Cuprite Programmer's Manual V1.0

An RF tester for the 802.15.4 O-QPSK standard operating in the 2.4GHz band, compatible with TLF3000 vector signal analyser and vector signal generator.

21 March 2018

1 Contents

1 Contents 2
2 Overview 8
3 Modes of Operation 9
3.1 Overview 9
3.2 Signal Generator 9
3.2.1 Wanted signal 9
3.2.2 Interfering signal 10
3.2.3 In-band CW signal 10
3.2.4 AWGN source 11
3.2.5 Out-of-band CW signal 11
3.3 Signal Analyser Mode 12
3.3.1 Overview 12
3.3.2 Capture 12
3.3.3 Measurements 12
3.3.4 Data capture 13
3.4 Packet Sniffer 14
3.4.1 Capture 14
3.4.2 Meta data 14
4 Host Interface 15
4.1 General 15
4.1.1 Supported interfaces 15
4.1.2 High-speed USB interface 15
4.1.3 Gigabit Ethernet interface 15
4.2 Asynchronous Data Interface 16
4.2.1 Overview 16
4.2.2 Header 17
4.2.3 Framing 17
4.2.4 Demodulated packet data message payload 18
4.2.5 Spectrum data 19
4.2.6 Test limit failure message payload 20
4.2.7 Environmental message payload 21
4.2.8 Input/Output power message payload 23
4.3 Command Interface 24

4.3.1 Overview....24
4.3.2 Command Packet Format....24
4.3.3 Command Response Packet Format....25
4.4 Command Summary....26
4.5 Supported Commands....28
4.5.1 Set Mode....28
4.5.2 Get Mode....29
4.5.3 Start Command....30
4.5.4 Pause Command....31
4.5.5 Resume Command....32
4.5.6 Stop Command....33
4.5.7 Set HWctrl Command....34
4.5.8 Get HWctrl....36
4.5.9 Set Wanted Signal....38
4.5.10 Get Wanted Signal....41
4.5.11 Set Interferer Signal....44
4.5.12 Get Interferer Signal....46
4.5.13 Set In-band CW....47
4.5.14 Get In-band CW....49
4.5.15 Set AWGN....50
4.5.16 Get AWGN....51
4.5.17 Set Out-of-band CW....52
4.5.18 Get Out-of-band CW....53
4.5.19 Set SigAna....54
4.5.20 Get SigAna....57
4.5.21 Set Test Limits....60
4.5.22 Get Test Limits....61
4.5.23 Poll Results Table....62
4.5.24 Poll Results Plot....65
4.5.24.1 Plotting basic quantities against channel or packet length....65
4.5.24.2 Obtaining histograms of basic quantities....68
4.5.24.3 Obtaining current value and plot vs time....71
4.5.24.4 Obtaining power spectral density plots....74
4.5.24.5 Obtaining constellation plots....76
4.5.24.6 Obtaining IQ samples....79

4.5.24.7 Obtaining FM deviation data....82
4.5.25 Poll Packet....85
4.5.26 Clear Results....86
4.5.27 Set Sniffer....87
4.5.28 Get Sniffer....88
4.5.29 Set Receiver Attenuation....89
4.5.30 Get Receiver Attenuation....90
4.5.31 Set Rx Port....91
4.5.32 Get Rx Port....92
4.5.33 Set DIO Volts....93
4.5.34 Get DIO Volts....94
4.5.35 Set PeriodicMessage Mask....95
4.5.36 Get Periodic Message Mask....96
4.5.37 Set Input/Output Power Period....97
4.5.38 Get Input/Output Power Period....98
4.5.39 Set Environmental Period....99
4.5.40 Get Environmental Period....100
4.5.41 Get Time....101
4.5.42 Get Features....102
4.5.43 Get Friendly Name....103
4.5.44 Get Serial Number....104
4.5.45 Get Manufacturer String....105
4.5.46 Get IP Address....106
4.5.47 Shutdown....107
4.5.48 Exit....108
4.5.49 Reset....109
4.6 Error Handling....110
4.6.1 Internal Error Response....110
4.6.2 Error Response....112
4.6.3 Reading the error queue....113
4.6.4 Clearing the error queue....114
5 Python Interface....115
5.1 Overview....115
5.1.1 Prerequistes....115
5.1.2 Connecting to the TLF3000....115

5.1.3 Launching the Cuprite application....117
5.1.4 Handling asynchronouos data....117
5.1.5 Handling errors....118
5.1.6 Closing down....118
5.2 Examples....119
5.2.1 Programming the signal generator....119
5.2.2 Analysing waveforms and obtaining data using the signal analyser....126
5.2.3 Running the packet sniffer....136
5.3 Library reference....142
5.3.1 Overview....142
5.3.2 setMode....142
5.3.3 getMode....142
5.3.4 startExec....142
5.3.5 PauseExec....143
5.3.6 resumeExec....143
5.3.7 stopExec....143
5.3.8 setHWCtrl....143
5.3.9 setWanted....145
5.3.10 setInterferer....147
5.3.11 setInBandCW....147
5.3.12 setAWGN....148
5.3.13 setOutOfBandCW....148
5.3.14 setSigAna....149
5.3.15 getTable....151
5.3.16 plotvsChan....151
5.3.17 plotvsPktLen....152
5.3.18 plotHist....153
5.3.19 plotPower....155
5.3.20 plotCarrier....156
5.3.21 plotEVM....156
5.3.22 plotPhi....157
5.3.23 plotPSD....157
5.3.24 plotConstellation....158
5.3.25 plotIQ....158
5.3.26 plotFM....159

5.3.27 getBits 159
5.3.28 clearResults 159
5.3.29 setLimits 160
5.3.30 getLimits 160
5.3.31 setSniffer 161
5.3.32 setRxAtten 161
5.3.33 getRxAtten 161
5.3.34 setRxPort 161
5.3.35 getRxPort 162
5.3.36 setExtDioVolts 162
5.3.37 getExtDioVolts 163
5.3.38 getError 163
5.3.39 exitApp 163
5.3.40 hardwareReset 163
5.3.41 powerDown 163
5.3.42 getFriendlyName 163
5.3.43 getSerialNumber 164
6 Connectors 165
6.1 Front Panel 165
6.1.1 Monitor In Port 165
6.1.2 Tx/Rx port 165
6.1.3 Receiver Specification 165
6.1.4 Video port 166
6.1.5 External Clock Input 166
6.1.6 Reference Clock Output 166
6.2 Rear Panel 167
6.2.1 Digital IO 167
6.2.2 USB Connector 167
6.2.3 Ethernet Connector 168
6.2.4 Power Connector 168
6.3 Connections for Cuprite Operating Modes 168
7 Uncertainty 169
8 Indicators 170
8.1 Front Panel 170
8.1.1 Status 170

8.1.2 RF Overload 170
8.1.3 Flash Update 170

2 Overview.

TLF3000 is a wideband, ultra-high dynamic range 2.4 GHz software-defined radio, signal analyser and signal generator. It captures and analyses the entire 2402-2480 MHz band simultaneously. It can also generate arbitrary waveforms occupying the band 2395-2485 MHz with a maximum peak level of 0 dBm. In addition, it includes a CW signal generator covering 25 MHz-6GHz with an output level of -50 dBm to -28 dBm

Cuprite is an 802.15.4 O-QPSK application for the *TLF3000* software-defined receiver, signal analyzer and signal generator. The Cuprite application can:

1. Act as a signal generator, creating signals required for receiver testing including interfering and blocking signals. A variety of pulsed and continuous signals can be generated for measuring the receiver ED, and AWGN may be added for LQI measurements.
2. Act as a signal analyser, performing transmitter tests on conducted or off-air signals without knowledge of the payload or RF channel. Tests include EVM, power spectral density, carrier offset, carrier drift, output power as well as symbol timing accuracy. The output of the carrier tracking loop is also available.
3. Monitor all 16 channels and either pipe the decoded packets directly into Wireshark for live protocol analysis or dump the packets to file. Data capture can be restricted to certain channels or by an RSSI threshold,

The application has been honed for speed. For example, EVM and spectral measurements are performed in parallel, exploiting the unique architecture of the *TLF3000*.

A key feature of the unit is its ability to perform C/I, receiver selectivity and intermodulation tests without the need for additional test equipment. This is possible due to *TLF3000's* ultra-linear wideband signal generator. This permits both wanted and interfering signals to be generated through the same signal path. The high linearity and low noise floor ensure that there is ample dynamic range to encompass both the wanted and interfering signals. Furthermore, high fidelity filtering of the interfering signals ensures that they are correctly bandlimited and that unwanted sidebands are not responsible for test failures; this is frequently overlooked when external test equipment is used to provide these signals. The single signal path also removes the need for time consuming and laborious calibration of signal combiners as well as eliminating the need to ensure that the injected interfering and wanted signals do not generate intermodulation products before arriving at the DUT.

Unique to *TLF3000* is a 25MHz to 6GHz signal generator. This enables the majority of the receiver blocking performance to be explored prior to committing the DUT for formal inspection at a test house with its associated costs.

3 Modes of Operation.

3.1 Overview

Cuprite has three operating modes:

1. Signal generator.
2. Signal analyser.
3. Packet sniffer

3.2 Signal Generator

The signal generator mode permits manual control over all the signal sources used for receiver testing. Any combination of the following signals can be generated simultaneously:

1. Packetized 802.15.4 O-QPSK signal with programmable contents.
2. Continuously modulated or packetised O-QPSK signal.
3. Up to 3 independent in-band CW signals, which may be pulsed.
4. In-band AWGN source, which may be pulsed.
5. Out-of-band CW signal..

All signal sources are fully parameterised.

3.2.1 Wanted signal

A packetised wanted signal to be received by the DUT can be generated. This is parameterized by:

1. Channel number in the range 11 to 26
2. Amplitude in the range -120dBm to 0dBm with a resolution of 0.1dBm
3. Packet payload
4. Payload length in the range 0 to 127 octets
5. Packet interval up to 500ms
6. Number of packets to be transmitted in the range 1 to 65535, or continuous
7. Inter-packet jitter timing magnitude
8. Inter-packet jitter timing distribution as either uniform or Gaussian
9. Carrier frequency offset of ±250kHz
10. Carrier drift magnitude of up to 70kHz
11. Carrier drift rate of up to 2440Hz
12. IQ amplitude imbalance of up to 3dB
13. IQ quadrature imbalance of up to 10°
14. Packet power ramp up/down times from 1.1µs to 10µs
15. Unmodulated carrier after power ramp up and prior to start of packet modulation (‘front porch’) of 0µs to 25µs
16. Unmodulated carrier after end of packet modulation to start of power ramp down (‘back porch’) of 0µs to 25µs
17. A chip error rate imposed on the modulated data of up to 6.25%
18. Digital output lines to be toggled when a wanted packet is transmitted

The drift follows a sinusoidal waveform whose period is given by the drift rate and whose amplitude is determined by the drift magnitude. The drift is always zero at the start of a packet.

An important feature of the wanted signal generator is its ability to create signals with amplitudes as low as -120 dBm.

If any of the wanted signal parameters are changed whilst the wanted signal is being transmitted, then the current transmission is aborted and the wanted signal plus any interfering signals restarted.

3.2.2 Interfering signal

An interfering signal with PRBS15 payload to be used in C/I and intermodulation tests. This signal is parameterised by:

1. Frequency in the range 2395MHz to 2485MHz in steps of 1Hz
2. Amplitude in the range -120dBm to 0dBm with a resolution of 0.1 dBm
3. The interfering signal may be continuous or packetised
4. If the signal is packetised, then the payload length can be specified in the range 0 to 127 octets
5. If the signal is packetised, then the interval between the start of packets may be specified up to 500ms
6. Digital output lines to be toggled when this signal is active

If any of the interfering signal parameters are changed whilst the interferer signal is active, then the current transmission is aborted and all wanted and interfering signals restarted.

3.2.3 In-band CW signal

Cuprite can generate up to 3 in-band (i.e. 2395MHz to 2485MHz) CW interferer signals. These signals are required to perform receiver intermodulation tests. The CW signals may also be used to characterise the ED function of a receiver.

The signal is parameterised by:

1. Frequency in the range 2395MHz to 2485MHz in steps of 1Hz
2. Amplitude in the range -120dBm to 0dBm with a resolution of 0.1 dBm
3. The signal may be continuous or pulsed
4. If the signal is pulsed, then the on and off periods may be individually specified in the range 0.1µs to 1.6s

The commencement and cessation of each pulse is abrupt; there is no power ramp up or down. As a consequence, power may be splattered across the band at the start and end of each burst.

If any of the in-band CW signal parameters are changed whilst the in-band CW signal is active, then the current transmission is aborted and all wanted and interfering signals restarted.

3.2.4 AWGN source

The AWGN source provides uniform output power from 2395MHz to 2485MHz. This source can be used to characterise the ED and LQI functions of a receiver. The AWGN source can be continuous or pulsed. It is parameterised by:

1. Amplitude in the range -162.1dBm/MHz to -39.1dBm/MHz
2. The signal may be continuous or pulsed
3. If the signal is pulsed, then the on and off periods may be individually specified in the range 0.1µs to 1.6s

The commencement and cessation of each pulse is abrupt; there is no power ramp up or down. As a consequence, power may be splattered across the band at the start and end of each burst.

If any of the AWGN source parameters are changed whilst the AWGN source is active, then the current transmission is aborted and all wanted and interfering signals restarted.

3.2.5 Out-of-band CW signal

An out-of-band CW signal to be used in receiver blocking tests. This signal can typically be generated at a level up to -25 dBm. The frequency of the signal can be in the range 25MHz to 6GHz with a resolution of 1 MHz.

The CW generator within the *TLF3000* unit possesses significant harmonic content. When a harmonic lies within the 2.4 GHz ISM band, it is possible for the DUT to be blocked by the harmonic and not the fundamental. This issue is not unique to the *TLF3000*; harmonic content is also an issue when using external signal generators to provide the blocking signal. For example, an R&S SMW200A specifies harmonic content as < -30 dBc. If this unit were to generate a blocking signal at 1201MHz with a level of -30dBm, then it may generate a second harmonic at 2402MHz with a level of -60dBm. Given that the DUT is usually being tested at a sensitivity level of better then -60dBm, it is clear that the DUT will be blocked by the harmonic and not the fundamental.

To help alleviate this problem, the *TLF3000* contains a 2.4 GHz notch filter in the out-of-band CW signal path. This notch filter is switched in whenever the blocking signal is below 1500 MHz. Hence the harmonic content of the blocker within the 2.4 GHz ISM band is greatly attenuated. However, when testing sensitive receivers it should always be borne in mind that any blocking failures could be due to harmonics of the blocking signal and not the fundamental.

The out-of-band CW signal can be modified at any time without affecting the wanted or in-band interfering signals

3.3 Signal Analyser Mode

3.3.1 Overview

The signal analyser mode monitors all 16 channels simultaneously. Any detected packets are captured and analysed in accordance with the 802.15.4 specification. Tests include EVM, power spectral density, carrier offset, carrier drift, output power as well as symbol timing accuracy.

3.3.2 Capture

The signal analyser can receive packets on one of two ports:

1. Tx/Rx port. This port has a noise figure of +46dB and can handle input signals as high as +27dBm. Use this port for conducted measurements.
2. Monitor In port. This port has a noise figure of +6dB and a maximum usable input signal level of -10dBm. Use this port for off-air measurements.

An RSSI threshold can be set for packets to be analysed. This permits weak packets from distant interfering devices to be excluded from the analysis.

Filters can be specified so that only packets on certain RF channels or with certain packet lengths are captured and analysed.

3.3.3 Measurements

The signal analyser is capable of making the following measurements:

Measurement	Decsription	Limit
P_{avg}	The power averaged over the entire packet	✓
$Pk - P_{avg}$	The peak power in the packet minus the average power over the entire packet	
Initial F_c	The carrier frequency at the start of the packet	✓
Min F_c	The minimum carrier frequency throughout the packet	✓
Max F_c	The maximum carrier frequency throughout the packet	✓
Drift	The maximum drift rate of the carrier frequency throughout the packet	
Symbol Timing	The symbol timing error averaged over packets	✓
EVM_{1000}	The maximum EVM averaged over 1000 chip intervals within the packet	✓
EVM_{rms}	The rms EVM averaged over the entire packet	
EVM_{pk}	The peak EVM within the packet	
$\Delta\varphi_{rms}$	The rms constellation phase error averaged over the entire packet	
$\Delta\varphi_{pk}$	The peak constellation phase error within the packet	
IQ Imbalance	The average IQ amplitude imbalance within the packet	
PSD_{abs}	The absolute power spectral density as per the 802.15.4 specification	✓
PSD_{rel}	The relative power spectral density as per the 802.15.4 specification	✓

For those measurements which are detailed in the 802.15.4 specification, a limit can be set. It is possible to cease data analysis when any limit is exceeded. In this instance, the measurements and waveforms relating to the failing packet can be retrieved from the Cuprite application.

3.3.4 Data capture

Besides the measurements listed above, the Cuprite application is capable of capturing the following data for further analysis:

1. Average power spectral density. This is generated using a Gaussian filter with a 100kHz RBW. The data is available with a resolution of approximately 50kHz. Different power spectral desnity measurements are maintained for packets received on each of the 16 RF channels. The spectrum averaging can be configured from 10µs to 6hrs.
2. Peak power spectral density. This is generated using a Gaussian filter with a 100kHz RBW. The data is available with a resolution of approximately 50kHz. Different power spectral desnity measurements are maintained for packets received on each of the 16 RF channels. The spectrum peak hold can be configured from 10µs to 6hrs. The longer intervals are useful for capturing infrequent events.
3. Constellation data. Offset constellation data for every chip within the packet are available.
4. Power profile. The received power as a function of time can be obtained. The time resolution can be selected from 50ns, 100ns, 200ns or 250ns resolution. The power profile starts approximately 50µs before the preamble and finishes approximately 50µs after the FCS. This is generally sufficient to include the power ramp up and down.
5. IQ samples. Raw IQ samples can be read back with a time resolution of 50ns, 100ns, 200ns or 250ns. The samples start approximately 50µs before the preamble and finish approximately 50µs after the FCS.
6. FM deviation. The FM deviation of the signal can read back with a time resolution of 50ns, 100ns, 200ns or 250ns. The FM deviation data start approximately 50µs before the preamble and finish approximately 50µs after the FCS. This is useful if the user wants an MSK interpretation of the modulation.
7. Carrier tracking loop. The instantaneous estimate of the carrier frequency as obtained by the carrier frequency tracking loop can be read back. This is sampled at each constellation point.
8. EVM. The EVM data for each constellation point within the packet is available.
9. $\Delta\phi$. The Phase error for each constellation point within the packet is available.
10. The decoded bitstream for the packet.

3.4 Packet Sniffer

The packet sniffer mode monitors all 16 channels simultaneously. Any packets which are detected are demodulated.

3.4.1 Capture

The packet sniffer can receive packets on one of two ports:

1. Tx/Rx port. This port has a noise figure of +46dB and can handle input signals as high as +27dBm. Use this port for conducted measurements.
2. Monitor In port. This port has a noise figure of +6dB and a maximum usable input signal level of -10dBm. Use this port for off-air measurements.

It is important that the receiver frontend attenuator is adjusted to ensure that the receiver does not overload during packet capture.

An RSSI threshold can be set for packets to be analysed. This permits weak packets from distant interfering devices to be excluded.

A synchroniser threshold can also be set. This provides a trade-off between sensitivity and false detections. Decreasing the threshold will increase sensitivity but may result in false detections.

A channel filter can be set so that only packets on specified RF channels are sniffed.

3.4.2 Meta data

Each packet has a link quality indicator associated with it. The LQI is based on the correlator output averaged over the entire packet. The scaling of the LQI has been adjusted to make it comparable with the LQI reported by TI devices.

The FCS for each packet received is calculated and compared against the received FCS. This calculation assumes a 2 octet FCS.

An RSSI is assigned to each packet. The RSSI is calculated over the preamable portion of the packet.

Each packet is timestamped. The resolution of the timestamp is 100ns.

4 Host Interface.

4.1 General

4.1.1 Supported interfaces

Cuprite supports the *TLF3000* native command set on either high-speed USB or gigabit Ethernet. It does not support the SCPI command set.

4.1.2 High-speed USB interface

Communications between Cuprite and the host may take place over a high speed USB link. Cuprite presents itself to the host as a USB gadget.

Three end-points are supported by Cuprite:

End point	Direction	Purpose
1	Cuprite to Host	Asynchronous Data
2	Host to Cuprite	Command
3	Cuprite to Host	Command Response

The Asynchronous Data end-point is used by Cuprite to transfer to the host:

1. Limit failure notifications when in signal analyser mode
2. Demodulated packets when in packet sniffer mode
3. Spectrum data when in packet sniffer mode
4. Periodic informational messages

There is no handshaking on this data transfer. The Asynchronous Data end-point will only stream data if the Cuprite application was launched via a host command on the USB interface. If the application was launched via a host command on the Ethernet interface, no data will be present on the Asynchronous Data end-point.

The Command and Command Response end-points are used by the host to:

1. Control the operation of Cuprite
2. Poll for test results when in signal analyser mode

Every command issued by the host on the Command end-point results in a corresponding command response from Cuprite on the Command Response end-point. Commands may be issued over USB and Ethernet simultaneously, irrespective of which interface was used to launch the application.

4.1.3 Gigabit Ethernet interface

Communications between Cuprite and the host may take place over Gigabit Ethernet using TCP/IP. The communications utilise the following ports:

Interface	Port number
Native Command Pipe	59361
Native Asynchronous Data Pipe	59362

The Native Asynchronous Data Pipe is used by Cuprite to transfer to the host:

1. Limit failure notifications when in signal analyser mode
2. Demodulated packets when in packet sniffer mode
3. Spectrum data when in packet sniffer mode
4. Periodic informational messages

There is no handshaking on this data transfer. Data will only be streamed if the Cuprite application was launched via a host command on the Ethernet interface. If the application was launched via a host command on the USB interface, no data will be present on the native Asynchronous Data pipe.

The Native Command Pipe is used by the host to:

1. Control the operation of Cuprite
2. Poll for test results when in signal analyser mode

Every command issued by the host to on the Native Command Pipe results in a corresponding command response from Cuprite on the same pipe. Commands may be issued over USB and Ethernet simultaneously, irrespective of which interface was used to launch the application.

4.2 Asynchronous Data Interface

4.2.1 Overview

Test failures, demodulated packets, spectrum data and informational messages are streamed from Cuprite to the host over the Asynchronous Data end-point (USB) or Native Asynchronous Data pipe (Ethernet). This is a one-way pipe with no acknowledgement from the host. The command interface can be used by the host to control which messages are sent and how frequently they are sent.

The following messages can be sent over the data interface:

1. Signal analyser test limit failure
2. Packet sniffer demodulated packet data
3. Packet sniffer spectrum data
4. Environmental monitoring messages
5. Input/output power message
6. Debug messages

All these messages have the same generic format. The messages are composed of a 4 byte header, a variable length payload and a 4 byte framing word.

4.2.2 Header

The header is composed of 2 fields:

Header

Message Length	Message Type
uint24_t	uint8_t

The *Message Length* field contains the length of the message in a unit24_t, excluding the header and the framing word. Hence the total number of bytes in the message is:

4 x (Message Length + 2)

Note: for the Asynchronous Data interface the header length field is in units of 4 bytes; for the command interface the header length field is in units of bytes.

The *Message Type* field denotes the purpose of the message and hence the format of the payload. The possible message types are:

Message Type	Payload Contents	Applicable Mode
0	Demodulated packet data	Packet sniffer
5	Spectrum data	Packet sniffer
64	Test limit failure	Signal analyser
193	Environmental message	All
194	Input/output power message	All
255	Debug data	All

4.2.3 Framing

The message is terminated by a 4 byte framing word. The framing word always has the value 0x55AA55AA.

Framing

0x55AA55AA
uint32_t

4.2.4 Demodulated packet data message payload

The *Demodulated packet data* message is issued each time the Cuprite application receives a new 802.15.4 packet when in packet sniffer mode. The messages contains the demodulated data plus various meta data.

Timestamp	Chan	RSSI	Count	Data	Data	…	Pad	CRC/LQI
uint64_t	uint8_t	int8_t	uint8_t	uint8_t	uint8_t		uint8_t	uint8_t

The *Pad* field may be present 0 to 3 times. Its purpose is to cause the CRC/LQI field to be aligned with a 32 bit boundary. The *Pad* fields contain no useful information.

The meaning of the fields within the payload are:

1. *Timestamp* field is a uint64_t containing the *TLF3000* system time associated with the message. The resolution of the timestamp is 100ns. Provided the *TLF3000* system time has been correctly set using the *Set Time* host command, then *Timestamp* represents the time since the Unix epoch of 1970-01-01 00:00:00 + 0000 (UTC) in units of 100ns. Hence the timestamp is an absolute time containing full details of the date and time at which the message was generated.
2. *Chan* field is a uint8_t containing the channel number on which the packet was received. This will be in the range 11 to 26.
3. *RSSI* field is an int8_t containing the estimated signal strength of the packet in units of dBm. The power is estimated from the packet preamble.
4. *Count* field is a uint8_t containing the PHR field. To obtain the actual number of octets in the phy payload the top bit must be masked off.
5. *Data* fields are the octets of the phy payload.
6. *CRC/LQI* field is composed of two parts. The MSB indicates whether the FCS of the MAC frame is valid. A 2 byte FCS is assumed. The lower 7 bits contain a link quality indicator. The LQI has been scaled so that it is roughly comparable with the LQI returned by TI devices.

4.2.5 Spectrum data

Timestamp	Unused	Data 0	Data 1	Data 2	Data 3		Data 1843
uint64_t	uint32_t	int16_t	int16_t	int16_t	int16_t		int16_t

The *Spectrum Data* message is issued periodically when the Cuprite application is in packet sniffer mode. These messages are sent every 84ms. The data contained within them represents the peak spectral power observed over the previous 84ms. The spectrum is based on a Gaussian passband filter with a 3dB bandwidth of 100kHz. The spectrum is oversampled by a factor of 2.

The meaning of the fields within the payload are:

1. *Timestamp* field is a uint64_t containing the *TLF3000* system time associated with the message. The resolution of the timestamp is 100ns. Provided the *TLF3000* system time has been correctly set using the *Set Time* host command, then *Timestamp* represents the time since the Unix epoch of 1970-01-01 00:00:00 + 0000 (UTC) in units of 100ns. Hence the timestamp is an absolute time containing full details of the date and time at which the message was generated.
2. *Data N* fields contain the spectral data in units of 1/16th dBm. There are 1844 data fields. Each field represents the spectral power on an adjacent frequency bin. The frequency bins are separated by 48.828kHz. The first data field corresponds to the power at 2394.980MHz.

4.2.6 Test limit failure message payload

The *Test limit failure* message is issued when a test limit failure is detected and the Cuprite application is in signal analyser mode with stop on limit failure set. The message signifies that signal analysis has ceased. It also returns an indication of which test failed.

The *Limit Failed* message payload is of the following format:

TestNum
uint32_t

The *TestNum* field indicates the nature of the last transmitter test to fail. Possible values are:

Value	Meaning
0	Output power below lower limit
1	Output power above upper limit
2	Carrier frequency exceeds limits
3	Symbol timing error exceeds limits
4	EVMrms exceeds limit
5	Absolute power spectral density exceeds limits
6	Relative power spectral density exceeds limits

4.2.7 Environmental message payload

Environmental Messages are generated periodically. The period can be adjusted using the *Set Environmental Period* host command. These messages are only transferred to the host if bit 11 in the capture mask has been set using a *Set Periodic Mask* host command.

Timestamp	Fan	Flags	Now	Min	Max	...	Err
uint64_t	uint16_t	uint16_t	int16_t	int16_t	int16_t	{Now,Min,Max} x 6	uint16_t

Rdgs[0]	...	Rdgs[7]	DacTemp	DacFlags
uint16_t	uint16_t	uint16_t	int16_t	uint16_t

The *Timestamp* field is a uint64_t containing the *TLF3000* system time associated with the message. The resolution of the timestamp is 100ns. Provided the *TLF3000* system time has been correctly set using the *Set Time* host command, then *Timestamp* represents the time since the Unix epoch of 1970-01-01 00:00:00 + 0000 (UTC) in units of 100ns. Hence the timestamp is an absolute time containing full details of the date and time at which the message was generated.

To recover date information from the timestamp, a C program could use functionality similar to:

```
time_t          tmsg;
struct          *tm;

tmsg = (time_t) ( TimeStamp / 10000000 );

tm = gmtime( &tmsg );
```

Note: the *Set Time* host command must be issued to *TLF3000* prior to the Cuprite application being downloaded and executed. It is not possible to issue the *Set Time* host command once the Cuprite application is running.

The *Fan* field is a uint16_t containing the current fan speed in rpm. A value of 0 indicates that the fan is not running. The fan should always be running when the Cuprite application is active.

The *Flags* field indicates the lock state of various PLLs and also whether an internal error has been detected. If an internal error has been detected, then the reason can be found in the *Err* field. The bits in this field are numbered from zero and their meanings are:

Bit	Mask	Meaning
0	0x0001	'1' indicates the clock generator PLL is locked, '0' indicates that the PLL is unlocked
1	0x0002	'1' indicates the RF synthesiser PLL is locked, '0' indicates that the PLL is unlocked
2	0x0004	'1' indicates that an internal error has been detected, '0' indicates no error detected
3	0x0008	'1' indicates that the ADC clock is present, '0' indicates no ADC clock
4	0x0010	'1' indicates the FPGA PLL is locked, '0' indicates that the PLL is unlocked
5	0x0020	'1' indicates loss of external clock, '0' indicates external clock present (this flag is only valid when bit 6 is set)
6	0x0040	'1' indicates locked to external clock, '0' indicates locked to internal TCXO
7	0x0080	'1' indicates that the DAC FIFO has errored, '0' indicates the DAC FIFO is good (this flag is only valid if the application is generating a signal)
8	0x0100	'1' indicates 1.0 V digital supply is good, '0' indicates the supply is bad
9	0x0200	'1' indicates 1.5 V digital supply is good, '0' indicates the supply is bad
10	0x0400	'1' indicates 1.8 V digital supply is good, '0' indicates the supply is bad
11	0x0800	'1' indicates 3.3 V analogue supply is good, '0' indicates the supply is bad
12	0x1000	'1' indicates 3.8 V analogue supply is good, '0' indicates the supply is bad
13	0x2000	'1' indicates 5.0 V analogue supply is good, '0' indicates the supply is bad
14	0x4000	'1' indicates 5.0 V fan supply is good, '0' indicates the supply is bad

There are 7 triplets of the *{Now, Min, Max}* fields. Each of these fields is an int16_t. The *Now* field represents the current value of a temperature or voltage, the *Min* field shows the minimum value observed since the application was launched and the *Max* field the maximum value observed since the application was launched. The triplets correspond to the following temperatures and voltages:

Triplet	Quantity	Units
0	Temperature	C
1	Vcc_aux	mV
2	Vcc_bram	mV
3	Vcc_int	mV
4	Vcc_oddr	mV
5	Vcc_paux	mV
6	Vcc_pint	mV

The *Err* field is a uint16_t containing an error code for the last reported internal error.

The *Rdgs* fields report the results of the FPGA internal ADC sampling the 8 multiplexed analogue lines. These contain information on the temperature of various devices on the PCB.

The *DacTemp* field is an int16_t containing the signal generator DAC temperature in degrees Celsius. This field is only valid if the signal generator is active.

The *DacFlags* field is a uint16_t containing the status flags read back from the signal generator DAC. This field should be zero if the DAC is operating correctly. This field is only valid if the signal generator is active.

4.2.8 Input/Output power message payload

Input/OutputPower Messages are generated periodically. The period can be adjusted using the *Set Power Period* host command. These messages provide an estimate of the power being transmitted and received, as well as the settings of transmit and receive attenuators, overload conditions and the receiver input port. These messages are only transferred to the host if bit 10 in the capture mask has been set using a *Set Periodic Mask* host command

Timestamp	Output	Input	Overflow	TxAtten	RxAtten
uint16_t	int16_t	int16_t	uint16_t	uint8_t	uint8_t

The *Timestamp* field is a uint64_t containing the *TLF3000* system time associated with the message. The resolution of the timestamp is 100ns. Provided the *TLF3000* system time has been correctly set using the *Set Time* host command, then *Timestamp* represents the time since the Unix epoch of 1970-01-01 00:00:00 + 0000 (UTC) in units of 100ns. Hence the timestamp is an absolute time containing full details of the date and time at which the message was generated.

To recover date information from the timestamp, a C program could use functionality similar to:

```
time_t          tmsg;
struct          *tm;

tmsg = (time_t) ( TimeStamp / 10000000 );

tm = gmtime( &tmsg );
```

Note: the *Set Time* host command must be issued to *TLF3000* prior to the Cuprite application being downloaded and executed. It is not possible to issue the *Set Time* host command once the Cuprite application is running.

The *Output* field is an int16_t containing an estimate of the current signal generator output in units of 0.1dBm.

The *Input* field is an int16_t containing an estimate of the current received power on the selected receiver port in units of 0.1 dBm.

The *Overflow* field is a uint16_t which indicates various overload. The meaning bits within this field are:

Bit position	Bit Mask	Meaning
0	0x0001	'1' indicates a transmit overload, '0' indicates no transmit overload
1	0x0002	'1' indicates a receive overload, '0' indicates no receive overload
2	0x0004	'1' indicates an overload condition, '0' indicates no overload condition

The *TxAtten* field is a uint8_t which contains the current transmitter attenuator setting in units of 0.5dB. The transmitter attenuator can be set in the range 0 to 49.5 dB.

The *RxAtten* field is a uint8_t. The lowest 5 bits contain the current setting of the receiver frontend attenuator in units of 0.5 dB. The receiver frontend attenuation can be set in the range 0 to 31.5dB. Bit 7 of this field (MSB) is used to indicate which receiver port is in use by the receiver. If bit 7 is set, then the Tx/Rx is used by the receiver. If bit 7 is clear, then the *Monitor In* port is used by the receiver.

4.3 Command Interface

4.3.1 Overview

The command interface works by the host sending a command packet to Cuprite and Cuprite replying with a command response packet. Cuprite never sends unsolicited packets across the command interface.

Packets from the host to Cuprite are transferred using the Command end-point (USB) or Native Command Pipe (Ethernet). Packets from Cuprite to the host are transferred using the Command Response end-point (USB) or Native Command Pipe (Ethernet).

It is recommended that after issuing a command packet, the host waits for the command response packet from Cuprite prior to sending further command packets.

Commands may be sent over USB and Ethernet simultaneously. The command response will be sent over the same interface as the command was received.

4.3.2 Command Packet Format

Command packets from the host to *TLF3000* consist of a 4 byte header, a variable length payload plus 1 framing byte.

Command packet

Header	Payload	Frame
uint32_t	uint8_t [Len]	uint8_t

The header is composed of 3 fields:

Header

Cmd	Trans ID	Len
uint8_t	uint8_t	uint16_t

The *Cmd* field is a single uint8_t and determines the purpose of the packet.

The *Trans ID* field is a single uint8_t which is returned in the command response packet from *TLF3000* to the host. This permits the host to associate command response packets with their corresponding command packets.

The *Len* field is the length of the payload in bytes. This value does not include the length of the header or the framing byte.

Frame

0xAA
uint8_t

The *Frame* field is a single uint8_t and is always set to 0xAA.

4.3.3 Command Response Packet Format

Whenever the host sends a command packet to Cuprite, Cuprite replies with a command response packet. Command response packets from Cuprite to the host consist of a 4 byte header, a variable length payload plus 1 framing byte.

Command response packet

Header	Payload	Frame
uint32_t	uint8_t [Len]	uint8_t

The header is composed of 3 fields:

Header

Cmd	Trans ID	Len
uint8_t	uint8_t	uint16_t

The *Cmd* field is a uint8_t which will either contain:

1. A copy of the *Cmd* field taken from the corresponding command packet sent from the host to *TLF3000*.
2. The value 0x7F, corresponding to a failed command.
3. The value 0x7E, indicating that an internal error occurred prior to reception of the command.

The *Trans ID* is a uint8_t and will contain a copy of the *Trans ID* field taken from the corresponding command packet sent from the host to Cuprite. This permits the host to associate command response packets with their corresponding command packets.

The *Len* field is a uint16_t and is the length of the payload in bytes. This value does not include the length of the header or the framing byte.

Frame

0xAA
uint8_t

The *Frame* field is a uint8_t and is always set to 0xAA.

4.4 Command Summary

Command	Purpose	Sig Gen	Sig Ana	Pkt Sniff
Mode Switching Commands				
Set Mode Get Mode	Switches the Cuprite operating mode	✓	✓	✓
Start/Stop commands				
Start	Starts operation of selected mode	✓	✓	✓
Pause	Pauses operation of selected mode	✓	✓	✓
Resume	Resumes operation of selected mode	✓	✓	✓
Stop	Stops operation of selected mode	✓	✓	✓
HW control	Programs hardware control of start, pause, resume and stop	✓	✓	✓
Signal Generator Commands				
Set Wanted Signal Get Wanted Signal	Program the packetised wanted signal	✓		
Set Interferer Signal Get Interferer Signal	Program the modulated interferer	✓		
Set In-band CW Get In-band CW	Program the in-band CW sources	✓		
Set AWGN Get AWGN	Program the AWGN source	✓		
Set Out-of-band CW Get Out-of-band CW	Program the out-of-band CW source	✓		
Signal Analyzer Commands				
Set SigAna Get SigAna	Program the signal analyser Read the signal analyser programming		✓	
Set Test Limits Get Test Limits	Program the signal analyser test limits Read the signal analyser test limits		✓	
Poll Results Table	Read back statistics from signal analyser		✓	
Poll Results Plot	Captures trace data and returns length of capture		✓	
Get Results plot	Reads back captured trace data		✓	
Poll Packet	Reads back demodulated bit data		✓	
Clear Results	Clear the signal analyser results		✓	
Packet Sniffer Commands				
Set Sniffer Get Sniffer	Programs the packet sniffer Read back the packet sniffer programming			✓
Receiver Configuration Commands				
Set Receiver Attenuation Get Receiver Attenuation	Set the receiver frontend attenuation Read back receiver frontend attenuation	✓	✓	✓
Set Rx Port Get Rx Port	Set whether the Tx/Rx or Monitor In port should be used on receive	✓	✓	✓

Command	Purpose	Sig Gen	Sig Ana	Pkt Sniff
Digital IO Configuration Commands				
Set DIO Volts Get DIO Volts	Selects the source of the digital IO voltage reference	✓	✓	✓
Periodic Message Commands				
Set Periodic Message Mask Get Periodic Message Mask	Determines which periodic messages will be broadcast to the host	✓	✓	✓
Set Input/Output Power Period Get Input/Output Power Period	Sets the period of the input/output power periodic messages	✓	✓	✓
Set Environmental Period Get Environmental Period	Sets the period of the environmental periodic messages	✓	✓	✓
Utility Commands				
Get Time	Queries the application time	✓	✓	✓
Get Friendly Name	Queries the TLF3000 friendly name	✓	✓	✓
Get Features	Queries the application version	✓	✓	✓
Get Serial Number	Queries the TLF3000 serial number	✓	✓	✓
Get Manufacturing String	Queries manufacturing information	✓	✓	✓
Get IP Address	Queries network information	✓	✓	✓
Termination Commands				
Shutdown	Powers down TLF3000	✓	✓	✓
Exit	Exit the Cuprite application	✓	✓	✓
Reset	Reboot the TLF3000	✓	✓	✓
Error Reporting Commands				
Read Error Queue	Read oldest error message	✓	✓	✓
Clear Error Queue	Clear messages on error queue	✓	✓	✓

Table 1: Command summary

4.5 Supported Commands

4.5.1 Set Mode

The *Set Mode* command determines the operating mode of the Cuprite application.

The payload consists of a single uint8_t which defines the operating mode:

Value	Operating mode
1	Signal generator
2	Signal analyser
4	Packet sniffer
5	Traffic generator (not yet supported)
Other	Illegal

Set Mode Command

Cmd	Trans ID	Len	Mode	Frame
0x12	uint8_t	0x0001	uint8_t	0xAA

The Set Mode response payload contains a single uint8_t indicating the new operating mode.

Set Mode Response

Cmd	Trans ID	Len	Mode	Frame
0x12	uint8_t	0x0001	uint8_t	0xAA

4.5.2 Get Mode

The *Get Mode* host command returns the current operating mode of the Cuprite application. The command has no payload.

Get Mode Command

Cmd	Trans ID	Len	Frame
0x92	uint8_t	0x0000	0xAA

The *Get Mode* command response has a single uint8_t in its payload which signifies the current operating mode:

Value	Mode
1	Signal generator
2	Signal analyser
4	Packet sniffer

Get Mode Response

Cmd	Trans ID	Len	Mode	Frame
0x92	uint8_t	0x0001	uint8_t	0xAA

4.5.3 Start Command

The *Start* command instructs the Cuprite application to commence executing the current mode, as selected by the *Set Mode* command.

The *Start* command has no payload.

Start Command

Cmd	Trans ID	Len	Frame
0x02	uint8_t	0x0000	0xAA

The *Start* command response is a copy of the *Start* command:

Start Command Response

Cmd	Trans ID	Len	Frame
0x02	uint8_t	0x0000	0xAA

4.5.4 Pause Command

The *Pause* command instructs the Cuprite application to pause execution of the current mode, as selected by the *Set Mode* command. This command should only be issued after a previous *Start* or *Resume* command.

The *Pause* command has no payload.

Pause Command

Cmd	Trans ID	Len	Frame
0x03	uint8_t	0x0000	0xAA

The *Pause* command response is a copy of the *Pause* command:

Pause Command Response

Cmd	Trans ID	Len	Frame
0x03	uint8_t	0x0000	0xAA

4.5.5 Resume Command

The *Resume* command instructs the Cuprite application to resume execution in the current mode, as selected by the *Set Mode* command. This command should only be issued after a *Pause* command.

The *Resume* command has no payload.

Resume Command

Cmd	Trans ID	Len	Frame
0x04	uint8_t	0x0000	0xAA

The *Resume* command response is a copy of the *Resume* command:

Resume Command Response

Cmd	Trans ID	Len	Frame
0x04	uint8_t	0x0000	0xAA

4.5.6 Stop Command

The *Stop* command instructs the Cuprite application to stop executing the current mode, as selected by the *Set Mode* command.

The *Stop* command has no payload.

StopCommand

Cmd	Trans ID	Len	Frame
0x05	uint8_t	0x0000	0xAA

The *Stop* command response is a copy of the *Stop* command:

Stop Command Response

Cmd	Trans ID	Len	Frame
0x05	uint8_t	0x0000	0xAA

4.5.7 Set HWctrl Command

The *Set HWctrl* command instructs the Cuprite application to start, pause, resume and stop execution of the currently selected mode under hardware control, ie by monitoring signals on the digital interface.

The payload of the *Set HWctrl* command has 5 fields:

1. *uint32_t : HWctrl.* Determines whether the application is running under hardware control. If this field is non-zero then the application is running under hardware control.

2. *uint32_t : Start.* A bit mask which defines the transitions required on the digital input lines to start executing the currently selected mode. This mask is ignored if *HWctrl* is zero.

Bits	Function
7:0	Defines which input lines are used to determine when execution should start. A '1' indicates that the corresponding input line is sampled. A '0' indicates that the corresponding input line is ignored.
15:8	Defines the state of the input lines prior to the start execution transition. A '1' indicates that the corresponding input line should be high, a '0' indicates that the corresponding input line should be low. Values for lines which are not included in the 7:0 mask are ignored.
23:16	Defines the state of the input lines after the start execution transition. A '1' indicates that the corresponding input line should be high, a '0' indicates that the corresponding input line should be low. Values for lines which are not included in the 7:0 mask are ignored.

3. *uint32_t : Pause.* A bit mask which defines the transitions required on the digital input lines in order to pause execution. This mask is ignored if *HWctrl* is zero.

Bits	Function
7:0	Defines which input lines are used to determine when execution should be paused. A '1' indicates that the corresponding input line is sampled. A '0' indicates that the corresponding input line is ignored.
15:8	Defines the state of the input lines prior to the pause execution transition. A '1' indicates that the corresponding input line should be high, a '0' indicates that the corresponding input line should be low. Values for lines which are not included in the 7:0 mask are ignored.
23:16	Defines the state of the input lines after the pause execution transition. A '1' indicates that the corresponding input line should be high, a '0' indicates that the corresponding input line should be low. Values for lines which are not included in the 7:0 mask are ignored.

4. *uint32_t : Resume*. A bit mask which defines the transitions required on the digital input lines in order to move execution from the paused state back to the running state. This mask is ignored if *HWctrl* is zero.

Bits	Function
7:0	Defines which input lines are used to determine when execution should be resumed. A '1' indicates that the corresponding input line is sampled. A '0' indicates that the corresponding input line is ignored.
15:8	Defines the state of the input lines prior to the resume execution transition. A '1' indicates that the corresponding input line should be high, a '0' indicates that the corresponding input line should be low. Values for lines which are not included in the 7:0 mask are ignored.
23:16	Defines the state of the input lines after the resume execution transition. A '1' indicates that the corresponding input line should be high, a '0' indicates that the corresponding input line should be low. Values for lines which are not included in the 7:0 mask are ignored.

5. *uint32_t : Stop*. A bit mask which defines the transitions required on the digital input lines in order to stop execution. This mask is ignored if *HWctrl* is zero.

Bits	Function
7:0	Defines which input lines are used to determine when execution should be stopped. A '1' indicates that the corresponding input line is sampled. A '0' indicates that the corresponding input line is ignored.
15:8	Defines the state of the input lines prior to the stop execution transition. A '1' indicates that the corresponding input line should be high, a '0' indicates that the corresponding input line should be low. Values for lines which are not included in the 7:0 mask are ignored.
23:16	Defines the state of the input lines after the stop execution transition. A '1' indicates that the corresponding input line should be high, a '0' indicates that the corresponding input line should be low. Values for lines which are not included in the 7:0 mask are ignored.

Set HWctrl Command

Cmd	Trans ID	Len	HWctrl	Start	Pause	Resume	Stop	Frame
0x15	uint8_t	0x0014	uint32_t	uint32_t	uint32_t	uint32_t	uint32_t	0xAA

The *Set HWctrl* command response has the same payload as the *Set HWctrl* command.

Set HWctrl Command Response

Cmd	Trans ID	Len	HWctrl	Start	Pause	Resume	Stop	Frame
0x15	uint8_t	0x0014	uint32_t	uint32_t	uint32_t	uint32_t	uint32_t	0xAA

4.5.8 Get HWctrl

The *Get HWctrl* command determines whether execution is under software or hardware control.

The *Get HWctrl* command has no payload:

Get HWctrl Command

Cmd	Trans ID	Len	Frame
0x95	uint8_t	0x000	0xAA

The *Get SHWctrl* command response has 5 fields:

1. *uint32_t : HWctrl.* Determines whether the application is running under hardware control. If this field is non-zero then the application is running under hardware control.

2. *uint32_t : Start.* A bit mask which defines the transitions required on the digital input lines to start executing the currently selected mode. This mask is invalid if *HWctrl* is zero.

Bits	Function
7:0	Defines which input lines are used to determine when execution should start. A '1' indicates that the corresponding input line is sampled. A '0' indicates that the corresponding input line is ignored.
15:8	Defines the state of the input lines prior to the start execution transition. A '1' indicates that the corresponding input line should be high, a '0' indicates that the corresponding input line should be low. Values for lines which are not included in the 7:0 mask are ignored.
23:16	Defines the state of the input lines after the start execution transition. A '1' indicates that the corresponding input line should be high, a '0' indicates that the corresponding input line should be low. Values for lines which are not included in the 7:0 mask are ignored.

3. *uint32_t : Pause.* A bit mask which defines the transitions required on the digital input lines in order to pause execution. This mask is invalid if *HWctrl* is zero.

Bits	Function
7:0	Defines which input lines are used to determine when execution should be paused. A '1' indicates that the corresponding input line is sampled. A '0' indicates that the corresponding input line is ignored.
15:8	Defines the state of the input lines prior to the pause execution transition. A '1' indicates that the corresponding input line should be high, a '0' indicates that the corresponding input line should be low. Values for lines which are not included in the 7:0 mask are ignored.
23:16	Defines the state of the input lines after the pause execution transition. A '1' indicates that the corresponding input line should be high, a '0' indicates that the corresponding input line should be low. Values for lines which are not included in the 7:0 mask are ignored.

4. *uint32_t : Resume*. A bit mask which defines the transitions required on the digital input lines in order to move execution from the paused state back to the running state. This mask is invalid if *HWctrl* is zero.

Bits	Function
7:0	Defines which input lines are used to determine when execution should be resumed. A '1' indicates that the corresponding input line is sampled. A '0' indicates that the corresponding input line is ignored.
15:8	Defines the state of the input lines prior to the resume execution transition. A '1' indicates that the corresponding input line should be high, a '0' indicates that the corresponding input line should be low. Values for lines which are not included in the 7:0 mask are ignored.
23:16	Defines the state of the input lines after the resume execution transition. A '1' indicates that the corresponding input line should be high, a '0' indicates that the corresponding input line should be low. Values for lines which are not included in the 7:0 mask are ignored.

5. *uint32_t : Stop*. A bit mask which defines the transitions required on the digital input lines in order to stop execution. This mask is invalid if *HWctrl* is zero.

Bits	Function
7:0	Defines which input lines are used to determine when execution should be stopped. A '1' indicates that the corresponding input line is sampled. A '0' indicates that the corresponding input line is ignored.
15:8	Defines the state of the input lines prior to the stop execution transition. A '1' indicates that the corresponding input line should be high, a '0' indicates that the corresponding input line should be low. Values for lines which are not included in the 7:0 mask are ignored.
23:16	Defines the state of the input lines after the stop execution transition. A '1' indicates that the corresponding input line should be high, a '0' indicates that the corresponding input line should be low. Values for lines which are not included in the 7:0 mask are ignored.

Get SigGen State Command Response

Cmd	Trans ID	Len	HWctrl	Start	Pause	Resume	Stop	Frame
0x95	uint8_t	0x0014	uint32_t	uint32_t	uint32_t	uint32_t	uint32_t	0xAA

4.5.9 Set Wanted Signal

The *Set Wanted Signal* command programs the wanted signal generator when the Cuprite application is in signal generator mode. This command can only be executed when the Cuprite application is in signal generator mode.

The payload of the *Set Wanted Signal* command is:

1. *uint8_t : On*. Flag to indicate whether the wanted signal generator should be active. If this flag is non-zero, then the wanted signal generator is active, otherwise the wanted signal generator is halted if it is already running. If this flag is not set, then there should be no other fields in the payload and the payload length should be set equal to 1.
2. *uint8_t : Chan*. The RF channel number for the wanted signal. Valid channels are in the range 11 to 26.
3. *int16_t : Pwr*. The power of the wanted signal in units of 0.1 dB. This should be in range -120 dBm to 0 dBm.
4. *uint16_t : DIO*. A mask indicating which digital IO lines should be toggled high when the wanted signal is active. Bits 0 to 7 in the mask correspond to digital output lines 0 to 7. If a bit is set to '1', then the corresponding digital output line will be set high when a packet is being transmitted.
5. *uint16_t : PktNum*. The number of packets to be transmitted. If this value is specified as 0, then the wanted signal transmitter will continue to generate packets until it is terminated. The maximum value which can be specified is 65535.
6. *uint32_t : PktInt*. The interval between the start of one packet and the start of the subsequent packet in units of µs.
7. *uint32_t : JitMag*. The magnitude of the peak timing jitter between packets in units of 100ns. A value of zero will disable timing jitter. The jitter is relative to the packets nominal start time. Valid values are from 0 to 1s.
8. *uint16_t : JitType*. The distribution of the timing jitter between packets. Possible values are:

0	None
1	Uniform
2	Gaussian

   If 'None' is selected then packet timing jitter will be disabled.
9. *uint16_t : Ramp*. The length of the power ramp at the start and end of the packet in units of 100ns. Valid values are from 1.1µs to 10µs.
10. *uint16_t : Front Porch*. The length of unmodulated carrier inserted after the power ramp up and the start of the packet modulation in units of 100ns. Valid values are from 0µs to 25µs.
11. *uint16_t : Back Porch*. The length of unmodulated carrier inserted after the end of the packet modulation and prior to the power ramp down in units of 100ns. Valid values are from 0µs to 25µs.
12. *int16_t : CarrierOff*. The carrier offset in units of 100Hz. Valid values are in the range -250kHz to +250kHz.
13. *int16_t : DriftMag*. Magnitude of the sinusoidal carrier drift which is imposed on the packet in units of 100Hz. Valid values are in the range -70KHz to +70kHz. Setting this parameter to zero will disable the carrier drift feature.

14. *uint16_t : DriftRate*. The frequency of the sinusoidal carrier drift which is imposed on the packet in units of 0.1 Hz. Valid values are in the range 0 to 2440Hz.
15. *int16_t : SymbolTime*. The symbol timing error in units of 0.1ppm. Valid values are in the range -100ppm to +100ppm.
16. *int16_t : IQ Bal*. The IQ amplitude imbalance in units of 0.001 dB. Valid values are in the range OdB to 3dB.
17. *int16_t :* IQ Phi. The IQ quadrature error in units of 0.001°. Valid values are in the range -10° to +10°
18. *uint32_t : CER*. The chip error rate in units of 0.000001 (ie 0.0001%). Maximum value is 0.0625 (ie 6.25%). When this parameter is non-zero, the modulator randomly inverts chips to achieve the specified chip error rate.
19. *char[N] : Payload*. The contents of the wanted signal's payload. The format of this parameter can take on two different forms.
    In the first form, each pair of asci characters represent the hex value of each octet in the payload. In this form there must always be an even number of characters.
    In the second form, it is possible to specify a length of a PRBS sequence. The parameter must take the form:
    nnnPRBSmm
    where *nnn* is the number of octets in the payload and *mm* determines which PRBS sequence to use. Possible values for *mm* are 9, 11, 15, 20, 23, 29 and 31.

Set Wanted Signal (long variant)

Cmd	Trans ID	Len	On = 1	Chan	Pwr	DIO	PktNum
0x0D	uint8_t	uint8_t	uint8_t	uint8_t	int16_t	uint16_t	uint16_t

PktInt	JitMag	JitType	Ramp	Front Porch	Back Porch
uint32_t	uint32_t	uint16_t	uint16_t	uint16_t	uint16_t

CarrierOff	DriftMag	DriftRate	SymbolTime	IQ Bal	IQ Phi
int16_t	int16_t	uint16_t	int16_t	int16_t	Int16_t

CER	Payload
uint32_t	N x char

Frame
0xAA

Set Wanted Signal (short variant)

Cmd	Trans ID	Len	On = 0	Frame
0x0D	uint8_t	0x0001	uint8_t	0xAA

The *Set Wanted Signal* command response is a copy of the *Set Wanted Signal* command:

Set Wanted Signal Response (long variant)

Cmd	Trans ID	Len	On = 1	Chan	Pwr	DIO	PktNum
0x0D	uint8_t	uint8_t	uint8_t	uint8_t	int16_t	uint16_t	uint16_t

PktInt	JitMag	JitType	Ramp	Front Porch	Back Porch
uint32_t	uint32_t	uint16_t	uint16_t	uint16_t	uint16_t

CarrierOff	DriftMag	DriftRate	SymbolTime	IQ Bal	IQ Phi
int16_t	int16_t	uint16_t	int16_t	int16_t	Int16_t

CER	Payload
uint32_t	N x char

Frame
0xAA

Set Wanted Signal Response (short variant)

Cmd	Trans ID	Len	On = 0	Frame
0x0D	uint8_t	0x001	uint8_t	0xAA

4.5.10 Get Wanted Signal

The *Get Wanted Signal* command returns the last successfully executed *Set Wanted Signal* command. This command can only be executed when the Cuprite application is in signal generator mode.

The *Get Wanted Signal* command has no payload.

Get Wanted Signal Command

Cmd	Trans ID	Len	Frame
0x8D	uint8_t	0x0000	0xAA

The payload of the *Get Wanted Signal* command response is an exact copy of the last successfully executed *Set Wanted Signal* command:

Get Wanted Signal Response

Cmd	Trans ID	Len	On = 1	Chan	Pwr	DIO	PktNum
0x8D	uint8_t	uint8_t	uint8_t	uint8_t	int16_t	uint16_t	uint16_t

PktInt	JitMag	JitType	Ramp	Front Porch	Back Porch
uint32_t	uint32_t	uint16_t	uint16_t	uint16_t	uint16_t

CarrierOff	DriftMag	DriftRate	SymbolTime	IQ Bal	IQ Phi
int16_t	int16_t	uint16_t	int16_t	int16_t	Int16_t

CER	Payload
uint32_t	N x char

Frame
0xAA

The meaning of the fields in the payload is as follows:

1. *uint8_t : On*. Flag to indicate whether the wanted signal generator should be active. If this flag is non-zero, then the wanted signal generator is active, otherwise the wanted signal generator is halted if it is already running. If this flag is not set, then there should be no other fields in the payload and the payload length should be set equal to 1.
2. *uint8_t : Chan*. The RF channel number for the wanted signal. Valid channels are in the range 11 to 26.

3. *int16_t : Pwr*. The power of the wanted signal in units of 0.1 dB. This should be in range -120 dBm to 0 dBm.
4. *uint16_t : DIO*. A mask indicating which digital IO lines should be toggled high when the wanted signal is active. Bits 0 to 7 in the mask correspond to digital output lines 0 to 7. If a bit is set to '1', then the corresponding digital output line will be set high when a packet is being transmitted.
5. *uint16_t : PktNum*. The number of packets to be transmitted. If this value is specified as 0, then the wanted signal transmitter will continue to generate packets until it is terminated. The maximum value which can be specified is 65535.
6. *uint32_t : PktInt*. The interval between the start of one packet and the start of the subsequent packet in units of µs.
7. *uint32_t : JitMag*. The magnitude of the peak timing jitter between packets in units of 100ns. A value of zero will disable timing jitter. The jitter is relative to the packets nominal start time. Valid values are from 0 to 1s.
8. *uint16_t : JitType.* The distribution of the timing jitter between packets. Possible values are:

0	None
1	Uniform
2	Gaussian

   If 'None' is selected then packet timing jitter will be disabled.
9. *uint16_t : Ramp*. The length of the power ramp at the start and end of the packet in units of 100ns. Valid values are from 1.1µs to 10µs.
10. *uint16_t : Front Porch*. The length of unmodulated carrier inserted after the power ramp up and the start of the packet modulation in units of 100ns. Valid values are from 0µs to 25µs.
11. *uint16_t : Back Porch*. The length of unmodulated carrier inserted after the end of the packet modulation and prior to the power ramp down in units of 100ns. Valid values are from 0µs to 25µs.
12. *int16_t : CarrierOff.* The carrier offset in units of 100Hz. Valid values are in the range -250kHz to +250kHz.
13. *int16_t : DriftMag*. Magnitude of the sinusoidal carrier drift which is imposed on the packet in units of 100Hz. Valid values are in the range -70KHz to +70kHz. Setting this parameter to zero will disable the carrier drift feature.
14. *uint16_t : DriftRate*. The frequency of the sinusoidal carrier drift which is imposed on the packet in units of 0.1 Hz. Valid values are in the range 0 to 2440Hz.
15. *int16_t : SymbolTime*. The symbol timing error in units of 0.1ppm. Valid values are in the range -100ppm to +100ppm.
16. *int16_t : IQ Bal.* The IQ amplitude imbalance in units of 0.001 dB. Valid values are in the range OdB to 3dB.
17. *int16_t :* IQ Phi. The IQ quadrature error in units of 0.001°. Valid values are in the range -10° to +10°
18. *uint32_t : CER*. The chip error rate in units of 0.000001 (ie 0.0001%). Maximum value is 0.0625 (ie 6.25%). When this parameter is non-zero, the modulator randomly inverts chips to achieve the specified chip error rate.
19. *char[N] : Payload*. The contents of the wanted signal's payload. The format of this parameter can take on two different forms.
    In the first form, each pair of asci characters represent the hex value of each octet in the payload. In

this form there must always be an even number of characters.
In the second form, it is possible to specify a length of a PRBS sequence. The parameter must take the form:

nnnPRBSmm

where *nnn* is the number of octets in the payload and *mm* determines which PRBS sequence to use. Possible values for *mm* are 9, 11, 15, 20, 23, 29 and 31.

4.5.11 Set Interferer Signal

The *Set Interferer Signal* command programs the modulated interferer signal generator when the Cuprite application is in signal generator mode. This command can only be executed when the Cuprite application is in signal generator mode.

The payload of the *Set Interferer Signal* command is:

1. *uint8_t : On*. Flag to indicate whether the modulated interferer signal generator should be active. If this flag is non-zero, then the modulated interferer signal generator is active, otherwise the modulated interferer signal generator is halted if it is already running. If this flag is not set, then there should be no other fields in the payload and the payload length should be set equal to 1.
2. *uint8_t : Cont*. Flag to indicate whether the interferer signal is packetised or continuous. If this flag is non-zero, then the interferer signal will be continuously modulated, otherwise it will be packetised.
3. *int16_t : Pwr*. The power of the modulated interferer signal in units of 0.1 dB. Valid range is -120dBm to 0dBm.
4. *uint16_t : DIO*. A mask indicating which digital IO lines should be toggled high when the interferer signal is active. Bits 0 to 7 in the mask correspond to digital output lines 0 to 7. If a bit is set to '1', then the corresponding digital output line will be set high when the interferer is active.
5. *uint16_t : Octets*. The number of octets in the packet payload. Valid values are in the range 0 to 127. This parameter is ignored if the *Cont* parameter is non-zero.
6. *uint32_t : Freq*. The carrier frequency for the interferer signal in Hz. Valid range is 2395 MHz to 2485 MHz inclusive.

The modulated interferer always uses a PRBS15 as its data source.

Set Interferer Signal (long variant)

Cmd	Trans ID	Len	On ≠ 0	Cont	Pwr	DIO	Octets	Freq
0x0E	uint8_t	0x0010	uint8_t	uint8_t	int16_t	uint16_t	uint16_t	uint32_t

PktInt	Frame
uint32_t	0xAA

Set Interferer Signal (short variant)

Cmd	Trans ID	Len	On = 0	Frame
0x0E	uint8_t	0x0001	uint8_t	0xAA

The *Set Interferer Signal* command response is a copy of the *Set Interferer Signal* command:

Set Interferer Signal Response (long variant)

Cmd	Trans ID	Len	On ≠ 0	Cont	Pwr	DIO	Octets	Freq
0x0E	uint8_t	0x0010	uint8_t	uint8_t	int16_t	uint16_t	uint16_t	uint32_t

PktInt	Frame
uint32_t	0xAA

Set Interferer Signal Response (short variant)

Cmd	Trans ID	Len	On = 0	Frame
0x0E	uint8_t	0x0001	uint8_t	0xAA

4.5.12 Get Interferer Signal

The *Get Interferer Signal* command returns the last successfully executed *Set Interferer Signal* command. This command can only be executed when the Cuprite application is in signal generator mode.

The *Get Interferer Signal* command has no payload.

Get Interferer Signal Command

Cmd	Trans ID	Len	Frame
0x8E	uint8_t	0x0000	0xAA

The payload of the *Get Interferer Signal* command response is an exact copy of the last successfully executed *Set Interferer Signal* command:

Get Interferer Signal Response

Cmd	Trans ID	Len	On ≠ 0	Cont	Pwr	DIO	Octets	Freq
0x8E	uint8_t	0x0010	uint8_t	uint8_t	int16_t	uint16_t	uint16_t	uint32_t

PktInt	Frame
uint32_t	0xAA

The meaning of the fields in the *Get Interferer Signal Response* is as follows:

1. *uint8_t : On*. Flag to indicate whether the modulated interferer signal generator should be active. If this flag is non-zero, then the modulated interferer signal generator is active, otherwise the modulated interferer signal generator is halted if it is already running. If this flag is not set, then there should be no other fields in the payload and the payload length should be set equal to 4.
2. *uint8_t : Cont*. Flag to indicate whether the interferer signal is packetised or continuous. If this flag is non-zero, then the interferer signal will be continuously modulated, otherwise it will be packetised.
3. *int16_t : Pwr*. The power of the modulated interferer signal in units of 0.1 dB. Valid range is -120dBm to 0dBm.
4. *uint16_t : DIO*. A mask indicating which digital IO lines should be toggled high when the interferer signal is active. Bits 0 to 7 in the mask correspond to digital output lines 0 to 7. If a bit is set to '1', then the corresponding digital output line will be set high when the interferer is active.
5. *uint16_t : Octets*. The number of octets in the packet payload. Valid values are in the range 0 to 127. This parameter is ignored if the *Cont* parameter is non-zero.
6. *uint32_t : Freq*. The carrier frequency for the interferer signal in Hz. Valid range is 2395 MHz to 2485 MHz inclusive.

The modulated interferer always uses a PRBS15 as its data source.

4.5.13 Set In-band CW

The *Set In-band CW* command programs one of the in-band CW signal generators when the Cuprite application is in signal generator mode. This command can only be executed when the Cuprite application is in signal generator mode.

The payload contents are:

1. *uint8_t : On.* Flag to indicate whether the in-band CW signal should be turned on or off. If this field is non-zero then the in-band CW signal is turned on. If this field is zero, then the remaining fields can be omitted and the length field set to 2.
2. *uint8_t : CW.* The number of the in-band CW signal to be programmed. Valid values are 0, 1 or 2.
3. *int16_t : Pwr.* The power of the in-band CW signal in units of 0.1 dB. Valid range is -120 dBm to 0 dBm.
4. *uint32_t : Freq.* The frequency of the in-band CW signal in Hz. Valid range is 2,395,000,000 Hz to 2,485,000,000 Hz inclusive.
5. *uint32_t : OnPeriod.* The burst duration of the in-band CW signal in units of 100ns. Valid values are in the range 100ns to 1.6s. If it is desired to have a non-bursted in-band CW signal then this parameter should be set to 1 and the *OffPeriod* should be set to 0.
6. *uint32_t : OffPeriod.* The period between bursts for which the in-band CW signal should be off in units of 100ns. Valid values are in the range 0 to 1.6s. If it is desired to have a non-bursted in-band CW signal then this parameter should be set to 1 and the *OffPeriod* should be set to 0.

Set In-band CW Command (long variant)

Cmd	Trans ID	Len	On	CW	Pwr	Freq
0x06	uint8_t	0x0010	uint8_t	uin8_t	int16_t	uint32_t

OnPeriod	OffPeriod	Frame
uint32_t	uint32_t	0xAA

Set In-band CW Command (short variant)

Cmd	Trans ID	Len	On = 0	CW	Frame
0x06	uint8_t	0x0002	uint8_t	uin8_t	0xAA

The *Set In-band CW* command response is a copy of the *Set In-band CW* command.

Set In-band CW Command Response (long variant)

Cmd	Trans ID	Len	On	CW	Pwr	Freq
0x06	uint8_t	0x0010	uint8_t	uin8_t	int16_t	uint32_t

OnPeriod	OffPeriod	Frame
uint32_t	uint32_t	0xAA

Set In-band CW Command Response (short variant)

Cmd	Trans ID	Len	On = 0	CW	Frame
0x06	uint8_t	0x0002	uint8_t	uin8_t	0xAA

4.5.14 Get In-band CW

The *Get In-band CW* command returns the last successfully executed *Set In-band CW* command. This command can only be executed when the Cuprite application is in signal generator mode.

The *Get In-band CW* command has a single uint8_t as its payload which denotes which in-band CW signal is being queried. Valid values are 0, 1 and 2.

Get In-band CW Command

Cmd	Trans ID	Len	CW	Frame
0x86	uint8_t	0x0001	uin8_t	0xAA

The payload of the *Get In-band CW* command response is a copy of the last successfully executed *Set In-band CW* command:

Get In-band CW Command Response

Cmd	Trans ID	Len	On	CW	Pwr	Freq
0x86	uint8_t	0x0010	uint8_t	uin8_t	int16_t	uint32_t

OnPeriod	OffPeriod	Frame
uint32_t	uint32_t	0xAA

The payload fields are:

1. *uint8_t : On.* Flag to indicate whether the in-band CW signal is turned on or off. If this field is non-zero then the in-band CW signal is turned on.
2. *uint8_t : CW.* The number of the in-band CW signal being interogated. Valid values are 0, 1 or 2.
3. *int16_t : Pwr.* The power of the in-band CW signal in units of 0.1 dB. Valid range is -120 dBm to 0 dBm.
4. *uint32_t : Freq.* The frequency of the in-band CW signal in Hz. Valid range is 2,395,000,000 Hz to 2,485,000,000 Hz inclusive.
5. *uint32_t : OnPeriod.* The burst duration of the in-band CW signal in units of 100ns. Valid values are in the range 100ns to 1.6s
6. *uint32_t : OffPeriod.* The period between bursts for which the in-band CW signal should be off in units of 100ns. Valid values are in the range 0 to 1.6s. If this value is zero then burst mode has been disabled.

4.5.15 Set AWGN

The *Set AWGN* command programs the AWGN signal generators when the Cuprite application is in signal generator mode. This command can only be executed when the Cuprite application is in signal generator mode.

The payload contents are:

1. *uint16_t : On.* Flag to indicate whether the AWGN signal should be turned on or off. If this field is non-zero then the AWGN signal is turned on. If this field is zero, then the remaining fields can be omitted and the length field set to 2.
2. *int16_t : Pwr.* The power of the AWGN signal in units of 0.1 dB. Valid range is -162 dBm/MHz to -39.1 dBm/MHz.
3. *uint32_t : OnPeriod.* The burst duration of the AWGN signal in units of 100ns. Valid values are in the range 100ns to 1.6s. If a continuous transmission is required then set this field to 1 and the *OffPeriod* field to 0.
4. *uint32_t : OffPeriod.* The period between bursts for which the AWGN signal should be off in units of 100ns. Valid values are in the range 0 to 1.6s. If this value is zero then burst mode is disabled.

Set AWGN Command (long variant)

Cmd	Trans ID	Len	On	Pwr	OnPeriod	OffPeriod	Frame
0x07	uint8_t	0x000C	uint16_t	int16_t	uint32_t	uint32_t	0xAA

Set AWGN Command (short variant)

Cmd	Trans ID	Len	On = 0	Frame
0x07	uint8_t	0x0001	uint16_t	0xAA

The *Set AWGN* command response is a copy of the *Set AWGN* command.

Set AWGN Command Response (long variant)

Cmd	Trans ID	Len	On	Pwr	OnPeriod	OffPeriod	Frame
0x07	uint8_t	0x000C	uint16_t	int16_t	uint32_t	uint32_t	0xAA

Set AWGN Command Response (short variant)

Cmd	Trans ID	Len	On = 0	Frame
0x07	uint8_t	0x0001	uint16_t	0xAA

4.5.16 Get AWGN

The *Get AWGN* command returns the last successfully executed *Set AWGN* command. This command can only be executed when the Cuprite application is in signal generator mode.

The *Get In-band CW* command has no payload.

Get AWGN Command

Cmd	Trans ID	Len	Frame
0x87	uint8_t	0x0000	0xAA

The payload of the *Get AWGN* command response is a copy of the last successfully executed *Set AWGN* command:

Get AWGN Command Response

Cmd	Trans ID	Len	On	Pwr	OnPeriod	OffPeriod	Frame
0x87	uint8_t	0x000C	uint16_t	int16_t	uint32_t	uint32_t	0xAA

The meaning of the fields in the payload are:

1. *uint16_t : On.* Flag to indicate whether the AWGN signal is on or off. If this field is non-zero then the AWGN signal is turned on.
2. *int16_t : Pwr.* The power of the AWGN signal in units of 0.1 dB. Valid range is -162 dBm/MHz to -39.1 dBm/MHz.
3. *uint32_t : OnPeriod.* The burst duration of the AWGN signal in units of 100ns. Valid values are in the range 100ns to 1.6s.
4. *uint32_t : OffPeriod.* The period between bursts for which the AWGN signal should be off in units of 100ns. Valid values are in the range 0 to 1.6s. If this value is zero then burst mode has been disabled.

4.5.17 Set Out-of-band CW

The *Set Out-of-band CW* command programs the out-of-band CW signal generator when the Cuprite application is in signal generator mode. This command can only be executed when the Cuprite application is in signal generator mode.

The payload contents are:

1. *uint16_t : On.* A flag to indicate whether the out-of-band CW generator should be turned on or off. If this flag is non-zero, then the out-of-band CW generator is turned on, otherwise the out-of-band CW generator is turned off. If this field is zero, then the remaining fields can be omitted and the length field set to 2.
2. *int16_t : Pwr.* The power of the out-of-band CW signal in units of 0.1 dB. Valid range is -50dBm to -25 dBm.
3. *uint32_t : Freq.* The frequency of the out-of-band CW signal in MHz. Valid range is 24 MHz to 6,000 MHz.

Set Out-of-band CW Command (long variant)

Cmd	Trans ID	Len	On	Pwr	Freq	Frame
0x0B	uint8_t	0x0008	uint16_t	int16_t	uint32_t	0xAA

Set Out-of-band CW Command (short variant)

Cmd	Trans ID	Len	On = 0	Frame
0x0B	uint8_t	0x0002	uint16_t	0xAA

The *Set Out-of-band CW* command response is a copy of the *Set Out-of-band CW* command. If the out-of-band CW generator is not active, then the contents of the *Freq* and *Pwr* fields are not valid.

Set Out-of-band CW Command Response (long variant)

Cmd	Trans ID	Len	On	Pwr	Freq	Frame
0x0B	uint8_t	0x0008	uint16_t	int16_t	uint32_t	0xAA

Set Out-of-band CW Command Response (short variant)

Cmd	Trans ID	Len	On = 0	Frame
0x0B	uint8_t	0x0002	uint16_t	0xAA

4.5.18 Get Out-of-band CW

The *Get Out-of-band CW* command returns the last successfully executed Set Out-of-band CW command. This command can only be executed when the Cuprite application is in signal generator mode.

The *Get Out-of-band CW* command has no payload.

Get Out-of-band CW Command

Cmd	Trans ID	Len	Frame
0x8B	uint8_t	0x000	0xAA

The payload of the *Get Out-of-band CW* command response is a copy of the last successfully executed *Set Out-of-band CW* command:

Get Out-of-band CW Command Response (long variant)

Cmd	Trans ID	Len	On	Pwr	Freq	Frame
0x8B	uint8_t	0x0008	uint16_t	int16_t	uint32_t	0xAA

The payload fields are:

1. *uint16_t : On*. A flag to indicate whether the out-of-band CW generator is on or off. If this flag is non-zero, then the out-of-band CW generator is turned on, otherwise the out-of-band CW generator is turned off.
2. *int16_t : Pwr*. The power of the out-of-band CW signal in units of 0.1 dB. Valid range is -50dBm to -25 dBm.
3. *uint32_t : Freq*. The frequency of the out-of-band CW signal in MHz. Valid range is 24 MHz to 6,000 MHz.

4.5.19 Set SigAna

The *Set SigAna* command determines various run parameters for the signal analyser. This command can only be executed when the Cuprite application is in signal analyser mode.

The *Set SigAna* command payload consists of 9 fields:

1. *uint16_t : ChanMask*. A 16 bit mask indicating which RF channels the signal analyser should examine. A '1' in the mask indicates that the corresponding RF channel should be analysed, a '0' in the mask indicates that the corresponding RF channel should be ignored. The least significant bit corresponds to channel 11 and the most significant bit to channel 26.
2. *uint16_t : PktMask*. A 16 bit mask to determine which packet lengths should be analysed. Packet lengths are divide into 16 groups. Each bit in the mask corresponds to one of the 16 packet length groups. If the bit is set to '1', then packets falling within the group will be analysed, otherwise the packets will be discarded. The 16 packet length groups are:

Group	Packet lengths
0	$x \leq 7$
1	$8 \leq x \leq 15$
2	$16 \leq x \leq 23$
3	$24 \leq x \leq 31$
4	$32 \leq x \leq 39$
5	$40 \leq x \leq 47$
6	$48 \leq x \leq 53$
7	$54 \leq x \leq 63$
8	$64 \leq x \leq 71$
9	$72 \leq x \leq 79$
10	$80 \leq x \leq 87$
11	$88 \leq x \leq 95$
12	$96 \leq x \leq 103$
13	$104 \leq x \leq 111$
14	$112 \leq x \leq 119$
15	$120 \leq x$

3. *int8_t : RSSIthresh*. An RSSI threshold for packets to be analysed. The threshold can be in the range -120dBm to +20dBm. Packets will only be analysed if their received signal strength is above this threshold.

4. *uin8_t : TestMask.* A bit mask indicating which types of data should be collected:

Bit position	Bit mask	Collect
0	0x01	Collect decoded bit stream
1	0x02	Collect basic measurements such as power, EVM, carrier frequency and symbol timing
2	0x04	Collect data from the carrier tracking loop
3	0x08	Collect raw IQ data
4	0x10	Collect average spectrum data
5	0x20	Collect peak spectrum data

5. *uint8_t : Oversample.* The oversampling ratio to be used to collect IQ waveform data, if selected by the *TestMask* field. Possible oversampling ratios are:

Value	Oversampling ratio
0	x2
1	x2.5
2	x5
3	x10

6. *uint8_t : NumAvg.*The time period over which spectral data are averaged or over which the spectrum peak hold function is applied. Valid values are:

Value	Time interval	Value	Time interval
0	10 µs	16	671 ms
1	20 µs	17	1.3 s
2	41 µs	18	2.7 s
3	82 µs	19	5.4 s
4	164 µs	20	10 s
5	328 µs	21	21 s
6	655 µs	22	43 s
7	1.3 ms	23	86 s
8	2.6 ms	24	171 s
9	5.2 ms	25	5.7 mins
10	10 ms	26	11 mins
11	21 ms	27	23 mins
12	42 ms	28	46 mins
13	84 ms	29	92 mins
14	168 ms	30	3 hrs
15	336 ms	31	6 hrs

7. *uint16_t : SyncThresh.* The threshold which the synchroniser uses to detect the commencement of a packet. A suitable value is 1488. Raising the threshold will result in loss of packets which are heavily distorted or masked by interference whilst reducing the threshold will result in more packets being detected but also the chance of false detections.
8. *uint8_t : StopOnFail.* A flag to indicate how the signal analyser should respond when a limit failure is detected. If this flag is zero, then the signal analyser ignores the limit failure and continues to run. If this flag is non-zero, then the signal analyser will complete the analysis of the current failing packet and then halt.

Set SigAna Command

Cmd	Trans ID	Len	ChanMask	PktMask	RSSIthresh	TestMask	Oversample
0x14	uint8_t	0x000B	uint16_t	uint16_t	int8_t	uint8_t	uint8_t

NumAvg	SyncThresh	StopOnFail	Frame
uint8_t	uint16_t	uint8_t	0xAA

The *Set SigAna* command response is a copy of the *Set SigAna* command:

Set SigAna Command Response

Cmd	Trans ID	Len	ChanMask	PktMask	RSSIthresh	TestMask	Oversampe
0x14	uint8_t	0x000B	uint16_t	uint16_t	int8_t	uint8_t	uint8_t

NumAvg	SyncThresh	StopOnFail	Frame
uint8_t	uint16_t	uint8_t	0xAA

4.5.20 Get SigAna

The *Get SigAna* command returns the last successfully executed *Set SigAna* command. This command can only be executed when the Cuprite application is in signal analyser mode.

The *Get SigAna* command has no payload.

Get Out-of-band CW Command

Cmd	Trans ID	Len	Frame
0x94	uint8_t	0x000	0xAA

The payload of the *Get SigAna* command response is a copy of the last successfully executed *Set SigAna* command:

Get SigAna Command Response

Cmd	Trans ID	Len	ChanMask	PktMask	RSSIthresh	TestMask	Oversampe
0x94	uint8_t	0x0008	uint16_t	uint16_t	int8_t	uint8_t	uint8_t

NumAvg	SyncThresh	StopOnFail	Frame
uint8_t	uint16_t	uint8_t	0xAA

The meaning of the paylod fields are:

1. *uint16_t : ChanMask*. A 16 bit mask indicating which RF channels the signal analyser should examine. A '1' in the mask indicates that the corresponding RF channel should be analysed, a '0' in the mask indicates that the corresponding RF channel should be ignored. The least significant bit corresponds to channel 11 and the most significant bit to channel 26.
2. *uint16_t : PktMask*. A 16 bit mask to determine which packet lengths should be analysed. Packet lengths are divide into 16 groups. Each bit in the mask corresponds to one of the 16 packet length groups. If the bit is set to '1', then packets falling within the group will be analysed, otherwise the packets will be discarded. The 16 packet length groups are:

Group	Packet lengths
0	x ≤ 7
1	8 ≤ x ≤ 15
2	16 ≤ x ≤ 23
3	24 ≤ x ≤ 31
4	32 ≤ x ≤ 39
5	40 ≤ x ≤ 47
6	48 ≤ x ≤ 53
7	54 ≤ x ≤ 63
8	64 ≤ x ≤ 71
9	72 ≤ x ≤ 79
10	80 ≤ x ≤ 87
11	88 ≤ x ≤ 95
12	96 ≤ x ≤ 103
13	104 ≤ x ≤ 111
14	112 ≤ x ≤ 119
15	120 ≤ x

3. *int8_t : RSSIthresh.* An RSSI threshold for packets to be analysed. The threshold can be in the range -120dBm to +20dBm. Packets will only be analysed if their received signal strength is above this threshold.
4. *uin8_t : TestMask.* A bit mask indicating which types of data should be collected:

Bit position	Bit mask	Collect
0	0x01	Collect decoded bit stream
1	0x02	Collect basic measurements such as power, EVM, carrier frequency and symbol timing
2	0x04	Collect data from the carrier tracking loop
3	0x08	Collect raw IQ data
4	0x10	Collect average spectrum data
5	0x20	Collect peak spectrum data

5. *uint8_t : Oversample.* The oversampling ratio to be used to collect IQ waveform data, if selected by the *TestMask* field. Possible oversampling ratios are:

Value	Oversampling ratio
0	x2
1	x2.5
2	x5
3	x10

6. *uint8_t : NumAvg.*The time period over which spectral data are averaged or over which the spectrum peak hold function is applied. Valid values are:

Value	Time interval	Value	Time interval
0	10 µs	16	671 ms
1	20 µs	17	1.3 s
2	41 µs	18	2.7 s
3	82 µs	19	5.4 s
4	164 µs	20	10 s
5	328 µs	21	21 s
6	655 µs	22	43 s
7	1.3 ms	23	86 s
8	2.6 ms	24	171 s
9	5.2 ms	25	5.7 mins
10	10 ms	26	11 mins
11	21 ms	27	23 mins
12	42 ms	28	46 mins
13	84 ms	29	92 mins
14	168 ms	30	3 hrs
15	336 ms	31	6 hrs

7. *uint16_t : SyncThresh.* The threshold which the synchroniser uses to detect the commencement of a packet. A suitable value is 1488. Raising the threshold will result in loss of packets which are heavily distorted or masked by interference whilst reducing the threshold will result in more packets being detected but also the chance of false detections.
8. *uint8_t : StopOnFail.* A flag to indicate how the signal analyser should respond when a limit failure is detected. If this flag is zero, then the signal analyser ignores the limit failure and continues to run. If this flag is non-zero, then the signal analyser will complete the analysis of the current failing packet and then halt.

4.5.21 Set Test Limits

The *Set Test Limits* command informs the Cuprite application of the test limits to be applied. The test limits are used in the signal analyser mode. This command can be executed at any time.

The *Set Test Limits* command has 7 int16_t fields:

1. *int16_t : MinPwr.* Minimum power for the packet in units of 0.1dBm. The packet power is averaged over the packet. Mimum value for limit is -120dBm abd the maximum value +30dBm.
2. *int16_t : MaxPwr.* Maximum power for the packet in units of 0.1dBm. The packet power is averaged over the packet. Mimum value for limit is -120dBm abd the maximum value +30dBm.
3. *int16_t : F_c.* The maximum permissible carrier frequency error in units of kHz. The carrier frequency is estimated throughout the packet. The minimum value for the limit is 0kHz and the maximum value 200kHz.
4. *int16_t : Symb.* The maximum symbol timing error in units of ppm. The symbol timing error is averaged over several packets in order to obtain a reliable result. The minimum value for the limit is 0ppm and the maximum value 100ppm.
5. *int16_t : EVM.* The maximum value of the EVM measured over 1000 chips in units of 0.1%. The minimum value for the limit is 0% and the maximum value 100%.
6. *int16_t : PSD_{abs}.* The maximum value of the power spectral density absolute measurement in units of 0.1dBm. The minimum vaue for the limit is -100dBm and the maximum value +20dBm.
7. *int16_t : PSD_{rel}.* The maximum value of the power spectral density relative measurement in units of 0.1dB. The minimum vaue for the limit is -100dB and the maximum value 0dB.

Set Test Limits Command

Cmd	Trans ID	Len	MinPwr	MaxPwr	F_c	Symb	EVM	PSD_{abs}	PSD_{rel}	Frame
0x1A	uint8_t	0x0012	int16_t	int16_t	int16_t	int16	int16_t	int16_t	int16_t	0xAA

The *Set Test Limits* command response is a copy of the *Set Test Limits* command:

Set Test Limits Command Response

Cmd	Trans ID	Len	MinPwr	MaxPwr	F_c	Symb	EVM	PSD_{abs}	PSD_{rel}	Frame
0x1A	uint8_t	0x0012	int16_t	int16_t	int16_t	int16	int16_t	int16_t	int16_t	0xAA

4.5.22 Get Test Limits

The *Get Test Limits* command retrieves the test limits held in the Cuprite application. The test limits are used in the signal analyser mode. This command can be executed at any time.

The *Get Test Limits command* has no payload.

Get Test Limits Command

Cmd	Trans ID	Len	Frame
0x9A	uint8_t	0x00	0xAA

The *Get Test Limits* command response contains 7 fields indicating the various limits which are currently enforced by the signal analyser:

Get Test Limits Command Response

Cmd	Trans ID	Len	MinPwr	MaxPwr	F_c	Symb	EVM	PSD_{abs}	PSD_{rel}	Frame
0x9A	uint8_t	0x0014	int16_t	int16_t	int16_t	int16	int16_t	int16_t	int16_t	0xAA

The meaning of the fields is:

1. *int16_t : MinPwr.* Minimum power for the packet in units of 0.1dBm. The packet power is aveaged over the packet. Mimum value for limit is -120dBm abd the maximum value +30dBm.
2. *int16_t : MaxPwr.* Maximum power for the packet in units of 0.1dBm. The packet power is averaged over the packet. Mimum value for limit is -120dBm abd the maximum value +30dBm.
3. *int16_t : F_c*. The maximum permissible carrier frequency error in units of kHz. The carrier frequency is estimated throughout the packet. The minimum value for the limit is 0kHz and the maximum value 200kHz.
4. *int16_t : Symb*. The maximum symbol timing error in units of ppm. The symbol timing error is averaged over several packets in order to obtain a reliable result. The minimum value for the limit is 0ppm and the maximum value 100ppm.
5. *int16_t : EVM.* The maximum value of the EVM measured over 1000 chips in units of 0.1%. The minimum value for the limit is 0% and the maximum value 100%.
6. *int16_t : PSD_{abs}*. The maximum value of the power spectral density absolute measurement in units of 0.1dBm. The minimum vaue for the limit is -100dBm and the maximum value +20dBm.
7. *int16_t : PSD_{rel}*. The maximum value of the power spectral density relative measurement in units of 0.1dB. The minimum vaue for the limit is -100dB and the maximum value 0dB.

4.5.23 Poll Results Table

The *Poll Results Table* command returns results in a tabular form when operating in signal analyser mode.

The results contributing to the returned table are filtered by:

1. An RF channel filter
2. A packet length group filter

Any packets which satisfy all the filters will contribute to the statistics in the returned table.

The *Poll Results Table* command payload consists of 3 fields:

1. *uint32_t : Meas*. The measurement set for which tabular results are requested. Possible values are:

Meas	Measurement set
0	Output power
1	Carrier frequency, drift & symbol timing
2	EVM
3	Spectrum

2. *uint32_t : ChanMask*. A bit mask indicating containing the RF channel filter. Only packets which pass the RF channel filter results will contribute to the statistics in the returned table. A '1' in the mask indicates that the corresponding RF channel results should be included in the table, a '0' in the mask indicates that the corresponding RF channel results should be ignored. The least significant bit in the mask corresponds to channel 11.
3. *uint32_t : PktMask*. This mask contains the packet length filter. Only packets which pass the packet length filter will contribute to the statistics in the returned table. There are 16 packet length groups. The first length group corresponds to packets with payloads in the range 0 to 7 octets, the second group to packets with payloads of 8 to 15 octets, etc. If a bit is '1', then the results for the corresponding packet length group will contribute to the statistics in the returned table. If a bit is '0', then the results for the corresponding packet length group are ignored.

Poll Results Table Command

Cmd	Trans ID	Len	Meas	ChanMask	PktMask	Frame
0x95	uint8_t	0x000C	uint32_t	uint32_t	uint32_t	0xAA

The *Poll Results Table* command response payload consists of a 2D array of floating point numbers. These are ordered by column and then by row. The number of rows in the array depends on the value of the *Meas* field in the *Poll Results Table* command. The general format of the *Poll Results Table* command response is:

Poll Results Table Command Response

Cmd	Trans ID	Len	Samples	Min	Max	Avg	Curr
0x95	uint8_t	20*N	float	float	float	float	float

…….	Frame
{Samples, Min, Max, Avg, Curr} repeated x (N-1)	0xAA

The fields in the payload are:

1. float : Samples. The number of samples which have contributed to this row in the results table. For many measurements this will be equal to the number of packets analysed.
2. float : Min. The minimum value which has been observed for the quantity corresponding to this row in the table.
3. float : Max. The maximum value which has been observed for the quantity corresponding to this row in the table.
4. float : Avg.The average value which has been observed for the quantity corresponding to this row in the table.
5. float : Curr. The last value which has been observed for the quantity corresponding to this row in the table.

The tables returned for the various measurement sets are:

1. Output power

Quantity	Packet Count	Minimum	Maximum	Average	Current
P_{avg}					
$Pk - P_{avg}$					

2. Carrier frequency, drift and symbol timing

Quantity	Packet Count	Minimum	Maximum	Average	Current
Initial F_c					
Min F_c					
Max F_c					
Drift Rate					
Symbol timing					

3. EVM

Quantity	Sample Count	Minimum	Maximum	Average	Current
EVM_{1000}					
EVM_{rms}					
EVM_{pk}					
$\Delta\phi_{rms}$					
$\Delta\phi_{rms}$					
IQ Imbalance					

4. Spectrum

Quantity	Sample Count	Minimum	Maximum	Average	Current
PSD_{abs}					
PSD_{rel}					

4.5.24 Poll Results Plot

To access plot data, a sequence of two commands is required:

1. *Poll Plot Results*. This command determines whether data is available, and if so, stashes it within the Cuprite application.
2. *Get Plot Results*. This command reads back sections of the stashed data.

Multiple *Get Plot Results* commands can be issued to retrieve different sections of the stashed data.

4.5.24.1 Plotting basic quantities against channel or packet length

This section describes how to obtain plots of basic quantities against one of two parameters:

1. RF channel
2. Packet length

The returned plot against the selected parameter can also be filtered on the basis of the remaining parameter.

Three sequences of values are returned. The first sequence corresponds to the minimum value of the quantity, the second sequence to the average value of the quantity and the third sequence to the maximum value of the quantity.

The payload of the *Poll Results Plot* command has 4 fields:

1. *uint32_t : Quantity*. The measured quantity for which the plot is requested. Possible values are:

Quantity	Measurement
0x00000000	P_{avg}
0x00000001	$Pk - P_{avg}$
0x00000002	$Initial\ F_c$
0x00000003	$Min\ F_c$
0x00000004	$Max\ F_c$
0x00000005	*Drift*
0x00000006	*Symbol Timing*
0x00000007	EVM_{1000}
0x00000008	EVM_{rms}
0x00000009	EVM_{pk}
0x0000000A	$\Delta\varphi_{rms}$
0x0000000B	$\Delta\varphi_{pk}$
0x0000000C	*IQ Imbalance*
0x0000000D	PSD_{abs}
0x0000000E	PSD_{rel}

.

2. *uint32_t : Method*. Describes the type of plot which is requested. Possible values are:

0	Quantity vs RF channel
1	Quantity vs packet length group

3. *uint32_t : ChanMask.* A bit mask indicating which RF channels' results should be included in the plot. A '1' in the mask indicates that the corresponding RF channel results should be included in the plot, a '0' in the mask indicates that the corresponding RF channel results should be ignored. The least significant bit corresponds to channel 11. This mask is ignored when *Method* equals 0.
4. *uint32_t : PktMask.* This mask indicates which packet length groups should be included in the results plot. There are 16 packet length groups. The first packet length group includes packets with payload lengths in the range 0 to 7 octets, the second group contains packets with payload lengths of 8 to 15 octets, etc. If a bit is '1', then the results for the corresponding packet length group are included in the plot. If a bit is '0', then the results for the corresponding packet length group are ignored. This mask is ignored when *Method* equals 1 or when the selected quantity is P_{avg} or P_{pk}-P_{avg}.

Poll Results Plot Command

Cmd	Trans ID	Len	Quantity	Method	ChanMask	PktMask	Frame
0x16	uint8_t	0x0010	uint32_t	uin32_t	uint32_t	uint32_t	0xAA

The *Poll Results Plots* command response message payload returns a single uint32_t which denotes the number of bytes of data which are available to be read back. If no plot data is available then this field will be zero.

Poll Results Plot Command Response

Cmd	Trans ID	Len	N	Frame
0x16	uint8_t	0x0004	uint32_t	0xAA

The number of samples available to be read back is:

Method	Number of returned samples	Bytes available to read
0	3*16 = 48	192
1	3*16 = 48	192

For *Method* = 0, the first 16 samples correspond to the minimum value observed over RF channels 11 to 26. The next 16 samples correspond to the average value observed over RF channels 11 to 26. The final 16 samples correspond to the maximum value observed over RF channels 11 to 26.

For *Method* = 1, the first 16 samples correspond the minimum value observed for each of the 16 packet length group. The next 16 samples correspond the average value observed for each of the 16 packet length groups. The final 16 samples correspond to the maximum value observed for each of the 16 packet length groups.

The units of the returned quantities are:

Measurement	Units
P_{avg}	dBm
$Pk - P_{avg}$	dB
Initial F_c	kHz
Min F_c	kHz
Max F_c	kHz
Drift	Hz/µs
Symbol Timing	ppm
EVM_{1000}	%
EVM_{rms}	%
EVM_{pk}	%
$\Delta\varphi_{rms}$	°
$\Delta\varphi_{pk}$	°
IQ Imbalance	dB
PSD_{abs}	dBm
PSD_{rel}	dB

To access the plot data, a *Get Results Plot* command must be issued. The payload of this command has two fields:

1. *uint32_t : Offset*. The index of the first byte to be returned. Bytes are indexed from zero. This field should be a multiple of 4.
2. *uint32_t : N*. The number of bytes to be returned. This field should be a multiple of 4.

Get Results Plot Command

Cmd	Trans ID	Len	Offset	N	Frame
0x96	uint8_t	0x0008	uint32_t	uint32_t	0xAA

The *Get Results Plot* command response contains a list of floats representing *Quantity* as a function of the specified *Method*. A total of *N/4* samples are returned, the first sample being at index *Offset/4*.

Get Results Plot Command Response

Cmd	Trans ID	Len	Value 1	Value 2	…	Value N/4	Frame
0x96	uint8_t	N	float	float	…	float	0xAA

4.5.24.2 Obtaining histograms of basic quantities

This section describes how a histogram of a basic quantity can be obtained. The histogram can be filtered by:

1. RF channel
2. Packet length

The payload of the *Poll Results Plot* command has 4 fields:

1. *uint32_t : Quantity*. The measured quantity for which the histogram is requested. Possible values are:

Quantity	Measurement
0x00000000	P_{avg}
0x00000001	$Pk - P_{avg}$
0x00000002	$Initial\ F_c$
0x00000003	$Min\ F_c$
0x00000004	$Max\ F_c$
0x00000005	*Drift*
0x00000006	*Symbol Timing*
0x00000007	EVM_{1000}
0x00000008	EVM_{rms}
0x00000009	EVM_{pk}
0x0000000A	$\Delta\varphi_{rms}$
0x0000000B	$\Delta\varphi_{pk}$
0x0000000C	*IQ Imbalance*
0x0000000D	PSD_{abs}
0x0000000E	PSD_{rel}

2. *uint32_t : Method*. Describes the type of plot which is requested. For a histogram this should be set to 2.
3. *uint32_t : ChanMask*. A bit mask indicating which RF channels' results should be included in the hisotgram. A '1' in the mask indicates that the corresponding RF channel results should be included in the hisotgram, a '0' in the mask indicates that the corresponding RF channel results should be ignored. The least significant bit corresponds to channel 11.
4. *uint32_t : PktMask*. This mask indicates which packet length groups should be included in the hisotgram. There are 16 packet length groups. The first packet length group includes packets with payload lengths in the range 0 to 7 octets, the second group contains packets with payload lengths of 8 to 15 octets, etc. If a bit is '1', then the results for the corresponding packet length group are included in the hisotgram. If a bit is '0', then the results for the corresponding packet length group are ignored. This parameter is ignored if the selected quantity is P_{avg} or P_{pk}-P_{avg}.

Poll Results Plot Command

Cmd	Trans ID	Len	Quantity	Method=2	ChanMask	PktMask	Frame
0x16	uint8_t	0x0010	uint32_t	uin32_t	uint32_t	uint32_t	0xAA

The *Poll Results Plots* command response message payload returns a single uint32_t which denotes the number of bytes which are available to be read back. If no hisotgram data is available then this field will be zero. If data is available, this field will be set to 512, corresponding to a histogram, of length 128.

Poll Results Plot Command Response

Cmd	Trans ID	Len	N	Frame
0x16	uint8_t	0x0004	uint32_t	0xAA

To access the hisotgram data, a *Get Results Plot* command must be issued. The payload of this command has two fields:

1. *uint32_t : Offset.* The index of the first byte to be returned. Bytes are indexed from zero. This field should be a multiple of 4.
2. *uint32_t : N.* The number of bytes to be returned. This field should be a multiple of 4.

Get Results Plot Command

Cmd	Trans ID	Len	Offset	N	Frame
0x96	uint8_t	0x0008	uint32_t	uint32_t	0xAA

The *Get Results Plot* command response contains a list floats representing a histogram of *Quantity.* A total of *N/4* samples are returned, the first sample being at index *Offset/4.*

Get Results Plot Command Response

Cmd	Trans ID	Len	Value Offset/4	Value Offset/4+1	...	Value (Offset-1+N)/4	Frame
0x96	uint8_t	N	float	float	...	float	0xAA

A maximum of 128 floating point samples can be returned. These form a histogram with equally spaced bins. The lower and upper edges of the first and last bins are:

Measurement	Lower edge of 1st bin	Upper edge of last bin
P_{avg}	-100 dBm	+28 dBm
$Pk - P_{avg}$	0 db	6.4 dB
Initial F_c	-256 kHz	+256 kHz
Min F_c	-256 kHz	+256 kHz
Max F_c	-256 kHz	+256 kHz
Drift	0 Hz/µs	+256 Hz/µs
Symbol Timing	-128 ppm	+128 ppm
EVM_{1000}	0 %	50 %
EVM_{rms}	0 %	50 %
EVM_{pk}	0 %	50 %
$\Delta\varphi_{rms}$	0 °	12.8 °
$\Delta\varphi_{pk}$	0 °	12.8 °
IQ Imbalance	-5.12 dB	+5.12 dB
PSD_{abs}	-120 dBm	+8 dBm
PSD_{rel}	-70 dB	-6 dB

4.5.24.3 Obtaining current value and plot vs time

This section describes how to obtain the current value of a quantity and obtain a plot vs time. The current value is defined as the value measured on the most recent packet which satisfies two filters:

1. RF channel filter
2. Packet length filter

The payload of the *Poll Results Plot* command has 4 fields:

1. *uint32_t : Quantity.* The measured quantity for which the current value is requested. Possible values are:

Quantity	Measurement	Associated time plot
0x00000000	P_{avg}	*Power vs time*
0x00000001	*Pk -* P_{avg}	*Power vs time*
0x00000002	*Initial* F_c	F_c *vs time*
0x00000003	*Min* F_c	F_c *vs time*
0x00000004	*Max* F_c	F_c *vs time*
0x00000005	*Drift*	F_c *vs time*
0x00000006	*Symbol Timing*	*Power vs time*
0x00000007	EVM_{1000}	*EVM vs time*
0x00000008	EVM_{rms}	*EVM vs time*
0x00000009	EVM_{pk}	*EVM vs time*
0x0000000A	$\Delta\varphi_{rms}$	$\Delta\varphi$ *vs time*
0x0000000B	$\Delta\varphi_{pk}$	$\Delta\varphi$ *vs time*
0x0000000C	*IQ Imbalance*	*Power vs time*
0x0000000D	PSD_{abs}	*Power vs time*
0x0000000E	PSD_{rel}	*Power vs time*

2. *uint32_t : Method.* This must be set to 3 to obtain the current value and a time plot.
3. *uint32_t : ChanMask.* A mask indicating which RF channels should be included in the search for the most recent packet. A '1' in the mask indicates that the corresponding RF channel should be included in the search, a '0' in the mask indicates that the corresponding RF channel should be ignored. The least significant bit corresponds to channel 11.
4. *uint32_t : PktMask.* This mask indicates which packet length groups should be included in the search for the most recent packet. There are 16 packet length groups. The first packet length group includes packets with payload lengths in the range 0 to 7 octets, the second group contains packets with payload lengths of 8 to 15 octets, etc. If a bit is '1', then the results for the corresponding packet length group are included in the search. If a bit is '0', then the results for the corresponding packet length group are ignored. This parameter is ignored if the selected quantity is P_{avg} or P_{pk}-P_{avg}.

Poll Results Plot Command

Cmd	Trans ID	Len	Quantity	Method=3	ChanMask	PktMask	Frame
0x16	uint8_t	0x0010	uint32_t	uin32_t	uint32_t	uint32_t	0xAA

The *Poll Results Plots* command response message payload returns a single uint32_t which denotes the number of bytes which are available to be read back. If no plot data is available then this field will be zero.

Poll Results Plot Command Response

Cmd	Trans ID	Len	N	Frame
0x16	uint8_t	4	uint32_t	0xAA

To access the plot data, a *Get Results Plot* command must be issued. The payload of this command has two fields:

1. *uint32_t : Offset*. The index of the first bytes to be returned. Bytes are indexed from zero. This field should be a multiple of 4.
2. *uint32_t : N*. The number of bytes to be returned. This field should be a multiple of 4.

Get Results Plot Command

Cmd	Trans ID	Len	Offset	N	Frame
0x96	uint8_t	8	uint32_t	uint32_t	0xAA

The *Get Results Plot* command response contains a total of *N* bytes, the first byte being at index *Offset*. If *Offset* is greater than or equal to 20, then the command will return *N/4* samples starting at index (*Offset-20)/24*

Get Results Plot Command Response

Cmd	Trans ID	Len	Value 1	Value 2	…	Value N	Frame
0x96	uint8_t	uint16_t			…		0xAA

The data which is available to be read back consists of either 4 or 5 words of metadata followed by a sequence of floats which represent a plotted quantity vs time:

1. *uint64_t : Timestamp*. Timestamp of start of packet in units of 100 ns since the Unix epoch of 1970-01-01 00:00:00 + 0000 (UTC).
2. *uint16_t : RF Chan*. The RF channel number on which the packet was collected.
3. *uint16_t : Octets*. The number of octets in the payload of the packet for which the data was collected.
4. *float : Now*. The current value of the selected quantity
5. *uint32_t : Oversample*. The oversampling ratio for the subsequent plotted quantity vs time. Possible oversampling values are:

Value	Oversampling ratio
0	x2
1	x2.5
2	x5
3	x10

This parameter is only present if the associated plot is power vs time.

If the metadata is not required, then the value of *Offset* in the *Get Results Plot* command should be set either to 20 (power vs time) or 16 (all other plots).

Data available to be read back

Timestamp	RF Chan	Octets	Now	(Oversample)
uint64_t	uint16_t	uint16_t	uint32_t	(uint32_t)

Value 1	...	Value N
float	...	float

The units of the returned quantities are:

Measurement	**Units**
P_{avg}	dBm
$Pk - P_{avg}$	dB
Initial F_c	kHz
Min F_c	kHz
Max F_c	kHz
Drift	Hz/µs
Symbol Timing	ppm
EVM_{1000}	%
EVM_{rms}	%
EVM_{pk}	%
$\Delta\varphi_{rms}$	°
$\Delta\varphi_{pk}$	°
IQ Imbalance	dB
PSD_{abs}	dBm
PSD_{rel}	dB

4.5.24.4 Obtaining power spectral density plots

This section describes how to obtain a plot of power spectral density. Separate power spectral density plots are maintained for each of the 16 RF channels. The channel associated with the returned power spectral density plot is determined by finding the most recent packet which satisfies the RF channel filter. The channel on which this packet was received corresponds to the channel for which the power spectral density data will be returned. If only one channel is selected in the RF channel filter, then data will be returned for that channel, if available.

The payload of the *Poll Results Plot* command has 4 fields:

1. *uint32_t : Quantity.* This field should be set to zero.
2. *uint32_t : Method.* To obtain power spectral density data this field should be set to 4.
3. *uint32_t : ChanMask.* A bit mask indicating which RF channels' should be include in the search for the most recent packet. A '1' in the mask indicates that the corresponding RF channel rsults should be included in the search, a '0' in the mask indicates that the corresponding RF channel should be ignored. The least significant bit corresponds to channel 11.
4. *uint32_t : PktMask.* This field is ignored.

Poll Results Plot Command

Cmd	Trans ID	Len	Quantity	Method = 4	ChanMask	PktMask	Frame
0x16	uint8_t	0x0010	uint32_t	uint32_t	uint32_t	uint32_t	0xAA

The *Poll Results Plots* command response message payload returns a single uint32_t which denotes the number of bytes which are available to be read back. If no plot data is available then this field will be zero.

Poll Results Plot Command Response

Cmd	Trans ID	Len	N	Frame
0x16	uint8_t	0x0004	uint32_t	0xAA

To access the data, a *Get Results Plot* command must be issued. The payload of this command has two fields:

1. *uint32_t : Offset.* The index of the first byte to be returned. Bytes are indexed from zero. This field should be a multiple of 2
2. *uint32_t : N.* The number of bytes to be returned. This field should be a multiple of 2

Get Results Plot Command

Cmd	Trans ID	Len	Offset	N	Frame
0x96	uint8_t	0x0008	uint32_t	uint32_t	0xAA

The *Get Results Plot* command response contains a total of *N* bytes, the first byte being at index *Offset*. If *Offset* is greater than or equal to 16, then the command will return *N/2* spectral samples starting at index *(Offset-16)/2*.

Get Results Plot Command Response

Cmd	Trans ID	Len	Value 1	Value 2	…	Value N	Frame
0x96	uint8_t	uint16_t			…		0xAA

The data which is available to be read back consists of 4 words of metadata followed by a sequence of int16_t which represent the power spectral density data:

1. *uint32_t : Unused.* The first field is unused and can be ignored.
2. *uint32_t : Flag,* A two bit flag which indicates which power spectral density is available to be read back:

Bit position	Bit mask	Meaning
0	0x1	Average power spectral density is available
1	0x2	Peak power spectral density is available

3. *uint32_t : RF Chan.* The RF channel number for which the power spectral density data applies to.
4. *float : Now.* The peak power spectral density within ±1MHz of the centre of the RF channel

If the metadata is not required, then the value of *Offset* in the *Get Results Plot* command should be set to 16.

Data available to be read back

Unused	Flag	RF Chan	Now
uint32_t	uint32_t	uint32_t	float

Value 1	…	Value N
int16_t	…	int16_t

Each int16_t corresponds to the power spectral density in units of 1/16th dB.

Each spectrum returned contains 1844 int16_t. The first sample corresponds to 2394.971MHz. Subsequent samples are spaced by 48.828kHz. The spectra are constructed using a Gaussian filter with a 3dB bandwidth of 100kHz. Hence the spectra are oversampled by approximately x2.

The first 1844 int16_t correspond to the average power spectral density, or if this is not available, the peak power spectral density. When both average and peak power spectral densities are available, the second set of 1844 int16_t correspond to the peak power spectral density.

4.5.24.5 Obtaining constellation plots

This section describes how the constellation plots can be obtained. The constellation data is returned for the most recent packet which satisfies two filters:

1. RF channel filter
2. Packet length filter

The series of commands also returns the value of one measured quantity associated with the selected packet.

The payload of the *Poll Results Plot* command has 4 fields:

1. *uint32_t : Quantity*. For each constellation plot it is possible to return one measured quantity of the corresponding packet. The possible measured quantities are:

Quantity	Measurement
0x00000000	P_{avg}
0x00000001	$Pk - P_{avg}$
0x00000002	$Initial\ F_c$
0x00000003	$Min\ F_c$
0x00000004	$Max\ F_c$
0x00000005	*Drift*
0x00000006	*Symbol Timing*
0x00000007	EVM_{1000}
0x00000008	EVM_{rms}
0x00000009	EVM_{pk}
0x0000000A	$\Delta\varphi_{rms}$
0x0000000B	$\Delta\varphi_{pk}$
0x0000000C	*IQ Imbalance*
0x0000000D	PSD_{abs}
0x0000000E	PSD_{rel}

2. *uint32_t : Method*. This field must be set to 5 to obtain data for constellation plot.
3. *uint32_t : ChanMask*. A bit mask indicating which RF channels should be included in the search for the most recent packet. A '1' in the mask indicates that the corresponding RF channel should be included in the search, a '0' in the mask indicates that the corresponding RF channel should be ignored. The least significant bit corresponds to channel 11.
4. *uint32_t : PktMask*. This mask indicates which packet length groups should be included in the search for the most recent packet. There are 16 packet length groups. The first packet length group includes packets with payload lengths in the range 0 to 7 octets, the second group contains packets with payload lengths of 8 to 15 octets, etc. If a bit is '1', then the corresponding packet length group is included in the search. If a bit is '0', then the corresponding packet length group is ignored. This parameter is ignored if the selected quantity is P_{avg} or P_{pk}-P_{avg}.

Poll Results Plot Command

Cmd	Trans ID	Len	Quantity	Method = 5	ChanMask	PktMask	Frame
0x16	uint8_t	0x0010	uint32_t	uin32_t	uint32_t	uint32_t	0xAA

The *Poll Results Plots* command response message payload returns a single uint32_t which denotes the number of bytes which are available to be read back. If no plot data is available then this field will be zero.

Poll Results Plot Command Response

Cmd	Trans ID	Len	N	Frame
0x16	uint8_t	0x0004	uint32_t	0xAA

To access the constellation data, a *Get Results Plot* command must be issued. The payload of this command has two fields:

1. *uint32_t : Offset*. The index of the first byte to be returned. Bytes are indexed from zero. This field should be a multiple of 8.
2. *uint32_t : N*. The number of bytes to be returned. This field should be a multiple of 8.

Get Results Plot Command

Cmd	Trans ID	Len	Offset	N	Frame
0x96	uint8_t	0x0008	uint32_t	uint32_t	0xAA

The *Get Results Plot* command response contains a total of *N* bytes, the first byte being at index *Offset*. If *Offset* is greater than or equal to 16, then the command will return *N/8* pairs of { I , Q } samples starting at index (*Offset-16)/8*.

Get Results Plot Command Response

Cmd	Trans ID	Len	Value 1	Value 2	…	Value N	Frame
0x96	uint8_t	uint16_t			…		0xAA

The data which is available to read back consists of 4 fields of metadata followed by a sequence of pairs of floats representing the normalised constellation points:

1. *uint64_t : Timestamp*. Timestamp of the start of the packet in units of 100 ns since the Unix epoch of 1970-01-01 00:00:00 + 0000 (UTC).
2. *uint16_t : RF Chan*. The RF channel number on which the packet was collected.
3. *uint16_t : Octets*. The number of octets in the payload of the packet for which the data was collected.
4. *float : Now*. The measured value of the requested quantity for the selected packet.
5. *float : I Value M, QValue M*. A sequence of pairs of normalised { I , Q } samples

Data available to read back

Timestamp	RF Chan	Octets	Now
uint64_t	uint16_t	uint16_t	Float

I Value 1	Q Value 1		…	I Value N	Q Value N
float	float		…	float	float

If the metadata is not required, then the value of *Offset* in the *Get Results Plot* command should be set to 16.

4.5.24.6 Obtaining IQ samples

This section describes how raw IQ samples can be obtained. The IQ samples are returned for the most recent packet which satisfies two filters:

1. RF channel filter
2. Packet length filter

The series of commands also returns the value of one measured quantity associated with the selected packet.

The payload of the *Poll Results Plot* command has 4 fields:

1. *uint32_t : Quantity*. For each set of IQ samples it is possible to return one measured quantity of the corresponding packet. The possible measured quantities are:

Quantity	Measurement
0x00000000	P_{avg}
0x00000001	$Pk - P_{avg}$
0x00000002	$Initial\ F_c$
0x00000003	$Min\ F_c$
0x00000004	$Max\ F_c$
0x00000005	*Drift*
0x00000006	*Symbol Timing*
0x00000007	EVM_{1000}
0x00000008	EVM_{rms}
0x00000009	EVM_{pk}
0x0000000A	$\Delta\varphi_{rms}$
0x0000000B	$\Delta\varphi_{pk}$
0x0000000C	*IQ Imbalance*
0x0000000D	PSD_{abs}
0x0000000E	PSD_{rel}

2. *uint32_t : Method*. This field must be set to 6 to obtain IQ samples.
3. *uint32_t : ChanMask*. A bit mask indicating which RF channels should be included in the search for the most recent packet. A '1' in the mask indicates that the corresponding RF channel should be included in the search, a '0' in the mask indicates that the corresponding RF channel should be ignored. The least significant bit corresponds to channel 11.
4. *uint32_t : PktMask*. This mask indicates which packet length groups should be included in the search for the most recent packet. There are 16 packet length groups. The first packet length group includes packets with payload lengths in the range 0 to 7 octets, the second group contains packets with payload lengths of 8 to 15 octets, etc. If a bit is '1', then the corresponding packet length group is included in the search. If a bit is '0', then the corresponding packet length group is ignored. This parameter is ignored if the selected quantity is P_{avg} or P_{pk}-P_{avg}.

5.

Poll Results Plot Command

Cmd	Trans ID	Len	Quantity	Method = 6	ChanMask	PktMask		Frame
0x16	uint8_t	0x0010	uint32_t	uin32_t	uint32_t	uint32_t		0xAA

The *Poll Results Plots* command response message payload returns a single uint32_t which denotes the number of bytes which are available to be read back. If no plot data is available then this field will be zero.

Poll Results Plot Command Response

Cmd	Trans ID	Len	N	Frame
0x16	uint8_t	0x0004	uint32_t	0xAA

To access the IQ samples, a *Get Results Plot* command must be issued. The payload of this command has two fields:

1. *uint32_t : Offset.* The index of the first byte to be returned. Bytes are indexed from zero. This field should be a multiple of 4.
2. *uint32_t : N.* The number of bytes to be returned. This field should be a multiple of 4.

Get Results Plot Command

Cmd	Trans ID	Len	Offset	N	Frame
0x96	uint8_t	0x0008	uint32_t	uint32_t	0xAA

The *Get Results Plot* command response contains a total of *N* bytes, the first byte being at index *Offset*. If *Offset* is greater than or equal to 20, then the command will return *N/4* IQ complex samples starting at index *(N-20)/4*.

Get Results Plot Command Response

Cmd	Trans ID	Len	Value 1	Value 2	…	Value N	Frame
0x96	uint8_t	uint16_t			…		0xAA

The data which is available to read back consists of 5 fields of metadata followed by a sequence of uint32_t which encode each IQ sample in a polar format:.

1. *uint64_t : Timestamp.* Timestamp of the start of the packet in units of 100 ns since the Unix epoch of 1970-01-01 00:00:00 + 0000 (UTC).
2. *uint16_t : RF Chan.* The RF channel number on which the packet was collected.
3. *uint16_t : Octets.* The number of octets in the payload of the packet for which the data was collected.
4. *float : Now.* The measured value of the requested quantity for the selected packet.

5. *uint32_t : OverSample.* The oversampling ratio of the IQ samples. Possible values are:

0	2x oversampling
1	2.5x oversampling
2	5x oversampling
3	10x oversampling

6. *uint32_t : Real/Imag M.* A sequence of uint32_t representing the IQ samples in a compressed polar format.

Data available to read back

Timestamp	RF Chan	Octets	Now	Oversample
uint64_t	uint16_t	uint16_t	Float	uint32_t

Real 1	Imag 1	…	Real N	Imag N
uint32_t	uint32	…	uint32	uint32

If the metadata is not required, then the value of *Offset* in the *Get Results Plot* command should be set to 20.

The IQ samples are encoded in a polar format. To extract the IQ samples from each uint32_t perform the following operations:

```
float scale = atanf(1.0f) / 256.0f
float phi = (x & 0x7FF) * scale;
float mag = (x >> 11) & 0x7FF;
float rot = 31 - (x >> 22);
float v = powf(2.0f, rot) * mag;
float iVal = v * cosf(phi);
float qVal = v * sinf(phi);
```

4.5.24.7 Obtaining FM deviation data

This section describes how FM deviation data can be obtained. The FM deviation data are returned for the most recent packet which satisfies two filters:

1. RF channel filter
2. Packet length filter

The series of commands also returns the value of one measured quantity associated with the selected packet.

The payload of the *Poll Results Plot* command has 4 fields:

1. *uint32_t : Quantity*. For each set of FM deviation data it is possible to return one measured quantity of the corresponding packet. The possible measured quantities are:

Quantity	Measurement
0x00000000	P_{avg}
0x00000001	$Pk - P_{avg}$
0x00000002	$Initial\ F_c$
0x00000003	$Min\ F_c$
0x00000004	$Max\ F_c$
0x00000005	*Drift*
0x00000006	*Symbol Timing*
0x00000007	EVM_{1000}
0x00000008	EVM_{rms}
0x00000009	EVM_{pk}
0x0000000A	$\Delta\varphi_{rms}$
0x0000000B	$\Delta\varphi_{pk}$
0x0000000C	*IQ Imbalance*
0x0000000D	PSD_{abs}
0x0000000E	PSD_{rel}

2. *uint32_t : Method*. This field must be set to 7 to obtain FM deviation data.
3. *uint32_t : ChanMask*. A 16 bit mask indicating which RF channels should be included in the search for the most recent packet. A '1' in the mask indicates that the corresponding RF channel should be included in the search, a '0' in the mask indicates that the corresponding RF channel should be ignored. The least significant bit corresponds to channel 11.
4. *uint32_t : PktMask*. This mask indicates which packet length groups should be included in the search for the most recent packet. There are 16 packet length groups. The first packet length group includes packets with payload lengths in the range 0 to 7 octets, the second group contains packets with payload lengths of 8 to 15 octets, etc. If a bit is '1', then the corresponding packet length group is included in the search. If a bit is '0', then the corresponding packet length group is ignored. This parameter is ignored if the selected quantity is P_{avg} or P_{pk}-P_{avg}.

5.

Poll Results Plot Command

Cmd	Trans ID	Len	Quantity	Method = 7	ChanMask	PktMask	Frame
0x16	uint8_t	0x0010	uint32_t	uin32_t	uint32_t	uint32_t	0xAA

The *Poll Results Plots* command response message payload returns a single uint32_t which denotes the number of bytes which are available to be read back. If no plot data is available then this field will be zero.

Poll Results Plot Command Response

Cmd	Trans ID	Len	N	Frame
0x16	uint8_t	0x0004	uint32_t	0xAA

To access the FM deviation data, a *Get Results Plot* command must be issued. The payload of this command has two fields:

1. *uint32_t : Offset.* The index of the first byte to be returned. Bytes are indexed from zero. This field should be a multiple of 4.
2. *uint32_t : N.* The number of bytes to be returned. This field should be a multiple of 4.

Get Results Plot Command

Cmd	Trans ID	Len	Offset	N	Frame
0x96	uint8_t	0x0008	uint32_t	uint32_t	0xAA

The *Get Results Plot* command response contains a total of *N* bytes, the first byte being at index *Offset*. If *Offset* is greater than or equal to 20, then the command will return *N/4* samples of FM deviation samples starting at index *(N-20)/4*.

Get Results Plot Command Response

Cmd	Trans ID	Len	Value 1	Value 2	…	Value N	Frame
0x96	uint8_t	uint16_t			…		0xAA

The data which is available to read back consists of 5 fields of metadata followed by a sequence of float which contain the FM deviation:

1. *uint64_t : Timestamp.* Timestamp of the start of the packet in units of 100 ns since the Unix epoch of 1970-01-01 00:00:00 + 0000 (UTC).
2. *uint16_t : RF Chan.* The RF channel number on which the packet was collected.
3. *uint16_t : Octets.* The number of octets in the payload of the packet for which the data was collected.
4. *float : Now.* The measured value of the requested quantity for the selected packet.

5. *uint32_t : OverSample*. The oversampling ratio of the FM deviation samples. Possible values are:

0	2x oversampling
1	2.5x oversampling
2	5x oversampling
3	10x oversampling

6. *float : FM M*. A sequence of FM deviation values in units of kHz.

Data available to read back

Timestamp	RF Chan	Octets	Pwr	Oversample
uint64_t	uint16_t	uint16_t	Float	uint32_t

FM 1	FM 2	…	FM N
float	float	…	float

If the metadata is not required, then the value of *Offset* in the *Get Results Plot* command should be set to 20.

4.5.25 Poll Packet

The *Poll Packet* command returns the demodulated bitstream associated with the packet used in the last *Poll Plot* command. This command can only be executed when Cuprite is in signal analyser mode.

The *Poll Packet* command has no payload.

Poll Packet Command

Cmd	Trans ID	Len	Frame
0x98	uint8_t	0x0000	0xAA

The *Poll Packet* command response contains the demodulated bitstream. The first 5 octets are always 0x00, 0x00, 0x00, 0x00, 0xA7 representing the preamble and SFD fields of the packet. If no data is available, the length field is set to zero.

Poll Packet Command Response

Cmd	Trans ID	Len	**Packet**	Frame
0x98	uint8_t	N	**uint8_t[N]**	0xAA

4.5.26 Clear Results

The *Clear Results* command erases all test results accumulated in the Cuprite application in the signal analyser mode. This command can only be executed in signal analyser mode.

The *Clear Results* command has the following payload:

1. *uint32_t : ChanMask*. This is a mask indicating the RF channels for which the results should be cleared. The location of each bit in the mask corresponds directly to the RF channel whose results should be cleared. The least significant bit corresponds to channel 11.
2. *uint32_t : PktMask*. This is a mask indicating the packet length groups for which the results should be cleared. There are 16 packet length groups, each group spanning 8 packet lengths. The first packet length group is for payload lengths in the range 0 to 7, the second group spans 8 to 15, etc. The location of each bit corresponds to the associated packet length group.

Clear Results Command

Cmd	Trans ID	Len	ChanMask	PktMask	Frame
0x0A	uint8_t	0x0008	uint32_t	uint32_t	0xAA

The *Clear Results* command response has no payload.

Clear Results Command Response

Cmd	Trans ID	Len	Frame
0x0A	uint8_t	0x0000	0xAA

4.5.27 Set Sniffer

The *Set Sniffer* command programs the operation of the packet sniffer. This command can only executed in packet sniffer mode.

The *Set Sniffer* command payload has 3 fields:

1. *uint16_t : ChanMask*. This is a mask indicating the RF channels for which packets should be sniffed. A '1' in the mask indicates that the associated channel should be sniffed, whilst a '0' indicates that the associated channel should be ignored. The least significant bit corresponds to channel 11
2. *int16_t : RSSIthresh*. The RSSI threshold for packets to be sniffed in units of dBm. Packets below the threshold will be ignored. Valid threshold values are in the range -120 dBm to +30dBm inclusive.
3. *uint16_t : SyncThresh*. The synchroniser threshold to be used when sniffing packets. Increasing the synchroniser threshold will decrease sensitivity but reduce the susceptibility to false detections. The default value for the synchroniser threshold is 1150. Valid values for the threshold are in the range 900 to 1500, inclusive.

Set Sniffer Command

Cmd	Trans ID	Len	ChanMask	RSSI thresh	SyncThresh	Frame
0x17	uint8_t	0x0006	uint16_t	int16_t	uint16_t	0xAA

The *Set Sniffer* command response is a copy of the *Set Sniffer* command:

Set Sniffer Command Response

Cmd	Trans ID	Len	ChanMask	RSSI thresh	SyncThresh	Frame
0x17	uint8_t	0x0006	uint16_t	int16_t	uint16_t	0xAA

4.5.28 Get Sniffer

The *Get Sniffer* command returns the current settings of the packet sniffer. This command can only be executed in packet sniffer mode.

The *Get Sniffer* command has no paylod.

Get Sniffer Command

Cmd	Trans ID	Len	Frame
0x27	uint8_t	0x0000	0xAA

The *Get Sniffer* command response has 3 fields in its payload:

1. *uint16_t : ChanMask*. This is a mask indicating the RF channels for which packets should be sniffed. A '1' in the mask indicates that the associated channel should be sniffed, whilst a '0' indicates that the associated channel should be ignored. The least significant bit corresponds to channel 11
2. *int16_t : RSSIthresh*. The RSSI threshold for packets to be sniffed in units of dBm. Packets below the threshold will be ignored. Valid threshold values are in the range -120 dBm to +30dBm inclusive.
3. *uint16_t : SyncThresh*. The synchroniser threshold to be used when sniffing packets. Increasing the synchroniser threshold will decrease sensitivity but reduce the susceptibility to false detections. The default value for the synchroniser threshold is 1150. Valid values for the threshold are in the range 900 to 1500, inclusive.

Get Sniffer Command Response

Cmd	Trans ID	Len	ChanMask	RSSI thresh	SyncThresh	Frame
0x27	uint8_t	0x0006	uint16_t	int16_t	uint16_t	0xAA

4.5.29 Set Receiver Attenuation

The *Set Receiver Attenuation* host command allows the host to set the receiver frontend attenuation. This command can be executed at any time.

The payload consists of a single uin32_t which denotes the receiver frontend attenuation in units of 0.5 dB. The permissible attenuator range is 0 to 31.5 dB.

Set Receiver Attenuation Command

Cmd	Trans ID	Len	Attenuation	Frame
0x09	uint8_t	0x0004	uint32_t	0xAA

The Set Receiver Attenuation Response is of the same format as the Set Receiver Attenuation Command packet.

Set Receiver Attenuation Response

Cmd	Trans ID	Len	Attenuation	Frame
0x09	uint8_t	0x0004	uint32_t	0xAA

4.5.30 Get Receiver Attenuation

The *Get Receiver Attenuation* host command allows the host to query the current setting of the receiver front-end attenuator. This command can be executed at any time.

This command has no payload.

Get Receiver Attenuation Command

Cmd	Trans ID	Len	Frame
0x89	uint8_t	0x0000	0xAA

The *Get Receiver Attenuation Response* has a payload which consists of a single uint32_t which denotes the current receiver front-end attenuator setting in units of 0.5 dB. Permissible values for the attenuator setting are in the range 0 to 31.5 dB.

Get Receiver Attenuation Response

Cmd	Trans ID	Len	Attenuation	Frame
0x89	uint8_t	0x0004	uint32_t	0xAA

4.5.31 Set Rx Port

The *Set Rx Port* command determines whether the Monitor In port or the Tx/Rx port should be used for reception. The Monitor In port is has a noise figure of +6 dB and so is ideally suited for performing off-air measurements. The Tx/Rx port has a noise figure of +46 dB but can handle signals as large as +27 dBm. It is therefore ideally suited for conducted measurements. This command can be executed at any time.

The *Set Rx Port* command payload consists of a single uint8_t which determines which port should be used for reception:

0	Monitor In
1	Tx/Rx port

Set Rx Port Command

Cmd	Trans ID	Len	Port	Frame
0x10	uint8_t	0x0001	uint8_t	0xAA

The Set Rx Port command response is a copy of the Set Rx Port command:

Set Rx Port Command Response

Cmd	Trans ID	Len	Port	Frame
0x10	uint8_t	0x0001	uint8_t	0xAA

4.5.32 Get Rx Port

The *Get Rx Port* command determines whether the Monitor In port or the Tx/Rx port is currently in use for reception. The Monitor In port is has a noise figure of +6 dB and so is ideally suited for performing off-air measurements. The Tx/Rx port has a noise figure of +46 dB but can handle signals as large as +27 dBm. It is therefore ideally suited for conducted measurements. This command can be executed at any time.

The *Get Rx Port* command has no payload.

Get Rx Port Command

Cmd	Trans ID	Len	Frame
0x90	uint8_t	0x000	0xAA

The *Get Rx Port* command response payload consists of a single unit8_t denoting which port is currently in use for reception:

0	Monitor In port
1	Tx/Rx port

Get Rx Port Command Response

Cmd	Trans ID	Len	Port	Frame
0x90	uint8_t	0x0001	uint8_t	0xAA

4.5.33 Set DIO Volts

The *Set DIO Volts* command determines whether the digital IO voltage is supplied by *TLF3000* or is supplied by an external source. If *TLF3000* supplies the digital IO voltage then it is fixed at 3.3 V. The *TLF3000* is able to provide up to 500 mA on the 3v3 supply. If an external voltage is used for the digital IO then it must be in the range 0.8 V to 3.6 V. This command can be executed at any time.

The *Set DIO Volts* command payload consists of a single uint8_t which determines the source of the digital IO voltage:

0	3.3 V IO supplied by TLF3000
1	0.8 V to 3.6 V IO supplied externally

Set DIO Volts Command

Cmd	Trans ID	Len	Source	Frame
0x11	uint8_t	0x0001	uint8_t	0xAA

The *Set DIO Volts* command response is a copy of the *Set DIO Volts* command:

Set DIO Volts Command Response

Cmd	Trans ID	Len	Source	Frame
0x11	uint8_t	0x0001	uint8_t	0xAA

4.5.34 Get DIO Volts

The *Get DIO Volts* command returns whether the digital IO voltage is supplied by *TLF3000* or is supplied by an external source. If *TLF3000* supplies the digital IO voltage then it is fixed at 3.3 V. The *TLF3000* is able to provide up to 500 mA on the 3v3 supply. If an external voltage is used for the digital IO then it must be in the range 0.8 V to 3.6 V. This command can be executed at any time.

The *Get DIO Volts* command has no payload.

Get DIO Volts Command

Cmd	Trans ID	Len	Frame
0x91	uint8_t	0x000	0xAA

The *Get DIO Volts* command response payload consists of a single unit8_t denoting whether the digital IO voltage is supplied by TLF3000 or is supplied externally:

0	3.3 V IO supplied by TLF3000
1	0.8 V to 3.6 V IO supplied externally

Get DIO Volts Command Response

Cmd	Trans ID	Len	Source	Frame
0x91	uint8_t	0x0001	uint8_t	0xAA

4.5.35 Set PeriodicMessage Mask

The *Set PeriodicMessage Mask* host command determines which periodic data messages will be sent from Cuprite to the host over the asynchronous data interface.

The payload consists of a single uint_16 which contains a mask determining which periodic data messages will be broadcast. Each bit in the mask corresponds to a different periodic data message. If the bit is set to '1' then the message is broadcast, if the bit is cleared to '0' then the message is not broadcast. Bits in the mask are numbered from zero.

Bit	Mask	Corresponding periodic data message
10	0x0400	Periodic input/output power messages
11	0x0800	Periodic environmental monitoring messages

Set Periodic Message Mask Command

Cmd	Trans ID	Len	Capture mask	Frame
0x00	uint8_t	0x0002	uint16_t	0xAA

The *Set Periodic Message Mask* response is merely a copy of the command packet.

Set Periodic Message Mask Response

Cmd	Trans ID	Len	Capture mask	Frame
0x00	uint8_t	0x0002	uint16_t	0xAA

4.5.36 Get Periodic Message Mask

The *Get Periodic Message Mask* host command permits the host to read back the current periodic message mask from Cuprite. This command has no payload.

Get Periodic Message Mask Command

Cmd	Trans ID	Len	Frame
0x80	uint8_t	0x0000	0xAA

The payload of the response consists of a single uint_16 which contains the mask determining which periodic data messages will be broadcast. Each bit in the mask corresponds to a different periodic data message. If the bit is set to '1' then the message is broadcast, if the bit is cleared to '0' then the message is not broadcast. Bits in the mask are numbered from zero.

Bit	Mask	Corresponding periodic data message
10	0x0400	Periodic input/output power messages
11	0x0800	Periodic environmental monitoring messages

Get Periodic Message Mask Response

Cmd	Trans ID	Len	Capture mask	Frame
0x80	uint8_t	0x0002	uint16_t	0xAA

4.5.37 Set Input/Output Power Period

The *Set Input/Output Power Period* host command allows the host to adjust the frequency at which *Input/Output Power* messages are generated. These messages will be sent to the host over the asynchronous data channel, provided bit 10 of the periodic message mask has been set using the *Set Periodic Message Mask* host command.

Input/Output Power Messages contain the following information:

1. An estimate of the current output power of the signal generator
2. An estimate of the current input power on the selected receiver port
3. An indication of any overload conditions on either the signal generator or the receiver
4. The current setting of the receiver frontend attenuator
5. The current setting of the transmitter attenuator

The payload consists of a single uint32_t. This sets the message reporting interval in units of 10 ms. The minimum reporting period is 10 ms and the maximum reporting interval is 10 seconds. If this field is zero then no message will be generated.

Set Input/Output Power Period Command

Cmd	Trans ID	Len	Period	Frame
0x20	uint8_t	0x0004	uint32_t	0xAA

The Set Input/Output Power Period Response is a copy of the command packet.

Set Input/Output Power Period Response

Cmd	Trans ID	Len	Period	Frame
0x20	uint8_t	0x0004	uint32_t	0xAA

4.5.38 Get Input/Output Power Period

The *Get Input/Output Power Period* host command allows the host to read back the frequency at which *Input/Output Power* messages are generated. These messages will be sent to the host over the asynchronous data channel, provided bit 10 of the periodic mask has been set using the *Set Periodic Message Mask* host command.

Input/Output Power Messages contain the following information:

1. An estimate of the current output power of the signal generator
2. An estimate of the current input power on the selected receiver port
3. An indication of any overload conditions on either the signal generator or the receiver
4. The current setting of the receiver frontend attenuator
5. The current setting of the transmitter attenuator

This command has no payload.

Get Input/Output Power Period Command

Cmd	Trans ID	Len	Frame
0xA0	uint8_t	0x0000	0xAA

The *Get Gain Period Response* payload consists of a single uint32_t. This contains the message reporting interval in units of 10 ms. The minimum reporting period is 10 ms and the maximum reporting interval is 10 seconds. If this field is zero then no messages will be generated.

Get Input/Output Power Period Response

Cmd	Trans ID	Len	Period	Frame
0xA0	uint8_t	0x0004	uint32_t	0xAA

4.5.39 Set Environmental Period

The *Set Environmental Period* host command allows the host to adjust the frequency at which *Environmental Messages* are generated. These messages will be sent to the host over the asynchronous data channel, provided bit 11 of the periodic message mask has been set using the *Set Periodic Message Mask* host command.

The *Environmental Messages* contain information relating to:

1. The current FPGA die temperature, as well as the maximum and minimum values observed.
2. The current FPGA voltages, as well as the maximum and minimum values observed.
3. The current fan speed.
4. Lock status for PLLs.
5. Last detected internal error.

The payload consists of a single uint32_t. This sets the message reporting interval in units of 100 ms. The minimum reporting period is 100 ms and the maximum reporting period is 10 seconds. If this value is set to zero then no *Environmental Messages* will be generated.

Set Environmental Period Command

Cmd	Trans ID	Len	Period	Frame
0x21	uint8_t	0x0004	uint32_t	0xAA

The *Set Environmental Period Response* is a copy of the command packet.

Set Environmental Period Response

Cmd	Trans ID	Len	Period	Frame
0x21	uint8_t	0x0004	uint32_t	0xAA

4.5.40 Get Environmental Period

The *Get Environmental Period* host command allows the host to read back the frequency at which *Environmental Messages* are generated. These messages will be sent to the host over the asynchronous data channel, provided bit 11 of the periodic message mask has been set using the *Set Periodic Message Mask* host command.

The *Environmental Messages* contain information relating to:

1. The current FPGA die temperature, as well as the maximum and minimum values observed.
2. The current FPGA voltages, as well as the maximum and minimum values observed.
3. The current fan speed.
4. Lock status for PLLs.
5. Last detected internal error.

This command has no payload.

Get Environmental Period Command

Cmd	Trans ID	Len	Frame
0xA1	uint8_t	0x0000	0xAA

The *Get Environmental Period Response* payload consists of a single uint32_t. This contains the message reporting interval in units of 100 ms. The minimum reporting period is 100 ms and the maximum reporting period is 10 seconds. If this value is set to zero then no *Environmental Messages* will be generated.

Get Environmental Period Response

Cmd	Trans ID	Len	Period	Frame
0xA1	uint8_t	0x0004	uint32_t	0xAA

4.5.41 Get Time

The *Get Time* command allows the host to retrieve the *TLF3000* system time. It has no payload.

Get Time Command

Cmd	Trans ID	Len	Frame
0x8F	uint8_t	0x0000	0xAA

The response to the *Get Time* command contains the *TLF3000* system time. The payload consists of a single uint32_t which contains the number of elapsed seconds since the Unix epoch of 1970-01-01 00:00:00 + 0000 (UTC).

Get Time Response

Cmd	Trans ID	Len	Time	Frame
0x8F	uint8_t	0x0004	uint32_t	0xAA

4.5.42 Get Features

The *Get Features* command allows the host to retrieve information about the Cuprite application.

Get Features Command

Cmd	Trans ID	Len	Frame
0xC0	uint8_t	0x0000	0xAA

The *Get Features Response* contains the following fields within its payload:

1. The *Application* which responded to the *Get Features Command* (uint32_t). For Cuprite, this field will be 14.
2. *FPGA version* indicating the FPGA version being used by the application
3. *FPGA date* indicating when the FPGA was built (uint32_t)
4. *FPGA time* indicating when the FPGA was built (uint32_t)
5. *FPGA feature* indicating the features supported by the FPGA (uint32_t)
6. *Linux & Cuprite version* numbers (null terminated string)

Get Features Response

Cmd	Trans ID	Len	Application	FPGA version	FPGA date	FPGA time
0xC0	uint8_t	uint16_t N	uint32_t = 1	uint32_t	uint32_t	uint32_t

FPGA feature	Linux & Cuprite version	Frame
uint32_t	char[N-20]	0xAA

4.5.43 Get Friendly Name

The *Get Friendly Name* command allows the host read back the friendly name which has been associated with the *TLF3000* unit. At manufacture this name is set to 'TLF3000'. It can subsequently be changed using the *Set Friendly Name* command which must be issued to *TLF3000*, not Cuprite.

This command has zero payload.

Get Friendly Name Command

Cmd	Trans ID	Len	Frame
0xE7	uint8_t	0x0000	0xAA

The response to the *Get Friendly Name* command contains the friendly name as a null terminated string within its payload. The maximum length of the string, including null termination, is 256 bytes.

Get Friendly Name Response

Cmd	Trans ID	Len	Friendly name	Frame
0xE7	uint8_t	uint16_t N	char[N]	0xAA

4.5.44 Get Serial Number

The *Get Serial Number* command allows the host to retrieve the manufacturer's serial number of the *TLF3000* unit. This serial number is also visible as the USB serial number. This command has zero payload.

Get Serial Number Command

Cmd	Trans ID	Len	Frame
0xE8	uint8_t	0x0000	0xAA

The response to the *Get Serial Number* command contains the serial number as a uint32_t within its payload. Only the 6 LSBs of the serial number are valid.

Get Serial Number Response

Cmd	Trans ID	Len	Serial Number	Frame
0xE8	uint8_t	0x0004	uint32_t	0xAA

4.5.45 Get Manufacturer String

The *Get Manufacturer String* command allows the host read back the manufacturer's string that was entered into the unit during production. This string contains the name of the manufacturer as well as the date and time of manufacture.

This command has zero payload.

Get Manufacturer String Command

Cmd	Trans ID	Len	Frame
0xEE	uint8_t	0x0000	0xAA

The response to the *Get Manufacturer String* command contains the manufacturer's string as a null terminated string within its payload. The maximum length of the string, including the null termination, is 256 bytes.

Get Manufacturer String Response

Cmd	Trans ID	Len	Manufacturer's string	Frame
0xEE	uint8_t	uint16_t N	char[N]	0xAA

4.5.46 Get IP Address

The *Get IP Address* command allows the host to read back the IP address configuration data held in Flash as well as the current IP address. This command has zero payload.

Get IP Address Command

Cmd	Trans ID	Len	Frame
0xFC	uint8_t	0x0000	0xAA

The response to the *Set IP Address* command contains the following seven uint32_t fields:

1. *DHCP*: If this field is non-zero, then DHCP will be used to attempt to obtain an IP address next time the *TLF3000* is booted. If this field is zero, then static IP address values in the subsequent three fields will be used to next time *TLF3000* is booted. This field does not indicate whether DHCP is currently operating.
2. *IP Addr Flash*: The static IP address stored in Flash. This will be used on the next boot if DHCP is disabled.
3. *Subnet Flash*: The subnet mask stored in Flash. This will be used on the next boot if DHCP is disabled.
4. *Broadcast Flash*: The default broadcast address stored in Flash. This will be used on the next boot if DHCP is disabled.
5. *IP Addr Curr*: The current IP address in use.
6. *Subnet Curr*: The current subnet mask in use.
7. *Broadcast Curr*: The current default broadcast address in use.

Get IP Address Response

Cmd	Trans ID	Len	DHCP	IP Addr Flash	Subnet Flash
0xFC	uint8_t	uint16_t 28	uint32_t	uint32_t	uint32_t

Broadcast Flash	IP Addr Curr	Subnet Curr	Broadcast Curr	Frame
uint32_t	uint32_t	uint32_t	uint32_t	0xAA

4.5.47 Shutdown

The *Shutdown* command causes the unit to power down. The host connection will be torn down. This command has no payload.

Shutdown Command

Cmd	Trans ID	Len	Frame
0xFD	uint8_t	0x0000	0xAA

The *Shutdown* command does not send a response.

4.5.48 Exit

The *Exit* command causes the Cuprite application to terminate and return control to the *TLF3000* supervisor program. Any subsequent host commands will be processed by *TLF3000*, not Cuprite. This command has no payload.

Exit Command

Cmd	Trans ID	Len	Frame
0xFE	uint8_t	0x0000	0xAA

The *Exit* response is merely a copy of the command packet.

Exit Response

Cmd	Trans ID	Len	Frame
0xFE	uint8_t	0x0000	0xAA

4.5.49 Reset

The *Reset* command causes the unit to reboot to a known clean state. The USB host connection will be torn down. This command has no payload.

Reset Command

Cmd	Trans ID	Len	Frame
0xFF	uint8_t	0x0000	0xAA

The *Reset* response is merely a copy of the command packet.

Reset Response

Cmd	Trans ID	Len	Frame
0xFF	uint8_t	0x0000	0xAA

4.6 Error Handling

4.6.1 Internal Error Response

A host command response packet normally contains the same *Cmd* field as the host command packet. However, if an internal error has been detected prior to receiving the host command, then the *Cmd* field in the response packet is set to 0x7E. The *Internal Error Response* packet contains various information about the internal state of TLF3000.The host command which initiated the response is not executed unless it is a *Reset* command.

When an internal error is detected, an error message is placed on the error queue.

Internal Error Response

Cmd	Trans ID	Len	Fan	Flags	Now	Min	Max
0x7E	uint8_t	0x004C	uint16_t	uint16_t	int16_t	int16_t	int16_t

…….
{Now, Min, Max} repeated x 6

Rdgs[0]	…	Rdgs[7]	DacTemp	DacFlags	Frame
uint16_t	…	uint16_t	int16_t	uint16_t	0xAA

The *Fan* field is a uint16_t containing the current fan speed. Valid values are in the range 0 to 4095. Higher values correspond to faster fan speeds. 0 indicates the fan is not running.

The *Flags* field indicates the lock state of various PLLs and also whether an internal errors has been detected. If an internal error has been detected, then the reason can be found in the *Err* field. The bits in this fields are numbered from zero and their meanings are:

Bit	Mask	Meaning
0	0x0001	'1' indicates the clock generator PLL is locked, '0' indicates that the PLL is unlocked
1	0x0002	'1' indicates the RF synthesizer PLL is locked, '0' indicates that the PLL is unlocked
2	0x0004	'1' indicates that an internal error has been detected, '0' indicates no error detected
3	0x0008	'1' indicates that the ADC clock is present, '0' indicates no ADC clock
4	0x0010	'1' indicates the FPGA PLL is locked, '0' indicates that the PLL is unlocked
5	0x0020	'1' indicates loss of external clock, '0' indicates external clock present
6	0x0040	'1' indicates locked to external clock, '0' indicates locked to internal TCXO
8	0x0100	'1' indicates 1.5V digital supply is good, '0' indicates the supply is bad
9	0x0200	'1' indicates 1.8V digital supply is good, '0' indicates the supply is bad
10	0x0400	'1' indicates 3.3V digital supply is good, '0' indicates the supply is bad
11	0x0800	'1' indicates 3.8V analogue supply is good, '0' indicates the supply is bad
12	0x1000	'1' indicates 5.0V analogue supply is good, '0' indicates the supply is bad
13	0x2000	'1' indicates 3.3V fan supply is good, '0' indicates the supply is bad

There are 7 triplets of the *{Now, Min, Max}* fields. Each of these fields is an int16_t. The *Now* field represents the current value of a temperature or voltage, the *Min* field shows the minimum value observed since power up and the *Max* field the maximum value observed since power up. The triplets correspond to the following temperatures and voltages:

Triplet	Quantity	Units
0	Temperature	C
1	Vcc_aux	mV
2	Vcc_bram	mV
3	Vcc_int	mV
4	Vcc_oddr	mV
5	Vcc_paux	mV
6	Vcc_pint	mV

The *Err* field is a uint16_t containing an error code for the last reported internal error.

The *Rdgs* fields report the results of the FPGA internal ADC sampling the 8 multiplexed analogue lines. These contain information on the temperature of various devices on the PCB.

The *DacTemp* field is an int16_t containing the signal generator DAC temperature in degrees Celsius. This field is only valid if the signal generator is active.

The *DacFlags* field is a uint16_t containing the status flags read back from the signal generator DAC. This field should be zero if the DAC is operating correctly. This field is only valid if the signal generator is active.

4.6.2 Error Response

A host command response packet normally contains the same *Cmd* field as the host command packet. However, if an error occurs, then the *Cmd* field in the response packet is set to 0x7F.The host command which initiated the response is not executed unless it is a reset command.

Error Response

Cmd	Trans ID	Len	Frame
0x7F	uint8_t	0x0000	0xAA

Reasons for an *Error Response* packet include:

1. Incorrect payload length in the host command.
2. The host command is not supported by the running application.
3. The Cuprite application is in a mode which does not support the command

When a command error occurs, an error message is placed on the error queue.

4.6.3 Reading the error queue

Whenever a command fails or an error is detected, an error is placed on the error queue. The error queue is a FIFO capable of holding 64 error messages. The oldest message on the queue can be retrieved using the *Get Error Message* command. Issuing this command will also remove the message from the error queue. The *Get Error Message* command has no payload.

Get Error Message

Cmd	Trans ID	Len	Frame
0xF3	uint8_t	0x0000	0xAA

The *Get Error Message command response* message payload consists of a single ASCII string containing a description of the error which was detected. The string is NULL terminated.

Get Error Message Response

Cmd	Trans ID	Len	Error message	Frame
0xF3	uint8_t	uint16_t N	char[N]	0xAA

4.6.4 Clearing the error queue

The error queue can be cleared using the *Clear Error Queue* command. This command has no payload.

Clear Error Queue

Cmd	Trans ID	Len	Frame
0xF4	uint8_t	0x0000	0xAA

The Clear Error Queue command response message is a copy of the Clear Error Queue command message.

Clear Error Queue Response

Cmd	Trans ID	Len	Frame
0xF4	uint8_t	0x0000	0xAA

5 Python Interface

5.1 Overview

Support is provided for driving the TLF3000 directly from Python on a Windows platform. This support is currently available for Pyhton 2.7 and 3.6. Both 32 bit and 64 bit versions are available.

The python library which is provided closely mimics the native device interface. It is therefore advisable to reference the description of the native interface to obtain a complete understanding of the various commands.

5.1.1 Prerequistes

To import the Cuprite module, include the location of the appropriate Cuprite library (ie Python 2.7 or 3.6, 32 bit or 64 bit) on the current path. Then import the following:

```
sys . path . append ('C:/Users/Tim/Documents/qtapps/build-PyCuprite_MSVC2017_32bit-Default')
from cuprite import MorephDevice, MorephSearch, MorephInterface, CupriteInterface, vector_uchar, vector_float
```

5.1.2 Connecting to the TLF3000

To connect to the TLF3000 it is first necessary to create a search engine by calling MorephSearch..get(). Having obtained a search engine, the following operations should be performed:

1. Attach a callback to the search engine by invoking the set_callback() method. The callback will process the TLF3000s that are discovered by the search engine.
2. Start the search engine by invoking the start() method.

Whenever a TLF3000 is discovered, the callback will be entered with a handle to the device and a flag to indicate whether it was discovered on USB or Ethernet. In the example code below, the callback appends the device to a list if it is a USB device.

In the example program, the main thread monitors the contents of the list. Whenever a new entry is placed on the list, the main thread obtains at transport by invoking getTransport() and from this derives an interface to the device by invoking the MorephInterface() method. Having obtained an interface to the device, the friendly name and serial number are interrogated. If these satisfy certain search criteria, then the required device has been found and opened, otherwise the entry is discarded from the list.

Once the required device has been found, the search engine is stopped by invoking the stop() method.

```
sys.path.append('C:/Users/Tim/Documents/qtapps/build-PyCuprite_MSVC2017_32bit-Default')

from cuprite import MorephDevice, MorephSearch, MorephInterface, CupriteInterface, vector_uchar, vector_float

MorephName = "My first Moreph"
MorephSerialNumber = 0

"""
Connect to first device found which matches either MorephName or MorephSerialNumber
"""

def connect():
    moreph = []

    ms = MorephSearch.get()

    def callback(m):
        if (m.type == MorephDevice.USB):
            moreph.append(m)

    ms.set_callback(callback)
    ms.start()

    found = False
    while True:
        while len(moreph):
            try:
                transport = moreph[0].getTransport();
                mi = MorephInterface(transport)
                name = mi.getFriendlyName()
                sn = mi.getSerialNumber()
                print( "Found" , name , sn )
                if( (name == MorephName) | (sn == MorephSerialNumber) ):
                    found = True
                    break
                del moreph[0]
                found = True
                break
            except:
                del moreph[0]
        if found:
            break
        sleep(0.1)

    ms.stop()

    print("Opening Moreph" )

    return transport
```

5.1.3 Launching the Cuprite application

The Cuprite application is launched by calling CupriteInterface(). This returns an interface to the Cuprite application.

A callback should be attached to handle asynchronous data from the TLF3000. This is done uiing the setDataCallback method.

```
transport = connect()                              # Connect to the Moreph unit
cuprite = CupriteInterface(transport)              # Launch Cuprite application
transport.setDataCallback(dataCallback,False)      # Register asynch callback
```

5.1.4 Handling asynchronouos data

Whenever an asynchronous message is received from the TLF3000, the asynchronous data callback will be entered. The data passed to the callback is described in the native language interface section of this document. In the example below, the data is unpacked to determine its length and type. Having determined the message type, it is dispatched to an appropriate routine.

```
"""
callbacks for handling aux messages
"""

def envData( d ):
    print( "Environmental data" )

def pwrData( d ):
    print( "Input/output power data" )

"""
Main callback for handling messages on the asynch data channel
These are then dispatched to the next level callback handler
"""

pktType = {
    193: envData,
    194: pwrData}

pkthdr = namedtuple('hdr', 'n nMsb typ')
pkthdrbin = '<HBB'

def dataCallback(d):
    s = pkthdr(*struct.unpack(pkthdrbin, d[0:4]))
    pktType.get( s.typ )(d)
```

5.1.5 Handling errors

Many of the calls to the Cuprite library return a flag to indicate wtheter the call was successful or not. When an error occurs the cause of the error can be read back by using the Cuprite interface DispErrors() method. Each invocation of this method will remove the oldest error from the Cuprite error queue and return it to the caller. When not more messages are available, an empty string is returned.

```
"""
Routine for retrieving errors messages
"""

def getError(s):
    msg = s.DispErrors()
    while( msg != "" ):
        print( msg )
        msg = s.DispErrors()
```

5.1.6 Closing down

The Cuprite application can be shutdown by invoking the Cuprite interface exitApp() method..

```
cuprite.exitApp()                    # Close the Cuprite application
```

5.2 Examples

5.2.1 Programming the signal generator

This example code demonstrates how to program the various signal generator sources and then turn on the signal generator output.

The sequence of operations is as follows:

1. A connection is made to the wanted *TLF3000* unit using the connection code described in Connecting to the TLF3000
1. The Cuprite application is launched using the code described in Launching the Cuprite application
2. A callback for asynchronous messages is registered using the code described in Handling asynchronous data
3. The operating mode is set to signal generator mode using setMode
4. The wanted signal is programmed using setWanted. At this stage there is not output from the signal generator since it has not been started.
5. An interferer signal is programmed using setInterferer
6. Three in-band CW blockers are programmed using setInBandCW
7. An AWGN source is programmed using setAWGN
8. An out-of-band CW blocker is programmed using setOutOfBandCW
9. The signal generator is started by callingstartExec . It is only at this point that the signal generator starts outputting power.
10. After 30 seconds the signal generator is stopped by calling stopExec. This does not affect the programming which has previously been performed.

```
# -*- coding: utf-8 -*-
"""
Created on Tue Apr 11 19:29:36 2017

@author: tim
"""
from __future__ import print_function
import sys, struct, array
from collections import namedtuple
from time import sleep

sys.path.append('C:/Users/Tim/Documents/qtapps/build-PyCuprite_MSVC2017_32bit-Default')

from cuprite import MorephDevice, MorephSearch, MorephInterface, CupriteInterface, vector_uchar

MorephName = "My first Moreph"
MorephSerialNumber = 0

"""
Connect to first device found which matches either MorephName or MorephSerialNumber
"""

def connect():
    moreph = []

    ms = MorephSearch.get()

    def callback(m):
        if (m.type == MorephDevice.USB):
            moreph.append(m)

    ms.set_callback(callback)
    ms.start()

    found = False
    while True:
        while len(moreph):
            try:
                transport = moreph[0].getTransport();
                mi = MorephInterface(transport)
                name = mi.getFriendlyName()
```

```
                sn = mi.getSerialNumber()
                print( "Found" , name , sn )
                if( (name == MorephName) | (sn == MorephSerialNumber) ):
                    found = True
                    break
                del moreph[0]
                found = True
                break
            except:
                del moreph[0]
        if found:
            break
        sleep(0.1)

    ms.stop()

    print("Opening Moreph" )

    return transport

"""
callbacks for signal analyser
"""

def stopOnFail( d ):
    print( "Signal analyser stopped because of test limit failure" )

"""
callbacks for packet sniffing
"""

pktPayload = namedtuple('payload', 'hdr timeMSB timeLSB chan rssi count')
pktpayloadbin = '<LLLBbB'
pktLQI = namedtuple( 'lqi' , 'lqi' )
pktlqibin = '<B'

def gotPkt( d ):
    s = pktPayload(*struct.unpack(pktpayloadbin, d[0:15]))
    print( "Received packet" )
    print( "Timestamp : %08X%08X" % ( s.timeMSB , s.timeLSB ) )
    print( "Channel   : " , s.chan )
```

```
    print( "RSSI      : " , s.rssi )
    print( "Octets    : " , s.count & 0x7F )
    n = 4 + 8 + 3 + (s.count & 0x7F)
    n = 4 * ( (n+3) / 4 )
    lqi = d[n] & 0x7F
    crc = (d[n] & 0x80) != 0
    print( "LQI       : " , lqi )
    print( "CRC       : " , crc )

def pkSpectrum( d ):
    return

"""
callbacks for handling aux messages which should not occur
"""

def envData( d ):
    print( "Environmental data" )

def pwrData( d ):
    print( "Input/output power data" )

"""
Main callback for handling messages on the asynch data channel
These are then dispatched to the next level callback handler
"""

pktType = {
    193: envData,
    194: pwrData}

pkthdr = namedtuple('hdr', 'n nMsb typ')
pkthdrbin = '<HBB'

def dataCallback(d):
    s = pkthdr(*struct.unpack(pkthdrbin, d[0:4]))
    pktType.get( s.typ )(d)

"""
Routine for retrieving errors messages from Moreph
"""
```

```
def getError(s):
    msg = s.DispErrors()
    while( msg != "" ):
        print( msg )
        msg = s.DispErrors()

"""
Main script
"""

if __name__ == '__main__':
    global done
    done = False

    transport = connect()                                   # Connect to the Moreph unit
    cuprite = CupriteInterface(transport)                   # Launch the Cuprite application
    transport.setDataCallback(dataCallback,False)           # Register callback for asynch data channel

    if( not cuprite.setMode( 1 ) ):                         # Enter signal generator mode
        getError(cuprite)

    if( not cuprite.setExtDioVolts( False ) ):              # Turn on 3.3V IO supply from Moreph
        getError(cuprite)


                                                            # Turn on signal generator
                                                            # Use channel 11
                                                            # Output power of -55dBm
                                                            # No DIO control
                                                            # Packet count of 0 means repeat for ever
                                                            # Packet interval of 1ms
                                                            # Packet timing jitter of 0
                                                            # Turn packet timing jitter off
                                                            # Ramp time of 10us
                                                            # Front porch of 1us
                                                            # Back porch of 1us
                                                            # No carrier frequency offset
                                                            # Zero carrier drif magnitude
                                                            # Zero carrier drift rate
                                                            # Zero symbol timing error
                                                            # 0dB IQ imbalance
```

```
                                                    # 0 degrees IQ quadrature error
                                                    # Np chip errors

payload = vector_uchar( array.array( "B" , ( 0 , 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 ) ) )

if( not cuprite.setWanted( True, 11, -55, 0, 0, 1000, 0, 0, 10, 1, 1, 0, 0, 0, 0, 0, 0, payload ) ):
    getError(cuprite)

                                                    # Turn on interferer
                                                    # Not continuous but packetised
                                                    # Output power of -50dBm
                                                    # No DIO control
                                                    # 127 octet payload
                                                    # Frequency of 2447.5MHz
                                                    # 4.5ms between start of packets
if( not cuprite.setInterferer( True, False, -50, 0, 127, 2447.5e6, 4500 ) ):
    getError(cuprite)

                                                    # Turn on a CW signal
                                                    # Use CW source #0
                                                    # Output power of -33dBm
                                                    # Freqeuncy of 2456.1MHz
                                                    # On period of 1us
                                                    # Off period of 0us, ie no bursting
if( not cuprite.setInBandCW( True, 0, -33, 2456.1e6, 1, 0 ) ):
    getError(cuprite)

                                                    # Turn on another CW signal
                                                    # Use CW source #1
                                                    # Output power of -46dBm
                                                    # Freqeuncy of 2475MHz
                                                    # On period of 1ms
                                                    # Off period of 1ms
if( not cuprite.setInBandCW( True, 1, -46, 2475.0e6, 1000, 1000 ) ):
    getError(cuprite)

                                                    # Turn on 3rd CW signal
                                                    # Use CW source #2
                                                    # Output power of -62dBm
                                                    # Freqeuncy of 2431MHz
                                                    # On period of 10ms
```

```
                                                        # Off period of 1ms
if( not cuprite.setInBandCW( True, 2, -62, 2431.0e6, 10000, 1000 ) ):
    getError(cuprite)
                                                        # Turn on thte AWGN generator
                                                        # -70dBm/MHz
                                                        # On period of 500ms
                                                        # Off period of 1s
if( not cuprite.setAWGN( True, -70, 500.0e3, 1.0e6 ) ):
    getError(cuprite)

                                                        # Turn on the out-of-band CW signal
                                                        # -35dBm output power
                                                        # 500MHz frequency
if( not cuprite.setOutOfBandCW( True, -35, 500 ) ):
    getError(cuprite)

if( not cuprite.startExec() ):                          # Start signal generator
    getError(cuprite)

sleep( 30.0 )                                           # Wait for 30s so we can check on spec analyser

if( not cuprite.stopExec() ):                           # Stop signal generator
    getError(cuprite)

getError(cuprite)

cuprite.exitApp()                                       # Close the Cuprite applications

cuprite = 0
transport = 0
```

5.2.2 Analysing waveforms and obtaining data using the signal analyser

This example code demonstrates how to program the signal analyser and then to extract data from it. The example explores only a subset of the data which can be read back.

The sequence of operations is as follows:

1. A connection is made to the wanted *TLF3000* unit using the connection code described in Connecting to the TLF3000
2. The Cuprite application is launched using the code described in Launching the Cuprite application. A callback for asynchronous messages is registered using the code described in Handling asynchronous data
3. The operating mode is set to signal generator mode using setMode
4. The receiver frontend attenuation is set using setRxAtten
5. The receiver port to be used is set using setRxPort
6. The signal analayser is programmed using the setSigAna command
7. The signal analyser is started using the startExec command.
8. After 1 second the signal analyser is halted using stopExec. It is not actually neceassry to stop the signal analyser running before reading back results.
9. Multiple calls are made to getTable to recover summary data which is printed out.
10. A plotvsChan call is made to determine how the received power varied across channels.
11. A call to plotHist is made so that a histogram of EVM readings can be displayed.
12. The constellation of the last packet received is retrieved by calling plotConstellation and the data plotted.
13. The FM deviation throughout the last packet is obtained by calling plotFM and then plotted.
14. The peak and average spectra can be recovered by calling plotPSD.
15. getBits is called to return the demodulated octets within the last received packet.

```
# -*- coding: utf-8 -*-
"""
Created on Tue Apr 11 19:29:36 2017

@author: tim
"""
from __future__ import print_function
import sys, struct, array
from collections import namedtuple
import numpy as np
import matplotlib.pyplot as plt
from time import sleep

sys.path.append('C:/Users/Tim/Documents/qtapps/build-PyCuprite_MSVC2017_32bit-Default')

from cuprite import MorephDevice, MorephSearch, MorephInterface, CupriteInterface, vector_uchar,
vector_float

MorephName = "My first Moreph"
MorephSerialNumber = 0

"""
Connect to first device found which matches either MorephName or MorephSerialNumber
"""

def connect():
    moreph = []

    ms = MorephSearch.get()

    def callback(m):
        if (m.type == MorephDevice.USB):
            moreph.append(m)

    ms.set_callback(callback)
    ms.start()

    found = False
    while True:
        while len(moreph):
            try:
```

```
                transport = moreph[0].getTransport();
                mi = MorephInterface(transport)
                name = mi.getFriendlyName()
                sn = mi.getSerialNumber()
                print( "Found" , name , sn )
                if( (name == MorephName) | (sn == MorephSerialNumber) ):
                    found = True
                    break
                del moreph[0]
                found = True
                break
            except:
                del moreph[0]
        if found:
            break
        sleep(0.1)

    ms.stop()

    print("Opening Moreph" )

    return transport

"""
routine for printing results tables
"""

PowerHeadings = (
        "Pavg",
        "Pk-Pavg"
        )

DriftHeadings = (
        "Initial Fc",
        "Min Fc",
        "Max Fc",
        "Drfit rate",
        "Symbol timing"
        )

EVMHeadings = (
```

```
        "EVM1000",
        "EVMrms",
        "EVMpk",
        "Phi rms",
        "Phi pk",
        "IQ err"
        )

PSDHeadings = (
        "PSDabs",
        "PSDrel"
        )

def printTable( headings , x ):
    print( "" )
    print( "%20s %10s %10s %10s %10s %10s" % ( "Quantity" , "# Pkts" , "Min" , "Max" , "Avg" , "Current" ))
    n = 0;
    for h in headings:
        print( "%20s " % h , end = "" )
        print( "%10d %10.1f %10.1f %10.1f %10.1f" % ( x[n], x[n+1], x[n+2], x[n+3], x[n+4] )  )
        n = n + 5
    print( "" )

"""
callbacks for signal analyser
"""

def stopOnFail( d ):
    print( "Signal analyser stopped because of test limit failure" )

"""
callbacks for packet sniffing
"""

pktPayload = namedtuple('payload', 'hdr timeMSB timeLSB chan rssi count')
pktpayloadbin = '<LLLBbB'
pktLQI = namedtuple( 'lqi' , 'lqi' )
pktlqibin = '<B'

def gotPkt( d ):
    s = pktPayload(*struct.unpack(pktpayloadbin, d[0:15]))
```

```
    print( "Received packet" )
    print( "Timestamp : %08X%08X" % ( s.timeMSB , s.timeLSB ) )
    print( "Channel   : " , s.chan )
    print( "RSSI      : " , s.rssi )
    print( "Octets    : " , s.count & 0x7F )
    n = 4 + 8 + 3 + (s.count & 0x7F)
    n = 4 * ( (n+3) / 4 )
    lqi = d[n] & 0x7F
    crc = (d[n] & 0x80) != 0
    print( "LQI       : " , lqi )
    print( "CRC       : " , crc )

def pkSpectrum( d ):
    return

"""
callbacks for handling aux messages which should not occur
"""

def envData( d ):
    print( "Environmental data" )

def pwrData( d ):
    print( "Input/output power data" )

"""
Main callback for handling messages on the asynch data channel
These are then dispatched to the next level callback handler
"""

pktType = {
    193: envData,
    194: pwrData}

pkthdr = namedtuple('hdr', 'n nMsb typ')
pkthdrbin = '<HBB'

def dataCallback(d):
    s = pkthdr(*struct.unpack(pkthdrbin, d[0:4]))
    pktType.get( s.typ )(d)
```

```
"""
Routine for retrieving errors messages from Moreph
"""

def getError(s):
    msg = s.DispErrors()
    while( msg != "" ):
        print( msg )
        msg = s.DispErrors()

"""
Main script
"""

if __name__ == '__main__':
    global done
    done = False

    transport = connect()                                   # Connect to the Moreph unit
    cuprite = CupriteInterface(transport)                   # Launch the Cuprite application
    transport.setDataCallback(dataCallback,False)           # Register callback for asynch data channel

    if( not cuprite.setMode( 2 ) ):                         # Enter signal analyser mode
        getError(cuprite)

    if( not cuprite.setRxAtten( 0 ) ):                      # Set rx frontend attenuation in units of 0.5dB
        getError(cuprite)

    if( not cuprite.setRxPort( 1 ) ):                       # Set the rx port to be tx/rx
        getError(cuprite)

    if( not cuprite.setExtDioVolts( False ) ):              # Turn on 3.3V IO supply from Moreph
        getError(cuprite)


                                                            # Program signal analyser to look at all
channels

                                                            # Program signal analyser to look at all packet
length groupds

                                                            # Set and RSSI threshold of -100dBm
                                                            # Collect all possible data
                                                            # Maximum oversampling rate
```

```
                                                                # 655us spectrm averaging
                                                                # Synchroniser threshold of 1488
                                                                # Don't stop on limit fail
    if( not cuprite.setSigAna( 0xFFFF , 0xFFFF, -100, 0x3F, 3, 6, 1488, False ) ):
        getError(cuprite)

    if( not cuprite.startExec() ):                              # Start signal analyser
        getError(cuprite)

    sleep( 1.0 )                                                # Wait for some data to be collected

    if( not cuprite.stopExec() ):                               # Stop signal analyser
        getError(cuprite)

    t = cuprite.getTable( 0, 0xFFFF, 0xFFFF )                   # Read back power results with no channel or
packet length filtering
    if( len(t) ):
        printTable( PowerHeadings , t )
    else:
        getError(cuprite)

    t = cuprite.getTable( 1, 0xFFFF, 0xFFFF )                   # Read back carrier/drift results with no
channel or packet length filtering
    if( len(t) ):
        printTable( DriftHeadings , t )
    else:
        getError(cuprite)

    t = cuprite.getTable( 2, 0xFFFF, 0xFFFF )                   # Read back EVM results with no channel or
packet length filtering
    if( len(t) ):
        printTable( EVMHeadings , t )
    else:
        getError(cuprite)

    t = cuprite.getTable( 3, 0xFFFF, 0xFFFF )                   # Read back PSD results with no channel or
packet length filtering
    if( len(t) ):
        printTable( PSDHeadings , t )
    else:
        getError(cuprite)
```

```
    t = cuprite.plotvsChan( 0, 0xFFFF )                    # Read back power vs channel
    if( len(t) ):
        for i in range(16):
            print( "Channel %2d : %.1f dBm" % ( i , t[i] ) )
    else:
        getError(cuprite)

    t = cuprite.plotHist( 7, 0xFFFF, 0xFFFF )              # Read back histogram of EVM values with no
packet filtering
    if( len(t) ):
        binW = 50.0 / 128
        plt.figure(1)
        plt.plot( np.arange(0.0,50.0,binW) , t )
        plt.xlabel( 'EVM (%)' )
        plt.ylabel( 'Occurrence' )
        plt.title( 'EVM histogram' )
        plt.grid( True )
        plt.show()
    else:
        getError(cuprite)

    t = cuprite.plotConstellation( 0xFFFF, 0xFFFF )        # Read back constellation plot for last packet
    n = len(t)
    if( n ):
        plt.figure(2)
        for i in range(0,n,2):
            plt.plot( t[i]  , t[i+1] , '.r' )
        plt.xlabel( 'I' )
        plt.ylabel( 'Q' )
        plt.title( 'Constellation' )
        plt.grid( True )
        plt.xlim( ( -1 , 1 ) )
        plt.ylim( ( -1 , 1 ) )
        plt.axis( 'equal' )
        plt.show()
    else:
        getError(cuprite)

    t = cuprite.plotFM( 0xFFFF, 0xFFFF )                   # Read back FM deviation for last packet
    n = len(t)
```

```
    if( n ):
        plt.figure(3)
        dt = 1.0 / cuprite.getPlotOversample()
        plt.plot( np.arange(n)*dt - 40.0  , t , 'r' )
        plt.xlabel( 'Time (us)' )
        plt.ylabel( 'Deviation (kHz)' )
        plt.title( 'FM deviation' )
        plt.grid( True )
        plt.show()
    else:
        getError(cuprite)

    t = cuprite.plotPSD( 0xFFFF )                              # Read back spectrum of last packet
    n = len(t)
    if( n ):
        plt.figure(4)
        df = 100.0 / 2048.0
        Fo = 102 * df + 2390
        plt.plot( np.arange(n/2)*df + Fo  , t[0:n/2] , 'r' , label = 'Average' )
        plt.plot( np.arange(n/2)*df + Fo  , t[n/2:n] , 'b' , label = 'Peak' )
        plt.xlabel( 'Frequency (MHz)' )
        plt.ylabel( 'Power (dBm/100kHz)' )
        plt.title( 'PSD' )
        plt.grid( True )
        plt.legend()
        plt.show()
    else:
        getError(cuprite)

    t = cuprite.getBits()
    if( len(t) ):
        for x in t:
            print( '%02X ' % x , end = "" )
        print( "" )
    else:
        getError(cuprite)

    getError(cuprite)

    cuprite.exitApp()                                         # Close the Cuprite applications
```

```
    cuprite = 0
    transport = 0
```

5.2.3 Running the packet sniffer

This example code demonstrates how to program the packet sniffer and process the packet data returned.

The sequence of operations is as follows:

1. A connection is made to the wanted *TLF3000* unit using the connection code described in Connecting to the TLF3000
2. The Cuprite application is launched using the code described in Launching the Cuprite application. A callback for asynchronous messages is registered using the code described in Handling asynchronous data
3. The operating mode is set to packet sniffer mode using setMode
4. The receiver frontend attenuation is set using setRxAtten
5. The receiver port to be used is set using setRxPort
6. The packet sniffer is programmed using the command setSniffer
7. The packet sniffer is started using the startExec command.

The received packets are processing in the gotPkt() callback.

```
# -*- coding: utf-8 -*-
"""
Created on Tue Apr 11 19:29:36 2017

@author: tim
"""
from __future__ import print_function
import sys, struct, array
from collections import namedtuple
from time import sleep

sys.path.append('C:/Users/Tim/Documents/qtapps/build-PyCuprite_MSVC2017_32bit-Default')

from cuprite import MorephDevice, MorephSearch, MorephInterface, CupriteInterface, vector_uchar

MorephName = "My first Moreph"
MorephSerialNumber = 0

"""
Connect to first device found which matches either MorephName or MorephSerialNumber
"""

def connect():
    moreph = []

    ms = MorephSearch.get()

    def callback(m):
        if (m.type == MorephDevice.USB):
            moreph.append(m)

    ms.set_callback(callback)
    ms.start()

    found = False
    while True:
        while len(moreph):
            try:
                transport = moreph[0].getTransport();
                mi = MorephInterface(transport)
                name = mi.getFriendlyName()
```

```
                sn = mi.getSerialNumber()
                print( "Found" , name , sn )
                if( (name == MorephName) | (sn == MorephSerialNumber) ):
                    found = True
                    break
                del moreph[0]
                found = True
                break
            except:
                del moreph[0]
        if found:
            break
        sleep(0.1)

    ms.stop()

    print("Opening Moreph" )

    return transport

"""
callbacks for signal analyser
"""

def stopOnFail( d ):
    print( "Signal analyser stopped because of test limit failure" )

"""
callbacks for packet sniffing
"""

pktPayload = namedtuple('payload', 'hdr timeMSB timeLSB chan rssi count')
pktpayloadbin = '<LLLBbB'
pktLQI = namedtuple( 'lqi' , 'lqi' )
pktlqibin = '<B'

def gotPkt( d ):
    s = pktPayload(*struct.unpack(pktpayloadbin, d[0:15]))
    print( "Received packet" )
    print( "Timestamp : %08X%08X" % ( s.timeMSB , s.timeLSB ) )
    print( "Channel   : " , s.chan )
```

```
    print( "RSSI      : " , s.rssi )
    print( "Octets    : " , s.count & 0x7F )
    n = 4 + 8 + 3 + (s.count & 0x7F)
    n = 4 * ( (n+3) / 4 )
    lqi = d[n] & 0x7F
    crc = (d[n] & 0x80) != 0
    print( "LQI       : " , lqi )
    print( "CRC       : " , crc )

def pkSpectrum( d ):
    return

"""
callbacks for handling aux messages which should not occur
"""

def envData( d ):
    print( "Environmental data" )

def pwrData( d ):
    print( "Input/output power data" )

"""
Main callback for handling messages on the asynch data channel
These are then dispatched to the next level callback handler
"""

pktType = {
    0: gotPkt,
    5: pkSpectrum,
    64: stopOnFail,
    193: envData,
    194: pwrData}

pkthdr = namedtuple('hdr', 'n nMsb typ')
pkthdrbin = '<HBB'

def dataCallback(d):
    s = pkthdr(*struct.unpack(pkthdrbin, d[0:4]))
    pktType.get( s.typ )(d)
```

```
"""
Routine for retrieving errors messages from Moreph
"""

def getError(s):
    msg = s.DispErrors()
    while( msg != "" ):
        print( msg )
        msg = s.DispErrors()

"""
Main script
"""

if __name__ == '__main__':
    global done
    done = False

    transport = connect()                                   # Connect to the Moreph unit
    cuprite = CupriteInterface(transport)                   # Launch the Cuprite application
    transport.setDataCallback(dataCallback,False)           # Register callback for asynch data channel

    if( not cuprite.setMode( 4 ) ):                         # Enter packet sniffer mode
        getError(cuprite)

    if( not cuprite.setRxAtten( 0 ) ):                      # Set rx frontend attenuation in units of 0.5dB
        getError(cuprite)

    if( not cuprite.setRxPort( 0 ) ):                       # Set the rx port to be Monitor In
        getError(cuprite)

    if( not cuprite.setExtDioVolts( False ) ):              # Turn on 3.3V IO supply from Moreph
        getError(cuprite)


                                                            # Monitor all channels
                                                            # Set and RSSI threshold of -120dBm
                                                            # Synchroniser threshold of 1488
    if( not cuprite.setSniffer( 0xFFFF, -120, 1488) ):
        getError(cuprite)

    if( not cuprite.startExec() ):                          # Start packet sniffer
```

```
        getError(cuprite)

    sleep( 30 )                                          # Capture packets for 30s

    if( not cuprite.stopExec() ):                        # Stop packet sniffer
        getError(cuprite)

    getError(cuprite)

    cuprite.exitApp()                                    # Close the Cuprite applications

    cuprite = 0
    transport = 0
```

5.3 Library reference

5.3.1 Overview

This section lists the Python library commands which are available for the Cuprite application. These broadly follow the same format as the native language commands, but there are minor alterations for ease of use, for example, powers are typically floating point numbers in dBm rather than integers in units of 0.1dBm.

5.3.2 setMode

setMode() sets the operating mode of the Cuprite application:

res = cuprite.setMode(mode)

mode defines the operating mode:

Value	Operating mode
1	Signal generator
2	Signal analyzer
4	Packet sniffer
Other	Illegal

If the operating mode is changed, then a stopExec command is implicitly executed by the application.

res will be set to False if the the command fails.

5.3.3 getMode

getMode() returns the current operarting mode of the Cuprite application:

mode = cuprite.getMode()

mode is the current operating mode. Possible returned values:

Value	Operating mode
1	Signal generator
2	Signal analyzer
4	Packet sniffer
-1	Command failed

5.3.4 startExec

startExec() instructs the Cuprite application to start executing the selected mode.

res = cuprite.startExec()

res will be set to False if the the command fails.

5.3.5 PauseExec

pauseExec() instructs the Cuprite application to pause execution of the selected mode. This command should only be issued subsequent to a startExec or resumeExec command.

res = cuprite.pauseExec()

res will be set to False if the the command fails.

5.3.6 resumeExec

resumeExec() instructs the Cuprite application to resume execution of the selected mode. This command should only be issued subsequent to a pauseExec command.

res = cuprite.resumeExec()

res will be set to False if the the command fails.

5.3.7 stopExec

stopExec() instructs the Cuprite application to stop execution of the selected mode. An implicit stopExec command is issued if the mode is changed using the setMode command.

res = cuprite.stopExec()

res will be set to False if the the command fails.

5.3.8 setHWCtrl

setHWCtrl determines whether Cuprite execution is under hardware control, ie execution is stopped and started by changes in the digital IO signals.

res = cuprite.setHWCtrl(on , start , pause , resume , stop)

on is a Boolean which determines whether execution is under hardware control. If True, then execution can be controlled by the digital IO lines.

start is a bit mask which defines the transitions required on the digital input lines to start execution. This mask is ignored if on is False.

Bits	Function
7:0	Defines which input lines are used to determine when execution should start. A '1' indicates that the corresponding input line is sampled. A '0' indicates that the corresponding input line is ignored.
15:8	Defines the state of the input lines prior to the start execution transition. A '1' indicates that the corresponding input line should be high, a '0' indicates that the corresponding input line should be low. Values for lines which are not included in the 7:0 mask are ignored.
23:16	Defines the state of the input lines after the start execution transition. A '1' indicates that the corresponding input line should be high, a '0' indicates that the corresponding input line should be low. Values for lines which are not included in the 7:0 mask are ignored.

pause is a bit mask which defines the transitions required on the digital input lines in order to pause execution. This mask is ignored if on is False.

Bits	Function
7:0	Defines which input lines are used to determine when execution should be paused. A '1' indicates that the corresponding input line is sampled. A '0' indicates that the corresponding input line is ignored.
15:8	Defines the state of the input lines prior to the pause execution transition. A '1' indicates that the corresponding input line should be high, a '0' indicates that the corresponding input line should be low. Values for lines which are not included in the 7:0 mask are ignored.
23:16	Defines the state of the input lines after the pause execution transition. A '1' indicates that the corresponding input line should be high, a '0' indicates that the corresponding input line should be low. Values for lines which are not included in the 7:0 mask are ignored.

resume is a bit mask which defines the transitions required on the digital input lines in order to move execution from the paused state back to the running state. This mask is ignored if on is False.

Bits	Function
7:0	Defines which input lines are used to determine when execution should be resumed. A '1' indicates that the corresponding input line is sampled. A '0' indicates that the corresponding input line is ignored.
15:8	Defines the state of the input lines prior to the resume execution transition. A '1' indicates that the corresponding input line should be high, a '0' indicates that the corresponding input line should be low. Values for lines which are not included in the 7:0 mask are ignored.
23:16	Defines the state of the input lines after the resume execution transition. A '1' indicates that the corresponding input line should be high, a '0' indicates that the corresponding input line should be low. Values for lines which are not included in the 7:0 mask are ignored.

stop is a bit mask which defines the transitions required on the digital input lines in order to stop execution. This mask is ignored if on is False.

Bits	Function
7:0	Defines which input lines are used to determine when execution should be stopped. A '1' indicates that the corresponding input line is sampled. A '0' indicates that the corresponding input line is ignored.
15:8	Defines the state of the input lines prior to the stop execution transition. A '1' indicates that the corresponding input line should be high, a '0' indicates that the corresponding input line should be low. Values for lines which are not included in the 7:0 mask are ignored.
23:16	Defines the state of the input lines after the stop eecutiom transition. A '1' indicates that the corresponding input line should be high, a '0' indicates that the corresponding input line should be low. Values for lines which are not included in the 7:0 mask are ignored.

res will be set to False if the the command fails.

5.3.9 setWanted

setWanted() programs the wanted signal generator when the Cuprite application is in signal generator mode. This command can only be executed when the Cuprite application is in signal generator mode. There will be no output from the signal generator until startExec is invoked.

res = cuprite.setWanted(on, chan, amp, dio, repeats, period, jitterMag, jitterTyp, ramp, before, after,
carrier, driftMag, driftRate, ppm, iqBal, iqPhi, cer, payload)

on is a Boolean to indicate whether the wanted signal generator should be started. If this flag set, then the wanted signal generator is started, otherwise the wanted signal generator is halted if it is already running.

chan is the RF channel number for the wanted signal. Valid channels are in the range 11 to 26.

amp is the power of the wanted signal in units of dBm. This should be in range -120 dBm to 0 dBm. The power has a resolution of 0.1dBm.

dio is a mask indicating which digital IO lines should be toggled high when the wanted signal is active. Bits 0 to 7 in the mask correspond to digital output lines 0 to 7. If a bit is set to '1', then the corresponding digital output line will be set high when a packet is being transmitted.

repeats is the number of packets to be transmitted. If this value is specified as 0, then the wanted signal transmitter will continue to generate packets until it is terminated. The maximum value which can be specified is 65535.

period is the interval between the start of one packet and the start of the subsequent packet in units of µs. The maximum period is 500ms. The period has a resolution of 100ns. If this is less than the packet duration, then it will forced to be a few µs greater than the packet duration.

jitterMag is the peak magnitude of the inter-packet timing jitter in units of µs. The maximum peak jitter is 1s. The peak jitter magnitude has a resolution of 100ns.

jitterTyp is the type of inter-packet timing jitter which will be applied. Possible values are:

0	Disable jitter
1	Uniform jitter
2	Gaussian jitter

ramp is the duration of the power ramp up and down at the start and end of the packet in units of µs. The ramp time must be in the range 1,1µs to 10 µs. The ramp time has a resolution of 100ns.

before is the length of unmodulated carrier between the end of the power ramp up and the start of the preamble in units of µs. The length of unmodulated carrier must be in the range 0µs to 25µs, The length of unmodulated carrier has a resolution of 100ns.

after is the length of unmodulated carrier between the end of the FCS and the start of the power ramp down in units of µs. The length of unmodulated carrier must be in the range 0µs to 10µs. The length of unmodulated carrier has a resolution of 100ns.

carrier is the carrier frequency offset in kHz. The carrier frequency offset must be in the range ±250kHz. The carrier frequency offset has a resolution of 100Hz.

driftMag is the magnitude of the sinusoidal carrier frequency drift which is superimposed on the packet in units of kHz. This may be positive or negative. The drift magnitude must be in the range ±70kHz. The drift magnitude has a resolution of 100Hz.

driftRate is the frequency of the sinuisoidal carrier frequency drift which is superimposed on the packet in units of Hz. The maximum drift rate is 2400Hz. The drift rate has a resolution of 0.1Hz.

ppm is the symbol timing error in ppm. The symbol timing error must be in the range ±100ppm. The symbol timing error has a resolution of 0.1ppm.

iqBal is the amplitude imbalance between the I and Q channels in dB. The amplitude imbalance must be in the range 0dB to 3dB. The IQ amplitude imbalance has a resolution of 0.001dB.

iqPhi is the quadrature error between the I and Q channels in degrees. The quadrature errors must be in the range ±10°. The quadrature error has a resolution of 0.001°.

cer is the chip error rate. If this parameter is not zero, then the modulator will randomly introduce chip errors. These errors are uniformly distributed. The maximum chip error rate is 0.0625 (6.25%) and the resolution of the chip error rate is 0.000001.

payload is a vector of uint8_t containg the packet payload. The preamble, SFD and PHR are automatically added by the modulator and need not be included in the payload. The FCS, if present, must be included in the payload.

res will be set to False if the command fails.

5.3.10 setInterferer

setInterferer() programs the modulated interferer signal generator when the Cuprite application is in signal generator mode. This command can only be executed when the Cuprtie application is in signal generator mode. There will be no output from the signal generator until startExec is invoked. The modulated interferer always uses a PRBS15 as its data source.

res = cuprite.setInterferer(on , cont , pwr , dio, octets, freq, period)

on is a Boolean to indicate whether the modulated interferer signal generator should be started. If this flag is set, then the modulated interferer signal generator is started, otherwise the modulated interferer signal generator is halted if it is already running.

cont is a Boolean to indicate whether the interferer is a continuous transmissison or a packetised transmission. If the transmission is continuous, then the octet and period parameters are ignored.

pwr is the power of the modulated interferer signal in dBm. Valid range is -120dBm to 0dBm. The power resolution is 0.1dBm.

dio is a mask indicating which digital IO lines should be toggled high when the interferer signal is active. Bits 0 to 7 in the mask correspond to digital output lines 0 to 7. If a bit is set to '1', then the corresponding digital output line will be set high when the interferer is active.

octets is the number of octets in the packet payload if the interefer is packetised. If the interferer is not packetised, then this parameter is ignored. The number of octets must be in the range 0 to 127.

freq is the carrier frequency for the interferer signal in Hz. Valid range is 2,395,000,000Hz to 2,485,000,000Hz inclusive. The frequency resolution is 1Hz.

period is the interval between the start of packets in units of seconds, if the interferer is packetised. If the interferer is not packetised, then this parameter is ignored. The maximum interval is 0.5s.

res will be set to False if the the command fails.

5.3.11 setInBandCW

setInBandCW() programs the in-band CW signal generator when the Cuprite application is in signal generator mode. This command can only be executed when the Cuprite application is in signal generator mode. There will be no output from the signal generator until startExec is invoked.

res = cuprite.setInBandCW(on , cw, pwr, freq , onPeriod, offPeriod)

on is a Boolean to indicate whether the in-band CW generator should be turned on or off. If set, then the in-band CW generator is turned on, otherwise the in-band CW generator is turned off.

cw indicates which of the 3 in-band CW sources is to be programmed. Valid values for this parameter are 0, 1 and 2.

pwr is the power of the CW signal in dBm. Valid range is -120dBm to 0dBm. The power resolution is 0.1dBm.

freq is the frequency of the in-band CW signal in Hz. Valid range is 2,395,000,000Hz to 2,485,000,000 Hz inclusive.

onPeriod determines the burst length of the CW signal in units of µs. Valid values are 100ns to 1.6s. To turn off the pulsing of the CW signal, set this parameter to 1 and the offPeriod to 0.

offPeriod determines the interval between bursts of the CW signal in units of µs. Valid values are 0s to 1.6s. To turn off pulsing of the CW signal, set this parameter to 0 and the onPeriod to 1.

res will be set to False if the the command fails.

5.3.12 setAWGN

setAWGN program thes in-band AWGN generator when the Cuprite application is in signal generator mode. This command can only be executed when the Cuprite application is in signal generator mode. There will be no output from the signal generator until startExec is invoked.

res = setAWGN(on, pwr, onPeriod, offPeriod)

on is a Boolean to indicate whether the in-band AWGN generator should be turned on or off. If set, then the in-band AWGN generator is turned on, otherwise the in-band AWGN generator is turned off.

pwr is the power of the AWGN signal in dBm/MHz. Valid range is -162.1dBm/MHz to -39.1dBm/MHz. The power resolution is 0.1dBm.

onPeriod determines the burst length of the AWGN signal in units of µs. Valid values are 100ns to 1.6s. To turn off the pulsing of the AWGN signal, set this parameter to 1 and the offPeriod to 0.

offPeriod determines the interval between bursts of the AWGN signal in units of µs. Valid values are 0s to 1.6s. To turn off pulsing of the AWGN signal, set this parameter to 0 and the onPeriod to 1.

res will be set to False if the the command fails.

5.3.13 setOutOfBandCW

setOutOfBandCW programs the out-of-band CW signal generator when the Cuprite application is in signal generator mode. This command can only be executed when the Cuprite application is in signal generator mode. There will be no output from the signal generator until startExec is invoked.

res = setOutOfBandCW(on , freq , pwr)

on is a Boolean to indicate whether the out-of-band CW generator should be turned on or off. If set, then the out-of-band CW generator is turned on, otherwise the out-of-band CW generator is turned off.

freq is the frequency of the out-of-band CW signal in MHz. Valid range is 24 MHz to 6,000 MHz.

pwr is the power of the out-of-band CW signal in units of dBm. Valid range is -50dBm to -28 dBm. The power resolution is 0.1dBm.

res will be set to False if the the command fails.

5.3.14 setSigAna

setSigAna() programs the operation of the signal analyser. This command can only be executed when the Cuprite application is in signal analyser mode. Operation of the signal analyser will only commence after startExec has been invoked.

res = cuprite.setSigAna(chanMask , pktMask , rssiThresh , testMask , oversample , numAvg, syncThresh, stopOnFail)

chanMask is a 16 bit mask indicating which RF channels the signal analyser should examine. A '1' in the mask indicates that the corresponding RF channel should be analysed, a '0' in the mask indicates that the corresponding RF channel should be ignored.The least significant bit in the mask corresponds to channel 11.

pktMask is a 16 mask determining which packet length groups should be analysed. There are 16 packet length groups. The first length group corresponds to packets with payloads in the range 0 to 7 octets, the second group to packets with payloads of 8 to 15 octets, etc. If a bit is '1', then the results for the corresponding packet length group will be analysed. If a bit is '0', then packets in the corresponding packet length group are ignored.

rssiThresh determines the minimum RSSI reading for packets which will be analysed in units of dBm. Valid values are -120dBm to +20dBm.

testMask is a bit mask indicating which types of data should be collected:

Bit position	Bit mask	Collect
0	0x01	Collect decoded bit stream
1	0x02	Collect basic measurements such as power, EVM, carrier frequency and symbol timing
2	0x04	Collect data from the carrier tracking loop
3	0x08	Collect raw IQ data
4	0x10	Collect average spectrum data
5	0x20	Collect peak spectrum data

oversample is the oversampling ratio to be used to collect IQ waveform data, if selected by the *testMask* field. Possible oversampling ratios are:

Value	Oversampling ratio
0	x2
1	x2.5
2	x5
3	x10

NumAvg it the time period over which spectral data are averaged or over which the spectrum peak hold function is applied. Valid values are:

Value	Time interval
0	10 µs
1	20 µs
2	41 µs
3	82 µs
4	164 µs
5	328 µs
6	655 µs
7	1.3 ms
8	2.6 ms
9	5.2 ms
10	10 ms
11	21 ms
12	42 ms
13	84 ms
14	168 ms
15	336 ms
16	671 ms
17	1.3 s
18	2.7 s
19	5.4 s
20	10 s
21	21 s
22	43 s
23	86 s
24	171 s
25	5.7 mins
26	11 mins
27	23 mins
28	46 mins
29	92 mins
30	3 hrs
31	6 hrs

SyncThresh is the threshold which the synchroniser uses to detect the commencement of a packet. A suitable value is 1488. Raising the threshold will result in loss of packets which are heavily distorted or masked by interference whilst reducing the threshold will result in more packets being detected but also the chance of false detections.

stopOnFail is a Boolean to indicate how the signal analyser should respond when a limit failure is detected. If clear, then the signal analyser ignores the limit failure and continues to run. If set, then the signal analyser will complete the analysis of the current failing packet and then halt.

res will be set to False if the the command fails.

5.3.15 getTable

getTable() returns results in a tabular form when operating in signal analyser mode

The results contributing to the returned table are filtered by:

1. An RF channel filter
2. A packet length group filter

Any packets which satisfy both the filters will contribute to the statistics in the returned table.

table = cuprite.getTable(meas , chanMask , pktMask)

meas is the measurement set for which tabular results are requested. Possible values are:

Meas	Measurement set
0	Output power
1	Carrier frequency, drift & symbol timing
2	EVM
3	Spectrum

chanMask is a 16 bit mask indicating containing the RF channel filter. Only packets which pass the RF channel filter results will contribute to the statistics in the returned table. A '1' in the mask indicates that the corresponding RF channel results should be included in the table, a '0' in the mask indicates that the corresponding RF channel results should be ignored. The least significant bit in the mask corresponds to channel 11.

pktMask is a mask containing the packet length filter. Only packets which pass the packet length filter will contribute to the statistics in the returned table. There are 16 packet length groups. The first length group corresponds to packets with payloads in the range 0 to 7 octets, the second group to packets with payloads of 8 to 15 octets, etc. If a bit is '1', then the results for the corresponding packet length group will contribute to the statistics in the returned table. If a bit is '0', then the results for the corresponding packet length group are ignored.

table is a vector_float type which must be imported from the Cuprite module. See example signal analyser code. It contains the requested table of results. See Poll Results Table for a description of the table contents and format.

5.3.16 plotvsChan

plotvsChan() instructs the Cuprite application to generate a plot of a specified quantity against channel number.

plot = cuprite.plotvsChan(quantity, pktMask)

quantity specifies the measurement results which are to be returned:

Quantity	Measurement	Units
0	P_{avg}	dBm
1	$Pk - P_{avg}$	dB
2	$Initial\ F_c$	kHz
3	$Min\ F_c$	kHz
4	$Max\ F_c$	kHz
5	*Drift*	Hz/µs
6	*Symbol Timing*	ppm
7	EVM_{1000}	%
8	EVM_{rms}	%
9	EVM_{pk}	%
10	$\Delta\varphi_{rms}$	°
11	$\Delta\varphi_{pk}$	°
12	*IQ Imbalance*	dB
13	PSD_{abs}	dBm
14	PSD_{rel}	dB

pktMask is a mask containing the packet length filter. Only packets which pass the packet length filter will contribute to the returned plot data. There are 16 packet length groups. The first length group corresponds to packets with payloads in the range 0 to 7 octets, the second group to packets with payloads of 8 to 15 octets, etc. If a bit is '1', then the results for the corresponding packet length group will contribute to the plot. If a bit is '0', then the results for the corresponding packet length group are ignored.

plot is a vector_float type which must be imported from the Cuprite module. See example signal analyser code. The first 16 values contain the minimum observed measurements of the specified quantity for each of the 16 channels. The next 16 values contain the average of the observed measurements and the final 16 values contain the maximum of the observed measurements.

5.3.17 plotvsPktLen

plotvsPktLen() instructs the Cuprite application to generate a plot of a specified quantity against packet length group. There are 16 packet length groups. The first length group corresponds to packets with payloads in the range 0 to 7 octets, the second group to packets with payloads of 8 to 15 octets, etc.

plot = cuprite.plotvsPktLen(quantity, chanMask)

quantity specifies the measurement results which are to be returned:

Quantity	Measurement	Units
0	P_{avg}	dBm
1	$Pk - P_{avg}$	dB
2	$Initial\ F_c$	kHz
3	$Min\ F_c$	kHz
4	$Max\ F_c$	kHz
5	*Drift*	Hz/µs
6	*Symbol Timing*	ppm
7	EVM_{1000}	%
8	EVM_{rms}	%
9	EVM_{pk}	%
10	$\Delta\varphi_{rms}$	°
11	$\Delta\varphi_{pk}$	°
12	*IQ Imbalance*	dB
13	PSD_{abs}	dBm
14	PSD_{rel}	dB

chanMask is a 16 bit mask indicating containing the RF channel filter. Only packets which pass the RF channel filter results will contribute to the returned plot data. A '1' in the mask indicates that the corresponding RF channel results should be included in the plot, a '0' in the mask indicates that the corresponding RF channel results should be ignored. The least significant bit in the mask corresponds to channel 11.

plot is a vector_float type which must be imported from the Cuprite module. See example signal analyser code. The first 16 values contain the minimum observed measurements of the specified quantity for each of the 16 packet length groups. The next 16 values contain the average of the observed measurements and the final 16 value contain the maximum of the observed measurements.

5.3.18 plotHist

plotHist() instructs the Cuprite application to generate a histogram of the specified quantity

hist = cuprite.plotHist(quantity, chanMask, pktMask)

quantity specifies the measurement for which the histogram is required:

Quantity	Measurement	Units
0	P_{avg}	dBm
1	$Pk - P_{avg}$	dB
2	$Initial\ F_c$	kHz
3	$Min\ F_c$	kHz
4	$Max\ F_c$	kHz
5	*Drift*	Hz/µs
6	*Symbol Timing*	ppm
7	EVM_{1000}	%
8	EVM_{rms}	%
9	EVM_{pk}	%
10	$\Delta\varphi_{rms}$	°
11	$\Delta\varphi_{pk}$	°
12	*IQ Imbalance*	dB
13	PSD_{abs}	dBm
14	PSD_{rel}	dB

chanMask is a 16 bit mask indicating containing the RF channel filter. Only packets which pass the RF channel filter results will contribute to the hisotgram. A '1' in the mask indicates that the corresponding RF channel results should be included in the hisotgram, a '0' in the mask indicates that the corresponding RF channel results should be ignored. The least significant bit in the mask corresponds to channel 11.

pktMask is a mask containing the packet length filter. Only packets which pass the packet length filter will contribute to the histogram. There are 16 packet length groups. The first length group corresponds to packets with payloads in the range 0 to 7 octets, the second group to packets with payloads of 8 to 15 octets, etc. If a bit is '1', then the results for the corresponding packet length group will contribute to the returned hisotgram. If a bit is '0', then the results for the corresponding packet length group are ignored.

hist is a vector_float type which must be imported from the Cuprite module. See example signal analyser code. The values in hist represent the 128 histogram bins. The values corresponding to the lower and upper bins of the histogram are:

Measurement	Lower edge of 1st bin	Upper edge of last bin
P_{avg}	-100 dBm	+28 dBm
$Pk - P_{avg}$	0 db	6.4 dB
Initial F_c	-256 kHz	+256 kHz
Min F_c	-256 kHz	+256 kHz
Max F_c	-256 kHz	+256 kHz
Drift	0 Hz/µs	+256 Hz/µs
Symbol Timing	-128 ppm	+128 ppm
EVM_{1000}	0 %	50 %
EVM_{rms}	0 %	50 %
EVM_{pk}	0 %	50 %
$\Delta\varphi_{rms}$	0 °	12.8 °
$\Delta\varphi_{pk}$	0 °	12.8 °
IQ Imbalance	-5.12 dB	+5.12 dB
PSD_{abs}	-120 dBm	+8 dBm
PSD_{rel}	-70 dB	-6 dB

5.3.19 plotPower

plotPower() instructs the Cuprite application to return a power profile of the most recent packet which satisfies both the specified channel and packet length filters.

plot = cuprite.plotPower(chanMask, pktMask)

chanMask is a 16 bit mask containing the RF channel filter. Only packets which pass the RF channel filter will be used in the search for the most recent packet. A '1' in the mask indicates that the corresponding RF channel should be included in the search, a '0' in the mask indicates that the corresponding RF channel results should be ignored. The least significant bit in the mask corresponds to channel 11.

pktMask is a mask containing the packet length filter. Only packets which pass the packet length filter will be used in the search for the most recent packet. There are 16 packet length groups. The first length group corresponds to packets with payloads in the range 0 to 7 octets, the second group to packets with payloads of 8 to 15 octets, etc. If a bit is '1', then the corresponding packet length group will be included in the search for the most recent packet. If a bit is '0', then the corresponding packet length group is ignored.

plot is a vector_float type which must be imported from the Cuprite module. See example signal analyser code. It contains the power profile of the most recent packet which satisfies both the channel and packet length filters. The power profile starts approximately 40µs prior to the preamble and extends approximately 40µs past the end of the FCS. The power profile is in units of dBm. The oversampling ratio is as defined in the setSigAna() command and can also be obtained through the getPlotOversample() command, provided this is called after the plotPower() command.

5.3.20 plotCarrier

plotCarrier() instructs the Cuprite application to return the output of the carrier tracking loop for the most recent packet which satisfies both the specified channel and packet length filters.

plot = cuprite.plotCarrier(chanMask, pktMask)

chanMask is a 16 bit mask containing the RF channel filter. Only packets which pass the RF channel filter will be used in the search for the most recent packet. A '1' in the mask indicates that the corresponding RF channel should be included in the search, a '0' in the mask indicates that the corresponding RF channel results should be ignored. The least significant bit in the mask corresponds to channel 11.

pktMask is a mask containing the packet length filter. Only packets which pass the packet length filter will be used in the search for the most recent packet. There are 16 packet length groups. The first length group corresponds to packets with payloads in the range 0 to 7 octets, the second group to packets with payloads of 8 to 15 octets, etc. If a bit is '1', then the corresponding packet length group will be included in the search for the most recent packet. If a bit is '0', then the corresponding packet length group is ignored.

plot is a vector_float type which must be imported from the Cuprite module. See example signal analyser code. It contains the output of the carrier tracking loop for the most recent packet which satisfies both the channel and packet length filters. The output of the carrier tracking loop shows the estimated carrier frequency in units of kHz for each pair of chips within the packet.

5.3.21 plotEVM

plotEVM() instructs the Cuprite application to return the EVM profile for the most recent packet which satisfies both the specified channel and packet length filters.

plot = cuprite.plotEVM(chanMask, pktMask)

chanMask is a 16 bit mask containing the RF channel filter. Only packets which pass the RF channel filter will be used in the search for the most recent packet. A '1' in the mask indicates that the corresponding RF channel should be included in the search, a '0' in the mask indicates that the corresponding RF channel results should be ignored. The least significant bit in the mask corresponds to channel 11.

pktMask is a mask containing the packet length filter. Only packets which pass the packet length filter will be used in the search for the most recent packet. There are 16 packet length groups. The first length group corresponds to packets with payloads in the range 0 to 7 octets, the second group to packets with payloads of 8 to 15 octets, etc. If a bit is '1', then the corresponding packet length group will be included in the search for the most recent packet. If a bit is '0', then the corresponding packet length group is ignored.

plot is a vector_float type which must be imported from the Cuprite module. See example signal analyser code. It contains the EVM for each offset constellation point within the packet. The EVM is in units of %.

5.3.22 plotPhi

plotPhi() instructs the Cuprite application to return the phase error for each offset constellation point for the most recent packet which satisfies both the specified channel and packet length filters.

plot = cuprite.plotPhi(chanMask, pktMask)

chanMask is a 16 bit mask containing the RF channel filter. Only packets which pass the RF channel filter will be used in the search for the most recent packet. A '1' in the mask indicates that the corresponding RF channel should be included in the search, a '0' in the mask indicates that the corresponding RF channel should be ignored. The least significant bit in the mask corresponds to channel 11.

pktMask is a mask containing the packet length filter. Only packets which pass the packet length filter will be used in the search for the most recent packet. There are 16 packet length groups. The first length group corresponds to packets with payloads in the range 0 to 7 octets, the second group to packets with payloads of 8 to 15 octets, etc. If a bit is '1', then the corresponding packet length group will be included in the search for the most recent packet. If a bit is '0', then the corresponding packet length group is ignored.

plot is a vector_float type which must be imported from the Cuprite module. See example signal analyser code. It contains the phase error for each offset constellation point within the packet. The phase error is in units of degrees.

5.3.23 plotPSD

plotPSD() instructs the Cuprite application to return the average and peak spectra. Spectra are stored separately for each channel. The channel filter is used to find the most recent packet, the channel of which is determined and then used to select which spectra to return.

plot = cuprite.plotPSD(chanMask,)

chanMask is a 16 bit mask indicating containing the RF channel filter. Only packets which pass the RF channel filter will be used in the search for the most recent packet. A '1' in the mask indicates that the corresponding RF channel should be included in the search, a '0' in the mask indicates that the corresponding RF channel should be ignored. The least significant bit in the mask corresponds to channel 11.

plot is a vector_float type which must be imported from the Cuprite module. See example signal analyser code. It contains 2 x 1844 entires. The first 1844 entries are the average spectrum and the last 1844 samples the peak spectrum. The spectra are in units of dBm/100kHz. If either spectrum is not available, then the corresponding samples are set to -200dBm/100kHz. The first samples corresponds to frequency 2394.971MHz. Subsequent samples are spaced by 100000/2048 kHz. The average time and peak hold time are determined by the numAvg parameter in the call to setSigAna().

5.3.24 plotConstellation

plotConstellation() instructs the Cuprite application to return the offset constellation points for the most recent packet which satisfies both the specified channel and packet length filters.

plot = cuprite.plotConstellation(chanMask, pktMask)

chanMask is a 16 bit mask containing the RF channel filter. Only packets which pass the RF channel filter will be used in the search for the most recent packet. A '1' in the mask indicates that the corresponding RF channel should be included in the search, a '0' in the mask indicates that the corresponding RF channel should be ignored. The least significant bit in the mask corresponds to channel 11.

pktMask is a mask containing the packet length filter. Only packets which pass the packet length filter will be used in the search for the most recent packet. There are 16 packet length groups. The first length group corresponds to packets with payloads in the range 0 to 7 octets, the second group to packets with payloads of 8 to 15 octets, etc. If a bit is '1', then the corresponding packet length group will be included in the search for the most recent packet. If a bit is '0', then the corresponding packet length group is ignored.

plot is a vector_float type which must be imported from the Cuprite module. See example signal analyser code. It contains the offset constellation points within the packet. Successive pairs of samples represent the I and Q values of the offset constellation point. The constellation points are normalised to unit power, ie the offset constellation points lie at { ±1/√2 , ±1/√2 }.

5.3.25 plotIQ

plotIQ() instructs the Cuprite application to returns the IQ samples of the most recent packet which satisfies both the specified channel and packet length filters.

plot = cuprite.plotIQ(chanMask, pktMask)

chanMask is a 16 bit mask containing the RF channel filter. Only packets which pass the RF channel filter will be used in the search for the most recent packet. A '1' in the mask indicates that the corresponding RF channel should be included in the search, a '0' in the mask indicates that the corresponding RF channel results should be ignored. The least significant bit in the mask corresponds to channel 11.

pktMask is a mask containing the packet length filter. Only packets which pass the packet length filter will be used in the search for the most recent packet. There are 16 packet length groups. The first length group corresponds to packets with payloads in the range 0 to 7 octets, the second group to packets with payloads of 8 to 15 octets, etc. If a bit is '1', then the corresponding packet length group will be included in the search for the most recent packet. If a bit is '0', then r the corresponding packet length group is ignored.

plot is a vector_float type which must be imported from the Cuprite module. See example signal analyser code. It contains the IQ samples of the most recent packet which satisfies both the channel and packet length filters. The IQ samples start approximately 40µs prior to the preamble and extends approximately 40µs past the end of the FCS. Each successive pair of samples correspond to the I and Q values at successive time steps. The oversampling ratio is as defined in the setSigAna() command and can also be obtained through the getPlotOversample() command, provided this is called after the plotIQ() command.

5.3.26 plotFM

plotFM() instructs the Cuprite application to returns the FM deviation of the most recent packet which satisfies both the specified channel and packet length filters.

plot = cuprite.plotFM(chanMask, pktMask)

chanMask is a 16 bit mask containing the RF channel filter. Only packets which pass the RF channel filter will be used in the search for the most recent packet. A '1' in the mask indicates that the corresponding RF channel should be included in the search, a '0' in the mask indicates that the corresponding RF channel should be ignored. The least significant bit in the mask corresponds to channel 11.

pktMask is a mask containing the packet length filter. Only packets which pass the packet length filter will be used in the search for the most recent packet. There are 16 packet length groups. The first length group corresponds to packets with payloads in the range 0 to 7 octets, the second group to packets with payloads of 8 to 15 octets, etc. If a bit is '1', then the corresponding packet length group will be included in the search for the most recent packet. If a bit is '0', then the corresponding packet length group is ignored.

plot is a vector_float type which must be imported from the Cuprite module. See example signal analyser code. It contains the FM deviation of the most recent packet which satisfies both the channel and packet length filters. The FM deviation samples start approximately 40µs prior to the preamble and extends approximately 40µs past the end of the FCS. The FM deviation is in units of kHz. The oversampling ratio is as defined in the setSigAna() command and can also be obtained through the getPlotOversample() command, provided this is called after the plotFM() command.

5.3.27 getBits

getBits() instructs the Cuprite application to return the demodulated bitstream associated with the packet analysed in the most recent plot command.

bits = getBits()

bits is a vector_uint8 type which must be imported from the Cuprite module. See example signal analyser code. It contains the demodulated octets associated with the packet which was analysed in the most recent plot command. The first 5 octets are always 0x00, 0x00, 0x00, 0x00, 0xA7 representing the preamble and SFD fields of the packet.

5.3.28 clearResults

clearResults() erases selected test results accumulated in the Cuprite application when in the signal analyser mode.

res = cuprite.clearResults(chanMask, pktMask)

chanMask is a 16 bit mask indicating the channels for which the results should be cleared. A '1' in the mask indicates that the results for the corresponding RF channel will be cleared, a '0' in the mask indicates that the results for the corresponding RF channel will be left unaltered. The least significant bit in the mask corresponds to channel 11.

pktMask is a mask indicating the packet length groups for which the results should be cleared. There are 16 packet length groups. The first length group corresponds to packets with payloads in the range 0 to 7 octets, the second group to packets with payloads of 8 to 15 octets, etc. If a bit is '1', then the results for the corresponding packet length group will be ckeared. If a bit is '0', then the results for the corresponding packet length group are left unaltered.

res will be set to False if the the command fails.

5.3.29 setLimits

setLimits() informs the Cuprite application of the test limits to be applied. The test limits are used in the signal analyser mode.

res = cuprite.setLimits(x)

x is a vector_int16 containing the test limits. vector_int16 must be imported from the Cuprite module (see example signal analyser code). This vector has 7 entries:

Entry	Measurement	Limit Type	Min value	Max value	Units
0	P_{avg}	Lower	-120dBm	+30dBm	0.1dBm
1	P_{avg}	Upper	-120dBm	+30dBm	0.1dBm
2	F_c	Upper	0kHz	200kHz	1kHz
3	*Symbol timing*	Upper	0ppm	100ppm	1ppm
4	EVM_{1000}	Upper	0%	100%	0.1%
5	PSD_{abs}	Upper	-100dBm	+20dBm	0.1dBm
6	PSD_{rel}	Upper	-100dB	0dB	0.1dB

res will be set to False if the the command fails.

5.3.30 getLimits

getLimits() returns from the Cuprite application the test limits which are being applied. The test limits are used in the signal analyser mode.

x = cuprite.getLimits( )

x is a vector_int16 containing the test limits. vector_int16 must be imported from the Cuprite module (see example signal analyser code). The vector has 7 entries:

Entry	Measurement	Limit Type	Min value	Max value	Units
0	P_{avg}	Lower	-120dBm	+30dBm	0.1dBm
1	P_{avg}	Upper	-120dBm	+30dBm	0.1dBm
2	F_c	Upper	0kHz	200kHz	1kHz
3	*Symbol timing*	Upper	0ppm	100ppm	1ppm
4	EVM_{1000}	Upper	0%	100%	0.1%
5	PSD_{abs}	Upper	-100dBm	+20dBm	0.1dBm
6	PSD_{rel}	Upper	-100dB	0dB	0.1dB

5.3.31 setSniffer

setSniffer() programs the operation of the packet sniffer. This command can only be executed when the Cuprite application is in packet sniffer mode. Operation of the spacket sniffer will only commence after startExec has been invoked.

res = cuprite.setSniffer(chanMask , rssiThresh, syncThresh)

chanMask is a 16 bit mask indicating which RF channels the packet sniffer should report. A '1' in the mask indicates that the corresponding RF channel should be received, a '0' in the mask indicates that the corresponding RF channel should be ignored.The least significant bit in the mask corresponds to channel 11.

rssiThresh determines the minimum RSSI reading for packets which will be received in units of dBm. Valid values are -120dBm to +20dBm.

SyncThresh is the threshold which the synchroniser uses to detect the commencement of a packet. A suitable value is 1488. Raising the threshold will result in loss of packets which are heavily distorted or masked by interference whilst reducing the threshold will result in more packets being detected but also the chance of false detections.

res will be set to False if the the command fails.

5.3.32 setRxAtten

setRxAtten() sets the receiver frontend attenuation. This command can be executed at any time.

res = cuprite.setRxAtten(atten)

atten contains the receiver frontend attenuation in units of 0.5 dB. The permissible attenuator range is 0 to 31.5 dB.

res will be set to False if the the command fails.

5.3.33 getRxAtten

getRxAtten returns the current setting of the receiver front end attenuator.

atten = cuprite.getRxAtten()

atten contains the receiver frontend attenuation in units of 0.5 dB. The permissible attenuator range is 0 to 31.5 dB.

5.3.34 setRxPort

setRxPort() determines whether the Monitor In port or the Tx/Rx port should be used for reception. The Monitor In port is has a noise figure of +6 dB and so is ideally suited for performing off-air measurements. The Tx/Rx port has a noise figure of +46 dB but can handle signals as large as +27 dBm. It is therefore ideally suited for conducted measurements. This command can be executed at any time.

res = cuprite.setRxPort(port)

port is the receiver port to use for reception and can be one of the following values:

0	Monitor In
1	Tx/Rx port

res will be set to False if the the command fails.

5.3.35 getRxPort

getRxPort() returns the port currently in use for reception. The Monitor In port is has a noise figure of +6 dB and so is ideally suited for performing off-air measurements. The Tx/Rx port has a noise figure of +46 dB but can handle signals as large as +27 dBm. It is therefore ideally suited for conducted measurements. This command can be executed at any time.

port = cuprite.getPort()

port is the receiver port used for reception and can be one of the following values:

0	Monitor In
1	Tx/Rx port

5.3.36 setExtDioVolts

setExtDioVolts() determines whether the digital IO voltage is supplied by *TLF3000* or by an external source. If *TLF3000* supplies the digital IO voltage then it is fixed at 3.3 V. The *TLF3000* is able to provide up to 500 mA on the 3v3 supply. If an external voltage is used for the digital IO then it must be in the range 0.8 V to 3.6 V. This command can be executed at any time.

res = cuprite.setExtDioVolts(ext)

ext determines whether the IO voltage is supplied internally or externally:

0	3.3 V IO supplied by TLF3000
1	0.8 V to 3.6 V IO supplied externally

res will be set to False if the the command fails.

5.3.37 getExtDioVolts

getExtDioVolts() returns whether the digital IO voltage is supplied by *TLF3000* or by an external source. If *TLF3000* supplies the digital IO voltage then it is fixed at 3.3 V. The *TLF3000* is able to provide up to 500 mA on the 3v3 supply. If an external voltage is used for the digital IO then it must be in the range 0.8 V to 3.6 V. This command can be executed at any time.

ext = cuprite.getDIOVolts( )

ext indicates whether the IO voltage is supplied internally or externally:

0	3.3 V IO supplied by TLF3000
1	0.8 V to 3.6 V IO supplied externally

5.3.38 getError

getError() returns the oldest error message from the Cuprite error queue. Once read, this eror message is removed from the error queue.

msg = cuprite.getError()

msg is a string containing the oldest message of the Cuprite error queue. If the queue is empty, then an empty string is returned.

5.3.39 exitApp

exitApp() will cause the Cuprite application to exit and control return to the *TLF3000* supervistor program,

res = cuprite.exitApp()

res will be set to False if the the command fails.

5.3.40 hardwareReset

hardwareReset() will cause the *TLF3000* to reboot.

cuprite.hardwareReset()

5.3.41 powerDown

powerDown() will power down the *TLF3000*.

cuprite.powerDown()

5.3.42 getFriendlyName

getFriendlyName() will return the friendly name of the *TLF3000* unit.

name = cuprite.getFriendlyName()

name is a string containg the friendly name of the *TLF3000* unit.

5.3.43 getSerialNumber

geSerialNumber() will return the serial number of the *TLF3000* unit.

sn = cuprite.getSerialNumber()

sn is an integer containing the serial number of the *TLF3000* unit.

6 Connectors.

6.1 Front Panel

6.1.1 Monitor In Port

Function	High sensitivity 2.4 GHz RF input Suitable for radiated measurements
Connector type	SMA
Noise figure	6 dB typ
IP3 @ max sensitivity	+7 dBm typ
SNR in 1MHz bandwidth	80 dB typ
Maximum input signal	27 dBm
Maximum usable signal	-10 dBm typ
Frequency range	2401-2481 MHz
Impedance	50 Ω
Coupling	AC
Maximum DC voltage	50 V

6.1.2 Tx/Rx port

Function	Low sensitivity 2.4GHz RF input and signal generator output Suitable for conducted measurements
Connector type	SMA
Impedance	50 Ω
Coupling	AC
Maximum DC voltage	50 V

6.1.3 Receiver Specification

Noise figure	46 dB typ
IP3 @ max sensitivity	+47 dBm typ
SNR in 1MHz bandwidth	80 dB typ
Maximum input signal	27 dBm
Maximum usable signal	27 dBm typ
Frequency range	2401-2481 MHz

6.1.4 Video port

Function	Not used in Cuprite application
Connector type	SMA
Impedance	50 Ω
Coupling	AC
Maximum DC voltage	5 V

6.1.5 External Clock Input

Function	External clock input Permits internal RF, sampling and timestamp clocks to be locked to an external reference
Connector type	SMA
Maximum input signal	-10 dBm
Minimum input signal	+20 dBm
Frequency range	10 MHz
Impedance	50 Ω
Coupling	AC
Maximum DC voltage	50 V

6.1.6 Reference Clock Output

Function	Reference clock output Provides a reference clock which can be used to synchronise other test equipment
Connector type	SMA
Output signal	-2 dBm
Frequency	10 MHz
Impedance	50 Ω
Coupling	AC
Maximum DC voltage	5 V

6.2 Rear Panel

6.2.1 Digital IO

Function	Digital input and output
Connector type	Hirose LX60-20S

Logic levels when IO voltage is internal 3.3V	
Logic input high	2.0 V (min)
Logic input low	0.8 V (max)
Logic output high	2.4 V (min)
Logic output low	0.55 V (max)
Output current	±24 mA

Pin	Name	Direction	Special function
1	Vio	I/O	Max 500 mA from internal 3.3 V supply. External voltage range 0.8 V to 3.6 V.
2	Vio	I/O	
3	GPI #0	I	
4	GPI #1	I	
5	GPI #2	I	
6	GPI #3	I	
7	GPI #4	I	
8	GPI #5	I	
9	GPI #6	I	
10	GPI #7	I	
11	GPO #0	O	
12	GPO#1	O	
13	GPO#2	O	
14	GPO#3	O	
15	GPO#4	O	
16	GPO#5	O	
17	GPO#6	O	
18	GPO#7	O	
19	Gnd	-	
20	Gnd	-	

6.2.2 USB Connector

Function	USB connection to host
Connector type	Micro-USB
Speed rating	High speed
VBUS load	2.2 µF, > 10 kΩ

6.2.3 Ethernet Connector

Function	Ethernet connection to host
Connector type	RJ45
Speed	10 / 100 / 1000

6.2.4 Power Connector

Function	DC power input
Connector type	2.5 mm jack
Input voltage	12 V DC
Power	10 W typ (application dependent)
Reverse polarity protection	Yes
Over voltage protection	Yes
Under voltage protection	Yes

6.3 Connections for Cuprite Operating Modes

Function	Signal Generator	Signal Analyser	Packet Sniffer
RF Tx to DUT	Tx/Rx	-	-
RF Rx from DUT	-	Tx/Rx or Monitor In	Tx/Rx or Monitor In
IO voltage	Ext Vio or 3v3 Int	Ext Vio or 3v3 Int	Ext Vio or 3v3 Int

7 Uncertainty

Quantity	Uncertainty (typical)
Absolute RF power (in-band)	±1.2 dB
Relative RF power (in-band)	±1 dB
Relative RF power (out-of-band)	±3 dB
Absolute frequency (internal reference)	±5 kHz ±2.5 kHz/year
Absolute frequency (external reference)	±1 kHz
Relative frequency	±500 Hz

8 Indicators

8.1 Front Panel

8.1.1 Status

The right hand LED indicates the status of the unit. When power is applied this will light YELLOW indicating that the internal power supplies are good. Once the unit has booted and is ready for host commands it will turn GREEN.

8.1.2 RF Overload

The left hand LED indicates whether an RF overload is present. The RF overload may be either on a receiver input or the signal generator output. The *Input/Output Power* messages can provide further details on the source of the overload.

The both the Monitor In port and the Tx/Rx port are protected for input signals up to a maximum level of 27 dB. Input signals beyond this will destroy the unit. If a receive overload condition is present, the results of the unit are uncertain. If the Monitor In port is in use, then swapping to the less sensitive Tx/Rx port may alleviate the RF overload condition.

8.1.3 Flash Update

When the Flash is being updated, the Status LED flash will flash between YELLOW and RED. Do not remove power from the unit whilst the Flash is being updated.