SCUBA-2 PCI Card DSP Code Overview

1. Summary

This document describes the operation of the PCI card utilised to interface the SCUBA2 Multi-Channel Electronics (MCE) with the linux data acquisition PC. The PCI card has a 250MHz fibre optic chipset to communicate with the MCE, and it communicates with the host PC over the 33MHz (32 bit) PCI bus. This PCI card is 3rd party hardware purchased from "Astronomical Research Cameras" (ARC), SDSU. The heart of the PCI card is a Motorola DSP56301 processor, whose program is bootstrapped from an EEPROM during power up. The program has been heavily modified (from that provided by the manufacturer) to suit our requirements and interface with our PCI card driver. The SCUBA2 implementation of the PCI card is an evolution of the system designed for ULTRACAM (and utilised by WFCAM).

2. References

[1] "250hzPCI_schematic.pdf", Bob Leech, ARC, SDSU

[2] "250hzPCI_layout.pdf", Bob Leech, ARC, SDSU

[3] "250hzPCI_U19 pld.pdf", Bob Leech, ARC, SDSU

[4] "SCUBA2 data acquisition software overview – part two protocols-V3.0", SC2/SOF/S200/014, X. Gao, UK ATC

[5] "DSP56301 user's manual", DSP56301UM/AD, Revision 3, March 2001, section 6.2, Motorola Inc.

[6] "PCI_SCUBA_header.asm", David Atkinson, ATC

[7] "PCI_SCUBA_initialisation.asm" David Atkinson, ATC

[8] "PCI_SCUBA_main.asm", David Atkinson, ATC

3. Introduction

Communication with the MCE is achieved using a duplex 250MHz fibre link. All communications to and from the MCE are 32-bit with a little-endian byte order. However, it should be noted that the PCI card's DSP is a big-endian machine and 24-bit. Consequently, some on-chip byte manipulation is required when communicating to and from the MCE. It is the PCI card's responsibility to ensure all data sent down the fibre is in the correct little-endian byte order, and furthermore that valid data packets received on the fibre end up in host memory in correct little-endian byte order.

Data incoming over the fibre link are written to a FIFO for retrieval and examination by the on-board DSP (actually two 8-bit FIFOs which appear as one 16-bit fifo to the DSP). Communication with the host PC is via the PCI bus interface. The card can act as either a PCI bus master or slave, and a set of registers are provided to mediate interaction over the bus [5]. Whilst the host (PC) does have access to a subset of these registers for communication with the DSP, it does not have access to the on-board memory in which image data, for instance, will be temporarily stored by the PCI card. These data must therefore be transferred by the PCI card, acting as a bus master, into memory on the PC for use by the host software.

4. PCI Command Set (interrupt driven actions)

The PCI driver software employs the host-accessible register HCVR (Host Command Vector Register), to interrupt the DSP when it requires some operation to be performed. This mechanism is used to issue "PCI commands" and can be considered to be like a function call, where the arguments are passed through the register HTXR-DRXR (Host Transmit Data Register – DSP Receive Data Register).

The PCI card command set can be broken into two sub-categories. Firstly, there are the six commands which can be issued directly from 'user space', and secondly there are two commands embedded within the PCI card driver software to instruct the PCI card how to handle packets for and from the MCE.

4.1 User Space Commands

- **Read_memory** ('RDM' command)– this command is executed to instruct the PCI card to return a single value from DSP memory. The DSP has three types of memory: P memory (where the executable programme is stored); X memory (where parameters are stored); and Y memory (where data from the MCE is temporarily stored before being written to host memory).

- **Write_memory** ('WDM' command)– this command is executed to instruct the PCI card to write a value to its on board memory. Typically, X memory would only ever be written to (to manipulate programme parameters), however applications can be downloaded to P memory.

- **Reset_pci** ('RST' command) – this command forces a software reset on the PCI card.

- **Reset_controller** ('RCO' command) – this command instructs the PCI card to send a 'special character' byte down the fibre to the MCE. On receipt of this 'special character' the MCE will reset its firmware.

- **Start_Application** ('GOA' command) – this command instructs the PCI card to start an application residing in the DSPs 'application space'. Applications are downloaded to the PCI card as and when they are needed to enable temporary specialised behaviour. Such applications are useful during the development stage and for diagnostics.

- **STOP_Application** ('STP' command) – this command instructs the PCI card to stop any application currently running in application space. This will result in the PCI card reverting to normal operation.

4.2 Driver Commands

These two commands are used by the driver when instructing the PCI card about communications with the MCE.

- **send_packet_to_controller** ('CON' command)– this command informs the PCI card that there is a packet to be forwarded to the outgoing fibre link. The said packet is retrieved from host memory (PCI card acting as bus master), then sent down the fibre to the MCE. For the SCUBA2 project the PCI card must ensure that "IDLE" characters exist in between each word of a command sent to the MCE. "IDLE" characters are inserted automatically by the (HotLINK) fibre chipset whenever data is not being transmitted. Consequently, the PCI card needs to ensure that there is at least a fibre transmitter clock cycle (40ns) between the transmission of adjacent words. This is required because the MCE uses the "IDLE" characters for word synchronisation.

- **send_packet_to_host** ('HST' command) – this command instructs the PCI card to write an MCE packet to host memory. The 'HST' command is only ever issued after the PCI card has notified the host that a valid packet has arrived on the fibre.

Flow diagrams for the Interrupt Service Routines (ISR) of these two special commands are provided in Appendix A1 and A2 respectively.

4.3 Command Arguments

The PCI command arguments, which are passed through the register HTXR-DRXR, are summarised in Table 4.1.

Command Name	**Arguments Passed through HTXR-DRXR (24-bit)**			
Write_memory	'WRM'	Memory Type	Address	Value
Read_memory	'RDM'	Memory Type	Adress	Dummy
Start_application	'GOA'	I.D.	Dummy	Dummy
Stop_application	'STP'	Dummy	Dummy	Dummy
Reset_pci	'RST'	Dummy	Dummy	Dummy
Reset_mce	'RCO'	Dummy	Dummy	Dummy
Send_packet_to_controller	'CON'	Buffer Addr Hi	Buffer Addr Lo	Go Flag
Send_packet_to_host	'HST'	Buffer Addr Hi	Buffer Addr Lo	Dummy

Table 4.1: *PCI Commands*

4.4 PCI Command Replies – 'Reply Messages'

A ***reply message*** must be generated for every command. The message contains four 24-bit words and its structure is revealed in Table 4.2. The ***reply message*** is communicated to the host by passing the four words to 24-bit register DTXS (DSP Slave Transmit Data Register), then interrupting the host over the PCI bus (INTA). The host can access the reply message through register HRXS (Host Slave Receive Data Register) [5] .

word 1 (24-bit)	word 2 (24-bit)	word 3 (24-bit)	word 4 (24-bit)
Reply Word 'REP'	**Command Echo i.e. 'RDM'**	**Reply status 'ACK' or 'ERR'**	**Data Word or Error Code**

Table 4.2: *PCI Command reply message*

4.5 PCI Command Summary

- All PCI commands are interrupt driven, and consequently each command has an associated interrupt service routine (ISR).
- Commands are instigated by the host writing to vector register HCVR.
- Command arguments (of which there are four) are passed for through register HTXR-DRXR
- Replies to commands (***reply messages***) are passed to the host through register DTXS-HRXS, signalled with the PCI bus interrupt available to the PCI card (INTA).

The next section goes on to describe the functionality of the main body of the DSP code, that is the code running while it is not servicing interrupt driven commands.

5. Main Body of PCI Code: MCE Packet Handling

The main body of the DSP code services the incoming fibre link, notifies the host when a packet has arrived, and once instructed (via the 'HST' command) writes the packet to host memory. A flow diagram for the main body of the code is provided in Appendix A3. It should be noted that during initialisation the DSP sets a bit called PACKET_CHOKE in a STATUS word. While this bit is set the DSP will discard any data arriving on the fibre. The PACKET_CHOKE bit is cleared when the first packet is sent to the MCE. That is the incoming fibre channel is 'opened' once communications with the MCE are instigated by the host (via the "CON" command).

Data arriving up the fibre is written to one of two 1024 deep 8-bit FIFOs, U14 and U18[1]. The DSP sees these as one 1024 deep 16-bit FIFO with a **not_empty** and **half_full** flag which it can pole.

All communication from the MCE must conform to the SCUBA 2 protocol [4]. All packets from the MCE consist of a packet header, a packet body, and a checksum. The packet header format is revealed in Table 5.1. As shown it begins with two preamble words. Any data arriving up the fibre without the correct preamble sequence is discarded. There are two types of permitted packets, namely reply packets and image data packets. The packet type is defined in word 3 of the header. An invalid packet type will result in the entire packet being discarded, that is the host is not notified of the packet and the DSP continually clears the receive FIFO until it finds the next valid preamble sequence. The 4th and final word of the packet header indicates the size of the packet (in 32-bit words).

word 1 (32-bit)	word 2 (32-bit)	word 3 (32-bit)	word 4 (32-bit)
Preamble 1 (0xA5A5A5A5)	**Preamble 2 (0x5A5A5A5A)**	**Packet Type (reply or image data)**	**Packet Size (no. of 32-bit words)**

Table 5.1: *MCE Packet Header*

Once a correct preamble sequence followed by a valid packet type has arrived on the fibre, the packet size (header word 4) is read from the FIFO and temporarily stored in DSP X memory (as two 16-bit words). A ***notify message*** is then constructed to inform the host that there is a packet arriving from the MCE to be written to host memory. The structure of the ***notify message*** is shown in Table 5.2. This four word message is written to the 24-bit register DTXS. The host is then interrupted over the PCI bus (INTA) to inform it that there is a message to read (accessed through HRXS on the host side). Note that the ***notify message*** is communicated to the host in the same fashion as the PCI command ***reply message***.

word 1 (24-bit)	word 2 (24-bit)	word 3 (24-bit)	word 4 (24-bit)
Notify Word 'NFY'	**Packet Type (reply or data)**	**Packet Size (high 16-bits)**	**Packet Size (low 16-bits)**

Table 5.2: *Packet Notify Message*

Once the ***notify message*** has been sent to the host, the DSP immediately begins to buffer the entire fibre packet to the DSP's "Y" memory. The packet is stored in contiguous memory locations starting from Y:$000000. Note that memory locations Y:$000000 to Y:$0007FF are on-chip while locations Y:$000800 onwards are located on external RAM.

Typically the host PC will interrupt the DSP with a **send_packet_to_host** ("HST") command during this packet buffering. The HST command informs the DSP that the host is ready to receive the packet and moreover provides a bus address. Alternatively, the host can issue a **fatal error fast interrupt**, which informs the DSP that it should abort the packet and re-initialise. The fatal error fast interrupt is described in more detail in Section 6.

Once the PCI card has completed buffering the fibre packet and has been issued with an HST command, it begins to write the packet to host memory as bus master. It does this by splitting the packet into a number of 64-word blocks (256 bytes: maximum burst size) each of which are sequentially DMAed to host memory as PCI write memory bursts. Finally any left over words are written as one final PCI burst.

Once the entire packet has been written to the host memory, the PCI card replies to the HST command (the only command whose reply isn't issued from its ISR) and goes back to checking the fibre for the next incoming packet.

6. Fast Interrupts

As discussed in Section 2.2, the HCVR provides a mechanism for the host processor to interrupt the DSP core to execute a vectored interrupt. Typically this mechanism is utilised to execute PCI command ISRs. However, it is also used to execute **fast interrupts**. Here the DSP core is interrupted and the core temporarily jumps to execute one line of code written in a special vectored location. The SCUBA2 bootcode employs three such fast interrupts. The first two are utilised for **message handshaking**, and the third allows the host to communicate a **fatal error** to the DSP.

6.1 Message Handshaking

There are two types of 'Message' that the DSP can send to the host: ***reply message*** (a reply to a PCI command) and ***notify message*** (to inform the host that a packet has arrived from the MCE)**.** As discussed in previous sections both of these messages comprise four 24-bit words, and are communicated to the host by passing them to register DTXS, then interrupting the host over the PCI bus (INTA). The DSP asserts this bus interrupt by setting a bit in the DSP Control Register (DCTR) [5]. This bit stays high until the host issues a ***fast interrupt*** to force the DSP to clear it. Typically the host will do this as soon as it detects the bus interrupt. Thereafter, once the host has finished processing the message it issues a second ***fast interrupt***, which clears a flag (INTA_FLAG) in the STATUS word in DSP memory. Clearing this flag informs the DSP that the host has finished processing the current message and is ready to receiver any subsequent messages.

Messages are communicated to the host using the "PCI_MESSAGE_TO_HOST" routine. A flow diagram for this routine is provided in Appendix A4.

6.2 Fatal Errors

The third and final ***fast interrupt*** employed by this system is utilised to inform the DSP that there has been a serious error and that it should stop what it is doing and re-initialise. The ***fatal error fast interrupt*** sets a flag (FATAL_ERROR) in the STATUS word in DSP memory. The DSP checks this at various stages of the main packet handling code (see flow diagram in Appendix A3), and re-initialises if set. In particular the flag is periodically checked while buffering a fibre packet to Y memory, and immediately before the HST reply is issued.

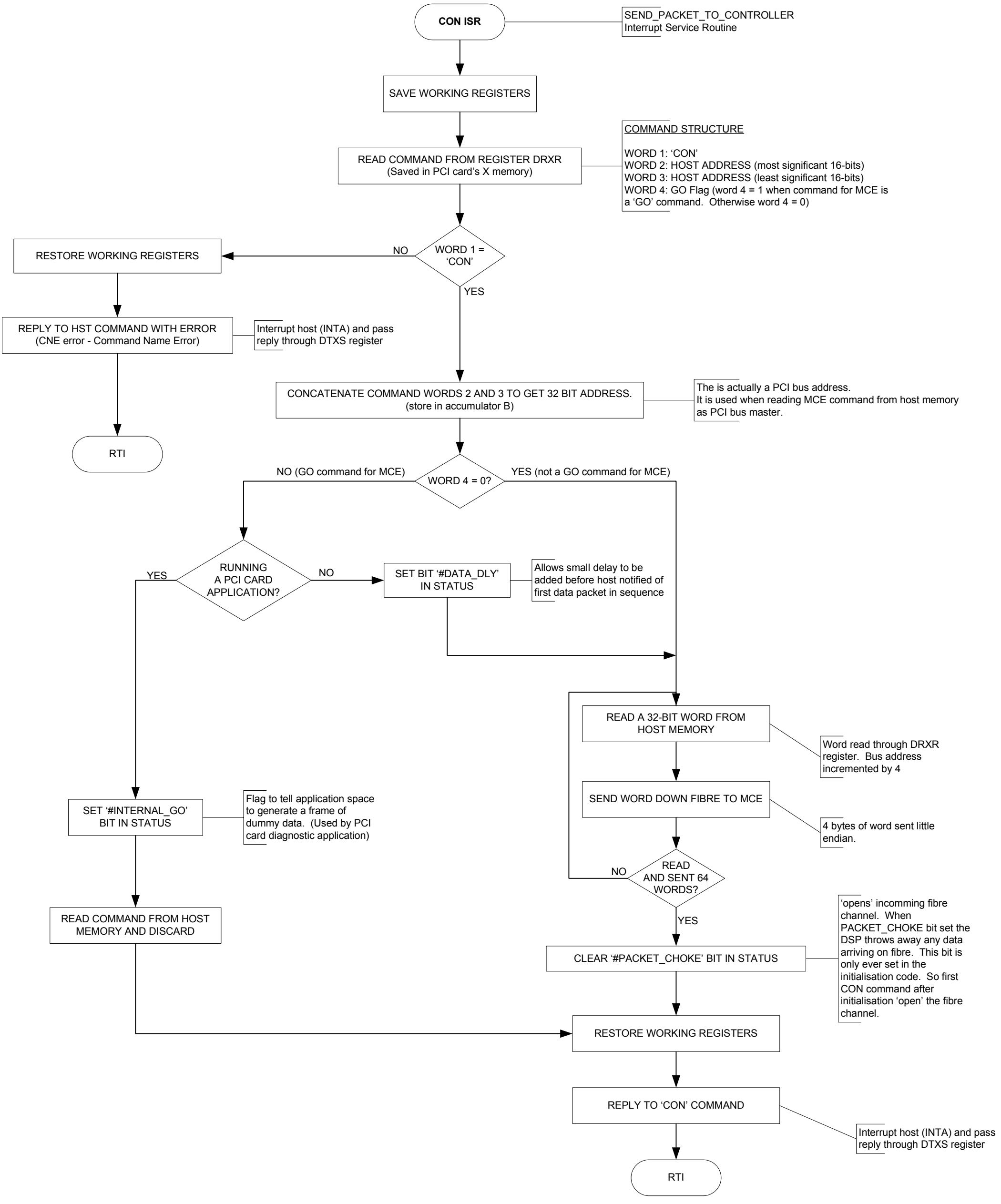
Once an HST command has been issued (by the host) the DSP has a finite period of time to DMA the current packet and reply to the command (via ***reply message)*** otherwise the host software forces the ***fatal error fast interrupt*** (due to an HST command Timeout). Typically, this would occur if a partial frame was sent by the MCE, and the DSP was waiting on data to arrive on the fibre. On receiving such a fatal error the DSP clears out any data from the fibre FIFO and stores it on off-chip Y memory (from address Y:$001000). The number of 16-bit words recovered from the FIFO and written to Y memory is indicated in by the parameter X:NUM_DUMPED (X:$000007). The number of 32-bit words successfully written to the host PC prior to the HST timeout is indicated by the parameter X:WORD_COUNT (X:$000006), which always reveals the number of words written across the bus in the current/last packet.

7. Closing Remarks

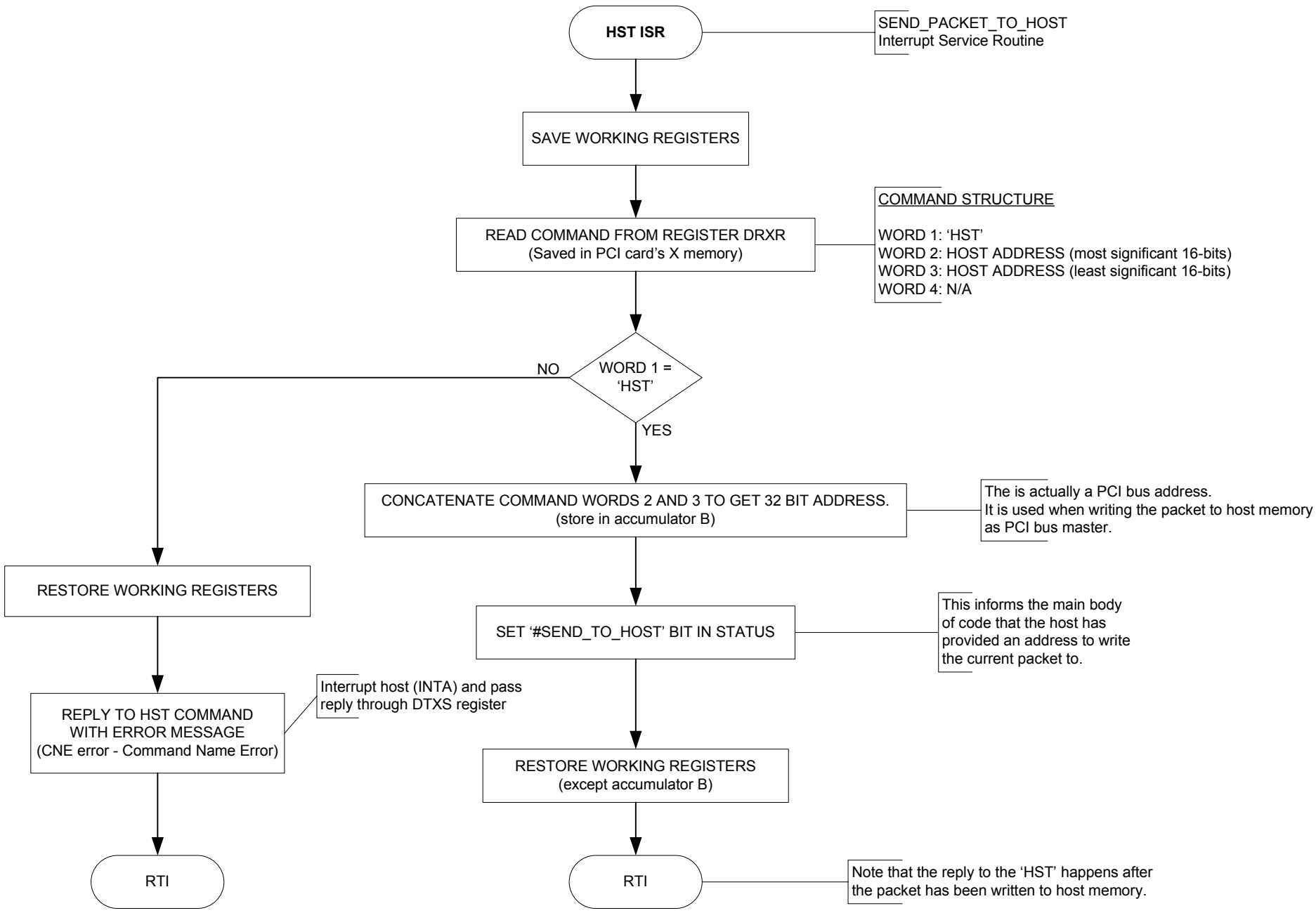
The PCI card has two main functions. Firstly, to respond and reply to any commands issued by the host PC, and secondly to service the incoming fibre link and notify the host when a valid packet has arrived from the MCE. It replies to commands and notifies the host of packets using the same mechanism - by writing a four word message to the DSP slave transmitter register (DTXS) and interrupting the host over the PCI bus (INTA).

After notifying the host that there is a valid packet from the MCE, the host will instruct the PCI card how to proceed. Typically, the host will issue an "HST" command to instruct the DSP to write the packet to a given address. The DSP then splits the packet into manageable blocks and transfers it over the PCI bus directly into host memory as bus master.

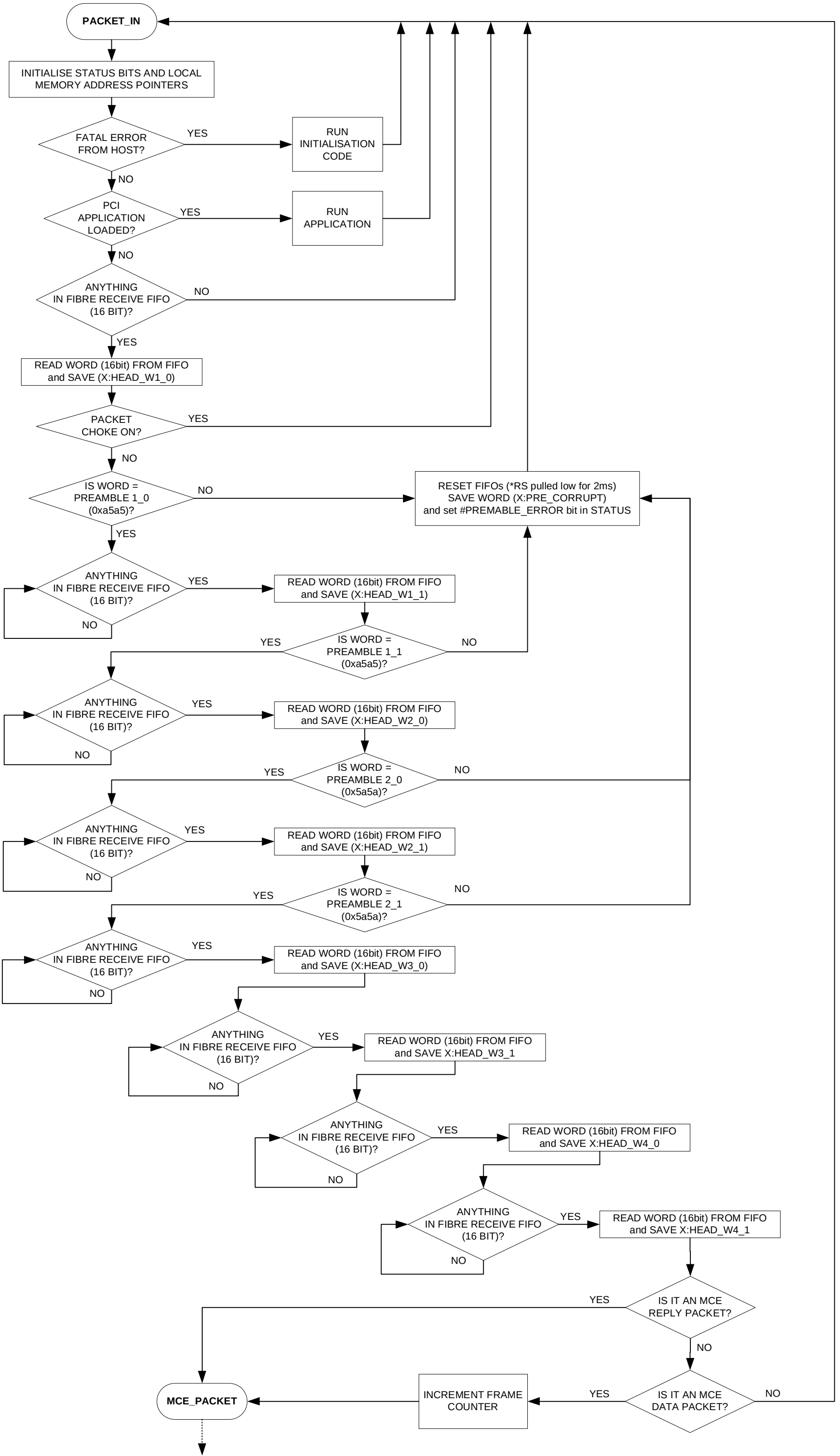
Appendix A1: *Flow Diagram for Send_Packet_to_Controller (CON) Command*

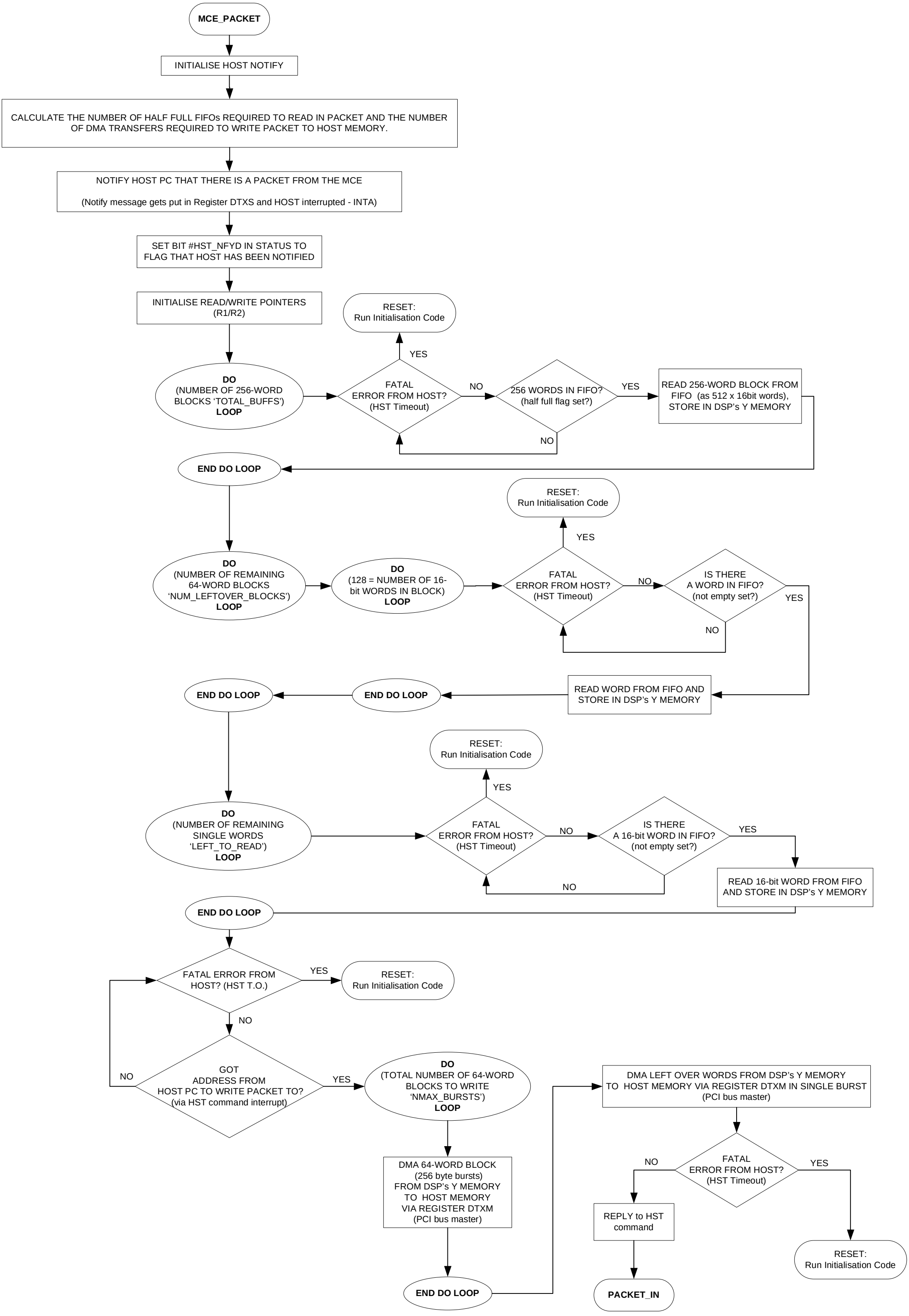


Appendix A2: *Flow Diagram for Send_Packet_to_Host (HST) Command*



Appendix A3: *Flow Diagram for Main Body of DSP Code*




MCE_PACKET
INITIALISE HOST NOTIFY
CALCULATE THE NUMBER OF HALF FULL FIFOs REQUIRED TO READ IN PACKET AND THE NUMBER OF DMA TRANSFERS REQUIRED TO WRITE PACKET TO HOST MEMORY.
NOTIFY HOST PC THAT THERE IS A PACKET FROM THE MCE
(Notify message gets put in Register DTXS and HOST interrupted - INTA)
SET BIT #HST_NFYD IN STATUS TO FLAG THAT HOST HAS BEEN NOTIFIED
INITIALISE READ/WRITE POINTERS (R1/R2)
DO (NUMBER OF 256-WORD BLOCKS 'TOTAL_BUFFS') LOOP
FATAL ERROR FROM HOST? (HST Timeout)
YES
RESET: Run Initialisation Code
NO
256 WORDS IN FIFO? (half full flag set?)
YES
NO
READ 256-WORD BLOCK FROM FIFO (as 512 x 16bit words), STORE IN DSP's Y MEMORY
END DO LOOP
DO (NUMBER OF REMAINING 64-WORD BLOCKS 'NUM_LEFTOVER_BLOCKS') LOOP
DO (128 = NUMBER OF 16-bit WORDS IN BLOCK) LOOP
FATAL ERROR FROM HOST? (HST Timeout)
YES
RESET: Run Initialisation Code
NO
IS THERE A WORD IN FIFO? (not empty set?)
YES
NO
READ WORD FROM FIFO AND STORE IN DSP's Y MEMORY
END DO LOOP
END DO LOOP
DO (NUMBER OF REMAINING SINGLE WORDS 'LEFT_TO_READ') LOOP
FATAL ERROR FROM HOST? (HST Timeout)
YES
RESET: Run Initialisation Code
NO
IS THERE A 16-bit WORD IN FIFO? (not empty set?)
YES
NO
READ 16-bit WORD FROM FIFO AND STORE IN DSP's Y MEMORY
END DO LOOP
FATAL ERROR FROM HOST? (HST T.O.)
YES
RESET: Run Initialisation Code
NO
GOT ADDRESS FROM HOST PC TO WRITE PACKET TO? (via HST command interrupt)
NO
YES
DO (TOTAL NUMBER OF 64-WORD BLOCKS TO WRITE 'NMAX_BURSTS') LOOP
DMA 64-WORD BLOCK (256 byte bursts) FROM DSP's Y MEMORY TO HOST MEMORY VIA REGISTER DTXM (PCI bus master)
END DO LOOP
DMA LEFT OVER WORDS FROM DSP's Y MEMORY TO HOST MEMORY VIA REGISTER DTXM IN SINGLE BURST (PCI bus master)
FATAL ERROR FROM HOST? (HST Timeout)
NO
YES
REPLY to HST command
RESET: Run Initialisation Code
PACKET_IN

Appendix A4: *Flow Diagram for PCI_MESSAGE_TO_HOST routine*