C²I² Systems

CCII Systems (Pty) Ltd Registration No. 1990/005058/07

Communications
Computer Intelligence
Integration

User Manual

for the

MIL-STD-1553B Adapter

VxWorks Software Driver

C²I² Systems Document No.	CCII/1553/6-MAN/001
Document Issue	1.1
Issue Date	2009-08-31
Print Date	2009-08-31
File Name	W:\1553\TECH\MAN\C53MAN01.WPD
Distribution List No.	DN 0649

Document prepared by C²I² Systems, Cape Town

Signature Sheet

Name	Signature	Date
Completed by pp X. Kruger	Project Engineer Board Level Products C²I² Systems	2009-08-31
Accepted by L. DE Lima	Project Manager Board Level Products C²I² Systems	2009-08-31
Accepted by X. Kruger	Quality Assurance C²I² Systems	2009-08-31

Signature Sheet

Name	Signature	Date
Completed by		
	Project Engineer Board Level Products C²I² Systems	
Accepted by		
	Project Manager Board Level Products C²I² Systems	
Accepted by		
	Quality Assurance C²I² Systems	

Amendment History

Issue	Description	Date	ECP No.
0.1	First draft.	2006-05-18	-
0.2	Updated after review.	2006-05-19	-
0.3	Added Bus Controller API.	2006-06-12	-
1.0	Added Bus Monitor API.	2006-06-29	-
1.1	Improve document naming consistency.	2009-08-31	CCII/1553/6-ECP/003

Contents

List of Figures

Abbreviations and Acronyms

API	Application Program Interface
BC	Bus Controller
BIT	Built-in Test
BM	Bus Monitor
Mbit/s	Megabits per second
N/A	Not Applicable
OS	Operating System
PC	Personal Computer
PCI	Peripheral Component Interconnect
RT	Remote Terminal
RX	Receive
TX	Transmit

1. Scope

1.1 Identification

This document is the user manual for the VxWorks Software Driver of the C²I² Systems MIL-STD-1553B Adapter.

1.2 System Overview

The MIL-STD-1553B Adapter provides the functionality of either a Bus Controller (BC), Remote Terminal (RT) or a Bus Monitor (BM) on a MIL-STD-1553B data bus.

A MIL-STD-1553B data bus is a half-duplex, dual redundant bus with a single active BC and up to 31 RTs. A BM does not partake in the message exchange, but records all messages on the bus for logging purposes. Message exchange is based on the command/response protocol between BC and RT.

Messages consist of up to 32 16-bit words and are sent at a rate of 1 Mbit/s. Messages are originated by the BC when it sends a command word followed by optional data words. The RT responds with a status word and data words when requested. A BC can also initiate a RT to RT message transfer.

The VxWorks Software Driver is a low level, device-dependant interface for transferring data over a MIL-STD-1553B Adapter. The driver binaries are provided with explicit installation instructions.

1.3 Document Overview

This document gives an overview of the MIL-STD-1553B VxWorks Software Driver installation procedure and its Application Program Interface (API).

2. Applicable and Reference Documents

2.1 Applicable Documents

None.

2.2 Reference Documents

2.2.1 MIL-STD-1553B, *Aircraft Internal Time Division Command/Response Multiplex Data Bus*, dated 21 September 1976.

2.2.2 MIL-STD-1553B Notice 1, 12 February 1980.

2.2.3 MIL-STD-1553B Notice 2, 08 September 1986.

2.2.4 MIL-STD-1553B Notice 3, 31 January 1993.

2.2.5 MIL-STD-1553B Notice 4, 15 January 1996.

3. Driver Distribution

The driver software distribution consists of (at least) the following files :

cc1553Lib<*arch*>V<*version*><*long*>.a	Host-architecture specific, driver object file : cc - C²I² Systems. 1553Lib - MIL-STD-1553B VxWorks Software Driver. <*arch*> - Host for which the binary is built : • X86 • Dy4PPC (for Dy4 PPC SVME / DMV181). <*version*> - Software version is a 3 digit integer : • 1st digit is the version number • 2nd digit is the revision number • 3rd digit is the beta number. <*long*> - driver compiled with -mlongcall flag (only for Dy4 PPC host).
cc1553Readme.txt	General information and installation notes.
cc1553Release.txt	Release notes and revision history : Please check this file for information on the latest updates.
cc1553HfilesV<*version*>.zip	Zip file which contains all header files that define the Application Program Interface (API) to the VxWorks Software Driver.

4. Installation Procedure

This paragraph describes the installation procedure for the VxWorks Software Driver (the examples given are for a Dy4 SVME/DMV181 PowerPC host).

4.1 To Build the VxWorks Software Driver into the VxWorks Kernel

Assume the BSP directory is given as : BSP_DIR = /tornado/target/config/dy4181

- Copy cc1553Lib<*arch*>V<*version*>.a to your $(BSP_DIR)/lib directory as cc1553.a.
-
- In the Builds view of the Project Workspace, change the build specification properties to include the cc1553.a library file with the Macros LIBS option.
-
- Rebuild all VxWorks images.

4.2 To Load the VxWorks Software Driver Software Separately

Note : This step is not required if the VxWorks Software Driver was built into the BSP.

If the VxWorks Software Driver is not built into the BSP, a user can load it separately :

- Copy cc1553Lib<*arch*>V<*version*>.a to your present working directory as cc1553.a.
- From the VxWorks shell type : "`ld < cc1553.a`".

5. Using the MIL-STD-1553B VxWorks Software Driver

5.1 Overview

The following flow chart shows the main functions of the VxWorks Software Driver :

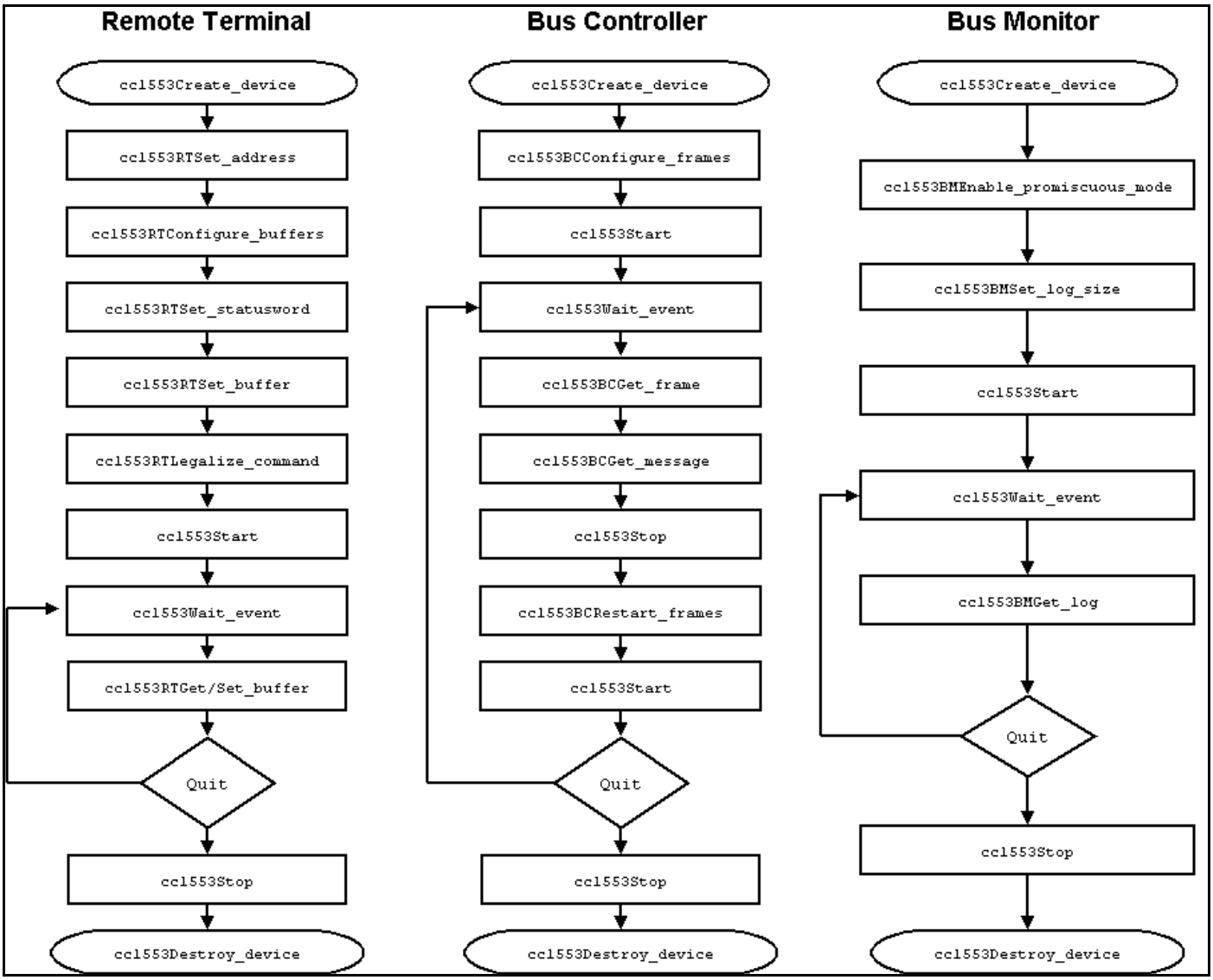

Figure 1 : Function Overview

5.2 Creating the Device

The VxWorks Software Driver supports multiple MIL-STD-1553B adapters on a single host. To establish a connection and construct all the device specific structures, a user must create each of the devices separately, using the device ID to identify it.

The device ID starts at 0 and increments by 1 for each of the devices. Device 0 refers to the device in the lowest slot. The VxWorks Software Driver can not be used until the user has created the device.

Example : For device 0 :

```
/* Create one device as a Remote Terminal. */
cc1553Create_device(0, MODE_RT);
```

The device ID is used in all calls to the VxWorks Software Driver to identify the correct device.

5.3 Destroying the Device

When the device is no longer required it should be destroyed to free system resources.

Example : Device 0 is no longer required :

```
/* Destroy device to free resources. */
cc1553Destroy_device(0);
```

5.4 Obtaining the Current Version Number

The following function prints out the current version number of the driver software :

```
/* Print current version number. */
cc1553Print_version();
```

5.5 Built-in Tests (BITs)

The following function displays each device's status : e.g. how many messages have been accepted / lost and how many errors were detected.

Example : Displaying the BIT results for device 0 :

```
cc1553Print_BIT(0);
```

Note : The tests executed during BIT writes patterns into the device's internal memory, overwriting any buffer allocations. This test is then destructive and `cc1553Destroy_device` should be called afterwards. A non-destructive subset of BIT is available by using `cc1553Get_BIT`.

5.6 Return Adapter Type

The following function returns the adapter type. The return value will be MODE_BC, MODE_BM or MODE_RT.

Example : Get adapter type of device 0 :

```
cc1553DeviceMode adapter_type;

cc1553Adapter_type(0, &adapter_type);
```

5.7 Message Processing

Once buffers or frames have been allocated, the device must be instructed to start processing messages on the bus.

Example : Starting device 0 :

```
cc1553Start(0);
```

In RT mode the device will now start to generate events when receiving or sending messages and also respond to bus commands from the Bus Controller. Use the function `cc1553Wait_event` to be notified of any bus messages and if a transmit is required, load the secondary buffer for use during the next transmit. The alternating buffers are encapsulated by the VxWorks Software Driver, and the user simply need to call `cc1553RTSet_buffer` to load the next message.

In BC mode the device will start sending the frames and store the RT responses in device memory. To access RT status or data use the `cc1553BCGet_frame` and `cc1553BCGet_message` functions.

In BM mode the device will start logging messages in device memory. Use `cc1553Wait_event` to wait for the log size to be reached or poll the logs with `cc1553BMGet_log`.

5.8 Allocating Buffers as Remote Terminal

To send and receive messages, the user must first allocate buffers for a specific sub address or mode code. For each sub address or mode code two buffers are allocated in device internal memory. This allows the user to access a message buffer even while the next message is being received by the device for the same sub address or mode code.

The user may be notified of any updates to the buffer by using the `cc1553Wait_event` function call, or the user may poll the buffer to check for any updates with `cc1553Was_buffer_accessed`.

Any messages received for a sub address or mode code that has not been allocated, will be lost.

Example : Allocating receive sub addresses 2 :

```
/* Allow messages for sub address 2, and enable notification. */
cc1553RTConfigure_buffers(0, RX, SUBADDR, 2, 10, cc1553TRUE);
```

5.9 Building Frames as Bus Controller

In BC mode, the adapter sends a sequence of command words with optional data words to the RTs on the bus. Each such command is called a frame and the data words are called a message. A message may be utilised to send the same data to many RTs, saving resources in device internal memory.

Frames and messages are referenced with zero-based indexes. Frame and message arrays are created by the user, and cross-referenced with the indexes into the supplied arrays.

The BC can deviate the sequence of frames based on opcodes and conditions. The conditions represent the flags in the RT status word. If any of the conditions are met, an alternate frame route may be taken depending on the opcode. These alternate routes are called minor frames.

Opcodes can cause the frame sequence to branch or skip directly to another frame in the supplied list. This method of operation calls for very little user intervention during RT error states if care is taken with the design of the frames.

5.10 Logging Messages as Bus Monitor

In BM mode, the device is able to capture all messages on the bus, regardless of which terminals are involved (promiscuous mode) or the device can capture only message of specific terminals (filtering).

Messages are logged in device memory in a ring buffer, the size of which is set with `cc1553BMSet_log_size`. `cc1553Wait_event` will block until this number of messages have been logged. However the device stores one more log than the configured log size after raising an event, and this additional log index must be checked with `cc1553BMGet_log` otherwise the message will be lost. If this additional ring buffer entry is empty, `cc1553BMGet_log` will return an error.

6. Application Program Interface (API)

6.1 Common Interface

The zip file cc1553HfilesV<*version*>.zip contains the following header files, which should always be included :

- 1553Defs.h
- 1553HostDriver.h
- 1553MilStd.h

The header files may also be included in a C++ file.

All functions return a status code, if the code is not "`cc1553_OK`" an error condition occurred. A text description for a specific code is available by using `cc1553Print_status`.

All data types are declared in "1553Defs.h". All function prototypes are declared in "1553HostDriver.h".

6.1.1 Create Device

Function : **cc1553Create_device**

Purpose : Create and initialise the MIL-STD-1553B device specific structures.

Arguments :

<dev_id>	Device identifier on the Peripheral Component Interconnect (PCI) bus. The device in the lowest PCI slot : <dev_id> = 0, next device : <dev_id> = 1, etc.
<mode>	The functionality required from the device, i.e. MODE_RT, MODE_BC, MODE_BM.

Returns :

cc1553_OK	On success.
cc1553_INVALID_PARAM	If <dev_id> is incorrect.
cc1553_DEV_ALREADY_INITIALIZED	If the function is called more than once for the same device.
cc1553_PCI_CFG_FAIL	If the PCI configuration failed.
cc1553_DEVICE_NOT_FOUND	If the device number <dev_id> was not found on the PCI bus.
cc1553_MODE_NOT_SUPPORTED	If the device does not support this <mode>.
cc1553_EVENT_QUEUE_FAIL	If the Operating System (OS) could not create a message queue for the driver.
cc1553_ACCESS_SEMAPHORE_FAIL	If the OS could not create a semaphore for the driver.

Note : This function has to be called (once per device) before any other function call to the specified device will be valid. It can also only be called once per device, unless `cc1553Destroy_device` is called.

6.1.2 Destroy Device

Function : **cc1553Destroy_device**

Purpose : Reset the MIL-STD-1553B device specific structures and free OS resources.

Arguments :

<dev_id>	Device identifier on the PCI bus. As used in cc1553Create_device.

Returns :

cc1553_OK	On success.
cc1553_INVALID_PARAM	If <dev_id> is incorrect.
cc1553_DEV_NOT_INITIALIZED	If no corresponding successful call was made to cc1553Create_device.
cc1553_EVENT_QUEUE_FAIL	If the OS could not delete the driver message queue.
cc1553_ACCESS_SEMAPHORE_FAIL	If the OS could not delete the driver semaphore.

Note : After this function is called, no other function call to the specified device will be valid, except for `cc1553Create_device`.

6.1.3 Print Out Current Version Information

Function : **cc1553Print_version**

Purpose : Prints the VxWorks Software Driver version.

Arguments :

None

Returns :

cc1553_OK	On success.

6.1.4 Built-in Test Results

Function : **cc1553Get_BIT**

Purpose : Retrieve the BIT result.

Arguments :

<dev_id>	Device identifier on the PCI bus. As used in cc1553Create_device.
<p_result>	Pointer to a user supplied buffer.

Returns :

cc1553_OK	On success.
cc1553_INVALID_PARAM	If <dev_id> is incorrect or <p_result> is invalid.
cc1553_DEV_NOT_INITIALIZED	If no corresponding successful call was made to cc1553Create_device.
<p_result>	The BIT results.

BIT Results :

dma_fail	The device has encountered a problem during access of its internal memory.
wrap_fail	The device's internal loopback checking of transmissions has indicated an error.
parity_fail	The RT address was set incorrectly.
channel_a_fail	A time-out was detected on bus A.
channel_b_fail	A time-out was detected on bus B.
nr_events_rx	The number of events received by the interrupt handler.
nr_events_lost	The number of events the interrupt handler had to drop due to load conditions. This normally indicates that cc1553Wait_event was not serviced often enough.

6.1.5 Print Out BIT Results

Function : **cc1553Print_BIT**

Purpose : Print the BIT results.

Arguments :

<dev_id>	Device identifier on the PCI bus. As used in cc1553Create_device.

Returns :

cc1553_OK	On success.
cc1553_INVALID_PARAM	If <dev_id> is incorrect.
cc1553_DEV_NOT_INITIALIZED	If no corresponding successful call was made to cc1553Create_device.
cc1553_NOT_ALLOWED_WHEN_STARTED	If the function is called after a call to cc1553Start.

Note : This function is destructive as it writes patterns into the device memory, always call `cc1553Destroy_device` after printing the BIT results and reinitialise the device with `cc1553Create_device`.

6.1.6 Return Adapter Type

Function : **cc1553Adapter_type**

Purpose : Retrieve the adapter type: either Bus Controller, Remote Terminal or Bus Monitor.

Arguments :

<dev_id>	Device identifier on the PCI bus. As used in cc1553Create_device.
<p_adapter_type>	Pointer to a user supplied buffer.

Returns :

cc1553_OK	On success.
cc1553_INVALID_PARAM	If <dev_id> is incorrect or <p_adapter_type> is invalid.
cc1553_DEV_NOT_INITIALIZED	If no corresponding successful call was made to cc1553Create_device.
<p_adapter_type>	The adapter type.

6.1.7 Starting the Device

Function : **cc1553Start**

Purpose : Instruct the device to start processing messages.

Arguments :

<dev_id>	Device identifier on the PCI bus. As used in cc1553Create_device.

Returns :

cc1553_OK	On success.
cc1553_INVALID_PARAM	If <dev_id> is incorrect.
cc1553_DEV_NOT_INITIALIZED	If no corresponding successful call was made to cc1553Create_device.

6.1.8 Stopping the Device

Function : **cc1553Stop**

Purpose : Instruct the device to stop processing messages.

Arguments :

<dev_id>	Device identifier on the PCI bus. As used in cc1553Create_device.

Returns :

cc1553_OK	On success.
cc1553_INVALID_PARAM	If <dev_id> is incorrect.
cc1553_DEV_NOT_INITIALIZED	If no corresponding successful call was made to cc1553Create_device.

6.1.9 Receiving VxWorks Software Driver Notifications

Function : **cc1553Wait_event**

Purpose : Blocks until an event is generated on the MIL-STD-1553B bus. Use the event type to retrieve or set the applicable buffer contents.

Arguments :

<dev_id>	Device identifier on the PCI bus. As used in cc1553Create_device.
<timeout_ticks>	Number of OS ticks to wait before returning when no events are received.
<p_event>	Pointer to a user supplied buffer.

Returns :

cc1553_OK	On success.
cc1553_INVALID_PARAM	If <dev_id> is incorrect or <p_event> is invalid.
cc1553_DEV_NOT_INITIALIZED	If no corresponding successful call was made to cc1553Create_device.
<p_event>	The event detail.

Event Detail :

is_hw_error		Indicates if the hw field contains errors.
hw		Indicates any message errors. Always check this status before using the content of the message.
	message_error	Message error, can be one of the following :
	dma_fail	Memory access error.
	wrap_fail	Transmission error.
	parity_fail	Parity error.
event_type		
	EVENT_TIMEOUT	No event was received during the <timeout_ticks> period.
	EVENT_BC	Bus Controller event, use bc field. Other fields are invalid.
	EVENT_BM	Bus Monitor event, use bm field. Other fields are invalid.
	EVENT_RT	Remote Terminal event, use rt field. Other fields are invalid.
bc		Valid when EVENT_BC set.
	event_type	END_FRAME_LIST, ILLOGICAL_FRAME, INVALID_OPCODE, RETRY_FAILED, FRAME_ACCESSED.
	frame_index	The frame that caused the event, zero based index.

bm		Valid when EVENT_BM set.
	log_index	Zero based index indicating that the log size has been reached.
rt		Valid when EVENT_RT set.
	addr_or_mode	SUBADDR or MODECODE.
	dir	RX or TX.
	entry	Sub address or mode code value.

6.2 Remote Terminal Interface

All functions applicable in this mode are prefixed with "`cc1553RT`". These functions will return an error code if `cc1553Create_device` was not called with MODE_RT.

Some of these function calls are not allowed after a call to `cc1553Start`.

6.2.1 Setting the Terminal Address

Function : **cc1553RTSet_address**

Purpose : Configure the Remote Terminal address.

Arguments :

<dev_id>	Device identifier on the PCI bus. As used in cc1553Create_device.
<addr>	Refer to [2.2.1].

Returns :

cc1553_OK	On success.
cc1553_INVALID_PARAM	If <dev_id> is incorrect or <addr> is not in the valid range.
cc1553_DEV_NOT_INITIALIZED	If no corresponding successful call was made to cc1553Create_device.
cc1553_NOT_ALLOWED_WHEN_STARTED	If the function is called after a call to cc1553Start.
cc1553_NOT_ALLOWED_IN_MODE	If the function is called when cc1553Create_device was not called with MODE_RT.

6.2.2 Allocating Receive/Transmit Buffers

Function : **cc1553RTConfigure_buffers**

Purpose : Allocate buffers for receive or transmit messages.

Arguments :

<dev_id>	Device identifier on the PCI bus. As used in cc1553Create_device.
<dir>	RX or TX.
<addr_or_mode>	SUBADDR or MODECODE.
<entry>	Sub address or mode code value.
<message_size>	The size of the message in words.
<enable_interrupt>	Generate an interrupt when the specified message has been received or transmitted. Allows the use of cc1553Wait_event.

Returns :

cc1553_OK	On success.
cc1553_DEV_NOT_INITIALIZED	If no corresponding successful call was made to cc1553Create_device.
cc1553_INVALID_PARAM	If a function argument is incorrect or out of range.
cc1553_NOT_ALLOWED_IN_MODE	If the function is called when cc1553Create_device was not called with MODE_RT.
cc1553_OUT_OF_MEMORY	The device does not have enough internal memory to allocated the message buffers.
cc1553_SUBADDR_INUSE	The sub address buffer has already been setup.
cc1553_MODECODE_INUSE	The mode code buffer has already been setup.
cc1553_NOT_ALLOWED_WHEN_STARTED	If the function is called after a call to cc1553Start.

Note : If the function returns cc1553_OUT_OF_MEMORY, no more buffers can be allocated in the device's internal memory. No function is supplied to release previously configured buffers.

6.2.3 Sending Data

Function : **cc1553RTSet_buffer**

Purpose : Set the contents of a transmit buffer that will be sent when requested by the BC.

Arguments :

<dev_id>	Device identifier on the PCI bus. As used in cc1553Create_device.
<addr_or_mode>	SUBADDR or MODECODE.
<entry>	Sub address or modecode value.
<p_message>	Pointer to the start of the message data.
<message_size>	The size of the message in words.

Returns :

cc1553_OK	On success.
cc1553_DEV_NOT_INITIALIZED	If no corresponding successful call was made to cc1553Create_device.
cc1553_INVALID_PARAM	If a function argument is incorrect or out of range.
cc1553_NOT_ALLOWED_IN_MODE	If the function is called when cc1553Create_device was not called with MODE_RT.
cc1553_MESSAGE_TOO_LARGE	If the message size exceeds the limit set by [2.2.1].
cc1553_BUFFER_TOO_SMALL	If the message size exceeds the value supplied in cc15553RTConfigure_buffers.
cc1553_SUBADDR_DISABLED	If cc1553RTConfigure_buffers was not called for the sub address.
cc1553_MODECODE_DISABLED	If cc1553RTConfigure_buffers was not called for the modecode.
cc1553_PINGPONG_FAIL	If the device failed to access the requested buffer.

6.2.4 Receiving Data

Function : **cc1553RTGet_buffer**

Purpose : Retrieve the contents of a receive buffer.

Arguments :

<dev_id>	Device identifier on the PCI bus. As used in cc1553Create_device.
<addr_or_mode>	SUBADDR or MODECODE.
<entry>	Sub address or modecode value.
<p_message>	Pointer to a user supplied buffer.
<p_message_size>	The size of the user supplied buffer in words.
<p_status>	Pointer to a user supplied buffer.

Returns :

cc1553_OK	On success.
cc1553_DEV_NOT_INITIALIZED	If no corresponding successful call was made to cc1553Create_device.
cc1553_INVALID_PARAM	If a function argument is incorrect or out of range.
cc1553_NOT_ALLOWED_IN_MODE	If the function is called when cc1553Create_device was not called with MODE_RT.
cc1553_BUFFER_TOO_SMALL	If the message size exceeds the value supplied in cc1553RTConfigure_buffers.
cc1553_SUBADDR_DISABLED	If cc1553RTConfigure_buffers was not called for the sub address.
cc1553_MODECODE_DISABLED	If cc1553RTConfigure_buffers was not called for the modecode.
cc1553_PINGPONG_FAIL	If the device failed to access the requested buffer.
<p_message>	The received message.
<p_message_size>	The message size or modecode value.
<p_status>	Status information about the message.

Status :

is_from_busA / is_from_busB	Indicates on which bus the message was received.
is_RTtoRT	The message was received from another RT and not the BC.
is_msg_error	The message content must be considered invalid as an error occurred during transmission.
is_broadcast	The message was not sent to this RT alone, but was sent to all RTs.
is_illegal_cmd	The message has been illegalised by the user.

is_timeout	Detail of is_msg_error. The message was shorter than expected.
is_overrun	Detail of is_msg_error. The message was longer than expected.
is_parity_error	Detail of is_msg_error. A parity error occurred.
is_decoding_error	Detail of is_msg_error. A Manchester decoding error occurred.
timer_us	The value of the device's internal timer in microseconds, i.e. the timestamp of the message.

6.2.5 Status Word Flags

Functions : **cc1553RTGet_statusword** and **cc1553RTSet_statusword**

Purpose : Set or retrieve the status word for the RT.

Arguments :

<dev_id>	Device identifier on the PCI bus. As used in cc1553Create_device.
<p_status>	Pointer to a user supplied buffer.

Returns :

cc1553_OK	On success.
cc1553_DEV_NOT_INITIALIZED	If no corresponding successful call was made to cc1553Create_device.
cc1553_INVALID_PARAM	If a function argument is incorrect or out of range.
cc1553_NOT_ALLOWED_IN_MODE	If the function is called when cc1553Create_device was not called with MODE_RT.
<p_status>	MIL-STD-1553B Status Word.

Status :

standard	MIL_STD_1553_B.
clear_immediately	The other status indicators are cleared immediately after first transmission.
instrumentation	Refer to [2.2.1].
service_request	Refer to [2.2.1].
is_busy	Refer to [2.2.1].
subsytem_flag	Refer to [2.2.1].
terminal_flag	Refer to [2.2.1].

6.2.6 Internal Timer

Function : **cc1553RTGet_timetag**

Purpose : Retrieve the value of the device's internal timer. The resolution is 64 µs.

Arguments :

<dev_id>	Device identifier on the PCI bus. As used in cc1553Create_device.
<p_timer>	Pointer to a user supplied buffer.

Returns :

cc1553_OK	On success.
cc1553_DEV_NOT_INITIALIZED	If no corresponding successful call was made to cc1553Create_device.
cc1553_INVALID_PARAM	If a function argument is incorrect or out of range.
cc1553_NOT_ALLOWED_IN_MODE	If the function is called when cc1553Create_device was not called with MODE_RT.
<p_timer>	Internal timer value in microseconds. Resolution 64 µs.

6.2.7 Legalising Messages

Function : **cc1553RTLegalize_command**

Purpose : Legalise or Illegalise a command to a (broadcast) sub address or mode code. `cc1553RTGet_buffer` will indicate the message status as illegal when set.

Arguments :

<dev_id>	Device identifier on the PCI bus. As used in cc1553Create_device.
<dir>	RX or TX.
<addr_or_mode>	SUBADDR or MODECODE.
<entry>	Sub address or modecode value.
<is_broadcast>	cc1553TRUE / cc1553FALSE.
<is_legal>	cc1553TRUE / cc1553FALSE.

Returns:

cc1553_OK	On success.
cc1553_DEV_NOT_INITIALIZED	If no corresponding successful call was made to cc1553Create_device.
cc1553_INVALID_PARAM	If a function argument is incorrect or out of range.
cc1553_NOT_ALLOWED_IN_MODE	If the function is called when cc1553Create_device was not called with MODE_RT.
cc1553_NOT_ALLOWED_WHEN_STARTED	If the function is called after a call to cc1553Start.

6.2.8 Checking Buffer State

Function : **cc1553RTWas_buffer_accessed**

Purpose : Indicates whether the buffer was accessed due to a transmission on the bus. Intended use when `cc1553RTConfigure_buffers` is called with interrupt notification disabled. Allows polling for message state.

Arguments :

<dev_id>	Device identifier on the PCI bus. As used in cc1553Create_device.
<dir>	RX or TX.
<addr_or_mode>	SUBADDR or MODECODE.
<entry>	Sub address or modecode value.
<p_was_accessed>	Pointer to a user supplied buffer.

Returns :

cc1553_OK	On success.
cc1553_DEV_NOT_INITIALIZED	If no corresponding successful call was made to cc1553Create_device.
cc1553_INVALID_PARAM	If a function argument is incorrect or out of range.
cc1553_NOT_ALLOWED_IN_MODE	If the function is called when cc1553Create_device was not called with MODE_RT.
cc1553_SUBADDR_DISABLED	If cc1553RConfigure_buffers was not called for the sub address.
cc1553_MODECODE_DISABLED	If cc1553RConfigure_buffers was not called for the modecode.
<p_was_accessed>	cc1553TRUE / cc1553FALSE.

Note : The state of a buffer is reset after a call to this function.

6.2.9 Setting Bus State

Functions : cc1553RTEnable_busA and **cc1553RTEnable_busB**

Purpose : Enable or disable transmission on the specified bus.

Arguments :

<dev_id>	Device identifier on the PCI bus. As used in cc1553Create_device.
<enable>	cc1553TRUE / cc1553FALSE.

Returns :

cc1553_OK	On success.
cc1553_DEV_NOT_INITIALIZED	If no corresponding successful call was made to cc1553Create_device.
cc1553_INVALID_PARAM	If a function argument is incorrect or out of range.
cc1553_NOT_ALLOWED_IN_MODE	If the function is called when cc1553Create_device was not called with MODE_RT.
cc1553_NOT_ALLOWED_WHEN_STARTED	If the function is called after a call to cc1553Start.

6.3 Bus Controller Interface

All functions applicable in this mode are prefixed with "cc1553BC". These functions will return an error code if `cc1553Create_device` was not called with MODE_BC.

Some of these function calls are not allowed after a call to `cc1553Start`.

6.3.1 Alternating Bus Retries

Function : **cc1553BCEnable_pingpong**

Purpose : If an error occurs during transmission, the BC can be configured to retry sending the message. The retry can either occur on the same bus or alternate the bus on each iteration.

Arguments :

<dev_id>	Device identifier on the PCI bus. As used in cc1553Create_device.
<enable>	cc1553TRUE / cc1553FALSE.

Returns :

cc1553_OK	On success.
cc1553_DEV_NOT_INITIALIZED	If no corresponding successful call was made to cc1553Create_device.
cc1553_INVALID_PARAM	If a function argument is incorrect or out of range.
cc1553_NOT_ALLOWED_IN_MODE	If the function is called when cc1553Create_device was not called with MODE_BC.
cc1553_NOT_ALLOWED_WHEN_STARTED	If the function is called after a call to cc1553Start.
cc1553_PINGPONG_FAIL	If the device does not support ping-pong mode for retries.

6.3.2 Configuring the Frame Sequence

Function : **cc1553BCConfigure_frames**

Purpose : Configure the message data and frame sequence that the BC will transmit. Each frame consists of one or two MIL-STD-1553B command words to send to RTs. The data words used by the frame is specified by an index into the list of messages. Frames can be linked together via index based on the opcode and condition flags, otherwise they will be processed as per the list sequence.

Arguments :

<dev_id>	Device identifier on the PCI bus. As used in cc1553Create_device.
<p_frames>	Pointer to user supplied list of frames.
<frames_nr>	Number of frames in <p_frames> list.
<p_messages>	Pointer to user supplied list of messages.
<messages_nr>	Number of messages in <p_messages> list.

Returns :

cc1553_OK	On success.
cc1553_DEV_NOT_INITIALIZED	If no corresponding successful call was made to cc1553Create_device.
cc1553_INVALID_PARAM	If a function argument is incorrect or out of range.
cc1553_NOT_ALLOWED_IN_MODE	If the function is called when cc1553Create_device was not called with MODE_BC.
cc1553_NOT_ALLOWED_WHEN_STARTED	If the function is called after a call to cc1553Start.
cc1553_INVALID_MESSAGE_INDEX	If a frame references a non-existent message.
cc1553_OUT_OF_MEMORY	If the device does not have enough memory to allocate all the frames and messages.

Message :

p_data	Pointer to a user supplied buffer for RX or TX data. When used for TX data this buffer may be referenced by multiple frames.
message_size	The number of MIL-STD-1553B data words in the message.

Frame :

control	Frame control detail.
cmd1	MIL-STD-1553B command word, use cc1553Encode_command_word to populate. During RT-to-RT transfers this is the receive command.
cmd2	Only applicable during RT-to-RT transfers. The transmit command.

msg_index	The zero-based index into the list of messages containing the data words for TX or the buffer location for RX. It is allowed to use the same TX message in more than one frame.
status1	MIL-STD-1553B status word returned by the RT. During RT-to-RT transfers it is the transmitting RT's status.
status2	Only applicable during RT-to-RT transfers. The receiving RT's status word.
branch_to_index	The zero-based index into the list of frames to jump to if required by the opcode and conditions.
timer_us	The value to set the minor frame timer or the message-to-message timer. Specified in microseconds.

Frame Control :

opcode	Frame operation, e.g. continue, branch. Some of the opcode actions depends on the condition flags, i.e. cond_XXX.
retries	The number of times the messages will be resent when the RT status word indicates an error. Range is 1 to 4.
channel	Indicates which bus to use for transmission : CHANNEL_A or CHANNEL_B. Affected by cc1553BCEnable_pingpong.
is_RTtoRT	Indicates that the frame is an RT-to-RT transfer and that cmd2 is used.
cond_no_response	Condition is met if the RT does not return a status word.
cond_msg_error	Condition is met if the RT status word Message Error bit is set.
cond_is_busy	Condition is met if the RT status word Busy bit is set.
cond_terminal_flag	Condition is met if the RT status word Terminal Flag bit is set.
cond_subsystem_fail	Condition is met if the RT status word Subsystem Fail bit is set.
cond_instrumentation	Condition is met if the RT status word Instrumentation bit is set.
cond_service_request	Condition is met if the RT status word Service Request bit is set.
block_access_error	Indicates a protocol error in the RT's response. Always initialise to zero.

Opcode :

END_OF_LIST	Indicates the last frame in the sequence, no data words are sent. cc1553Wait_event will raise a END_FRAME_LIST event.

LOAD_MSG_TO_MSG_TIMER	Loads the message-to-message timer with <timer_us>. This will cause the specified delay before the device proceeds to the next frame. No data words are sent.
LOAD_MINORFRAME_TIMER	Loads the minor frame timer with <timer_us>. No data words are sent.
GOTO	Proceed immediately to the frame in <branch_to_index>. No data words are sent.
CONTINUE	Execute the current frame and continue with the next frame in the list.
RAISE_EVENT_CONTINUE	Raise the event FRAME_ACCESSED and continue with the next frame in the list.
BRANCH_UNCONDITIONALLY	Execute the current frame and then proceed to the <branch_to_index> frame.
BRANCH_ON_CONDITION	Execute the current frame and if any of the conditions are met, proceed to the <branch_to_index> frame. Otherwise the next frame in the list will be processed.
RETURN_TO_BRANCH	Returns to the frame saved by BRANCH_UNCONDITIONALLY or BRANCH_ON_CONDITION. No data words are sent.
RETRY_ON_CONDITION	If any of the conditions are met, resend the frame <retries> times. Otherwise the next frame in the list will be processed.
RETRY_ON_CONDITION_THEN_BRANCH	After retries proceed to the frame in <branch_to_index>. Otherwise the next frame in the list will be processed.
RETRY_ON_CONDITION_THEN_BRANCH_IF_RETRIES_FAIL	If all the retries fail proceed to the frame in <branch_to_index>.
CALL	Proceed to the frame <branch_to_index> and save the current frame for RETURN_TO_CALL. No nested calls are allowed, each call should be followed with a return before another call is used. No data words are sent.
RETURN_TO_CALL	Return to the frame saved by CALL.

Note : Refer to Figure 2 for an example of frame sequence.

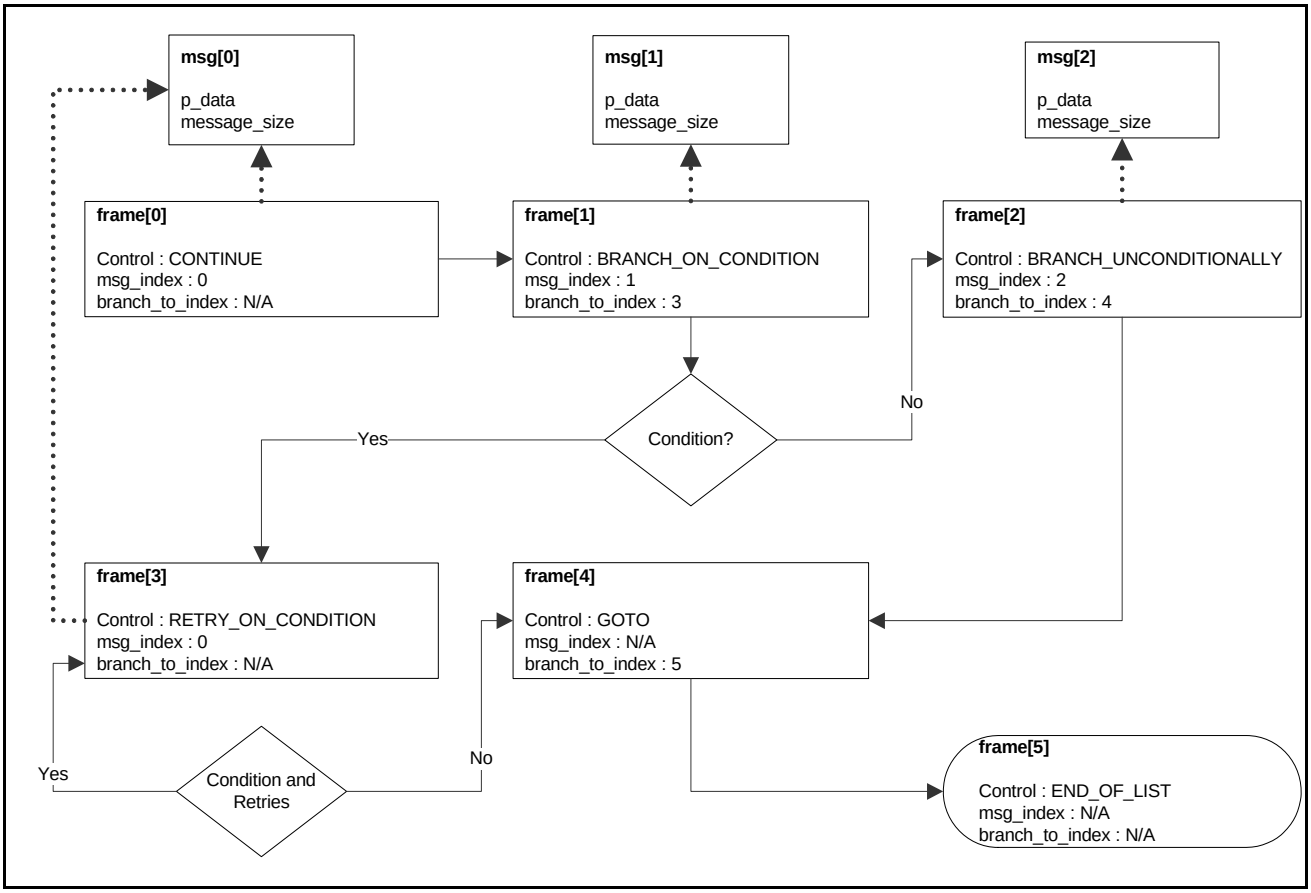

Figure 2 : Frame Example

6.3.3 Creating a Command Word

Function : **cc1553Encode_command_word**

Purpose : Create a MIL-STD-1553B command word.

Arguments :

<rt_addr>	Remote Terminal address.
<dir>	RX or TX from the RT point of view.
<sub_addr>	Sub address or mode code.
<size>	Number of words in the message or the mode code value.

Returns :

MIL-STD-1553B command word.

6.3.4 Retrieving Frame Status

Function : **cc1553BCGet_frame**

Purpose : Retrieve frame information, e.g. the RT status word.

Arguments :

<dev_id>	Device identifier on the PCI bus. As used in cc1553Create_device.
<p_frame>	Pointer to user supplied buffer.
<index>	Position of the frame in the list as used in cc1553BCConfigure_frames.

Returns :

cc1553_OK	On success.
cc1553_DEV_NOT_INITIALIZED	If no corresponding successful call was made to cc1553Create_device.
cc1553_INVALID_PARAM	If a function argument is incorrect or out of range.
cc1553_NOT_ALLOWED_IN_MODE	If the function is called when cc1553Create_device was not called with MODE_BC.
cc1553_INVALID_FRAME_INDEX	If <index> references a non-existent frame.
<p_frame>	The frame detail.

6.3.5 Retrieving Message Data

Function : **cc1553BCGet_message**

Purpose : Retrieves message content, e.g. when a RT has transmitted data.

Arguments :

<dev_id>	Device identifier on the PCI bus. As used in cc1553Create_device.
<p_message>	Pointer to user supplied buffer.
<index>	Position of the message in the list as used in cc1553BCConfigure_frames.

Returns :

cc1553_OK	On success.
cc1553_DEV_NOT_INITIALIZED	If no corresponding successful call was made to cc1553Create_device.
cc1553_INVALID_PARAM	If a function argument is incorrect or out of range.
cc1553_NOT_ALLOWED_IN_MODE	If the function is called when cc1553Create_device was not called with MODE_BC.
cc1553_INVALID_MESSAGE_INDEX	If index references a non-existent message.
<p_message>	The message detail.

6.3.6 Resending the Frame Sequence

Function : **cc1553BCRestart_frames**

Purpose : Reset the frames, to allow resending the frame sequence set via cc1553BCConfigure_frames.

Arguments :

<dev_id>	Device identifier on the PCI bus. As used in cc1553Create_device.

Returns :

cc1553_OK	On success.
cc1553_DEV_NOT_INITIALIZED	If no corresponding successful call was made to cc1553Create_device.
cc1553_INVALID_PARAM	If a function argument is incorrect or out of range.
cc1553_NOT_ALLOWED_IN_MODE	If the function is called when cc1553Create_device was not called with MODE_BC.
cc1553_INVALID_FRAME_INDEX	If no frames have been configured.

6.4 Bus Monitor Interface

All functions applicable in this mode are prefixed with "cc1553BM". These functions will return an error code if cc1553Create_device was not called with MODE_BM.

Some of these function calls are not allowed after a call to cc1553Start.

6.4.1 Promiscuous Mode

Function : **cc1553BMEnable_promiscuous_mode**

Purpose : Allows the monitoring of all RTs on the bus, or allows the filtering of specific terminals with cc1553BMMonitor_terminal.

Arguments :

<dev_id>	Device identifier on the PCI bus. As used in cc1553Create_device.
<enable>	cc1553TRUE / cc1553FALSE.

Returns :

cc1553_OK	On success.
cc1553_DEV_NOT_INITIALIZED	If no corresponding successful call was made to cc1553Create_device.
cc1553_INVALID_PARAM	If a function argument is incorrect or out of range.
cc1553_NOT_ALLOWED_IN_MODE	If the function is called when cc1553Create_device was not called with MODE_BM.
cc1553_NOT_ALLOWED_WHEN_STARTED	If the function is called after a call to cc1553Start.

6.4.2 Monitoring a Specific Terminal

Function : **cc1553BMMonitor_terminal**

Purpose : Allows the monitoring of specific terminals.

Arguments :

<dev_id>	Device identifier on the PCI bus. As used in cc1553Create_device.
<addr>	Remote Terminal Address to monitor.
<enable>	cc1553TRUE / cc1553FALSE

Returns :

cc1553_OK	On success.
cc1553_DEV_NOT_INITIALIZED	If no corresponding successful call was made to cc1553Create_device.
cc1553_INVALID_PARAM	If a function argument is incorrect or out of range.
cc1553_NOT_ALLOWED_IN_MODE	If the function is called when cc1553Create_device was not called with MODE_BM.

cc1553_NOT_ALLOWED_WHEN_STARTED	If the function is called after a call to cc1553Start.

6.4.3 Retrieving Logged Messages

Function : **cc1553BMGet_log**

Purpose : Retrieves an entry from the monitor log. If used in conjunction with cc1553Wait_event, <index> can be set to one more than the value of the event <log_index>.

Arguments :

<dev_id>	Device identifier on the PCI bus. As used in cc1553Create_device.
<index>	Log entry index.
<p_log>	Pointer to a user supplied buffer.

Returns :

cc1553_OK	On success.
cc1553_DEV_NOT_INITIALIZED	If no corresponding successful call was made to cc1553Create_device.
cc1553_INVALID_PARAM	If a function argument is incorrect or out of range.
cc1553_NOT_ALLOWED_IN_MODE	If the function is called when cc1553Create_device was not called with MODE_BM.
cc1553_INVALID_LOG_INDEX	The monitor block is invalid.
<p_log>	The captured log detail.

Log Detail :

channel	CHANNEL_A or CHANNEL_B.
is_RTtoRT	Indicates a RT-to-RT message.
is_message_error	Indicates a message error.
is_mode_code_no_data	Indicates a mode code with no data word.
is_broadcast	Indicates a broadcast message.
error_timeout	Too few data words received.
error_overrun	Too many data words received.
error_parity	Parity error detected.
error_decoding	Manchester decoding error detected.
cmd1	During RT-to-RT transfers this is the receive command.
cmd2	Only applicable during RT-to-RT transfers. The transmit command.
data	Message contents.
data_size	Message size in words.

status1	RT response status word. During RT-to-RT it is the transmitting RT.
status2	Only applicable during RT-to-RT transfers. The receiving RT.
timer_us	Internal time-tag value, i.e. message timestamp in microseconds.

6.4.4 Configuring Log Size

Function : **cc1553BMSet_log_size**

Purpose : Configures the number of logs before an event is raised. The device will store one more log than <size> in internal memory.

Arguments :

<dev_id>	Device identifier on the PCI bus. As used in cc1553Create_device.
<size>	Number of log entries.

Returns :

cc1553_OK	On success.
cc1553_DEV_NOT_INITIALIZED	If no corresponding successful call was made to cc1553Create_device.
cc1553_INVALID_PARAM	If a function argument is incorrect or out of range.
cc1553_NOT_ALLOWED_IN_MODE	If the function is called when cc1553Create_device was not called with MODE_BM.
cc1553_NOT_ALLOWED_WHEN_STARTED	If the function is called after a call to cc1553Start.
cc1553_OUT_OF_MEMORY	The logs exceeds the device memory.

7. Getting Started

This section contains example code on how to use the VxWorks Software Driver.

```
#include <vxWorks.h>
#include "1553HostDriver.h"
#include <sysLib.h>
#include <stdio.h>
#include <string.h>

void cc1553RTExample(void)
{
  cc1553Status       func_status;
  cc1553DeviceMode   adapter_type;
  int                cnt;
  cc1553RTStatusWord rt_status;
  cc1553Event        event;
  char               event_desc[50];

  printf("Creating device.\n");
  func_status = cc1553Create_device(0, MODE_RT);
  if (func_status != cc1553_OK)
  {
    cc1553Print_status(func_status);
    return;
  }

  printf("Retrieving adapter type.\n");
  adapter_type = MODE_INVALID;
  func_status = cc1553Adapter_type(0, &adapter_type);
  if ( (func_status != cc1553_OK) || (adapter_type != MODE_RT) )
  {
    cc1553Print_status(func_status);
    return;
  }

  rt_status.standard         = MIL_STD_1553_B;
  rt_status.clear_immediatly = 1;
  rt_status.instrumentation  = 0;
  rt_status.service_request  = 0;
  rt_status.is_busy          = 0;
  rt_status.subsytem_flag    = 0;
  rt_status.terminal_flag    = 0;

  printf("Setup status word.\n");
  func_status = cc1553RTSet_statusword(0, &rt_status);
  if (func_status != cc1553_OK)
  {
    cc1553Print_status(func_status);
    return;
  }

  printf("Allocating buffers.\n");
  for (cnt = 1; cnt <= 5; cnt++)
  {
    func_status = cc1553RTConfigure_buffers(0, RX, SUBADDR,
                                            cnt, cc1553_MAX_MESSAGE_WORDS, cc1553TRUE);
    if (func_status != cc1553_OK) break;

    func_status = cc1553RTLegalize_command(0, RX, SUBADDR,
                                           cnt, cc1553FALSE, cc1553TRUE);
    if (func_status != cc1553_OK) break;

    func_status = cc1553RTLegalize_command(0, RX, SUBADDR,
                                           cnt, cc1553TRUE, cc1553TRUE);
    if (func_status != cc1553_OK) break;
  }

  printf("Subaddresses 1 to %i (0x%x), RX buffers allocated and legalized.\n", cnt, cnt);

  printf("Setting terminal address.\n");
  func_status = cc1553RTSet_address(0, cnt);
  if (func_status != cc1553_OK)
  {
    cc1553Print_status(func_status);
    return;
  }

  cc1553Start(0);
```

```
  printf("Waiting for an event...\n");
  cc1553Wait_event(0, 20*sysClkRateGet(), &event);

  if (event.event_type == EVENT_RT)
  {
    if (event.rt.addr_or_mode == SUBADDR)
      strcat(event_desc, "Subaddress ");
    else
      strcat(event_desc, "Mode code ");

    if (event.rt.dir == RX)
      strcat(event_desc, "Rx");
    else
      strcat(event_desc, "Tx");

    printf("Address : %s %i\n", event_desc, event.rt.entry);
  }

  if (event.event_type == EVENT_TIMEOUT)
  {
    printf("No event received\n");
  }

  cc1553Stop(0);

  printf("Destroying device.\n");
  func_status = cc1553Destroy_device(0);
  if (func_status != cc1553_OK)
  {
    cc1553Print_status(func_status);
    return;
  }
}

void cc1553BCExample(void)
{
  cc1553Status        func_status;
  cc1553DeviceMode    adapter_type;
  int                 data_cnt;
  int                 msg_cnt;
  cc1553Event         event;
  char                text[80];
  cc1553BCControlWord control;
  const int           data_size   = 5;
  const int           msg_size    = 2;
  const int           frame_size  = 3;
  cc1553BCMessage     msgs[msg_size];
  cc1553BCFrame       frames[frame_size];

  printf("Creating device.\n");
  func_status = cc1553Create_device(0, MODE_BC);
  if (func_status != cc1553_OK)
  {
    cc1553Print_status(func_status);
    return;
  }

  printf("Retrieving adapter type.\n");
  adapter_type = MODE_INVALID;
  func_status = cc1553Adapter_type(0, &adapter_type);
  if ( (func_status != cc1553_OK) || (adapter_type != MODE_BC) )
  {
    cc1553Print_status(func_status);
    return;
  }

  printf("Setup messages.\n");
  for (msg_cnt = 0; msg_cnt < msg_size; msg_cnt++)
  {
    msgs[msg_cnt].p_data = malloc(data_size * cc1553_WORD_SIZE);
    for (data_cnt = 0; data_cnt < data_size; data_cnt++)
    {
      msgs[msg_cnt].p_data[data_cnt] = (0x1000U*msg_cnt) + (0x100U*msg_cnt) + (0x10U*msg_cnt) +
                          msg_cnt;
    }
    msgs[msg_cnt].message_size = data_size;
  }

  /* Set Defaults */
```

```
memset(frames, 0, sizeof(frames));

control.opcode              = CONTINUE;
control.retries             = 1;
control.channel             = CHANNEL_A;
control.is_RTtoRT           = cc1553FALSE;
control.cond_no_response    = 0;
control.cond_msg_error      = 0;
control.cond_is_busy        = 0;
control.cond_terminal_flag  = 0;
control.cond_subsystem_fail = 0;
control.cond_instrumentation = 0;
control.cond_service_request = 0;
control.block_access_error  = 0;

printf("Configure frames.\n");
control.opcode      = CONTINUE;
frames[0].control   = control;
frames[0].cmd1      = cc1553Encode_command_word(0x12, RX, 2, data_size);
frames[0].msg_index = 0;

control.opcode      = CONTINUE;
frames[1].control   = control;
frames[1].cmd1      = cc1553Encode_command_word(0x12, RX, 3, data_size);
frames[1].msg_index = 1;

control.opcode      = END_OF_LIST;
frames[2].control   = control;

func_status = cc1553BCConfigure_frames(0, frames, frame_size, msgs, msg_size);
if (func_status != cc1553_OK)
{
  cc1553Print_status(func_status);
  return;
}

cc1553Start(0);

while (1)
{
  printf("Waiting for an event...\n");
  cc1553Wait_event(0, 20*sysClkRateGet(), &event);

  if (event.event_type == EVENT_BC)
  {
    switch (event.bc.event_type)
    {
      case END_FRAME_LIST  : sprintf(text, "End of list");     break;
      case ILLOGICAL_FRAME : sprintf(text, "Illogical frame"); break;
      case INVALID_OPCODE  : sprintf(text, "Invalid opcode");  break;
      case RETRY_FAILED    : sprintf(text, "Retries failed");  break;
      case FRAME_ACCESSED  : sprintf(text, "Frame accessed");  break;
      default              : sprintf(text, "Unknown");
    }
    printf("\n***BC Event*** %s, frame index: %i\n", text, event.bc.frame_index);
  }

  if (event.event_type == EVENT_TIMEOUT)
  {
    printf("No event received\n");
  }

  cc1553Stop(0);
  cc1553BCRestart_frames(0);
  cc1553Start(0);
}

cc1553Stop(0);

printf("Release message buffers.\n");
for (msg_cnt = 0; msg_cnt < msg_size; msg_cnt++)
{
  free(msgs[msg_cnt].p_data);
}

printf("Destroying device.\n");
func_status = cc1553Destroy_device(0);
if (func_status != cc1553_OK)
{
  cc1553Print_status(func_status);
```

```
  }
}

void cc1553BMExample(void)
{
  cc1553Status      func_status;
  cc1553DeviceMode  adapter_type;
  cc1553Event       event;
  cc1553BMLog       log;
  int               cnt;
  int               logs_rx;

  printf("Creating device.\n");
  func_status = cc1553Create_device(0, MODE_BM);
  if (func_status != cc1553_OK)
  {
    cc1553Print_status(func_status);
    return;
  }

  printf("Retrieving adapter type.\n");
  adapter_type = MODE_INVALID;
  func_status = cc1553Adapter_type(0, &adapter_type);
  if ( (func_status != cc1553_OK) || (adapter_type != MODE_BM) )
  {
    cc1553Print_status(func_status);
    return;
  }

  cc1553BMEnable_promiscuous_mode(0, cc1553TRUE);
  cc1553BMSet_log_size(0, 1);

  cc1553Start(0);

  while (1)
  {
    printf("Waiting for an event...\n");
    cc1553Wait_event(0, 20*sysClkRateGet(), &event);

    if (event.event_type == EVENT_BM)
    {
      printf("\n***BM Event*** log_index: %i\n", event.bm.log_index);

      for (cnt = 0; cnt <= (event.bm.log_index+1); cnt++)
      {
        func_status = cc1553BMGet_log(0, cnt, &log);
        if (func_status == cc1553_OK)
        {
          logs_rx++;
        }
      }
    }

    if (event.event_type == EVENT_TIMEOUT)
    {
      printf("No event received\n");
    }
  }

  cc1553Stop(0);

  printf("Destroying device.\n");
  func_status = cc1553Destroy_device(0);
  if (func_status != cc1553_OK)
  {
    cc1553Print_status(func_status);
  }
}
```

8. Contact Details

8.1 Contact Person

Direct all correspondence and / or support queries to the Project Manager at C²I² Systems.

8.2 Physical Address

C²I² Systems
Unit 3, Rosmead Place, Rosmead Centre
67 Rosmead Avenue
Kenilworth
Cape Town
7708
South Africa

8.3 Postal Address

C²I² Systems
P.O. Box 171
Rondebosch
7701
South Africa

8.4 Voice and Electronic Contacts

Tel : (+27) (0)21 683 5490
Fax : (+27) (0)21 683 5435
Email : info@ccii.co.za
Email : support@ccii.co.za
URL : http://www.ccii.co.za/

8.5 Product Support

Support on C²I² Systems products is available telephonically between Monday and Friday from 09:00 to 17:00 CAT. Central African Time (CAT = GMT + 2).

Annexure A

Making Changes to sysLib.c for x86

The PCI free memory space needs to be defined in the memory descriptor table. Consult the relevant reference manual and obtain the upper address of the PCI memory. Allocate at least 1 MByte of memory per adapter. Subtract that amount from the upper address of the PCI memory, and use this value as the base of the PCI memory space.

Note : If there are other devices on the PCI bus, it may be necessary to allocate more memory.

Example : Allocate 10 MBytes of memory. If the upper address of the PCI memory space is defined as 0xFFF00000, then subtracting 10 MBytes gives a base address of : 0xFFF00000 - 0xA00000 = 0xFF500000.

In the Personal Computer (PC) 386/486/Pentium/Pentiumpro system-dependent library (sysLib.c), code (**shown in bold text**) needs to be added to the memory descriptor table, sysPhysMemDesc[] :

```
#ifndef CPU_PCI_MEM_ADRS
#define CPU_PCI_MEM_ADRS    0xFF500000  /* base of PCI MEM addr */
#endif

PHYS_MEM_DESC sysPhysMemDesc [] =
{
    /* adrs and length parameters must be page-aligned (multiples of 4KB/4MB) */
    #if(VM_PAGE_SIZE == PAGE_SIZE_4KB)

        /* lower memory */
            ...
        /* video ram, etc */
            ...
        /* upper memory for OS */
            ...
        /* upper memory for Application */
            ...
        /* PCI I/O space  */
        {
            (void *) CPU_PCI_MEM_ADRS,
            (void *) CPU_PCI_MEM_ADRS,
            (0xA00000),
            VM_STATE_MASK_VALID | VM_STATE_MASK_WRITABLE | VM_STATE_MASK_CACHEABLE,
            VM_STATE_VALID | VM_STATE_WRITABLE | VM_STATE_CACHEABLE_NOT
        },

        /* entries for dynamic mappings - create sufficient entries */
        DUMMY_MMU_ENTRY,
        DUMMY_MMU_ENTRY,
        DUMMY_MMU_ENTRY,
            ...
            ...

    #else
        ...
```