NOTICE

BMC Communications Corp. reserves the right to change the product described in this document as well as the document itself at any time and without notice.

DISCLAIMER

BMC COMMUNICATIONS CORP. MAKES NO WARRANTIES, EITHER EXPRESSED OR IMPLIED, WITH RESPECT TO THIS DOCUMENT OR WITH RESPECT TO THE PRODUCT DESCRIBED IN THIS MANUAL, ITS QUALITY, PERFORMANCE, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT SHALL BMC COMMUNICATIONS CORP. BE LIABLE FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT IN THE PRODUCT.

Rev. 3.1 June 15, 2008

Table Of Contents

GENERAL BOARD FUNCTIONS

(GSR)

The functions read and write into first control block defined as gsr1553 in the CRAM1553.H

General Board Functions (GSR)

cram_set_board

DESCRIPTION

Select active board's address

USAGE

#include <cram.h>

int cram_set_board(WORD address);

address	**the base segment of the board address**

REMARKS

This routine tries to find a CRAM-1553 board at the specified address. If it finds one, it selects it as the currently selected board (global variable _CRAM_BOARD). All subsequent functions and commands will refer to the board at this address.

RETURN VALUE

CRAM_SUCCESS	**Operation successful**
CRAM_NO_BOARD	**No board found at specified address.**

EXAMPLE

```
        if (cram_set_board(OxD000) =CRAM_SUCCESS)
                printf(”Board not presentn”);
else{
        cram_sys_id(&min, &maj);
        printf (“CRAM-1553 Version %d.%d at D000:0000”,min,maj);
}
```

SEE ALSO

cram_sys_id

cram_sys_id

DESCRIPTION

Get system ID and version number

USAGE

```
#include <cram.h>

int cram_sys_id(int *major int *minor);
```

major	major version number
minor	minor version number

REMARKS

This routine reads the system ID string from the common memory. If it recognizes the CRAM-1553 signature, it reads the version number and loads it into the supplied integer pointers *major* and *minor*.

RETURN VALUE

CRAM_SUCCESS	Operation successful
CRAM_NO_BOARD	No board signature found

EXAMPLE

```
if (cram_set_board(OxD000) =CRAM_SUCCESS)
        printf(”Board not present\n”);
else{
        cram_sys_id(&min ,&maj);
        printf (“CRAM-l 553 Version %d.%d at 0000:0000”,min,maj);
}
```

SEE ALSO

cram_set_board

cram_select_channel

DESCRIPTION

Selects either Channel A or B for Transmit/Receive operation.

USAGE

```
#include <cram.h>

int cram_select_channel (BYTE channel)
```

channel either A (‘0’) or B (‘1’)

REMARKS

The CRAM system can transmit on one of two channels at any time (but not both simultaneously). This function selects either A or B. Note that the board only looks for the setting of the channel register in the GSR during changes in the board operating mode. Therefore, this function calls the cram_seLmode function to againset the board to the current operating mode obtained from the cram_get_mode function.

RETURN VALUE

CRAM_SUCCESS successful

CRAM_INV_CHANNEL invalid channel

EXAMPLE

```
int result;

result = cram_bm_select_channel (0);
```

SEE ALSO

cram_bc_exec_instruction

cram_class

DESCRIPTION

Sets the CRAM operational mode CLASS A or B.

USAGE

#include <cram.h>

int cram_class (WORD *class*);

class	**value 1 for CLASS A,** **value 0 for CLASS B (default)**

REMARKS

The operational class mode pertains to a variance **in the data word structure. Additional functions are added to CLASS Bmode to keep** accommodate **for MIL compliance.**

RETURN VALUE

CRAM_SUCCESS	**successful**
CRAM_INV_CRAMD	**invalid argument**

cram_reset

DESCRIPTION

This command creates a soft reset and reinitializes the board.

USAGE

#include <cram.h>

int cram_reset (BYTE *speed*);

speed	**Reset Speed: value 2= FAST,** **value 1 = SLOW**

REMARKS

This routine sets the *Soft_Reset* field in the SPECIAL_CTL register to one of the two integral values defined in the header file *CRAM.H*.

The value entered for speed will determine how the board is reset. The reset is desired the DPM will be cleared to its power on state

The *Soft_Reset* field in the SPECIAL_CTL register is the last register in the DPM to be cleared. This register can be used as a status flag to confirm when the board has completed its reset cycle.

RETURN VALUE

CRAM_SUCCESS	**successful**

cram_set_mode

DESCRIPTION

Sets the board operating mode to IDLE, BC, RT, BM, BC/BM, BC/RT, or RT/BM.

USAGE

```
#include <cram.h>

int cram_set_mode (INTEGER mode);
```

mode	Board operating mode: one of CRAM_IDLE, CRAM_BC, CRAM_RT, CRAM_BM, CRAM_BCBM, CRAM_BCRT, or CRAM_RTBM

REMARKS

This routine sets the Set Mode field in the GSR to one of the six integral values defined in the header file CRAM.H. For further information on the characteristics and operation of each mode please see Chapter 4 of the User's Manual. Depending on the model you purchased, some operating modes may not be supported by your board.

The procedure for setting the board mode is that the user enters the desired mode into the Mode Set field of the GSR which is read by the board. If that mode is valid for your board, the board will confirm by returning the corresponding value into the Mode field of the GSR, and also by returning CRAM_SUCCESS in the Result Code field of the GSR. This function checks for these confirmations and gives a Return Value indicating a success or not.

RETURN VALUE

CRAM_SUCCESS	successful
CRAM_INV_MODE	invalid mode value

EXAMPLE

```
/ Set CRAM board to operate as a BC */
int result;
result = cram_set_mode(CRAM_BC);
```

cram_rx_receive

DESCRIPTION

Checks whether the receiver on the board has received any data.

USAGE

```
#include <cram.h>

int cram_rx_receive (void);
```

REMARKS

This routine checks to see if the board has received any data. The function works by examining the first bit of the board's rx_tx_indicator register. If the bit is set the function returns seccess, then clears the bit.

RETURN VALUE

YES	**Data received**
NO	**No data received**

EXAMPLE

```
if (cram_rx_receive ())
{
        …
}
```

SEE ALSO

cram_tx_transmit

cram tx transmit

DESCRIPTION

Checks whether the transmitter on the board has transmitted data.

USAGE

```
#include <cram.h>

int cram_tx_transmit ();
```

REMARKS

This routine checks if the board has transmitted data. The function works by examining the second bit of the board's *rx_tx_indicator* register. If the bit is set the function returns seccess, then clears the bit.

RETURN VALUE

YES	**Data transmitted**
NO	**No data transmitted**

EXAMPLE

```
if ( cram_tx_transmit 0)
{
        …
}
```

SEE ALSO

cram_rx_receive

cram_tx_count

DESCRIPTION

Returns value of GSR transmit frame counter.

USAGE

```
#include <cram.h>

WORD cram_txcount(void);
```

REMARKS

The transmit frame counter is a board-mode independent counter in the GSR (Global System Registers) which increments every time a new frame is transmitted until it reaches the maximum value that can be stored (65536) or until the user resets it. This function returns the current value of the counter. See Chapter 4 for further information.

RETURN VALUE

Current value of counter.

EXAMPLE

```
        WORD count;

count = cram_tx_countO;
```

SEE ALSO

cram_rx_count
cram_dr_tx_count
cram_clr_rx_count

cram_rx_count

DESCRIPTION

Returns value of GSR receive frame counter.

USAGE

```
#include <cram.h>

WORD cram_rx_count(void);
```

REMARKS

The receive frame counter is a board-mode independent counter in the GSR (Global System Registers) which increments every time a new frame is received until it reaches the maximum value that can be stored (65536), or until the user resets it. This function returns the current value of the counter. See Chapter 4 for further information.

RETURN VALUE

Current value of counter.

EXAMPLE

```
        WORD count;

count = cram_rx_countO;
```

SEE ALSO

cram_tx_count
cram_clr_rx_count
cram_dr_tx_count

cram_tx_complete

DESCRIPTION

Checks whether a newly completed transmission has occured.

USAGE

```
#include <cram.h>

int cram_tx_complete (void);
```

REMARKS

Checks the TX bit in the rx_tx_indicator register in the GSR, and then clears it if found to be set. This bit is set every time a completed transmission occurs.

RETURN VALUE

YES (‘1’)

NO (‘0’)

EXAMPLE

```
int result;

result = cram_tx_complete;
```

SEE ALSO

cram_rx_complete

cram_rx_complete

DESCRIPTION

Checks whether a newly completed transmission has occured.

USAGE

```
#include <cram.h>

int cram_rx_complete (void);
```

REMARKS

Checks the RX bit in the rx_tx_indicator register in the GSR, and then clears it if found to be set. This bit is set every time a completed reception occurs.

RETURN VALUE

YES (‘1’)

NO (‘0’)

EXAMPLE

```
int result;

result = cram_rx_complete;
```

SEE ALSO

cram_tx_complete

cram_clr_tx_count

DESCRIPTION

Clears value of GSR transmit frame counter.

USAGE

```
#include <cram.h>

int cram_clr_tx_count(void);
```

REMARKS

The transmit frame counter is a board-mode independent counter in the GSR (Global System Registers) which increments every time a new frame is transmitted until it reaches the maximum value that can be stored (65536) or until the user resets it. This function clears (sets to 0) the current value of the counter. See Chapter 4 for further information.

RETURN VALUE

CRAM_SUCCESS successful

EXAMPLE

```
        int result;

result = cram_clr_tx_countO;
```

SEE ALSO

cram_tx_count
cram_rx_count
cram_clr_rx_count

cram_clr_rx_count

DESCRIPTION

Clears value of GSR receive frame counter.

USAGE

```
#include <cram.h>

int cram_rx_count(void);
```

REMARKS

The receive frame counter is a board-mode independent counter in the GSR (Global System Registers) which increments every time a new frame is received until it reaches the maximum value that can be stored (65536), or until the user resets it. This function clears (sets to 0) the counter. See Chapter 4 for further information.

RETURN VALUE

CRAM_SUCCESS successful

EXAMPLE

```
WORD count;

count = cram_rx_countO;
```

SEE ALSO

cram_rx_count
cram_tx_count
cram_clr_tx_count

cram_ei_rxcomplete

DESCRIPTION

Enables interrupt on receive complete.

USAGE

```
#include <cram.h>

int cram_ei_rxcomplete (void);
```

REMARKS

This function sets the appropriate bit in the rx_tx_interrupt register of the GSR which will cause the board to issue an IRQ (interrupt request) when a complete frame has been received.

RETURN VALUE

CRAM_SUCCESS successful

SEE ALSO

cram_mrt_ei_datarcvd
cram_mrt_di_datarcvd
cram_mrt_ei_datarcvd_all
cram_mrt_di_datarcvd_all
cram_di_rxcomplete

cram_ei_tx complete

DESCRIPTION

Enables interrupt on transmit complete.

USAGE

#include <cram.h>

int cram_ei_txcomplete (void);

REMARKS

This function sets the appropriate bit in the rx_tx_interrupt register of the GSR which will cause the board to issue an IRQ (interrupt request) when a complete frame has been transmitted.

RETURN VALUE

CRAM_SUCCESS successful

SEE ALSO

cram_mrt_ei_datarcvd
cram_mrt_di_datarcvd
cram_mrt_ei_datarcvd_all
cram_mrt_di_datarcvd_all
cram_di_txcomplete

cram_di_rxcomplete

DESCRIPTION

Disables interrupt on receive complete.

USAGE

```
#include <cram.h>

int cram_di_rxcomplete (void);
```

REMARKS

This function resets (sets to '0') the appropriate bit in the rx_tx_interrupt register of the GSR which will cause the board to cease sending an IRQ (interrupt request) when a complete frame has been received.

RETURN VALUE

CRAM_SUCCESS successful

SEE ALSO

cram_mrt_ei_datarcvd
cram_mrt_di_datarcvd
cram_mrt_ei_datarcvd_all
cram_mrt_di_datarcvd_all
cram_ei_rxcomplete

cram_di_txcomplete

DESCRIPTION

Disables interrupt on transmit complete.

USAGE

```
#include <cram.h>

int cram_di_txcomplete (void);
```

REMARKS

This function resets (sets to ‘0’) the appropriate bit in the rx_tx_interrupt register of the GSR which will cause the board to cease sending an IRQ (interrupt request) when a complete frame has been transmitted.

RETURN VALUE

CRAM_SUCCESS successful

SEE ALSO

cram_mrt_ei_datarcvd
cram_mrt_di_datarcvd
cram_mrt_ei_datarcvd_all
cram_mrt_di_datarcvd_all
cram_ei_txcomplete

cram_get_mode

DESCRIPTION

Returns the board operating mode (IDLE, BC, RT, BM, BC/BM, BC/RT, or RT/BM).

USAGE

```
#include <cram.h>

BYTE cram_set_mode (void);
```

REMARKS

This routine reads the current value of the mode field in the GSR. For further information on the characteristics and operation of each mode please see Chapter 4 of the User's Manual. Depending on the model you purchased, some operating modes may not be supported by your board.

RETURN VALUE

MODE current board operating mode, one of:

CRAM_IDLE

CRAM_BC
CRAM_RT
CRAM_BM
CRAM_BCBM
CRAM_BC RT
CRAM_RTBM

EXAMPLE

```
/* Check current CRAM operating mode *1 int result;

result = cram_get_modeQ;
```

cram_writebuf

DESCRIPTION

Write a section of CRAM memory

USAGE

```
#include <cram.h>

mt cram_writebuf( WORD offset, void *src WORD byte_count);
```

offset	destination address (in board)
src	current address of data (in host)
byte_count	number of bytes to be written

REMARKS

This function is used to transfer a block of data into CRAM board Dual-Port memory. The most typical use is to load outgoing data into the board before issuing a transmit command.

RETURN VALUE

CRAM_SUCCESS transfer complete

EXAMPLE

see cram_tx_cramd

SEE ALSO

cram_readbuf

cram readbuf

DESCRIPTION

Read a section of CRAM memory

USAGE

#include <cram.h>

mt cram_readbuf(void *dst WORD offset, WORD byte_count);

dest	destination address (in host)
offset	current address of data (in board)
byte_count	number of bytes to be read

REMARKS

This function is used to transfer a block of data from CRAM board Dual-Port memory into program memory. The most typical use is to transfer received data from the board after a reception.

RETURN VALUE

CRAM_SUCCESS transfer complete

EXAMPLE

See cram_rx_cramd example

SEE ALSO

cram_writebuf

Bus Controller Mode Functions

(BC)

The functions read and write into first control block defined as “bc_ctl” in the CRAM1553.H

BC Board Functions (bc_ctl)

// Command Register functions

cram_com_rem ..……………………………….

cram_com_sam ……………………………………

cram_com_count ………………………………..

cram_com_tr ……………………………………

cram_com_rt_rec_count…………………..

cram_com_rt_rec_rem ……….…………..

cram_com_rt_rec_sam..…………………………….

cram_com_rt_rec_tr……………………………….

cram_com_rt_tran_count..…………………………….

cram_com_rt_tran_rem……………………………

cram_com_rt_tran_sam ..…………………………….

cram_com_rt_tran_tr …………………………………..

cram_com_rem ..……………………………….

cram_com_sam ……………………………………

cram_com_rem ..……………………………….

cram_com_sam ……………………………………

cram_com_rem ..……………………………….

cram_com_sam ……………………………………

……………………..SET COMMANDS…………………………

cram_bc_set_err_parity…………………………………

cram_ bc_set_err_sync..…………………………………

cram_ bc_set_mm_time_tag……………………………….

cram_ bc_set_mm_word_count…………………………..

cram_ bc_set_mm_command……………………………..

cram_ bc_set_mm_com_rem……………………………….

cram_ bc_set_mm_mm_com_tr…………………………..

cram_ bc_set_mm_mm_com_sam……………………………..

cram_ bc_set_mm_com_count………………………………..

cram_ bc_set_mm_data_word……………………………

cram_ bc_set_mm_command……………………………..

cram_ bc_set_mm_data_word…………………………………

cram_ bc_set_noerror……………………………

cram_ bc_set_rx_start_address………………………………

……………………………RESET COMMAND ……………

………………………….. GET COMMANDS………………..

cram_com_rem

DESCRIPTION

Sets Remote Terminal Address field in Command word.

USAGE

#include <cram.h>

int cram_com_rem (WORD *remote_add*);

remote_add	**the remote terminal address (1 to 31)**

REMARKS

This routine sets the remote terminal address field in the command word sent by the board in BC mode.

RETURN VALUE

CRAM_SUCCESS	**successful**
CRAM_INV_PARAM	**invalid remote terminal address**

EXAMPLE

```
/* Set remote address 300 in command word */

cram_com_rem (300);
```

SEE ALSO

cram_com_tr
cram_com_sam
cram_com_count

cram_com_tr

DESCRIPTION

Sets Transmit/Receive bit in Command word.

USAGE

#include <cram.h>

int cram_com_tr (char *tr_bit*);

tr_bit	**Transmit/Receive bit ("0" or "1")**

REMARKS

This routine sets direction of message transmission ("1": RT will transmit; "0": RT will receive) in the command word sent by the BC.

RETURN VALUE

CRAM_SUCCESS	**successful**
CRAM_INV_PARAM	**invalid or unconfigured parameter (>1)**

EXAMPLE

```
/* Set transmit/receive bit in command word to "1 ".

cram_com_tr (1);
```

SEE ALSO

cram_com_rem
cram_com_sam
cram_com_count

cram_com_sam

DESCRIPTION

Sets Subaddress/Mode field in Command word.

USAGE

#include <cram.h>

int cram_com_sam (int *subaddr*);

subadd	**subaddress**

REMARKS

This routine selects the specific subaddress source/destination within a remote terminal for use in the current data transfer operation. If this value is "0" or "31", the next field (word count/mode code) will contain a mode command.

RETURN VALUE

CRAM_SUCCESS	**Successful**
CRAM_I NV_PARAM	**Invalid subaddress parameter**

EXAMPLE

```
/* set subadress 25 */

cram_com_sam (25);
```

SEE ALSO

cram_com_rem
cram_com_tr
cram_com_count

cram_com_count

DESCRIPTION

Sets Word Count/Mode Code field of Command word.

USAGE

```
#include <cram.h>

int cram_com_count (int count);
```

count	**number of Data words or Mode Code**

REMARKS

This routine sets the word count/mode code field in the command word to be transmitted by the BC. In a regular Command this indicates the number of data words to be transfered; in a Mode Command it contains the Mode Code. The value 32 is coded as 0 (“00000b”) in the Command word.

RETURN VALUE

CRAM_SUCCESS	**successful**
CRAM_INV_PARAM	**invalid number of data words**

EXAMPLE

```
/ Set Word Count/Mode Code field in Command word to 8 */
 cram_com_count (8);
```

SEE ALSO

cram_com_rem
cram_com_tr
cram_com_sam

cram_com_rt_rec_rem

DESCRIPTION

Sets Remote Terminal Address field in RT-RT Receive Command word.

USAGE

#include <cram.h>

int cram_com_rt_rec_rem (int *remote_add*);

remote_add **the remote terminal address (0 to 30)**

REMARKS

This routine sets the remote terminal address in the command word sent by the BC to the receiving RT in an RT-RT transfer.

RETURN VALUE

CRAM_SUCCESS successful

CRAM_INV_PARAM invalid remote terminal address

EXAMPLE

```
/* Receive terminal address = 8 */

cram_com_rt_rec_rem (8);
```

SEE ALSO

cram_com_rt_rec_sam
cram_com_rem cram_
cra rn_co m_rt_rec_count
cram_com_rt_rec_tr
cram_com_rt_tran_sam
com_rt_tran_rem
cram_com_rt_tran_count
cram_com_rt_tran_tr

cram_com_rt_tran_rem

DESCRIPTION

Sets Remote Terminal Address field in RT-RT Transmit Command word.

USAGE

```
#include <cram.h>

int cram_com_rt_tran_rem (int remote_add);
```

remote_add	**the remote terminal address (0 to 30)**

REMARKS

This routine sets the remote terminal address in the command word sent by the BC to the transmitting RT in an RT-RT transfer.

RETURN VALUE

CRAM_SUCCESS	**successful**
CRAM_INV_PARAM	**invalid remote terminal address**

EXAMPLE

```
/* Set remote terminal address field to 15 */

cram_com_rt_tran_rem (15);
```

SEE ALSO

cram_com_rt_rec_sam	**cram_com_rt_tran_sam**
cram_com_rt_rec_rem	**cram_com_rem**
cram_com_rt_tran_count	**cram_com_rt_rec_tr**
cram_com_rt_tran_tr	**cram_com_rt_rec_count**

cram_com_rt_rec_sam

DESCRIPTION

Sets Subaddress field in RT-RT Receive Command word.

USAGE

```
#include <cram.h>

int cram_com_rt_rec_sam (int subadd);
```

subadd	**internal subaddress within receiving remote terminalln RT-RT transfer.**

REMARKS

This routine sets the subaddress field in the command word sent by the BC to the receiving RT in an RT-RT transfer.

RETURN VALUE

CRAM_SUCCESS	**successful**
CRAM_INV_PARAM	**invalid subaddress**

EXAMPLE

```
/* Set Receiving RT's subaddress field to 8 */

cram_com_rt_rec_sam (8);
```

SEE ALSO

cram_com_sam	**cram_com_rt_tran_sam**
cram_com_rt_rec_rem	**cram_com_rt_tran_rem**
cram_com_rt_rec_count	**cram_com_rt_tran_count**
cram_com_rt_rec_tr	**cram_com_rt_tran_tr**

cram_com_rt_tran_sam

DESCRIPTION

Sets Subaddress field in RT-RT Transmit Command word.

USAGE

#include <cram.h>

int cram_com_rt_tran_sam (int *subadd*);

subadd	**internal subaddress within transmitting remote terminal - RT-RT transfer.**

REMARKS

This routine sets the subaddress field in the command word sent by the BC to the transmitting RT in a RT-RT transfer.

RETURN VALUE

CRAM_SUCCESS	**successful**
CRAM_INV_PARAM	**invalid subaddress**

EXAMPLE

```
/* Set Transmitting RTs subaddress field to 8 */

cram_com_rt_tran_sam (8);
```

SEE ALSO

cram_com_rt_rec_sam
cram_com_rt_rec_rem
cram_co m_rt_rec_count
cram_co m_rt_rec_tr

cram_com_sam
cram_co m_rt_tra n_rem
cram_co m_rt_tran_count
cram_com_rt_tran_tr

cram_com_rt_rec_tr

DESCRIPTION

Sets Transmit/Receive bit in RT-RT Receive Command word.

USAGE

#include <cram.h>

int cram_com_rt_rec_tr (int *tr_bit*);

tr_bit	**transmit/receive bit (0 or 1)**

REMARKS

This routine sets the transmit/receive bit in the command word sent by the BC to the receiving RT in a RT-RT transfer. Note that by definition this would normally be set to "0".

RETURN VALUE

CRAM_SUCCESS	**successful**
CRAM_INV_PARAM	**invalid tlr bit**

EXAMPLE

```
/* Set t/r bit to 0 (receive) */

cram_com_rt_rec_tr (0);
```

SEE ALSO

cram_com_rt_rec_sam
cram_com_rt_rec_rem
cram_com_rt_rec_count
cram_com_rt_tran_tr
cram_com_rt_tran_sam
cram_com_rt_tran_rem
cram_com_rt_tran_count
cram_com_tr

cram_com_rt_tran_tr

DESCRIPTION

Sets Transmit/Receive bit in RT-RT Transmit Command word.

USAGE

```
#include <cram.h>

int cram_com_rt_tran_tr (int tr_bit);
```

tr_bit	**transmit/receive bit (0 or 1)**

REMARKS

This routine sets the transmit/receive bit in the command word sent by the BC to the transmitting RT in a RT-RT transfer. Note that by definition this would normally be set to "1".

RETURN VALUE

CRAM_SUCCESS	**successful**
CRAM_INV_PARAM	**invalid t/r bit**

EXAMPLE

```
/* Set t/r bit to I (transmit) */

cram_com_rt_tran_tr (1);
```

SEE ALSO

cram_com_rt_rec_sam
cram_com_rt_rec_rem
cram_com_rt_rec_count
cram_com_rt_rec_tr

cram_co m_rt_tra n_sam
cram_co m_rt_tra n_rem
cram_com_rt_tran_count
cram_com_tr

cram_com_rt_tran_count

DESCRIPTION

Sets Word Count field of Transmit Command word.

USAGE

#include <cram.h>

int cram_com_rt_tran_count (*int count*);

count	**number of data words**

REMARKS

This routine sets the word count/mode code field in the command word to be transmitted by the BC to the transmitting RT in RT-RT transfers. This indicates the number of data words to transmit immediately after its Status word. The value 32 is coded as 0 ("00000b") in the command word.

RETURN VALUE

CRAM_SUCCESS	**successful**
CRAM_INV_PARAM	**invalid number of data words**

EXAMPLE

```
/* Transmit 8 data words after Status word */

cram_com_rt_tran_count (8);
```

SEE ALSO

cram_com_rt_rec_sam
cram_com_rt_rec_rem
cram_com_rt_rec_count
cram_co m_rt_rec_tr

cram_com_rt_tra n_sam
cram_com_rt_tra n_rem
cram_com_count
cram_com_rt_tran_tr

cram_com_rt_rec_count

DESCRIPTION

Sets Word Count field of Receive Command word.

USAGE

#include <cram.h>

int cram_com_rt_rec_count (int *count*);

count	**number of Data words**

REMARKS

This routine sets the word count field in the command word to be transmitted by the BC to the receiving RT in RT-RT transfers. This indicates the number of data words it should expect from the transmitting RT following the transmit command. The value 32 is coded as 0 ("00000b") in the command word.

RETURN VALUE

CRAM_SUCCESS	**successful**
CRAM_INV_PARAM	**invalid number of data words**

EXAMPLE

```
/* Set Word Count field of RT-RT Command word to 8 */

cram_com_rt_rec_count (8);
```

SEE ALSO

cram_com_rt_rec_sam	**cram_com_rt_tran_sam**
cram_co m_rt_rec_rem	**cram_com_rt_tran_rem**
cram_com_count	**cram_com_rt_tran_count**
cram_com_rt_rec_tr	**cram_com_rt_tran_tr**

cram_bc_set_noerror

DESCRIPTION

Disables all transmission error injection.

USAGE

```
#include <cram.h>

int cram_bc_set_noerror (void);
```

REMARKS

This routine disables any error injection in the transmission frame. This includes error injection in the parity bit, and error injection in the sync pulse.

RETURN VALUE

CRAM_SUCCESS	**successful**

EXAMPLE

```
cram_bc_set_noerror ( );
```

SEE ALSO

cram_bc_set_err_syn
cram_bc_set_err_parity
cram_bc_res_err_syn
cram_bc_res_err_parity

cram_bc_set_err_sync

DESCRIPTION

Turns on Error Injection insync signal.

USAGE

```
#include <cram.h>

int cram_bc_set_err_sync (void);
```

REMARKS

This routine sets error injection on the sync pulse at the beginning of each command or status word. The system will transmit a Data sync signallnstead of a Command sync signal at the beginning of each data frame transfer.

RETURN VALUE

CRAM_SUCCESS	successful

EXAMPLE

```
cram_bc_set_err_sync ( );
```

SEE ALSO

cram_bc_set_noerror
cram_bc_res_err_syn
cram_bc_set_err_parity
cram_bc_res_err_parity

cram_bc_set_mode_data

DESCRIPTION

Loads an outgoing data word to follow a Mode command.

USAGE

```
#include <cram.h>

int cram_bc_mode_data (WORD value);

value                                   Data word
```

REMARKS

In a mode command, a maximum of 1 data word can be sent by the BC to the RT following the command word. This function allows the user to load that word into the BC Control Block for use in the upcoming mode command.

RETURN VALUE

CRAM_SUCCESS

EXAMPLE

```
WORD value = 0xabcd;
cram_bc_mode_data (value);
```

SEE ALSO

cram_bc_get_rx_mode_data

cram_bc_set_rx_start_address

DESCRIPTION

Sets the start address in the data buffer to use for storing incoming data words.

USAGE

```
#include <cram.h>

int cram_bc_set_rx_start_address (WORD rx_start_address);
```

rx_start_address	**address of first word**

REMARKS

The area from offset 0000-07FOO is the data area on the cram board. The user is given freedom to allocate this memory range as he sees fit. This function allows the user to specify where incoming data words to the BC are to be stored. The user should be careful not to overwrite other data which he may have previously placed in the same location unless it is no longer needed.

RETURN VALUE

CRAM_SUCCESS	**successful**
CRAM_INV_ADDR	**address not in range**

EXAMPLE

```
int result;
WORD rx_start_address;
result=cram_bc_set_rx_start_address (rx_start_address);
```

SEE ALSO

cram_rt_set_rx_start_address

cram_bc_set_err_parity

DESCRIPTION

Turns on Error Injection in Parity bit.

USAGE

```
#include <cram.h>

int cram_bc_set_err_parity (void)
```

REMARKS

This routine causes the unit to transmit even parity instead of odd parity as required by MIL-STD-1553.

RETURN VALUE

CRAM_SUCCESS	**successful**

EXAMPLE

```
cram_bc_set_err_parity ( );
```

SEE ALSO

cram_bc_set_noerror
cram_bc_set_err_syn
cram_bc_res_err_syn
cram_bc_res_err_parity

cram_bc_set_mm_time_tag

DESCRIPTION

Sets the Time Tag field for a particular MM block

USAGE

#include <cram.h>

int cram_bc_set_mm_time_tag
(WORD start_address, int block, BYTE time_tag);

start_address	**address of first block in buffer**
block	**block number to be set (0-31)**
time_tag	**intervalln ticks between transmissions**

REMARKS

Each block contains a Time Tag field which the user can set to a value between 1 and 255 ticks. A tick is 10 milliseconds in duration. The board then transmits that block whenever this interval of time elapses. Flexibility is given to the user to schedule a specific time interval for each block.

To disable a particular block from the cycle, set the time tag to ‘0’.

RETURN VALUE

CRAM_SUCCESS successful

CRAM_INV_PARAM invalid block number

EXAMPLE

See example for cram_bc_init_mm

cram_bc_set_mm_word_count

DESCRI PTION

Sets the Word Count field for a particular MM block

USAGE

```
        #include <cram.h>

int cram_bc_set_mm_word_count
        (WORD start_address, int block, BYTE word_count);
```

start_address	address of first block in buffer
block	block number to be set (0-31)
word_count	number of data words to transmit

REMARKS

Each block contains a Word Count field which tells the CRAM system how many data words should be transmitted following the command word. This field alone determines this number. The Word Count field and the T/R field in the Command Word do not have any effect on the actual number of words to be transmitted. The user must remember to set this number to 0 for any message types which do not require the BC to transmit data words.

RETURN VALUE

CRAM_SUCCESS successful

CRAM_INV_PARAM invalid block number

EXAMPLE

```
        WORD start_address = 0;

        BYTE block = 10, word_count = 5;

cram_bc_set_mm_word_count(start_address, block, word_count);
```

SEE ALSO

cram_bc_init_mm

cram_bc_set_mm_command

DESCRIPTION

Sets the Command Word field for a particular MM block.

USAGE

```
        #include <cram.h>

int cram_bc_set_mm_command
        (WORD start_address, int lock,MIL_WORD command);
```

start_address	address of first block in buffer
block	block number to be set (0-31)
command	MIL-STD-1553 Command Word

REMARKS

Each block contains a Command Word field which is the first word transmitted in the block. This function lets the user enter the entire Command Word at once. The following functions allow the user to enter the Command Word a field at a time.

RETURN VALUE

CRAM_SUCCESS successful

CRAM_INV_PARAM invalid block number

EXAMPLE

```
        WORD start_address = 0;

        BYTE block= 10;

MIL_WORD command =28a5;
cram_bc_set_mm_command(start_address, block, command);
```

SEE ALSO

cram_bc_init_mm cram_bc_set_mm_com_rem
cram_bc_set_mm_time_tag cram_bc_set_mm_word_count

cram_bc_set_mm_com_rem

DESCRIPTION

Sets the Remote Terminal Address field in the Command Word for a particular MM block.

USAGE

```
        #include <cram.h>

int cram_bc_set_mm_com_rem
        ( WORD start_address, int block, WORD remote_address);
```

start_address	address of first block in buffer
block block	number to be set (0-31)
remote_address	Remote Address field (0-31)

REMARKS

Each block contains a Command Word field which is the first word transmitted in the block. This function lets the user enter the Remote Terminal Address field of the command word.

RETURN VALUE

CRAM_SUCCESS successful

CRAM_INV_PARAM invalid block number

EXAMPLE

```
        WORD start_address = 0;

        BYTE block = 10;

WORD rem_add=5;
cram_bc_set_mm_com_rem(start_address, block, rem_add);
```

SEE ALSO

cram_bc_init_mm

cram_bc_set_mm_time_tag

cram_bc_set_mm_command

cram_com_rem

cram_bc_set_mm_com_tr

DESCRIPTION

Sets the Transmit/Receive bit in the Command Word for a particular MM block.

USAGE

```
        #include <cram.h>

int cram_bc_set_mm_com_tr
        (WORD start_address, int block, WORD tr_bit);
start_address           address of first block in buffer
block                   block number to be set (0-31)
tr_bit                  Transmit/Receive bit:
('l'—RT transmits; 'O'—RT receives)
```

REMARKS

Each block contains a Command Word field which is the first word transmitted in the block. This function lets the user enter the Transmit/Receive bit of the command word.

RETURN VALUE

CRAM_SUCCESS successful

CRAM_I NV_PARAM invalid block number

EXAMPLE

```
        WORD start_address = 0;

        BYTE block = 10;

WORD tr=0;
cram_bc_set_mm_com_tr(start_address, block, tr);
```

SEE ALSO

cram_bc_init_mm	cram_bc_set_mm_command
cram_bc_set_mm_time_tag	cram_bc_set_mm_word_count
cram_com_tr	

cram_bc_set_mm_com_sam

DESCRIPTION

Sets the Subaddress/Mode field in the Command Word for a particular MM block.

USAGE

#include <cram.h>

int cram_bc_set_mm_com_sam
(WORD start_address, int block, WORD sam);

start_address	**address of first block in buffer**
block	**block number to be set (0-31)**
sam	**Subaddress/Mode field (0-31)**

REMARKS

Each block contains a Command Word field which is the first word transmitted in the block. This function lets the user enter the Subaddress/Mode field of the command word.

RETURN VALUE

CRAM_SUCCESS successful

CRAM_I NV_PARAM invalid block number

EXAMPLE

```
        WORD start_address = 0;

        BYTE block= 10;

WORD subadd=5;
cram_bc_set_mm_com_sam(start_address, block, subadd);
```

SEE ALSO

cram_bc_init_mm **cram_bc_set_mm_command**
cram_bc_set_mm_time_tag **cram_bc_set_mm_word_count**
cram_com_sam

cram_bc_set_mm_com_count

DESCRIPTION

Sets the Word Count field in the Command Word for a particular MM block.

USAGE

```
        #include <cram.h>

int cram_bc_set_mm_com_count
        ( WORD start_address, int block, WORD count);
```

start_address	address of first block in buffer
block	block number to be set (0-31)
count	Word Count field (0-31, 0=32)

REMARKS

Each block contains a Command Word field which is the first word transmitted in the block. This function lets the user enter the Word Count field of the command word.

RETURN VALUE

CRAM_SUCCESS successful

CRAM_I NV_PARAM invalid block number

EXAMPLE

```
        WORD start_address = 0;

        BYTE block = 10;

WORD count=5;
cram_bc_set_mm_com_count(start_address, block, count);
```

SEE ALSO

cram_bc_init_mm
cram_bc_set_mm_time_tag
cram_com_count
cram_bc_set_mm_command
cram_bc_set_mm_word_count

cram_bc_set_mm_data_word

DESCRIPTION

Writes a single data word in a particular MM block.

USAGE

```
        #include <cram.h>

int cram_bc_set_mm_data_word
  (WORD start_address, int block, MIL_WORD data_word, int word_num);
```

start_address	address of first block in buffer
block block	number to be set (0-31) data_word,
MIL-STD-1553	data word
word_num	position in block to be written (0-31)

REMARKS

Each block contains space for an array of 32 Data words. This function enters single Data word into a specified location of a particular block

RETURN VALUE

CRAM_SUCCESS successful

CRAM_INV_PARAM invalid block number

EXAMPLE

```
WORD start_address = 0;
BYTE block = 10;
 MIL_WORD data=1111;
int num=5;
        cram_bc_set_mm_data_word(start_address, block, subadd, num);
```

SEE ALSO

cram_bc_init_mm	cram_bc_write_mm_data_words
cram_bc_set_mm_time_tag	cram_bc_set_mm_word_count

cram_bc_res_err_sync

DESCRIPTION

Turns off Error Injection insync signal.

USAGE

```
#include <cram.h>

int cram_bc_res_err_sync (void);
```

REMARKS

This routine resets (turns off) Error Injection on the sync pulse at the beginning of each command or status word.

RETURN VALUE

CRAM_SUCCESS	successful

EXAMPLE

```
cram_bc_res_err_sync ( );
```

SEE ALSO

cram_bc_set_noerror
cram_bc_set_err_syn
cram_bc_set_err_parity
cram_bc_res_err_parity

cram_bc_res_err_parity

DESCRIPTION

Turns off Error Injection in Parity bit.

USAGE

```
#include <cram.h>

int cram_bc_set_err_parity (void);
```

REMARKS

This routine resets (turns off) parity error injection. The system transmits odd parity at the end of each word.

RETURN VALUE

CRAM_SUCCESS	**successful**

EXAMPLE

```
cram_bc_res_err_parity 0;
```

SEE ALSO

cram_bc_set_noerror
cram_bc_set_err_syn
cram_bc_res_err_syn
cram_bc_set_err_parity

cram_bc_get_rt_rt_rec_status

DESCRIPTION

Retrieves incoming status word from receiving RT in RT-RT transfers.

USAGE

```
#include <cram.h>

MlL_WORD cram_bc_get_rt_rt_rec_status (void)
```

REMARKS

In RT-RT transfers, after the receiving RT concludes its reception of data words, it must respond with a status word to the BC to confirm receipt. This function allows the user to obtain the last such status word received by the BC.

RETURN VALUE

Contents of status word.

EXAMPLE

```
MIL_WORD recstat;

recstat = cram_bc_get_rt_rt_rec_status( );
```

SEE ALSO

cram_bc_get_rt_rt_tran_status

cram_bc_get_rt_rt_tran_status

DESCRIPTION

Retrieves incoming status word from transmitting RT in RT-RT transfers.

USAGE

```
#include <cram.h>

MIL_WORD cram_bc_get_rt_rt_tran_status (void)
```

REMARKS

In RT-RT transfers, before the transmitting RT commences its transmission of data words, it must respond with a status word to the BC to confirm receipt of the command. This function allows the user to obtain the last such status word received by the BC.

RETURN VALUE

Contents of status word.

EXAMPLE

```
MIL_WORD transtat;
transtat = cram_bc_get_rt_rt_tran_status();
```

SEE ALSO

cram_bc_get_rt_rt_rec_status

cram_bc_get_rx_mode_data_1553

DESCRIPTION

Retrieves incoming data word from RT following a mode command from BC.

USAGE

```
#include <cram.h>

MIL_WORD cram_bc_get_rx_mode_data_1553(void);
```

REMARKS

In a mode command, a maximum of 1 data word can be requested from the RT by the BC. This function allows the user to obtain the last such data word received by the BC.

RETURN VALUE

Contents of data word.

EXAMPLE

```
MIL_WORD rx_mode_data;
rx_mode_data = cram_bc_get_rx_mode_data_1553( );
```

SEE ALSO

cram_rt_get_rx_mode_data
cram_bc_mode_data

cram_bc_get_rx_prev_stat

DESCRIPTION

Retrieves previously received status word.

USAGE

```
#include <cram.h>

MIL_WORD cram_bc_get_rx_prev_stat (void);
```

REMARKS

The CRAM system stores the previously received status word in addition to the current or latest status word. This function allows the user to obtain that status word.

RETURN VALUE

Contents of status word.

EXAMPLE

```
MIL_WORD rx_prev_status;
rx_prev_status = cram_bc_get_rx_prev_stat
```

SEE ALSO

cram_bc_get_rx_curr_status

cram_bc_get_result

DESCRIPTION

This routine checks the BC command response.

USAGE

```
#include <cram.h>

int cram_bc_get_result (void);
```

REMARKS

This routine returns the BC command response.

RETURN VALUE

board command response

cram_bc_get_rx_word_count

DESCRIPTION

Retrieves the received word count.

USAGE

```
#include <cram.h>

WORD cram_bc_get_rx_word_count (void);
```

REMARKS

In CRAM BC mode the incoming word count (from an RT) is stored in the BC control block. This function allows the user to obtain that number.

RETURN VALUE

Number of words received.

EXAMPLE

```
WORD count;
count = cram_bc_get_rx_word_count
```

SEE ALSO

cram_rt_get_rx_word_count

cram_bc_get_rx_curr_stat

DESCRIPTION

Returns the current received status.

USAGE

#include <cram.h>

int cram_bc_get_rx_curr_stat (void);

REMARKS

This routine returns the most recent received status word.

RETURN VALUE

last received status

SEE ALSO

cram_bc_get_rx_prev_stat

cram_bc_get_mm_command

DESCRIPTION

Retrieves the Command Word field from a particular MM block.

USAGE

#include <cram.h>

MIL_WORD cram_bc_get_mm_command
(WORD start_address, int block);

start address	**address of first block in buffer**
block	**block number to be set (0-31)**

REMARKS

Each MM block may contain a Command Word field which is the first word transmitted in the block. This function lets the user retrieve the entire Command word at once.

RETURN VALUE

Contents of Command word field.

EXAMPLE

```
MIL_WORD result;

WORD start_address = 0;
BYTE block = 10;
result = cram_bc_set_mm_command(start_address, block);
```

SEE ALSO

cram_bc_init_mm **cram_bc_set_mm_time_tag**
cram_bc_set_mm_com_rem **cram_bc_set_mm_command**

cram_bc_get_mm_time_tag

DESCRIPTION

Checks the number of “Ticks” for a specific block message number in BC multi-mode.

USAGE

#include <cram.h>

DWORD cram_bc_get_mm_time_tag
(WORD start_address, INT num);

start_address BC multi-mode data block start address

REMARKS

This function returns the time period between two consecutive message in BC multi-mode.

RETURN VALUE

TIME Number of ticks between messages.

SEE ALSO

cram_bc_tick

cram_bc_get_mm_word_count

DESCRIPTION

Returns the number of words to be transmitted in BC multi-mode for a specific buffer.

USAGE

#include <cram.h>

int cram_bc_get_mm_word_count (WORD start_address INT num);

start_address	**BC multi-mode data block start address**
num data	**block number offset (0 - 31)**

REMARKS

This function returns the number of words transmitted in BC multi-mode.

RETURN VALUE

WORD_COUNT Number or Words to be Transmitted

cram_bc_get_mm_data_start_address

DESCRIPTION

Returns the start address of a specific data block to be transmitted in BC multi-mode.

USAGE

#include <cram.h>

```
int cram_bc_get_mm_data_start_address
     (WORD start_address,  INT num);
start_address         BC multi-mode data block start address
num                   number of data blocks (0 - 31)
```

REMARKS

This function sets the offset start address of the data block to be transmitted in BC multi-mode.

RETURN VALUE

Block data start address (offset)

cram_bc_init_mm

DESCRIPTION

Initializes BC Multiple Mode Scheduling Buffer in BC mode.

USAGE

#include <cram.h>

int cram_bc_in it_mm (WORD start_address);

start_address address of first block in buffer

REMARKS

The CRAM Multiple Mode allows a user to set up a custom tailored rotation among RTs for the CRAM BC to follow. The Data area is divided into 32 blocks**. Each consists of: a Time Tag which counts down from the user programmed value (in ticks) until reaching '0', at which point transmission of the block commences; a MIL-STD-1553 Command Word field; a Word Count field containing the number of data words to follow, and an array of 32 Data Words. The blocks are contiguous with each one starting at the next location following the previous block. Since a block consists of 68 bytes, the total space needed is 2176 bytes. This function checks that there is sufficient space beginning at the start address until the end of the Data area (0x7F00 or 0xF00) to accomodate this space requirement. If there is, the function then zeros the entire range of 2176 bytes to provide a clean slate before the user programs the individuals fields.**

This function should be called before any other in MM mode, and the result should be checked for CRAM_SUCCESS before proceeding further.

RETURN VALUE

CRAM_SUCCESS successful

CRAM_INV_ADDRESS insufficient space at Start Address

EXAMPLE

```
WORD add;
if (cram_bc_init_mm(add)==CRAM_SUCCESS)
{
cram_bc_set_mm_time_tag (10);
}
```

SEE ALSO

cram_bc_set_mm_time_tag

cram_bc_write_mm_data_words

DESCRIPTION

Writes 32 Data words into a particular MM block.

USAGE

#include <cram.h>

int cram_bc_write_mm data_words
(WORD start_address, int block, void *src);

start_address	**address of first block in buffer**
block	**block number to be set (0-31)**
src	**Pointer to location of source of MIL-STD-1553 data words to be written**

REMARKS

Each block contains space for an array of 32 Data words. This function writes all 32 Data words to a particular block using a pointer to the source of the Data words.

RETURN VALUE

CRAM_SUCCESS successful

CRAM_I NV_PARAM invalid block number

EXAMPLE

```
WORD start_address = 0;
BYTE block = 10;
MIL_WORD new_data[32]={Oxl 111, 0x2222, 0x3333, . .
cram_bc_write_mm_data_words(start_address, block, new_data);
```

SEE ALSO

cram_bc_init_mm **cram_bc_set_mm_data_word**
cram_bc_set_mm_time_tag **cram_bc_set_mm_word_count**

cram_bc_resp_gap

DESCRIPTION

This functions returns the response time gap.

USAGE

```
#include <cram.h>

int cram_bc_resp_gap ();
```

REMARKS

Each time a command is given, a response should be received by the BC from a remote terminal. This function returns the response gap between the end of a transmission, and the beginning of the received response.

RETURN VALUE

This function returns the value as an unsigned character (Max. value = 255), in resolution steps of 60 nanoseconds. i.e. 255 = 16 microseconds. The 9th bit indicates a time overflow when is '1'.

cram_bc_exec_instruction

DESCRIPTION

Sends BC Transmit Command to CRAM system

USAGE

#include <cram.h>

int cram_bc_exec_instruction
(BYTE *channel*, BYTE *instruction*, WORD *start*, WORD *count*, WORD *delay*);

channel	**The transmit channel: A (‘0’) or B (‘1’)**
instruction	**One of:** **CRAM_CRAMD_IDLE** **CRAM_BC_NORMAL,** **CRAM_BC_LOOP,** **CRAM_BC_STOP,** **CRAM_BC_RT_RT,** **CRAM_BC_RT_RT_LOO P,** **CRAM_BC_MODE,** **CRAM_BC_MODE_DATA,** **CRAM_BC_MULTIPLE_MM,** **CRAM_BC_CABLE_TEST.**
start	**Location (offset from board base address) of the first data word for transmission. (Must be in range 0x0000 to 0x0F00.)**
count	**Data Count--number of data words to be transmitted, starting address *start*.**
delay	**Inter-message delay in ticks (0 to 65,536)**

REMARKS

This routine sets-up a Transmit Command in the board's BC Control Block and waits for the board's response. Before calling this function, the data to be transmitted (not applicable for STOP command) should be placed in CRAM memory--normally by using the cram_wntebuf function--at the address offset start. In the case of a CRAM_BC_NORMAL command CRAM will initiate a time delay of delay microseconds after which it will transmit count words starting at address start in the memory area allocated as the transmit buffer. In a CRAM_BC_LOOP command, CRAM will perfom the same operations as in NORMAL mode, but after the entire buffer is transmitted, it will re-initiate the entire sequence (delay+transmission) and will keep cycling until it receives a CRAM_BC_STOP command. In a CRAM_BC_STOP command, all the other parameters are ignored and transmission (if any) is stopped on the channel.

To facilitate specialized communications needs, the CRAM board supports certain additional commands. A CRAM_BC_RT_RT command causes the CRAM system to transmit two consecutive command words addressed to two specific RTs respectively. (This can be done repetitively with CRAM_BC_RT_RT_LOOP.) The first is a receive command; the second a transmit command. See Introduction to MIL-STD-1553 (Chapter 1) for details on this message format. A CRAM_BC_MODE command causes the CRAM system to transmit a mode command without a following data word; a CRAM_BC_MODE_DATA command transmits a mode command with a following data word. In this case the data word is stored in a special slot in the BC Control Block rather than in the transmit buffer so as to make it easier for the user to keep track of it. In each of these preceding cases the CRAM system will initiate the usual delay of delay microseconds before executing the instruction.

The CRAM board supports one additional type of operation which is useful for testing a fully loaded bus containing multiple Remote Terminals as one would encounter in a reallife situation--the CRAM_BC_MULTIPLE_MM mode. In this mode the Transmit Buffer is fixed insize at 32 cells of 34 words each for a total of 1088 words or 2176 bytes, starting at the specified start address in the function call. Each of these 32 cells contains outgoing traffic for "any" particular Remote Terminal. In the first byte of each cell, the user must write a number (1-255) which is a time interval or time tag in units of 10 msec ticks which is the amount of time to elapse before transmission of that block in each cycle. In the second byte the user must write the number of data words to be transmitted in the specific block. In the succeeding locations of the cell the user must write the command word and data words to be transmitted (up to a maximum of 32 words). Upon a successfull execution of a transmit command in this mode, The CRAM processor will scan the buffer every 10 msec and will decrement the Time Tag of each RT. Whenever an interval value reaches zero, the data words in that cell are enqueued for transmission. The time intervalls then reinitialized to its original value to prepare for the next cycle. The slowest

transfer period possible for a given cell is 10 msec * 255, i.e., approximately 2.5 seconds.

CRAM_BC_CABLE_TEST transmits the BC_ctl1553 command register through channel A, increments the tx_counter, sets the appropriate bit on the rx_tx_Indicator, and expects the response to be received on channel B. The unit stores the response in the BC_ctl1553 rx_current_status, increments the rx_counter, and sets the appropriate bit in the rx_tx_indicator register. NOTE: One end of the cable must be terminated with the proper ohm resister in order to not create a reflected signal.

RETURN VALUE

CRAM_SUCCESS	**command was accepted: execution has started**
CRAM_INV_CHANNEL	**invalid or unconfigured channel specified**
CRAM_INV_ADDRESS	**start address does not fall in the 0-7F00 (0-F00) Range**
CRAM_INV_SIZE	**count is such that the last data word would fall outside of the 0-7F00 (0-F00) range (start + count> 7F00 (F00)).**
CRAM_CHNL_BUSY	**A command other than CRAM_BC_STOP has been issued while the board is in the middle of a transmission.**
CRAM_BOARD_RESP	**board did not respond to the command within the response interval defined by CRAM_RESP_ TIMEOUT**

EXAMPLE

See TXCRAMD.C

SEE ALSO

cram_writebuf
cram_tx_count
cram_di_txcomplete
cram_tx_complete
cram_ei_txcomplete

Remote Controller Mode Functions (RT) The functions read and write into first control block defined as "rt_ctl" in the CRAM1553.H

RT Board Functions (rt_ctl)

cram_rt_resp_gap ....

.... RT Status Word Reset and Reset bits ....
cram_sta_rem ....
cram_sta_set_busy ....
cram_sta_res_busy ....
cram_sta_set_dyn ....
cram_sta_res_dyn ....
cram_sta_set_err ....
cram_sta_res_err ....
cram_sta_set_ins...
cram_sta_res_ins...
cram_sta_set_serv ....
cram_sta_res_serv ....
cram_sta_set_subs ....
cram_sta_res_subs ....
cram_sta_set_term ....
cram_sta_res_term ....
cram_sta_set_broad ....
cram_sta_res_broad ....

cram_rt_set_ address...
cram_rt_set_rx_address..
cram_rt_set_start_address...
cram_rt_set_wait_response...

cram_rt_load_all_mode_responses.................................
cram_rt_load_al_subaddress ....
cram_rt_load_single_mode_responses.................................
cram_rt_load_single_subaddress ....

cram_rt_resp_gap

DESCRIPTION

Returns the response time between the BC and RT command transmissions.

USAGE

```
#include <cram.h>

int cram_rt_resp_gap (void);
```

REMARKS

The timing gap between the end of the BC transmission and the beginning of the RT response is returned by this function. The return value has a 60 nSec resolution.

RETURN VALUE

Response Gap 1-255 (60 nSec resolution)

cram_sta_rem

DESCRIPTION

Sets Remote Terminal Address field in status word.

USAGE

#include <cram.h>

int cram_sta_rem (int *remote_add*);

remote_add	**the remote terminal address (0 to 30)**

REMARKS

This routine sets the Remote Terminal address field in the Status word. Note, however, that this does not necessarily mean that the CRAM board's RT address has been set to that value. This must be set separately by means of the function cram_set_rt_address

RETURN VALUE

CRAM_SUCCESS	**successful**
CRAM_INV_PARAM	**invalid remote terminal address**

EXAMPLE

```
/ Set remote address field to 30 in status word */

cram_sta_rem (30);
```

cram_sta_set_err

DESCRIPTION

Sets Message Error bit in status Word.

USAGE

```
#include <cram.h>

int cram_sta_set_err (void);
```

REMARKS

This routine sets the Message Error bit to "1" in the Status word. Note that in future versions of the board, the board logic will be able to set this bit automatically in response to actual error conditions. Use of this bit is optionalln MIL-STD-1553. Please see chapter 1 of this User's Guide for further information.

RETURN VALUE

CRAM_SUCCESS	successful

EXAMPLE

```
/ Set message error bit in status word */

cram_sta_set_err ( );
```

SEE ALSO

cram_sta_set_i ns
cram_sta_setserv
cram_sta_set_broad
cram_sta_set_busy
cram_sta_set_subs
cram_sta_set_dyn
cram_sta_set_term
cram_sta_res_err

cram_sta_res_i ns
cram_sta_res_serv
cram_sta_res_broad
cram_sta_res_busy
cram_sta_res_subs
cram_sta_res_dyn
cram_sta_res_term

cram_sta_res_err

DESCRIPTION

Resets Message Error bit in status word.

USAGE

#include <cram.h>

int cram_sta_res_err (void);

REMARKS

This routine resets to “0” the message error bit in the status word.

RETURN VALUE

CRAM_SUCCESS successful

EXAMPLE

```
/* Reset error bit in status word */

cram_sta_res_err 0;
```

SEE ALSO

cram_sta_set_ins	cram_sta_res_i ns
cram_sta_set_serv	cram_sta_res_serv
cram_sta_set_broad	cram_sta_res_broad
cram_sta_set_busy	cram_sta_re s_busy
cram_sta_set_subs	cram_sta_res_subs
cram_sta_set_dyn	cram_sta_res_dyn
cram_sta_set_term	cram_sta_res_term
cram_sta_set_err	

cram_sta_set_ins

DESCRIPTION

Sets Instruction bit in status word.

USAGE

```
#include <cram.h>

int cram_sta_set_ins (void);
```

REMARKS

This routine sets to ‘1”the instruction bit in the status word.

RETURN VALUE

CRAM_SUCCESS successful

EXAMPLE

```
/ Set instruction bit in status word */

cram_sta_set_ins ( );
```

SEE ALSO

cram_sta_set_err
cram_sta_set_serv
cram_sta_set_broad
cram_sta_set_busy
cram_sta_set_subs
cram_sta_set_dyn
cram_sta_set_term
cram_sta_res_i ns

cram_sta_res_err
cram_sta_res_serv
cram_sta_res_broad
cram_sta_res_busy
cram_sta_res_subs
cram_sta_res_dyn
cram_sta_res_term

cram_sta_res_ins

DESCRIPTION

Resets Instruction bit in status word.

USAGE

```
#include <cram.h>

int cram_sta_res_ins (void);
```

REMARKS

This routine resets to “0” the instruction bit in the status word.

RETURN VALUE

CRAM_SUCCESS successful

EXAMPLE

```
I Set instruction bit in status word */

cram_sta_reset_ins ( );
```

SEE ALSO

cram_sta_set_err
cram_sta_set_ins
cram_sta_res_serv
cram_sta_res_broad
cram_sta_res_busy
cram_sta_res_subs
cram_sta_res_dyn
cram_sta_res_term
cram_sta_res_err
cram_sta_set_serv
cram_sta_set_broad
cram_sta_set_busy
cram_sta_set_subs
cram_sta_set_dyn
cram_sta_set_term

cram_sta_set_serv

DESCRIPTION

Sets Service bit in status word.

USAGE

#include <cram.h>

int cram_sta_set_serv (void);

REMARKS

This routine sets to "1" the service bit in the status word.

RETURN VALUE

CRAM_SUCCESS successful

EXAMPLE

```
        /* Set instruction bit in status word */

cram_sta_set_serv ( );
```

SEE ALSO

cram_sta_set_err
cram_sta_set_ins
cram_sta_res_serv
cram_sta_res_broad
cram_sta_res_busy
cram_sta_res_subs
cram_sta_res_dyn
cram_sta_res_term
cram_sta_res_err
cram_sta_res_ins
cram_sta_set_broad
cram_sta_set_busy
cram_sta_set_subs
cram_sta_set_dyn
cram_sta_set_term

cram_sta_res_serv

DESCRIPTION

Reset Service bit in status word.

USAGE

```
#include <cram.h>

int cram_sta_res_serv (void);
```

REMARKS

This routine resets to “0” the service bit in the status word.

RETURN VALUE

CRAM_SUCCESS successful

EXAMPLE

```
I Set service bit in status word */

cram_sta_reset_serv 0;
```

SEE ALSO

cram_sta_set_err
cram_sta_set_ins
cram_sta_set_serv
cram_sta_res_broad
cram_sta_res_busy
cram_sta_res_subs
cram_sta_res_dyn
cram_sta_res_term

cram_sta_res_err
cram_sta_res_ins
cram_sta_set_broad
cram_sta_set_busy
cram_sta_set_subs
cram_sta_set_dyn
cram_sta_set_term

cram_sta_set_broad

DESCRIPTION

Sets broadcast bit in status word.

USAGE

```
#include <cram.h>

int cram_sta_set_broad (void);
```

REMARKS

This routine sets to “1” the broadcast command received bit in the status word. In future versions of the board this bit will be set automatically by the board logic.

RETURN VALUE

CRAM_SUCCESS successful

EXAMPLE

```
/* Set broadcast bit in status word */

cram_sta_set_broad 0;
```

SEE ALSO

cram_sta_set_err
cram_sta_set_ins
cram_sta_set_serv
cram_sta_set_busy
cram_sta_set_subs
cram_sta_set_dyn
cram_sta_set_term
cram_sta_res_broad

cram_sta_res_err
cram_sta_res_i ns
cram_sta_res_serv
cram_sta_res_busy
cram_sta_res_subs
cram_sta_res_dyn
cram_sta_res_term

cram_sta_res_broad

DESCRIPTION

Resets Broadcast bit in status word

USAGE

```
#include <cram.h>

int cram_sta_res_broad (void);
```

REMARKS

This routine resets to “0” the broadcast bit in the status word.

RETURN VALUE

CRAM_SUCCESS successful

EXAMPLE

```
        /* Reset broadcast bit in status word */

cram_sta_reset_broad 0;
```

SEE ALSO

cram_sta_set_err
cram_sta_set_i ns
cram_sta_set_serv
cram_sta_set_busy
cram_sta_set_subs
cram_sta_set_dyn
cram_sta_set_term
cram_sta_set_broad

cram_sta_res_err
cram_sta_res_ins
cram_sta_res_serv
cram_sta_res_busy
cram_sta_res_subs
cram_sta_res_dyn
cram_sta_res_term

cram_sta_set_busy

DESCRIPTION

Sets Busy bit in status word.

USAGE

```
#include <cram.h>

int cram_sta_set_busy (void);
```

REMARKS

This routine sets to “1” the busy bit in the status word.

RETURN VALUE

CRAM_SUCCESS successful

EXAMPLE

```
/* Set busy bit in status word */

cram_sta_set_busy 0;
```

SEE ALSO

cram_sta_set_err
cram_sta_set_ins
cram_sta_set_serv
cram_sta_set_broad
cram_sta_set_subs
cram_sta_set_dyn
cram_sta_set_term
cram_sta_res_busy

cram_sta_res_err
cram_sta_res_ins
cram_sta_res_serv
cram_sta_re s_broad
cram_sta_res_subs
cram_sta_res_dyn
cram_sta_res_term

cram_sta_res_busy

DESCRI PTION

Resets Busy bit in status word.

USAGE

#include <cram.h>

int cram_sta_res_busy (void);

REMARKS

This routine resets to PIQU the busy bit in the status word.

RETURN VALUE

CRAM_SUCCESS successful

EXAMPLE

```
/* Reset busy bit in status word */

cram_sta_res_busy 0;
```

SEE ALSO

cram_sta_set_err
cram_sta_set_ins
cram_sta_set_serv
cram_sta_set_broad
cram_sta_set_subs
cram_sta_set_dyn
cram_sta_set_term
cram_sta_set_busy

cram_sta_res_err
cram_sta_res_ins
cram_sta_res_serv
cram_sta_res_broad
cram_sta_res_subs
cram_sta_res_dyn
cram_sta_res_term

cram_sta_set_subs

DESCRIPTION

Sets Subsystem bit in status word.

USAGE

#include <cram.h>

int cram_sta_set_subs (void);

REMARKS

This routine sets to “1” the subsystem bit in the status word.

RETURN VALUE

CRAM_SUCCESS successful

EXAMPLE

```
/* Set subsystem bit in status word */

cram_sta_set_subs ( );
```

SEE ALSO

cram_sta_set_err
cram_sta_set_ins
cram_sta_set_serv
cram_sta_set_broad
cram_sta_set_busy
cram_sta_set_dyn
cram_sta_set_term
cram_sta_res_subs

cram_sta_res_err
cram_sta_res_ins
cram_sta_res_serv
cram_sta_res_broad
cram_sta_res_busy
cram_sta_res_dyn
cram_sta_res_term

cram_sta_res_subs

DESCRIPTION

Resets Subsystem bit in status word.

USAGE

```
#include <cram.h>

int cram_sta_res_subs (void);
```

REMARKS

This routine resets to “0” the subsystem bit in the status word.

RETURN VALUE

CRAM_SUCCESS successful

EXAMPLE

```
/* Set Service bit in status word */

cram_sta_res_subs ( );
```

SEE ALSO

cram_sta_set_err
cram_sta_set_ins
cram_sta_set_serv
cram_sta_set_broad
cram_sta_set_busy
cram_sta_set_dyn
cram_sta_set_term
cram_sta_set_subs

cram_sta_res_err
cram_sta_res_ins
cram_sta_res_serv
cram_sta_res_broad
cram_sta_res_busy
cram_sta_res_dyn
cram_sta_res_term

cram_sta_set_dyn

DESCRIPTION

Sets Dynamic bit in status word.

USAGE

```
#include <cram.h>

int cram_sta_set_dyn (void);
```

REMARKS

This routine sets to “l”the dynamic bus control accept bit in the status word.

RETURN VALUE

CRAM SUCCESS successful

EXAMPLE

Set dynamic bit in status word

```
cram_sta_set_dyn 0;
```

SEE ALSO

cram_sta_set_err
cram_sta_set_ins
cram_sta_set_serv
cram_sta_set_broad
cram_sta_set_busy
cram_sta_set_subs
cram_sta_set_term
cram_sta_res_dyn

cram_sta_res_err
cram_sta_res_ins
cram_sta_res_serv
cram_sta_res_broad
cram_sta_res_busy
cram_sta_res_subs
cram_sta_res_term

cram_sta_res_dyn

DESCRIPTION

Resets Dynamic bit in status Word.

USAGE

```
#include <cram.h>

int cram_sta_res_dyn (void);
```

REMARKS

This routine resets to “0” the dynamic bus control accept bit in the status word.

RETURN VALUE

CRAM_SUCCESS successful

EXAMPLE

```
/* Reset dynamic bit in status word */

cram_sta_res_dyn 0;
```

SEE ALSO

cram_sta_set_err
cram_sta_set_i ns
cram_sta_set_serv
cram_sta_set_broad
cram_sta_set_busy
cram_sta_set_subs
cram_sta_set_term
cram_sta_set_dyn

cram_sta_res_err
cram_sta_res_ins
cram_sta_res_serv
cram_sta_res_broad
cram_sta_res_busy
cram_sta_res_subs
cram_sta_res_term

cram_sta_set_term

DESCRIPTION

Sets Terminal bit in status word.

USAGE

#include <cram.h>

int cram_sta_set_term (void);

REMARKS

This routine sets to 1"the terminal bit in the status word.

RETURN VALUE

CRAM_SUCCESS successful

EXAMPLE

/* Set subsystem bit in status word */

cram_sta_set_term 0;

SEE ALSO

cram_sta_set_err	**cram_sta_res_err**
cram_sta_set_ins	**cram_sta_res_ins**
cram_sta_set_serv	**cram_sta_res_serv**
cram_sta_set_broad	**cram_sta_res_broad**
cram_sta_set_busy	**cram_sta_res_busy**
cram_sta_set_subs	**cram_sta_res_subs**
cram_sta_set_dyn	**cram_sta_res_dyn**
cram_sta_set_term	

cram_sta_res_term

DESCRIPTION

Resets Terminal bit in status word.

USAGE

```
#include <cram.h>

int cram_sta_res_term (void);
```

REMARKS

This routine resets to “0” the terminal bit in the status word.

RETURN VALUE

CRAM_SUCCESS successful

EXAMPLE

```
/* Set Service bit in status word */

cram_sta_res_term 0;
```

SEE ALSO

cram_sta_set_err
cram_sta_set_i ns
cram_sta_set_serv
cram_sta_set_broad
cram_sta_set_busy
cram_sta_set_subs
cram_sta_set_dyn
cram_sta_set_term

cram_sta_res_err
cram_sta_res_i ns
cram_sta_res_serv
cram_sta_res_broad
cram_stajes_busy
cram_sta_res_subs
cram_sta_res_dyn

cram_rt_set_wait_response

DESCRIPTION

Sets additional wait time for RT-RT command and RT transmit response.

USAGE

```
#include <cram.h>

int cram_rt_setwait_response (BYTE wait time);
```

wait time additional wait time

REMARKS

This procedure sets additional wait time for the CRAM board for RT-RT transfers and RT transmit responses. The time added is = (5 * wait time). This register is also used in_bm_mode.

RETURN VALUE

CRAM_SUCCESS successful

cram_rt_set_remote_address

DESCRIPTION

Sets Remote Terminal address to which the board will respond.

USAGE

```
#include <cram.h>

int cram_rt_set_remote_address (BYTE address);
```

address the RT address (0-30)

REMARKS

This function sets the actual RT address to which the board will respond to commands from the bus controller. It should not be confused with the function cram_sta_rem which sets the RT address in the status word. The user is given the flexibility to set the board to a certain address, and yet transmit status words which contain a different originating address, even though that would be an error according to MIL_STD_1553.

RETURN VALUE

CRAM_SUCCESS successful

CRAM_INV_PARAM incorrect address (not from 0-30)

EXAMPLE

```
/* Set Remote Terminal Address to 15 */

cram_rt_set_remote_address (15);
```

cram_rt_set_rx_start_address

DESCRIPTION

Sets the start address in the data buffer to use for storing incoming data words.

USAGE

```
#include <cram.h>

int cram_rt_set_rx_start_address (WORD tx_start_address);
```

tx_start_address address of first word

REMARKS

The area from offset 0000-07F00 (0-F00) is the data area on the cram board. The user is given freedom to allocate this memory range as he sees fit. This function allows the user to specify where incoming data words to the RT are to be stored. The user should be careful not to overwrite other data which he may have previously placed in the same location unless it is no longer needed.

RETURN VALUE

CRAM_SUCCESS successful

CRAN_INV_ADDR address not in range

EXAMPLE

```
        int result;

        WORD rx_start_address = Ox100;

result=cram_rt_set_rx_start_address (rx_start_address);
```

SEE ALSO

cram_bc_set_rx_start_address

cram_rt_set_start_address

DESCRIPTION

Sets starting address in data buffer for outgoing data words.

USAGE

```
#include <cram.h>

int cram_rt_set_start_address (WORD start_address);
```

REMARKS

The area from offset 0000-7F00 (0F00) is the data area on the cram board. The user is given freedom to allocate this memory range as he sees fit. This function allows the user to specify where outgoing data words to the BC are to be stored. The user should be careful not to overwrite other data which he may have previously placed in the same location unless it is no longer needed. Note that this function is called by the functions cram_rt_load_all_subaddresses and cram_rt_load_single_subaddress to load the CRAM RT with either data words for a single subaddress or data words for all subaddresses in the CRAM RT. The user would not normally have a need to call this function separately.

RETURN VALUE

CRAM_SUCCESS successful

CRAM_INV_ADDRESS address out of range

EXAMPLE

```
        int result;

        WORD start_address;

result = cram_rt_set_start_address (start_address);
```

SEE ALSO

cram_rt_load_allsubaddresses cram_rt_load_single_subaddress

cram_rt_load_single_subaddress

DESCRIPTION

Loads outgoing data words into a single subaddress within the CRAM RT.

USAGE

#include <cram.h>

int cram_rt_load_sing le_subaddresses
 (WORD start_address, BYTE subaddress, void *src)

start_address	beginning of area to be allocated for outgoing RT data.
subaddress	subaddress to load
src	pointer to source of data words to be loaded.

REMARKS

According to MIL_STD_1553 data in an RT is organized into 31 subaddresses. Command words from the BC requesting data must specify from which subaddress it should be sent. This function loads the data area of the board corresponding to a particular subaddress with 32 words the user has stored into some array called src.

This function actually does two things: first, it tells the board where the outgoing data words begin altogether (via the function cram_rt_set_start_address), i.e., it allocates a range in the data area for outgoing data words; and second, it loads 32 words (via the function cram_writebuf) into a particular subaddress. (These words will be placed at their proper location, not at start_address). The user should feel free to do these things separately if he prefers. Keep in mind that if the start address is changed, then all data which had previously been loaded will be interpreted on the basis of the new start address, counting subaddresses (groups of 32 words) from that point. Recall that the user is responsible to insure that the entire range of 31 subaddresses of 32 words is free and has not been located for another purpose such as for incoming data, or data may be overwritten.

RETURN VALUE

CRAM_SUCCESS	successful
CRAM_INV_ADDRESS	address out of range (0-7EFF (EFF))

EXAMPLE

Allocate offsetl 5O hex for outgoing data, and then load words into subaddr
MIL_WORD tx_data[32] ={aaaa, bbbb, cccc }   // 32 words *1

cram_rt_load_single subaddress (Cxl 50, 5, tx_data);

SEE ALSO

cram_rt_load_allsubaddresses   cram_rt_set_start_address
cram_rt_writebuf

cram_rt_load_single_mode_response

DESCRIPTION

Loads a single mode response for an RT.

USAGE

#include <cram.h>

WORD cram_rt_load_sing le_mode_response
(DWORD mode_response, INT num);

mode_response	response to be transmitted during a mode transmission.
num	RT address

REMARKS

Loads a single mode response to be transmitted by the board. This function can match an individual RT with a mode response.

SEE ALSO

cram_rt_load_all_mode_responses

cram_rt_load_all_mode_responses

DESCRIPTION

Loads mode responses to be transmitted.

USAGE

```
#include <cram.h>

WORD cram_rt_load_all_mode_responses (void *src);
```

REMARKS

Loads all mode responses to be transmitted by the board.

SEE ALSO

cram_rt_load_single_mode_response

cram_rt_load_all_subaddresses

DESCRIPTION

Loads outgoing data words into all subaddresses within the CRAM RT.

USAGE

```
#include <cram.h>

int cram rt load all subaddresses (WORD start_address, void *src)
```

start_address	beginning of area to be allocated for outgoing RT data.
src	pointer to source of data words to be loaded.

REMARKS

According to MIL_STD_1553 data in an RT is organized into 30 subaddresses. Command words from the BC requesting data must specify from which subaddress it should be sent. This function loads the data area of the board starting at the location start_address with words the user has stored into some array called src. The first 32 words are recognized by the board as subaddress '0', the next 32 as subaddress 'I', etc. (Note that the words placed at subaddress '0' are actually ignored by the board since the the standard dictates that setting the Subaddress Field to '0' actually signifies a Mode command.)

This function actually does two things: first, it tells the board where the outgoing data words begin altogether (via the function cram_rt_set_start_address), i.e., it allocates a range in the data area for those words; and second, it actually loads all the outgoing data words (via the function cram_writebuf). The user should feel free to do these things separately if he prefers. Keep in mind that if the start address is changed, then all data which had previously been loaded will be interpreted on the basis of the new start address, counting subaddresses (groups of 32 words) from that point.

Recall that the user is responsible to insure that the entire range of 31 subaddresses of 32 words is free and has not been allocated for another purpose such as for incoming data, or data may be overwritten.

RETURN VALUE

CRAM_SUCCESS successful

CRAM_INV_ADDRESS address out of range (0-7EFF (EFF))

EXAMPLE

```
 Load outgoing data at offset 50 hex */
 MIL_WORD tx_data[1000] =
{0,0,0  /* 32 words, ignored */
la, ib, ic, 300a, 300b, 3Cc }/* 32 words
cram_rt_load_all_subaddresses (Cxl 50, tx_data);
```

SEE ALSO

cram_rt_load_single_subaddress cram_rt_set_start_address
cram_rt_writebuf

cram_rt_get_curr_command

DESCRIPTION

Retrieves last received command word.

USAGE

#include <cram.h>

MIL_WORD cram_rt_get_curr_command (void);

REMARKS

The current command word is stored in the RT Control Block (separately from the current data words which are stored in the data buffer area). This function allows the user to obtain that command word.

RETURN VALUE

Value of command word.

EXAMPLE

MIL_WORD value;

value = cram_rt_get_curr_command 0;

SEE ALSO

cram_rt_get_prev_command

cram_rt_get_prev_command

DESCRIPTION

Retrieves previously received command word.

USAGE

#include <cram.h>

MIL_WORD cram rt get previous command (void);

REMARKS

As per MIL_STD_1553 the previously received command word is also stored in the RT Control Block. This function allows the user to obtain that command word.

RETURN VALUE

Value of command word.

EXAMPLE

```
        MIL_WORD value;

value = cram_rt_get_prev_command 0;
```

SEE ALSO

cram_rt_get_curr_command

cram_rt_get_rx_mode_data

DESCRIPTION

Retrieves incoming data word from RT following a mode command from BC.

USAGE

```
#include <cram.h>

MIL_WORD cram_rt_get_rx_mode_data(void);
```

REMARKS

In a mode command, a maximum of I data word can be sent to the RT by the BC. This function allows the user to obtain the last such data word received by the RT.

RETURN VALUE

Contents of data word.

EXAMPLE

```
        MIL_WORD rx_mode_data;

rx_mode_data = cram_rt_get_rx_mode_dataO;
```

SEE ALSO

cram_bc_get_rx_mode_data
cram_bc_mode_data

cram_rt_get_rx_word_count

DESCRIPTION

Retrieves the received word count.

USAGE

```
#include <cram.h>

WORD cram_rt_get_rx_word_count (void);
```

REMARKS

In CRAM RT mode the incoming word count (from the BC) is stored in the RT control block. This function allows the user to obtain that number.

RETURN VALUE

Number of words received.

EXAMPLE

```
        WORD count;

count = cram_rt_get_rx_word_count
```

SEE ALSO

cram_bc_get_rx_word_count

cram_rt_sam_di_datarcvd

DESCRIPTION

Disables an interrupt on data received from a specific SAM (sub-address memory 0-31).

USAGE

#include <cram.h>

int cram_rt_sam_di_datarcvd (INT sam);

sam -address memory.

REMARKS

This function disables an interrupt on data received from a specific SAM (during CRAM RT MODE)

RETURN VALUE

CRAM_SUCCESS successful

CRAM_INV_CHANNEL invalid sub-address

SEE ALSO

cram_rt_sam_di_datarcvd_all
cram_rt_sam_ei_datarcvd
cram_rt_sam_ei_datarcvd_all

cram_rt_sam_di_datarcvd_all

DESCRIPTION

Disables an interrupt on data received from any SAM (sub-address memory).

USAGE

```
#include <cram.h>

int cram_rt_sam_di_datarcvd_all (void);
```

REMARKS

This function disables an interrupt on data received from all SAMs (during CRAM RT MODE).

RETURN VALUE

CRAM_SUCCESS successful

SEE ALSO

cram_rt_sam_di_datarcvd **cram_rt_sam_ei_datarcvd**
cram_rt_sam_ei_datarcvd_all

cram_rt_sam_ei_datarcvd

DESCRIPTION

Enables an interrupt on data received from a specific SAM (sub-address memory (0 - 31).

USAGE

#include <cram.h>

int cram_rt_sam_ei_datarcvd (void);

sam sub-address memory.

REMARKS

This function enables an interrupt on data received from a specific SAM (during CRAM RT MODE)

RETURN VALUE

CRAM_SUCCESS successful

CRAM_INV_CHANNEL invalid sub-address

SEE ALSO

cram_rt_sam_di_datarcvd
cram_rt_sam_di_datarcvd_all
cram_rt_sam_ei_datarcvd_all

cram_rt_sam_ei_datarcvd_all

DESCRIPTION

Enables an interrupt on data received from all SAMs (sub-address memory).

USAGE

#include <cram.h>

int cram_rt_sam_di_datarcvd_all (void);

REMARKS

This function enables an interrupt on data received from all SAMs (during CRAM RT MODE)

RETURN VALUE

CRAM_SUCCESS successful

SEE ALSO

cram_rt_sam_di_datarcvd cram_rt_sam_di_datarcvd_all
cram_rt_sam_ei_datarcvd

cram_rt_resp_gap

DESCRIPTION

This functions returns the response time gap.

USAGE

#include <cram.h>

Int cram_rt_resp_gap ();

REMARKS

Each time a command is given, a response should be received by the remote terminal. This function returns the response gap between the end of a transmission, and the beginning of the received response.

RETURN VALUE

This function returns the value as an in (Max. value = 255), in resolution steps of 60 nanoseconds. i.e. 255 = 16 microseconds. The 9th bit indicates time overflow.

cram_get_result

DESCRIPTION

Reads the rt result respone code from the board.

USAGE

```
#include <cram.h>

int cram_get_result (void);
```

REMARKS

This function returns the rt result respone code from the CRAM board.

RETURN VALUE

RT result code

cram_get_rt_rt_rec

DESCRIPTION

Reads the BC RT_RT receive command .

USAGE

#include <cram.h>

int cram_rt_get_rt_rt_rec (void);

REMARKS

This function returns the RT_RT receive command received from a BC command.

RETURN VALUE

BC command code

cram_get_rt_rt_tra

DESCRIPTION

Reads the BC RT_RT transmit command

USAGE

#include <cram.h>

int cram_rt_get_rt_rt_rec (void);

REMARKS

This function returns the RT_RT receive command received from a BC command.

RETURN VALUE

BC command code

cram_rt_com_type

DESCRIPTION

Enquires for command type received

USAGE

#include <cram.h>

int cram_rt_com_type (void)

REMARKS

This function returns the command tyoe received from the BC .

RETURN VALUE

command type received

Multiple Remote Controller Mode Functions

(MRT)

The functions read and write into first control block defined as “mrt_ctl” in the CRAM1553.H

MRT Board Functions (rt_ctl)

cram_mrt_exec_instruction

DESCRIPTION

In Multiple RT mode, this command notify the CRAM thtat there is a new command from host.

USAGE

#include <cram.h>

```
int cram_mrt_exec_instruction
    (WORD instruction );
                  Instruction          command from host.
```

REMARKS

This function is used to enable/disable mrt operation of the

RETURN VALUE

CRAM_SUCCESS successful

CRAM_INV_CMD   invalid command

SEE ALSO

cram_mrt_ ena_RT

DESCRIPTION

In Multiple RT mode, enables operation as a particular Remote Terminal.

USAGE

```
#include <cram.h>

int cram_mrt _ena_RT
         (WORD start_address, int  RT_address, void *src);
start_address        location in memory where the RT response table is to be loaded.
RT_address           remote terminal address (0-30).
src                  Pointer to location of source of MIL-STD-1553 data
                     words to be written
```

REMARKS

In Multiple RT mode this function is used to enable operation of the unit as a particular remote terminal by setting the appropriate bit in the RT_indicator register in the MRT Control Block. In this mode, the unit can operate and respond as if it were simulating multiple RT's simultaneously; each with its own RT address. Each bit in the MRT_indicator register corresponds to a particular RT address.

RETURN VALUE

CRAM_SUCCESS successful

CRAM_INV_ADD invalid BUFFER address – must be a multiple of 2048 and lower than 28672 (2 power 14)

CRAM_INV_SIZE – maximum RTs (15) already configured

CRAM_INV_CHANNEL invalid or unconfigured remote terminal

SEE ALSO

cram_mrt_dis_RT

cram_mrt_dis_all

cram_mrt_dis_RT

DESCRIPTION

In Multiple RT mode, disables operation of a particular Remote Terminal.

USAGE

```
        #include <cram.h>

int cram_mrt_dis_RT (int RT_address);
```

address	**remote terminal address (0-30)**

REMARKS

In Multiple RT mode this function is used to disable operation of a particular remote terminal by clearing the appropriate bit in the MRT_indicator register in the MRT Control Block.

RETURN VALUE

CRAM_SUCCESS successful

CRAM_INV_CHANNEL invalid or unconfigured remote terminal

SEE ALSO

cram_mrt_ena_RT
cram_mrt_dis_all

cram_mrt_dis_all

DESCRIPTION

In Multiple RT mode disables operation of all Remote Terminals.

USAGE

```
#include <cram.h>

int cram_mrt_dis_all (void);
```

REMARKS

In Multiple RT mode, this function is used to disable operation of any or all 31

Remote Terminals. It clears all bits in the MRT_indicator register in the MRT Control Block.

RETURN VALUE

CRAM_SUCCESS successful

SEE ALSO

cram_mrt_load_ena_RT
cram_mrt_dis_RT

cram_total_mrt_set_address

DESCRIPTION

In Multiple RT mode, checks for the total number of enabled RT's.

USAGE

#include <cram.h>

int cram_total_mrt_set_address (void);

REMARKS

In Multiple RT mode this function is used to querry a CRAM board to see how many RT's are enabled at once. This information is stored in the MRT Control Block in the total_rt_sel register.

RETURN VALUE

integer Number of enabled RT's

SEE ALSO

cram_mrt_RT_enabled

cram_mrt_remote_enable

DESCRIPTION

In Multiple RT mode, checks if a particular RT is enabled.

USAGE

```
#include <cram.h>

int cram_mrt_total_enabled (int RT_address);
```

address remote terminal address (0-30).

REMARKS

In Multiple RT mode this function is used to querry a CRAM board to see id a particular RT is enabled. This information is stored in the MRT Control Block in the MRT_indicator register. Each bit in the MRT_indicator register corresponds to a particular RT address. Bit 0 represents RT address 0, bit 1 represents RT address 1, etc.

RETURN VALUE

YES RT is enabled

No RT is disabled

SEE ALSO

cram_mrt_total_enabled

cram_mrt_remote_disable

DESCRIPTION

In Multiple RT mode, checks if a particular RT is disabled.

USAGE

```
#include <cram.h>

int cram_mrt_total_enabled (int RT_address);
```

address remote terminal address (0-30).

REMARKS

In Multiple RT mode this function is used to querry a CRAM board to see id a particular RT is disabled. This information is stored in the MRT Control Block in the MRT_indicator register. Each bit in the MRT_indicator register corresponds to a particular RT address. Bit 0 represents RT address 0, bit 1 represents RT address 1, etc.

RETURN VALUE

YES RT is enabled

No RT is disabled

SEE ALSO

cram_mrt_remote_enabled

cram_mrt_ei_RT

DESCRIPTION

Enables interrupt on new data received for a specific remote terminal address.

USAGE

#include <cram.h>

int cram_mrt_ei_datatrcvd (remote);

remote remote terminal address (0-30)

REMARKS

This function sets the appropriate bit in the mrt_interrupt register of the RT Control Block which will cause the board to issue an IRQ (interrupt request) when one or more new data word(s) have been received for the specified remote terminal address.

RETURN VALUE

CRAM_SUCCESS	**successful**
CRAM_INV_CHANNEL	**invalid remote terminal address**

EXAMPLE

```
/* Set interrupt for remote terminal 25 */

cram_mrt_ei_datarcvd (25);
```

SEE ALSO

cram_mrt_di_RT

cram_mrt_di_RT

DESCRIPTION

Disables interrupt on new data received for a specific remote terminal address.

USAGE

#include <cram.h>

int cram_mrt_di_RT (remote);

remote emote terminal address (0-30)

REMARKS

This function resets (sets to '0') the appropriate bit in the mrt_interrupt register of the RT Control Block which will cause the board to cease issuing an IRQ (interrupt request) when one or more new word(s) have been received for the specified remote terminal address.

RETURN VALUE

CRAM_SUCCESS successful

CRAM_INV_CHANNEL invalid remote terminal address

EXAMPLE

```
/* Reset interrupt for remote terminal 25 */

cram_mrt_di_RT  (25);
```

SEE ALSO

cram_mrt_ei_RT

cram_mrt_int_ena_all

DESCRIPTION

Enables interrupt on any new data received.

USAGE

```
#include <cram.h>

int cram_mrt_int_ena_all (void);
```

REMARKS

This function sets all bits in the mrt_interrupt register of the RT Control Block which will cause the board to issue an IRQ (interrupt request) when one or more new word(s) have been received for any remote terminal address.

RETURN VALUE

CRAM_SUCCESS successful

SEE ALSO

cram_mrt_ei_RT **cram_mrt_di_RT**
cram_mrt_int_dis_all

cram_mrt_int_dis_all

DESCRIPTION

Disables interrupt on any new data received.

USAGE

```
#include <cram.h>

int cram_mrt_int_dis_all;
```

REMARKS

This function resets (sets to '0') all bits in the mrt_interrupt register of the RT Control Block which will cause the board to cease sending an IRQ (interrupt request) when one or more new word(s) have been received for any remote terminal address.

RETURN VALUE

CRAM_SUCCESS successful

SEE ALSO

cram_mrt_ei_RT	cram_mrt_di_RT
cram_mrt_int_dis_all	cram_mrt_int_ena_all

cram_mrt_intr

DESCRIPTION

Disables interrupt on any new data received.

USAGE

```
#include <cram.h>

int cram_mrt_intr;
```

REMARKS

This function returns Rts interrupts enable/disable register from the MRT Control Segment.

RETURN VALUE

DWORD – "mrt_int_ena_dis" register

SEE ALSO

cram_mrt_ei_RT	cram_mrt_di_RT
cram_mrt_int_dis_all	cram_mrt_int_ena_all

cram_mrt_stat

DESCRIPTION

Disables interrupt on any new data received.

USAGE

#include <cram.h>

REMARKS

This function returns Rts enable/disable register status from the MRT Control Segment.

RETURN VALUE

DWORD – "mrt_ena_dis" register

SEE ALSO

cram_mrt_ei_RT	**cram_mrt_di_RT**
cram_mrt_int_dis_all	**cram_mrt_int_ena_all**

The functions read and write control block defined as “bm_ctl” in the CRAM1553.H

BM Board Functions (bm_ctl)

cram_bm_select_channel...
cram_bm_set_start_address
cram_bm_block_address..
cram_bm_block_numbers..
cram_bm_set_address ....

// check functions
cram_bm_check_block_int
cram_bm_check_msg_rcvd
cram_bm_check_half_buff
cram_bm_check_full_buff
cram_bm_check_block_add
cram_bm_confirm_int_msg_rcvd
cram_bm_confirm_int_half_buff
cram_bm_confirm_int_full_buff

// get commands
cram_bm_get_msg_ptr...
cram_bm_get_command1_field
cram_bm_get_command2_field
cram_bm_get_word_count_field
cram_bm_get_data_word
cram_bm_get_data_start_address
cram_bm_get_time_tag_field
cram_bm_get_resp

//_bm_clear functions
cram_bm_clr_msg_ptr
cram_bm_clr_command1_field
cram_bm_clr_command2_field
cram_bm_clr_word_count_field
cram_bm_clr_data_word
cram_bm_clr_all_data_words
cram_bm_clr_time_tag_field

// enable interrupts
cram_bm_ei_all
cram_bm_ei_msg_rcvd
cram_bm_ei_half_buff
cram_bm_remote_enable

cram_bm_ei_msg_rcvd
cram_bm_ei_half_buff
cram_bm_ei_full_buff
cram_bm_ei_rt_datarcvd
cram_bm_remote_enable
cram_bm_remote_ena_all
cram_bm_remote_ena_all

// disable functions
cram_bm_di_all
cram_bm_di_msg_rcvd
cram_bm_di_half_buff
cram_bm_di_full_buff
cram_bm_remote_disable
cram_bm_remote_disable
cram_bm_remote_dis_all
cram_bm_di_block_add
cram_bm_di_rt_datarcvd

cram_bm_select_channel

DESCRIPTION

Selects either Channel A or B for Bus Monitor Operation.

USAGE

```
#include <cram.h>

int cram_bm_select_channel (BYTE channel)
```

channel either A (‘0’) or B (‘1’)

REMARKS

The CRAM system can transmit on one of two channels at any time (but not both simultaneously). This function selects either A or B.

RETURN VALUE

CRAM SUCCESS successful

CRAM_INV_CHANNEL invalid channel

EXAMPLE

```
cram_bm_select_channel (0):
```

SEE ALSO

cram_bc_exec_instruction
cram_rt_select_channel

cram_bm_set_start_address

DESCRIPTION

Sets address of first message block.

USAGE

```
#include <cram.h>

int cram_bm_set_start_address (WORD address);
```

address first block address

REMARKS

The CRAM system stores up to 300 incoming messages in consecutive blocks of 80 bytes each. This function sets the address in the data area of the beginning of the first block. The function checks that the user has allowed enough space so that the entire 300 messages will fit inside the data area.

RETURN VALUE

CRAM_SUCCESS successful

CRAM_INV_SIZE invalid memory address for 300 buffers

EXAMPLE

```
int result;

result = cram_bm_set_start_address(100);
```

cram_bm_block_add

DESCRI PTION

Stores block number into register

USAGE

#include <cram.h>

int cram_bm_block_add (word number) number block address specified

REMARKS

During_bm_mode whenever data is stored into a specified,block, it sets bit 4 on the_bm_receive indicator. If bit 4 of the_bm_interrupt receive register is set, an interrupt will be supplied by the board to the host.

RETURN VALUE

CRAM_SUCCESS successful

SEE ALSO

cram_bm_ei_block_add
cram_bm_di_block_add
cram_bm_check_block_add
cram_bm_check_block_add_int

cram_bm_block_numbers

DESCRIPTION

Select the number of blocks to be stored during_bm_mode

USAGE

#include <cram.h>

int cram_bm_block_numbers (word number)

REMARKS

This function specifies the number of blocks to be used during_bm_mode. The maximum value may vary from 50 to 300 buffers depending on the board.

RETURN VALUE

CRAM_SUCCESS successful

SEE ALSO

cram_bm_block_add
cram_bm_ei_block_add
cram_bm_check_block_add
cram_bm_check_block_add_int

cram_bm_check_block_int

DESCRIPTION

Tests the BLOCK NUMBER bit in the INT_INDICATOR register for its current value.

USAGE

```
#include <cram.h>

int cram_bm_check_block_int (void);
```

REMARKS

Tests the BLOCK NUMBER bit in the INT_INDICATOR register for its current value.

RETURN VALUE

YES if BUFFER ADD bit is set

NO if BUFFER ADD bit is 0

cram_bm_check_msg_rcvd

DESCRIPTION

Checks whether the CRAM_bm_has received a new message.

USAGE

```
#include <cram.h>

int_cram_bm_check_msg_rcvd(void);
```

REMARKS

This function polls the CRAM_bm_one time as to whether there has been an occurrence of a newly received message by checking the appropriate bits in the rec_indicator register in the_bm_control block.

RETURN VALUE

YES (1)

NO (0)

SEE ALSO

cram_bm_check_msg_rcvd
cram_bm_check_half_buff
cram_bm_check_full_buff

cram_bm_check_half_buff

DESCRIPTION

Checks whether half of the CRAM_bm_message blocks have been filled.

USAGE

#include <cram.h>

int_cram_bm_check_half_buff(void);

REMARKS

This function polls the CRAM_bm_one time as to whether the first 15 message blocks have been filled by checking the appropriate bits in the rec_indicator register in the_bm_control block.

RETURN VALUE

YES (1)

NO (0)

SEE ALSO

cram_bm_check_msg_rcvd
cram_bm_check_half_buff
cram_bm_check_full_buff

cram_bm_check_full_buff

DESCRIPTION

Checks whether all of the CRAM_bm_message blocks have been filled.

USAGE

#include <cram.h>

Int_cram_bm_check_full_buff(void);

REMARKS

This function polls the CRAM_bm_one time as to whether all 300 message blocks have been filled by checking the appropriate bits in the rec_indicator register in the_bm_control block.

RETURN VALUE

YES (1)

NO (0)

SEE ALSO

cram_bm_check_msg_rcvd
cram_bm_check_half_buff
cram_bm_check_full_buff

cram_bm_check_block_add

DESCRIPTION

Checks whether the block number specified has data stored.

USAGE

#include <cram.h>

int cram_bm_check_block_add (void)

REMARKS

This function polls the CRAM_bm_one time as to whether the block defined in the BM_ADD register has been filled by checking the appropriate bit in rec_indicator register in the_bm_control block.

RETURN VALUE

YES (1)

NO (0)

SEE ALSO

cram_bm_block_add
cram_bm_ei_block_add
cram_bm_check_block_add
cram_bm_check_block_add_int

cram_bm_confirm_int_msg_rcvd

DESCRIPTION

Confirms that an interrupt has occurred because the CRAM_bm_has received a new message.

USAGE

#include <cram.h>

int_crambm_confirm_int_msg_rcvd(void);

REMARKS

The CRAM_bm_can be set to signal the user when certain conditions have been met in order that the user's software may branch to a desired routine to handle the event. This function confirms that the cause of the interrupt was the occurrence of a newly received message by checking the appropriate bits in the int indicator register in the_bm_control block.

RETURN VALUE

YES (1)

NO (0)

SEE ALSO

cram_bm_confirm_int_msg_rcvd
cram_bm_confirm_int_half_buff
cram_bm_confirm_int_full_buff

cram_bm_confirm_int_half_buff

DESCRIPTION

Confirms that an interrupt has occurred because half of the CRAM BM message blocks have been filled.

USAGE

#include <cram.h>

int_cram_bm_confirm_int_half_buff(void);

REMARKS

The CRAM_bm_can be set to signal the user when certain conditions have been met in order that the users software may branch to a desired routine to handle the event. This function confirms that the cause of an interrupt was the fact that the first 15 message blocks have been filled by checking the appropriate bits in the int indicator register in the_bm_control block.

RETURN VALUE

YES (1)

NO (0)

SEE ALSO

cram_bm_confirm_int_msg_rcvd
crambm_confirm_int_half_buff
cram_bm_confirm_int_full_buff

cram_bm_confirm_int_full_buff

DESCRIPTION

Enables an interrupt whenever all of the CRAM_bm_message blocks have been filled.

USAGE

```
#include <cram.h>

int_cram_bm_confirm_int_full_buff (void);
```

REMARKS

The CRAM_bm_can be set to signal the user when certain conditions have been met in order that the user's software may branch to a desired routine to handle the event. This function confirms that the cause of an interrupt was the fact that all 300 message blocks have been filled by checking the appropriate bits in the inL indicator register in the_bm_control block.

RETURN VALUE

YES (1)

NO (0)

SEE ALSO

cram_bm_confirm_int_msg_rcvd
cram_bm_confirm_int_half_buff
cram_bm_confirm_int_full_buff

cram_bm_get_msg_ptr

DESCRIPTION

Retrieves current value of message index.

USAGE

```
#include <cram.h>

BYTE cram_bm_get_msg_ptr (void);
```

REMARKS

The CRAM Bus Monitor can track up to 300 messages simultaneously. They are stored starting at the address in the data area which the user has allocated (via the function cram_bm_seLstart_address), and are stored consecutively with 80 bytes per message. After 300 messages have been received, the next message is stored back in the first location. The message index is a number between 0 and 299 which tracks the storage location of the last message. This function returns the current value of the message index.

RETURN VALUE

index value of index of most recent message

EXAMPLE

```
        BYTE index;

index = cram_bm_get_msg_ptr
```

SEE ALSO

cram_bm_clr_msg_ptr

cram_bm_get_command1_field

DESCRIPTION

Retrieves the Commandi Field of a given message block.

USAGE

#include <cram.h>

MIL_WORD cram_bm_get_command1_field (int index);

index Message Block Index (0-299)

REMARKS

The CRAM_bm_stores up to 300 messages consecutively each in its own message block. This function returns the Commandi Field of a given block which contains the only Command word sent by the BC in a regular message transfer, or the first Command word (to the Receiving RT) in an RT-RT transfer.

RETURN VALUE

Commandi Field

EXAMPLE

```
        /* Return Command1 Field of most recent message *1

int index = (int) cram_bm_get msg_ptr ( );
MIL_WORD command = cram_bm_get_command1 field (index);
```

SEE ALSO

cram_bm_get_msg_ptr

cram_bm_get_command2_field

DESCRIPTION

Retrieves the Command2 Field of a given message block.

USAGE

```
#include <cram.h>

MIL_WORD cram_bm_get_command1_field (int index);
```

index Message Block Index (0-299)

REMARKS

The CRAM_bm_stores up to 300 messages consecutively each in its own message block. This function returns the Command2 field of a given block which contains the second Command word (to the Transmitting RT) in the case of an RT-RT transfer, or 0 otherwise.

RETURN VALUE

Command2 Field

EXAMPLE

```
/* Return Command2 Field of most recent message I

int index = (int) cram_bm_get msg_ptrO;
MIL_WORD command1 = cram_bm_get_command1 field (index);
```

SEE ALSO

cram_bm_get_msg_ptr

cram_bm_get_word_count_field

DESCRIPTION

Retrieves the Word Count Field of a given message block.

USAGE

```
#include <cram.h>

WORD cram_get_word_count_field (Int index);
```

index **message block index**

REMARKS

The CRAM_bm_stores up to 300 messages consecutively each in its own message block. This function returns the Word Count Field of a given block which contains the number of Data words in the message. Note that the Data words could have been sent by the BC or by the RT. The way to determine their source is by checking the Type Field of that message block.

RETURN VALUE

Word Count Field

EXAMPLE

```
/ Return Word Count Field of most recent message I

int index = (int) cram_bm_get msg_ptrO;
int count = cram_bm_get_word_count_field (index);
```

SEE ALSO

cram_bm_get_msg_ptr

cram_bm_get_data_word

DESCRIPTION

Retrieves a specific Data word from a given message block

USAGE

```
#include <cram.h>

MIL_WORD cram_bm_get_data_word (int index, int word_num);
```

index	message block index
word_	num Data word number

REMARKS

The CRAM_bm_stores up to 300 messages consecutively each in its own message block. This function returns a single Data word (specified by the parameter word_num) of a given block (specified by the parameter index). Note that the Data words could have been sent by the BC or by the RT. The way to determine their source is by checking the Type Field of that message block.

RETURN VALUE

Data[word_num]	The requested Data word

EXAMPLE

```
/* Return first Data word of most recent message */

 int word_num = 0;

int index = (int) cram_bm_get msg_ptr ;
MIL_WORD data = cram_bm_get_data_word (index, word_num);
```

SEE ALSO

cram_bm_get_msg_ptr

cram_bm_get_data_start_address

DESCRIPTION

Gets address of first data word.

USAGE

```
#include <cram.h>

WORD cram_bm_get_data_start_address (int index)
```

index message block index

REMARKS

This function is provided to make it easier for applications programmers to display all Data words with a single function call using pointers, rather than retrieving them one at at time with the cram_bm_geLdata_word function.

NOTE: This function may not be compatible with all implementations of C.

RETURN VALUE

The address offset of the first Data word in the given message block.

EXAMPLE

```
        int tmp, index = cram_bm_get_msg_ptr;

 cram_set_board(d000);
 for (tmp = 0; tmp < (cram_bm_get_word_count_field(index)); tmp++)
        printf (“ %4x”, * (WORD *) ((char *) _CRAM_BOARD +
                crambm_get_data_start_address(i ndex) + tmp * 2));
```

SEE ALSO

cram_bm_get_word_count_field
cram_bm_get_data_word
cram_bm_get_msg_ptr

cram_bm_get_time_tag_field

DESCRIPTION

Retrieves the Time Tag Field of a given message block.

USAGE

```
#include <cram.h>

DWORD cram_bm_get_time_tag_field (int index);
```

index Message Block Index (0-299)

REMARKS

The CRAM_bm_stores up to 300 messages consecutively each in its own message block. This function returns the Time Tag Field of a given block which contains a 4 byte word equal to the value of the CRAM System's clock at the time of message reception.

RETURN VALUE

Time Tag Field

EXAMPLE

```
/* Return Time Tag Field of most recent message /

int index = (int) cram_bm_get msg_ptr
DWORD rcv_time = cram_bm_get_time_tag_field (index);
```

SEE ALSO

cram_bm_get_msg_ptr

cram_bm_get_resp

DESCRIPTION

/* Returns the RT response time from the BC command */

USAGE

#include <cram.h>

int cram_bm_get_resp (INT n);

INT n_bm_block pointer (0 - 300)

REMARKS

The timing gap between the BC command and the RT response is returned by this function. The return value (1-255) is in 60 nSec resolution.

RETURN VALUE

Resopnse Gap 1-255 (60 nSec resolution) a maximum value of 16 uSec. The 9th bit indicates overflow.

cram_bm_clr_msg_ptr

DESCRIPTION

Clears (sets to ‘0’) the message index.

USAGE

#include <cram.h>

int cram_bm_clr_msg_ptr (void);

REMARKS

The CRAM Bus Monitor can track up to 300 messages simultaneously. They are stored starting at the address in the data area which the user has allocated (via the function cram_bm_seLstart_address), and are stored consecutively with 80 bytes per message. After 300 messages have been received, the next message is stored back in the first location. The message index is a number between 0 and 299 which tracks the storage location of the last message. This function resets the message index to 0 (actually to 299 since the index is incremented before the message is stored) so that the next message will be placed at the first location.

RETURN VALUE

CRAM_SUCCESS successful

EXAMPLE

int result;

result = cram_bm_clr_msg_ptr

SEE ALSO

cram_bm_get_msg_ptr

cram_bm_clr_command1_field

DESCRIPTION

Clears the Commandi field of a given message block.

USAGE

```
#include <cram.h>

int cram_bm_clr_command1_field (int index);
```

index Message Block Index (0-299)

REMARKS

The CRAM_bm_stores up to 300 messages consecutively each in its own message block. This function clears the Command I field of a given block.

RETURN VALUE

CRAM_SUCCESS successful

CRAM_INV_PARAM invalid message block index (not 0-299)

EXAMPLE

```
/* Clear Commandi field of Message Block 5 */

int result = cram_bm_clr_command1_field (5);
```

SEE ALSO

cram_bm_get_command1_field

cram_bm_clr_command2_field

DESCRIPTION

Clears the Command2 field of a given message block.

USAGE

```
#include <cram.h>

int cram_bm_clr_command2_field (int index);
```

index Message Block Index (0-299)

REMARKS

The CRAM_bm_stores up to 300 messages consecutively each in its own message block. This function clears the Command2 field of a given block.

RETURN VALUE

CRAM_SUCCESS successful

CRAM_INV_PARAM invalid message block index (not 0-299)

EXAMPLE

```
/* Clear Command2 field of Message Block 5 /

int result = cram_bm_clr_command2_field (5);
```

SEE ALSO

cram_bm_get_command2_field

cram_bm_clr_word_count_field

DESCRIPTION

Clears the Word Count field of a given message block.

USAGE

```
#include <cram.h>

int cram_bm_clr_word_count_field (int index);
```

index Message Block Index (0-299)

REMARKS

The CRAM_bm_stores up to 300 messages consecutively each in its own message block. This function clears the Word Count field of a given block.

RETURN VALUE

CRAM_SUCCESS successful

CRAM_INV_PARAM invalid message block index (not 0-299)

EXAMPLE

```
/* Clear Word Count field of Message Block 5 */

int result = cram_bm_clr_word_count_field (5);
```

SEE ALSO

cram_bm_get_word_count_field

cram_bm_clr_data_word

DESCRIPTION

Clears a specific Data word from a given message block.

USAGE

```
#include <cram.h>

int cram_bm_clr_data_word (int index, int word_num);
```

index	Message Block Index (0-299)
word_num	Data word number

REMARKS

The CRAM_bm_stores up to 300 messages consecutively each in its own message block. This function clears a single Data word (specified by the parameter word_num) of a given block (specified by the parameter index).

RETURN VALUE

CRAM_SUCCESS successful

CRAM_INV_PARAM invalid message block index (not 0-299)

EXAMPLE

```
/* Clear Data Word 2 of Message Block 5 */

int result = cram_bm_clr_data_word (5, 2);
```

SEE ALSO

cram_bm_get_data_word
cram_bm_clr_all_data_words

cram_bm_clr_all_data_words

DESCRIPTION

Clears all Data words of a given message block.

USAGE

```
#include <cram.h>

int cram_bm_clr_all_data_words (int index);
```

index Message Block Index (0-299)

REMARKS

The CRAM_bm_stores up to 300 messages consecutively each in its own message block. This function clears all Data words in a given block.

RETURN VALUE

CRAM_SUCCESS successful

CRAM_INV_PARAM invalid message block index (not 0-299)

EXAMPLE

```
I Clear all Data Words in Message Block 5 */

int result = cram_bm_clr_all_data_words (5);
```

SEE ALSO

cram_bm_clr_data_word　　**cram_bm_clr_word_count**

cram_bm_clr_time_tag_field

DESCRIPTION

Clears the Time Tag field of a given message block.

USAGE

```
#include <cram.h>

int cram_bm_clr_time_tag_field (int index);
```

index Message Block Index (0-299)

REMARKS

The CRAM_bm_stores up to 300 messages consecutively each in its own message block. This function clears the Time Tag field of a given block.

RETURN VALUE

CRAM_SUCCESS successful

CRAM_INV_PARAM invalid message block index (not 0-299)

EXAMPLE

```
/* Clear Time Tag field of Message Block 5 */

int result = cram_bm_clr_time_tag_field (5);
```

SEE ALSO

cram_bm_get_time_tag_field

`

cram_bm_ei_all

DESCRIPTION

Enables an interrupt for any of the following events: the CRAM_bm_ has received a new message; half the CRAM_bm_message blocks have been used; or all of the CRAM_bm_message blocks have been used.

USAGE

#include <cram.h>

int_cram_bm_ei_all (void);

REMARKS

The CRAM_bm_can be set to signal the user when any of the above conditions have been met in order that the user's software may branch to a desired routine to handle the event. This function enables an interrupt upon the occurrence of any of the conditions by setting the appropriate bits in the interrupt_set_mask register in the_bm_control block.

RETURN VALUE

CRAM_SUCCESS successful

SEE ALSO

cram_bm_ei_msg_rcvd
cram_bm_ei_half_buff
cram_bm_ei full buff
cram_bm_di all

cram_bm_ei_msg_rcvd

DESCRIPTION

Enables an interrupt whenever the CRAM_bm_has received a new message.

USAGE

#include <cram.h>

int_cram_bm_ei_msg_rcvd(void);

REMARKS

The CRAM_bm_can be set to signal the user when certain conditions have been met in order that the user's software may branch to a desired routine to handle the event. This function enables an interrupt upon the occurrence of a newly received message by setting the appropriate bit in the interrupt_seLmask register in the_bm_control block.

RETURN VALUE

CRAM_SUCCESS successful

SEE ALSO

cram_bm_ei all
cram_bm_ei_half_buff
cram_bm_ei full buff
cram_bm_di_msg_rcvd

cram_bm_ei_half_buff

DESCRIPTION

Enables an interrupt whenever half of the CRAM_bm_message blocks have been filled.

USAGE

#include <cram.h>

int_cram_bm_ei_half_buff(void);

REMARKS

The CRAM_bm_can be set to signal the user when certain conditions have been met in order that the user's software may branch to a desired routine to handle the event. This function enables an interrupt when the first 15 message blocks have been filled by setting the appropriate bits in the interru pt set mask register in the_bm_control block.

RETURN VALUE

CRAM_SUCCESS successful

SEE ALSO

cram_bm_ei all
cram_bm_ei_msg_rcvd
cram_bm_ei_full_buff
cram_bm_di_half_buff

cram_bm_ei_full buff

DESCRIPTION

Enables an interrupt whenever all of the CRAM_bm_message blocks have been filled.

USAGE

```
#include <cram.h>

int_cram_bm_ei_ful l_buff(void);
```

REMARKS

The CRAM_bm_can be set to signal the user when certain conditions have been met in order that the user's software may branch to a desired routine to handle the event. This function enables an interrupt when all 300 message blocks have been filled by setting the appropriate bits in the interrupL set mask register in the_bm_control block.

RETURN VALUE

CRAM_SUCCESS successful

SEE ALSO

cram_bm_ei all
cram_bm_ei_msg_rcvd
cram_bm_ei_half_buff
cram_bm_di_full buff

cram_bm_ei_block_add

DESCRIPTION

Enables an interrupt on data stored into block number predefined

USAGE

#include <cram.h>

int cram_bm_ei_block_add (void)

REMARKS

This function sets bit 4 of the_bm_receive interrupt register. It enables the system to send an interrupt request when data is stored into the block number set by "cram_bm_block_add".

RETURN VALUE

CRAM_SUCCESS successful

SEE ALSO

cram_bm_block_add
cram_bm_di_block_add
cram_bm_check_block_add
cram_bm_check_block_add_int

cram_bm_ei_rt_datarcvd

DESCRIPTION

Enables an interrupt on new data received from a specific remote terminal address.

USAGE

#include <cram.h>

int cram_bm_ei_rt_datatrcvd (remote);

remote remote terminal address (0-30)

REMARKS

This function sets the appropriate bit in the mrt_interrupt register of the_bm_Control Block which will cause the board to issue an IRQ (interrupt request) when one or more new data word(s) have been received from the specified remote terminal address.

RETURN VALUE

CRAM_SUCCESS successful

CRAM_INV_CHANNEL invalid remote terminal address

EXAMPLE

I Set an interrupt for remote terminal 25 */

cram_bm_ei_rt_datarcvd (25);

SEE ALSO

cram_bm_ei_rt_datarcvd_all
cram_bm_di_rt_datarcvd_all
cram_bm_di_rt_datarcvd

cram_bm_remote_enable

DESCRIPTION

In_bm_mode, enables a store operation for a particular Remote Terminal.

USAGE

```
#include <cram.h>

int cram_bm_remote_enable (BYTE remote);
```

remote remote terminal address (0-31)

REMARKS

In_bm_mode this function is used to enable a store operation, storing information from a particular remote terminal by setting the appropriate bit in the MRT_mask register in the_bm_Control Block. Each bit in the MRT_mask register corresponds to a particular RT address.

RETURN VALUE

CRAM_SUCCESS successful

CRAM_INV_CHANNEL invalid or unconfigured remote terminal

SEE ALSO

cram_bm_remote_disable
cram_bm_remote_ena_all
cram_bm_remote_dis_all

cram_bm_remote_ena_all

DESCRIPTION

In_bm_mode enables storage operations for all 32 Remote Terminals.

USAGE

#include <cram.h>

int cram_remote_ena_all (void);

REMARKS

In_bm_mode, this function is used to enable storage operations for all 32 Remote

Terminals, simultaneously. It sets all bits in the MRT_mask register in the BM Control Block. (See cram_remote_enable.)

RETURN VALUE

CRAM_SUCCESS successful

SEE ALSO

cram_bm_remote_enable
cram_bm_remote_disable
cram_bm_remote_dis_all

cram_bm_di_all

DESCRIPTION

Disables all CRAM_bm_interrupts.

USAGE

#include <cram.h>

int_cram_bm_di_all(void);

REMARKS

The CRAM_bm_can be set to signal the user when certain conditions have been met in order that the user's software may branch to a desired routine to handle the event. This function disables an interrupt upon the occurrence of any of the conditions by clearing the appropriate bits in the interrupL set mask register in the_bm_control block.

RETURN VALUE

CRAM_SUCCESS successful

SEE ALSO

cram_bm_di_msg_rcvd
cram_bm_di_half_buff
cram_bm_di_full_buff
cram_bm_ei_all

cram_bm_di_msg_rcvd

DESCRIPTION

Disables an interrupt whenever the CRAM_bm_has received a new message.

USAGE

#include <cram.h>

int_cram_bm_di_msg_rcvd(void);

REMARKS

The CRAM_bm_can be set to signal the user when certain conditions have been met in order that the user's software may branch to a desired routine to handle the event. This function disables an interrupt upon the occurrence of a newly received message by clearing the appropriate bits in the interrupt_set_mask register in the_bm_control block.

RETURN VALUE

CRAM_SUCCESS successful

SEE ALSO

cram_bm_di all
cram_bm_di_half_buff
cram_bm_di_full_buff
cram_bm_ei_msg_rcvd

cram_bm_di_half_buff

DESCRIPTION

Disables an interrupt whenever half of the CRAM_bm_message blocks have been filled.

USAGE

#include <cram.h>

int_cram_bm_di_half_buff(void);

REMARKS

The CRAM_bm_can be set to signal the user when certain conditions have been met in order that the user's software may branch to a desired routine to handle the event. This function disables an interrupt from occurring when the first 15 message blocks have been filled by clearing the appropriate bits in the interrupt set mask register in the_bm_control block.

RETURN VALUE

CRAM_SUCCESS successful

SEE ALSO

cram_bm_di all
cram_bm_di_msg_rcvd
cram_bm_di_full_buff
cram_bm_ei_half_buff

cram_bm_di_full_buff

DESCRIPTION

Disables an interrupt whenever all of the CRAM_bm_message blocks have been filled.

USAGE

```
#include <cram.h>

int_cram_bm_di_full_buff (void);
```

REMARKS

The CRAM_bm_can be set to signal the user when certain conditions have been met in order that the user's software may branch to a desired routine to handle the event. This function disables an interrupt from occurring when all 300 message blocks have been filled by clearing the appropriate bits in the interruptset mask register in the_bm_control block.

RETURN VALUE

CRAM_SUCCESS successful

SEE ALSO

cram_bm_di_all
cram_bm_di_msg_rcvd
cram_bm_di_half_buff
cram_bm_ei_full_buff

cram_bm_remote_disable

DESCRIPTION

In_bm_mode, disables a store operation for a particular Remote Terminal.

USAGE

#include <cram.h>

int crambm_remotedisable (char remote);

remote remote terminal address (0-31)

REMARKS

In_bm_mode this function is used to disable a store operation for a particular remote terminal by clearing the appropriate bit in the MRT_mask register in the_bm_Control Block. (See cram_remote_enable.)

RETURN VALUE

CRAM_SUCCESS successful

CRAM_I NV_C HANNEL invalid or unconfigured remote terminal

SEE ALSO

cram_bm_remote_enable
cram_bm_remote_ena_all
cram_bm_remote_dis_all

cram_bm_remote_dis_all

DESCRIPTION

In_bm_mode, disables storage operations for all 32 Remote Terminals.

USAGE

#include <cram.h>

int cram_bm_remote_dis_all (void);

REMARKS

In_bm_mode, this function is used to disable storage operations for all 32 Remote

Terminals, simultaneously. It clears all bits in the MRT_mask register in the BM Control Block. (See cram_remote_enable.)

RETURN VALUE

CRAM_SUCCESS successful

SEE ALSO

cram_bm_remote_enable
cram_bm_remote_d isable
cram_bm_remote_ena_all

cram_bm_di_block_add

DESCRIPTION

Disables an interrupt on data stored into predefined block number

USAGE

#include <cram.h>

int cram_bm_di_block_add (void)

REMARKS

This function clears bit 4 of the_bm_receive interrupt register. It disables the system from sending an interrupt request when data is stored into the block number set by "cram_bm_block_add".

RETURN VALUE

CRAM_SUCCESS successful

SEE ALSO

cram_bm_block_add
cram_bm_ei_block_add
cram_bm_check_block_add
cram_bm_check_block_add_int

cram_bm_di_rt_datarcvd

DESCRIPTION

Disables an interrupt on new data received from a specific remote terminal address.

USAGE

```
#include <cram.h>

int cram_bm_di_rt_datarcvd (remote);
```

remote remote terminal address (0-300)

REMARKS

This function resets (sets to '0') the appropriate bit in the mrt_interrupt register of the_bm_Control Block which will cause the board to cease issuing an IRQ (interrupt request) when one or more new word(s) have been received from the specified remote terminal address.

RETURN VALUE

CRAM_SUCCESS successful

CRAM_I NV_CHANNEL invalid remote terminal address

EXAMPLE

```
/* Reset interrupt for remote terminal 25 */

cram_bm_di_rt_datarcvd (25);
```

SEE ALSO

cram_bm_ei_rt_datarcvd_all
cram_bm_di_rt_datarcvd_all
cram_bm_ei_rt_datarcvd

cram_bm_di_rt_all

DESCRIPTION

Disables the_bm_globalIRQ enable register.

USAGE

#include <cram.h>

int_cram_bm_di_rt_all(void);

REMARKS

The CRAM_bm_can be set to signal the user when certain conditions have been met in order that the user's software may branch to a desired routine to handle the event. This function disables the global_bm_IRQ enable register, subsequently disabling all interrupts. This function does not effect the individualIRQ register bits set in the interru pt set mask register in the_bm_control block.

RETURN VALUE

CRAM_SUCCESS successful

SEE ALSO

cram_bm_ei_rt_all

cram_bm_ei_rt_all

DESCRIPTION

Enables the_bm_global_IRQ enable register.

USAGE

#include <cram.h>

int_cram_bm_ei_rt_all(void);

REMARKS

The CRAM_bm_can be set to signal the user when certain conditions have been met in order that the users software may branch to a desired routine to handle the event. This function enables the global_bm_IRQ enable register, subsequently enabling all interrupts. This function does not effect the individualIRQ register bits set in the interrupt set mask register in the_bm_control block.

RETURN VALUE

CRAM_SUCCESS successful

SEE ALSO

cram_bm_di_rt_all

cram_set_i rq

DESCRIPTION

Set IRQ (interrupt request) number

USAGE

#include <cram.h>

int cram_set_irq (int irq);

irq the IRQ number (3-7)

REMARKS

This function is used to tell the API which IRQ (interrupt request line) is tied to the board (via Jumper Block 5). The number is loaded into API global variable _CRAM_IRQ. When a user interrupt handler is installed (via cram_set_irq), the API will tie the CRAM interrupt handler to the vector associated with the IRQ number.

RETURN VALUE

CRAM_SUCCESS successful

EXAMPLE

/* Set CRAM board to service interrupt request 3 */

cram_set_irq (3);

SEE ALSO

cram_setup_intr cram_restore_intr
cram_api_isr

cram_setup_intr

DESCRIPTION

Installs a routine to service CRAM interrupts.

USAGE

```
#include <cram.h>

int cram_setup_intr (void (far *uselfunc) (void));
```

userfunc the user-supplied interrupt service routine

REMARKS

This function installs a user-supplied function to be activated when the CRAM board sends an interrupt. The user's function can be any routine which accepts no parameters. cram_install_intr first saves the current vector hooked to the selected IRQ, then hooks the API interrupt handler to that IRQ. The API interrupt handler essentially calls the user-supplied interrupt service routine in addition to performing certain necessary tasks related to the interrupt mechanism.

RETURN VALUE

CRAM_SUCCESS successful

SEE ALSO

cram_set_irq
cram_api_isr
cram_restore_intr

cram restore intr

DESCRI PTION

Uninstalls interrupt service routine.

USAGE

#include <cram.h>

int cram_restore_intr (void);

REMARKS

This function uninstalls the CRAM interrupt service routine and restores the vector which was originally hooked to _CRAM_IRQ.

RETURN VALUE

CRAM_SUCCESS successful

SEE ALSO

cram_set_irq
cram_setup_intr
cram_api_isr

cram_api_isr

DESCRIPTION

API Interrupt Service Routine.

USAGE

#include <cram.h>

int cram_api_intr (void);

REMARKS

This is the API interrupt service routine. It is hooked-up to the IRQ via cram_set_irq (irq) and it will be automatically activated when the board receives an interrupt. The function performs certain necessary tasks, and then calls the user's own interrupt service routine. Upon returning from the user's function, it sends a non-specific end-of-interrupt to the PC's interrupt controller chip, and then relinquishes control to the main program.

RETURN VALUE

CRAM_SUCCESS successful

SEE ALSO

cram_setup_intr
cram_restore_intr
cram_set_i rq

cram_bc_tick

DESCRIPTION

Set the desired tick length

USAGE

```
#include <cram.h>

int cram_bc_tick ( word length)
```

REMARKS

The tick is a period that supplies an interrupt to the on board CPU. This period can be programmmed by the user by loading a value into SPECIAL_CTL (Ticks) register. A value of less than 2000 will be ignored and will result in the default value of 6250. The actual time period is resulting from the register value multiplied by 0.4 microseconds (n * 0.4 microseconds).

RETURN VALUE

CRAM_SUCCESS successful

1553 ERROR INJECTION CONTROL WORD

BC ERROR INJECTION REGISTER

Default: 0 – disable 1- enable

Bit 0 – Reserved

Bit 1 – Reserved

Bit 2 – Reserved

Bit 3 – Reserved

Bit 4 – Manchester Code Command error

Bit 5 – Parity Command Error

Bit 6 – Command Sync Overflow Error

Bit 7 – Manchester Data Error

Bit 8 – Parity Data Error

Bit 9 – Sync Command Error

Bit 10 – Sync Data Error

Bit 11 – Data Word Gap error

Bit 12- Reserved

Bit 13 – Reserved

Bit 14 – Reserved

Bit 15 – Reserved

1553 RT STATUS WORD

Default: 0 – disable 1- enable

Bit 0 – CLASS A – '1' CLASS B –'0' – Default – Internal

Bit 1 – Shut down channel A – Internal

Bit 2 – Shut down channel B – Internal

Bit 3 – Mode Syn – Status Word

Bit 4 – Mode Select Shut - Internal

Bit 5 – Mode Overflow Shut - Internal

Bit 6 – Instrumentation – Status Word

Bit 7 – Service Request – Status Word

Bit 8 – Reserved – Status Word

Bit 9 – Reserved – Status Word

Bit 10 – Reserved – Status Word

Bit 11 – Reserved – Status Word

Bit 12- Busy – Status Word

Bit 13 – S_Flag – Status Word

Bit 14 – Din – Status Word

Bit 15 – T_Flag – Status Word

INTERNAL STATUS REGISTER

Default: 0 – disable 1- enable

Bit 0 – Parity Command Error

Bit 1 – Parity Data Error

Bit 2 – Sub-address Error

Bit 3 – Broadcast

Bit 4 – Mode Command, Sub-Address 0 or 31

Bit 5 – Data received Bus – A -0 B-1

Bit 6 – BC RT-RT Command

Bit 7 – Manchester Code

Bit 8 – Number Data Error – last 5 bits in the BC-RT Command

Bit 9 – RT Status response Error Bit

Bit 10 – RT_RT Selected – two words with two sync command

Bit 11 – Com_OV_Error – too many sync commands

Bit 12- Data_OV_error – too many command sync

Bit 13 Manchester Data Error

Bit 14 – Mode Code Error

Bit 15 – Sync First Sample – 0- Command 1-Data