SCUBA 2 data acquisition software overview

Part two protocols

1. Summary

The purpose of this document is to provide detailed protocol for the communication between SCUBA2 data acquisition software and the Multi-channel Electronics (MCE). It is a preliminary description of the protocols. Some of the items have still to be finalised between the firmware and software implementers.

2. References

[1]" Data Link protocol.doc", Michael MacIntosh. UK Astronomy Technology Centre
[2] "DSP_interfaces_v3.doc", Chris Tierney, David Atkinson, Derek Ives, Steven Beard
UK Astronomy Technology Centre
[3]" SCUBA-2 Data Acquisition to Data Processing Interface", Dennis Kelly. UK Astronomy Technology Centre
[4] " BBInstructions.xls" Bryce Burger, UBC

3. Introduction

The approach to the design of the SCUBA2 data acquisition software is to reuse as much as possible of WFCam / Ultracam low level software developed at ATC so that the overall effort both in designing and maintaining can be largely saved by ATC (UK) and JAC (Hawaii). The common data acquisition layer for all three instruments is illustrated in Figure 1. They all use the Linux Operation system on a PC with the kernel modified to utilise the real-time Linux RTLinux from RTAI.
(http://www.aero.polimi.it/RTAI/rtai-24.1.13.tgz) . The communication between the data acquisition computer and the MCE is done through a SDSU PCI card with 200Mbits/s optic fiber. However, there are differences among these instruments, which inevitably lead to changes being made to our design. In the next sections, we outline the protocol used by the SCUBA2 in detail.

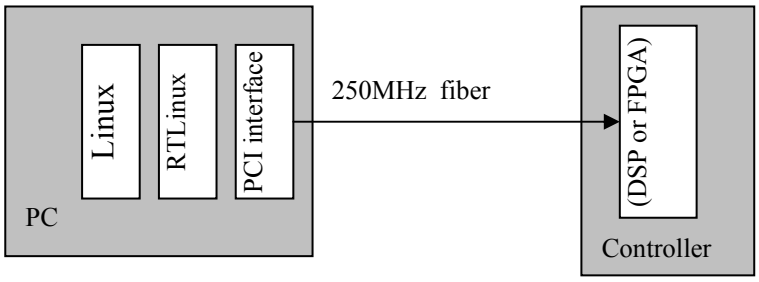


Figure 1 The low level data acquisition layer

4. The SCUBA2 low level protocol

As mentioned in section 3, we want to reuse as much as possible of the WFCam/Ultracam low level software, i.e. try to keep our protocol very similar to that used for WFCam/Ultracam. For SCUBA2, the data coming out of the controller will be in 32-bit format (But, in engineering mode, data may be in 16-bit format). For simplicity, the MCE must always send data in multiple of 4 bytes (32-bit). In order to make the protocol (see later) work, we first specify the operation of the MCE.

All the commands and data sent to the MCE from the RTL PC software are in *Little-endian* format, it is expected that all replies and data received from the MCE by the RTL PC software are in *Little-endian* format.

4.1 The MCE operation principles

The MCE will consist of several cards, including clock and control cards. It communicates with the SCUBA2 data acquisition system through a single bi-direction 250MHz optical fibre. There will be some FIFO buffers in the FPGA on the control board. Bytes of incoming data are written directly from the fibre(hotlink) receiver to the FPGA firmware block 'fibre_rx_fifo'. This block flags to subsequent blocks that there are bytes of data to be processed. The FPGA must accept commands on its fibre in a standard format (see section 4.2). There will also be a large buffer area for holding several array data frames. The data sent back to the data acquisition system could be derived from two sources: reply to commands and data for array readout. All packets of data originating from the MCE, whether command reply or frame data, must be preceded with a header, which indicates in which mode (reply or frame data) the data in the following packet is transmitted and the length of the packet.

To reliably distinguish the commands (MCE received) or the headers/replies (data acquisition system received, PCI DSP) from corrupted data, a special preamble or synchronisation of two 32-bits words shall be used, i.e. all commands to the MCE and all replies/ frame data from the MCE shall be preceded with the preamble words. In SCUBA 2, the frame data can in principle be any 32-bit (4 bytes) value. However, it is expected that the average of the data values returned will change only slowly as the sky background changes and that there will be no large changes from one data value to the next. Therefore, two 32-bit words, which are the logical inverse of one another, shall be adequate for the preamble. For added robustness, a checksum shall be added.

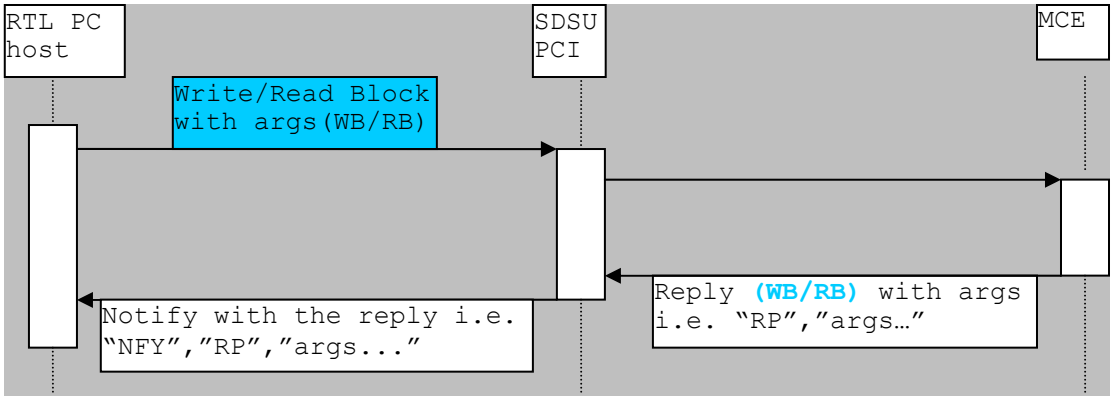
The MCE must generate a standard format reply upon commands implemented, with the exceptions: "**reset**" and "**start_application**", which generate replies before execution. Such standard replies indicate success or failure of the command.

Upon receiving a "start_application" command, the MCE begins to execute the current application. Until completion of this application, the *only* valid command that may be sent to the MCE is "**stop_application**". So_called "on-the-fly" modification or examination of parameters is not allowed (***may change later ?***). If the STOP command is received while the MCE is running an application, then the next frame sent back will have the 'last frame' flag set (i.e. bit 0), and also the 'stop' flag set (i.e.bit1) in the **frame status** word, thereby indicating a forced stop. After sending the last frame, the MCE replies to the STOP command. **The rule is that NO reply from a MCE command should be sent to PCI in the middle of a data packet.**

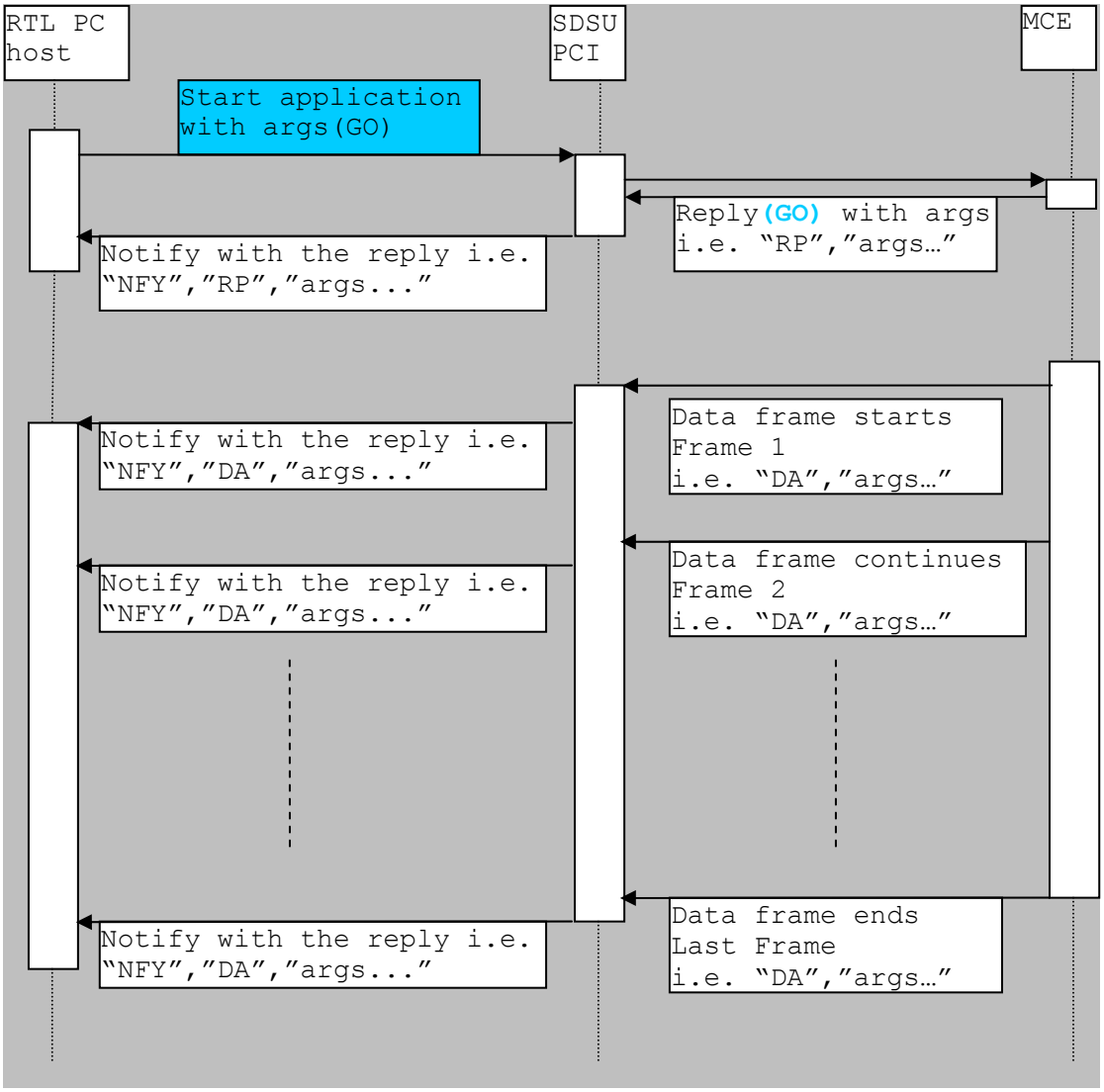
It is required that applications can be either downloaded onto the MCE by host software or loaded locally (here, it may just be to set a bit). All applications reside at a defined start address in on-board memory, the first word of which must contain an identifier that is unique to each version of application code. Due to the programming nature of FPGAs, new FPGA configuration files will be downloaded from the RTL PC into SRAMs on the MCE boards, and the FPGAs are programmed locally inside the MCE by a command issued from the RTL PC.

Figure2 demonstrates simply the command and reply/data sequence for the MCE, and Figure3 shows the command and reply sequence for the PCI. The detailed protocols are described in the next section.

a). Commands(WRITE BLOCK, READ BLOCK)→PCI→MCE→Execution→Reply→PCI→host

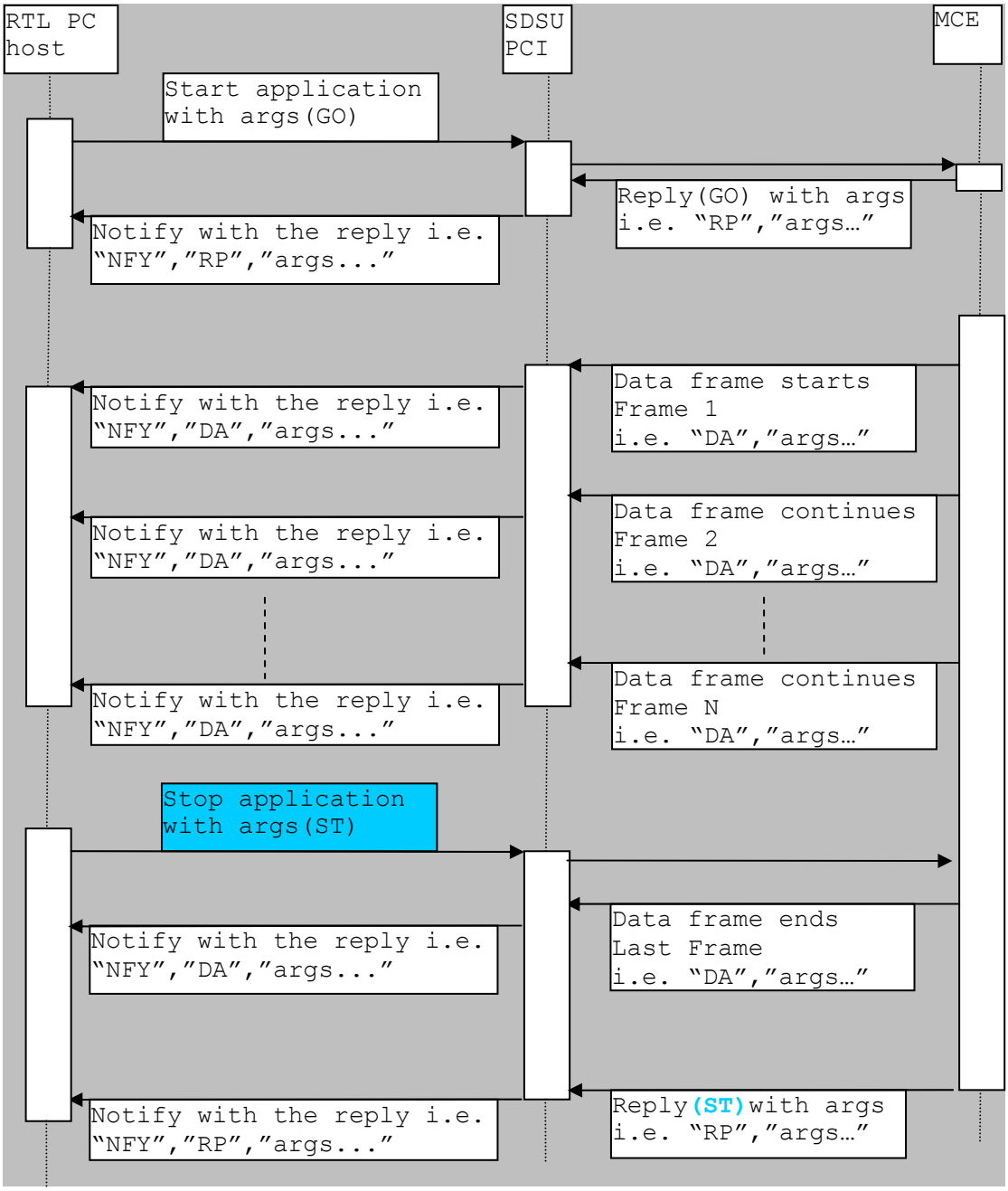


b). Command(START APPLICATION)→PCI→MCE→before Execution→Reply→PCI→host
some time later, MCE→frame data →PCI→host



c). Command(STOP APPLICATION)→PCI→MCE→ indicate the current frame as the last one→

Reply→PCI→host



d). Command(RESET)→PCI→MCE→before Execution→Reply→PCI→host

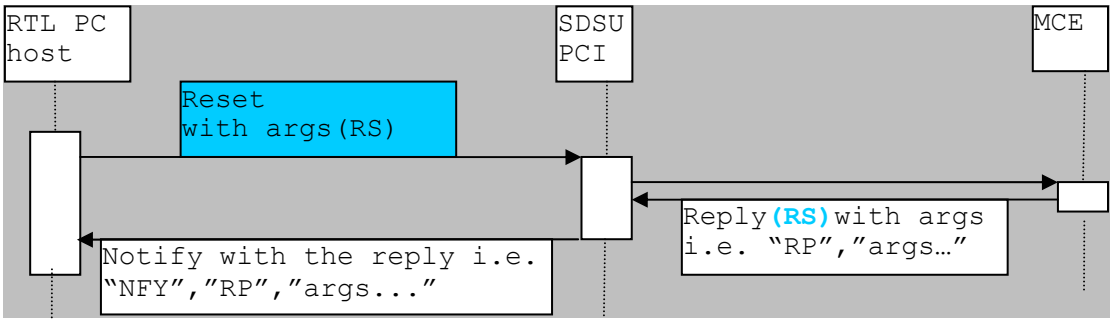


Figure 2. The command and reply sequence (MCE)

e). Commands (WRITE, READ, RESET, START APPLICATION, STOP APPLICATION,

SEND_PKT_TO_CONTROLLER, SEND_PKT_TO_HOST, HARDWARE_RESET_BAC/MCE)
→PCI→Execution→Reply→host

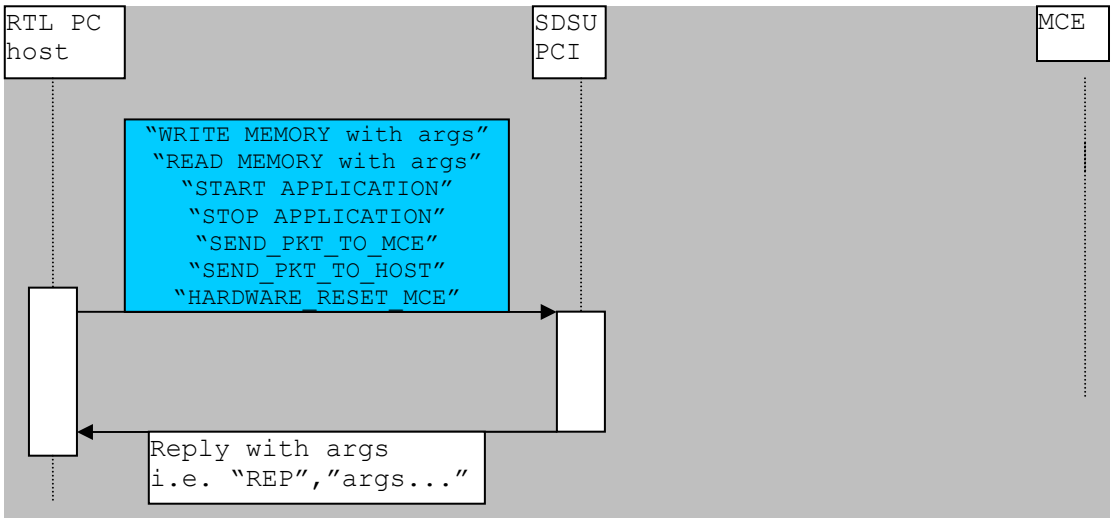


Figure 3. The command and reply sequence (PCI)

4.2 The commands to the MCE

All the commands are sent with a fixed number of bytes (256 for the time being) across the fibre to the MCE.

TABLE 1

Preamble (2 X 32 bits)	**Command (32bit)**	**Card Id**	**ParaId**	**Number of Data**	**DataBlock (232 bytes)**	**Checksum (32bit)**
A5A5A5A5 5A5A5A5A		**(higher 16 bit)**	**(lower 16 bit)**			
Word1/ word2	**Word3**	**Word4**		**Word5**	**Word6~word63**	**Word64**
The standard **commands** is:						
Write_block memory	**" WB" 0x20205742**	**CardId+ParaId**		**Value (range: 1..58)**	**Values**	**Value**
Start_ application	**" GO" 0x2020474F**	**CardId+ ParaId**		**Value (always = 1)**	**Word 6 = identifier Word 7..63 = 0x0**	**Value**
Stop_ application	**" ST" 0x20205354**	**CardId+ ParaId**		**Value (always = 1)**	**Word 6 = identifier Word 7..63 = 0x0**	**Value**
Reset	**" RS" 0x20205253**	**CardId+ ParaId**		**Value (always = 1)**	**Word 6 = identifier Word 7..63 = 0x0**	**Value**
Read_block memory	**" RB" 0x20205242**	**CardId+ ParaId**		**Value (N=number of reads mapped to um_data_o range: 1..58)**	**All 0x0**	**Value**

Each command is preceded with the preamble of $A5A5A5A5 5A5A5A5A and ended with the checksum. The checksum (word64, initial=0) use XOR method, i.e. word64=word3 XOR word64; word64=word4 XOR word64; word64=word5 XOR word64, and so on. The checksum will include all the 58 data. The word5 (**Number of data**) refers to how many valid 4 bytes (32 bits) in the DataBlock. The maximum value for the **Number of data** is 58. Zero values will be assigned to non-valid data.

4.3 The reply (MCE->PCI) for command and data for frame array readout

A). Replies for commands apart from the Read Block memory command

TABLE 2

2*(32 bit)	(32 bit)	(32 bit)	(32 bit)	2* (32 bit)		(32 bit)
0xA5A5A5A5 0x5A5A5A5A	" RP" 0x20205250	Packet size (always=4)	Reply	Reply arguments		Checksum
Header1/ Header2	Header3	header4	Word1	Word2	Word3	Word4

The packet size counts in Number of 32-bit (4 bytes) words. The Header1 and 2 are only seen by the PCI DSP, not the RTL software. The checksum (word4, initial=0) use XOR method, i.e. word4=word1 XOR word4; word4=word2 XOR word4, and word4=word3 XOR word4.

SUCCESS **TABLE 2-1**

	Word 1	Word2	word3	Word4
Start_application	"GOOK"(0x474F4F4B)	CardId+ ParaId	0x0	Checksum
Stop_application	"STOK"(0x53544F4B)	CardId+ ParaId	0x0	Checksum
Reset	"RSOK" (0x52534F4B)	CardId+ ParaId	0x0	Checksum
Write_Block memory	"WBOK"(0x57424F4B)	CardId+ ParaId	0x0	Checksum

FAILURE **TABLE 2-2**

	Word 1	Word2	word3	Word4
Start_application	"GOER"(0x474F4552)	CardId+ ParaId	Errno	Checksum
Stop_application	"STER"(0x53544552)	CardId+ ParaId	Errno	Checksum
Reset	"RSER"(0x52534552)	CardId+ ParaId	Errno	Checksum
Write_Block memory	"WBER"(0x57424552)	CardId+ ParaId	Errno	Checksum

B). The reply for "RB"

SUCCESS **TABLE 3**

2*(32 bit)	(32 bit)	(32 bit)	(32 bit)	(32 bit)	N*(32bit)	(32 bit)
0xA5A5A5A5 0x5A5A5A5A	" RP" (0x20205250)	Packet size (=N+3)	"RBOK" (0x52424F4B)	CardId+ ParaId	Read_Bloack data (N<= 58)	Checksum
Header1/ Header2	Header3	Header4	Word1	Word2	Word3~ Word(3-(N-1))	Word(3+N)

Where N = the requested number of reads from the Read_Block command.

The checksum (Word(3+N), initial=0) use XOR method, i.e. word(3+N)=word1 XOR word(3+N); word(3+N)=word2 XOR word(3+N), and so on, until word(3+N)=word(3+N-1) XOR word(3+N).

FAILURE **TABLE 4**

2*(32 bit)	(32 bit)	(32 bit)	(32 bit)	(32 bit)	(32bit)	(32 bit)
0xA5A5A5A5 0x5A5A5A5A	" RP" (0x20205250)	0x 00000004	"RBER" (0x52424552)	CardId+ ParaId	Error No	Checksum
Header1/ Header2	Header3	Header4	Word1	Word2	Word3	Word4

The checksum (Word4, initial=0) use XOR method, i.e. word4=word1 XOR word4; word4=word2 XOR word4, and word4=word3 XOR word4.

C).The data for frame array readout

TABLE 5

2*(32 bit)	(32 bit)	(32 bit)	N*(32 bit) Frame block			(32bit)
0xA5A5A5A5 0x5A5A5A5A	**“ DA” 0x20204441**	**Packet size**	**Frame status**	**…**	**…**	**Checksum**
Header1/ Header2	**Header3**	**Header4**	**Word1**	**Word2**	**…**	**WordM**

The packet size counts in Number of 32-bit (4 bytes) words.

The checksum (wordM, initial=0) use XOR method, i.e. WordM=word1 XOR wordM; wordM=word2 XOR wordM…. So on.

Each bit in the **frame status** represents a certain condition. For example, if bit_X is set, then the data will be in 16 bit format, otherwise, it will be in 32 bit format. The **frame status** will be read by the host. The possible frame data block will be defined in [3]. The frame status is defined in [4], also reproduced here

PCI Status/Data Format (1339 X 32-bits) : Status Block : Data Block

MSByte Byte 3								Byte 2	Byte 1	LSByte Byte 0	Word #	Word Description
										b a	Word 0	Frame Status Word 0
Frame Sequence Number											Word 1	Frame Status Word 1
Board Temperature								+15VA Supply	-15VA Supply	+7VA Supply	Word 2	Power Supply Card Status 0
-7VA Supply								+5VD Supply	+4.5VD Supply	+2.5VD Supply	Word 3	Power Supply Card Status 1
7	6	5	4	3	2	1	0	FPGA Temperature	Board Temperature	Cycles out of Sync	Word 4	Clock Card Status
7	6	5	4	3	2	1	0	FPGA Temperature	Board Temperature	Cycles out of Sync	Word 5	Readout Card 4 Status
7	6	5	4	3	2	1	0	FPGA Temperature	Board Temperature	Cycles out of Sync	Word 6	Readout Card 3 Status
7	6	5	4	3	2	1	0	FPGA Temperature	Board Temperature	Cycles out of Sync	Word 7	Readout Card 2 Status
7	6	5	4	3	2	1	0	FPGA Temperature	Board Temperature	Cycles out of Sync	Word 8	Readout Card 1 Status
7	6	5	4	3	2	1	0	FPGA Temperature	Board Temperature	Cycles out of Sync	Word 9	Bias Card 3 Status
7	6	5	4	3	2	1	0	FPGA Temperature	Board Temperature	Cycles out of Sync	Word 10	Bias Card 2 Status
7	6	5	4	3	2	1	0	FPGA Temperature	Board Temperature	Cycles out of Sync	Word 11	Bias Card 1 Status
7	6	5	4	3	2	1	0	FPGA Temperature	Board Temperature	Cycles out of Sync	Word 12	Address Card Status

4.4 The commands to the PCI board (four 24-bit words):

We will keep the commands and replies for the PCI board the same as those defined in Ultracam

The PCI shall always use DMA Burst mode to get the data from memory.

TABLE 6

	Command (24 bit)	Argument 1 (24 bit)	Argument 2 (24 bit)	Argument 3 (24 bit)
The format for each command is as follows:				
Write_memory	**WRM**	**Memory type**	**Address**	**value**
Read_memory	**RDM**	**Memory type**	**Address**	**dummy**
Start_application	**GOA**	**Identifier**	**Dummy**	**dummy**
Stop_application	**STP**	**Dummy**	**Dummy**	**dummy**
Reset	**RST**	**Dummy**	**Dummy**	**dummy**
Send_packet_to_MCE	**CON**	**Buffer addr hi**	**Buffer addr lo**	**dummy**
Send_packet_to_host	**HST**	**Buffer addr hi**	**Buffer addr lo**	**dummy**
Hardware_Reset_MCE	**RCO**	**Dummy**	**dummy**	**dummy**

Start_application – start an application on the PCI card's DSP, not the MCE board
Stop_application – stop an application on the PCI card's DSP, not the MCE board
send_packet_to_MCE – instruct the PCI card to forward a packet of data to the outgoing fibre link. This packet must be retrieved by the PCI interface, from host memory.
send_packet_to_host – instruct the PCI card to send a packet of data over the PCI bus, into memory accessible to the host software.
receive_packet – this "command" is never called by the host software. It is, in fact, a background process that is constantly looking for incoming data on the fibre link. When data arrives, it is streamed to a buffer in on-board memory. Once a full packet has been stored, the host is notified that a packet awaits delivery. The host will respond by causing execution of the send_packet_to_host command, shown above.
Hardware_Reset_MCE-instruct the PCI card to send a special character to the MCE, which is Hardware decoded to reset the whole MCE .

All commands are internal ones between the RTL PC software and the PCI DSP. These commands are never sent out to the fibre cable apart from the "Hardware_Reset_MCE".

4.5 The reply from the PCI card to the Host

The reply for every command comprises four 24-bit words:

TABLE 7

Word 1	word 2	Word 3	Word 4
Reply/Notify	**CMD copy**	**Reply word**	**Data word**

The first word is a flag, indicating that this packet of data is a command reply from PCI card or notification from the MCE board

4.5.1 The reply from the PCI card

Reply/Notify = **'REP'**

TABLE 8

Word 1	Word 2	Word 3	Word 4
'REP'	**CMD copy**	**Reply word**	**Data word**

Replies are of two types, indicating either the success or failure of a command:

TABLE 9

	Success reply		Failure reply	
	Word 3	Word 4	word 3	Word 4
Write_memory	**"ACK"**	**Dummy**	**"ERR"**	**Error no.**
Read_memory	**"ACK"**	**Value**	**"ERR"**	**Error no.**
Start_application	**"ACK"**	**Identifier**	**"ERR"**	**Error no.**
Stop_application	**"ACK"**	**Dummy**	**"ERR"**	**Error no.**
Reset	**"ACK"**	**Dummy**	**"ERR"**	**Error no.**
Send_packet_to_controller	**"ACK"**	**Dummy**	**"ERR"**	**Error no.**
Send_packet_to_host	**"ACK"**	**Dummy**	**"ERR"**	**Error no.**

4.5.2 The notification from the MCE board

Reply/Notify = **'NFY'**.

All communication data from the MCE arrives at the PCI as packets, the PCI DSP must perform an operation termed notify_host (**NFY**). The **NFY** command indicates the size of the packets (number of 32-bit words)

TABLE 10

Word 1	Word 2	word 3	Word 4
'NFY'	**Header3**	**header 4(31-16)**	**Header4(15-0)**

The first word is a flag indicating that this is not a command reply from the PCI but a notification of data packets received from the MCE. The next three words comprise the header3, header4's bits of 31 to 16 bit and header4's bits of 15-0 from TABLE 2, TABLE 3, TABLE 4 and TABLE 5, as received, thus informing the host of the mode and quantity of words in the packet.

4.6 The handling of corrupted frames by the PCI DSP

The DSP on the PCI card is responsible for checking the frame synchronisation. Once an out of synchronisation occurs, the DSP shall

1). discard the rest of the data until it sees the preamble again.

2). Count frames received from the MCE and record it in order for the high level software to examine where a frame missing is occurred.