12 - Polled Mode Driver Architecture

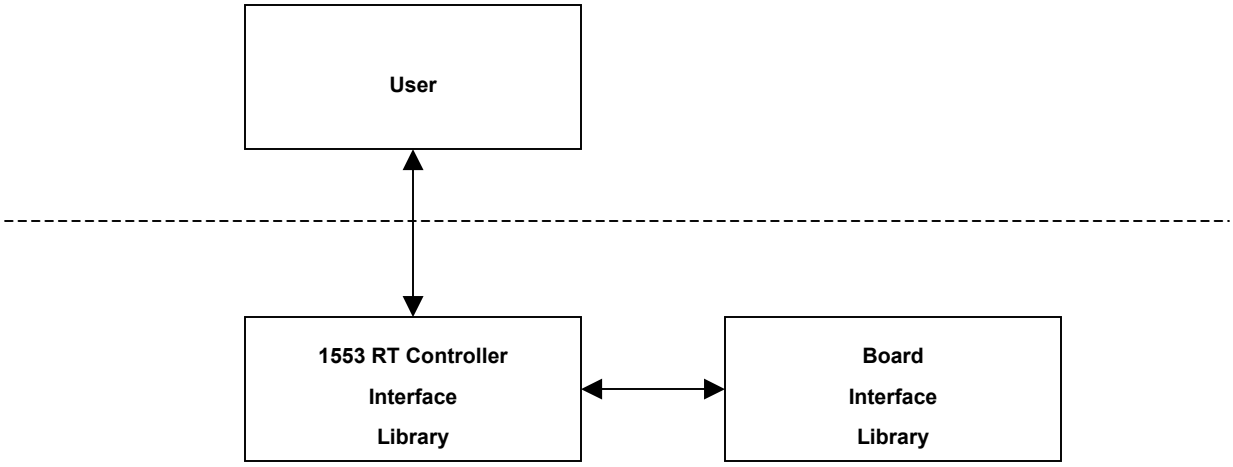


The driver will initialize the 1553 controller device to generate bus status bits in the device shared memory, but no interrupt signal, whenever a valid subaddress message arrives on the bus or when the device reports an error. This event poll loop is driven by the user application. The event polling interface notifies the application of events of interest and transfers a data message to the user if the event is a 1553 receive subaddress data message. The send interface allows the user to transfer data to the 1553 telemetry transmit subaddress.

3.1 Driver Libraries

The polled mode driver libraries have a much simpler structure than the interrupt mode libraries, particularly since the polled mode drivers do not support the parsing of CCSDS packets within the 1553 data messages. Essentially, the task of parsing and validating the contents of the 1553 bus data is pushed back on to the user application. The polled mode interface recognizes the 1553 data message as the data exchange structure.

Figure 13 - Polled Mode Driver Component Libraries



The user only interacts with the polled mode driver at the 1553 controller library level. The controller interface library contains all of the device specific knowledge of the target 1553 controller, but is written to a common interface specification; therefore, all controller interface functions present the same prototypes to the user, with just the name prefixes differing. The board interface library deals with any board-specific operations for the target platform that the 1553 controller will run on. It is used privately by the controller interface libraries.

3.1.0 Controller Interface Libraries

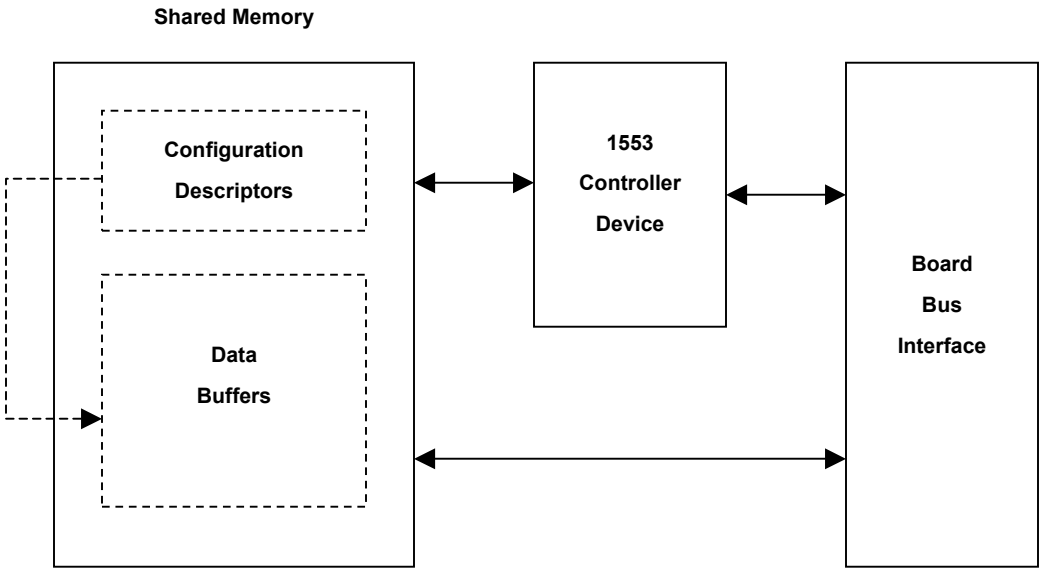
The polled mode controller libraries are the software responsible for interfacing to particular 1553 controller devices. For each controller supported, there are multiple versions, with each version supporting a different board interface.

Each of the possible combinations supported is mapped to a different CMX constituent library. Each controller library is labeled:

lib<controller>_rt_poll_<board>.o

The controller interface libraries assume that the 1553 controller registers and associated shared memory can be accesses with standard programmed I/O methods. The figure below shows the basic layout of the hardware interface.

Figure 14 - Polled Mode Driver Hardware Interface



The shared memory is divided into two regions. The configuration descriptors region is an array of remote terminal subaddress descriptors, which dictate which subaddresses are valid and which contain pointers to the data buffer region. The data buffers region is the area where data messages are transmitted and received from the bus.

3.1.0.0 UTMC Summit

The UTMC Summit is the 1553 controller that is available to the LAT SIU during flight. In the flight configuration, the Summit controller is mounted on the SIB cPCI board. The table below lists the library polled mode combinations supported for the Summit controller.

The SIB variants of the Summit drivers are designed to work with both the flight/EM model SIB board (PCI Device ID = 0x0844) and the evaluation model SIB board (PCI Device ID 0x0001). The differences between the software drivers is minimal.

Table 5- Polled Mode UTMC Summit Libraries

Package	Board	Bus Function	Tags	Constituent
CTDB	SIB	Remote Terminal	rad750 mcp750	sumt_rt_poll_sib
	Alphi PMC-1553B	Remote Terminal	mv2304 mcp750	sumt_rt_poll_pmc1553

3.1.1 External Library Requirements

The 1553 polled mode drivers require some additional libraries for needed services. A set of functions is required for parsing the CCSDS packet headers.

Table 6 – Interrupt Mode Driver Required External Libraries

Package	Constituent
CCSDS	ccsds_pkt_boot
PBS	pbs_boot
SIB	sib_boot (SIB versions of the drivers only)

3.2 Driver Controller Interface

Each 1553 controller interrupt mode driver exports a set of functions for initializing and interacting with the 1553 bus interface. In all of the function interface definitions, the name “DEV” is used to represent a specific 1553 controller driver.

3.2.0 Remote Terminal

Each function exported by a remote terminal polled mode controller driver has the following naming convention:

<DEV>_rtPoll<Function>

DEV is the name of the 1553 controller device and Function is the detailed action to implement.

3.2.0.0 Device Control Functions

The 1553 remote terminal drivers provide a set of functions for initializing and controlling the bus interface.

3.2.0.0.1 DEV_rtPollGet

```
DEV_RT_Poll* DEV_rtPollGet(void)
```

This function returns a pointer to the global polled mode remote terminal descriptor. Only one instance of the driver is allowed. A pointer to the descriptor area is passed as a parameter to all of the other device interface functions.

3.2.0.0.2 DEV_rtPollInit

```
unsigned int DEV_rtPollInit(DEV_RT_Poll *rt, const DEV_RT_Bus_Config
    *busConfig, unsigned int busAddr)
```

This function initializes the 1553 remote terminal driver and device. The *rt* parameter is a pointer to the device descriptor. The *busAddr* parameter is the base bus address at which the 1553 controller board resides. The Summit PCI drivers will ignore this value and instead use the equivalent information found in the PCI configuration header. Most of the specific device and bus configuration information is contained in the structure pointed to by the *busConfig* parameter. See Section 2.3.0.0.3 for the details of this structure's members. Note that *DEV_rtPollStart()* must be called before the remote terminal device will start listening on the 1553 bus.

3.2.0.0.3 DEV_rtPollStart

```
unsigned int DEV_rtPollStart(const DEV_RT_Poll *rt)
```

This function starts the 1553 remote terminal device processing bus messages. Until this point, the remote terminal may not appear to respond to the bus controller.

3.2.0.0.4 DEV_rtPollQuery

```
unsigned int DEV_rtPollQuery(DEV_RT_Poll *rt, unsigned short *subAddr,
    void *buf)
```

This is the main entry point into the driver. The repeated call of this function is necessary for the driver to check up on any possible 1553 bus events. If the 1553 controller device has detected an error since the last call, the error code is returned. Otherwise, the remote terminal interface is checked for any new data messages. If there is no activity to report, the function returns with a special MSG code *CTDB_TIMEOUT* indicating no new events. If a 1553 data message or mode command message has occurred that the remote terminal has been configured to respond to, the function returns with the success code *CTDB_SUCCESS*. The *subAddr* parameter is a pointer to a location where the driver will store the subaddress ID values. The possible values are shown below:

Table 7 - Polled Mode Remote Terminal Subaddress Codes

Subaddress ID	Description
DEV_RT_SUBADDR_CMD_RX	A new telecommand message has been received.
DEV_RT_SUBADDR_TELEM_END	The last telemetry packet block has been transmitted.

The values listed above are OR'd together in the *subAddr* parameter location. If the ID value indicates that the remote terminal received data on a subaddress, then the function will copy the receive data message to the user buffer *buf*. If the ID value indicates that the remote terminal transmitted data on the telemetry subaddresses, then the function will simply return.

3.2.0.0.5 DEV_rtPollTelem

```
unsigned int DEV_rtPollTelem(DEV_RT_Poll *rt, void *buf, void *pkt, int
    size)
```

This function allows the user to update the data being output to the CCSDS telemetry packet subaddress. The user may call *DEV_rtPollQuery()* to determine when the data has actually been transmitted. The *buf* parameter should point to a memory buffer which is at least the size of the GLAST 1553 protocol telemetry block. This size is equal to 64 bytes multiplied by the number of 1553 telemetry messages in a packet block, which should equal the *telem_msg_count* member value passed to *DEV_rtPollInit()*. The driver will manage the flow control word, which is the first word in the telemetry block, and the zero-filled trailing portion of the telemetry block. The *pkt* parameter should point to one or more CCSDS telemetry packets chained together already in memory. The *size* parameter gives the size in bytes of the telemetry packet chain to be sent.

3.3 Driver Examples

These examples show how to setup and interact with the polled mode 1553 drivers.

3.3.0 Remote Terminal

This example shows how to setup a SUMT 1553 polled mode driver for the remote terminal bus function.

```
    #include "CTDB/SUMT_rt_poll.h"
    #include "CTDB/CTDB_config.h"
    #include "CTDB/CO1553_util.h"

    /* the remote terminal bus configuraion */

0   SUMT_RT_Bus_Config sumt_bus_config =
    {
1       CTDB_CONFIG_RT_ADDR,                    /* rt_addr */
2       CTDB_CONFIG_SUBADDR_CMD_RX,             /* cmd_rx_subaddr */
3       CTDB_CONFIG_SUBADDR_CMD_TX,             /* cmd_tx_subaddr */
4       CTBD_CONFIG_SUBADDR_TELEM_FIRST,        /* telem_subaddr */
5       CTDB_CONFIG_SUBADDR_TELEM_NUM,          /* telem_msg_count */
6       CTDB_CONFIG_SUBADDR_WRAP,               /* wrap_subaddr */
7       {
            0, 0,
            SUMT_RT_MODE_FLAG_VALID,            /* mode code 2 */
            0,
            SUMT_RT_MODE_FLAG_VALID,            /* mode code 4 */
            SUMT_RT_MODE_FLAG_VALID,            /* mode code 5 */
            0, 0,
            SUMT_RT_MODE_FLAG_VALID,            /* mode code 8 */
            0, 0, 0, 0, 0, 0, 0,
            0, 0, 0, 0, 0, 0, 0, 0,
            0, 0, 0, 0, 0, 0, 0, 0,

        }
    };

    void sumt_rt_go(void)
    {
      SUMT_rt *rt;
      int status, size;
      unsigned short subaddr;
      unsigned char data[CO1553_MSG_SIZE];


    /*----------------------- initialize the device ----------------------*/


    /* get the RT object */

    size = SUMT_rtPollSizeof();
    rt = (SUMT_RT*) malloc(size);
    if(rt == NULL)
    {
          printf("cannot alloc RT poll object\n");
          return;
    }

    /* initialize the RT */

8   status = SUMT_rtPollInit(rt, &sumt_bus_config, 0);
    if(_msg_success(status) == 0)
    {
```

```
            /* error handler */
            return;
      }

      /* start the remote terminal */

 9    status = SUMT_rtPollStart(rt);
      if(_msg_success(status) == 0)
      {
            /* error handler */
            return;
      }

      while(1)
      {
        do
        {
10          status = SUMT_rtPollQuery(rt, &subaddr, data);
         } while(_msg_match(status, CTDB_TIMEOUT) == 1);

        if(_msg_success(status) == 0)
        {
            /* error handler */
            return;
        }

        if(subaddr & SUMT_RT_SUBADDR_CMD_RX)
        {
11
            /* process a new command message */

        }

12      if(subAddr & SUMT_RT_SUBADDR_TELEM_END)
        {

            /* the last telemetry block has been sent */

            /* get the next one ready and send it */

            status = SUMT_rtPollTelem(rt, telemBlkPtr, hkpPkt, hkpPktSize);
            if(_msg_success(status) == 0)
            {
                   /* error handler */
                   return;
            }
        }
      }
```

Notes on Polled Mode RT Driver Example

0. This instance of the *SUMT_RT_Bus_Config* structure defines the 1553 bus interface of the remote terminal.

1. The terminal address of the LAT SIU remote terminal is set to ‘1’.

2. The telecommand receive subaddress value for the remote terminal is set to ‘27’.

3. The telecommand transmit subaddress value for the remote terminal is set to ‘29’. The polled mode remote terminal drivers ignore this value.

4. The base telemetry subaddress value for the remote terminal is set to ‘11’. The polled mode remote terminal drivers ignore this value.

5. There are ‘15’ 1553 telemetry subaddress data messages for a frame. The polled mode remote terminal drivers ignore this value.

6. The data wraparound subaddress for the remote terminal is ‘30’.

7. There are 4 valid 1553 mode commands recognized, numbers 2, 4, 5, and 8.

8. The remote terminal driver and device are initialized here. In this example take from an Alphi PMC board, the PCI base address of the board is set to 0xfd000000.

9. The call to *SUMT_rtPollStart()* connects the remote terminal to the 1553 bus.

10. The repeated call to *SUMT_rtPollQuery()* is the main 1553 event processing loop. The function is called again when the return status is *CTDB_TIMEOUT*, indicating no new 1553 bus event occurred. Other useful application work could also be done within this loop as long as the next call to *SUMT_rtPollQuery()* is soon enough to capture the next bus events.

11. The query determined that a new data message arrived on the telecommand receive subaddress.

12. The query determined that the last data has been transmitted on the telemetry subaddress. In this example, the next telemetry data block is prepared immediately and made available with the call to *SUMT_rtPollTelem()*.

4 Errors and Diagnostics

The SIU 1553 drivers are able to report a large set of error and information conditions. These reports are delivered in one of two ways. The definitions of the reports are done with the MSG system. For both interrupt and polled mode drivers, the call to any public driver function may return a MSG code. Usually the codes returned by the driver functions directly indicate that a parameter value is out of acceptable range. For interrupt mode drivers, MSG codes may also be reported indirectly from the driver itself. This mode of error reporting occurs when the interface hardware detects a bad condition on the bus interface and cannot wait for a user to directly call a function.

For both methods of error information delivery, there are two classes of error codes. The first class of error codes is common to all drivers. The second class of error codes is specific to a particular hardware controller device. This second class is necessary because the error reporting capabilities differ slightly among the hardware controller families.

The SIU 1553 remote terminal drivers also maintain a list of diagnostics statistics which report the number of errors, packets, and bits occurring on each subaddress.

4.0 Generic Driver Error Codes

The generic error codes values are common to all device controller drivers. Table 8 lists the MSG values that are common to all of the hardware controller drivers and software simulators.

Table 8 - Generic Remote Terminal MSG Codes

MSG Facility	MSG Code	Description	Parameters
CTDB	CTDB_SUCCESS	Success.	None
	CTDB_TIMEOUT	Successful except that the specified timeout period expired.	None
	CTDB_EDRVPARM	A parameter passed to one of the public driver functions was out of range	The name of the bad parameter.
	CTDB_EDRVBRD	The local bus interface to the hardware controller I/O board could not be initialized or the I/O board could not be found at the specified address or interrupt setting.	The name and action of the system call.
	CTDB_EDRVOBJ	A queue, list, or buffer object could not be created or properly initialized.	The name of the bad object.
	CTDB_EDRVTASK	The VxWorks event service task could not be created or properly started.	The name of the failed task control function.
	CTDB_EDRVSYS	A system call inside the driver failed.	The name and action of the system call.
	CTDB_EDRVCNFG	The 1553 bus configuration information for the driver had out of range or inconsistent values.	The name of the bad configuration parameter.
	CTDB_EDRVMEMA	The driver could not allocate a piece of required memory.	The name of the resource being allocated.
	CTDB_EDRVTIMO	The driver detected a timeout based on the bus configuration values entered at startup.	None.
	CTDB_EPKTFORM	An improperly formatted CCSDS packet header was prefixed to a packet.	The hex value of the CCSDS packet header ID word.
	CTDB_EPKTSIZE	A CCSDS packet had a total size that exceeded the limit for a particular subaddress.	The size in bytes of the packet.

	CTDB_EPKTDROP	An incoming CCSDS packet was lost because reader applications are not servicing an input queue fast enough.	The hex value of the CCSDS packet header ID word.

4.1 Summit Controller Error Codes

The Summit controllers have the ability to monitor for 1553 bus error conditions and internal device malfunctions.

4.1.0 Remote Terminal

Table 9 lists the error codes specific to the Summit remote terminal drivers.

Table 9 - Summit Specific Remote Terminal MSG Codes

MSG Facility	MSG Code	Description	Parameters
	CTDB_EDEVINIT	The Summit controller does not respond to the initialization process.	The value of the Summit status register.
	CTDB_EDEVDMAF	The Summit controller reported a DMA failure when attempting to access shared memory.	The value of the Summit status register.
	CTDB_EDEVTAPF	The Summit controller reported a RT terminal address parity failure.	The value of the Summit status register.
	CTDB_EDEVBITF	The Summit controller failed the built in test procedure.	The value of the Summit BIT results register.
	CTDB_EDEVSYSF	The Summit controller is asserting the system fail signal.	The value of the Summit status register.
	CTDB_EDEVBUS	The Summit controller detected a 1553 bi-phase, parity, or link level protocol error.	The command word from the Summit last command register.

	CTDB_EDEVCMD	The Summit controller detected a 1553 bus message directed to the SIU remote terminal that was sent on an unrecognized subaddress.	The command word from the Summit last command register.
	CTDB_EDEVWRPF	The Summit data wraparound comparison failed.	The value of the Summit status register.
	CTDB_IDEVWRAP	The Summit controller detected a 1553 transaction on the data wraparound subaddress.	A string “Tx” or “Rx” indicating the wrap command direction.

4.2 Diagnostics

4.2.0 Remote Terminal

The statistics counters for the remote terminal drivers are obtained with a call to *DEV_rtDiagGet()*. The diagnostics information is returned in a *DEV_RT_Diag* structure shown below.

```
typedef struct _DEV_RT_Diag
{
 unsigned int error_count;
 unsigned int intr_count;
 unsigned int cmd_rx_pkt_count;
 unsigned int cmd_rx_byte_count;
 unsigned int cmd_tx_pkt_count;
 unsigned int cmd_tx_byte_count;
 unsigned int hkp_pkt_count;
 unsigned int hkp_byte_count;
 unsigned int telem_pkt_count;
 unsigned int telem_byte_count;
} DEV_RT_Diag;
```

The *error_count* member is incremented once for each error caught by the RT driver and delivered with an error code to the user installed error handler. The *intr_count* member is incremented once for each interrupt serviced by the driver. The *xxx_pkt_count* members report the number of CCSDS packets successfully delivered to each subaddress interface. The *xxx_byte_count* members report the number of packet bytes successfully delivered to each subaddress interface. The HKP telemetry packets are accounted for separately from all other

telemetry packet types. All of the counters are initialized to ‘0’ on the first call to *DEV_rtStart()*, and are reset to ‘0’ on every call to *DEV_rtStart()* and *DEV_rtDiagClear()*.

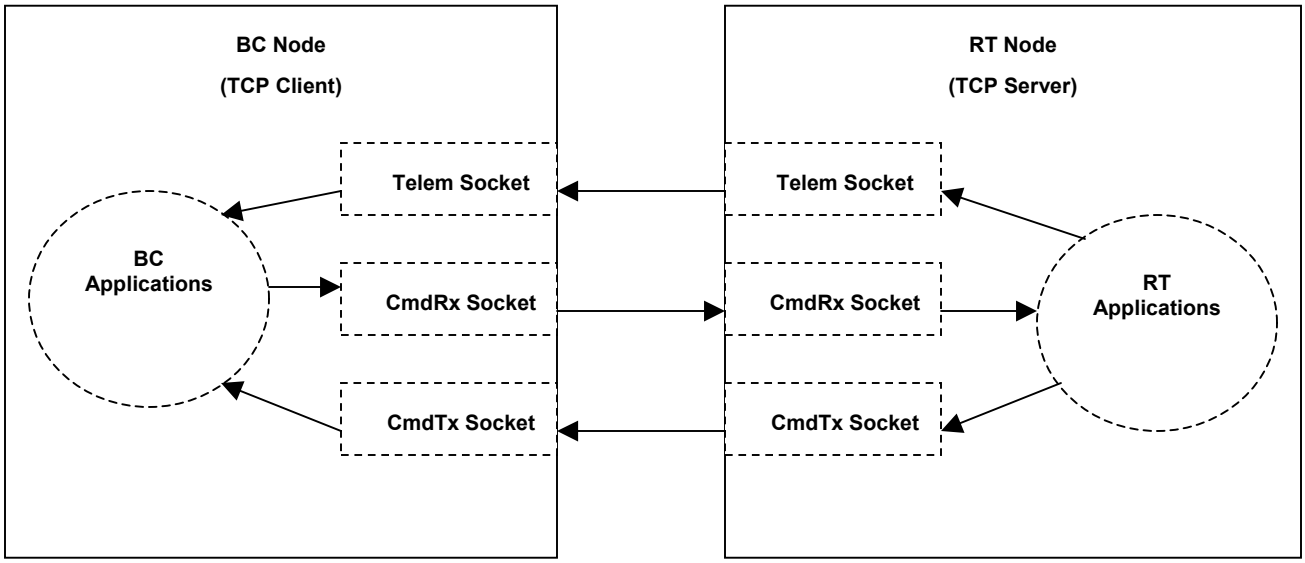
5 1553 Simulators

Because real 1553 equipment is somewhat cumbersome and expensive, the LAT benefits from a 1553 simulator environment which transfers CCSDS packets between machines using the same basic software interfaces as the real drivers. Essentially, the 1553 bus medium is replaced by standard Ethernet, allowing almost any two machines to communicate using the GLAST 1553 data protocol. This being said, the LAT 1553 simulators do not attempt to simulate the details of the 1553 hardware bus protocol, but rather the data flow protocol as specified for the GLAST mission.

5.0 Simulator Architecture

The diagram below shows the basic architecture of the 1553 simulator components.

Figure 15 - Simulator Architecture



Each simulator system consists of two Ethernet nodes, both of which must have the standard TCP/IP stack running and the standard socket interface library available. One node acts as the RT, and one node acts as the BC. Each subaddress is assigned to a different TCP port number, which is configured at the simulator startup. The RT node sockets are initialized as the TCP server side of the communications path, and the BC node sockets are initialized as the TCP client side of the communications path. When the RT simulator starts, it will listen for three connection requests from the BC simulator. The BC simulator will connect to the RT sockets when it starts.

The following limitations of the simulation environment are important to understand:

1. The simulator does not monitor data bandwidth across the Ethernet interface. The simulator should be able to transfer data at a much higher rate than the actual 1553 bus. No actual bus schedule or bus frames are run.
2. None of the link level features of the 1553 bus are simulated. None of the error reporting features of the actual 1553 bus are available.
3. The simulator does not support the data wraparound subaddress or any of the mode codes.
4. The simulator does not keep diagnostics statistics.
5. The simulator only uses a single output telemetry queue on the RT side.

Otherwise, the simulator interface attempts to mimic the 1553 driver interface as well as possible.

5.1 Simulator Libraries

5.1.0 Simulator Interface Libraries

The simulator interface libraries provide a set of CCSDS packet queues for transferring data to another simulator node. The simulator libraries also contain all of the code to create and maintain the TCP sockets. The table below lists the supported simulator libraries.

Table 10 - 1553 Simulator Libraries

Bus Function	Package	Constituent
Remote Terminal	CTDB	co1553_rt_sim
Bus Controller	CTDB	co1553_bc_sim

5.1.1 External Library Requirements

The 1553 simulators require some additional libraries for needed services. A set of functions is required for parsing the CCSDS packet headers. In addition, a library for creating and manipulating queues and buffers is necessary. The MSG libraries are required for condition signaling.

Table 11 - Simulator External Library Requirements

Package	Constituent
CCSDS	ccsds_pkt
PBS	pbs
MSG	msg_mt

Additional libraries may be needed on the target machine to satisfy the dependency requirements of the libraries listed above. The target machine must also have a TCP socket interface library available.

5.2 Simulator Interface

The 1553 simulator exports a set of functions for initializing and interacting with the 1553 simulation environment. The interface varies slightly depending on the bus function the node is supporting.

5.2.0 Remote Terminal

Each function exported by the remote terminal simulator has the following naming convention:

CO1553_rtSim<Function>

<Function> is the detailed action to implement.

5.2.0.0 Simulator Control Functions

The remote terminal simulator provides a set of functions for initializing and controlling the simulated bus interface.

5.2.0.0.1 CO1553_rtSimSizeof

```
unsigned int CO1553_rtSimSizeof(void)
```

This function returns the size of the simulator remote terminal descriptor. The user is responsible for allocating a block of memory of at least this size to hold the simulator descriptor. A pointer to the descriptor area is passed as a parameter to all of the other simulator interface functions.

5.2.0.0.2 CO1553_rtSimInit

```
unsigned int CO1553_rtSimInit(CO1553_RT_Sim *rt, const
    CO1553_RT_Sim_Config *simConfig)
```

This function initializes the remote terminal simulator node. The *rt* parameter is a pointer to an uninitialized simulator descriptor. The *simConfig* parameter points to a structure containing the configuration information for the simulation remote terminal node.

```
typedef struct _CO1553_RT_Sim_Config
{
 unsigned short cmd_rx_port;
 unsigned int cmd_rx_pkt_size;
 unsigned int cmd_rx_user_size;
 unsigned int cmd_rx_queue_size;
 unsigned short cmd_tx_port;
 unsigned int cmd_tx_pkt_size;
 unsigned short telem_port;
 unsigned int telem_pkt_size;
} CO1553_RT_Sim_Config;
```

The *xxx_port* members provide the TCP port numbers on which the different types of subaddress data are transferred. The *xxx_pkt_size* members limit the maximum size, including the header, of packets transferred on the sockets. The *cmd_rx_queue_size* member provides the number of CmdRx packet buffers to allocate for incoming packets.

5.2.0.0.3 CO1553_rtSimExit

```
unsigned int CO1553_rtSimExit(CO1553_RT_Sim *rt)
```

This function stops the remote terminal simulator from running and releases all resources acquired during the call to *CO1553_rtSimInit()*.

5.2.0.0.4 CO1553_rtSimStart

```
unsigned int CO1553_rtSimStart(CO1553_RT_Sim *rt)
```

This function starts the remote terminal simulator TCP server listening for connections from the bus controller simulator on all of the available sockets.

5.2.0.0.5 CO1553_rtSimStop

```
unsigned int CO1553_rtSimStop(CO1553_RT_Sim *rt)
```

This function causes the remote terminal simulator TCP server to close all connections from the bus controller simulator on all of the available sockets.

5.2.0.0.6 CO1553_rtSimHkpSet

```
unsigned int CO1553_rtSimHkpSet(CO1553_RT_Sim *rt, void *handler, void
    *arg)
```

This function is provided for compatibility with the 1553 controller drivers. The HKP callback function is never actually invoked by the simulator. The *handler* and *arg* values are simply stored for retrieval by the function *CO1553_rtSimHkpGet()*.

5.2.0.0.7 CO1553_rtSimHkpGet

```
unsigned int CO1553_rtSimHkpGet(CO1553_RT_Sim *rt, void **handler, void
    **arg)
```

This function is provided for compatibility with the 1553 controller drivers. The HKP callback function is never actually invoked by the simulator. The *handler* and *arg* values are simply stored from the last call to function *CO1553_rtSimHkpSet()*.

5.2.0.1 Packet Queue Functions

The 1553 remote terminal simulator provides a set of functions for sending and receiving CCSDS packets in the 1553 simulation environment.

5.2.0.1.1 CO1553_rtSimPktTelemSend

```
unsigned int CO1553_rtSimPktTelemSend(CO1553_RT_Sim *rt, void *pkt)
```

This function places a CCSDS telemetry packet from the user on the remote terminal telemetry output queue. The *pkt* parameter should point to the packet buffer with an already completed CCSDS packet header inserted at the top. This function will always block until bandwidth is available to send the packet.

5.2.0.1.2 CO1553_rtSimPktCmdTxSend

```
unsigned int CO1553_rtSimPktCmdTxSend(CO1553_RT_Sim *rt, void *pkt)
```

This function places a CCSDS LAT telecommand output packet from the user on the remote terminal telecommand output queue The *pkt* parameter should point to the packet buffer with an already completed CCSDS packet header inserted at the top. This function will always block until bandwidth is available to send the packet.

5.2.0.1.3 CO1553_rtSimPktCmdRxRecv

```
unsigned int CO1553_rtSimPktCmdRxRecv(CO1553_RT_Sim *rt, void **pkt)
```

This function waits for a new incoming telecommand packet to become available from the remote terminal. The *pkt* parameter will contain a pointer to the packet buffer when the function returns successfully. The packet contents are stored in a buffer allocated from a driver internal pool. The user may borrow this packet memory, but should be careful since a limited number of packet buffers are available to hold the incoming command packets. The pointer returned in the pkt parameter points to the beginning of the CCSDS telecommand packet header. A number of bytes equal to the *CO1553_RT_Sim_Config.cmd_rx_user_size* initialization parameter is reserved just before this address. The user may use this memory however he pleased. The user

should call *CO1553_rtSimPktCmdRxFree()* when the packet buffer is no longer needed. This function will always block until a packet is actually available.

5.2.0.1.4 CO1553_rtSimPktCmdRxFree

```
unsigned int CO1553_rtSimPktCmdRxFree(CO1553_RT_Sim *rt, void *pkt)
```

This function returns a remote terminal input telecommand packet buffer to the driver packet buffer pool. It should be called when the user no longer needs the packet buffer provided by the call to *DEV_rtSimPktCmdRxRecv()*.

5.2.1 Bus Controller

Each function exported by the bus controller simulator has the following naming convention:

CO1553_bcSim<Function>

Function is the detailed action to implement.

5.2.1.0 Simulator Control Functions

The bus controller simulator provides a set of functions for initializing and controlling the simulated bus interface.

5.2.1.0.1 CO1553_bcSimSizeof

```
unsigned int CO1553_bcSimSizeof(void)
```

This function returns the size of the simulator bus controller descriptor. The user is responsible for allocating a block of memory of at least this size to hold the simulator descriptor. A pointer to the descriptor area is passed as a parameter to all of the other simulator interface functions.

5.2.1.0.2 CO1553_bcSimInit

```
unsigned int CO1553_bcSimInit(CO1553_BC_Sim *bc, const
    CO1553_BC_Sim_Config *simConfig, const char *rtAddr)
```

This function initializes the bus controller simulator node. The *bc* parameter is a pointer to an uninitialized simulator descriptor. The *simConfig* parameter points to a structure containing the configuration information for the simulation bus controller node.

```
typedef struct _CO1553_BC_Sim_Config
{
 unsigned short cmd_rx_port;
 unsigned int cmd_rx_pkt_size;
 unsigned short cmd_tx_port;
 unsigned int cmd_tx_pkt_size;
 unsigned int cmx_tx_queue_size;
 unsigned short telem_port;
 unsigned int telem_pkt_size;
 unsigned int telem_queue_size;
} CO1553_BC_Sim_Config;
```

The *xxx_port* members provide the TCP port numbers on which the different types of subaddress data are transferred. The *xxx_pkt_size* members limit the maximum size, including the header, of packets transferred on the sockets. The *xxx_queue_size* members give the number of packet receive buffers to provide for incoming packets.

The *rtAddr* parameter gives the IP address of the remote terminal simulator node that the bus controller simulator will connect to.

5.2.1.0.3 CO1553_bcSimExit

```
unsigned int CO1553_bcSimExit(CO1553_BC_Sim *bc)
```

This function stops the bus controller simulator from running and releases all resources acquired during the call to *CO1553_bcSimInit()*.

5.2.1.0.4 CO1553_bcSimStart

```
unsigned int CO1553_bcSimStart(CO1553_BC_Sim *bc)
```

This function initiates the bus controller simulator TCP client connections to the remote terminal server node. The connections for all of the supported sockets are attempted.

5.2.1.0.5 CO1553_bcSimStop

```
unsigned int CO1553_bcSimStop(CO1553_BC_Sim *bc)
```

This function causes the bus controller simulator TCP client to close all connections from to the remote terminal simulator on all of the available sockets.

5.2.1.1 Packet Queue Functions

The 1553 bus controller simulator provides a set of functions for sending and receiving CCSDS packets in the 1553 simulation environment.

5.2.1.1.1 CO1553_bcSimPktTelemRecv

```
unsigned int CO1553_bcSimPktTelemRecv(DEV_BC_Sim *bc, void *pkt)
```

This function waits for a new telemetry packet to become available from the bus controller. The *pkt* parameter is the user buffer where the packet contents are placed. This function will always block until a packet is actually available.

5.2.1.1.2 CO1553_bcSimPktCmdTxRecv

```
unsigned int CO1553_bcSimPktCmdTxRecv(DEV_BC_Sim *bc, void *pkt)
```

This function waits for a new LAT telecommand output packet to become available from the bus controller simulator. The *pkt* parameter is the user buffer where the packet contents are placed. This function will always block until a packet is actually available.

5.2.1.1.3 CO1553_bcSimPktCmdRxSend

```
unsigned int CO1553_bcSimPktCmdRxSend(DEV_BC_Sim *bc, void *pkt)
```

This function places a CCSDS command packet from the user on the bus controller simulator command output queue. It will arrive at the remote terminal simulator on the telecommand receive socket. The *pkt* parameter should point to the packet buffer with an already completed CCSDS packet header inserted at the top. This function will always block until bandwidth is available to send the packet.