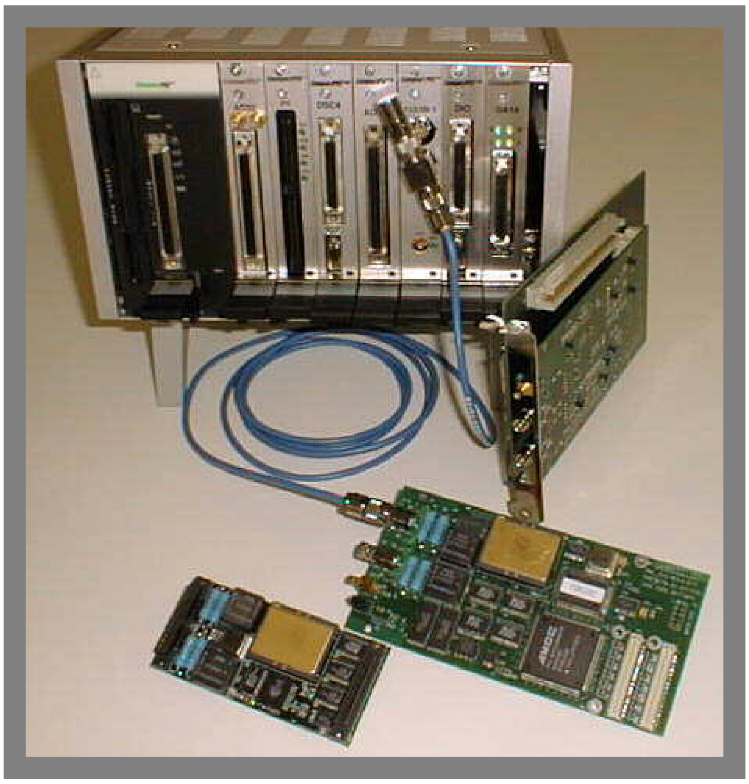
PCI-1553B 1553 Interface Card User Documentation



Omicron Development Ltd,
Unit 1, Meader's Yard, Station Rd, Stalbridge, Dorset DT10 2RQ

Tel / fax :	(44) 1963 – 363632
Web site :	http://www.omicrondev.co.uk
e-mail :	sales@omicrondev.co.uk

LIABILITY

ACKNOWLEDGEMENTS

The information presented in this document uses information provided by the following companies. Omicron is greatly obliged to Alphi Technology and Aeroflex for permission to use and reproduce this material.

Aeroflex UTMC & Aeroflex Circuit Technology
35, South Service Rd
Plainview New York 11803
Tel: 516-694-6700

Alphi Technology Corporation
6202 South Maple Ave #120
Tempe, Arizona 85283
Tel : 480-838-2428

Table of contents

1 General Description

The PCI-1553B module provides Bus Controller (BC), Remote Terminal (RT) and Bus Monitor (BM) dual-redundant operations on the MIL-STD-1553 bus. The PCI form factor provides easy installation.

- UTMC Summit RISC based processor unit.
- 64K x 16 bit dual ported SRAM
- Supports Bus Controller, Remote Terminal and Bus Monitor mode
- RT address and operational modes are program or jumper selectable
- 1553 bus long or short stub jumper option

2 Functional Description

A functional block diagram of the interface card is depicted below in Figure 1. The PCI-1553B is designed around the SUMMIT Risc controller that is used to manage the 1553 BUS.

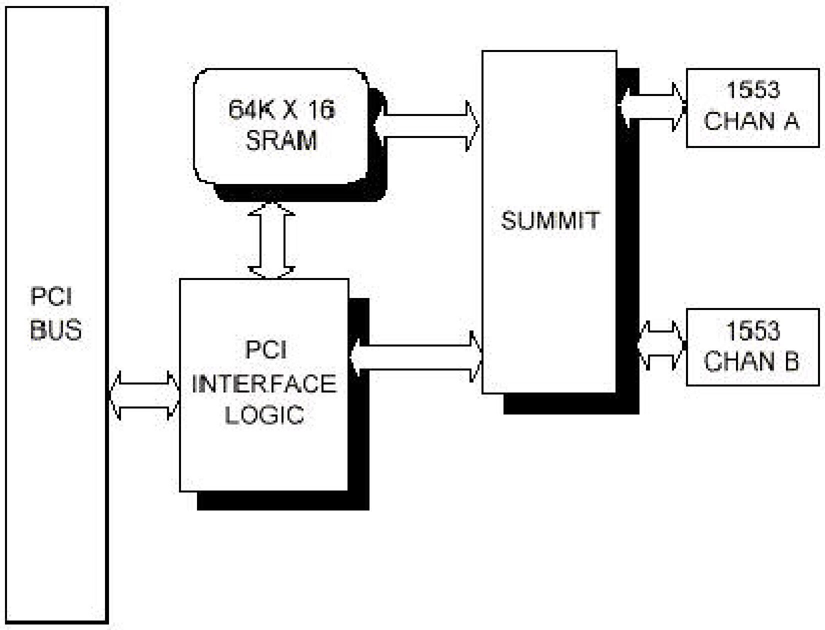


Figure 1

3 Host (PCI) side

3.1 PCI configuration registers

The card presents the following configuration values to the PCI system, based on the values stored in the NVRAM device read by the AMCC PCI interface chip.

Register	Value (Meaning)
Vendor ID	$13C5 (Alphi)
Device ID	$0306 (PCI-1553)
Revision ID	$000
Class Code	$FF0000 (Not in defined class codes)
Interrupt Line	$FF
Interrupt Pin	A
Multifunction Device	No
Build In Self Test	No
Latency Timer	$00
Minimum Grant	$00
Maximum Latency	$00
Expansion ROM Size	None

Table 1: PCI Configuration Registers

3.2 Memory and register map summary

The addresses for the following registers and memory locations are based upon the assignment from the host processor. The PCI-1553 uses 3 of the 5 AMCC mapped base address registers. These base address registers are written by the PCI configurator after scanning the AMCC configuration space. The AMCC is normally programmed at the factory to request the following resources:

64 bytes of memory space for the AMCC PCI Operation Registers.
128 bytes for the SUMMIT registers and local STATUS Registers.
128 Kbytes of space for the Dual Ported SRAM

Base address registers 2 and 4 are not used and are disabled at the factory.

The following table specifies address offsets relative to the AMCC base address registers. For example, the AMCC PCI registers occupy 64 bytes of contiguous space relative to Base address 0. If the PCI configuration programs Base address 0 with the value $6000, then the AMCC registers would occupy memory locations 6000 to 603F.

BASE ADDR	FROM	TO	DESCRIPTION	TYPE
0	$00000000	$0000003F	AMCC PCI Operation Registers	MEM
1	$00000000	$0000007F	SUMMIT IO Space	MEM
3	$00000000	$0001FFFF	SUMMIT DUAL PORTED SRAM	MEM

Table 2

NOTE: The AMCC has been programmed to request memory above 1 Mbyte.

3.3 PCI operation registers

The host processor communicates with the PCI-1553 module via the AMCC pass-through interface. After the base address registers have been programmed by the PCI configurator, PCI bus cycles are "passed through" to the local bus. Therefore, the PCI Operation Registers are not normally used by applications software. Only IMB4 Byte 3, MBEF, and INTCSR are useful to the PCI-1553 module. IMB4 is written by the PCI-1553 hardware when the SUMMIT generates an interrupt. The PCI Operation Register Offsets are shown below:

Offset	Register Name
$00	OMB1 Outgoing Mailbox Register 1
$04	OMB2 Outgoing Mailbox Register 2
$08	OMB3 Outgoing Mailbox Register 3
$0C	OMB4 Outgoing Mailbox Register 4
$10	IMB1 Incoming Mailbox Register 1
$14	IMB2 Incoming Mailbox Register 2
$18	IMB3 Incoming Mailbox Register 3
$1C	IMB4 Incoming Mailbox Register 4
$20	FIFO Register Port (bi-directional)
$24	MWAR Master Write Address Register
$28	MWTC Master Write Transfer Counter
$2C	MRAR Master Read Address Register
$30	MRTC Master Read Transfer Counter
$34	MBEF Mailbox Empty/Full Status
$38	INTCSR Interrupt Control/Status Register
$3C	MCSR Bus Master Control/Status Register

Table 3

For more information about these registers refer to the AMCC PCI controller manual.

3.4 Internal organization

The PCI-1553 facilitates host access to the :

- SUMMIT Registers
- Dual Port SRAM
- Status register

4 Summit registers

4.1 Remote terminal registers

Register Number	Name	Register Offset
0	Control	$00
1	Operational Status	$02
2	Current Command	$04
3	Interrupt Mask	$06
4	Pending Interrupt	$08
5	Interrupt Log List Pointer	$0A
6	BIT Word	$0C
7	Time-Tag	$0E
8	Remote Terminal Descriptor Pointer	$10
9	Status Word Bits	$12
10-15	Not used	$14-$1E
16-31	Illegalisation Registers	$20- $3E

Table 4

4.2 Bus controller registers

Register Number	Name	Register Offset
0	Control	$00
1	Operational Status	$02
2	Current Command Block	$04
3	Interrupt Mask	$06
4	Pending Interrupt	$08
5	Interrupt Log List Pointer	$0A
6	BIT Word	$0C
7	Minor-Frame Timer	$0E
8	Command Block Pointer	$10
9	Not used	$12
10	BC Command Block Initial Count	$14
11-31	Not Applicable	$16- $3E

Table 5

4.3 Monitor terminal registers

Register Number	Name	Register Offset
0	Control	$00
1	Operational Status	$02
2	Current Command Block	$04
3	Interrupt Mask	$06
4	Pending Interrupt	$08
5	Interrupt Log List Pointer	$0A
6	BIT Word	$0C
7	Time-Tag	$0E
8-10	Not used	$10-$14
11	Initial Monitor Command Block Pointer	$16
12	Initial Monitor Data Pointer	$18
13	Monitor Block Counter	$1A
14	Monitor Filter	$1C
15	Monitor Filter	$1E
16-31	Not used	$20- $3E

Table 6

4.4 Shared memory SRAM

The PCI-1553 has a 64K x 16-bit Shared Memory. The base address of the SRAM is provided by the PCI host controller. A local Flash E2 prom defines the resources needed by the PCI-1553 module. SRAM access are only in 16-bit mode. Arbitration between the Summit and the PCI Bus is made by the local hardware. Summit access to the SRAM takes priority over any pending host access. Therefore, the host access will be held off until the Summit access completes.

4.5 PCI summit resource requirements

Shared memory space

NAME	OFFSET	DATA	R/W	COMMENTS
MEM SPACE	$0 - $1FFFF	D00-D15	R/W	Shared / Static RAM 64K x 16-bit (128KB)

Summit register space

NAME	OFFSET	DATA	R/W	COMMENTS
SUMMIT REGISTERS	$00-$3F	D00-D15	R/W	SUMMIT Registers
PCI STATUS	$42-$43	D00-D15	R	Status Register Only D00-D07 valid

Status register

The STATUS register is used to determine the status of PCI-1553 jumper settings. The Status Register provides the following status bits:

Bit	Name	Function
0	MSEL0	Summit mode of Operation
1	MSEL1	
2	LOCK	Status of the Lock input Pin
3	READY	Status of the Ready Output Pin
4-7	NOT USED	

Table 7

4.6 *Summit mode of operation*

Mode select 0, in conjunction with Mode select 1, determines the Summit mode of operation. The table below describes these modes.

MSEL1	MSEL0	Mode Of Operation
0	0	Bus controller = SBC
0	1	Remote Terminal = SRT
1	0	Monitor Terminal = SMT
1	1	SMT/SRT

Table 8

Lock
This read only bit shows the inverted state of the LOCK input pin. The LOCK pin is latched on the rising edge of MRST. If the mode of operation must change, the user must perform a MRST.

Ready
This read only bit shows the inverted state of the output pin READY and is cleared on reset. This signal indicates the Summit has completed initialisation or BIT, and regular execution may begin.

4.7 Local interrupt sources

The SUMMIT has two interrupt lines. These interrupts are OR-ed to the INTREQ0 pin on the PCI bus. The Summit will provided the Interrupt Vector Register contents to the PCI bus on D00-D07 during an interrupt acknowledge cycle.

The source of each interrupt is listed below:

Interrupt name	Description
MSG_INT	Message interrupt. This pin is active for three clock cycles upon the occurrence of interrupt events which are enabled.
YF_INT	You Failed interrupt. This pin is active for three clock cycles upon the occurrence of interrupt events which are enabled.

Table 9

4.7 Byte-swap enable

The PCI-1553 has the ability to perform hardware byte-swapping. The jumper TTJ1 determines the behaviour of the byte-swapping hardware. When TTJ1 is OPEN, no swapping occurs (low byte first). When TTJ1 is SHORTED, swapping occurs (high byte first). This feature allows the PCI-1553 memory and registers to appear correctly to high byte first processors such as the PowerPC.

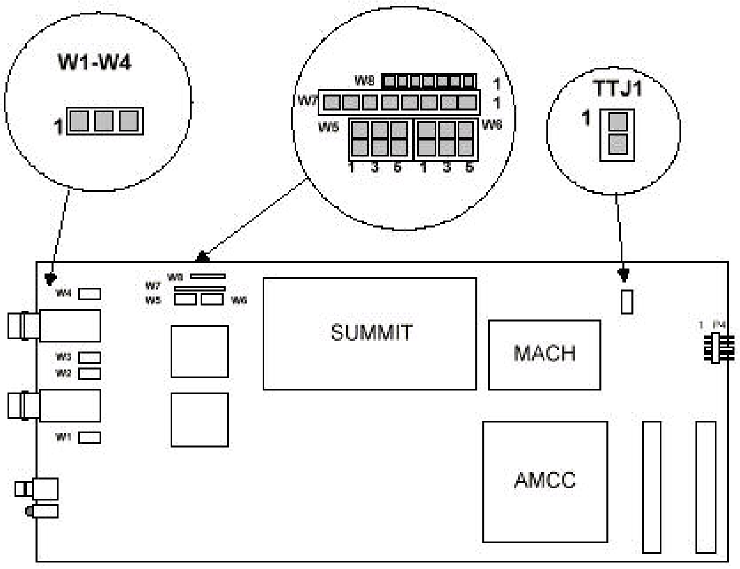


Figure 2

4.8 Jumper Settings

JUMPER	FACTORY SETTING	DESCRIPTION
W1	1-2	Long Stub/Short Stub Output channel A - (Short Stub)
W2	2-3	Long Stub/Short Stub Output channel A + (Short Stub)
W3	2-3	Long Stub/Short Stub Output channel B + (Short Stub)
W4	1-2	Long Stub/Short Stub Output channel B - (Short Stub)
W5	None	Remote terminal address RTA3-RTPT
W6	None	Remote terminal address RTA0-RTA2
W7	None	Mode of Operation of the SUMMIT
W8	None	Summit JTAG test
TTJ1	None	Byte-Swap Enable
P4	None	MACH 445 programming plug

Table 10

W7 Mode of Operation Jumper Selection

Signal	Signal Jumper set	Description
A/B* STD	1-2	Military standard – set = Mil_STD_1553A, clear = Mil_STD_1553B
LOCK	3-4	When set prevents software change to the RT address, A/B* STD and Mode select
MSEL1	5-6	See Mode of Operation Table above
MSEL0	7-8	See Mode of Operation Table above

Table 11

W6 RT Address Selection

Signal	Jumper set	Description
RTA2	1-2	RT Address BIT 2
RTA1	3-4	RT Address BIT 1
RTA0	5-6	RT Address BIT 0

Table 12

W5 RT Address Selection

Signal	Jumper set	Description
RTPT	1-2	RT Address Parity
RTA4	3-4	RT Address BIT 4
RTA3	5-6	RT Address BIT 3

Table 13

TTJ1 Byte-Swap Enable

Jumper set	Description
1-2	Open = low byte first (PCI Default), Shorted = high byte first

Table 14

5 1553 Bus Controller - user documentation

5.1 BC initialization structure

The following structure shall be used by the **ms1553_bc_init** function to specify the initial device options and operating modes:

```
typedef struct bc_init
{
uword        BC_Control;       /* SBC Control Word Preferences      */
             boolean   BC_MS1553A; /* 1 = 1553A Standard, 0 = 1553B */
uword        DataBufferN;      /* Max Number of 32 word data buffers*/
uword        ModeBufferN;      /* Max Number of Mode code buffers   */
uword        BC_IrqMask;       /* SBC High Priority Interrupt Mask  */
} BC_INIT;
```

BC_Control
This field shall be used to specify options for the Summit Control register. These options can be combined to form a bit mask. The following option is currently supported:

RT_BCENEnables Broadcast operation; subaddress 31 is broadcast

BC_MS1553A
This field specifies the use of 1553A or 1553B standards. 1 = 1553A, 0 = 1553B.

DataBufferN
This field specifies the maximum number of 32 word data buffers.

ModeBufferN
This field specifies the maximum number of single word mode code buffers.

BC_IrqMask
This field shall determine which high priority interrupts (YF_INT) are enabled. The following high priority interrupts are currently defined:

BC_BITF	Enable BIT Fail Interrupt
BC_WRAPF	Enables Wrap Fail Interrupt
BC_DMAF	Enables DMA Fail Interrupt

5.2 BC device creation

The following example illustrates how to create a Bus Controller device:

```
BC_INIT     bcInit;
SYS_CONFIG  Cfg;

sys1553Init(&Cfg);

/* Setup SYS_CONFIG structure -----------------------------------*/
   Cfg.BC_IOAddr = (SUMMIT_BC *)CPU_SUMMIT_IO_ADRS;
                                /* point to i/o ports           */
   Cfg.BC_MemAddr= (uword *) CPU_SUMMIT_DUAL_PORT_SRAM_ADRS;
                                /* point to dual-port mem       */
   Cfg.BC_IntVector = PCI_PRI_INTA_VEC;
   Cfg.BC_IntLevel  = PCI_INT_LVL1;
/* Setup BC_INIT structure -----------------------------------------*/
   bcInit.BC_Control    = BC_BCEN;/* SBC Control Word Preferences  */
   bcInit.BC_MS1553A    = 0;      /* 1 = 1553A Standard, 0 = 1553B */
   bcInit.DataBufferN   = 128;    /* Max Number of 32 word data
.                                    buffers  */
   bcInit.ModeBufferN   = 32;     /* Max No of Mode code buffers   */
   bcInit.BC_IrqMask    = $0000;  /* SBC High Priority Int. Mask   */

   ms1553_bc_init( &Cfg, &bcInit ); /* Call init function          */
```

5.3 Driver data structures

This structure is used to communicate status information to the application layer code.

```
   typedef struct status_info
   {
   boolean   completed;    /* TRUE when command is done           */
   boolean   error;        /* TRUE indicates an error has occurred */
   uword     runtime_error;/* run time error code                 */
   uword     status1;      /* status word returned from RT        */
   uword     status2;      /* status of source RT in RT-RT command */
   ulong     msg_count;    /* number of intact messages completed  */
   ulong     error_count;  /* number of messages with errors      */
   boolean   refresh_flag; /* indicates a refresh request         */
   uword     *refresh_data;/* pointer to data for refresh         */
   boolean   skip_flag;    /* used by driver - dynamic skip request*/
   boolean   toggle_flag;  /* used by driver - message-toggle fn   */
   } STAT_INFO;
```

5.4 I/O request data structures

TRDATA_PACKET

```
typedef struct _trdata_packet
{
int rtAddr;           /* Remote Terminal Address             */
int trBit;            /* Transmit/Receive BIT                */
int subAddr;          /* Remote Terminal Sub-Address         */
int nWords;           /* Number of words to send or receive */
uword *data;          /* Pointer to data                     */
STAT_INFO *pInfo;     /* */
COMBLOCK *cb;         /* Command Block pointer from driver  */
} TRDATA_PACKET;
```

TRMODE_PACKET

```
typedef struct _trmode_packet
{
int rtAddr;           /* Remote Terminal Address             */
int trBit;            /* Transmit/Receive BIT                */
int mcode;            /* Mode Code                           */
uword *data           /* Pointer to data                     */
STAT_INFO *pInfo;     /* */
COMBLOCK *cb;         /* Command Block pointer from driver  */
} TRMODE_PACKET;
```

RTRT_PACKET

```
typedef struct _rtrt_packet
{
int SrcAddr;          /* Remote Terminal Address SOURCE     */
int SrcSubAddr;       /* Remote Terminal Subaddress SOURCE  */
int DstAddr;          /* Remote Terminal Address DESTINATION*/
int DstSubAddr;       /* Remote Terminal Subaddress DEST.   */
int nWords;           /* Number of words to send or receive */
uword *data;          /* Pointer to data                     */
STAT_INFO *pInfo;     /* */
COMBLOCK *cb;         /* Command Block pointer from driver  */
} RTRT_PACKET;
```

IO_PACKET

```
typedef struct _io_packet
{
uword type;
union
    {
    TRDATA_PACKET    trData;
    TRMODE_PACKET    trMode;
    RTRT_PACKET      rtRT;
    } pT;
} IO_PACKET;
```

SKIP_PACKET

```
typedef struct _skip_packet
{
COMBLOCK *cb;
boolean  skip;
} SKIP_PACKET;
```

EXEC_PACKET

```
typedef struct _exec_packet
{
COMBLOCK *cb;
boolean  wait;
} EXEC_PACKET;
```

IRQ_PACKET

```
typedef struct
{
void (*yf_int)(uword status); /* Fn to call on YF_INT   */
void (*msg_int)(uword,uword); /* Fn to call on MSG_INT  */
void (*msg_err)(uword,uword); /* Fn to call on MSG-Error*/
} IRQ_PACKET;
```

EOF_PACKET

```
typedef struct
{
COMBLOCK *cb;
boolean loop;
} EOF_PACKET;
```

FRAME

/* Frame template used by ms1553_compile_frame() */

```
typedef struct user_msg
{
ulong period;  /* period in uS to repeat this message */
ulong last_period;
IO_PACKET p;
COMBLOCK *com;
} FRAME;
```

FRAME_PACKET

```
typedef struct
{
FRAME   *msg;
uword   size;
boolean wait;
} FRAME_PACKET;
```

5.5 Remote Terminal I/O

IO_PACKET

```
typedef struct _io_packet
{
uword type;
union
   {
   TRDATA_PACKET trData;
   TRMODE_PACKET trMode;
   RTRT_PACKET   rtRT;
   } pT;
} IO_PACKET;
```

The ‘type’ field in the above structure indicates the type of I/O packet. The types currently defined are:

BC_TRDATA
BC_TRMCODE
BC_RTRT

TRDATA_PACKET
The TRDATA_PACKET is used to command the Remote Terminal to transmit or receive up to 32 16-bit words of data.

TRDATA_PACKET

```
typedef struct _trdata_packet
{
int rtAddr;          /* Remote Terminal Address          */
int trBit;           /* Transmit/Receive BIT             */
int subAddr;         /* Remote Terminal Sub-Address      */
int nWords;          /* Number of words to send/receive */
uword *data;         /* Pointer to data                  */
STAT_INFO *pInfo;    /* */
COMBLOCK *cb;        /* Command Block pointer returned  */
} TRDATA_PACKET;
```

STATUS INFORMATION
Command block status information is reported to the application through the STAT_INFO structure:

STATUS-INFO

```
typedef struct status_info
{
BOOLEAN error;       /* TRUE if error has occurred       */
UWORD runtime_error;/* run time error code              */
UWORD status1;       /* status word returned from RT     */
UWORD status2;       /* status word RT-RT command        */
ULONG msg_count;     /* number of error-free messages    */
ULONG error_count;   /* number of messages with errors   */
} STAT_INFO;
```

If any errors occurred, the “error” flag is set TRUE. The “runtime_error” word will contain an error code to indicate which run time error(s) have occurred. This error code is the “OR” of the possible error code bits, so there may be more than one error at a time. For BC mode these are:

BC_NO_ERROR
BC_MESSAGE_ERROR
BC_NO_RESPONSE
BC_RETRY_FAIL

If the value is equal to BC_NO_ERROR, no BC run time errors have occurred. The “status1” word is the RT status returned for this command. If it is an RT to RT transfer, “status1” will contain the destination RT status word and “status2” will contain the source RT status word. The interrupt service routine will accumulate a count of the number of messages without errors in “msg_count” and the number of messages with errors in “error_count."

ms1553_reset

This function shall perform a software reset. This only affects the internal encoder/decoders. This function shall not force the application to reinitialise the driver or any data pointers.

```
int ms1553_reset( void )
```

ms1553_set_interrupt

This function shall be use to install user defined interrupt handlers. The driver processes two types of interrupts. The first type is a high priority interrupt which is asserted via the YF_INT pin. The second type is a normal message interrupt asserted via the MSG_INT pin. The application can install a function to handle each type of interrupt. If a function pointer is not NULL, the function will be called by the interrupt service routine. The YF_INT handler function shall be passed a copy of the interrupt status word. The MSG_INT handler function shall be passed the Interrupt Identification (IIW) and Interrupt Address (IAW) Words taken from the Interrupt Log List. This function expects a pointer to the following data structure:

IRQ_PACKET

```
typedef struct irq_packet
{
void (*yf_int)( UWORD status );
void (*msg_int)( UWORD, UWORD );
} IRQ_PACKET;
```

int ms1553_set_interrupt(SYS_CONFIG *pCfg, IRQ_PACKET *pIrq);

ms1553_bc_tr_data

Compile a command block on the card to instruct an RT/subaddress to either receive or to transmit data. Remember that the T/R bit is with respect to the RT. This means for the RT to receive data from the BC, you are instructing the RT to receive. This command may be placed in a frame, which means it will not be executed until the frame is executed. Likewise, if you are compiling it in temporary mode, it will not execute until the current temporary buffer is executed. This means either this command or a later one must have the send" flag TRUE.

This command requires a pointer to the following data structure:

TRDATA_PACKET

```
typedef struct _trdata_packet
{
int rtAddr;         /* Remote Terminal Address            */
int trBit;          /* Transmit/Receive Bit               */
int subAddr;        /* Remote Terminal Sub-Address        */
int nWords;         /* Number of words to send or receive */
uword *data;        /* Pointer to data                    */
boolean send;       /* 1=Immediate, 0=Compile             */
COMBLOCK **cb;      /* Command Block pointer from driver  */
} TRDATA_PACKET;
```

int ms1553_bc_tr_data(TRDATA_PACKET *pData)

The following error codes are possible:

COMMAND_BLOCK_POOL_EMPTY
DATA_BLOCK_POOL_EMPTY

ms1553_bc_tr_same

This routine is used for the second through Nth repetition of an BC_TRDATA command in a frame. To understand why this is required, we must look at the way in which a typical frame is compiled. To achieve a set frequency for any particular frame command, it must be repeated one or more times, with appropriate delays.

If you were to use multiple BC_TRDATA commands, this would consume a separate data area for each command. If you wish to update the data that is being sent to an RT, you would have to update the data for each command, wasting host CPU time, generating phasing errors on your data in the frame, and wasting memory on the board, as well as in the application layer. The other parameters passed allow you to specify different parameters for the remainder of the command, if desired. The end result is common data areas.

If you wish to update data for this common area, on the fly, you will use the BC_CHGDATA command, and the data area on the board will be updated for ALL commands referencing it.

This command requires a pointer to the following data structure:

TRDATA-PACKET

```
typedef struct _trdata_packet
{
int rtAddr;         /* Remote Terminal Address           */
int trBit;          /* Transmit/Receive Bit              */
int subAddr;        /* Remote Terminal Sub-Address       */
int nWords;         /* Number of words to send or receive*/
uword data;         /* Pointer to data                   */
boolean send;       /* 1=Immediate, 0=Compile            */
COMBLOCK **cb;      /* Command Block pointer from Driver */
} TRDATA_PACKET;
```

int ms1553_bc_tr_same(TRDATA_PACKET *pData)

The following error codes are possible:

COMMAND_BLOCK_POOL_EMPTY
DATA_BLOCK_POOL_EMPTY

ms1553_bc_tr_mode_code

This command transmits a mode code and uses a unique data word buffer if it is a mode-code- with-data-word. If the mode code is between 16 and 31 (mode-code-with-data-word), then the associated data word pointed to by "data" will be transferred also.

This command requires a pointer to the following data structure:

TRMODE-PACKET

```
typedef struct _trmode_packet
{
int rtAddr;    /* Remote Terminal Address           */
int trBit;     /* Transmit/Receive Bit              */
int mcode;     /* Mode Code                         */
uword *data;   /* Pointer to data                   */
boolean send;  /* 1=Immediate, 0=Compile            */
COMBLOCK **cb; /* Command Block pointer from driver */
} TRMODE_PACKET;
```

int ms1553_bc_tr_mode_code(TRMODE_PACKET *pMode)

The following error codes are possible:

COMMAND_BLOCK_POOL_EMPTY
DATA_BLOCK_POOL_EMPTY
ILLEGAL_MODE_CODE

ms1553_bc_tr_mode_same

This command, as the others which end in "_same", is for the second through Nth repetition of a mode code command in a frame where the data is to be shared. This will use the same data word on the board as well as the final application destination. Obviously, this is for mode-code-with-data-word type commands.

This command requires a pointer to the following data structure:

TRMODE-PACKET

```
typedef struct _trmode_packet
{
int rtAddr;        /* Remote Terminal Address           */
int trBit;         /* Transmit/Receive Bit              */
int mcode;         /* Mode Code                         */
uword *data;       /* Pointer to data                   */
boolean send;      /* 1=Immediate, 0=Compile            */
COMBLOCK **cb;     /* Command Block pointer from river  */
} TRMODE_PACKET;
```

int ms1553_bc_tr_mode_same(TRMODE_PACKET *pMode)

The following error codes are possible:

COMMAND_BLOCK_POOL_EMPTY
DATA_BLOCK_POOL_EMPTY
ILLEGAL_MODE_CODE

ms1553_bc_rtrt

This command sets up a command block for a remote terminal to remote terminal transmission. The data flows from source RT to destination RT with an additional destination in the BC. The BC will therefore monitor the transmission and the data will be placed in the array pointed to by "data" associated with the command whose "cb" pointer is specified.

This command requires a pointer to the following data structure:

RTRT-PACKET

```
    typedef struct _rtrt_packet
    {
    int SrcAddr;          /* Remote Terminal Address SOURCE     */
    int SrcSubAddr;       /* Remote Terminal Subaddress SOURCE  */
    int DstAddr;          /* Remote Terminal Address DEST.      */
    int DstSubAddr;       /* Remote Terminal Subaddress DEST.   */
    int nWords;           /* Number of words to send or receiver*/
    uword *data;          /* Pointer to data                    */
    boolean send;         /* 1=Immediate, 0=Compile             */
    COMBLOCK **cb;        /* Command Block pointer from driver  */
    } RTRT_PACKET;
```

int ms1553_bc_rtrt(RTRT_PACKET *pRT)

The following error codes are possible:

COMMAND_BLOCK_POOL_EMPTY

ms1553_bc_rtrt_same

This command sets up a command block for a remote terminal to remote terminal transmission. The data flows from source RT to destination RT with an additional destination in the BC. The BC will therefore monitor the transmission and the data will be placed in the array pointed to by "data" associated with the command whose "cb" pointer is specified. This version of the command will also use the same data buffer on the board, thereby saving valuable board space. This is important to note when you are compiling frames in which the RT to RT commands with monitoring must be repeated many times.

This command requires a pointer to the following data structure:

RTRT-PACKET

```
typedef struct _rtrt_packet
{
int SrcAddr;         /* Remote Terminal Address SOURCE      */
int SrcSubAddr;      /* Remote Terminal Subaddress SOURCE   */
int DstAddr;         /* Remote Terminal Address DESTINATION */
int DstSubAddr;      /* Remote Terminal Subaddress DEST.    */
int nWords;          /* Number of words to send or receive  */
uword *data;         /* Pointer to data                     */
boolean send;        /* 1=Immediate, 0=Compile              */
COMBLOCK **cb;       /* Command Block pointer from driver   */
} RTRT_PACKET;
```

int ms1553_bc_rtrt_same(RTRT_PACKET *pRT)

The following error codes are possible:

COMMAND_BLOCK_POOL_EMPTY

ms1553_bc_delay

When a delay is required in a frame longer than what can be specified for an individual message, the required delay can be inserted in the frame with this command. This delay is achieved by inserting the proper number of skipped messages provided there is ample memory.

This command must be passed the following item:

ULONG usec

Delay in microseconds. Resolution is 64 microseconds.

int ms1553_bc_delay(ulong usec);

The following error codes are possible:

COMMAND_BLOCK_POOL_EMPTY

ms1553_bc_skip

If “skip” is TRUE, set the skip flag for the command block whose address is passed. This will force the SUMMIT to skip the 1553 command and delay based on timer value. This means the message is skipped or it is now “hidden”. The process is reversed when “skip” is FALSE.

This command requires a pointer to the following data structure:

SKIP-PACKET

```
typedef struct _rtrt_packet
{
COMBLOCK *cb;         /* Command Block pointer          */
boolean skip;         /* 1=SKIP, 0=NO SKIP              */
} SKIP_PACKET;
```

COMBLOCK *cb;
This is the address for the command block which will have the skip flag set or cleared.

BOOLEAN skip;
If TRUE, set the flag to skip the command. If FALSE, clear the skip flag and the message will execute.

int ms1553_bc_skip(SKIP_PACKET *p);

ms1553_bc_1noskip

This command is for those cases where a message is to appear only once and disappear again. For example, if a series of "hidden" messages for various cockpit displays are in a frame, the BC_CHGDATA command could be used to update data for a display and then BC_1NOSKIP could be executed to update that display only once and disappear again.The interrupt routine will handle the changing of the bits after the command has executed once.

This command must be passed the following item:

COMBLOCK *ptr

int ms1553_bc_1noskip(COMBLOCK *ptr)

ms1553_bc_mtoggle

This command is for those circumstances when you have two messages for a device and either the first or the second is to be executed, but not both. The function is effectively a "momentary toggle" in which the first message will disappear for one cycle and the second message will appear for one cycle. After which the first message will be continually sent and the second skipped or "hidden". The way the commands are generated must be followed exactly.

To use the function, generate two messages contiguously to one another.

The first message will be the "normally closed" function of the toggle. For this message use a zero delay term. The second message is then generated, which is the "normally open" or "hidden message". After generating the second message, set the skip flag for it. When you wish to execute the function, you will pass the address pointer for the first message to the procedure.

This command must be passed the following item:

COMBLOCK *ptr

int ms1553_bc_mtoggle(COMBLOCK *ptr)

ms1553_bc_chgdata

This routine is used to update data for an RT receive command that is compiled on the board. As discussed in section on frames, this is the way to update data that must be sent to RT's. Since the data must exist in the dual access RAM on the board for the SUMMIT to send it on the bus, this provides the pathway to refresh it. For this routine to work properly you must have more than one command in a frame that does not refer to this data buffer. It may be a skipped command, a command with a delay of greater than 4.1 milliseconds, the use of the BC_DELAY command, etc. This is because the driver must ensure no command is transmitting the buffer at the current time so that moving the data will not cause phasing problems.

This command requires a pointer to the following data structure:

CHGDATA-PACKET

```
typedef struct _chgdata_packet
{
COMBLOCK **cb;  /* Command Block pointer returned by Driver  */
uword *data;    /* Pointer to Data                           */
} CHGDATA_PACKET;
```

ms1553_bc_end_of_frame

This is the command used to terminate a frame. You are given the choice of looping back to a point in the frame continually, if "loop" is TRUE, or if "loop" is FALSE, the frame will just execute once and stop. It is important to note here that the address to loop back to does not have to be the first address in the frame. If there is some initial data that must be sent to your system, which only needs to be sent once, it can be placed at the beginning of the frame and the loop point can be following those commands. Following this command, you can execute another BC_GET_ADDR command and begin compiling another frame. When all frames are compiled, you execute BC_BUILD_FRAME to indicate the frames are complete.

```
int ms1553_bc_end_of_frame( COMBLOCK *first, boolean loop )
COMBLOCK *first;       /* Command Block pointer returned by Driver   */
boolean loop;          /* 1=Loop, 0=No Loop                          */
```

Possible Returns

FRAME_BUILD_NOT_IN_PROGRESS
NO_COMMANDS_IN_FRAME

ms1553_bc_exec_frame

This command will instruct the SUMMIT to execute the frame whose address is passed. If “wait” is TRUE, the frame is checked to assure it is not an infinite loop. If it is not, the frame will execute once, after which this routine will return to the calling program. If it is an infinite loop, it will NOT execute the frame, but will return an error code. Normally, this would be used for infinite looping frames and the “wait” parameter is always FALSE. Once a frame has been started, the BC_HALT_FRAME command is used to halt the execution.

```
int ms1553_bc_exec_frame( COMBLOCK *p, boolean wait )

COMBLOCK *p;            /* Command Block pointer                         */
boolean wait;           /* 1=Wait for frame to complete, 0=No wait       */
```

Possible Returns

INFINITE_LOOP_AND_WAIT

ms1553_bc_halt_frame

If you wish to terminate execution of a frame, this procedure must be called with the start address of the frame. The routine will assure that execution at the start of the frame. It always stops at the beginning of a frame, it does not just halt the SUMMIT at the current location. This was done to assure an orderly halt with complete frames being executed.

```
int ms1553_bc_halt_frame( COMBLOCK *p )
```

RETURNS

None.

6. 1553 Remote Terminal - user documentation

6.1 MS1553RT Driver functional description

Status Structure

The library routines communicate status and information to the application through the STAT_INFO structure:

STAT-INFO

```
typedef struct status_info
{
volatile        booleancompleted; /* Done flag                 */
volatile uword  runtime_error;    /* run time error code       */
volatile uword  msg_status;       /* message status RT         */
volatile ulong  msg_count;        /* number of messages done   */
volatile ulong  error_count;      /* number with errors        */
volatile uword  time_tag;         /* time tag                  */
} STAT_INFO;
```

The **completed** field indicates that the I/O request has completed. This flag will be set even if the message transaction completed with errors. Examine the next field, **runtime_error** for the error cause. The **msg_status** field is the message status word taken from the RT subaddress where the message was transacted. This message status word contains important information. Refer below to a detail of the message status field:

BITS 15 - 11	10	09	08	07	6-5	04	03	02	01	00
WC (4:0)	N/A	CHA/B	RTRT	ME	N/A	ILL	TO	OVR	PRTY	MAN

WC (4:0)	Word Count
N/A	Not Applicable
CHA/B	1 = Message Received on Channel A,0 = Channel B
RTRT	Command was an RT to RT transfer
ME	Message Error
N/A	Not Applicable
ILL	Illegal Command Received
TO	Time-Out Error
OVR	Overrun Error
PRTY	Parity Error
MAN	Manchester Error

The **msg_count** and **error_count** fields keep track of the number of messages received with out errors and those with errors. The **time_tag** field is the value of the hardware time tag counter at the time of message transaction. The time tag resolution is 64 **m**s per bit.

6.2 Library routines

ms1553_rt_init

int ms1553_rt_init(RT_INIT *pInit)

RT_INIT *pInit
Pointer to a RT_INIT structure used to initialise the IP1553 hardware subaddresses for RT operation. This function should be called prior to using any other library routines. The initialisation structure for preparing board hardware and memory map is as follows:

RT-INIT
```
typedef struct rt_init
{
uword      *RT_MemAddr;      /* Address of Shared Memory        */
uword      *RT_IOAddr;       /* Address of I/O registers        */
uword      RT_IntVector;     /* Interrupt Vector Number         */
uword      RT_IntLevel;      /* Interrupt Level Number          */
uword      RT_Control;       /* SRT Control Word Preferences    */
boolean    RT_MS1553A;       /* 1 = 1553A Standard, 0 = 1553B   */
ulong      IRx_Mask;         /* Illegal Receive Mask            */
ulong      ITx_Mask;         /* Illegal Transmit Mask           */
ulong      IBRx_Mask;        /* Illegal Broadcast Receive       */
ulong      IBTx_Mask;        /* Illegal Broadcast Transmit      */
ulong      IMRx_Mask;        /* Illegal Mode Code Receive       */
ulong      IMTx_Mask;        /* Illegal Mode Code Transmit      */
ulong      IMBRx_Mask;       /* Illegal Mode Code B'cast RCV    */
ulong      IMBTx_Mask;       /* Illegal Mode Code B'cast XMT    */
uword      RT_IrqMask;       /* SRT High Priority InterruptMask */
    } RT_INIT;
```

RT_MemAddr
This field should contain the physical address of the IP1553 shared memory.

RT_IOAddr
This field should contain the physical address of the IP1553 I/O registers.

RT_IntVector
This field should contain the interrupt vector number to be furnished by the IP1553 during an interrupt acknowledge cycle (IACK) cycle.

RT_IntLevel
This field should contain the interrupt level. The interrupt level is not currently used by the driver routines, but may be required to install or hook and interrupt on some operating systems.

RT_Control
This field shall be used to specify options for the Remote Terminal Control register. These options can be combined to form a bit mask. The following options are currently supported:

```
RT_XMTSW     Enables transmit last Status word mode code (1553A
             only)
RT_DYNBC     Enables Dynamic Bus Ctrl Acceptance mode code
RT_BCEN      Enables Broadcast operation; subaddress 31 is
             broadcast
RT_ETCE      Enables External Time Tag
RT_CHBEN     Enables RT Channel B
RT_CHAEN     Enables RT Channel A
```

RT_Address
This field shall be used to specify the address for the RT on the 1553B bus. This value will be used to write to the remote Terminal address register. Some hardware has jumper blocks which can prevent this address from being changed by the software. Be sure and check state of the **LOCK** pin on the board for the correct level. When **LOCK** is asserted, the IP1553 mode and address cannot be changed by software.

RT_MS1553A
This field shall specify the use of 1553A or 1553B standards. 1 = 1553A, 0 = 1553B.

IRx_Mask
This 32 bit field specifies an illegal receive subaddress mask. Bits 0 - 31 correspond to subaddresses 0 - 31. For each bit that is a 1, the subaddress is illegal. For each bit that is a 0, the subaddress is legal.

ITx_Mask
This 32 bit field specifies an illegal transmit subaddress mask. Bits 0 - 31 correspond to subaddresses 0 - 31. For each bit that is a 1, the subaddress is illegal. For each bit that is a 0, the subaddress is legal.

IBRx_Mask
This 32 bit field specifies an illegal broadcast receive subaddress mask. Bits 0 -31 correspond to subaddresses 0 - 31. For each bit that is a 1, the subaddress is illegal. For each bit that is a 0, the subaddress is legal.

IBTx_Mask
This 32 bit field specifies an illegal broadcast transmit subaddress mask. Bits 0 -31 correspond to subaddresses 0 - 31. For each bit that is a 1, the subaddress is illegal. For each bit that is a 0, the subaddress is legal.

IMRx_Mask
This 32 bit field specifies an illegal mode code receive subaddress mask. Bits 0 -31 correspond to subaddresses 0 - 31. For each bit that is a 1, the subaddress is illegal. For each bit that is a 0, the subaddress is legal.

IMTx_Mask
This 32 bit field specifies an illegal mode code transmit subaddress mask. Bits 0 - 31 correspond to subaddresses 0 - 31. For each bit that is a 1, the subaddress is illegal. For each bit that is a 0, the subaddress is legal.

IMBRx_Mask
This 32 bit field specifies an illegal broadcast mode code receive subaddress mask. Bits 0 - 31 correspond to subaddresses 0 - 31. For each bit that is a 1, the subaddress is illegal. For each bit that is a 0, the subaddress is legal.

IMBTx_Mask
This 32 bit field specifies an illegal broadcast mode code transmit subaddress mask. Bits 0 - 31 correspond to subaddresses 0 - 31. For each bit that is a 1, the subaddress is illegal. For each bit that is a 0, the subaddress is legal.

RT_IrqMask
This field shall determine which high priority interrupts (YF_INT) are enabled. The following high priority interrupts are currently defined:

```
        RT_BITF     Enable BIT Fail Interrupt
        RT_TAPF     Enables Terminal Address Parity Fail Interrupt
        RT_WRAPF    Enables Wrap Fail Interrupt
        RT_DMAF     Enables DMA Fail Interrupt
```

ms1553_rt_tr_data

This routine is used to transmit and receive data from the specified subaddress. The cnt parameter is only used during transmit because the Bus Controller determines the word count for received messages. Received subaddress data is copied by the interrupt service routine to the area pointed to by **pData**. The **msg_status** field, in the STAT_INFO structure, contains the word count. Transmit data are copied directly to the IP1553 subaddress data area. Set the W**ait** parameter **TRUE** to wait for the interrupt service routine to process the I/O request before returning control to the application. Note: if **Wait** is set to FALSE the STAT_INFO structure must be checked later to determine when or if the message transaction completed. This function also works in conjunction with the **ms1553_rt_set_timeout,** if **Wait** is TRUE, so the application does not loop forever waiting for a subaddress message to arrive.

int ms1553_rt_tr_data (boolean TrBit, uword SubAddr, uword WordCount, uword *pData, STAT_INFO *pInfo, boolean Wait)

boolean TrBit	Transmit/Receive bit (TBIT, RBIT)
uword SubAddr	The subaddress 1-30 transacting the message
uword WordCount	Number of data words to transmit/receive
uword *pData	Pointer to application data buffer
STAT_INFO *pInfo	Pointer to status structure
boolean Wait	If TRUE wait for completion signal by driver

ms1553_rt_tr_mcdata

This function will write the data word pointed to by **pData** to the specified mode code area if **TrBit** = TBIT. The IP1553 will then transmit this word when it receives a mode code with data command. If **TrBit** = RBIT the mode code data word is copied to **pData**.

int ms1553_rt_tr_mcdata(uword TrBit, uword ModeCode, uword *pData, STAT_INFO *pInfo, boolean Wait)

uword TrBit	Transmit/Receive (TBIT, RBIT)
uword ModeCode	Mode code to transmit/receive
uword *pData	Pointer to data area for mode codes with data
STAT_INFO *pInfo	Pointer to STAT_INFO structure
boolean Wait	If TRUE wait for completion signal by driver

ms1553_rt_set_timeout

This function allows the application to install a time-out such that the functions **ms1553_rt_tr_data** and **ms1553_rt_tr_mcdata** will time-out if data is not received or transmitted within the time specified by the parameter **milliseconds**.

void ms1553_rt_set_timeout(ulong milliseconds)

ulong milliseconds	Number of milliseconds to wait for I/O

ms1553_exit

This function should be called prior to returning to the operating system. Any pending IP1553 interrupts are cleared and the interrupt mask register is reset. The user should remove or unhook any hardware interrupts that were installed.

void ms1553_exit(void)

ms1553_rt_status_word

This function is used to specify a value for the outgoing MIL-STD-1553 status word.

FUNC void ms1553_rt_status_word(uword status)

uword status - Bit mask of the Remote Terminal flags

The following values can be combined to form a bit mask for the status word:

RT_SWB_TF	Terminal Flag
RT_SWB_SSYSF	Subsystem Flag
RT_SWB_BUSY	Busy Bit
RT_SWB_SRQ	Service Request Bit
RT_SWB_INS	Instrumentation Bit
RT_SWB_IMCLR	Immediate Clear Function

ms1553_rt_chg_adr

This function will program the RT address register with the variable new_rt_adr. This function call will replace any address that was programmed by the ms1553_rt_init() function. **Note: external hardware jumpers may prevent this address from being accepted by the IP1553.**

void ms1553_rt_chg_adr(uword new_rt_adr)

```
uword new_rt_adr         RT address on 1553B to set
```

ms1553_reset

This function will perform a software reset by writing to the IP1553 control register. This only affects the internal encoder/decoders.

void ms1553_rt_reset(void)

ms1553_set_interrupt

This function is used to install user defined interrupt handlers. The driver processes two types of interrupts. The first type is a high priority interrupt which is asserted via the YF_INT pin. The second type is a normal message interrupt asserted via the MSG_INT pin. The application can install a function to handle each type of interrupt. If a function pointer is not NULL, the function will be called by the interrupt service routine. The YF_INT handler function shall be passed a copy of the interrupt status word. The MSG_INT handler function shall be passed the Interrupt Identification (IIW) and Interrupt Address (IAW) Words taken from the Interrupt Log List.

int ms1553_set_interrupt(void (*function_pntr)(void), uword type)

void (*function_pntr)()	Pointer to user interrupt function
uword type	1=YF_INT function, 0=MSG_INT function

ms1553_rt_bit_test

This function will command the IP1553 to perform its BIT (Bilt In Test).

int ms1553_rt_bit_test(void)

This function can return the following error codes:

BIT_FAIL
BIT_TIMEOUT