

# Integrator Readout Interface: Version 5

IFAE 2003

## Reference Manual

### Table of Contents

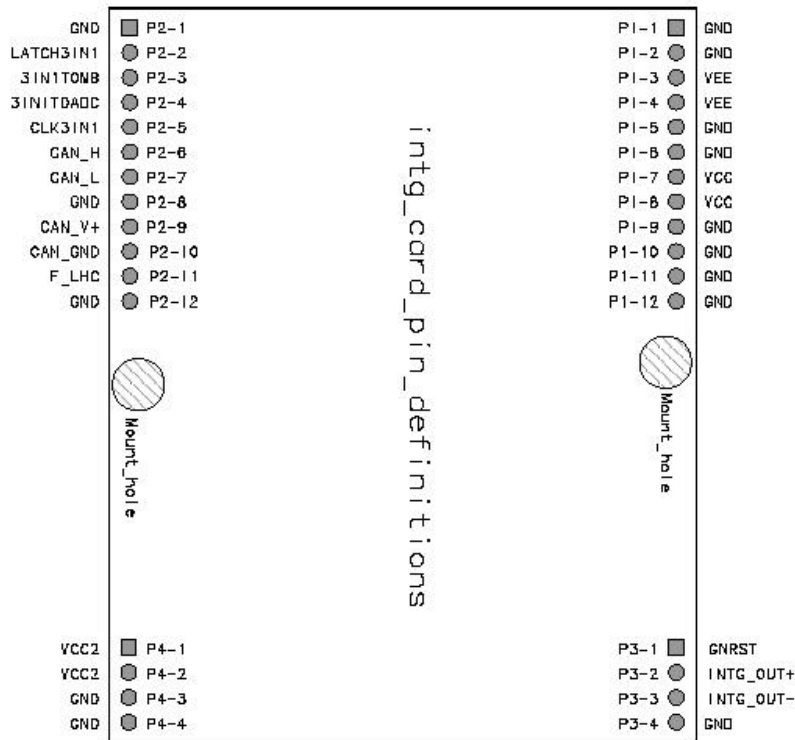
1.	Hardware Description.....	2
2.	Basic Operation and minimum requirements.....	3
3.	Memory Structure .....	4
4.	Identifier allocation an startup procedure.....	6
5.	Initialization.....	7
6.	In system reprogrammability of the DAQ section. ....	7
7.	DAQ Mode.....	8
8.	Commands Supported.....	10
9.	C library.....	17
10.	Example.....	21
11.	Revision history.....	21
12.	Troubleshooting guide.....	22
	12.1 The board cannot be idallocated. ....	22
	12.2. The cabling is correct but the board still can't be idallocated. ....	<b>Error! Bookmark not defined.</b>
	12.3. The cable is correct, the termination is correct, the power supply is o ..	<b>Error! Bookmark not defined.</b>

# 1. Hardware Description

The integrator readout interface is a small mezzanine board designed to sample voltage signals available at its differential input. This board must be plugged on a dedicated mother board. The version described here applies to those boards loaded with the firmware version 4. The core of the board is a SAE-C501 microcontroller (8051 architecture), running at a clock speed of 20MHz that is derived from a 40 MHz input clock. The peripherals that it controls are:

- A SAE81C91 CANbus controller chip, used as communication interface (runs at 250kbps).
- A MAX190BCWG 12 bits ADC, used to sample the integrators voltages.
- An AD558 8 bits DAC, used to adjust a global pedestal for the integrators signals.
- A serial port used to communicate with the 3in1 cards (bidirectionnal).

The program is located in two flash memories in a way to support a full in system programmability. An additional SRAM is also supported for temporary data storage.



## 2. Basic Operation and minimum requirements.

The board has to be plugged into a motherboard that matches the pin definition as defined at:

[http://www.pc.ifaes.Tilecal\\_Electronics/intg\\_card.pdf\\_2.pdf](http://www.pc.ifaes.Tilecal_Electronics/intg_card.pdf_2.pdf)

The Tilecal motherboard is equipped with low profile Harwin pins, however on other motherboard standard pins can be used as well. For proper operation the following power supplies are required:

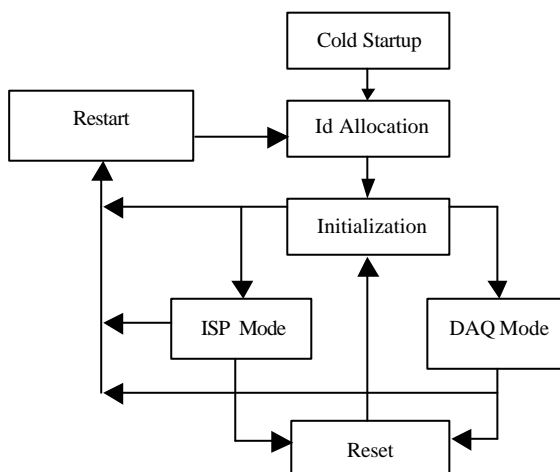
VCC: +5.00 VDC  
VCC2: +12.00 VDC to +15.00 VDC  
VEE: -5.00 VDC to -12.00 VDC

The CANbus interface requires an external power supply as well for proper communication:

CAN\_V+ to CAN\_V-: 8.00 VDC minimum  
16.00 VDC maximum

Care must be taken to avoid excessive voltage drops inside the CANbus cable when long cables are used and/or many nodes are tied to the cable. The standard cable to be used is Belden 9842 for single can lines and Belden 9844 for dual CAN lines (Tilecal case). ATLAS requires a special cable that is halogen free.

The board performs a *cold startup* on power up or if the mother board pulses the GNRST pin. To insure a proper cold startup during the power up, a fast rising edge on the +5V power supply must be insured. The boards have serial number burned in the flash memories that can be used to identify them. A global state machine is implemented as follows:



Upon startup, the board enters the *Identifier Allocation state*. In this state the board can only process broadcasted IDALLOC commands (CAN identifier equal to zero). The IDALLOC broadcast command contains as first parameter the serial number of the targeted card, and as second parameter the base address that the card should use. All cards connected on the bus, that are in Id Allocation state, will receive the broadcasted IDALLOC command. The card that has a serial number matching the first parameter will set its base address to the base address sent as second parameter. The identifier allocation procedure ends with the transmission of an IDALLOC acknowledgement from the ADC card to the host, using the new base address.

Once the base address is allocated, the board enters the *Initialization state*. Two modes can be initialized, the first one being the *In System Programmability* mode, while the other is the *DAQ* mode. The first mode allows to upload, modify and download a new DAQ firmware in system through the CAN link. The second mode is the standard mode used to sample the integrators outputs; additionally, the DAQ mode permits also to upload, modify and download a new ISP firmware. These two firmwares, ISP and DAQ, reside in the Flash memory A and B, respectively: while a piece of code is executed in one of the memories, the other can be modified. It must be noted that the ISP features must be reserved to experts as the download of a wrong firmware will cause an unrecoverable error.

The board might enter again in the Initialization State by means of a *Reset command*, and might enter in the Idallocation state by means of a *Restart command*. It must be pointed out that the reset command does not allow to reallocate the based address: this can only be done at cold startup, by pulsing the GNRST line or by sending the *Restart command*. In Tilecal, the GNRST line is pulsed through the TTC optical fiber.

### 3. Memory Structure

Two distinct memory spaces are defined: Program and Data. At startup, the program is fetched in the FLASH A memory that covers the lower 16Kbytes, and the Flash B is seen as data memory (Figure 1). The Startup Section of Flash A configures the CANbus for 500Kbps and supports the CANbus message processor, the identifier allocation and initialization routines, and the in system programmability routines as well.

	PROGRAM	XDATA
0xE000 - 0xFFFF	N/A	CAN Registers
0xD000 - 0xDFFF	N/A	ADC Registers
0xC000 - 0xCFFF	N/A	DAC Register
0x8000 - 0xBFFF	Not Available	SRAM
0x4000 - 0x7FFF	Not Available	DAQ Section (Flash B)
0x0000 - 0x3FFF	Startup Section (Flash A)	Not Available

**Figure 1: ISP Mode memory configuration**

Once the IDALLOC command has been issued successfully, the Startup Section keeps waiting for the CAN messages that will initialize the system in one of these modes:

- Initialization of the ISP Mode: to allow for in system reprogramming of the Flash B section
- Initialization of the FB Mode: to start executing the code located in Flash B that was previously reprogrammed within the ISP Mode.

The ISP mode uses the ISP memory configuration to reprogram the DAQ section, that is seen as external data. Once a valid firmware is loaded in the DAQ section, the system must be reinitialized with a new CAN message that will indicate that the DAQ section must be called. Upon reception of this second initialization command, the system swaps to the DAQ mode memory configuration (Figure 2):

	PROGRAM	XDATA
0xE000 - 0xFFFF	N/A	CAN Registers
0xD000 - 0xDFFF	N/A	ADC Registers
0xC000 - 0xCFFF	N/A	DAC Register
0x8000 - 0xBFFF	Not Available	SRAM
0x4000 - 0x7FFF	DAQ Section (Flash B)	Not Available
0x0000 - 0x3FFF	Not Available	Startup Section (Flash A)

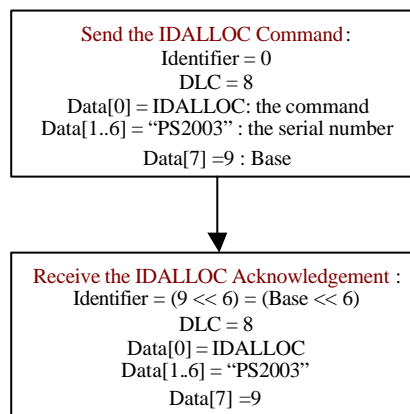
**Figure 2: DAQ mode memory configuration**

In this new mode, the program is fetched from the Flash B, and the Flash A that contains the Startup Section can be reprogrammed or monitored. This scheme will allow to look for bit upsets into the Flash A in radioactive environments. If this occurs, the system will be able to download the Startup Section through CAN and to reprogram it into the Flash A, insuring in this way that the board will always be able to boot up during a cold restart. If a bit upset occurs in the Flash B section, the board will most likely fail: in this case the system must be shut down and restarted to execute from scratch the ISP application, that will allow to download the DAQ Section again and will then recover from its failing state. Only a bit upset in the CAN message processor that is located into the Flash A is able to cause a definitive failure because the DAQ section loses the communication to download again the good ISP section.

#### 4. Identifier allocation an startup procedure.

At cold startup, i.e. after cycling the power or pulsing the GENRST input, the board enters the *Identifier Allocation State*. Each card is identified by its serial number, and all cards have an unique serial number. In order to communicate over the CANbus with the card, a base address must be allocated to the card identified by its serial number. In this way, no DIP switches settings are required prior to installation of the cards, providing a complete flexibility during their installation in the detector. The base addresses are allocated by software. A database of the the cards positions referred to their serial number is required.

A card with the serial number “PS2003” sharing a CANbus branch with other cards must be id allocated to use the base address 9 as follows (refer to the IDALLOC Command and Acknowledgement in the Commands section):



The IDALLOC acknowledgement must be received successfully in order to access the rest of commands. Once the IDALLOC command has been processed successfully on a given card, this card must reset by hardware (GENRST input) to be able to allocate a new base address. 16 cards can share a common CANbus branch, that implies that the Base Address must be comprised between 1 and 16.

Once the Base address has been allocated successfully, all next messages use the following 11 bits CAN identifiers structure:



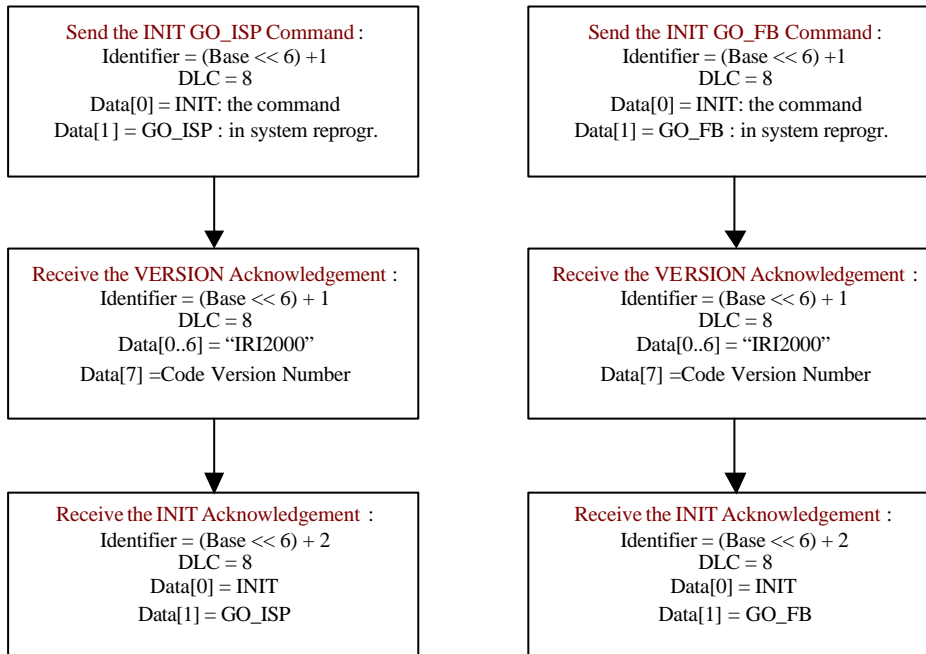
The identifiers are calculated as follow:  $Id = (Base \ll 6) + Offset$ .

The board can be reset in the following ways:

- Cycling the power. To insure proper operation the power supply should be kept off for at least 5 seconds.
- Pulsing the GNRST line. To reset the card, this input must be brought to VCC for at least 1 second, and then brought back to GND. Once at GND, the application should foresee 1 second boot time prior to attempting initialization.

## 5. Initialization

The board can be initialized in ISP or DAQ state. The procedure is identical for the two possible configurations. The first acknowledgement indicates a valid INIT command was received, and indicates which code version is in use. The second acknowledgement indicates that the ISP or DAQ section was started successfully. A RESET Command must be sent prior to sending a new INIT command.

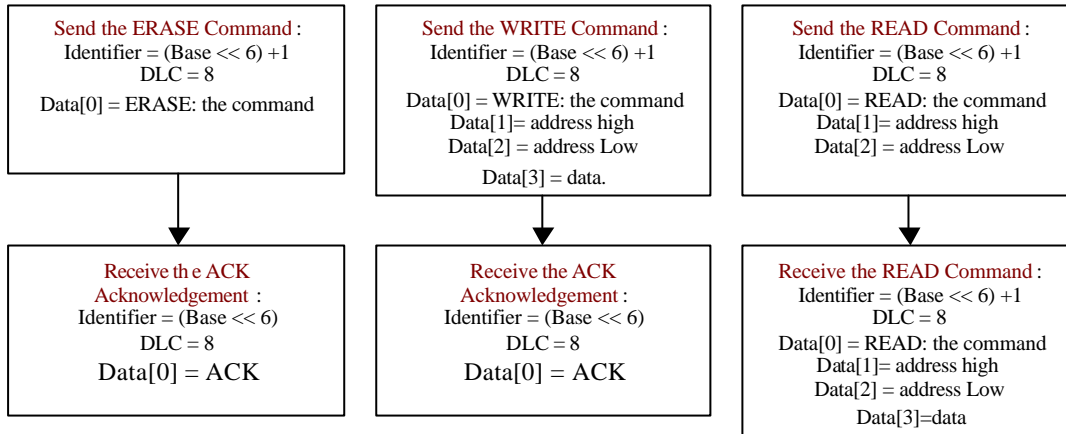


## 6. In system reprogrammability of the DAQ section.

Once entered the ISP mode, it is possible to read and modify the contents of the Flash B area (0x4000 to 0x7FFF) that contains the DAQ section. Three commands are available:

- Read: returns the contents of a given memory location. used to blankcheck and to verify the contents of the DAQ section.
- Erase: erases the whole Flash B. Once the command has been issued, it is necessary to read the content of a memory location of the DAQ section until the read value is 0xFF.
- Write: writes a data into a memory location of the DAQ section.

The erase and write commands are acknowledged with an ACK command sent back by the ADC card. The read command does not send back an ACK command, as it returns the read value in a READ message.



## 7. DAQ Mode.

The DAQ is the normal operating mode. It is used to communicate with the 3in1 cards through the 3in1 serial interface, and to sample the voltages of the integrators.

- **Communication with the 3in1 cards.**

Two functions are used to communicate with the 3in1 cards. This is done by means of a 16 bits pattern that can be shifted into the mother board PLD, or that can be read from it. The function CANSET shifts in the bit pattern, while the function CANGET shifts it out. Refer to the TTC commands manual for a complete information about the coding of the 16 bits bit pattern.<sup>1</sup>

- **Adjusting the global pedestal**

The output voltage of the integrators can exhibit a negative pedestal, that cannot be measured by the ADC. To avoid this source of errors, a global pedestal can be added to this voltage, insuring the sampled voltage is always positive. This is achieved with the DACSET command, that controls an 8 bits DAC. The adjustment ranges from -0.5V to +0.5V.

- **Adjusting the sampling delay**

Once a card has been selected, a time is required to get a stable input voltage at the ADC input. This waiting time can be set with the DELAY command in step of 8 usec. This delay is issued for each conversion, with the CONVERT command and the TRIGGER command, as well as in automatic scan mode.

- **Requesting parameters**

To read back a given parameter, a REQUEST command must be issued. Parameters that can be requested are: TIMER, NPMT, MAXSCAN, PMTLIST, DELAY; a non programmable parameter, the serial number, can also be requested in this way.

<sup>1</sup> The 3in1Get function in Version 5 does not toggle the 3in1Latch line before shifting in the 16 bits pattern.

- **Performing a single conversion**

The CONVERT command is used to perform an unique sampling. It samples the data and sends it back.<sup>1 2</sup>

- **Single scan**

A list of cards to scan can be programmed by means of the PMTLIST and NPMT commands. A maximum of 48 cards can be sampled. Also, a maximum of scans can be set with the MAXSCAN command. To perform scans, the pmtlist must be built, the number of pmts it contains ust be set accordingly, and the maximum number of scans must be set to a non zero value.

Then, the TRIGGER command will cause the card to perform one CONVERT call on each card that was programmed. The data is formatted in messages whose identifiers offsets range from 2 to 13. Up to 4 results can be transmitted per message. The number of messages is calculated by:

$$nmsg = \text{ceil}(npmt/4).$$

- **Automatic scans**

A constant rate of scans can be triggered by means of the START and STOP commands. The trigger rate is defined by the TIMER command. All the configuration steps of CONVERT and TRIGGER apply. Then, the START command will cause the card to perform periodic TRIGGER calls at a rate defined by the TIMER parameter. The data is formatted and sent in the same way as for the TRIGGER function, until a STOP command is issued or the maximum number of scans is reached.

---

<sup>1</sup> The CONVERT function in Version 4 or higher does not perform the cardselect to insure the compatibility between minimum bias monitoring and physics data taking. Hence the function does not require any parameter to be sent, and returns the CONVERT command code followed by 2 result bytes, plus 5 unused bytes.

<sup>2</sup> The CONVERT function in Version 5 or higher returns a 3 bytes DLC CAN Frame instead of 8 bytes in order to improve minimum bias monitoring sampling rate.

## 8. Commands Supported.

Description IDALLOC Command - Broadcast command to set dynamically a base address to a board that has a given serial number		
Identifier	0	
Direction	Host -> ADC	
Byte 0	IDALLOC	
Byte 1	serial number	ASCII code #1
Byte 2	serial number	ASCII code #2
Byte 3	serial number	ASCII code #3
Byte 4	serial number	ASCII code #4
Byte 5	serial number	ASCII code #5
Byte 6	serial number	ASCII code #6
Byte 7	Base address	1 to 16

Description IDALLOC Acknowledgement – Acknowledges the succesfull allocation fo a base address for a given card		
Identifier	$(\text{Base} \ll 6) + 0$	
Direction	ADC -> Host	
Byte 0	IDALLOC	
Byte 1	serial number	ASCII code #1
Byte 2	serial number	ASCII code #2
Byte 3	serial number	ASCII code #3
Byte 4	serial number	ASCII code #4
Byte 5	serial number	ASCII code #5
Byte 6	serial number	ASCII code #6
Byte 7	Base address	1 to 16

Description INIT Command – Configures the board in ISP or DAQ modes. The board sends back a VERSION acknowledgement. Afterwards the board acknowledges it has entered the requested mode by sending an INIT Acknowledgement.		
Identifier	$(\text{Base} \ll 6) + 1$	
Direction	Host -> ADC	
Byte 0	INIT	
Byte 1	ACTION	GO_ISP / GO_FB

<b>Description</b> VERSION Acknowledgement – This message is sent back as an acknowledgement to an INIT command, indicating its code version and the IRI2000 string (integrators readout interface 2000)		
Identifier	(Base << 6) + 1	
Direction	ADC -> Host	
Byte 0	'I'	ASCII code
Byte 1	'R'	ASCII code
Byte 2	'I'	ASCII code
Byte 3	'2'	ASCII code
Byte 4	'0'	ASCII code
Byte 5	'0'	ASCII code
Byte 6	'0'	ASCII code
Byte 7	Version code	1 for prototype

<b>Description</b> INIT Acknowledgement – This message is sent back as a second acknowledgement to an INIT command, indicating that the requested mode was started successfully.		
Identifier	(Base << 6) + 2	
Direction	ADC -> Host	
Byte 0	INIT	
Byte 1	ACTION	GO_ISP / GO_FB

<b>Description</b> ERASE Command – erase the DAQ section. This command can only be processed while in ISP mode. An ACK command is sent back.		
Identifier	(Base << 6) + 1	
Direction	Host -> ADC	
Byte 0	ERASE	

<b>Description</b> WRITE Command – programs a byte in the DAQ section. This command can only be processed while in ISP mode. An ACK command is sent back.		
Identifier	(Base << 6) + 1	
Direction	Host -> ADC	
Byte 0	WRITE	
Byte 1	ADDRESS HIGH	High Byte of the adress (0x40 to 0x7F)
Byte 2	ADDRESS LOW	Low Byte of the address
Byte 3	DATA	Data to program

<b>Description</b>	<b>READ Command – reads a byte from a memory location. Valid addresses are from 0 to 0x3FFF for Flash A, 0x4000 to 0x7FFF for Flash B and 0x8300 to 0xA300 for SRAM buffer. The boards send back the same message with the fourth byte containing the read data.</b>	
Identifier	(Base << 6) + 1	
Direction	Host -> ADC / ADC -> Host	
Byte 0	READ	
Byte 1	ADDRESS HIGH	High Byte of the address
Byte 2	ADDRESS LOW	Low Byte of the address
Byte 3	DATA	Contains the read data in the returned message

<b>Description</b>	<b>TIMER Command – Configures the trigger rate to be used for automatic triggers. <math>F_{trigger} = 20MHz / (12 * (65535 - N))</math>. This command can only be processed in DAQ mode. An ACK command is sent back.</b>	
Identifier	(Base << 6) + 1	
Direction	Host -> ADC	
Byte 0	TIMER	
Byte 1	N HIGH	High Byte of the timer parameter
Byte 2	N LOW	Low Byte of the timer parameter

<b>Description</b>	<b>NPMT Command – Configures the number of cards to be read at each trigger. This command can only be processed in DAQ mode. An ACK command is sent back.</b>	
Identifier	(Base << 6) + 1	
Direction	Host -> ADC	
Byte 0	NPMT	
Byte 1	N	1 to 48

<b>Description</b>	<b>MAXSCANS Command – Configures the maximum number of triggers during the automatic scans. This command can only be processed in DAQ mode. An ACK command is sent back.</b>	
Identifier	(Base << 6) + 1	
Direction	Host -> ADC	
Byte 0	MAXSCANS	
Byte 1	N HIGH	High Byte of the maxscans parameter
Byte 2	N LOW	Low Byte of the maxscans parameter

<b>Description</b> PMTLIST Command – Sets up a card address in the table of cards that are scanned at each trigger. This command can only be processed in DAQ mode. An ACK command is sent back.		
Identifier	(Base << 6) + 1	
Direction	Host -> ADC	
Byte 0	PMTLIST	
Byte 1	POSITION	Scan table index where to set up the new card (0 to 47)
Byte 2	3in1 Pattern High	High byte of Card Select Command
Byte 3	3in1 Pattern Low	Low Byte of Card Select Command

<b>Description</b> DACSET Command – Configures the global pedestal this added to integrator output. The pedestal added varies linearly between 0.5V (DACSET=0) and – 0.5V (DACSET=255). this command can only be processed in DAQ mode. An ACK command is sent back.		
Identifier	(Base << 6) + 1	
Direction	Host -> ADC	
Byte 0	DACSET	
Byte 1	N	0 to 255

<b>Description</b> DELAY Command – Configures delay between a card is selected and the ADC is triggered, in order to wait for a proper stabilization time. Each count adds 8usec delay. In automatic triggers mode, delay cannot exceed 5. For single conversions, delay can get any value. This command can only be processed in DAQ mode. An ACK command is sent back.		
Identifier	(Base << 6) + 1	
Direction	Host -> ADC	
Byte 0	DELAY	
Byte 1	N HIGH	High Byte of the delay parameter
Byte 2	N LOW	Low Byte of the delay parameter

<b>Description</b> CANSET Command – Sends a 3in1 control pattern to the serial interface, according to the 3in1 controls interface specification of the mother board. This command can only be processed in DAQ mode. An ACK command is sent back.		
Identifier	(Base << 6) + 1	
Direction	Host -> ADC	
Byte 0	CANSET	
Byte 1	N HIGH	High Byte of the bit pattern
Byte 2	N LOW	Low Byte of the bit pattern

<b>Description</b> CANGET Command – Reads a 3in1 control pattern from the serial interface, according to the 3in1 controls interface specification of the mother board. The request only requires byte 0 to be transmitted, and the board sends back the command with bytes 1 and 2 containing the read pattern. This command can only be processed in DAQ mode.		
Identifier	(Base << 6) + 1	
Direction	Host -> ADC / ADC -> Host	
Byte 0	CANSET	
Byte 1	N HIGH	High Byte of 3in1 Pattern
Byte 2	N LOW	Low Byte of 3in1 Pattern

<b>Description</b> REQUEST Command – Requests the ADC to send back a given parameter. The ADC board sends back the message appending the parameters list as they are sent with the respective commands to set them. This command can only be processed in DAQ mode.		
Identifier	(Base << 6) + 1 (host to adc), (Base << 6) + 14 (ADC to host)	
Direction	Host -> ADC / ADC to Host	
Byte 0	REQUEST	
Byte 1	COMMAND	Command code of the requested parameter
Byte 2	PARAM1	Returned parameter 1
Byte 3	PARAM2	Returned parameter 2
Byte 4	PARAM3	Returned parameter 3
Byte 5	PARAM4	Returned parameter 4

<b>Description</b> REQUEST SERIALNUM Command– Requests the ADC to send back its serial number. The ADC board sends back the message appending its serial number in ASCII format. This command can only be processed in DAQ mode.		
Identifier	(Base << 6) + 1 (host to adc), (Base << 6) + 14 (ADC to host)	
Direction	Host -> ADC / ADC to Host	
Byte 0	REQUEST	
Byte 1	SERIALNUM	Command code of the requested parameter
Byte 2	PARAM1	First character of the serial number string
Byte 3	PARAM2	
Byte 4	PARAM3	
Byte 5	PARAM4	
Byte 6	PARAM5	
Byte 7	PARAM6	Last character of the serial number string

<b>Description</b>	<b>CONVERT Command – Requests a conversion of the selected card. The board sends back the message appending the result to command code. This command can only be processed in DAQ mode.</b>	
Identifier	(Base << 6) + 1 (host to adc), (Base << 6) + 14 (ADC to host)	
Direction	Host -> ADC / ADC to Host	
Byte 0	CONVERT	
Byte 1	RESULT HIGH	High Byte of the conversion result
Byte 2	RESULT LOW	Low Byte of the conversion result

<b>Description</b>	<b>TRIGGER Command – Triggers a scan of the cards listed in the scanning table. The results are sent back with messages which offsets vary from 2 to 13. Each message contains up to 4 conversions. This command can only be processed in DAQ mode.</b>	
Identifier	(Base << 6) + 1	
Direction	Host -> ADC	
Byte 0	TRIGGER	

<b>Description</b>	<b>START Command – Start the automatic triggers to scan periodically the scanning table at the rate programmed with the TIMER command. Data sampled is sent back as for the TRIGGER command. The automatic triggers stop whenever the maximum number of scans is reached or a STOP command is transmitted. This command can only be processed in DAQ mode.</b>	
Identifier	(Base << 6) + 1	
Direction	Host -> ADC	
Byte 0	START	

<b>Description</b>	<b>STOP Command – Stops the automatic triggers. This command can only be processed in DAQ mode. An ACK command is sent back.</b>	
Identifier	(Base << 6) + 1	
Direction	Host -> ADC	
Byte 0	STOP	

<b>Description</b>	<b>RESET Command – Command to exit from the DAQ or ISP mode and return to the Initialization State.</b>	
Identifier	(Base << 6) + 1	
Direction	Host -> ADC	
Byte 0	RESET	

<b>Description</b> SAVE FLASH A Command – copies the contents of the Flash A into the SRAM in order to edit in system the Flash A contents. An ACK command is sent back. Only in DAQ Mode.		
Identifier	(Base << 6) + 1	
Direction	Host -> ADC	
Byte 0	SAVE_FA	

<b>Description</b> WRITE RAM Command – modifies a byte in the SRAM buffer when updating the Flash A contents. An ACK command is sent back. Only in DAQ Mode.		
Identifier	(Base << 6) + 1	
Direction	Host -> ADC	
Byte 0	WRITE_RAM	
Byte 1	ADDRESS HIGH	High Byte of the adress (0x40 to 0x7F)
Byte 2	ADDRESS LOW	Low Byte of the adress
Byte 3	DATA	Data to program

<b>Description</b> REPROGRAM Command – Starts the embedded algorithm that copies the SRAM buffer into the Flash A. This operation erases the Flash A, and then copies the buffer into the erased sector. <b>WARNING:</b> this function erases the CAN command processor and the boot section, so the user must make sure that the new program he's loading is able to handle again the CAN processor and that it is compatible with the flash B contents. The embedded algorithm can last few seconds. Once started, the board must remain powered until completion. An ACK message is sent back upon completion. Only in DAQ Mode.		
Identifier	(Base << 6) + 1	
Direction	Host -> ADC	
Byte 0	REPROGRAM	

<b>Description</b> ACK Acknowledgement – Command to acknowledge some commands that do not return any data otherwise.		
Identifier	(Base << 6) + 1	
Direction	AC -> Host	
Byte 0	ACK	

## 9. C library.

Two C libraries were developed to control and communicate with the ADC board:

- Using a PCICAN2 CANBus controller from National Instruments under Windows. The library project for Visual C++6.0 can be found at:

[http://wwwwpc.ifaef.es/Tilecal\\_Electronics/ADC%20Library/adclib.zip](http://wwwwpc.ifaef.es/Tilecal_Electronics/ADC%20Library/adclib.zip)

- Using a Readout Buffer VME board to be operated by a CETIA VMPC6a under LynxOS.

[http://wwwwpc.ifaef.es/Tilecal\\_Electronics/ADC%20Library/adclib03.c](http://wwwwpc.ifaef.es/Tilecal_Electronics/ADC%20Library/adclib03.c)

The Windows library is described here below. The LynxOS library differs in the parameters to be passed: the VME base address and Redaout Buffer channel to be used are 2 additional parameters common to all functions when used together with a readout buffer. The LynxOS library supports an additional function described at the end on the Windows library.

To support other CAN controllers (on Windows and LynxOS), three functions have to be adapted:

### Windows library:

- *adc\_init(char action, char branchname):*  
*To initialize the CANbus and to configure the masks and hardware filters The queue lengths configurations are specific to the PCICAN2 board and can be suppressed if other boards are used that do not support message queing.*
- *can\_tx(int id, char dlc, char rtr, char data[8]):*  
*To transmit a CAN message with identifier id, data length code dlc, remote frame flag rtr and 8 data bytes in data[8]. The function exits upon completion of the transmission.*
- *can\_rx(int \*id, char \*dlc, char \*rtr, char data[8]):*  
*To receive a CAN message. The functions waits for a message to come and returns the received identifier, data length code, remote frame flag and data fields. The PCICAN2 version of this function might exit also upon timeout (one second timeout).*

The other functions of the library use the *can\_tx* and *can\_rx* functions. These are briefly described:

- *adc\_idalloc (unsigned char serie[6], int ident)*  
*Identifier allocation call.*
- *adc\_erase(void):*  
*Requests the erasure of the DAQ Section in the Flash B, while in ISP mode. The host must afterwards poll any memory location of the DAQ section until it reads back the value 0xFF, that indicates that the embedded erase alorithm has completed.*
- *adc\_blankcheck(unsigned int \*adr):*  
*Checks that the DAQ section in the Flash B has all its memory locations set to 0xFF. If an error occurs the failing address location is returned in the \*adr pointer.*
- *adc\_writemem(int adr, char data):*  
*Writes data at address adr in the DAQ section of the Flash B while in ISP mode.*

- *adc\_readmem(unsigned int adr, char \*data):*  
Reads the contents of address *adr* in the DAQ section of the Flash B while in ISP mode. The read data is contained in the pointer *\*data*.
- *adc\_save\_fa(void):*  
Copy the 0x2000 lower address contents of the Flash A into the SRAM buffer located between 0x8300 and 0xA300.
- *adc\_writeram(unsigned int position, unsigned char data):*  
Writes data into a position of the SRAM buffer. *position* is the address of the Flash A target memory location.
- *adc\_reprogram\_fa(void):*  
Copy the contents of the SRAM buffer into the Flash A. **WARNING: this function erases the CAN command processor and the boot section, so the user must make sure that the new program he's loading is able to handle again the CAN processor and that it is compatible with the flash B contents. The embedded algorithm can last few seconds. Once started, the board must remain powered until completion.**
- *adc\_rate(unsigned short timer):*  
Write the timer setting for automatic triggers. The *timer* parameter can take values between 0 and 65535. The rate is defined as:

$$\text{Rate} = \frac{20}{12 \cdot (65536 - N)} \text{ MHz}$$

- *adc\_npmt(char npmt):*  
Set up the number of 3in1 cards contained in the superdrawer. This parameter is used during automatic triggers: at each trigger the ADC will select and sample the *npmt* cards which addresses are programmed in a cards scanning table with the *pmtlist* command.
- *adc\_maxscans(unsigned short maxscans):*  
Set up the maximum number of scans allowed during automatic triggers. Before starting the automatic triggers an internal counter is set to zero. At each trigger the counter is incremented and, whenever the number of triggers reaches the maximum number of triggers allowed, the ADC returns to idle state. This allows to insure that in case of communication loss or host failure, the ADC can return to a known default state in an autonomous way.
- *adc\_pmtlist(char pos, char zone, char sector, char card):*  
Sets up a card in the cards scanning list used for automatic triggers. The table has 48 positions. Each position has an entry for the *zone*, the *sector* and the *card* number. During automatic scans, the positions 0 to *npmt* are scanned. The position index *pos* can take values between 0 and 47.
- *adc\_dacset(char dac):*  
Sets the DAC output level that controls the global pedestal. The voltage level that enters the ADC can be shifted up or down by  $\pm 500\text{mV}$  in this way. The shift varies linearly from  $+500\text{mV}$  (*dac=0*) to  $-500\text{mV}$  (*dac=255*).
- *adc\_delay(unsigned short delay):*  
A delay can be programmed between the card selection and the ADC trigger operations, in order to provide more stabilization time after a card selection. Each increment adds an 8 usec delay. During automatic triggers the delay cannot exceed 5. During single conversions, the delay parameter can take any value, up to 65535 (0.5 sec).

- *adc\_3in1set(unsigned short dat);*  
Shifts the 16 bits of the *dat* pattern into the mother EPLD that communicates with the 3in1 cards. this function is useful to select and configure cards, and provides a full control over the 3in1 cards through the CANbus.
- *adc\_3in1get(unsigned short \*dat);*  
Shifts out of the mother board EPLD a 16 bit pattern that is returned in the *dat* parameter. This function is used to read back the bit patterns that are passed to the 3in1 cards with the *adc\_3in1set* function.
- *adc\_convert(unsigned short \*result);*  
Performs a single ADC conversion. The 12 bits result is returned in the *result* parameter.
- *adc\_trigger(char npmt, unsigned short result[48]);*  
Triggers a scan of the cards scanning table. The *npmt* cards that were programmed in the table using the *adc\_pmtlist* function are selected and sampled. The *npmt* results are transmitted back using messages with offset 2 to 13.
- *adc\_start(void);*  
Starts the automatic triggers. The rate of automatic triggers is set using the *adc\_rate* function. Once started, the ADC board generates internally *adc\_trigger* function calls. The boards exits this mode upon reception of an *adc\_stop* command or if the maximum number of scans that was set with the *adc\_maxscans* function is reached.
- *adc\_auto(char npmt, unsigned short result[48]);*  
Recovers the results of an automatic trigger, that is *npmt* conversions from messages 2 to 13. This function must be called periodically after the *adc\_start* command was sent to receive the data being transmitted by the ADC board, and this until it is called *maxscans* times, or until an *adc\_stop* function is sent.
- *adc\_stop(void);*  
Stops the automatic triggers.
- *adc\_get\_rate(unsigned short \*timer);*  
Reads back the *timer* parameter that defines the automatic triggers rate.
- *adc\_get\_npmt(char \*npmt);*  
Reads back the *npmt* parameter, that defines the number of valid cards in the scanning table.
- *adc\_get\_maxscans(unsigned short \*maxscans);*  
Reads back the maximum number of automatic scans.
- *adc\_get\_pmtlist(char pos, char \*zone, char \*sector, char \*card);*  
Reads back the *zone*, *sector* and *card* number of the position *pos* in the scanning table.
- *adc\_get\_delay(unsigned short \*delay);*  
Reads back the delay setting.
- *adc\_get\_serialnumber(unsigned char serialn[8]);*  
Read back the serial number.
- *long adc\_restart(void);*  
Hard software reset. Puts the card in a not idallocated state.
- *long adc\_reset(void);*  
Soft reset. Puts the card in a not initialized state.

### LynxOS library:

The LynxOS version of the library is identical to the Windows one, but it supports a special Convert function to be used together with the *rb\_getfastdump()* function specific of the Readout Buffer card:

```
// ADC communication protocol functions declarations
// In System Programability Functions

int  adc_init      (unsigned int rb, int  chn, int Base, char action, unsigned int version[8]);
long adc_erase    (unsigned int rb, int  chn, int Base );
long adc_blankcheck (unsigned int rb, int  chn, int Base, unsigned int *adr);
long adc_save_fa   (unsigned int rb, int  chn, int Base );
long adc_reprogram_fa(unsigned int rb, int  chn, int Base );
long adc_writeram  (unsigned int rb, int  chn, int Base, unsigned int position, unsigned char data);
long adc_writemem  (unsigned int rb, int  chn, int Base, int  adr, unsigned char data);
long adc_readmem   (unsigned int rb, int  chn, int Base, unsigned int adr, unsigned char *data);
long adc_get_serialnumber (unsigned int rb, int  chn, int Base, unsigned char serialn[8]);
int  adc_idalloc   (unsigned int rb, int  chn, unsigned char serie[7], int ident);

//          Readout Interface Control Functions

long adc_rate      (unsigned int rb, int  chn, int Base, unsigned short timer);
long adc_npmt      (unsigned int rb, int  chn, int Base, char npmt);
long adc_maxscans  (unsigned int rb, int  chn, int Base, unsigned short maxscans);
long adc_pmtlist   (unsigned int rb, int  chn, int Base, char pos, char card);
long adc_dacset    (unsigned int rb, int  chn, int Base, unsigned char dac);
long adc_delay     (unsigned int rb, int  chn, int Base, unsigned short delay);
long adc_3in1set   (unsigned int rb, int  chn, int Base, unsigned short dat);
long adc_3in1get   (unsigned int rb, int  chn, int Base, unsigned short *dat);
long adc_convert   (unsigned int rb, int  chn, int Base, unsigned short *result);
long adc_trigger   (unsigned int rb, int  chn, int Base, char npmt, unsigned short result[48]);
void adc_start     (unsigned int rb, int  chn, int Base );
void adc_auto      (unsigned int rb, int  chn, int Base, char npmt, unsigned short result[48]);
void adc_stop      (unsigned int rb, int  chn, int Base );
long adc_get_rate  (unsigned int rb, int  chn, int Base, unsigned short *timer);
long adc_get_npmt  (unsigned int rb, int  chn, int Base, char *npmt);
long adc_get_maxscans (unsigned int rb, int  chn, int Base, unsigned short *maxscans);
long adc_get_pmtlist (unsigned int rb, int  chn, int Base, char pos, char *card);
long adc_get_delay  (unsigned int rb, int  chn, int Base, unsigned short *delay);
void adc_reset     (unsigned int rb, int  chn, int Base);

// CAN communication functions

void can_tx        (unsigned int rb, int  chn, int id, int dlc, int rtr, unsigned int data[8]);
int  can_rx        (unsigned int rb, int *chn, int *id, int *dlc, int *rtr, unsigned int data[8]);

// Restart function

long adc_restart   (unsigned int rb, int chn, int Base);

// Dump mode functions

long adc_dm_convert(unsigned int rb, int  chn, int Base);
```

The *adc\_dm\_convert* function sends a convert request command but does not return any result. For proper operation, the user should have turned on previously the DUMP mode of the Readout Buffer and configured the interrupts accordingly so that upon reception of the CONVERT result in a REQUEST frame an interrupt is called and a FASTDUMP frame is pushed in the output fifo of the readout buffer. The user should refer to the Readout Buffer User Manual for details, available at:

[http://wwwpc.ifaes.Tilecal\\_Electronics/Rbuf/Documentation/Quadruple%20CANBus%20VME%20Readout%20Buffer.pdf](http://wwwpc.ifaes.Tilecal_Electronics/Rbuf/Documentation/Quadruple%20CANBus%20VME%20Readout%20Buffer.pdf)

## 10. Examples

Several examples are available on the production web site for Windows and LynxOS:

[http://wwwpc.ifaes.Tilecal\\_Electronics/fabricat.htm](http://wwwpc.ifaes.Tilecal_Electronics/fabricat.htm)

## 11. Revision history.

- Firmware version 3 used in Tilecal: initial production firmware, year 2000.
  - IRI2000.HEX firmware.
  - CAN runs at 500kbps.
  - Maximum CANbus length is 60m.
  - The convert function performs selection of a channel by sending a cardselect bit pattern through the 3in1 serial port.
- Firmware version 3 used in MAGIC : modified version 3 firmware.
  - DC\_MON.HEX firmware.
  - CAN runs at 250kbps
  - Maximum CANbus length is 200m.
  - The pmtlist table is extended to 96 entries instead of 48.
- Firmware Version 4 used in Tilecal: modified version 3 firmware.
  - IRI2002.HEX firmware.
  - CAN runs at 250kbps.
  - Maximum CANbus length (through superdrawers) is 180m including daisy chain sections.
  - The cardselect feature is removed from the convert function.
- Firmware Version 5 used in Tilecal: modified Version 4.
  - IRI2003.HEX firmware.
  - CAN runs at 250kbps.
  - The 3in1Get function does not toggle anymore the 3in1Latch line.
  - The CONVERT function sends back 3 bytes of data instead of 8.
  - A RESTART function allows to perform a hard software reset of the card. This function sets the card in a not idallocated state, as a cold startup.

## 12. Troubleshooting guide.

The ADC cards used in ATLAS and on the Testbeams have passed a complete quality control process and is fully documented in a database and in the production web page. Despite this troubles can appear usually in the form of communication loss. Here is a stepping list that aims to help solving problems.

***The board cannot be idallocated (“It doesn’t work!”):***

*Before opening the drawer:*

1. First make sure that the drawer is powered and that nobody else is trying to access the card at the same time. If it still fails then go to next step.
2. Is the CIS working properly?
  - a. YES: The drawer electronics is working fine, including the clock distribution and low voltage power supplies. Go to step 3.
  - b. NO: The whole drawer electronics is not working properly. Check the LHC clock distribution and the low voltage power supplies.
3. Are there more nodes on the link?
  - a. YES. Try the other nodes.
    - None are working.
      - SOLUTION: Check the CAN Controller you are using.
      - SOLUTION: Check the CANbus cable. Check that the canbus cable is properly terminated by measuring the resistivity between CAN\_H and CAN\_L: 60 ohms indicates continuity through all the bus, 120 ohms reveals that the cable is broken in one place, and high impedance shows broken cable in at least two places.
    - Other nodes are OK, but there is still one failing.
      - SOLUTION: if the failing node is the last one, make sure you get the proper CAN power voltage on that last node. Make sure the maximum length does not exceed 180m in total. Check the terminators.
  - b. NO. There is only one node.
    - SOLUTION: proceed as described in 3.a (none working).
4. I still have one node failing to start. Other nodes are OK, clock, LV and cable are OK.
  - a. Connect a laptop with a PCMCIA–CAN2 card somewhere in the link. Run the Bus Monitor tool to spy the bus traffic. If when trying to initialize you see the frames but the ADC does not answer, go to step 5.
  - b. If the ADC answers but the Readout Buffers does not see it, check your program.
  - c. You can use the SendCan program to manually send frames to the ADC and to try to see if it can answer to something.
  - d. You can connect the laptop close to the drawer and run SingleQC. If you fail to communicate with the card go to step 5.
5. There is a hardware failure.
  - a. You have to open the drawer and replace the ADC. The broken card must be returned to Barcelona for repair.