TOF CANbus High Level Protocol (HLP) Version 3l

0.0 Introduction

The TOF CANbus network will be divided into several sub-networks.
From the PC, there will be 4 top-level networks, each connecting 1 THUB and 30 or 31 TCPUs. Each TCPU will have a tray-level network connecting the 8 TDIGs on a tray and 1 TCPU. So each network will have up to 33 nodes connected to it.
Based on this assumption, the HLP is structured as described in this document. Messages and commands are the same for both TDIG and TCPU with the exception that TCPU will report as "invalid" messages which target functions not available on TCPU boards.

0.1.0 CANbus Packet ID - Standard

Each CANbus packet contains an identifier field (ID) and up to 8 bytes of payload. For messages within one level of the networks we restrict the ID to so-called "standard" IDs, which contain 11 bits.
The message ID bits are separated into a "Node ID" field and a "Command" field as follows:

MsgID[10:4] = N[6:0] = NodeID
MsgID[3:0] = C[3:0] = Command

ID[10]	ID[9]	ID[8]	ID[7]	ID[6]	ID[5]	ID[4]	ID[3]	ID[2]	ID[1]	ID[0]
N[6]	N[5]	N[4]	N[3]	N[2]	N[1]	N[0]	C[3]	C[2]	C[1]	C[0]

TDIG boards will respond only to messages with standard addressing.

0.1.1 CANbus Packet ID - Extended

For messages that have to traverse networks (i.e. from top level network to tray-level network or the other way around) we will use so called "extended" IDs, which contain 29 bits: 11 bits equivalent to the standard ID bits, and 18 bits of the extended ID bits.

MsgID[28:22] = N[6:0] = NodeID (standard)
MsgID[21:18] = C[3:0] = Command (standard)
MsgID[17:7] = "zero" = not used, zero filled
MsgID[6:0] = E[6:0] = Extended ID

ID[28]	ID[27]	ID[26]	ID[25]	ID[24]	ID[23]	ID[22]	ID[21]	ID[20]	ID[19]	ID[18]
N[6]	N[5]	N[4]	N[3]	N[2]	N[1]	N[0]	C[3]	C[2]	C[1]	C[0]

ID[17]	ID[16]	ID[15]	ID[14]	ID[13]	ID[12]	ID[11]	ID[10]	ID[9]	ID[8]	ID[7]
0	0	0	0	0	0	0	0	0	0	0

ID[6]	ID[5]	ID[4]	ID[3]	ID[2]	ID[1]	ID[0]
E[6]	E[5]	E[4]	E[3]	E[2]	E[1]	E[0]

The standard ID bits are defined as described above (containing the nodeID of the originating node, or the nodeID of the ultimate destination). The Extended ID will be used to encode the NodeID of the TCPU which will forward the message from the top level network (CANBus #2) to the tray level network (CANBus #1) or from the tray level network to the top level network. The nodeID will be right adjusted (i.e. E[0] = NodeID[0]). A TCPU receiving a CAN message with an extended ID will therefore always strip the extended ID bits from the message ID and send the otherwise unmodified CAN message with the standard message ID on its other network to TDIG. TDIG boards will respond only to messages with standard addressing. TCPU boards must respond to appropriately addressed standard messages received on CAN2 (system level network) and may respond to appropriately addressed standard messages received on CAN1 (tray level network).

0.2 Node ID

The nodeIDs in each sub-network are uniquely assigned to each CAN processor. NodeID "0" will be forbidden; nodeID "127" (0x7F) is the broadcast ID (i.e. all nodes in a sub-network are supposed to respond to this nodeID).

0.2.1 Node ID - TDIG

For TDIG boards, the NodeID N[6:0] will be [0010bbb] where [bbb] represents the "board position ID" set by the board's position in the tray (board position switch). TDIG NodeIDs will therefore be (16+switch_setting). TDIG boards will respond only to Standard node addresses (because they are on only one CANbus).

0.2.2 Node ID - TCPU

For TCPU boards the NodeID N[6:0] on both the tray level network (CANBus #1) and the top-level network (CANBus #2) will be [01bbbbb] where [bbbbb] represents the "board position ID" set by the board's position in the system (board position switch). TCPU NodeIDs will therefore be (32+switch_setting).

0.2.3 Node ID - THUB

The THUB will have nodeID 64 [100 0000]

0.3 Byte Ordering

The bytes of a multi-byte word within a CAN message will be sent with the least significant byte contained in Byte 0 of the CAN message, followed by Byte 1, Byte 2, and Byte 3 of each 4-byte word. Only as many bytes as necessary for the word size will be sent (i.e. no byte padding).

1. Command Codes

The following command codes are currently defined. The payload of each CAN packet further qualifies the command as shown later in this document. Note that the LSbit of the Command Code identifies "direction" of data movement "0"= From Host to TCPU to TDIG ("down the tree"), 1= From TDIG to TCPU to Host ("up the tree"). Undefined Command Codes will not be processed.

Command Code	Meaning
1	DATA
2	"WRITE" command
3	Write Response (optional)
4	"READ" command
5	Read Response
7	"STATUS" or "ALERT"

Command Codes 0, 6, and 8 through 15 are undefined "reserved". Note that these Command Codes are distinct from the subcommand codes which qualify the exact function performed.

1.1 "DATA" command (Code=1)

This is defined just like in Justin's HLP, i.e. 4 or 8 bytes of payload with TDC/Trigger data. The NodeID will be the node ID of the originating board (*i.e.* Board Position). The NodeID may be from either TDIG or TCPU.

1.2 "WRITE" command (Code=2)

This command causes data to be written to the node, or causes some action to happen.
The nodeID field in the CAN message header is the nodeID of the addressed node, i.e. the node being written to. What specifically should happen is determined by the payload:

Write (2)	**Write Command**
Byte 0:	"Address" of this "WRITE". The address can be interpreted as a "sub" command as shown below. It follows, that a total of 256 different "actions" can be defined this way. The address can also be an actual memory start address to write to. The number of bytes to "write" follows from the packet length.
Bytes 1 – 7:	Payload of the sub command, data to write. This is further qualified for each "Address" below, if this "Address" is not an actual memory to write to, but rather an action to take.

All "Write" commands result in a response from the addressed node, the "Write Response" command will be used as a feedback to the sending node.

Write Response (3)	**Length = 2**
Byte 0:	"Address" or "Sub-Command" of this "WRITE" (copied from the received message.
Byte 1:	Status resulting from this write or sub-command. A value=0 indicates successful completion. Non-zero values indicate (various) errors (to be determined; see *Table x*)

1.2.1 "Simple" Write commands (Code=2)

This form of Write command transmits commands or data where the entire data field requires 7 bytes or less of additional data (e.g. a threshold setting value).

Write (2)	**Write Command (Lengths=1 to 8)**
Byte 0:	"Address" of this "WRITE". The address can be interpreted as a "sub" command as shown below. It follows, that a total of 256 different "actions" can be defined this way. The address can also be an actual memory start address to write to. The number of bytes to "write" follows from the packet length.
Bytes 1 – 7:	Payload of the sub command, data to write. This is further qualified for each "Address" below, if this "Address" is not an actual memory to write to, but rather an action to take.

All "Write" commands result in a response from the addressed node, the "Write Response" command will be used as a feedback to the sending node.

Write Response (3)	**Length = 2**
Byte 0:	"Address" or "Sub-Command" of this "WRITE" (copied from the received message.
Byte 1:	Status resulting from this write or sub-command. A value=0 indicates successful completion. Non-zero values indicate (various) errors (to be determined; see *Table x*)

1.2.2 Large-Block Write Sequences

Transmitting large blocks of data (*e.g.* HPTDC setup information, new FPGA code destined for EEPROM #2, or new MCU code) requires more data than can be contained in a single CAN message. Therefore a sequence of sub-commands is necessary. The sequence will consist of "Block-Start", "Block-Data", "Block-End", and "Block-Disposition". The previously available "Checksum" command is not implemented. Its function is supplied via the Block-End reply containing the buffer checksum. The checksum is a simple 32-bit summing of data bytes received.

1.2.2.1 Block-Start

This "Write" sub-command will initialize the download sequence. The buffer will be cleared to zero, the current-location pointer and checksum will be initialized. "Block-Start" is required before any large-block data is transmitted. Receipt of a Block-Start sub-command will restart the Large-Block sequence.

Write	**Block-Start sub-command (Lengths= 1 to 8)**
Byte 0:	0001 0000 (0x10)
Bytes 1 – 7:	Byte 0 – Byte 6 of download data. LSByte first. Bytes if present are stored sequentially into download buffer and also update the running checksum. These bytes are optional; the only required payload byte is Byte 0 specifying the Block-Start subcommand.

Response: "Write Response" with status.

Write Response (3)	**Length = 2**
Byte 0:	"Address" or "Sub-Command" of this "WRITE" (copied from the received message.
Byte 1:	Status resulting from this write or sub-command. A value=0 indicates successful completion. Non-zero values indicate (various) errors (to be determined; see *Table x*)

1.2.2.2 Block-Data

This "Write" sub-command transfers the next bytes of actual data. Payload data is copied to the buffer and the pointer and checksum are updated. The number of bytes transferred is determined from the packet length. Receipt of a "Data Transmit" without a preceding "Block-Start" is an error. "Data Transmit" which results in overfilling the buffer space is an error and will result in an Overrun error status response; in this situation the buffer will be filled to its maximum extent.

Write (2)	**Block-Data sub-command (Lengths=1 to 8)**
Byte 0:	0010 0000 (0x20)
Bytes 1 – 7:	Byte x – Byte x+6 of download data. Bytes are stored sequentially into download buffer and also update the running checksum. The number of bytes downloaded from each payload is determined by the CANbus message length and may be less than the maximum 7 bytes.

Response: "Write Response" with status.

Write Response (3)	**Length = 2**
Byte 0:	"Address" or "Sub-Command" of this "WRITE" (copied from the received message.
Byte 1:	Status resulting from this write or sub-command. A value=0 indicates successful completion. Non-zero values indicate (various) errors (to be determined; see *Table x*)

1.2.2.3 Block-End

This marks the end of data transfer. "Block-End" without a preceding "Block-Start" is an error.

Write (2)	**Block-End sub-command (Length=1)**
Byte 0:	0011 0000 (0x30)

Response: 8-byte "Write Response" with status, number of bytes received, and checksum of bytes received.

Write Response (3)	**Length = 8**
Byte 0:	"Address" or "Sub-Command" of this "WRITE" (copied from the received message.
Byte 1:	Status resulting from this write or sub-command. A value=0 indicates successful completion. Non-zero values indicate (various) errors (to be determined; see *Table x*)
Bytes 2-3:	Total number of bytes in block transferred (LSByte first)
Bytes 4-7:	Checksum of data bytes transferred (does not include protocol bytes)

1.2.2.4 Block-Disposition (Block-Target)

This tells the MCU what to do with the buffer of data *(e.g.* use it to configure an HPTDC). "Block-Disposition" without "Block-End" is an error. The Block-Disposition message, depending on the target, may contain additional address-type information (to be determined, *e.g.* EEPROM memory address).

Write (2)	**Block-Disposition sub-command (Lengths=1 or 6)**

Byte 0:	`0100 tttt`
	`(0x4t)`
	tttt = target for disposition of the buffered data
	0x40 = All 3 HPTDCs chip Configuration (volatile).
	0x41 = HPTDC1 chip Configuration (volatile).
	0x42 = HPTDC2 chip Configuration (volatile).
	0x43 = HPTDC3 chip Configuration (volatile).
	0x44 = All 3 HPTDCs chip Configuration Memories (FLASH).
	0x45 = HPTDC1 chip Configuration Memory (FLASH).
	0x46 = HPTDC2 chip Configuration Memory (FLASH).
	0x47 = HPTDC3 chip Configuration Memory (FLASH).
	0x48 = All 3 HPTDCs chip Control Memory (FLASH).
	0x49 = HPTDC1 chip Control Memory (FLASH).
	0x4A = HPTDC2 chip Control Memory (FLASH).
	0x4B = HPTDC3 chip Control Memory (FLASH).
	0x4C = MCU Program Memory (FLASH).
	0x4E = FPGA Configuration EEPROM #2 (EEPROM).

Bytes 1 – 5:	For Target = HPTDC or HPTDC Memories these bytes are not used and not present. For Target = EEPROM #2 or MCU, these bytes will contain 1-4 memory-start address; 5 erase flag (1=erase).

Response: "Write Response" with status. For disposition to EEPROM #2 with "erase" the disposition operation may take up to (3 seconds). Block-Disposition to HPTDC or HPTDC Memory is not valid when directed to a TCPU board.

Write Response (3)	**Length = 2**
Byte 0:	"Address" or "Sub-Command" of this "WRITE" (copied from the received message.
Byte 1:	Status resulting from this write or sub-command. A value=0 indicates successful completion. Non-zero values indicate (various) errors (to be determined; see *Table x*)

1.2.2.5 Block-Checksum

This feature will not be implemented. The function code is reserved. This sub-command is optional and may be used multiple times during large-block transfers. The payload of the "Block-Checksum" command will be compared against the computed current-value of the buffer checksum. After comparison, the checksum will be reset. If "Block-Checksum" is issued after "Block-End", then the checksum computed over the actual received data length will be re-computed and compared. The function of this command has been superceeded by the Block-End Reply containing bytecount and checksum.

Byte 0:	0101 0000 **Block-Checksum sub-command**
Bytes 1 – tbd:	The checksum to be compared against the checksum computed from the downloaded data. Details to be determined.

Response: "Write Response" with status.

1.3 "WRITE" Response (Code=3)

The nodeID is the nodeID of the responding node. Only the node that initiated the "WRITE" should listen for this message (e.g. with a state machine change to "WAIT FOR WRITE RESPONSE").

Write Response (3)	**Length = 2**
Byte 0:	"Address" or "Sub-Command" of this "WRITE" (copied from the received message.
Byte 1:	Status resulting from this write or sub-command. A value=0 indicates successful completion. Non-zero values indicate (various) errors (to be determined; see *Table x*)

1.4 "READ" command (Code=4)

This command causes data to be sent back to the requesting node. The nodeID field in the CAN message header is the nodeID of the addressed node, i.e. the node being read from. What data specifically is being sent back is determined by the payload:

Read (4)	**Lengths= 1, 2, 3, 4, or 5**
Byte 0:	"Address" of this "READ". The address can be interpreted as a "sub" command as shown below. "Read" to an invalid or unimplemented function results in a 1 byte return payload echoing the "address" with no data.

1.5 "READ" Response (Code=5)

The response depends on the specific address being read (see below). This should be the read data. In this case the nodeID in the CAN message header is the nodeID of the responding node. Only the node that initiated the "READ" should listen for this message (e.g. with a state machine change to "WAIT FOR READ RESPONSE"). An invalid "READ" request results in a "Read Response" with only 1 byte (the Read "Address") in the payload.

Read Response (5)	**Length 1 to 8 bytes**
Byte 0:	"Address" of this "READ" (copied from the received message).
Bytes 1-7:	1 to 7 additional bytes representing data value resulting from "reading" the requested address. Lack of these bytes (*i.e.* a short message) indicates an invalid read address/request.

1.7 "ALERT" (Code=7)

This packet type is issued spontaneously "up the tree" when the TDIG or TCPU first starts-up or when it encounters an unexpected condition (*e.g.* an FPGA error, Overtemperature condition, etc). The issuing board will encode its BoardID in the packet header. The payload will contain information identifying the nature of the Alert.

Alert (7)	**Alert (Lengths 1, 3, 4)**
Byte 0:	Nature of the Alert. See section <X.X> 0xFF = MCU Start-up. Additional bytes will be all 0x00. 0xCF = TDIG Clock failed 0xC2 = TCPU CANBus#2 overrun/error. 0xC1 = TDIG or TCPU CANBus#1 (Tray CAN) overrun/error 0x04 = FPGA CRC error. 0x09 = Overtemperature alert. Additional byte contains source of overtemperature (bit 0 = Board U37; bit 1 = TINO1; bit 2 = TINO2)
Bytes 1-7:	0 to 7 additional bytes containing additional information about the Alert.

2. HPTDC Control and Status:

The commands defined in the original TDIG CANbus HLP (version 1.6) document can be mapped onto into the new scheme as shown following for each command

2.1 SET_CONTROL Word (to HPTDC)

Writing to an HPTDC CONTROL register using the JTAG instruction ins[3:0]=0b1001 is a "WRITE" command with the following payload:

Write (2)	**Set Control to HPTDC (Length=6)**
Byte 0:	`0000 01tt` (tt = HPTDC number, 1 – 3; ***tt=00 indicates set-control to all 3 HPTDCs***) `(0x04, 0x05, 0x06, or 0x07)`

Bytes 1 – 5:	`cccccccc cccccccc cccccccc cccccccc cccccccc`
	`^bit[0] ^bit[39]`
	(40 bit control word, bit 0 is the right most bit of byte 1)

This message is invalid when sent to a TCPU board.
See HPTDC Chip documentation Section 17.6 for details on bit functions.
This function updates only the volatile (in-chip and in-MCU-RAM) control word images; see Section 2.2 for updating the default Control Word images.
Response: Write Response with the following payload:

Write Response (3)	**(Length=2)**

Byte 0:	`0000 01tt`
	(tt = TDC number, 1 – 3)
	`(0x04, 0x05, 0x06, or 0x07)`

Byte 1:	Status of the Write. 0= Successful; Non-zero <to be determined> non-successful or "warning" *Table x*

2.2 SET Default CONTROL Word to MCU Memory

During power-on initialization and through the Reset HPTDC command, the MCU retrieves a compiled-in default Control word from memory. This default can be overwritten using the large-block download and memory-write sequence (Section 1.2.2) using Block Target = CONTROL MEMORY after download.

2.2.1 Block Start

	Block-Start sub-command (Length=6)
Byte 0:	`0001 0000` (0x10)
Bytes 1 – 5:	40-bit Control Word to store. LSbit is bit 0 of first byte.

As an alternative, the Block-Start; Block-Data; Block-End sequence may be used with no data transferred in the Block-Start message.

Response: "Write Response" with status.

Write Response (3)	**Length = 2**
Byte 0:	0001 0000 (0x10)
Byte 1:	Status resulting from this write or sub-command. A value=0 indicates successful completion. Non-zero values indicate (various) errors (to be determined; see *Table x*)

2.2.2 Block End

This marks the end of data transfer. "Block-End" without a preceding "Block-Start" is an error.

Write (2)	**Block-End sub-command (Length=1)**
Byte 0:	0011 0000 (0x30)

Response: 8-byte "Write Response" with status, number of bytes received, and checksum of bytes received.

Write Response (3)	**Length = 8**
Byte 0:	"Address" or "Sub-Command" of this "WRITE" (copied from the received message.
Byte 1:	Status resulting from this write or sub-command. A value=0 indicates successful completion. Non-zero values indicate (various) errors (to be determined; see *Table x*)
Bytes 2-3:	Total number of bytes in block transferred (LSByte first)
Bytes 4-7:	Checksum of data bytes transferred (does not include protocol bytes)

2.2.3 Block-Disposition (Block-Target)HPTDC CONTROL MEMORY

Directs the received Control Word block to the MCU flash memory area used at the next power-up or reset cycle. "Block-Disposition" without "Block-End" is an error. The Block-Disposition message for HPTDC Control memory will be 1 byte long. This function is invalid when addressed to a TCPU board.

Write (2)	**Block-Disposition sub-command (Length=1)**
Byte 0:	0100 tttt (0x4t) tttt = target for disposition of the buffered data 0x48 = All 3 HPTDCs chip Control Memory (FLASH). 0x49 = HPTDC1 chip Control Memory (FLASH). 0x4A = HPTDC2 chip Control Memory (FLASH). 0x4B = HPTDC3 chip Control Memory (FLASH).

Response: "Write Response" with status.

Write Response (3)	**Length = 2**
Byte 0:	0100 tttt (0x4t) "Address" or "Sub-Command" of this "WRITE" (copied from the received message).
Byte 1:	Status resulting from this write or sub-command. A value=0 indicates successful completion. Non-zero values indicate (various) errors (to be determined; see *Table x*)

2.3 Read CONTROL Word (From HPTDC)

This function returns the last-used Control Word for the HPTDC chip from the MCU's dynamic-memory copy. This control word will represent the last control word presented to the HPTDC as a result of power-on, HPTDC Reset, or Set_Control_Word CANBus command.

Read (4)	**Read Control from HPTDC (Length=1)**
Byte 0:	0000 00tt (tt = HPTDC number, 1 – 3; ***tt=00 indicates read-control from all 3 HPTDCs***) (0x00, 0x01, 0x02, or 0x03)

This message is invalid when sent to a TCPU board.
When Byte 0 is 0x00, indicating "All 3 HPTDCs", there will be three (3) response messages with Byte 0 codes 0x01, 0x02, and 0x03 and the corresponding HPTDC Control Word.

Response(s): Read Response(s) each with the following payload:

Read Response (5)	**(Length=6)**
Byte 0:	0000 00tt (tt = HPTDC number, 1 – 3) (0x01, 0x02,or 0x03)
Bytes 1 – 5:	cccccccc cccccccc cccccccc cccccccc cccccccc ^bit[0] ^bit[39] (40 bit control word, bit 0 is the right most bit of byte 1)

2.4 GET_STATUS (HPTDC)

This function returns the contents of the HPTDC Status Register (see HPTDC documentation section 17.7. This is a "READ" command with the following payload:

Read (4)	**Get HPTDC Status (Length=1)**
Byte 0:	`0000 01tt` (tt = TDC number, 1 – 3) (0x05, 0x06, or 0x07)

This message is invalid when sent to a TCPU board.
The Read Response will consist of **two** packets.
First a READ Response packet of length 8, containing the following payload:

Read Response (5) (1 of 2)	**Length=8**
Byte 0:	`0000 01tt` (tt = TDC number, 1 – 3) (0x05, 0x06, or 0x07)

Bytes 1 – 7:	`ssssssss ssssssss ssssssss ssssssss ssssssss`
	`^bit[0]`
	`ssssssss ssssssss`
	`^bit[55]`
	(s = TDC status [55:0], bit 0 is the right most bit of byte 1)

Second a READ Response packet of length 2, containing the following payload:

Read Response (5) (2 of 2)	**Length = 2**
Byte 0:	`0000 01tt` (tt = TDC number, 1 – 3) (0x05, 0x06, or 0x07)
Byte 1:	`ssssssss` (s = TDC status [63:56], bit 56 is the right most bit of byte 1)

2.5 CONFIGURE_HPTDC

The TDC configuration consists of 647 bits that need to be downloaded to each TDC. 647 bits is one bit shy of 81 bytes. A 0 should be appended to the *end* (MSB) of the configuration data(just above parity), and the very first bit sent should be the LSB of the configuration data. Correctly setting the parity bit is optional, however the parity bit will be included in the checksum calculation defined below. The MCU will calculate the correct parity after adjusting the TDC ID nibble of byte 5 so as to assign the appropriate HPTDC ID according to the following table:

Board ID	TDC #	Assigned ID		Board ID	TDC #	Assigned ID
0	1, 2, 3	0, 1, 2		4	1, 2, 3	0, 1, 2
1	1, 2, 3	4, 5, 6		5	1, 2, 3	4, 5, 6
2	1, 2, 3	8, 9, A		6	1, 2, 3	8, 9, A
3	1, 2, 3	C, D, E		7	1, 2, 3	C, D, E

Transmission of the configuration data from the host is sub-divided into 4 stages:

- **Block-Start:** instructs the MCU to initialize the download buffer. Data will follow in later packets. Sending the START instruction will always re-start the download sequence. Seven (7) bytes of configuration data (LSB first) can be included in this message.
- **Block-Data:** contains the actual configuration data. Since there are a total of 81 bytes to send. If 7 bytes are sent with the Block-Start, then 11 additional Block-Data messages are sent with 7 bytes in the first 10 and 4 bytes in the 11th message. There are a total of 11 Block-Data messages (minimum).
- **Block-End:** advises the MCU that the PC is finished sending download data. The reply to this message will contain Status, Number of bytes received, Checksum of received bytes. The Host can use this to confirm proper reception of configuration data prior to issuing a Block-Disposition command.
- **Block-Disposition-ConfigureTDC:** The MCU will check to see that it has received a full TDC cofiguration (81 bytes). If that check is successful, the MCU will perform the following actions: a) load the updated configuration settings into a "base" configuration buffer; b) compute the TDC# nibble for the addressed TDC (#1, 2, or 3) per the table above; c) compute the proper Parity bit; d) save the configuration into the per-TDC configuration buffer, and e) program the configuration into the TDC. Block-Disposition Configure TDC #0 will cause all 3 HPTDCs to be configured using the appropriately modified configuration block.
  This message is invalid when sent to a TCPU board.

Each of the transmission sub-commands in the download sequence will be acknowledged with the appropriate "Write Response" and status (See section 1.2.2).

2.5.1 Block-Target Configure TDC

This tells the MCU what to do with the buffer of data *(e.g.* use it to configure an HPTDC). "Block-Disposition" without "Block-End" is an error. The Block-Disposition message, depending on the target, may contain additional address-type information (to be determined, *e.g.* EEPROM memory address).

Write (2)	**Block-Target sub-command (Length=1)**

Byte 0:	`0100 tttt`
	`(0x4t)`
	tttt = target for disposition of the buffered data
	0x40 = All 3 HPTDCs chip Configuration (volatile).
	0x41 = HPTDC1 chip Configuration (volatile).
	0x42 = HPTDC2 chip Configuration (volatile).
	0x43 = HPTDC3 chip Configuration (volatile).

Response: "Write Response" with status.

Write Response (3)	**Length = 2**

Byte 0:	`0100 tttt`
	`(0x4t)`
	tttt = target for disposition of the buffered data copied from Block-Target command.

Byte 1:	Status resulting from this write or sub-command. A value=0 indicates successful completion. Non-zero values indicate (various) errors (to be determined; see *Table x*)

To reapply the current configuration to the TDCs, send additional Block-Disposition-CONFIGURE_TDC subcommand(s) addressing the desired TDC (TDC #'s 1, 2, or 3). This will cause the MCU to configure the TDC with the configuration bits stored in MCU memory (previously applied configuration), or default configuration, if no previous change has been made. Block-Disposition Configure TDC #0 will cause all 3 HPTDCs to be configured using the appropriately modified configuration block

2.6 Save to HPTDC CONFIGURATION MEMORY

The initial TDC configuration consists of a built-in table of 647 bits used at power-up and MCU reset. That default configuration can be overwritten using the large-block download procedure (Section 1.2.2) with a block-target addressing the default configuration area. that need to be downloaded to each TDC. 647 bits is one bit shy of 81 bytes.

Transmission of the configuration data from the host is sub-divided into 4 stages:

- **Block-Start:** instructs the MCU to initialize the download buffer. Data will follow in later packets. Sending the START instruction will always re-start the download sequence. Seven (7) bytes of configuration data (LSB first) can be included in this message.
- **Block-Data:** contains the actual configuration data. Since there are a total of 81 bytes to send. If 7 bytes are sent with the Block-Start, then 11 additional Block-Data messages are sent with 7 bytes in the first 10 and 4 bytes in the 11^{th} message. There are a total of 11 Block-Data messages (minimum).
- **Block-End:** advises the MCU that the PC is finished sending download data. The reply to this message will contain Status, Number of bytes received, Checksum of received bytes. The Host can use this to confirm proper reception of configuration data prior to issuing a Block-Disposition command.
- **Block-Disposition-TDC Configuration Memory:**

Each of the transmission sub-commands in the download sequence will be acknowledged with the appropriate "Write Response" and status (See section 1.2.2).

2.6.1 Block-Target Configuration MEMORY (Non Volatile)

This tells the MCU what to do with the buffer of data *(e.g.* use it to configure an HPTDC). "Block-Disposition" without "Block-End" is an error. The Block-Disposition message, depending on the target, may contain additional address-type information (to be determined, *e.g.* EEPROM memory address).

Write (2)	**Block-Target sub-command (Length=1)**

Byte 0:	`0100 tttt`
	`(0x4t)`
	tttt = target for disposition of the buffered data
	0x44 = All 3 HPTDCs chip Configuration Memories (FLASH).
	0x45 = HPTDC1 chip Configuration Memory (FLASH).
	0x46 = HPTDC2 chip Configuration Memory (FLASH).
	0x47 = HPTDC3 chip Configuration Memory (FLASH).
	0x48 = All 3 HPTDCs chip Control Memory (FLASH).

Response: "Write Response" with status.

Write Response (3)	**Length = 2**

Byte 0:	`0100 tttt`
	`(0x4t)`
	tttt = target for disposition of the buffered data copied from Block-Target command.

Byte 1:	Status resulting from this write or sub-command. A value=0 indicates successful completion. Non-zero values indicate (various) errors (to be determined; see *Table x*)

2.7 Read HPTDC Configuration

This function returns the last-programmed Configuration setup (647 bits) for the HPTDC chip from the MCU's in-memory copy.

Read (4)	**Read Configuration from HPTDC (Length=1)**
Byte 0:	`0000 01tt` (0x4tt = HPTDC number, 1 – 3; ***tt=00 indicates read-control from all 3 HPTDCs***) `(0x40, 0x41, 0x42, or 0x43)`

This message is invalid when sent to a TCPU board.

Responses: For each HPTDC Configuration requested there will be 12 Read Responses. The first payload byte of each response message will indicate the HPTDC number (0x41, 0x42, or 0x43). The remaining 7 bytes will contain successive bytes of successive bits. Bit 0 of the configuration will appear in bit 0 of the first data byte (LSByte). Subsequent messages will contain successive bits in order. The first 11 response messages will be of length 8, the final message will be length 5 for a total transfer of 81 data bytes.

When Byte 0 is 0x40, indicating "All 3 HPTDC Configurations", there will be three (3) ***sets*** of response messages with Byte 0 codes 0x41, 0x42, and 0x43 and the corresponding HPTDC Configuration bits.

First Response Message:

Read Response (5)	**(Length=8)**
Byte 0:	0000 01tt (tt = HPTDC number, 1 – 3) (0x41, 0x42,or 0x43)
Bytes 1 – 7:	cccccccc cccccccc cccccccc cccccccc cccccccc ^bit[0] cccccccc cccccccc ^bit[55]

Response Messages 2 through 11:

Read Response (5)	**(Length=8)**
Byte 0:	0000 01tt (tt = HPTDC number, 1 – 3) (0x41, 0x42,or 0x43)
Bytes 1 – 7:	cccccccc cccccccc cccccccc cccccccc cccccccc cccccccc cccccccc subsequent bits of configuration image.

Final Response Message 12:

Read Response (5)	**(Length=5)**
Byte 0:	0000 01tt (tt = HPTDC number, 1 – 3) (0x41, 0x42,or 0x43)
Bytes 1 – 4:	cccccccc cccccccc cccccccc 0ccccccc ^bit[647] uppermost bits of configuration image.

2.8 Issue HPTDC Reset Sequence

This command causes the MCU to issue sequential JTAG commands to the HPTDC Chip(s) in order to perform a reset of the chips. The sequence consists of successive control words which a) reset_all; b) lock_pll; c) lock_dll; d) global_reset; e) final_control (from the non-volatile control-word image).

Write (2)	**Issue HPTDC Reset Sequence (Length=1)**
Byte 0:	1001 000t (t = TDC number, 1 – 3; ***t=0 indicates reset to all 3 HPTDCs***) (0x90, 0x91, 0x92, 0x93)

This message is invalid when sent to a TCPU board.
Response: Write Response with the following payload will be issued when reset sequence(s) are complete:

Write Response (3)	**Length =2**

Byte 0:	`1001 000t`
	(t = TDC number, 1 – 3; ***t=0 indicates reset to all 3 HPTDCs***)
	`(0x90, 0x91, 0x92, 0x93)`

Byte 1:	Status byte, 0x00 = success, <tbd> error

3. BOARD Options and Status:

The commands in this section are intended to set or read board options and status.

3.1 Board Status Requests:

These commands return (read) information about board condition and settings:

3.1.1 GET_STATUS (TDIG Board)

To request the TDIG board status, the "READ" command payload looks like this:

Read (4)	**Get Status (TDIG Board) (Length=1)**
Byte 0:	1011 0000 (0xB0)

The "READ Response" message of length 8 will contain:

Read Response (5)	**Length = 8**
Byte 0:	`1011 0000` `(0xB0)`
Bytes 1 – 7:	`aaaa0000 bbbbbbbb cccccccc dddddddd 0000dddd eeeeeeee 0000eeee` a = "fraction" byte of TDIG temperature reading (at chip location U37) b = "units" byte of TDIG temperature reading (at chip location U37) units are signed degrees C. See the description of the "Read Temperature" message (0x09) or the Microchip MCP9801 datasheet for description. c = Expander CSR bits (see command 0xB4 "Read ECSR) for more information) c.0 = PLD Config Done (usually 1) c.1 = PLD Init Done (usually 1) c.2 = PLD CRC Error (usually 0) c.3 = PLD nSTATUS (usually 1) c.4 = TDC Power Error_B (usually 1, 0=power error) c.5 = Enable TDC Power (output, reads 1 when HPTDC Power ON) c.6 = Tino Test MCU (output) c.7 = Spare PLD signal. d = TINO temp 1 digitization. e = TINO temp 2 digitization. TINO digitization values are unsigned integers. The (ideal) transfer function is: V(volts) = (value/4096) x 3.3 IF the TINO board temperature chip is the TC1047 then correspondence of these values to TINO temperature is via the (ideal) transfer function: temperature°C = ((voltage – 0.5) / 0.010 The combined transfer function is: Temperature (°C) = ((value * 330)/4096) – 50

3.1.2 GET_STATUS (TCPU Board)

To request the TCPU board status, the "READ" command payload looks like this:

Read (4)	**Get_Status (TCPU Board) (Length=1)**
Byte 0:	1011 0000 (0xB0)

The "READ Response" message of length 8 will contain:

Read Response (5)	**Length = 8**
Byte 0:	1011 0000 (0xB0)
Bytes 1 – 7:	aaaa0000 bbbbbbbb cccccccc 0x00 0x00 0x00 0x00 a = "fraction" byte of TDIG temperature reading (at chip location U37) b = "units" byte of TDIG temperature reading (at chip location U37) units are signed degrees C. See the description of the "Read Temperature" message (0x09) or the Microchip MCP9801 datasheet for description. c = Expander CSR bits (see command 0xB4 "Read ECSR) for more information: c.0 = PLD Config Done (normally 1) c.1 = PLD Init Done (normally 1) c.2 = PLD CRC Error (normally 0) c.3 = PLD nSTATUS (normally 1) c.4 = Pushbutton SW1 (1 = button pressed) c.5 = Jumper JU2 pins 5-6 (1=jumper installed) c.6 = Jumper JU2 pins 3-4 (1=jumper installed) c.7 = Jumper JU2 pins 1-2 (1=jumper installed)

3.1.3 Read MCU Firmware Identifiers:

"Read" command for requesting the firmware ID codes from the MCU and FPGA. See section <tbd> for commands which make actual firmware checksums available.

Read (4)	**Get_Firmware ID (Length=1)**
Byte 0:	1011 0001 (0xB1)

Message Response: "Read Response" with contents of the software defined version identifier.

Read Response (5)	**Length = 8**
Byte 0:	1011 0001 (0xB1)
Byte 1:	LS Byte of MCU firmware identifier (FIRMWARE_ID_0)
Byte 2:	MS Byte of MCU firmware identifier (FIRMWARE_ID_1)
Byte 3:	Contents of FPGA Register 7 (ID Register)

These bytes are software defined within the source code of the MCU and FPGA firmware and are intended to assist with configuration control.

3.1.4 Read Board Hardware Serial Number (TDIG and TCPU)

"Read" command for reading the board hardware serial number. This number is unique to the board but is not the same as the "human readable" serial number tag.

Read (4)	**Get_Hardware_SN (Length=1)**
Byte 0:	1011 0010 (0xB2)

Message Response: "Read Response" with Contents of hardware serial number chip (U60).

Read Response (5)	**Length = 8**
Byte 0:	1011 0010 (0xB2)
Bytes 1-7:	Contents of Board Serial Number chip (U60) bytes 1 thru 7. Byte 0, the fixed byte, is not sent.

3.1.5 Read Switch/Jumper configuration (TDIG and TCPU)

"Read" command for reading the TDIG or TCPU board Switch/Jumper Inputs. This is read through the expander port at U35 (MCP23008), I2C address 0x42 . The Switch/Jumper input port is configured for "Inverted" inputs and pull-ups enabled. This means an open-circuit "high level" is read as a zero-bit. A grounded input "low level" is read as a one-bit. Therefor, jumpers INSTALLED are returned as "One-bits". Schematic text showing "adr=44x" for U35 is incorrect.

Read (4)	**Get_JumperSwitch (Length=1)**
Byte 0:	1011 0011 (0xB3)

Message Response: "Read Response" with contents of Jumper/Switch register (U35).

Read Response (5)	**Length = 2**	
Byte 0:	1011 0011 (0xB3)	
Byte 1:	Contents of the Jumper/Switch register:	
	For TDIG: Bit 0 = Button status; 1=button pressed	For TCPU: Bit 0 = SW4.1 (Units.1)
	Bit 1 = SW4.4	Bit 1 = SW4.2 (Units.2)
	Bit 2 = SW4.2	Bit 2 = SW4.4 (Units.4)
	Bit 3 = SW4.1	Bit 3 = SW4.8 (Units.8)
	Bit 4 = JU2.7-8; 1=jmpr installed	Bit 4 = SW5.1 (Tens.1)
	Bit 5 = JU2.5-6; 1=jmpr installed	Bit 5 = SW5.2 (Tens.2)
	Bit 6 = JU2.3-4; 1=jmpr installed	Bit 6 = SW5.4 (Tens.4)
	Bit 7 = JU2.1-2; 1=jmpr installed	Bit 7 = SW5.8 (Tens.8)

3.1.6 Read Control/Status (TDIG)

"Read" command for reading the TDIG or TCPU Extended Control/Status register (ECSR)
This command is somewhat redundant because the same information is included in the "GET STATUS" reply. This is read through the expander port at U36 (MCP23008), I2C address 0x44 . The ECSR port contains both inputs (bits 0..4,7) and outputs (bits 5,6). The inputs are configured for "Normal, non-inverted" inputs and pull-ups enabled. This means an open-circuit "high level" is read as a one-bit. A grounded input "low level" is read as a zero-bit. Schematic text showing "adr=42x" for U36 is incorrect.

Read (4)	**Get_ExtendedCSR (Length=1)**
Byte 0:	1011 0100 (0xB4)

Message Response: "Read Response" with contents of Jumper/Switch register (U36).

Read Response (5)	**Length = 2**
Byte 0:	1011 0100 (0xB4)
Byte 1:	Contents of the ECSR register: Bit 0 = PLD_CONFIG_DONE (Normally 1) 1 = PLD_INIT_DONE (Normally 1) 2 = PLD_CRC_ERROR (Normally 0) 3 = PLD_nSTATUS (Normally 1) 4 = TDC_POWER_ERROR_B (Normally 0) 5 = ENABLE_TDC_POWER (output) (Normally 1) 6 = TINO_TEST_MCU (output) (Normally 0) 7 = SPARE_PLD (undefined)

Write TDIG Control/Status
"Write" command variant to defined for Writing to the TDIG Extended Control/Status settings (writeable bits only). Write Response <to be defined>. This command will be used for enabling TDC power and for triggering MCU-generated test signals.

3.1.7 Read Control/Status (TCPU)

"Read" command for reading the TCPU Extended Control/Status register (ECSR)
This command is somewhat redundant because the same information is included in the "GET STATUS" reply. This is read through the expander port at U36 (MCP23008), I2C address 0x44 . The ECSR for TCPU contains only input bits. The Jumper/Switch inputs (bits 4..7) are configured for "Inverted" inputs and pull-ups enabled. This means an open-circuit "high level" is read as a one-bit. A grounded input "low level" is read as a zero-bit. PLD Status bits are not inverted. Schematic text showing "adr=42x" for U36 is incorrect.

Read (4)	**Get_ExtendedCSR (Length=1)**
Byte 0:	1011 0100 (0xB4)

Message Response: "Read Response" with contents of Extended Control/Status Register (ECSR)

Read Response (5)	**Length = 2**
Byte 0:	1011 0100 (0xB4)
Byte 1:	Contents of the ECSR register: Bit 0 = PLD_CONFIG_DONE (usually=1) 1 = PLD_INIT_DONE (usually=1) 2 = PLD_CRC_ERROR (usually= 0) 3 = PLD_nSTATUS (usually= 1) 4 = SW1 pushbutton (pressed=1) 5 = Jumper JU2 pins 5-6 (jumper installed=1) 6 = Jumper JU2 pins 3-4 (jumper installed=1) 7 = Jumper JU2 pins 1-2 (jumper installed=1)

3.1.8 Read MCU Status

"Read" command for reading the MCU status (This command is only parially implemented)

Read (4)	**Get_MCU_Status (Length=1)**
Byte 0:	1011 0100 (0xB5)

Message Response: "Read Response" with contents of Extended Control/Status Register (ECSR)

Read Response (5)	**Length = 3 (if returning clock status) or 5 (if returning reset vector contents)**
Byte 0:	1011 0100 (0xB4)
Bytes 1-4:	Contents of the MCU reset vector (address) or contents of the MCU Clock control register.

3.2 Board Control Requests:

These commands set board options and return (read) information about board settings:

3.2.1. Write Threshhold (TDIG Board)

A "Write" command with the following payload:
This message is invalid when sent to a TCPU board.

Write (2)	**Threshhold Set (TDIG board) (Length=3)**
Byte 0:	0000 1000 (0x08)
Bytes 1 – 2:	ddddddddd 0000dddd ^bit[0] (12 bit control word, bit 0 is the right most bit of byte 1)

d = DAC word [11:0]

The DAC word is related to the threshold voltage sent to the TDC-TINO connector by the following (idealized) equation:

V_{out} = 3.3V * (D / 4095.)

The maximum threshold setting is approximately 3300 mV, DAC word = 0xFFF. The minimum threshold is approximately 0mV, DAC word = 0x000.

The MCU initializes the DAC to a Vout of 2500 mV during start-up (this default setting is defined in source file TDIG-F_Board.h)

Message Response: "Write Response" with status. There is no provision for ADC readback of the threshhold voltage.

Write Response (3)	**Length = 2**
Byte 0:	0000 1000 (0x08)
Byte 1:	Status (Table <tbd>)

Example:
To set the DAC output on Board 0:

DAC output	**Send**			**Receive**		
Volts (nominal)	Message	Length	Payload	Message	Length	Payload
0	0x102	3	0x08 0x00 0x00	0x103	2	0x08 0x00
3.3	0x102	3	0x08 0xFF 0x0F	0x103	2	0x08 0x00
2.5	0x102	3	0x08 0x1E 0x0C	0x103	2	0x08 0x00
1.25	0x102	3	0x08 0x0F 0x06	0x103	2	0x08 0x00
1.	0x102	3	0x08 0xD9 0x04	0x103	2	0x08 0x00

3.2.2 THRESHOLD READ

A "READ" command with the following payload:
This message is invalid when sent to a TCPU board.

Read (4)	**Threshhold Read (Length=1)**
Byte 0:	0000 1000 (0x08)

Message Response: "Read Response" with threshhold set value. There is no provision for ADC readback of the threshhold voltage setting to the DAC.

Read Response (5)	**(Length= 3)**
Byte 0:	0000 1000 (0x08)
Bytes 1 – 2:	dddddddd 0000dddd ^bit[0] (12 bit control word, bit 0 is the right most bit of byte 1)

d = DAC word [11:0]

The DAC word is related to the threshold voltage sent to the TDC-TINO connector by the following (idealized) equation:

$V_{out} = 3.3V * (d / 4095.)$

To read the DAC setting on Board 0:

DAC output	**Send inquiry**			**Receive reply**		
Volts (nominal)	Message	Length	Payload	Message	Length	Payload
0	0x104	1	0x08	0x103	2	0x08 0x00 0x00
3.3	0x104	1	0x08	0x103	2	0x08 0xFF 0x0F
2.5	0x104	1	0x08	0x103	2	0x08 0x1E 0x0C
1.25	0x104	1	0x08	0x103	2	0x08 0x0F 0x06
1.	0x104	1	0x08	0x103	2	0x08 0xD9 0x04

3.2.3 Request Board Temperatures

"Read" commands for obtaining the board temperature and digitized voltage from TINO temperature monitors. The same command is used for reading from TDIG-TINO and TCPU. The replies are different.

Read (4)	**Temperature Read (Length=1)**
Byte 0:	0000 1001 (0x09)

3.2.3.1 Board Temperatures (TDIG and TINO) Response

Message Response: "Read Response" from TDIG board with results of temperature measurement (U37) and A/D conversion.

Read Response (5)	**Length = 7**
Byte 0:	0000 1001 (0x09)
Bytes 1-2:	Temperature read from U35 temperature chip (LSByte, MSByte). MSByte is temperature in degrees-C. LSByte is fractional degrees with zero-filled bits.
Bytes 3-4:	Digitized voltage from TINO_TEMP1 signal (LSByte, MSByte). 12-bit digitization. Unused high-order bits are zero-filled.
Bytes 5-6:	Digitized voltage from TINO_TEMP2 signal (LSByte, MSByte). 12-bit digitization. Unused high-order bits are zero filled.

Bytes 1 and 2 are for TDIG temperature as sensed at the chip U37 approximately in the center of the TDIG board. A value representing the temperature in degrees C is returned. Two bytes are interpreted as a *signed* two's compliment 16-bit integer having an implied binary point between the two bytes and the 4 LSBits always zero. The reported temperature is thus: (float)Value / 256.0 = Temperature (°C). If only the MSByte is considered as a signed value (0xC9 to 0x7D equal to -55. to +125.) , it represents the temperature truncated to integer (°C). The chip can report temperatures from -55 °C to +125 °C. See the datasheet for the Microchip MCP9801 for more information.

Bytes 3 and 4; and 5 and 6 are used to return an unsigned integer representing 12 bit digitization of the voltage present at signals TINO_TEMP1 (J13 pin 2) and TINO_TEMP2 (J13 pin 4). TINO digitization values are unsigned integers. The (ideal) transfer function is: V(volts) = (value/4096) x 3.3

IF the TINO board temperature chip is the TC1047 then correspondence of these values to TINO temperature is via the (ideal) transfer function: Temperature(°C)= ((Voltage – 0.5) / 0.010

The combined transfer function is:
Temperature (°C) = ((value * 330)/4096) – 50.

Read TINO Temperatures

This command has been incorporated into the Read TDIG temperature message (0x09).

3.2.3.2 Board Temperatures (TCPU) Response

Message Response: "Read Response" with current temperature values.

Read Response (5)	**Length = 3**
Byte 0:	0000 1001 (0x09)
Byte 1:	"fraction" byte of temperature sensed at U37, least-significant bits will be all zero.
Byte 2:	"units" byte of temperature sensed at U37

Temperature is sensed at the chip U37 located about 1 inch above and slightly left of the FPGA on a TCPU board. A value representing the temperature in degrees C is returned. Two bytes are interpreted as a *signed* two's compliment 16-bit integer having an implied binary point between the two bytes and the 4 LSBits always zero. The reported temperature is thus: (float)Value / 256.0 = Temperature (°C). If only the MSByte is considered as a signed value (0xC9 to 0x7D equal to -55. to +125.) , it represents the temperature truncated to integer (°C). The chip can report temperatures from -55 °C to +125 °C. See the datasheet for the Microchip MCP9801 for more information.

3.2.4 SET Overtemperature Limits

These commands are used to set upper limits on board temperatures. Board temperatures are continually monitored by the temperature sensing chips (U37) which generates an interrupt if the observed temperature exceeds the limit. This interrupt ultimately leads to the MCU issuing an "alert" message via the CANBus. In addition, the TDIG MCU continually monitors the digitized value of the (temperature-related) voltage reported by TINO and issues an "alert" message if an upper limit is exceeded.

3.2.4.1 SET Overtemperature Limits (TDIG)

"Write" command variant defined for configuring the TDIG board temperature limit. When temperature exceeds this limit an "Alert" message will be issued.

Write (2)	**Temperature Alert Set (Length=7)**
Byte 0:	0000 1001 (0x09)
Bytes 1–6:	dddddddd dddddddd eeeeeeee eeeeeeee ffffffff ^bit[0] ffffffff Where d = U37 (Board) temperature alert upper limit; e = TINO1 voltage limit equivalent; f = TINO2 voltage limit.

Bytes 3 and 4; and 5 and 6 are used to set an unsigned integer representing 12 bit value of the voltage present at signals TINO_TEMP1 (J13 pin 2) and TINO_TEMP2 (J13 pin 4). Based on the ideal transfer function TINO digitization values are computed as value = 4096 * (V(volts) / 3.3).

IF the TINO board temperature chip is the TC1047 then correspondence of these values to TINO temperature is via the (ideal) transfer function: Temperature(°C)= ((Voltage – 0.5) / 0.010

The combined transfer function for TINO settings are
Value = (Temperature (°C) + 50) * 4096 / 330

When the actual temperature exceeds the limit temperature, an "Overtemperature Alert" message will be issued. The Alert messages will continue at a rate of about once-per-5-seconds until the actual temperature is below the threshhold. For the TDIG and TCPU Board temperatures (U37), the "lower limit" is automatically set 5 degreesC below the upper limit. The default upper and lower limits existing at power-up for U37 are 80 and 75 degreesC. Once set, the values will be retained across MCU reset conditions unless power is cycled off.

Setting the TINO threshholds to an all-zeroes value disables recognition of overtemperature conditions related to the TINO voltages. The default condition at power-up and after reset is for the TINO over-temperature recognition to be disabled.

Message response: Write Response with status:

Write Response (3)	**(Length=2)**
Byte 0:	`0000 1000` `(0x09)`
Byte 1:	Status (table <tbd>)

3.2.4.2 SET Overtemperature Limits (TCPU)

"Write" command variant defined for configuring the TCPU board temperature limit. When temperature exceeds this limit an "Alert" message will be issued.

TCPU Variant: Beginning with TCPU MCU version 2G the follwing Overtemperature Limit setting message is defined:

Write (2)	**(Length=3)**
Byte 0:	`0000 1001` `(0x09)`

Bytes 1 – 2:	`dddddddd dddddddd`
	`^bit[0]`
	
	Where d = U37 (Board) temperature alert upper limit.

The same computations, default conditions, and actions pertain to TCPU as for TDIG.

Message response: Write Response with status:

Write Response (3)	**(Length=2)**
Byte 0:	`0000 1000`

	(0x09)
Byte 1:	Status (table <tbd>)

3.2.5 Set Board Clock Options (TDIG)

"Write" command variant defined for configuring the source of board clocking.

Write (2)	**Set Board Clock Source (Length=3)**
Byte 0:	0000 1101 (0x0D)
Byte 1:	Select Code: 0x00 == Use On-Board clock (Local clock) 0x01 == Use MCU internal clock (HPTDC not clocked) 0x08 == Use Tray-provided clock 0xFF == Use selection as set by jumpers (default)
Byte 2:	Select Code repeated (2 bytes must match)

Message response: Write Response with status:

Write Response (3)	**(Length=2)**
Byte 0:	0000 1000 (0x09)
Byte 1:	Status (table <tbd>)

3.2.6 Request Board Clock Options (TDIG)

"Read" command variant defined for determining the status of clocking.

Read (4)	**Read Board Clock Status (length=1)**
Byte 0:	0000 1101 (0x0D)

Message response: Read Response with status:

Read Response (5)	**(Length=5)**
Byte 0:	0000 1101 (0x0D)
Bytes 1-2:	Contents of MCU OSCCON register <tbd>
Byte 3:	Normal/Failed Status <tbd>
Byte 4:	Last requested Clock source type <tbd>

3.2.7 Issue TINO_TEST_MCU signal (TDIG)

"Write" command variants to cause the MCU to toggle the hardware line TINO_TEST_MCU once or "N" times.

3.2.7.1 Issue TINO_TEST_MCU signal Once (TDIG)

"Write" command variant to cause the MCU to toggle the hardware line TINO_TEST_MCU once. Issued pulse will be low-high-low with the high state lasting approximately <tbd>.

Write (2)	**Toggle TINO_TEST_MCU Once (length=1)**
Byte 0:	0000 1111 (0x0F)

Message response: Write Response with status will be issued when pulse sequence is complete:

Write Response (5)	**(Length=2)**
Byte 0:	0000 1111 (0x0F)
Byte 1:	Status (<tbd>)

3.2.7.2 Issue TINO_TEST_MCU signal Multiple times (TDIG)

"Write" command variants to cause the MCU to toggle the hardware line TINO_TEST_MCU multiple times. Issued pulses will be low-high-low with the high state lasting approximately <tbd> and interval <tbd>.

Write (2)	**Toggle TINO_TEST_MCU Multiple (length=3)**
Byte 0:	0000 1111 (0x0F)
Bytes 1-2:	llllllll hhhhhhhh LSByte, MSByte of 16 bit count value for number of pulses to be generated

Message response: Write Response with status will be issued ***when pulse sequence is complete:***

Write Response (5)	**(Length=2)**
Byte 0:	0000 1111 (0x0F)
Byte 1:	Status (<tbd>)

3.2.8 RESERVED FOR DIAGNOSTIC FUNCTIONS:

"Read" and "Write" command variants reserved for firmware debugging. Length, values, and interpretation are implementation dependent.

3.2.8.1DIAGNOSTIC WRITE FUNCTION:

Implementation dependent "Write" command.

Write (2)	**Diagnostic Write (length=1 to 8)**
Byte 0:	1111 1111 (0xFF)
Bytes 1-7: (optional)	Implementation dependent.

Message response: Read Response with status:

Write Response (5)	**(Length=1 to 8, implementation dependent)**
Byte 0:	1111 1111 (0xFF)
Bytes 1-7:	Implementation dependent

3.2.8.2 DIAGNOSTIC READ FUNCTION:

Implementation dependent "Read" command. These commands are considered "reserved" and should not be used except during program development and debugging. The use and contents are undefined for the purpose of this document.

Read (4)	**Diagnostic Read (length=1 to 8, implementation dependent)**
Byte 0:	1111 1111 (0xFF)
Bytes 1-7:	Implementation dependent.

Message response: Read Response with status:

Read Response (5)	**(Length=1 to 8, implementation dependent)**
Byte 0:	1111 1111 (0xFF)
Bytes 1-7:	Implementation dependent

3.2.9 Write pattern to LED (TDIG and TCPU):

Write bit pattern to LED register.

Write (2)	**Write To LEDs (length=2)**
Byte 0:	0000 1010 (0x0A)
Byte 1:	Bit pattern to write to LED register. Bit 0 = LED D0 Bit 1 = LED D1 .... Bit 7 = LED D7

Message response: Write Response with status:

Write Response (3)	**(Length=2)**
Byte 0:	0000 1010 (0x0A)
Bytes 1-7:	Status (see table <tbd>)

This is a direct-write to the driver latch register (U34) and writes the entire 8-bit pattern for (D0) through (D7). A (1) bit indicates LED_ON. This command overwrites any previous LED pattern and may (will) be overwritten by MCU operations. Other LEDs are not accessable to the MCU through this register.

4. FPGA/EEPROM Control and Configuration:

The commands in this section are intended to set or read FPGA registers, configure or reconfigure the FPGA, and download new firmware code into EEPROM #2 (alternate configuration EEPROM).

4.1 FPGA_WRITE_REG (PLD_WRITE_REG)

This command is used to load specific FPGA register(s) with data. Interpretation of these registers and data values is <tbd>. This command is a "WRITE" command with length 3, 5, or 7 bytes which allows for writing 1, 2, or 3 FPGA registers in one message. Registers are accessed in message-byte order. Repeated writing to the same address is allowed (e.g. to toggle a value).

Write (2)	**FPGA_Write_Register (Length=3, 5, or 7)**
Byte 0:	0000 1110 (0x0E)
Byte 1:	Register address to be written (see *Table y)*
Byte 2:	Value to be written. Some registers are "strobe" registers in which case the value is not important
[Byte 3] optional:	Optional second register address to be written (see *Table y)*
[Byte 4]:	Second value to be written; only present if second register is specified.
[Byte 5] optional:	Optional third register address to be written (see *Table y)*
[Byte 6]:	Third value to be written; only present if third register is specified.

Message Response: "Write Response" with status. Writing to undefined registers may produce unexpected results.

Write Response (3)	**(Length=2)**
Byte 0:	0000 1110 (0x0E)
Bytes 1-7:	Status (see table <tbd>)

Because multiple write accesses to the same register are allowed it is possible to set-and-clear a bit within a register using a single CANBus message. Writes are processed in order from byte 1 through byte 6.

4.2 FPGA_READ_REG (PLD_READ_REG)

This command is used to read specific FPGA register. Interpretation of these registers and data values is dependent on hardware/firmware within the FPGA. This command is a "READ" command with length 2, 3, or 4 bytes which allows for reading up to 3 FPGA registers in one message. Registers are accessed in message-byte order. Repeated reading of the same address is allowed. No checking of the FPGA address is performed, unexpected results may occur if undefined registers are accessed.

Read (4)	**FPGA_Read_Register (Length= 2, 3, or 4)**
Byte 0:	0000 1110 (0x0E)
Byte 1:	Register address to be read (see Table y)
[Byte 2] optional:	Optional second register address to be read.
[Byte 3] optional:	Optional third register address to be read.

Message Response: "Read Response" with data as follows. Reading from undefined registers may produce unexpected results and/or undefined values. Payload bytes 3 through 6 are only present if additional register(s) were specified in the FPGA_READ_REG message.

Read Response (5)	**(Length=3, 5, or 7)**
Byte 0:	0000 1110 (0x0E)
Byte 1:	Register address read (see Table y)
Byte 2:	Value read from address.
[Byte 3] optional:	Second register address read (see Table y) if a second register was specified in the Read.
[Byte 4:	Value read from second register.
[Byte 5] optional:	Thid register address read (see Table y) if a third register was specified in the Read.
[Byte 6]:	Value read from third address.

4.3 FPGA_CONFIGURE from EEPROM #1

"Write" command variants defined for configuring the FPGA from EEPROM #1 (non-volatile). This is the same image as is loaded upon power-up.

Write (4)	**Reconfigure from EEPROM #1 (length=5)**
Byte 0:	1000 1001 (0x89)
Bytes 1 – 4:	01101001 10010110 10100101 01011010 (0x69, 0x96, 0xA5, 0x5A) Fixed pattern to guard against spurious reconfigurations.

Message Response: "Write Response" with status and contents of FPGA Register 7 (Identification register)

Write Response (5)	**(length=3)**
Byte 0:	1000 1001 (0x89)
Byte 1:	Status: 0 = OK, Success.
Byte 2:	Result of reading FPGA register 7 after configuration is complete. The exact value will be determined by the FPGA VHDL code.

4.4 FPGA_CONFIGURE from EEPROM #2

"Write" command variants defined for configuring the FPGA from EEPROM #2 (Downloadable).

Write (4)	**Reconfigure from EEPROM #2 (length=5)**
Byte 0:	1000 1010 (0x8A)
Bytes 1 – 4:	01101001 10010110 10100101 01011010 (0x69, 0x96, 0xA5, 0x5A) Fixed pattern to guard against spurious reconfigurations

Message Response: "Write Response" with status and contents of FPGA Register 7 (Identification register)

Write Response (5)	**(length=3)**
Byte 0:	1000 1010 (0x8A)
Byte 1:	Status: 0 = OK, Success.
Byte 2:	Result of reading FPGA register 7 after configuration is complete. The exact value will be determined by the FPGA VHDL code.

4.5 FPGA_RESET (PLD_RESET)

This command is used to "hardware" reset the FPGA (equivalent to a power-on reset). The FPGA code image is not reloaded. Following this hardware reset the internal registers will be restored to their power-on default condition. This command is a "WRITE" command with length exactly 5 bytes and the following payload (bytes 1 – 4 are a fixed code to prevent accidental resets):

Write (2)	**FPGA Reset**
Byte 0:	0000 1100 (0x0C)
Bytes 1 – 4:	01101001 10010110 10100101 01011010 (0x69, 0x96, 0xA5, 0x5A) Fixed pattern to guard against spurious resets.

Message Response: "Write Response" with status.

Write Response (5)	**(length=2)**
Byte 0:	0000 1100 (0x0C)
Byte 1:	Status: 0 = OK, Success.

4.6 Downloading EEPROM #2 FPGA Code:

The series of commands is used to download a new FPGA firmware image into EEPROM #2 from an Altera programming file (RBF, Raw Binary File). This process consists of a series of block transfers using the "large-block" protocol (Sections 1.2.2 and following).
For each block of 256 bytes from the FPGA code "RBF" file, the following sequence of "Write" commands will be used. Because the EEPROM is limited to 256 bytes-per-write the maximum large-block transfer is limited to this amount.
a. Block-Start (with or without data bytes) (section 1.2.2.1);
b. Block-Data (36 full-size 8-byte payloads each with 1 byte subcommand + 7 bytes data) (section 1.2.2.2);
c. Block-Data (5 byte payload = 1 byte subcommand + 4 bytes data) (section 1.2.2.3);
d. Block-End (section 1.2.2.4);
e. The Block-End message results in a reply back to the host containing an 8-byte payload consisting of 1 byte command echo, 1 byte status, 2 bytes of bytecount, 4 bytes of checksum.
f. The host uses this echo to confirm (via length and checksum) that the target has received the proper number and checksum of bytes.
g. Block-Disposition. If the length and checksum are considered by the host to be valid, the host issues the block-disposition command containing a page-address to be written and a flag indicating whether the page is to be erased prior to writing. .

4.6.1 Block Start

See section 1.2.2.1 for a description of the Block-Start command.

4.6.2 Block Data

See section 1.2.2.2 for a description of the Block-Data command.

4.6.3 Block End

See section 1.2.2.3 for a description of the Block-Start command.

4.6.4 Block Target EEPROM#2

This command writes the (up to 256 byte) block to EEPROM #2 starting at the address in the message.

Write (2)	**Block-Disposition sub-command (Length=6)**
Byte 0:	0100 1110 (0x4E)=FPGA EEPROM #2.
Bytes 1 – 4:	These bytes will contain 1-4 memory-start address for the block.
Byte 5:	Erase flag (1=erase). A value=1 is normally required since the previous contents of the EEPROM are not guaranteed.

Response: "Write Response" with status. For disposition to EEPROM #2 with "erase" the disposition operation may take up to (3 seconds).

Write Response (3)	**Length = 2**
Byte 0:	0100 1110 (0x4E)=FPGA EEPROM #2.
Byte 1:	Status: 0 indicates successful completion. Non-zero values indicate (various) errors (to be determined; see *Table x*)

4.7 Reading EEPROM #2 Contents (TDIG and TCPU):

Due to the hardware design it is not possible to read the contents of EEPROM#1.

4.7.1 READ_EEPROM #2_Contents:

"Read" command for reading the contents of EEPROM#2 program memory.

Read (4)	**EEPROM2_Read_Memory (Length=5)**
Byte 0:	0100 1110 (0x4E)
Bytes 1 – 4:	4-byte starting address within EEPROM #2 space.

Message Response: "Read Response" containing seven bytes read from EEPROM #2 configuration memory starting at the specified address.

Read Response (5)	**(Length=8)**
Byte 0:	0100 1110 (0x4E) = Indicator of EEPROM#2 readout.
Bytes 1-7:	7-bytes of memory contents.

Note: EEPROM#1 cannot be read under program control.

4.8 CHECKSUM_EEPROM #2_Contents:

"Read" command for computing a checksum on the contents of EEPROM#2 memory.

Read (4)	**Checksum_EEPROM2_Contents (Length=8)**
Byte 0:	0100 1111 (0x4F)
Bytes 1 – 4:	4-byte starting address within EEPROM #2 space.
Bytes 5 – 7:	Number of 256-byte "sectors"over which to compute the checksum.

Message Response: "Read Response" containing checksum value computed over the specified range read from EEPROM #2.

Read Response (5)	**(Length=5)**
Byte 0:	0100 1111 (0x4F) = Indicator of EEPROM#2 checksum.
Bytes 1-4:	Checksum value 4-bytes (LSByte to MSByte). Computed as the sum-of-bytes.

NOTE:

1) For the EEPROMs used on both TDIG and TCPU, the maximum number of pages is 2048 (0x800).
2) This command will require about 1 second per 256 (0x100) pages, approximately 8 seconds for the entire EEPROM. The Read Reply message will not begin until the computation has completed.

5. MCU Control and Configuration (TDIG and TCPU):

The commands in this section are intended to start or restart MCU code-images.

5.1 MCU_RESTART (into first code image) (TDIG and TCPU):

This command first shuts off the TDC clocks, then starts executing the MCU's initialization code. This is as close to a power-off reset as one can get without actually turning off the power. *The TDCs will be restored to their default configuration. TDCs may be powered-off <TBD> if this message is applied to a TDIG board.*
This is a "WRITE" command with length exactly 5 bytes and the following payload (bytes 1 – 4 are an arbitrary code to prevent accidental restarts):

Write (2)	**MCU Restart (Length=5)**
Byte 0:	1000 1111 (0x8F)
Bytes 1 – 4:	01101001 10010110 10100101 01011010 (0x69 0x96 0xA5 0x5A)

Message Response: The message response is *two-fold*. First a "Write Response" with status. Upon completion of the Reset-Restart sequence, the normal "Alert" startup-status message will be sent.

Write Response (5)	**Length=2**
Byte 0:	1000 1111 (0x8F)
Bytes 1:	Status (see table <tbd>)

The Write Response message will be followed by an "Alert" message (section 1.7) after the MCU has performed its reinitialization.

5.2 MCU_START_SECOND_IMAGE

This command is used to begin execution of the "second image" (downloaded) MCU code.
This is a "WRITE" command with length exactly 5 bytes and the following payload (bytes 1 – 4 are a fixed code to prevent accidental restarts):

Write (2)	**MCU Restart (Length=5)**
Byte 0:	1000 1101 (0x8D)
Bytes 1 – 4:	01101001 10010110 10100101 01011010 (0x69 0x96 0xA5 0x5A)

Message Response: The message response is *two-fold*. First a "Write Response" with status.

Write Response (5)	**Length=2**
Byte 0:	1000 1101 (0x8D)
Bytes 1:	Status (see table <tbd>)

Upon completion of the Reset-Restart sequence (and depending on the nature of the second-image code) an "Alert" startup-status message (section 1.7) will be sent.

5.3 Downloading Code Image #2 into MCU (TDIG and TCPU):

The series of commands is used to download a new firmware image into the MCU Flash memory from a Microchip MPLAB programming file (.HEX). This process consists of a series of block transfers using the "large-block" protocol (Sections 1.2.2 and following).
For each block of 256 bytes from the Hex code "HEX" file, the following sequence of "Write" commands will be used.
a. Block-Start (with or without data bytes) (section 1.2.2.1);
b. Block-Data (36 full-size 8-byte payloads each with 1 byte subcommand + 7 bytes data) (section 1.2.2.2);
c. Block-Data (5 byte payload = 1 byte subcommand + 4 bytes data) (section 1.2.2.3);
d. Block-End (section 1.2.2.4);
e. The Block-End message results in a reply back to the host containing an 8-byte payload consisting of 1 byte command echo, 1 byte status, 2 bytes of bytecount, 4 bytes of checksum.
f. The host uses this echo to confirm (via length and checksum) that the target has received the proper number and checksum of bytes.

g. Block-Disposition. If the length and checksum are considered by the host to be valid, the host issues the block-disposition command containing a memory-address to be written and a flag indicating whether the page is to be erased prior to writing. It is not possible to write to EEPROM #1.

5.3.1 Block Start

See section 1.2.2.1 for a description of the Block-Start command.

5.3.2 Block Data

See section 1.2.2.2 for a description of the Block-Data command.

5.3.3 Block End

See section 1.2.2.3 for a description of the Block-Start command.

5.3.4 Block Target MCU2nd Image

This command writes the (up to 256 byte) block to EEPROM #2 starting at the address in the message.

Write (2)	**Block-Disposition sub-command (Length=6)**
Byte 0:	0100 1100 (0x4C)=MCU image #2.
Bytes 1 – 4:	These bytes will contain 1-4 memory-start address for the block.
Byte 5:	Erase flag (1=erase). A value=1 is normally required since the previous contents of the Flash memory are not guaranteed.

Response: "Write Response" with status. For disposition to EEPROM #2 with "erase" the disposition operation may take up to (3 seconds).

Write Response (3)	**Length = 2**
Byte 0:	0100 1100 (0x4C)=MCU image #2.
Byte 1:	Status: 0 indicates successful completion. Non-zero values indicate (various) errors (to be determined; see *Table x*)

5.4 Reading MCU Program Memory Contents (TDIG and TCPU):

5.4.1 READ_MCU_Memory_Contents:

"Read" command for reading the contents of MCU memory.

Read (4)	**EEPROM2_Read_Memory (Length=5)**
Byte 0:	0100 1100 (0x4C)
Bytes 1 – 4:	4-byte address within MCU code space.

Message Response: "Read Response" containing 4 bytes read from the MCU memory starting at the specified address.

Read Response (5)	**(Length=5)**
Byte 0:	0100 1100 (0x4C) = Indicator of MCU readout.
Bytes 1-4:	4-bytes of memory contents.

5.4.2 READ_MCU_Memory_Continuation:

Once the "Read" command for reading the contents of program memory has been issued with a starting-address specified, subsequent read requests do not need to have the address supplied.

Read (4)	**MCU_Read_Memory (Length=1)**
Byte 0:	0100 1100 (0x4C)

Message Response: "Read Response" containing four bytes read from EEPROM #2 configuration memory starting at the specified address.

Read Response (5)	**(Length=8)**
Byte 0:	0100 1100 (0x4C) = Indicator of EEPROM#2 readout.
Bytes 1-7:	7-byte memory contents.

5.4.3 READ_MCU_Memory (invalid address):

If an invalid or non-existant address is given for a "Read" command, the address will not be read.

Message Response: a short (1 byte) "Read Response" will be returned.

Read Response (5)	**(Length=1)**
Byte 0:	0100 1100 (0x4C) = Indicator of EEPROM#2 readout.

NOTE:

1) Reading "off the end" of the MCU physical address space will have the following effects:
    a. TDIG firmware versions prior to 11U - Hardware reset/restart.
    b. TDIG firmware versions at or after 11U - Single-byte Read Response with no data.
    c. TCPU firmware versions prior to 2F - Hardware reset/restart.
    d. TDIG firmware versions at or after 2F - Single-byte Read Response with no data.

5.5 CHECKSUM_MCU_Memory:

"Read" command for computing a checksum on the contents of MCU memory.

Read (4)	**Checksum_MCU_Contents (Length=8)**
Byte 0:	0100 1101 (0x4D)
Bytes 1 – 4:	4-byte starting address within MCU program memory.
Bytes 5 – 7:	Number of MCU "words"over which to compute the checksum.

Message Response: "Read Response" containing checksum of MCU memory contents over the address range specified.

Read Response (5)	**(Length=5)**
Byte 0:	0100 1101 (0x4D) = Indicator of EEPROM#2 checksum.
Bytes 1-4:	Checksum value 4-bytes (LSByte to MSByte). Computed as the sum-of-bytes.

NOTE:

1) Starting address is given in "bytes", length is given in "words".
2) Reading "off the end" of the MCU physical address space will stop the computation.
3) This command requires about 400 milliseconds per 0x1000 addresses. Message output does not begin until the computation has completed.

6. Data Transmission:

6.1MCU_DATA_READOUT_MODE

This command enables TDC data transmission and consists of a "Write" command with the following payload:

Bytes 0 – 1:	0000 1100 0000 00mm where m = mode select bits

m = 00 : Silent mode (default)
m = 01: Serial readout via PLD
m = 10: JTAG readout.

Message Response: "Write Response" with status. If the Write begins transmission of data, "DATA" packets will be sent.

The MCU can read out TDC data in three different modes:

PLD (serial) readout:

Data is read serially from the TDCs by the PLD and stored in an internal FIFO. The MCU reads this data from the PLD as 32-bit words and transmits those words up the CANbus via the "DATA" commands. This data can include any words inserted by the PLD, including data separators, geographical words, etc. Note that setting the MCU to this mode will cause any data in the PLD FIFOs to immediately be sent.

JTAG readout:

This is a special function mode used only for major debugging. In this mode, data is read directly from the TDCs by the MCU via the JTAG connector. Note that *the TDCs must be manually configured to allow JTAG readout.*

Silent mode (no readout):

In this mode, no data will be read out via CANbus. This is the default mode selected upon MCU startup.

7. ALERT conditions:

7.1 MCU_STARTUP

In order to keep the user of a potentially reprogrammed system informed, a message is sent at the beginning of the MCU code. The original spec uses the ID of a "reprogramming" packet.
This is now mapped to an "ALERT" packet with the following payload:

ALERT (7)	**MCU Startup (Length = 4)**
Bytes 0 – 3:	11111111 00000000 00000000 00000000 (0xFF 0x00 0x00 0x00)

The startup response packet includes the nodeID in the message header as described above. The payload of the startup response packet includes an initial byte of all 1's followed by three bytes of all zero's, representing the memory location of the startup code (location 0x000000). Newly reprogrammed code should indicate its memory location by inserting the startup location of the new code.

7.2 CLOCK_FAIL

ALERT (7)	**TDIG Clock Fail (Length =1)**
Byte 0:	11111100 (0xFC)

7.3 CANBUS_ERROR (TCPU)

ALERT (7)	**CANBus error on CANBus #1 (Length =2)**
Byte 0:	11000001 (0xC1)
Byte 1:	Error code

ALERT (7)	**CANBus error on CANBus #2 (Length =2)**
Byte 0:	11000010 (0xC2)
Byte 1:	Error code

7.4 HPTDC_CONFIGURATION_MISMATCH

7.5 TDC_POWER_ERROR

7.6 OVERTEMPERATURE

7.6.1 OVERTEMPERATURE (TDIG)

ALERT (7)	**Overtemperature Alert from TDIG/TINO (Length =2)**
Byte 0:	00001001 (0x09)
Byte 1:	Error code bit mask: Bit 0 indicates TDIG board overtemperature Bit 1 indicates TINO #1 voltage over limit Bit 2 indicates TINO #2 voltage over limit

7.6.1 OVERTEMPERATURE (TCPU)

ALERT (7)	**Overtemperature Alert from TCPU (Length =2)**
Byte 0:	00001001 (0x09)
Byte 1:	Error code bit mask: Bit 0 indicates TDIG board overtemperature

7.7 FPGA_CRC_ERROR

ALERT (7)	**FPGA CRC Error Reported (Length =1)**
Byte 0:	00000100 (0x04)

8. Obsolete Commands from earlier protocols:

8.1 CHANGE_MCU_PROGRAM

This command sequence is used to change the program code running on the MCU or PLD. Justin's original spec defined a sequence of sub commands similar to the CONFIGURE_TDC sequence. The commands defined are: START, DATA, CHECKSUM_64, PROGRAM_64, FINAL_CHKSUM, and JUMP_PC. See the original spec for the meaning of these sub commands.

For this new definition, we map these sub commands into the Large-Block Write Sequence as described above (Block-Start, Block-Data, Data End, Block-Disposition, and Checksum). The only difference in the sequence is in the final Block-Disposition sub-command. The sub-command will target the MCU for Block-Disposition of the data.

8.2 CHANGE_MCU_START

This command is superceded by the Block-Start subcommand and its response.

8.3 CHANGE_MCU_DATA

This command is superceded by the Block-Data subcommand and its response.

8.4 CHANGE_MCU_CHECKSUM_64

This command is superceded by issuing a Data-Checksum subcommand at the appropriate interval (*i.e.* each 64 bytes).

8.5 CHANGE_MCU_PROGRAM_64

This command is superceded by the Data-Block-Disposition subcommand and response containing the appropriate address.
The MCU commands remain <TBD> until the new-code download procedure is defined for the MCU.

8.6 CHANGE_MCU_FINAL_CHKSUM

This command has been superceded by the "End-Block" subcommand response which reports the number of bytes received and the checksum.

8.7 CHANGE_MCU_JUMP_PC

To Be Done:

- Add new commands for TDIG that take advantage of new hardware design (*e.g.* Write Temperature Alert, etc.)
- Update Command Summary Tables below to reflect defined usage.
- THUB message definition is outside the scope of this document.
- More?

COMMAND SUMMARY TABLES (see file CANMessage3jxls):

COMMAND STATUS TABLE (see file CANMessage3j.xls):

These values appear in the Status (Payload byte 1) field of WRITE reply messages.

Value in Payload[1]	Condition reported	From messages
0	OK / Success	
1	Invalid / Not Implemented	
2	Block Data / Block End / Disposition without Block Start	
3	Block Buffer Overrun	Block Data
4	Block Target Unknown	Block Disposition
5	Checksum Error (reserved)	
6	Block Target Length Incorrect	Block Disposition TDC length must be 61 bytes Block Disposition EEPROM2 length must be 256 bytes
7	Error during HPTDC Re-configuration	Block Disposition TDC
8	Error during EEPROM #2 Write	Block Disposition EEPROM2
9	Timeout during reconfiguration from EEPROM #2	PLD Configure from EEPROM2
A	MCU second image download address invalid	Block Disposition MCU second image addresses must be [0x100..0x200] or [0x4000..]

TCPU Extended CSR Bits Table (see file CANMessage3j.xls):

These values appear in the ECSR Payload byte field of READ Board Status or Read ECSR reply messages.

Value in Payload	Hardware line or Condition reported	
Bit 0	PLD_CONFIG_DONE	Usually 1
Bit 1	PLD_INIT_DONE	Usually 1
Bit 2	PLD_CRC_ERROR	Usually 0
Bit 3	PLD_nSTATUS	Usually 1
Bit 4	Pushbutton	Usually 0, 1=button pressed
Bit 5	JU2 jumper 5-6	1 = jumper installed

Value in Payload	Hardware line or Condition reported	
Bit 6	JU2 jumper 3-4	1 = jumper installed
Bit 7	JU2 jumper 1-2	1 = jumper installed

TDIG Extended CSR Bits Table (see file CANMessage3j.xls):

These values appear in the ECSR Payload byte field of READ Board Status or Read ECSR reply messages.

Value in Payload	Hardware line or Condition reported	
Bit 0	PLD_CONFIG_DONE	Usually 1
Bit 1	PLD_INIT_DONE	Usually 1
Bit 2	PLD_CRC_ERROR	Usually 0
Bit 3	PLD_nSTATUS	Usually 1
Bit 4	TDC_POWER_ERROR_B	Usually 1 (0 indicates power regulator error)
Bit 5	ENABLE_TDC_POWER	(output)
Bit 6	TINO_TEST_MCU	(output)
Bit 7	SPARE_PLD	Usually 1 (depends on FPGA code)

TCPU Switch Bits Table (see file CANMessage3j.xls):

These values appear in the Read Jumper/Switch settings reply message. Actual board position value is masked to the range 0..31.

Value in Payload	Hardware line or Condition reported	Board Position switch
Bit 0	SW4 weight 1	Board Position 1
Bit 1	SW4 weight 2	Board Position 2
Bit 2	SW4 weight 4	Board Position 4
Bit 3	SW4 weight 8	Board Position 8
Bit 4	SW5 weight 1	Board Position 10
Bit 5	SW5 weight 2	Board Position 20
Bit 6	SW5 weight 4	Board Position 40
Bit 7	SW5 weight 8	Board Position 80

TDIG Switch Bits Table (see file CANMessage3j.xls):

These values appear in the Read Jumper/Switch settings reply message. Actual board position value is masked to the range 0..7.

Value in Payload	Hardware line or Condition reported	
Bit 0	SW1 pushbutton	Usually 0, 1=button pressed
Bit 1	SW4 weight 4	Board Position 4
Bit 2	SW4 weight 2	Board Position 2
Bit 3	SW4 weight 1	Board Position 1
Bit 4	JU2 jumper 7-8	“1” = jumper installed
Bit 5	JU2 jumper 5-6	“1” = jumper installed
Bit 6	JU2 jumper 3-4	“1” = jumper installed
Bit 7	Ju2 jumper 1-2	“1” = jumper installed

TDIG Overtemperature Mask Bits Table (see file CANMessage3j.xls):

These values appear in the Overtemperature Alert message.

Value in Payload	Hardware line or Condition reported	
Bit 0	MCU is over limit.	
Bit 1	TINO1 value is over limit.	
Bit 2	TINO2 value is over limit.	
Bit 3		
Bit 4		
Bit 5		
Bit 6		
Bit 7		

Revision History:

Revision 3j – in process from Revision 3h:
Renumber and reorder some sections for flow and consistency.
Rename "Alarm" to "Alert" for consistency (also applies to CANMessage3j.xls).
Add Alarm message tables.
Alter name of CANMessage spreadsheet.

Revision 3h – in process from Revision 3g:
Restructured in concert with CANMessage3h.xls command/response spreadsheet; Renumbered and reordered sections and commands.
Incorporate CANScript opcodes.

Revision 3g-in process from Revision 3f:
Reword and clarify certain sections and tables.
Add description of Overtemperature alerts and temperature limit setting.
Add PLD_CHECKSUM and MCU_CHECKSUM descriptions.
Correct the description of Read Board Status replies sections 2.3.1, 2.3.2 and command summary tables.
Correct labeling of TDIG Switch Bits Table. Expand and correct description of switch inputs. Correct bit-names in ECSR status return.
Document analog input (temperature) from TINO.

Revision History section added.
Section 1.2.1.4: Block-Disposition – corrected length and assignment of byte 5.
Section 2.6.2: FPGA Reset – defined condition of internal registers following reset.
Read Board Serial Number – message and response defined.
Read Board Switch/Jumper Inputs – message and response defined.
Read Board Extended Control/Status register (ECSR) – message and response defined.
Read Board Temperature – message and response defined.
Read Firmware ID – message and response defined.
PLD_CONFIGURE – payload code reassigned.
Section 2.5.1 – examples added.
TDIG Numbering Scheme Changed.
TCPU NodeID scheme changed.
MCU Restart command defined.
Reconfiguration timeout status defined.
TCPU ECSR Bits Table added

TDIG ECSR Bits Table added.