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. 3.7A June 19, 2013

ARINC 429 MEMORY CONTROL SEGMENT

X7E00	**GSR**	**X7E00**	**System_ID (8)**	**BYTE**
		X7E08	**Channe_cfg**	**WORD**
ARINC		**X7E0A**	**STATUS_TX_RX**	**WORD**
GENERAL		**X7E0C**	**INTERR_TX_RX**	**WORD**
REGISTERS		**X7E0E**	**PARAM_1**	**WORD**
		X7E10	**PARAM_2**	**WORD**
		X7E12	**PARAM_3**	**WORD**
		X7E14	**PARAM_4**	**WORD**
		X7E16	**TX_COUNT[4]**	**WORD**
		X7E1E	**RX_COUNT[4]**	**WORD**
		X7E26	**RESERVED [2]**	**WORD**
X7E28	**Tx_ctl [4]**	**X7E28**	**Command**	**WORD**
		X7E2A	**Start_Addr**	**WORD**
ARINC	**CHANNEL 0**	**X7E2C**	**Word_Count**	**WORD**
TRANSMIT		**X7E2E**	**Delay**	**WORD**
CONTROL		**X7E30**	**Status**	**INT.**
REGISTERS		**X7E32**	**Command**	**WORD**
		X7E34	**Start_Addr**	**WORD**
	CHANNEL1	**X7E36**	**Word_Count**	**WORD**
		X7E38	**Delay**	**WORD**
		X7E3A	**Status**	**INT.**
		X7E3C	**Command**	**WORD**
	CHANNEL2	**X7E3E**	**Start_Addr**	**WORD**
		X7E40	**Word_Count**	**WORD**
		X7E42	**Delay**	**WORD**
		X7E44	**Status**	**INT.**
		X7E46	**Command**	**WORD**
	CHANNEL3	**X7E48**	**Start_Addr**	**WORD**
		X7E4A	**Word_Count**	**WORD**
		X7E4C	**Delay**	**WORD**
		X7E4E	**Status**	**INT.**
X7E50	**Rx_ctl[4]**	**X7E50**	**Command**	**WORD**
		X7E52	**Start_Addr**	**WORD**
ARINC	**CHANNEL 0**	**X7E54**	**Buffer_Size**	**WORD**
RECEIVE		**X7E56**	**Label_Mask[16]**	**WORD**
CONTROL		**X7E76**	**Status**	**INT.**
REGISTERS		**X7E78**	**Command**	**WORD**
		X7E7A	**Start_Addr**	**WORD**
	CHANNEL 11	**X7E7C**	**Buffer_Size**	**WORD**

		XX7E7E	Label_Mask[16]	WORD
		XX7E8E	Status	INT.
		XX7E90	Command	WORD
	CHANNEL 22	XX7E92	Start_Addr	WORD
		XX7E94	Buffer_Size	WORD
		XX7E96	Label_Mask[16]	WORD
		XX7EA6	Status	INT.
		XX7EA8	Command	WORD
	CHANNEL 33	XX7EAA	Start_Addr	WORD
		XX7EAC	Buffer_Size	WORD
		XX7EAE	Label_Mask[16]	WORD
		XX7EBE	Status	INT.

X7EC0 ARINC BUS MONITOR REGISTERS	MONITOR_CTL	X7EC0	COUNTER	WORD
		X7EC2	STATUS	WORD
		X7EC4	INTERR_COND	WORD
		X7EC6	RESERVED	WORD

ARINC 708 MEMORY CONTROL SEGMENT

X7D90	**GSR708**	**X7D90**	**Channe_cfg**	**WORD**
		X7D92	**STATUS_TX_RX**	**WORD**
ARINC		**X7D94**	**INTERR_TX_RX**	**WORD**
708		**X7D96**	**PARAM**	**WORD**
GENERAL		**X7D98**	**TX_COUNT**	**WORD**
REGISTERS		**X7D9A**	**RX_COUNT**	**WORD**
		X7D9C	**RESERVED [2]**	**WORD**
X77DA0	**TX708_ctl**	**X7DA0**	**Command**	**WORD**
		X7 DA2	**Start_Addr**	**WORD**
ARINC		**X7DA4**	**Word_Count**	**WORD**
TRANSMIT		**X7 DA6**	**Delay**	**WORD**
CONTROL		**X7 DA8**	**Repeat**	**WORD**
REGISTERS		**X7 DAA**	**RESERVED [2]**	**WORD**
		X7 DAE	**Status**	**INT.**
X7E50	**RX708_ctl**	**X7E50**	**Command**	**WORD**
		X7E52	**Start_Addr**	**WORD**
RECEIVE	**CHANNEL 0**	**X7E54**	**Buffer_Size**	**WORD**
CONTROL		**X7E56**	**Label_Mask[16]**	**WORD**
REGISTERS		**X7E76**	**Status**	**INT.**
	ANNEL1	**X7E78**	**Command**	**WORD**
		X7E7A	**Start_Addr**	**WORD**
		X7E7C	**Buffer_Size**	**WORD**
		X7E7E	**Label_Mask[16]**	**WORD**
		X7E8E	**Status**	**INT.**
		X7E90	**Command**	**WORD**
		X7E92	**Start_Addr**	**WORD**
	CHANNEL3	**X7E94**	**Buffer_Size**	**WORD**
		X7E96	**Label_Mask[16]**	**WORD**
		X7EA6	**Status**	**INT.**
		X7EA8	**Command**	**WORD**
		X7EAA	**Start_Addr**	**WORD**
		X7EAC	**Buffer_Size**	**WORD**
		X7EAE	**Label_Mask[16]**	**WORD**
		X7EBE	**Status**	**INT.**

ARINC 717 MEMORY CONTROL SEGMENT

The units has up to two independent ARINC 717 transmit/receive channels.

X7D50	**GSR717**	**X7D50**	**Channe_cfg**	**WORD**
		X7D52	**STATUS_TX_RX**	**WORD**
ARINC		**X7D54**	**INTERR_TX_RX**	**WORD**
717		**X7D56**	**PARAM**	**WORD**
GENERAL		**X7D58**	**TX_COUNT [2]**	**WORD**
REGISTERS		**X7D5C**	**RX_COUNT [2]**	**WORD**
X7D64	**TX717_ctl**	**X7D60**	**Command**	**WORD**
		X7D62	**Start_Addr**	**WORD**
ARINC	**Channel 1**	**X7 D64**	**Word_Count**	**WORD**
TRANSMIT		**X7D66**	**Repeat**	**WORD**
CONTROL		**X7 D68**	**Delay**	**WORD**
REGISTERS		**X7D6A**	**Status**	**INT.**
	Channel 2	**X7 D6C**	**Command**	**WORD**
		X7D6E	**Start_Addr**	**WORD**
		X7 D70	**Word_Count**	**WORD**
		X7D72	**Repeat**	**WORD**
		X7 D74	**Delay**	**WORD**
		X7 D76	**Status**	**INT.**
X7D7C	**RX717_ctl[4]**	**X7E30**	**Command**	**WORD**
		X7E32	**Start_Addr**	**WORD**
RECEIVE	**CHANNEL 0**	**X7E34**	**Buffer_Size**	**WORD**
CONTROL		**X7E36**	**Label_Mask[16]**	**WORD**
REGISTERS		**X7E76**	**Rec_Index**	**INT.**
	CHANNEL1	**X7E78**	**Command**	**WORD**
		X7E7A	**Start_Addr**	**WORD**
	Channel 2	**X7E7C**	**Buffer_Size**	**WORD**
		X7E7E	**Label_Mask[16]**	**WORD**
		X7EBE	**Rec_Start**	**INT.**
		X7E90	**Command**	**WORD**
		X7E92	**Start_Addr**	**WORD**
	CHANNEL3	**X7E94**	**Buffer_Size**	**WORD**
		X7E96	**Label_Mask[16]**	**WORD**
		X7EA6	**Status**	**INT.**
		X7EA8	**Command**	**WORD**
		X7EAA	**Start_Addr**	**WORD**
		X7EAC	**Buffer_Size**	**WORD**
		X7EAE	**Label_Mask[16]**	**WORD**
		X7EBE	**Status**	**INT.**

Table of Contents

IRQ (TX_RX) Commands:

artic_di_datarcvd - ARINC429 (channel # 1-8) & 708 (channel 9) & 717 (channel 10-11)
artic_di_halfbuf_full - ARINC429 (channel # 1-8) , 708 (channel 9), 717 (channel 10-11)
artic_di_rcvfull - ARINC429 (channel # 1-8) & 708 (channel 9) & 717 (channel 10-11)
artic_di_txcomplete - ARINC429 (channel # 1-8) & 708 (channel 9) & 717 (channel 10-11)
artic_dis_monfull
artic_dis_monhalf
artic_dis_monitor
artic_ei_datarcvd - ARINC429 (channel # 1-8) & 708 (channel 9) & 717 (channel 10-11)
artic_ei_halfbuff- ARINC429 (channel # 1-8) & 708 (channel 9) & 717 (channel 10-11)
artic_ei_rcvfull- ARINC429 (channel # 1-8) & 708 (channel 9) & 717 (channel 10-11)
artic_ei_tx_complete - ARINC429 (channel # 1-8) , 708 (channel 9) & 717 (channel 10-11)

BM ARINC429 Commands:

artic_bm_available
artic__bm_mode
artic_ena_monfull
artic_ena_monhalf
artic_ena_monitor
artic_mon_halfbuf_full
artic_mon_fullbuff
artic_rx_mon_count

ARINC 708 special Commands:

arinc708_sel_channel(int channel)
arinc708_tx_repeat(WORD repeat)
arinc708_Harvard_NRZ(int error, int clr_set)

ARINC 717 special Commands

arinc717_val_channel(short channel) ;
arinc717_act_channel(short channel, short receive) ;
arinc717_tx_repeat (short repeat, short channel) ;
arinc717_Harvad_NRZ(short nrz, short channel) ;
artic717_txrxrate_set (WORD rate, short channel) ;
artic717_int_loop (WORD set, short channel) ;
artic717_rx_index (short channel) ;

artic_bm_available

DESCRIPTION

ARTIC data bus monitor receive mode

USAGE

#include <uadi32k.h>

int artic_bm_available ( )

REMARKS

This function checks the the appropriate bit in the board's Monitor status register if the ARTIC board includes the BUS MONITOR option.

RETURN VALUE

YES

NO

SEE ALSO

artic_dis_monitor	**artic_ena_monhalf**
artic_dis_monhalf	**artic_ena_monfull**
artic_dis_monfull	**artic_mon_halfbuff_full**
artic_mon_fullbuff	**artic_bm_available**
artic_bm_mode	**artic_rx_mon_count**

artic_bm_mode

DESCRIPTION

ARTIC data bus monitor receive mode

USAGE

#include <uadi32k.h>

int artic_bm_mode ( )

REMARKS

This function checks the the appropriate bit in the board's Monitor status register if the ARTIC board is in monitor mode or not.

RETURN VALUE

YES

NO

SEE ALSO

artic_dis_monitor

artic_dis_monhalf

artic_dis_monfull

artic_mon_fullbuff

artic_bm_mode

artic_ena_monhalf

artic_ena_monfull

artic_mon_halfbuff_full

artic_bm_available

artic_rx_mon_count

artic_datarcvd

DESCRIPTION

Check if new data received

USAGE

#include <uadi32k.h>

int artic_datarcvd (channel)

int channel; receive channel number

REMARKS

This function checks the appropriate bit in the boards Status Register. This bit is set by the ARTIC board in all receive modes when any new data (at least one word) has been received since the last time this bit was polled. After reading the bit, the function clears it for the next operation. This function should be used in a polled (non-interrupt) fassion to check if new data was received. In interrupt mode, it should be called from the Interrupt Service Routine to identify the source of the interrupt.

RETURN VALUE

YES New data received

NO No new data since last time checked

ARTIC_INV_CHNL Invalid or unconfigured channel

EXAMPLE

```
/* polled receive operation: wait for 10 ARINC words on channel 1 */

/* and display them when done: */
ARINC_WORD rcv_buff[10];
int i;


artic_rx_cmd( 1, RX_NORMAL, 0x0000, 10); /* start receiving */
while (artic_datarcvd(1 )=NO); 1* wait for incomming word */
artic_readbuf(rcv_buff,0x0000, 10*4); /*read data from board into our buffer*/ *
display data on screen: *1
for (i=0; i<10; i++)
prinff("Data word #%d = %0IXn, i+1, rcv_buff[i]);
```

SEE ALSO

artic_ei_datarcvd

artic_rcvbuf_full

artic_halfbuf_full

artic_di_datarcvd

DESCRIPTION

Disable Interrupt on New Data Received

USAGE

#include <uadi32k.h>

int artic_di_datarcvd (channel)

int channel; the receive channel number

REMARKS

This function clears the appropriate bit in the appropriate interrupt condition register which will prevent the board from issuing an IRQ (interrupt request) when the following condition is met: New data (at least one new word) received on the channel.

RETURN VALUE

ARTIC_SUCCESS

ARTIC_INV_CHNL Invalid or unconfigured channel

SEE ALSO

artic_ei_datarcvd

artic_ei_rcvbuf_full

artic_di_rcvbuf_full

artic_di_halfbuf_full

DESCRIPTION

Disable Interrupt on Receive Buffer half full

USAGE

#include <uadi32k.h>

int artic_di_halfbuf_full (channel)

int channel; the receive channel number

REMARKS

This function clears the appropriate bit in the appropriate interrupt condition register which will prevent the board from issuing an IRQ (interrupt request) when the following condition is met: the flow of incoming data just passed the half-length mark of the Receive Buffer.

RETURN VALUE

ARTIC_SUCCESS

ARTIC_INV_CHNL Invalid or unconfigured channel

SEE ALSO

artic_ei_halfbuf_full

artic_di_datarcvd **artic_ei_d atarcvd**

artic_ei_rcvbuf_full **artic_di_rcvbuf_full**

artic_di_rcvfull

DESCRIPTION

Disable Interrupt on Receive Buffer Full

USAGE

#include <uadi32k.h>

int artic_di_rcvfull (channel)

int channel; the receive channel number

REMARKS

This function clears the appropriate bit in the appropriate interrupt condition register which will prevent the board from issuing an IRQ (interrupt request) when the following condition is met: the last location of the Receive Buffer has been written with new data.

RETURN VALUE

ARTIC_SUCCESS

ARTIC_INV_CHNL Invalid or unconfigured channel

SEE ALSO

artic_ei_rcvbuf_full **artic_di_datarcvd**

artic_ei_datarcvd

artic_ei_halfbuf_full **artic_di_halfbuf_full**

artic_di_txcomplete

DESCRIPTION

Disable Interrupt on Transmit complete

USAGE

#include <uadi32k.h>

int artic_di_txcomplete (channel)

int channel; the transmit channel number

REMARKS

This function clears the appropriate bit in the appropriate interrupt condition register which will prevent the board from issuing an IRQ (interrupt request) when the following condition is met: the entire contents of the buffer specified in a Transmit Command has been transmitted to the ARINC channel.

RETURN VALUE

ARTIC_SUCCESS

ARTIC_INV_CHNL Invalid or unconfigured channel

SEE ALSO

artic_di_datarcvd

artic_ei_rcvbuf_full

artic_di_rcvbuf_full

artic_ei_rcvbuf_full

artic dis monfull

DESCRIPTION

Disable Interrupt on Monitor Buffer full

USAGE

#include <uadi32k.h>

int artic_dis_monfull ( )

REMARKS

This function clears the appropriate bit in the appropriate monitor interrupt condition register which will prevent the board to issue an IRQ (interrupt request) when the following condition is met: the flow of incoming data passed the length mark of the Monitor Receive Buffer (3800 bytes- 380 ARINC words+TIME TAG+CHANNEL) and wrap around in the circular buffer.
The ARTIC board will accept the command only if the board has a BUS MONITOR option.

RETURN VALUE

ARTIC_SUCCESS

SEE ALSO

artic_ena_monitor	**artic_dis_monitor**
artic_ena_monhalf	**artic_dis_monhalf**
artic_dis_monfull	**artic_mon_halfbuff_full**
artic_mon_fullbuff	**artic_bm_available**
artic_bm_mode	**artic_rx_mon_count**

artic_dis_monhalf

DESCRIPTION

Enable Interrupt on Monitor Buffer half full

USAGE

#include <uadi32k.h>

int artic_dis_monhalf ( )

REMARKS

This function clears the appropriate bit in the appropriate monitor interrupt condition register which will prevent the board from issuing an IRQ (interrupt request) when the following condition is met: the flow of incoming data just passed half length mark of the monitor receive data; 1900 bytes (190 ARINC words + TIME tag + channel number).

RETURN VALUE

ARTIC_SUCCESS

SEE ALSO

artic_ena_monitor	**artic_dis_mon itor**
artic_ena_monhalf	**artic_ena_monfull**
artic_dis_monfull	**artic_mon_halfbuff_full**
artic_mon_fullbuff	**artic_bm_available**
artic_bm_mode	**artic_rx_mon_count**

artic_dis_monitor

DESCRIPTION

Disable ARTIC data monitor receive mode

USAGE

#include <arlic.h>

int artic_dis_monitor ( )

REMARKS

This function clears the appropriate bit in the board's Monitor status register the monitor receive data mode.

RETURN VALUE

ARTIC_SUCCESS

SEE ALSO

artic_dis_monitor	**artic_ena_monhalf**
artic_dis_monhalf	**artic_ena_monfull**
artic_dis_monfull	**artic_mon_halfbuff_full**
artic_mon_fullbuff	**artic_bm_available**
artic_bm_mode	**artic_rx_mon_count**

artic_ei_datarcvd

DESCRIPTION

Enable an Interrupt on new data received
ARINC429 (channel # 1-8) & 708 (channel 9) & 717 (channel 10-11)

USAGE

#include <uadi32k.h>

int artic_ei_datarcvd (channel)

int channel; the receive channel number

REMARKS

This function sets the appropriate bit in the appropriate interrupt condition register which will cause the board to issue an IRQ (interrupt request) when the following condition is met: One or more new data word(s) have been received on the channel.

RETURN VALUE

ARTIC_SUCCESS

ARTIC_INV_CHNL Invalid or unconfigured channel

SEE ALSO

artic_di_datarcvd

artic_ei_rcvbuf_full

artic_di_rcvbuf_full

artic_ei_halfbuff

DESCRIPTION

Enable Interrupt on Receive Buffer half full
ARINC429 (channel # 1-8) & 708 (channel 9) & 717 (channel 10-11)

USAGE

#include <uadi32k.h>

int artic_eihalfbuf_full (channel)

int channel; the receive channel number

REMARKS

This function sets the appropriate bit in the appropriate interrupt condition register which will cause the board to issue an IRQ (interrupt request) when the following condition is met: the flow of incoming data just passed the half-length mark of the Receive Buffer. This interrupt condition is useful for handling large bursts of received data. Getting an interrupt at the half-buffer mark allows to read the first half while the second half is filling up. Upon receiving the Buffer Full interrupt, the second half of the buffer can be read while the first half is filling up, and so-on.

RETURN VALUE

ARTIC_SUCCESS

ARTIC_INV_CHNL Invalid or unconfigured channel

SEE ALSO

artic_di_halfbuf_full **artic_di_datarcvd**

artic_ei_datarcvd **artic_ei_rcvbuf_full**

artic_di_rcvbuf_full

artic_ei_rcvfull

DESCRIPTION

Enable Interrupt on Receive Buffer Full
ARINC429 (channel # 1-8) & 708 (channel 9) & 717 (channel 10-11)

USAGE

#include <uadi32k.h>

int artic_ei_rcvfull (channel)

int channel; the receive channel number

REMARKS

This function sets the appropriate bit in the appropriate interrupt condition register which will cause the board to issue an IRQ (interrupt request) when the following condition is met: the last location of the Receive Buffer has been written with new data.

RETURN VALUE

ARTIC_SUCCESS

ARTIC_INV_CHNL Invalid or unconfigured channel

SEE ALSO

artic_di_rcvbuf_full **artic_di_datarcvd**

artic_ei_datarcvd **artic_ei_halfbuf_full**

artic_di_halfbuf_full

artic_ei_txcomplete

DESCRIPTION

Enable Interrupt on Transmit complete
ARINC429 (channel # 1-8) & 708 (channel 9) & 717 (channel 10-11)

USAGE

#include <artich>

int artic_ei_txcomplete (channel)

int channel; the transmit channel number

REMARKS

This function sets the appropriate bit in the appropriate interrupt condition register which will cause the board to issue an IRQ (interrupt request) when the following condition is met: the entire contents of the buffer specified in a Transmit Command has been transmitted to the ARINC channel.

RETURN VALUE

ARTIC_SUCCESS

ARTIC_INV_CHNL Invalid or unconfigured channel

EXAMPLE

```
        ARINC_WORD tx_buff[500];

intr_handler 0
{
        if (artic txcomplete(channel)) {
        }
}

main()
{
artic_setup_intr( intr_handler); artic_ei_txcomplete(channel);
artic_writebuf(0x0000,tx_buff,500); I transfer data to board*I
artic_tx_cmd(channel,ARTIC_TX_NORMAL, 0x0000, 500,0);
r when transmit buffer completed intr_handler will */
/* be activated from the IRQ. */
}
```

SEE ALSO

artic_di_txcomplete

artic_ena_monfull

DESCRIPTION

Enable Interrupt on Monitor Buffer full

USAGE

#include <uadi32k.h>

int artic_ena_monfull ( )

REMARKS

This function sets the appropriate bit in the appropriate monitor interrupt condition register which will cause the board to issue an IRQ (interrupt request) when the following condition is met: the flow of incoming data passed the length mark of the Monitor Receive Buffer (3800 bytes- 380 ARINC words+TIME TAG+CHANNEL) and wrap around in the circular buffer. This interrupt condition is useful for handling large bursts of received data.
The ARTIC board will accept the command only if the board has a BUS MONITOR option.

RETURN VALUE

ARTIC_SUCCESS

SEE ALSO

artic_ena_monitor
artic_ena_monhalf
artic_dis_monfull
artic_mon_fullbuff
artic_bm_mode
artic_dis_monitor
artic_dis_monhalf
artic_mon_halfbuff_full
artic_bm_available
artic_rx_mon_count

artic_ena_monhalf

DESCRIPTION

Enable Interrupt on Monitor Buffer half full

USAGE

#include <uadi32k.h>

int artic_ena_monhalf ( )

REMARKS

This function sets the appropriate bit in the appropriate monitor interrupt condition register which will cause the board to issue an IRQ (interrupt request) when the following condition is met: the flow of incoming data just passed the half-length mark of the Monitor Receive Buffer (1900 bytes- 190 ARINC+TIME TAG+CHANNEL). This interrupt condition is useful for handling large bursts of received data. Getting an interrupt at the half-buffer mark allows to read the first half while the second half is filling up. Upon receiving the Buffer Full interrupt, the second half of the buffer can be read while the first half is filling up, and so-on. The ARTIC board will accept the command only if the board has a BUS MONITOR option.

RETURN VALUE

ARTIC_SUCCESS

SEE ALSO

artic_ena_monitor	**artic_dis_monitor**
artic_dis_monhalf	**artic_ena_monfull**
artic_dis_monfull	**artic_mon_halfbuff_full**
artic_mon_fullbuff	**artic_bm_available**
artic_bm_mode	**artic_rx_mon_count**

artic_ena_monitor

DESCRIPTION

Enable ARTIC data monitor receive mode

USAGE

#include <uadi32k.h>

int artic_ena_monitor ( )

REMARKS

This function sets the appropriate bit in the appropriate status monitor condition register which will enable the board to store ARINC receive data according to the monitor data structure; ARINC WORD TIME TAG (32 bits - 2 usec resolution) and channel number. The ARTIC board will accept the command only if the board has a BUS MONITOR option.

RETURN VALUE

ARTIC SUCCESS

SEE ALSO

artic_dis_monitor	**artic_ena_monhalf**
artic_dis_monhalf	**artic_ena_monfull**
artic_dis_monfull	**artic_mon_halfbuff_full**
artic_mon_fullbuff	**artic_bm_available**
artic_bm_mode	**artic_rx_mon_count**

artic_halfbuf_ full

DESCRIPTION

Check if Receive Buffer is half full
ARINC429 (channel # 1-8) & 708 (channel 9) & 717 (channel 10-11)

USAGE

#include <uadi32k.h>

int artic_halfbuf_full (channel)

int channel; receive channel number

REMARKS

This function checks the appropriate bit in the board's Status Register. This bit is set by the ARTIC board in all receive modes when the flow of incoming data just passed the half-length mark of the Receive Buffer. After reading the bit, the function clears it for the next operation. This function should be used in a polled (non-interrupt) fassion to check if half-buffer is full was received. In interrupt mode, it should be called from the Interrupt Service Routine to identify the source of the interrupt.

RETURN VALUE

YES Half-buffer full

NO Half-buffer mark not reached since last checked

ARTIC_INV_CHNL Invalid or unconfigured channel

SEE ALSO

artic_ei_halfbuf_full

artic_rcvbuf_full

artic_datarcvd

artic_Label_dis_all

DESCRIPTION

Disable all Labels in Label Mask

USAGE

#include <uadi32k.h>

int artic_Label_dis_all (Labelmask)

BYTE *Label mask pointer to 32 byte (256 bit) Label mask

REMARKS

This function clears all the bits of the supplied Label Mask. This function should be used when the user wants to enable only a few Labels, by first calling artic_Label_dis_all then enabling the individual Labels via artic_Label_enable for each reievant Label. The Label mask can then be passed to artic_rx_cmd()

RETURN VALUE

ARTIC_SUCCESS

EXAMPLE

```
/* receive 10 words with Labels 17 and 19 only: */
BYTE Label mask[32];
ar[ic_Label_dis_all(Label_mask);
artic_Label_enable(Label_mask, 17); artic_Label_enable(Label_mask, 19);
artic_rx_cmd(channel, RX_NORMAL,0x0000, 10, Label_mask);
```

SEE ALSO

artic_Labelena_all artic_Label_enable artic_Label_disable artic_rx_cmd

artic_Label_disable

DESCRIPTION

Disable Label (Clear Label Bit in Label Mask)

USAGE

#include <uadi32k.h>

int artic_Label_disable (Label mask,Label)

BYTE *Label mask pointer to 32 byte (256 bit) Label mask

BYTE Label; Label value to be ignored (disabled)

0-255

REMARKS

This function is used to disable a particular Label in a Label mask, by clearing the appropriate bit in the Label mask.

RETURN VALUE

ARTIC_SUCCESS

EXAMPLE

```
        /* Assume an ARINC link should carry only Label 113 words. */
         /* The following program can be used to verify no other */
         /* Labels are passed though the link: */
BYTE Label_mask[32]; /* allocate a 256-bit Label Mask */
artic_Label_ena_all(Label_mask); I receive all Labels ... */
artic_Label_disable(Label_mask, 113) /* except for 113 */
result =artic_rx_cmd(channel ,ARTIC_RX_NORMAL, 0x0000, 500, Label_mask);
if (result==ARTIC_SUCCESS) {
printf("Watching for wrong Label..") ;
while (artic_rcvbuf_full(channel)NO) {

prinff( "wrong Labels received") ;
}
printf("n 500 or more wrong Labels received, exiting exitO;
}
```

SEE ALSO

artic_label_enable
artic_Label_ena_all
artic_Label_dis_all
artic_rx_cmd

artic_Label_ena_all

DESCRIPTION

Enable all Labels in Label Mask

USAGE

#include <uadi32k.h>

i nt artic_Label_ena_all (Label mask)

BYTE *Label mask pointer to 32 byte (256 bit) Label mask

REMARKS

This function sets all the bits of the supplied Label Mask. This function should be used when the user wants to receive all Labels, or disable only a few Labels. In the latter case, first call arlic_Label_ena_all then disable the individual Labels via artic_Label_disable for each Label to be ignored. The Label mask can then be passed to artic_rx_cmd.

RETURN VALUE

ARTIC_SUCCESS

EXAMPLE

/* receive 10 words, all Labels enabled: */

```
BYTE Label mask[32];
artic_Label_ena_all( Label mask);
artic_rx_cmd(channel, RX_NORMAL, 0x0000, 10, Label_mask);
```

SEE ALSO

artic_Label_dis_all
artic_Label_enable
artic_Label_disable
artic_rx_cmd

artic_Label_enable

DESCRIPTION

Enable Label (set Label bit in Label Mask)

USAGE

#include <uadi32k.h>

int artic_Label_enable (Label mask, Label)

BYTE *Label mask pointer to 32 byte (256 bit) Label mask

BYTE Label; Label value to be enable (0-255)

REMARKS

This function is used to enable a particular Label in a Label mask, by setting the appropriate bit in the Label mask.

RETURN VALUE

ARTIC_SUCCESS

EXAMPLE

```
/* receive 10 words with Labels 17 and 19 only: */
 BYTE Label_mask[32];
artic_Label_dis_all( Label_mask);
artic_Label_enable(Label_mask, 17);
artic_Label_enable(Label_mask, 19);
artic_rx_cmd(channel,ARTIC_RX_NORMAL,0x0000, 10, Label mask);
```

SEE ALSO

artic_Label_disable **artic_Label_ena_all**

artic_Label_dis_all **artic_rx_cmd**

artic_last_Label

DESCRIPTION

Read the last received Label

USAGE

#include <uadi32k.h>

int artic_last_label (channel);

int channel; channel number

REMARKS

This routine returns the Label Value of the last received ARINC data word on the channel. The user should check if any data is available before calling this function.

RETURN VALUE

Label Value (0 to 255) If data available

ARTIC_INV_CHANNEL Invalid or unconfigured channel

ARTIC_NO_DATA No data was received since the last Receive

EXAMPLE

/* Start receiving in LABEL mode, display receive data:*/

```
int channel 1; /* channel 1 */
BYTE Label_mask[32j; P Label mask for receive command */
int Label; /* variable to store rcvd Label value */
/* start receiving, rx buffer at 0x0000 (size=DONT CARE): */
 artic_rx_cmd( I ,ARTIC_RX_LABEL, 0x0000, 0x0000, Label_mask);
while (1) {
while (artic_data_rcvd(1 )==NO) {
/* waiting for data */
}
/* data received, display it in hexadecimal format: */
Label = artic_Last_Label(1);
printf ("Data Received = %081X\n", artic_rx_Label(1 ,Label);
}
```

SEE ALSO

artic_rx_Label **artic_rx_cmd**

artic_datarcvd **artic_rx_count**

artic_mon_halfbuf_full

DESCRI PTION
Check if Monitor Buffer is half full

USAGE

```
#include <uadi32k.h>

int artic_mon_halfbuf_full 0
```

REMARKS

This function checks the appropriate bit in the board's Monitor Status Register. This bit is set by the ARTIC board in all receive modes when the flow of incoming data just passed the half-length mark of the Monitor Receive Buffer(1900 bytes - 190 ARINC words+ TIME tag +CHANNEL number). After reading the bit, the function clears it for the next operation. This function should be used in a polled (non-interrupt) fassion to check if half-buffer is full was received. In interrupt mode, it should be called from the Interrupt Service Routine to identify the source of the interrupt.

RETURN VALUE

YES Half-buffer full

NO Half-buffer mark not reached since last checked

SEE ALSO

artic_ena_monitor	**artic_dis_monitor**
artic_ena_monhalf	**artic_dis_monhalf**
artic_ena_monfull	**artic_dis_monfull**
artic_mon_fullbuff	**artic_bm_available**
artic_bm_mode	**artic_rx_mon_count**

artic_mon_fullbuff

DESCRIPTION

Checks if Monitor Buffer if full

USAGE

#include <uadi32k.h>

int artic_mon_fullbuff ( )

REMARKS

This function checks the appropriate bit in the board's Monitor Status Register. This bit is set by the ARTIC board in both Normal and Continuous modes when the last location in the receive buffer was written by ARTIC with a new data word. After reading the bit, the function clears it for the next operation. This function should be used in a polled (non-interrupt) fassion to check if the buffer is full. In interrupt mode, it should be called from the Interrupt Service Routine to identify the source of the interrupt.

RETURN VALUE

YES Receive fuffer is full

NO Recive buffer not full

EXAMPLE

```
        /* polled receive operation: wait for 400 ARINC words */

/* and display them when done */
char rcv_buffI3600];
int i;
while (artic_mon_fullbuff() ==NO) {
        /* wait for receive complete */
/* Please include delay time */
}
artic_readbuf(rcv_buff, 0x0000, 3600);
for (i=0; i<3600; i+9) { printf("Data word = %OIX rcv_buff[i]); printf("Time tag =
%OIX ",rcv_buff[i+41); printf("Channel # = %dln", rcv_buff [i+8]);
}
```

SEE ALSO

Artic_ena_monitor
artic_dis_monitor
artic_ena_monhalf
artic_dis_monhalf
artic_ena_monfull
artic_dis_monfull
artic_mon_fullbuff

artic_parity_disable

DESCRIPTION

Disable parity in ARINC data transmission

USAGE

#include <uadi32k.h>

int artic_parity_disable (channel);
int channel; the receive channel number (1-4)

REMARKS

This routine selects parity for the specified channel. The disable parity will be applied at the NEXT Receive or Transmit command, i.e. if one of the channels is currently receiving or transmitting with NO parity selected the parity polarity will not be changed "on the fly".
The procedure clear bit 15 from selected cahannel TX_CTL.PARAMS register.

RETURN VALUE

ARTIC_SUCCESS Successful

ARTIC_INV_CHNL Invalid or unconfigured channel

EXAMPLE

if ((result=artic_parity_disable(3)) != ARTIC_SUCCESS)

printf("Failed to select parity for chan 3, error code %d",result);
SEE ALSO

artic_parity_enable

DESCRIPTION

Enable parity bit in ARINC data transmission

USAGE

#include <uadi32k.h>

int artic_parity_enable (channel);
int channel; the receive channel number (1-4)

REMARKS

This routine selects parity for the specified channel. The disable parity will be applied at the NEXT Receive or Transmit command, i.e. if one of the channels is currently receiving or transmitting with NO parity selected the parity polarity will not be changed "on the fly".
The procedure set bits 15 from TX_CTL.PARAMS channel register.

RETURN VALUE

ARTIC_SUCCESS Successful

ARTIC_INV_CHNL Invalid or unconfigured channel

EXAMPLE

if ((result=artic_parity_enable(3)) != ARTIC_SUCCESS)

printf("Failed to select parity for chan 3, code error %d",result);
SEE ALSO

artic_parity_even

DESCRIPTION

Set EVEN parity

USAGE

#include <uadi32k.h>

int artic_parity_even (channel);
int channel; the receive channel number (1-4)

REMARKS

This routine seiects EVEN parity for the specified channel. The EVEN parity will be applied at the NEXT Receive or Transmit command, i.e. if one of the channels is currently receiving or transmitting with ODD parity selected the parity polarity will not be changed "on the fly".

RETURN VALUE

ARTIC_SUCCESS Successful

ARTIC_INV_CHNL Invalid or unconfigured channel

EXAMPLE

if ((result=artic_parity_even(3)) != ARTIC_SUCCESS)

printf("Failed to set EVEN parity for chan 3, error code %d",result);

SEE ALSO

artic_parity_odd

artic_parity_odd

DESCRIPTION

Set ODD parity

USAGE

#include <uadi32k.h>

int artic_parity_odd (channel);
int channel; the receive channel number (1-4)

REMARKS

This routine seiects ODD parity for the specified channel. The ODD parity will be applied at the NEXT Receive or Transmit command, i.e. if one of the channels is currently receiving or transmitting with EVEN parity selected the parity polarity will not be changed "on the fly".

RETURN VALUE

ARTIC_SUCCESS Successful

ARTIC_INV_CHNL Invalid or unconfigured channel

EXAMPLE

if ((result=artic_parity_odd(1)) != ARTIC_SUCCESS)

prinff("Failed to set ODD parity for chan 1, error code %d",result);

SEE ALSO

artic_parity_even

artic_txrate_user_set

DESCRIPTION

Set ARINC transmission baud rate.

USAGE

#include <uadi32k.h>

int artic_txrate_user_set(short channel, WORD rate);
int channel; the receive channel number (1-4)
WORD rate; transmission baud rate : 8 MHz / rate

REMARKS

The ARTIC board has an internal 16 MHz. The baud rate is a division of this frequency. The procedure set the value into register TX_CTL.PARAM from the requested channel.
Rate must be bigger than 7 for proper operation

RETURN VALUE

ARTIC_SUCCESS Successful

ARTIC_INV_CHNL Invalid or unconfigured channel

EXAMPLE

```
if ( (result=artic_ txrate_user_set (1,8)) != ARTIC_SUCCESS)
// set channel 1 for MHz transmit baud rate
prinff("Failed to set transmission rate  for chan 1, error code %d",result);
```

SEE ALSO

artic_parity_even
artic_parity_odd
artic_parity_disable

artic_rcvbuf_full

DESCRIPTION

Checks if Receive Buffer if full

USAGE

#include <uadi32k.h>

int artic_rcvbuf_full (channel)

int channel; receive channel number

REMARKS

This function checks the appropriate bit in the boards Status Register. This bit is set by the ARTIC board in both Normal and Continuous modes when the last location in the receive buffer was written by ARTIC with a new data word. After reading the bit, the function clears it for the next operation. This function should be used in a polled (non-interrupt) fassion to check if the buffer is full. In interrupt mode, it should be called from the Interrupt Service Routine to identify the source of the interrupt.

RETURN VALUE

YES Receive fuffer is full

NO Recive buffer not full

ARTIC_INV_CHNL Invalid or unconfigured channel

EXAMPLE

```
/* polled receive operation: wait for 10 ARINC words on channel 1 */

l and display them when done: */
ARINC_WORD rcv_buffll 0];
int i;

artic_rx_cmd(1,RX_NORMAL,0x0000,1O); /* start receiving */
while (artic_rcvbuf_full(1 )== NO); /* wait for receive complete */
/* receive complete: */
/* read data from board into our buffer: */
artic_readbuf(rcv_buff,0x0000, I O4);
/* display data on screen: */
for (1=0; i<10; i++)
printf("Data word #%d = %0lXn", i+1, rcv_buff]);
```

SEE ALSO

artic_ei_rcvbuf_full

artic_halfbuf_full

artic_datarcvd

artic_readbuf

DESCRIPTION

Read a section of ARTIC memory

USAGE

#include <uadi32k.h>

int artic_readbuf (dst,src,size)

void *dest; destination address

void far *src; address of data to read from

int size; number of bytes to be read starting from src

REMARKS

This function is used to transfer a block of data from the ARTIC board Dual-Ported memory into program memory. The most typical use is to transfer Receive Data from the board after receiving data. This function should be used rather than direct memory operations in order to maintain compatibility with drivers for protected-memory operating systems.

RETURN VALUE

ARTIC_SUCCESS Transfer complete

EXAMPLE

SEE artic_rx_cmd example

SEE ALSO

artic_writebuf

artic_rx_busy

DESCRIPTION

Return whether Receiver Channel is busy

USAGE

#include <uadi32k.h>

int artic_rx_busy (channel)

int channel; receive channel number

REMARKS

This function reads the appropriate bit in the board's Channel Activity byte and returns whether the channel is currently executing a receive command or if it is free to accept another command (other than STOP, of course). However, it is not mandatory to use this function before issuing a command since the command function will return ARTIC_CHNLBUSY if the channel is busy. It is mostly used for display purposes in a software window, such as in displaying channel activity.

RETURN VALUE

YES Channel busy

NO channel not busy

ARTIC_INV_CHNL Invalid or unconfigured channel

SEE ALSO

artic_tx_busy

artic_rx_cmd

DESCRIPTION

Send a Receive Command to an ARTIC board

USAGE

#include <uadi32k.h>

int artic_rx_cmd (channel, command, start, bufsize, Label mask)

int channel; The receive channel number (1-4)

WORD command; Command code: ARTIC_RX_NORMAL,
ARTIC_RX_CONTNS,
ARTIC_RX_LABEL, or
ARTIC_RX_STOP.
WORD start Location (offset from board baseaddresss) of the beginning of the bufferin which received data should be stored(0x0000 to 0xOfOC - don't care if command = ARTIC_RX_STOP).

WORD bufsize; /* Size of receive data buffer in ARINC data words (1 to 9600 - don't care if command=ARTIC_RX_STOP or command=ARTIC_RX_LABEL – in ARTIC_RX_LABEL the buffer size is fixed to 256 *3= 768 bytes)

BYTE *Label mask pointer to 32-byte-long Label Mask.

REMARKS

This routine sets-up a Receive Command in the board's Receive Channel Control Block and waits for the board's response. If command is equal to ARTIC_RK_NORMAL, ARTIC will start receiving and will fill the supplied buffer (count words starting at offset start) with the incoming data. When the buffer is full, i.e. count words were received, the board will stop receiving and set the channel's bit in the tx_rx_complete interrupt status register. (this bit can be polled and cleared via the artic_rx_complete function) If interrupt on receive complete is enabled (done via artic_eircvfull), an Interrupt Request (IRQ) will be sent. If command is equal to ARTIC_RX_CONTNS, the board will receive in the continuous mode, i.e. it will operate like in NORMAL mode, but it will not stop receiving when the buffer is full, it will

continue receiving and filling the buffer from the beginning, overwriting the oldest data. In this mode, the board will send an IRQ (if enabled) also when the buffer is half-full, in addition to when the buffer is full, to allow the application software to start transfering data before the buffer is full. If command equals ARTIC_RX_LABEL, the board will receive in the Label mode. In this mode, the receive buffer is of a fixed size of 256 x 3 = 768 bytes, each ceil of 3 bytes representing a Label value. In this mode, each received word's data bits without the Label (24 bits = 3 bytes) is placed in the ceil corresponding to its Label value and the Label value is placed in the Last Label byte in the Global System Registers section (can be read via articjast_Label).

The following information applies to all 3 above-described receiving modes:

1) The Label Mask is always applied to incoming data. Any received data with a Label value disabled in the Label Mask will be ignored and treates as if it were never received.

2) Each time a new word is received, the ARTIC processor will set the channel's bit in the board's Data Received interrupt status byte (can be read via artic_data_rcvd). If interrupt on Data Received is enabled (via artic_ei_datarcvd), the board will also send an IRQ.

If command is equal to ARTIC_RX_STOP, the board will stop receiving (if it was receiving at all) on the channel.

RETURN VALUE

ARTIC_SUCCESS command was accepted: execution started.

ARTIC_INV_CHANNEL invalid or unconfigured channel specified.

ARTIC_INV_ADDRESS start address does not fall in the O-f00 range.

ARTIC_INV_SIZE count is such that the last data word would fall outside of the O-f00 range (start + count> F00).

ARTIC_CHNL_BUSY command=ARTIC_RX_NORMAL,

ARTIC_RX_CONTNS or

ARTIC_ TX_LOOP and the channel is already receiving.

ARTIC_BOARD_RESP board did not respond to the command within the response interval defined by _ARTIC_RESP_ TIMEOUT.

EXAMPLE

SEE RXNORMAL.C example program.

SEE ALSO

artic_rx_complete
artic_rx_count
artic_ei_data_rcvd
artic_Label_disable
artic_Label_dis_all
artic_datarcvd
artic_eirxcomplete
artic_label_enable
artic_Label_ena_all

artic_rxcount

DESCRIPTION

Read Receive Data Counter

USAGE

#include <uadi32k.h>

int artic_rx_count (channel);

int channel; channel number

REMARKS

This function returns the number of ARINC data words received on the channel since the last Receive Command was given modulo 32,768.

RETURN VALUE

Data count (0 to 32,767) Received Data Counter.

ARTIC_INV_CHANNEL Invalid or unconfigured channel.

EXAMPLE

```
/* CONTINUOUS mode receive operation interrupt handler: */ /* In this
mode the board sends an IRQ each time the buffer */
/* is full and each time the buffer is half-full. The only */
?* way to sense if we have a buffer full or half-full *1
/* condition is the Receive Count: */
/* Let's assume the main program sent an ARTIC_RX_CONTNS */
/* command with buffer size 200 words starting at 0x0000 */
/* and enabled interrupt on Receive Complete.
ARINC_WORD rx_buff {BUFFSIZE];/* can be as big as you want */
int I = 0;
void intr_handler()
{
```

```
if (artic_rx_complete(channel) {
        if ((artic_tx_count(channel)%200)> 100) {
        /* half buffer full - read 1st half: */
                artic_readbuf(&rx_buff[i], 0x0000, 100*4);
                i= i+100;
        }
        eise{
                /* 2nd half full, read it */
                artic_readbuf(&rx_buff], 0x0000+ 100,100*4);
                i= i+100;
        }
}
}
```

SEE ALSO

artic_tx_count

artic_rx_Label

DESCRIPTION

Return last received word for specified Label

USAGE

#include <uadi32k.h>

int artic_rx_Label (channel, Label, addr, data)

int channel; the receive channel number (1-4)

BYTE Label; the Label value (0-255)
ARINC_WORD *data; pointer to store the data

REMARKS

This function returns the value of the last received data word with a Label matching to the specified parameter. Note that this function is applicable for LABEL MODE ONLY (ARTIC_RX_LABEL). The function stores the 32 bit ARINC data word into the pointer specified by the function's third parameter.

RETURN VALUE

ARTIC_SUCCESS successful, read result in *data

ARTIC_NO_DATA no data received for this Label.

ARTIC_INV_CHNL Invalid or unconfigured channel

EXAMPLE

SEE the Label Mode example in artic_rx_data

SEE ALSO

artic_la st_Ia be I
artic_rx_cmd

artic_rx_mon_count

DESCRIPTION

Read Receive Data Monitor Counter

USAGE

#include <uadi32k.h>

int artic_rx_mon_count 0;

REMARKS

This function returns the number of ARINC data words received on all channels since the total buffer size is 3800 bytes (380 ARINC WODS)

RETURN VALUE

Data count (0 to 380) Received Data Counter.

EXAMPLE

```
        /* CONTINUOUS mode receive operation interrupt handler: */

/* In this mode the board sends an IRQ each time the buffer is full and each time
the buffer is half-full. The only way to sense if we have a buffer full or half-full
condition is the Receive Count:
Let's assume the main program sent an ARTIC_RX_CONTNS command with
buffer size 200 words starting at 0x0000 and enabled interrupt on Receive
Complete. */
ARINC_WORD rx_buff[BUFFSIZE]; /* can be as big as you want */
int i = 0;
void intr_handler0
{
        if (artic_rcvbuf_full(channel) {
                if (artic_rx_mon_count O > 0) {
                /* half buffer full - read 1 st half: */
                artic_readbuf(&rx_buff[i],0x0000,200*9); i = i+1 900;
        }
        else {
                /* 2nd half full, read it /
                artic_readbuf(&rx_buff[i],0x0000+1800, 190*10);
        }
}
}
```

SEE ALSO

artic_dis_monitor	artic_ena_monhalf
artic_dis_monhalf	artic_ena_monfull
artic_dis_monfull	artic_mon_halfbuff_full
artic_mon_fullbuff	artic_bm_available
artic_bm_mode	

artic_set_board

DESCRIPTION

Select active board's address

USAGE

```
#include <uadi32k.h>

int artic_set_board (address pointer);
```

Address Pointer; the base segment of board address

REMARKS

This routine calls the PCI libraries and set the PHYSICAL board address in the OS. It calls functions from the PCI bridge IC. If it is able to open the device it reads an ARTIC-429 pattern at address 0x7E00. All subsequent functions and commands will refer to the board address.
If the board has it already open it returns a message "Device already Open".
This routine should be called ONLY at the beginning of the program.
According to the PCI interface manufacture multiple calls can cause a major system failure.

RETURN VALUE

ARTIC_SUCCESS Operation successful

ARTIC_NO_BOARD No board found at specified address.

EXAMPLE

```
if (arlic_set_board(0xD000) '=ARTIC_SUCCESS) {

prinff("Board not present\n");
}
eise {
artic_sys_id(&min, &maj);
prinif ("ARTIC-429 Version %d.%d at 0000:0000",min,maj);
}
```

SEE ALSO

artic_sys_id **artic_close_board**

artic_sys_id

DESCRIPTION

Get system ID and version number

USAGE

#include <uadi32k.h>

int artic_sysjd (major, minor);

int * major, major version number

int * minor minor version number

REMARKS

This routine reads the system ID string from the GSR section of the common memory. If it recognizes the ARTIC-429 signature, it reads the version number and loads it into the supplied integer pointers major and minor.

RETURN VALUE

ARTIC SUCCESS Operation successful

ARTIC_NO_BOARD No board signature found

EXAMPLE

if (artic_set_board(0xD000)!=ARTIC_SUCCESS)

```
printf(”Board not present /n”);
eise{
artic_sys_id(&min, &maj);
printf (“ARTIC-429 Version %d.%d at D000:0000”,min,maj); }
```

SEE ALSO

artic_set_board

artic_close_board

DESCRIPTION

This function call the PCI OS functions to close the opened board

USAGE

#include <uadi32k.h>

int artic_close_boar (void);

REMARKS

This routine calls the PCI OS setup and closes the BMC device.
This routine should be called ONLY at the end of program.
According to the PCI interface manufacture if no device is open the close routine can cause a major system failure.

RETURN VALUE

ARTIC SUCCESS Operation successful

ARTIC_NO_BOARD No board signature found

EXAMPLE

if (artic_set_board(0xD000)!=ARTIC_SUCCESS)

```
printf(”Board not present /n”);
eise{
artic_sys_id(&min, &maj);
printf (“ARTIC-429 Version %d.%d at D000:0000”,min,maj); }
```

SEE ALSO

artic_set_board

artic_tx_busy

DESCRIPTION

Return whether Transmit Channel is busy

USAGE

#include <uadi32k.h>

int artic_tx_busy (channel)

int channel; transmit channel number

REMARKS

This function reads the appropriate bit in the boards Channel Activity byte and returns whether the channel is currently executing a transmit command or if it is free to accept another command (other than STOP, of course). However, it is not mandatory to use this function before issuing a command since the command function will return ARTIC_CHNL_BUSY if the channel is busy. It is used more for display purposes in a software window, for example, in showing channel activity.

RETURN VALUE

YES Channel busy

NO Channel not busy

ARTIC_INV_CHNL Invalid or unconfigured channel

SEE ALSO

artic_rx_busy

artic_tx_complete

DESCRIPTION

Check if entire buffer was transmitted

USAGE

#include <uadi32k.h>

int artic_tx_complete (channel)

int channel; transmit channel number

REMARKS

This function checks the appropriate bit in the board's Status Register. This bit is set by the ARTIC board in both TX_NORMAL and TX_LOOP modes when the entire data in the buffer specified in the Transmit Command was sent out to the ARINC. After reading the bit, the function clears it for the next operation. This function should be used in a polled (non-interrupt) fassion to check if the buffer was transmitted. In interrupt mode it should be called from the Interrupt Service Routine to identify the source of the interrupt.

RETURN VALUES

YES Transmit fuffer is complete

NO Transmit buffer not complete

ARTIC_INV_CHNL Invalid or unconfigured channel

EXAMPLE

/* polled transmit operation: transmit 3 ARINC words on channel 1 */

```
ARINC_WORD tx_buff[3] = {0xl 1111111, 0x22222222, 0x33333333 };
/*transfer data into the board's memory starting at offset 0x0000: */
artic_writebuf(0x0000,tx_buff, 3*4);
artic_tx_cmd(1,TX_NORMAL,0x0000,3,O); /* start transmission */
while (artic_tx_complete(1 )== N0); /* wait for transmit complete */
printf ("3 words transmitted.\n"); I complete */
```

SEE ALSO

artic_tx_count

artic_tx_count

DESCRIPTION

Read Transmit Counter value

USAGE

#include <uadi32k.h>

int artic_tx_count (channel)

int channel; the transmit channel number (1-4)

REMARKS

This function returns the number of transmitted data words modulo the buffer size since the last transmit command, i.e. reiative to the beginning of the buffer. For example, if the buffer size specified in an TX_LOOP command is 100 and 230 words were transmitted since the command was accepted, this function will return 30.

RETURN VALUE

Transmit data counter value

ARTIC_INV_CHNL Invalid or unconfigured channel

EXAMPLE

```
artic_txcmd(channel,TX_NORNAL,0x0000, 100,0);

while (!artic_tx_completeO)
printf("\r %d words transmitted", artic_tx_count(channel));
/* tx complete, display final transmit count (should be 100)*/
printf("\rTransmit complete, total %d words transmitted";
        artic_tx_count(channel));
```

SEE ALSO

artic_rx_count

artic_tx_complete

artic_tx_cmd

DESCRIPTION

Send Transmit Command to an ARTIC board
ARINC429 (channel # 1-4) & 708 (channel 9) & 717 (channel 10-11)

USAGE

#include <uadi32k.h>

int artic_tx_cmd (channel, command, start, count, delay);
int channel; The transmit channel number (ARINC 429 1-4) (ARINC 708 – 9) (ARINC 717 - 10,11)

WORD command;
Command code: ARTIC_TX_NORMAL,ARTIC_TX_LOOP, ARTIC_TX_REPEAT or ARTIC_TX_STOP
WORD start,; Location (offset from board base addresss) of the first transmit data word (0x0000 to 0x7D00 - don't care if command = ARTIC_TX_STOP).
WORD count; Data Count: number of ARINC data words to be transmitted, starting at address start. (1 to 9600)
WORD delay, Pre-transmission delay in milioseconds

REMARKS

This routine sets-up a Transmit Command in the board's Transmit 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 ARTIC's memory, normally by using the artic_writebuf function, at the address offset start. If command is equal to ARTIC_ TX_NORMAL, ARTIC will initiate a time delay of delay microseconds after which it will transmit count words starting at address start. If command is equal to ARTIC_TX_LOOP, ARTIC 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 an ARTIC_ TX_STOP command). If command is equal to ARTIC_TX_STOP, all the other parameters are ignored and transmission (if any) is stopped on the channel.
If command is equal to ARTIC_TX_REPEAT it will transmit the block the number of times defined in the reapeat register. At the end it will stop and clear all internal registers.

RETURN VALUE

ARTIC_SUCCESS command was accepted: execution started.

ARTIC_INV_CHANNEL invalid or unconfigured channel specified

ARTIC_INV_ADDRESS start address does not fall in the 0-7D00 range

ARTIC_INV_SIZE count is such that the last data word would fall outside of the 7D00 range (start + count> 00).

ARTIC_CHN L_BUSY command=ARTIC_TX_NORMAL or ARTIC_ TX_LOOP and the channel or its twin channel is already transmitting.

ARTIC_BOARD_RESP board did not respond to the command within the response interval defined by _ARTIC_RESP_ TIMEOUT

EXAMPLE

See TXCMD.C example program

SEE ALSO

artic_tx_complete

artic_tx_count

artic_ei_txcomplete

artic_txrate_high

DESCRIPTION

Set HIGH Transmit Data Rate

USAGE

#include <uadi32k.h>

int artic_txrate_high (channel);
int channel; the receive channel number (1-4)

REMARKS

This routine sets the Transmit Data Rate to HIGH (100 Kilobit/sec) for the specified channel. The HIGH data rate will be applied at the NEXT transmit command, i.e. if one of the channels is currently transmitting at a LOW data rate, the data rate will not be changed "on the fly".

RETURN VALUE

ARTIC_SUCCESS Successful

ARTIC_INV_CHNL Invalid or unconfigured channel

EXAMPLE

```
        /* Input data rate from user: *1 char s[10];

int channel = 3;
 do {
printf("Enter Transmit Data Rate [High='H' or Low='L' ");
gets(s);
} while ( (*s !='H') && (*s!= 'L') );
if  (*s == 'H')
        artic_txrate_high(channel);

else
artic_txrate_low(channel);
```

SEE ALSO

artic_txrate_low

artic_txrate_low

DESCRIPTION

Set LOW Transmit Data Rate

USAGE

#include <uadi32k.h>

int artic_txrate_low (channel)
int channel; the receive channel number (1-4)

REMARKS

This routine sets the Transmit Data Rate to LOW (12.5 Kilobitlsec) for the specified channel. The LOW data rate will be applied at the NEXT transmit command, i.e. if one of the channel is currently transmitting at a HIGH data rate, the data rate will not be changed "on the fly".

RETURN VALUE

ARTIC_SUCCESS Successful

ARTIC_INV_CHNL Invalid or unconfigured channel

EXAMPLE

```
        /* Input data rate from user: 3 char s[1O];
int channel = 3;
do {
        printf("Enter Transmit Data Rate [H]igh or [L]ow: ");
        gets(s);
}while( (*s!= 'H')&&(*s!= 'L') );
if (*s==H)
        artic_txrate_high(channel);
eise
        artic_txrate_low(channel);
```

SEE ALSO

artic_txrate_high

artic_val_rx_chan

DESCRIPTION

Validate Receiver channel number

USAGE

#include <uadi32k.h>

int artic_val_rxchan (channel);

int channel; channel number

REMARKS

This routine checks if the specified number represents a valid configured Receiver Channel on the currently selected board by reading the board's Channel Configuration Register.

RETURN VALUE

YES Valid channel

NO Invalid channel

EXAMPLE

/* Try to receive 10 words on every available channel */

```
ARINC429_WORD buffer[4][1 0]; /*4 buffers of 10 word each */
for (chan=1; chan<=4; chan++) {
if (artic_val_rx_chan(chan)==YES) {
        artic_rx_cmd(chan, RX_NORMAL, buffer[chan-1 1 10);
        }
}
```

SEE ALSO

artic_val_tx_chan

artic_val_tx_chan

DESCRIPTION

Validate Transmitter channel number

USAGE

#include <uadi32k.h>

int artic_val_tx_chan (channel);

int channel; channel number

REMARKS

This routine checks if the specified number represents a valid configured Transmitter Channel on the currently selected board by reading the boards Channel Configuration Register.

RETURN VALUE

YES Valid channel

NO Invalid channel

EXAMPLE

1* count # of valid channels on the board *1

```
for (n=O,chan=1; chan<=4; chan++) {
if (artic_val_tx_chan(chan)) {
)
}
printf ("%d Transmitter Channels found\n",n);
```

SEE ALSO

artic_val_rx_chan

artic_writebuf

DESCRIPTION

Write a section of ARTIC memory

USAGE

#include <uadi32k.h>

int artic_writebuf (offset, src, byte_count) WORD offset

VOID *src;

WORD byte_count;

REMARKS

This function is used to transfer a block of data into the ARTIC board Dual-Ported memory. The most typical use is to transfer Transmit Data to the board before issuing a Transmit Command. This function should be used rather than direct memory operations in order to maintain compatibility with drivers for protected-memory operating systems.

RETURN VALUE

ARTIC_SUCCESS Transfer complete

EXAMPLE

SEE artic_tx_cmd example

SEE ALSO

artic_readbuf

arinc708_sel_channel

DESCRIPTION

Select data ARINC 708 output channel in the board

USAGE

#include <uadi32k.h>

int arinc708_sel_channel(int channel)

int channel; // 0,1,2 or 3

REMARKS

This function select data Input/Output in the board.
0 – channel A
1- channel B
2- internal loop back
3- TTL output

RETURN VALUE

ARTIC_SUCCESS or ARTIC_INV_CHANNEL

EXAMPLE

SEE ALSO

arinc708_tx_repeat(WORD repeat)
arinc708_Harvard_NRZ(int error, int clr_set)

arinc708_tx_repeat

DESCRIPTION

Define number of times the predefined ARINC 708 message will be transmited

USAGE

#include <uadi32k.h>

arinc708_tx_repeat(WORD repeat)

WORD repeat ;

REMARKS

This function accepts any value bigger than zero

RETURN VALUE

ARTIC_SUCCESS or ARTIC_INV_PARAM

EXAMPLE

SEE ALSO

arinc708_sel_channel(int channel)
arinc708_Harvard_NRZ(int error, int clr_set)

arinc708_Harvard_NRZ

DESCRIPTION

The function enable or disable error injection ARINC 708 during data transmission

USAGE

#include <uadi32k.h>

arinc708_Harvard_NRZ(int error, int clr_set)

int error - 1 - Sync error value 1
2 - Manchester Code value 2

int clr_set - value 1 enable error injection
value 0 disable error injection

REMARKS

This function returns invalid parameter for incorrect values.

RETURN VALUE

ARTIC_SUCCESS or ARTIC_INV_PARAM

EXAMPLE

SEE ALSO

arinc708_sel_channel(int channel)
arinc708_tx_repeat(WORD repeat)

arinc717_val_channel

DESCRIPTION

Check channel availability

USAGE

#include <uadi32k.h>

int arinc717_val_channel(int channel)

int channel; // 1 or 2

REMARKS

This function select data Input/Output in the board.
1 – first ARINC717 channel
2 – second ARINC717 channel

RETURN VALUE

ARTIC_SUCCESS ARTIC_INV_CHANNEL

EXAMPLE

SEE ALSO

arinc717_act_channel(int channel)
arinc717_tx_repeat(WORD repeat)
arinc717_Harvard_NRZ(int error, int clr_set, int channel)
artic717_txrxrate_set(WORD rate,int channel)
arinc717_int_loop (int set, int channel)

arinc717_act_channel

DESCRIPTION

Check channel operational mode – transmit receive

USAGE

#include <uadi32k.h>

int arinc717_val_channel(int channel)

int channel; // 1 or 2
int receive; 1 (0 – check transmit operational status))

REMARKS

This function select data Input/Output in the board.
1 – first ARINC717 channel
2 – second ARINC717 channel

RETURN VALUE

ARTIC_SUCCESS ARTIC_INV_CHANNEL , NULL – idle state

EXAMPLE

SEE ALSO

arinc717_val_channel(int channel) ;
arinc717_tx_repeat(WORD repeat)
arinc717_Harvard_NRZ(int error, int clr_set, int channel)
artic717_txrxrate_set(WORD rate,int channel)
arinc717_int_loop (int set, int channel)

arinc717_tx_repeat

DESCRIPTION

Define number of times the predefined ARINC 717 message will be transmited using transmit REPEAT command

USAGE

#include <uadi32k.h>

arinc717_tx_repeat(INT repeat, INT channel)

WORD repeat ;

REMARKS

This function accepts any value bigger than zero

RETURN VALUE

ARTIC_SUCCESS or ARTIC_ INV_CHANNEL

EXAMPLE

SEE ALSO

arinc717_act_channel(int channel)
arinc717_val_channel(int channel)
arinc717_Harvard_NRZ(int error, int clr_set, int channel)
artic717_txrxrate_set(WORD rate,int channel)
arinc717_int_loop (int set, int channel)

arinc717_Harvard_NRZ

DESCRIPTION

The function select between Harvard or NRZ data ARINC 717 during data data receive operation
The procedure set NRZ bit 4 for Channel 1 and bit 11 for channel 2 .
There are independent Harvard and NRZ transmission output signals on board D-type connector.

USAGE

#include <uadi32k.h>

arinc717_Harvard_NRZ(int nrz, int channel)

int nrz - 0 – Hravard bi-phase enconding
1 – NRZ

REMARKS

This function returns invalid parameter for incorrect values.

RETURN VALUE

ARTIC_SUCCESS or ARTIC_ INV_CHANNEL or ARTIC_ INV_PARAMETER

EXAMPLE

SEE ALSO

arinc717_act_channel(int channel)
arinc717_val_channel(int channel)
arinc717_tx_repeat(WORD repeat)
artic717_txrxrate_set(WORD rate,int channel)
arinc717_int_loop (int set, int channel)

artic_717_txrxrate_set

DESCRIPTION

Set ARINC 717 transmission baud rate.

USAGE

#include <uadi32k.h>

int artic717_txrxrate_set(WORD rate,int channel);
int channel; the channel number (1-2)
WORD rate; transmission baud rate : 1,2,3,4 >>> 64 – 128-256 or 512 ARINC 717 words (12 bits) p/second

REMARKS

The ARTIC 717 board supports four differents transmission/receive baud rate.
The procedure set the value (1,2,3,4) into the bits (Channel 1 bits 0-2, channel 2 bits 8-10) of register GSR717.PARAM.

RETURN VALUE

ARTIC_SUCCESS Successful

ARTIC_INV_CHNL Invalid or unconfigured channel

ARTIC_ INV_PARAMETER - incorrect baud rate selection

EXAMPLE

SEE ALSO

arinc717_act_channel(int channel)
arinc717_val_channel(int channel)
arinc717_tx_repeat(WORD repeat)
arinc717_Harvard_NRZ(int error, int clr_set, int channel)
arinc717_int_loop (int set, int channel)

arinc717_int_loop

DESCRIPTION

The function enable or disable receive internal loop. The unit receive data ARINC 717 during data transmission.
The function can be useful for board BIT.

USAGE

#include <uadi32k.h>

arinc717_int_loop (int set, int channel)
set 1 enable
set 0 disable

REMARKS

This function returns invalid parameter for incorrect values.

RETURN VALUE

ARTIC_SUCCESS or ARTIC_ INV_CHANNEL or ARTIC_ INV_PARAMETER

EXAMPLE

SEE ALSO

arinc717_act_channel(int channel)
arinc717_val_channel(int channel)
arinc717_tx_repeat(WORD repeat)
arinc717_Harvard_NRZ(int error, int clr_set, int channel)

arinc717_rx_index

DESCRIPTION

The function read dual port memory data store index.

USAGE

#include <uadi32k.h>

arinc717_rec_index (int channel)

REMARKS

This function returns the memory store index.

RETURN VALUE

A WORD with the memory location

EXAMPLE

SEE ALSO

arinc717_act_channel(int channel)
arinc717_val_channel(int channel)
arinc717_tx_repeat(WORD repeat)
arinc717_Harvard_NRZ(int error, int clr_set, int channel)