Phase Lock
TCP/IP Protocol

Update History

Version	Date	Engineer	Details
1	11/04/17	Euan Cochrane	First release of TCP commands for Phase Lock
2	19/05/17	Euan Cochrane	Added “aux beat” parameter to LO profile configuration.

1. Introduction

This document contains the definition of the TCP/IP interface to be used with Phase Lock ICE-BLOCS. The interface allows 3rd. party machines to communicate with Phase Lock.

In a TCP/IP network communication sequence one processor needs to be the Server and one processor needs to be the Client. Once the link has been established the two processors have equal status on the link and so both must monitor for received transmissions as well as sending requests. In this instance Phase Lock is the Server.

1.1. TCP Details

The Network Configure page on Phase Lock contains a Remote Interface definition. This feature allows a user to define and enable a TCP/IP link to a 3rd. party instrument. The user defines the TCP/IP parameters on this page and can also establish the link for the first time. Once established the user can instruct the ICE-BLOC to re-create the link whenever it switches on. The page can also be used to stop a link which is no longer required.

The fields on the Network Configure page are as follows:-

IP Address – The IP address of the 3rd. party instrument.
Port – A port identifier between 1024 and 65535.
Enable (button) – Instruct the ICE-BLOC to make this connection.
Disable (button) – Instruct the ICE-Bloc to break this connection.
Save – The link status, connected or otherwise, is saved and re-established at power up.

1.1.1. IP Address

The user must supply the IP address of the 3rd. party instrument to the Phase Lock.

1.1.2. Port

The user needs to supply a port number for the TCP/IP interface. Both boxes on the communications link need to be given the same port number. If the user is unsure of which port to use he should either consult his system administrator or use the internet to determine which ports are commonly allocated to other network operations and select an unused one. Ports 0 to 1023 are sometimes referred to as "system" or "well known" ports and Unix type systems may not allow binding to these ports. An Internet search of "common TCP/UDP port numbers" will produce both tables and tools which may be used to establish if a port is commonly used. The user should select a port number between 1024 and 65535 which is not commonly used.

1.1.3. Enable

The Enable operation instructs the box to try to establish a connection across the defined interface. Visual indication of the state of the link is given and once establish both boxes will monitor the interface for requests.

1.1.4. Disable

The Disable operation instructs the box to remove the TCP/IP data link. The link is closed and all port monitoring stops.

1.1.5. Save

The Save operation saves all the fields defined above so that they may be restored during power up. If the link is connected when Save is used the ICE-BLOC will re-establish the link when power is applied.

2. Message Transmissions

Once the TCP link has been established the boxes use text messages to issue tasks to each other and to send back results. When a box needs the other box to do something it sends a message containing a request, or task. If all is well with the request the receiving box performs the task and then replies with the task reply. Most of the commands in this protocol are defined as a task and task reply message pair. Commands which have no reply are highlighted below as they occur.

Some operations may run for a prolonged period and it is not normally appropriate for the commanding box to wait for these to complete. When this is the case the operation is specified as having control tasks and status tasks so that it may be switched on and then monitored periodically. It is the responsibility of the commanding box to request status on a regular basis when an operation of this type is being executed.

The message parser can reject a message if it doesn't adhere to the standard. When this happens the error report must be able to identify the message and the point at which the parser failed. For this reason the message contains a message identification field which can be easily extracted if the parser fails.

2.1. ICE-BLOC Message Structure

The structure of each message uses the JSON format to specify a task. JSON is a acronym of JavaScript Object Notation and is defined at the web address www.json.org. The main item in the JSON notation is called an object and in its simplest form is as follows:-

```
{ string : value }                              // single item, JSON format
{ string : value , string : value, ........}     // 2 or more items JSON format
```

Key

{ : , } - Mandatory JSON characters

string – Text in inverted commas, no white space, no minus characters, usually a tag or descriptive field.

value – The value of the item.

Text string in inverted commas, “text_without_white_space”.

Numeric in square brackets [25].

An array of numerics also in square brackets [25, 29, 47, 73].

A further embedded JSON object.

This method is similar in concept to the XML data format where every data item is tagged. JSON however does not need a closing tag as is used in XML.

2.1.1. Task Structure

The JSON specification states that an item's value may be a string, a number, an array of data values or another item. Embedded items allow a data stream to adopt a structure similar to the data structures used by the software to process it. The following structure shows a simple measure voltage task using embedded items within a JSON format:-

```
{
        “op”: “measure_voltage”,
        “parameters”:
        {
                “card”: [1],                    // sample value
                “bank”: [4],                    // sample value
                “pin”: [15]                     // sample value
        }
}
```

2.1.2. Message Structure

Tasks are transmitted between the boxes in messages. Each message contains a single task, as above, and an identification which is echoed as part of the message reply. Data transmissions, or messages, all adhere to a common structure as shown below.

```
{
        “message”:
        {
                “transmission_id”: [tx identification],
                “op” : operation,
                “parameters” :
                {
                        parameter string: parameter,
                        parameter string: parameter
                }
        }
}
```

Key

{ } : , [ ]	-	mandatory json characters, ascii 7b, 7d, 3a, 2c, 91, 93 hex
“message”	-	mandatory protocol string, case sensitive in quotes
“transmission_id”	-	mandatory protocol string, case sensitive in quotes
[tx identification]	-	numeric identifier in square brackets
“op”	-	mandatory protocol string, case sensitive in quotes
operation	-	task specific text string identifying the task, in quotes
“parameters”	-	optional protocol string, case sensitive, in quotes
parameter string	-	optional task specific parameter string, in quotes

parameter	-	optional task specific variable field, numeric or string in quotes

In the above definition only the tx identification, operator, parameter strings and parameters vary for each message. The tx identification is system generated at the time of execution and is not part of the details of an operational command. All formal command definitions for the remainder of this document shall be expressed in terms of operation, parameter string(s) and parameter(s).

2.1.3. Additional ICE-BLOC Rules

In addition to the above JSON structure some additional protocol rules are as follows:-

string fields – String items within the protocol are specified in double quotes and must not include white space characters nor minus characters(-). When naming text fields and text variables, underline(_) characters should be used to improve readability.

numeric fields – Numeric fields are enclosed in square brackets so that they appear as single element arrays, e.g. [27].

2.2. Establishing TCP/IP Link

Within the TCP/IP operation the formal terms for establishing the link are listen, connect and accept. The Server listens, the Client connects, the Server accepts. Once this process has been successfully completed both processors may send and receive messages across the communications link as required.

In this implementation Phase Lock is the Server and the 3rd. party instrument is the Client. Once the link has been established Phase Lock will wait for a message from the Client to validate the connection, see Start Link Command below. Phase Lock will then either allow the link to continue or shut it down. When the link is successfully made Phase Lock will reply with the Start Link Reply, see below.

Once the link is established Phase Lock monitors the link for requests and only transmits data in response to 3rd. party requests. The developer for the 3rd. party software can rely on this so that the port need only be monitored when a transmission has been made. The reply to a transmission finishes the sequence and the port need not be read again until a further request is made.

Phase Lock uses a Start Link command as the first message transfer once a connection has been established. The 3rd. party instrument, Client, should transmit a Start Link Command once the connection has been accepted, see below Start Link Command. The Phase Lock, Server, shall transmit a Start Link Reply when it receives a Start Link Command. The Start Link Command and the Start Link Reply are both described in subsequent paragraphs.

2.2.1. Start Link Command

In the command shown below the 3rd. party instrument, Client ,at IP address 192.168.1.222 is attempting to link to Phase Lock, Server.

```
{
        “message”:
        {
                “transmission_id”: [999] ,                    // sample value
                “op”:“start_link”,
                “parameters” :
                {
                        “ip_address”: “192.168.1.222”
                }
        }
}
```

2.2.2. Start Link Reply

In the message shown below the Phase Lock, Server, at IP address 192.168.1.111 is replying to a link command.

```
{
        “message”:
        {
                “transmission_id”: [1003],            // sample value
                “op”: “start_link_reply”,
                “parameters “:
                {
                        “ip_address”: “192.168.1.111”,
                        “status”: “ok”
                }
        }
}
```

2.2.3. Start Link Command Definition

This definition of the Start Link command is as follows:-

Command	
operator	“start_link”
parameter 1 string	“ip_address”
parameter 1 value	ip address in a string, e.g. “192.168.1.222”, Client address

Reply	
operator	“start_link_reply”
parameter 1 string	“ip_address”
parameter 1 value	ip address in a string, e.g. “192.168.1.111”, Server address
parameter 2 string	“status”
parameter 2 value	“ok” or “failed”

2.3. Parse Fail

This command is sent by a receiving box to report that a transmission contains a task which cannot be processed. The receiving software can identify two causes of failure within a transmission. Firstly the received data may not be acceptable according to the JSON definition. When this occurs the reply shall contain the received string showing the point at which the parser objects to the format. Secondly the received data may constitute a valid JSON structure but contains either erroneous or non existent commands. When this occurs the reply shall contain an error code indicating why the command is unacceptable.

This command is a reply to some other failing task so it does not have a command definition.

2.3.1. Parse Fail Reply Only

Command	
There is no command defined for this command	
Reply	
operator	“parse_fail”
parameter 1 string	“transmission”
parameter 1 value	transmission id from the top of the failing message if known
parameter 2 string	“protocol_error”
parameter 2 value	Error Code, see below.
parameter 3 string	“JSON_parse_error”
parameter 3 value	received buffer showing reported JSON non compliance, see Parse Error below.

Parse Error

The text shown in parameter 2 above shows the point at which the parser could not proceed followed by the remainder of the buffer. The user may need to use the original transmission to look both before and after this area of text to establish the nature of the fault.

Error Code

1	-	JSON parsing error, invalid start command, wrong IP address.
2	-	“message” string missing.
3	-	“transmission_id” string missing.
4	-	No transmission id value.
5	-	“op” string missing.
6	-	No operation name.
7	-	Operation not recognised.
8	-	“parameters” string missing.
9	-	Invalid parameter tag or value.

2.4. Ping Command

This command causes the receiving box to invert the case of the received text and send it back.

Command	
operator	“ping”
parameter 1 string	“text_in”
parameter 1 value	sample text string, e.g. “CheckThis”
Reply	
operator	“ping_reply”
parameter 1 string	“text_out”
parameter 1 value	modified text string, e.g. “cHECKtHIS”

3. Phase Lock Commands

A series of commands has been developed to operate the Phase Lock remotely. These commands may be issued by another ICE Bloc or a remote interface client.

3.1. Tune Resonator

This command adjusts the resonator tuning.

Command	
operator	“tune_resonator”
parameter 1 string	“setting”
parameter 1	Resonator tuning, see below
parameter 2 string	*“report” // optional end task report*
parameter 2 value	*“finished”*
Reply	
operator	“tune_resonator_reply”
parameter 1 string	“status”
parameter 1 value	Status value, see below
Final Report	*// optional report, final transmission format*
operator	*“tune_resonator_f_r”*
parameter 1 string	*“report”*
parameter 1 value	*Report value, see below*

Resonator tuning

0 – 100	The Resonator tuning range is expressed as a percentage where 100 is full scale.

Status value

0	-	operation completed.
1	-	setting out of range.
2	-	command failed.

Report value

0	-	Task completed.
1	-	Task failed.

3.2. Main Lock

This command puts the main lock on or off.

Command	
operator	“main_lock”
parameter 1 string	“operation”
parameter 1	Request condition, see below
parameter 2 string	*“report” // optional end task report*
parameter 2 value	*“finished”*
Reply	
operator	“main_lock_reply”
parameter 1 string	“status”
parameter 1 value	Status value, see below
Final Report	*// optional report, final transmission format*
operator	*“main_lock_f_r”*
parameter 1 string	*“report”*
parameter 1 value	*Report value, see below*

Request condition

“off”	-	Request to remove the lock.
“on”	-	Request to apply the lock.

Status value

0	-	operation completed.
1	-	operation failed.

Report value

0	-	Task completed.
1	-	Task failed.

3.3. Main Lock Status

This command gets the current status of the main lock

Command	
operator	“main_lock_status”

There are no parameters for this operation. Parameter fields omitted from command.	
Reply	
operator	“main_lock_status_reply”
parameter 1 string	“status”
parameter 1 value	Status value, see below
parameter 2 string	“condition”
parameter 2 value	Current lock condition, see below.

Status value

0	-	operation completed.
1	-	command failed.

Current lock condition

“off”	-	the lock is off
“on”	-	the lock is on
“debug”	-	the lock is in a debug condition
“error”	-	the lock operation is in error
“search”	-	the lock search algorithm is active
“low”	-	the lock is off due to low output.

3.4. Aux Lock

This command puts the aux lock on or off.

Command	
operator	“aux_lock”
parameter 1 string	“operation”
parameter 1	Request condition, see below
parameter 2 string	*“report”* *// optional end task report*
parameter 2 value	*“finished”*
Reply	
operator	“aux_lock_reply”
parameter 1 string	“status”
parameter 1 value	Status value, see below
Final Report	*// optional report, final transmission format*

operator	*"aux_lock_f_r"*
parameter 1 string	*"report"*
parameter 1 value	*Report value, see below*

Request condition

"off"	-	Request to remove the lock.
"on"	-	Request to apply the lock.

Status value

0	-	operation completed.
1	-	operation failed.

Report value

0	-	Task completed.
1	-	Task failed.

3.5. Aux Lock Status

This command gets the current status of the aux lock

Command	
operator	"aux_lock_status"
There are no parameters for this operation. Parameter fields omitted from command.	
Reply	
operator	"aux_lock_status_reply"
parameter 1 string	"status"
parameter 1 value	Status value, see below
parameter 2 string	"condition"
parameter 2 value	Current lock condition, see below.

Status value

0	-	operation completed.
1	-	command failed.

Current lock condition

"off"	-	the lock is off
"on"	-	the lock is on

"debug"	-	the lock is in a debug condition
"error"	-	the lock operation is in error
"search"	-	the lock search algorithm is active
"low"	-	the lock is off due to low output.

3.6. ECD Lock

This command puts the ECD lock on or off.

Command	
operator	"ecd_lock"
parameter 1 string	"operation"
parameter 1	Request condition, see below
parameter 2 string	*"report" // optional end task report*
parameter 2 value	*"finished"*
Reply	
operator	"ecd_lock_reply"
parameter 1 string	"status"
parameter 1 value	Status value, see below
Final Report	*// optional report, final transmission format*
operator	*"ecd_lock_f_r"*
parameter 1 string	*"report"*
parameter 1 value	*Report value, see below*

Request condition

"off"	-	Request to remove the lock.
"on"	-	Request to apply the lock.

Status value

0	-	operation completed.
1	-	operation failed.

Report value

0	-	Task completed.
1	-	Task failed.

3.7. ECD Lock Status

This command gets the current status of the ecd lock

Command	
operator	“ecd_lock_status”
There are no parameters for this operation. Parameter fields omitted from command.	
Reply	
operator	“ecd_lock_status_reply”
parameter 1 string	“status”
parameter 1 value	Status value, see below
parameter 2 string	“condition”
parameter 2 value	Current lock condition, see below.

Status value

0	-	operation completed.
1	-	command failed.

Current lock condition

“off”	-	the lock is off
“on”	-	the lock is on
“debug”	-	the lock is in a debug condition
“error”	-	the lock operation is in error
“search”	-	the lock search algorithm is active
“low”	-	the lock is off due to low output.

3.8. Select LO Profile

This command changes the currently selected LO profile.

Command	
operator	“select_lo_profile”
parameter 1 string	“profile”
parameter 1	0 - 7
parameter 2 string	*“report” // optional end task report*
parameter 2 value	*“finished”*

Reply	
operator	"select_lo_profile_reply"
parameter 1 string	"status"
parameter 1 value	Status value, see below
Final Report	*// optional report, final transmission format*
operator	*"select_lo_profile_f_r"*
parameter 1 string	*"report"*
parameter 1 value	*Report value, see below*

Status value

0	-	operation completed.
1	-	operation failed.

Report value

0	-	Task completed.
1	-	Task failed.

3.9. Configure LO Profile

This command reconfigures the current LO profile.

Command	
operator	"configure_lo_profile"
parameter 1 string	"main_synth"
parameter 1	"enable" or "disable"
parameter 2 string	"aux_synth"
parameter 2	"enable" or "disable"
parameter 3 string	"aux_detector_mode"
parameter 3	"ecd" or "aux"
parameter 4 string	"input_frequency"
parameter 4	Input frequency value in Hz
parameter 5 string	"beat_frequency_trim"
parameter 5	Beat frequency trim value in Hz (ECD mode only)
parameter 6 string	"chirp_rate"
parameter 6	Chirp rate value in Hz/s (ECD mode only)

parameter 7 string	“chirp duration”
parameter 7	Chirp duration in s (ECD mode only)
parameter 8 string	“aux_beat”
parameter 8	“fundamental” or “2nd_harmonic” (Aux mode only)
parameter 9 string	*“report” // optional end task report*
parameter 9	*“finished”*
Reply	
operator	“configure_lo_profile_reply”
parameter 1 string	“status”
parameter 1 value	Status value, see below
Final Report	*// optional report, final transmission format*
operator	*“configure_lo_profile_f_r”*
parameter 1 string	*“report”*
parameter 1 value	*Report value, see below*

Status value

0	-	operation completed.
1	-	operation failed.

Report value

0	-	Task completed.
1	-	Task failed.

3.10. Configure AOM

This command reconfigures the AOM.

Command	
operator	“configure_aom”
parameter 1 string	“aom_synth”
parameter 1	“enable” or “disable”
parameter 2 string	“drive_frequency”
parameter 2	Drive frequency value in Hz

parameter 3 string	*"report"* *// optional end task report*
parameter 3	*"finished"*
Reply	
operator	"configure_aom_reply"
parameter 1 string	"status"
parameter 1 value	Status value, see below
Final Report	*// optional report, final transmission format*
operator	*"configure_aom_f_r"*
parameter 1 string	*"report"*
parameter 1 value	*Report value, see below*

Status value

0	-	operation completed.
1	-	operation failed.

Report value

0	-	Task completed.
1	-	Task failed.

3.11. Apply monitor A

This command switches the requested signal to monitor A output port

Command	
operator	"monitor_a"
parameter 1 string	"signal"
parameter 1	Requested signal, see below
parameter 2 string	*"report"* *// optional end task report*
parameter 2 value	*"finished"*
Reply	
operator	"monitor_a_reply"
parameter 1 string	"status"
parameter 1 value	Status value, see below

Final Report	*// optional report, final transmission format*
operator	*“monitor_a_f_r”*
parameter 1 string	*“report”*
parameter 1 value	*Report value, see below*

Requested signal

1	-	Aux lock output.
2	-	Main phase error.
3	-	IF phase error.
4	-	Aux phase error.
5	-	EOM output.
6	-	M3 fast output.
7	-	Main input power.
8	-	Aux input power.

Status value

0	-	operation completed.
1	-	operation failed.

Report value

0	-	Task completed.
1	-	Task failed.

3.12. Apply monitor B

This command switches the requested signal to monitor B output port

Command	
operator	“monitor_b”
parameter 1 string	“signal”
parameter 1	Requested signal, see below
parameter 2 string	*“report”* *// optional end task report*
parameter 2 value	*“finished”*
Reply	
operator	“monitor_b_reply”
parameter 1 string	“status”
parameter 1 value	Status value, see below

__Final Report__	*// optional report, final transmission format*
operator	*“monitor_b_f_r”*
parameter 1 string	*“report”*
parameter 1 value	*Report value, see below*

Requested signal

1	-	Aux lock output.
2	-	Main phase error.
3	-	IF phase error.
4	-	Aux phase error.
5	-	EOM output.
6	-	M3 fast output.
7	-	Main input power.
8	-	Aux input power.

Status value

0	-	operation completed.
1	-	operation failed.

Report value

0	-	Task completed.
1	-	Task failed.

3.13. Frequency reference selection

This command selects the source of the frequency reference.

Command	
operator	“select_freq_reference”
parameter 1 string	“setting”
parameter 1	“internal” or “external”
parameter 2 string	*“report”* *// optional end task report*
parameter 2 value	*“finished”*
Reply	
operator	“select_freq_reference_reply”
parameter 1 string	“status”
parameter 1 value	Status value, see below

Final Report	*// optional report, final transmission format*
operator	*“select_freq_reference_f_r”*
parameter 1 string	*“report”*
parameter 1 value	*Report value, see below*

Status value

0	-	operation completed.
1	-	command failed.

Report value

0	-	Task completed.
1	-	Task failed.

3.14. Frequency reference trim

This command sets a trim level on the frequency reference.

Command	
operator	“trim_freq_reference”
parameter 1 string	“setting”
parameter 1	Trim voltage value, 0 – 10V floating point
parameter 2 string	*“report” // optional end task report*
parameter 2 value	*“finished”*
Reply	
operator	“trim_freq_reference_reply”
parameter 1 string	“status”
parameter 1 value	Status value, see below
Final Report	*// optional report, final transmission format*
operator	*“trim_freq_reference_f_r”*
parameter 1 string	*“report”*
parameter 1 value	*Report value, see below*

Status value

0	-	operation completed.
1	-	command failed.

Report value

0	-	Task completed.
1	-	Task failed.

3.15. Main LO selection

This command selects the source of the main LO.

Command	
operator	“select_main_lo”
parameter 1 string	“setting”
parameter 1	“internal” or “external”
parameter 2 string	*“report” // optional end task report*
parameter 2 value	*“finished”*
Reply	
operator	“select_main_lo_reply”
parameter 1 string	“status”
parameter 1 value	Status value, see below
Final Report	*// optional report, final transmission format*
operator	*“select_main_lo_f_r”*
parameter 1 string	*“report”*
parameter 1 value	*Report value, see below*

Status value

0	-	operation completed.
1	-	command failed.

Report value

0	-	Task completed.
1	-	Task failed.

3.16. System Status

This command obtains the current system status

Command	
operator	“get_status”
There are no parameters for this operation. Parameter fields omitted from command.	
Reply	
operator	“get_status_reply”
parameter 1 string	“status”
parameter 1 value	Status value, see below
parameter 2 string	“beat_freq”
parameter 2 value	Beat frequency in Hz
parameter 3 string	“main_synth_freq”
parameter 3 value	Main synthesizer frequency in Hz
parameter 4 string	“aux_synth_freq”
parameter 4 value	Aux synthesizer frequency in Hz
parameter 5 string	“aom_synth_freq”
parameter 5 value	AOM synthesizer frequency in Hz
parameter 6 string	“dds_freq”
parameter 6 value	DDS frequency in Hz
parameter 7 string	“main_synth_status”
parameter 7 value	Main synthesizer status, see below
parameter 8 string	“aux_synth_status”
parameter 8 value	Aux synthesizer status, see below
parameter 9 string	“aom_synth_status”
parameter 9 value	AOM synthesizer status, see below
parameter 10 string	“freq_ref_source”
parameter 10 value	“internal” or “external”
parameter 11 string	“main_lo_source”
parameter 11 value	“internal” or “external”
parameter 12 string	“main_input_power”
parameter 12 value	Main input power value
parameter 13 string	“main_input_prescaler”
parameter 13 value	1, 2, 4 or 8
parameter 14 string	“aux_input_power”
parameter 14 value	Aux input power value
parameter 15 string	“aux_input_prescaler”
parameter 15 value	1, 2, 4 or 8
parameter 16 string	“main_lock_error”

parameter 16 value	Main lock error value
parameter 17 string	“aux_lock_error”
parameter 17 value	Aux lock error value
parameter 18 string	“eom_drive”
parameter 18 value	EOM drive value
parameter 19 string	“if_lock_error”
parameter 19 value	IF lock error value
parameter 20 string	“main_lock_status”
parameter 20 value	Current lock condition, see below.
parameter 21 string	“resonator_voltage”
parameter 21 value	Resonator voltage value
parameter 22 string	“aux_lock_status”
parameter 22 value	Current lock condition, see below.
parameter 22 string	“ecd_lock_status”
parameter 22 value	Current lock condition, see below.

Status value

0	-	operation completed.
1	-	operation failed.

Synthesizer status value

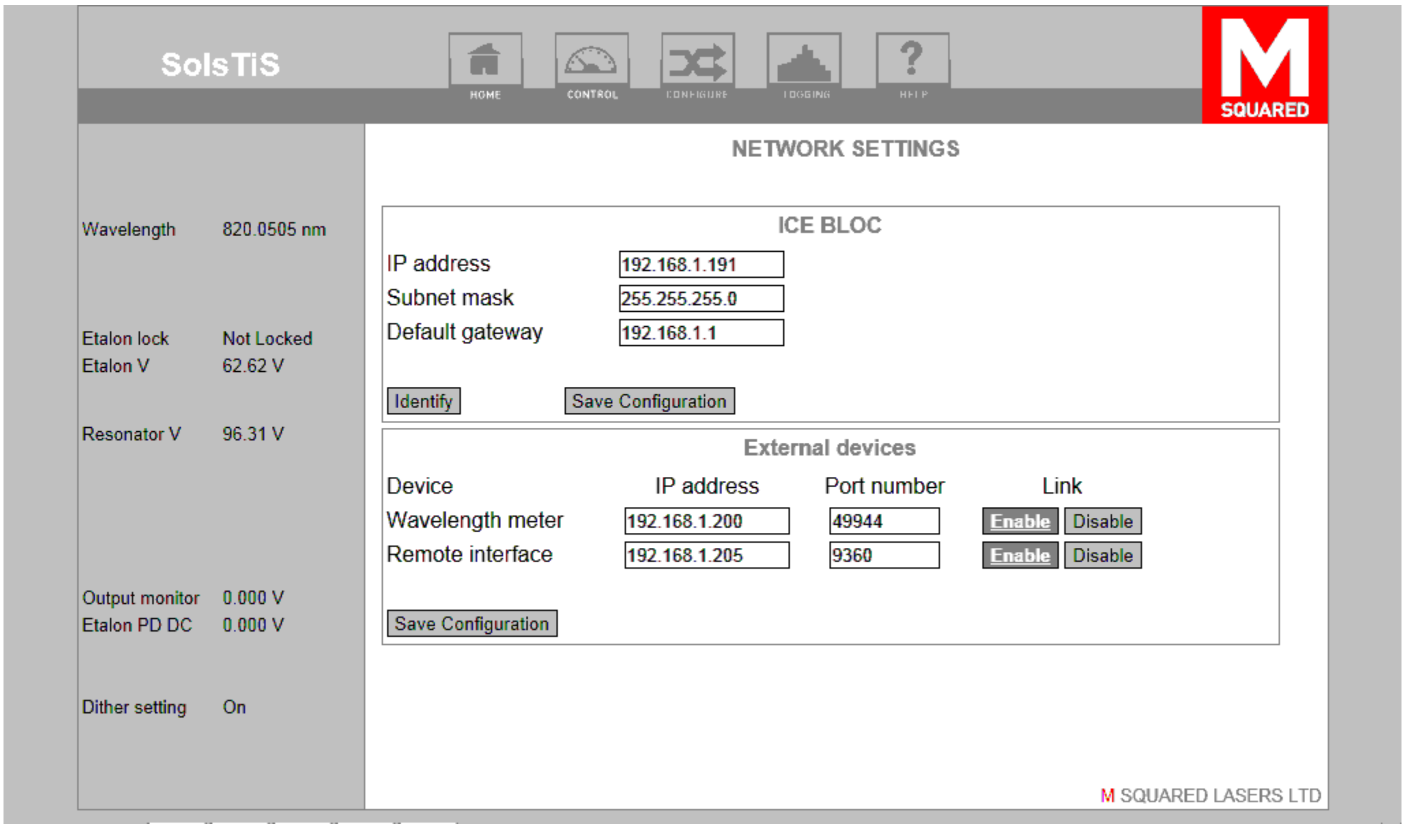
0	-	OK
1	-	VCO out of limits

Current lock condition

“off”	-	the lock is off
“on”	-	the lock is on
“debug”	-	the lock is in a debug condition
“error”	-	the lock operation is in error
“search”	-	the lock search algorithm is active
“low”	-	the lock is off due to low output.

Appendix 1 – Worked Examples

Appendix 1.1 Configuring the Port, Network Page



NETWORK SETTINGS

Wavelength	820.0505 nm
Etalon lock	Not Locked
Etalon V	62.62 V
Resonator V	96.31 V
Output monitor	0.000 V
Etalon PD DC	0.000 V
Dither setting	On

ICE BLOC

IP address	192.168.1.191
Subnet mask	255.255.255.0
Default gateway	192.168.1.1

Identify Save Configuration

External devices

Device	IP address	Port number	Link
Wavelength meter	192.168.1.200	49944	Enable Disable
Remote interface	192.168.1.205	9360	Enable Disable

Save Configuration

M SQUARED LASERS LTD

The Network page which is entered from the Configuration page allows the user to define the IP address and port for the Remote interface, see earlier section 1.1 of this document. Enable and disable buttons allow the user to switch on and switch off the interface. If the Remote interface is enabled when Save Configuration is pressed the communications link will be opened whenever power is applied.

Appendix 1.2 Connecting to the Remote interface.

Once the Remote interface is enabled the Phase Lock end of the interface behaves as a TCP/IP server to any Clients which attach to it. The code snippet shown below is the beginning of a Client sequence we have used previously on other projects. The machine attempting to communicate with the Phase Lock must contain a Client function similar to that below and then exchange the Start Link command defined in section 2.2 of this document. Once this has been successfully completed the link is open and the Phase Lock will respond to the commands defined in sections 2 and 3 of this document.

```
--------------------------------------------------------------------------------

Client_Sock = socket(AF_INET, SOCK_STREAM, 0);
if (Client_Sock < 0)
      printf("socket failed\n");

Socket_Def.sin_family = AF_INET;
ip_addr = Sys.Network.wavelength_ip_address.val;
Socket_Def.sin_addr.s_addr = ip_addr;     // Client needs address of server

Socket_Def.sin_port = htons(Sys.Network.wavelength_port.val);
length = sizeof(struct sockaddr_in);

// the socket is blocking at this moment

status = connect(Client_Sock, (struct sockaddr *)&Socket_Def, length);
printf("Connect status %d\n", status);
if (status < 0)
{
      printf("Client failed to connect, status %d\n", status);
      tx_thread_sleep(100);
}

if (status == 0)
{
      // connection OK, send and receive the opening messages

      memset(Client_Out_Buffer, 0, 1000);
      Build_Start_Link (Client_Out_Buffer);
      t_length = strlen(Client_Out_Buffer);

      // make the socket non blocking

      setting = 1;
      status = setsockopt(Client_Sock, SOL_SOCKET, SO_NONBLOCK,
                                    (char *)&setting, sizeof(int));
      // printf("setsocketopt status = %d\n", status);

      while (TRUE)
      {
            // send the opening message

            // printf("Sending message after connect\n");
            out = send(Client_Sock, Client_Out_Buffer, t_length, 0);
            in_ptr = Client_In_Buffer;
            memset(Client_In_Buffer, 0, 1000);
            tx_thread_sleep(20);
            reply_on_count = 0;
            got_data = FALSE;

            while(TRUE)
            {
                  // loop until the reply is complete

                  in = recv(Client_Sock, in_ptr, 1000, 0);

                  // this reply will not exceed 1000

                  if (in > 0)
                  {
                        // got data
-----------------------------------------------------------------
```

Appendix 1.3 Start Link Command

A sample Start Link command is shown here.

```
{"message":{"transmission_id":[1],"op":"start_link","parameters":{"ip_address":"192.
168.1.205"}}}
```

The reply from the Phase Lock is as follows:-

```
{"message":{"transmission_id":[1],"op":"start_link_reply","parameters":{"ip_address"
:"192.168.1.191","status":"ok"}}}
```

In this example the Client introduces itself from address 192.168.1.205. This must match the address entered on the Network page of the Phase Lock shown in Appendix 1.1. The Phase Lock replies with its own IP address and an "ok" status. The link is now ready for operation.

Appendix 1.4 Ping Command

A sample Ping command is shown here. This command may be issued at any time to confirm that the TCP link is operating satisfactorily.

```
{"message":{"transmission_id":[2],
"op":"ping","parameters":{"text_in":"ABCDEFabcdef"}}}
```

The reply from Phase Lock is as follows:-

```
{"message":{"transmission_id":[2],"op":"ping_reply","parameters":{"text_out":"abcde
fABCDEF"}}}
```

When the ping command is executed the Phase Lock converts upper case characters to lower case and lower case characters to upper case. Hence in the above example the string "ABCDEFabcdef" becomes "abcdefABCDEF". e.g. "Glasgow" would become "gLASGOW".