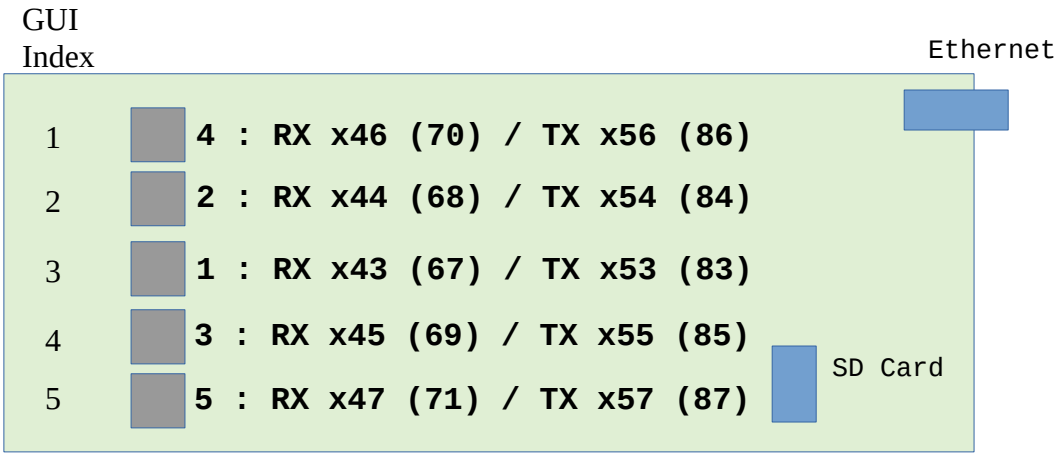
2019-06-17 Leap Module Development

There are 5 Leap module locations on the Talon-DX motherboard. On those 5 locations, 10 leap modules can be installed – 5 for RX and 5 for TX.

The I2C mapping for these devices are:



Since no Linux driver exists for these devices in the current Linux distribution, control is not specified via the device tree file. Rather, the TANGO device server is designed to access the I2C bus directly to communicate with the physical devices.

The above diagram shows the placement of the 5 Leap module sites, with orientation on the motherboard provided by the placement of the SFP module providing Ethernet connectivity (top right) and the SD card providing the root filesystem and bitstream images.

Each Leap site is associated with an index (1 through 5) with corresponding i2c addresses for both the RX and TX leap modules that may be affixed to each site. The hex based addresses (prefixed by ‘x’) are the value that the sysfs system uses to reference the individual modules on each site. These addresses are provided to the leap module TANGO device server as a decimal number (provided in brackets).

Testing configuration is with sites 1 and 2 populated with both RX and TX Leap modules. Site 2 has a fiber optic loopback cable attached.

LeapID

The LeapID value is stored as a device property, and represents the position of the Leap MBO module on the Talon-DX board. This value may

be an index for i2c address values which could be provided by a lookup table in the device server header file. See diagram above.

LeapType

The leapType value is stored as a device property. This value indicates whether the device is a TX or RX module. (TX=0, RX=1). This value is necessary as some properties of the MBO do not share exact register addressing across leapType from one module to another. Upon instantiation of a device within the server, the device property leapType will be set, and not changed.

Temperature

The TANGO device reads 2 8-bit registers to compose the value for temperature. The first 8 bits encode the integer part of the temperature in 2’s complement. The second 8 bits encode the decimal part of the temperature as a fractional part in units of 1/256 degrees, coded in binary. The TANGO device returns an integer value that comprises both 8-bit numbers, the higher order 8 bits represents the integer part, and the lower order 8 bits represents the fractional part. The python GUI is responsible for conversion into a floating point displayed number.

Note that the temperature monitor point is only available on the TX module.

Voltage

Voltage is read and returned by the TANGO device as a 16-bit integer in units of 100uV. The python GUI is responsible for conversion to integer voltage value for display. (divide by 10^6 to read out value in Volts).

Note Vcc33 and VccHI are supported for TX, and only Vcc33 is supported for RX.

Control

Since there is no viable driver in the Linux distribution for the LEAP MBO module, we are forced to communicate with this module over I2C bus, using the i2ctools library. A TANGO device has been written, based on the TalonDXBSMCBase module called TalonDXBSMCLeap.

The TalonDXBSMCBase module contains references to all the required i2c commands necessary to read and write values on the i2c bus.

The TalonDXBSMCLeap module has methods written which take advantage of the base class functions, to read and write in the various formats required to communicate with the data structures on the module.

TANGO Device Server Base Class

Base Class

The TANGO device server is built upon the TalonDXBSMCBase class. This class provides basic information for the device under control, which is stored in the TANGO database as device properties. The list of device properties in the base class are as follows:

- **busID** – the I2C bus address for the HW device
- **i2cAddress** – the I2C device address for the HW device
- **mod** – the device module name (as referenced in the device tree file)
- **compat** – the device compatibility string as detected by the Linux driver (if available)
- **name** – the device name (not required)
- **deviceID** – internal index for device (if multiple devices)
- **hwFilter** – a string used to construct the *hwPath* variable within the TANGO device server.
- **HwPrefix** – a string used to construct the *hwPath* variable within the TANGO device server.

One additional device property in the base class is ***hwPath***. This property is constructed within the *init* method of the TANGO device server. This is the path where device attributes are exposed to the user space within the HPS Linux distribution.

A suite of TANGO device commands accompany these attributes within the base class, providing access to the values to any software able to access the device.

Note that the device properties associated with i2c-sysfs addressing and attribute location are not used within this TANGO device server. Instead, functions exist within the TalonDXBSMCBase class to access the I2C bus directly: *i2c_read* and *i2c_write*. The device server is still reliant on the device properties for storage of addressing information for the individual physical devices.

In addition to the device properties supplied by the base class, this TANGO device server requires two additional device properties to identify specific Leap MBO modules:

- **leapID** – the internal leap module ID (1-5 – see diagram above)
- **leapType** – an integer indicating the leap module type (TX=0, RX=1)

One TANGO device per Leap Module per functionality (RX or TX) is required to be defined within the TANGO database. This amounts to 10 individual TANGO devices, accessed by the same device server instance.

Device Class

The TANGO device server class responsible for monitor and control of the Leap MBO controller is named TalonDXBSMCLeap. The class provides monitor and control over the following scalar attributes:

- **temperature** – reading from the temperature sensor (TX module only)
- **voltage33** – reading from the input voltage 3.3V (TX & RX modules)
- **voltageHI** – reading from the input voltage HI (TX & RX)
- **presenceReg** – the presence register value, provides information about the physical hardware
- **statusReg** – the status register providing aggregate station information
- **globalCDR** – provides control of the global Clock Data Recovery bypass setting
- **lossSignal** – provides the signal loss flag for all 12 channels encoded as 12 bits within a 16 bit value
- **lossLock** – provides the signal lock loss flag for all 12 channels encoded as 12 bits within a 16 bit value
- **fault** – provides the fault flag bit for all 12 channels encoded as 12 bits within a 16 bit value
- **disable** – provides the channel disable bit for all 12 channels encoded as 12 bits within a 16 bit value
- **bypassCDR** – provides the Clock Data Recovery bypass control bit for all 12 channels encoded as 12 bits within a 16 bit value
- **outputDisable** – provides the output disable control bit for all 12 channels encoded as 12 bits within a 16 bit value

In addition to the scalar attributes, the TANGO device server provides monitor and control over the following spectrum attributes:

- **inputEQ** – the input equalization value for Leap TX modules. This value is validated by the GUI as being in the range 0x0 to 0xf. The inputEQ value is encoded over 6 internal 8-bit device registers, with pairs of channels being encoded per register. This is abstracted by the TANGO device server, which provides a 12 element array of DevULong values.
- **inputMidEQ** – the input mid equalization value for Leap TX modules. This value is validated by the GUI as being in the range 0x0 to 0xf. The inputEQ value is encoded over 6 internal

8-bit device registers, with pairs of channels being encoded per register. This is abstracted by the TANGO device server, which provides a 12 element array of DevULong values.

- **outputAmplitude** – the output amplitude value for Leap RX modules. This value is validated by the GUI as being in the range 0x0 to 0x7. The inputEQ value is encoded over 6 internal 8-bit device registers, with pairs of channels being encoded per register. This is abstracted by the TANGO device server, which provides a 12 element array of DevULong values.
- **outputDeemphasis** – the output de-emphasis value for Leap RX modules. This value is validated by the GUI as being in the range 0x0 to 0xf. The inputEQ value is encoded over 6 internal 8-bit device registers, with pairs of channels being encoded per register. This is abstracted by the TANGO device server, which provides a 12 element array of DevULong values.

All attributes are polled within the TANGO device server on a 3s interval. All attributes are READ ONLY. Changes to attribute values are accomplished through TANGO commands as described below:

- **regWrite** – Generic register write. Writes an 8-bit value to the provided register address. The input argument for this command is a DevULong value. Bits within the input argument masked at 0xff encode the register address. The value to be written is masked at 0xff00.
- **RegRead** – Generic register read. Reads an 8-bit value from the specified register address. The input argument is the 8 bit register address and the output is the value read from the specified register, provided as a DevULong integer.
- **GetLeapType** – returns the leap type device property of the current leap module. TX=0, RX=1
- **getLeapID** – returns the internal leap ID (GUI ID from the diagram above) for the current leap module (range is 1-5)
- **reset** – invokes a reset of the device by writing to the reset register.
- **BypassON** – activates the channel CDR bypass for the specified channel. Input argument is the channel ID (0-11)
- **BypassOFF –** deactivates the channel CDR bypass for the specified channel. Input argument is the channel ID (0-11)
- **bypassALL** – writes the channel CDR bypass for all channels simultaneously. The input argument is the new CDR Bypass value (0=OFF, 1=ON).
- **getIntLStatus** – Returns the INT_L status (it is not clear according to the product specification what the INT_L value represents – fill this in based on information from hardware designer – todo).
- **GetTXLOSStatusSummary** – returns the Loss of Signal Status summary for the TX device. This is a 16 bit number (read from 2 consecutive registers from the TX module) that encodes one bit

for each channel. Each bit signifies the loss of signal status for that channel. Channels are indexed from 0 through 11. Bits 12 through 15 are discarded.

- **GetRXLOSStatusSummary** – returns the Loss of Signal Status summary for the RX device. This is a 16 bit number (read from 2 consecutive registers from the RX module) that encodes one bit for each channel. Each bit signifies the loss of signal status for that channel. Channels are indexed from 0 through 11. Bits 12 through 15 are discarded.
- **GetTXLOLStatusSummary** – returns the Loss of Lock Status summary for the TX device. This is a 16 bit number (read from 2 consecutive registers from the TX module) that encodes one bit for each channel. Each bit signifies the loss of lock status for that channel. Channels are indexed from 0 through 11. Bits 12 through 15 are discarded.
- **GetRXLOLStatusSummary** – returns the Loss of Signal Status summary for the RX device. This is a 16 bit number (read from 2 consecutive registers from the RX module) that encodes one bit for each channel. Each bit signifies the loss of signal status for that channel. Channels are indexed from 0 through 11. Bits 12 through 15 are discarded.
- **GetTXFaultStatusSummary** – returns the Fault Status summary for the TX device. This is a 16 bit number (read from 2 consecutive registers from the TX module) that encodes one bit for each channel. Each bit signifies the fault status for that channel. Channels are indexed from 0 through 11. Bits 12 through 15 are discarded.
- **GetRXFaultStatusSummary** – returns the Fault Status summary for the RX device. This is a 16 bit number (read from 2 consecutive registers from the RX module) that encodes one bit for each channel. Each bit signifies the loss of signal status for that channel. Channels are indexed from 0 through 11. Bits 12 through 15 are discarded.
- **GetTXBiasStatusSummary** – returns the TX Bias Hi-Lo Alarm Status summary for the TX device. This is a 16 bit number (read from 2 consecutive registers from the TX module) that encodes one bit for each channel. Each bit signifies the Bias Hi-Lo Alarm status for that channel. Channels are indexed from 0 through 11. Bits 12 through 15 are discarded.
- **GetTXModuleStatusSummary** – returns the Module Status summary for the device. This is a 16 bit number (read from 2 consecutive registers from the TX module) that encodes one bit for each channel. Each bit signifies the module status for that channel. Channels are indexed from 0 through 11. Bits 12 through 15 are discarded. Ths alarm is an aggregate alarm, asserted when any alarm is asserted on the TX module.
- **GetRXModuleStatusSummary** – returns the Module Status summary for the device. This is a 16 bit number (read from 2 consecutive registers from the RX module) that encodes one bit for each

channel. Each bit signifies the module status for that channel. Channels are indexed from 0 through 11. Bits 12 through 15 are discarded. Ths alarm is an aggregate alarm, asserted when any alarm is asserted on the RX module.

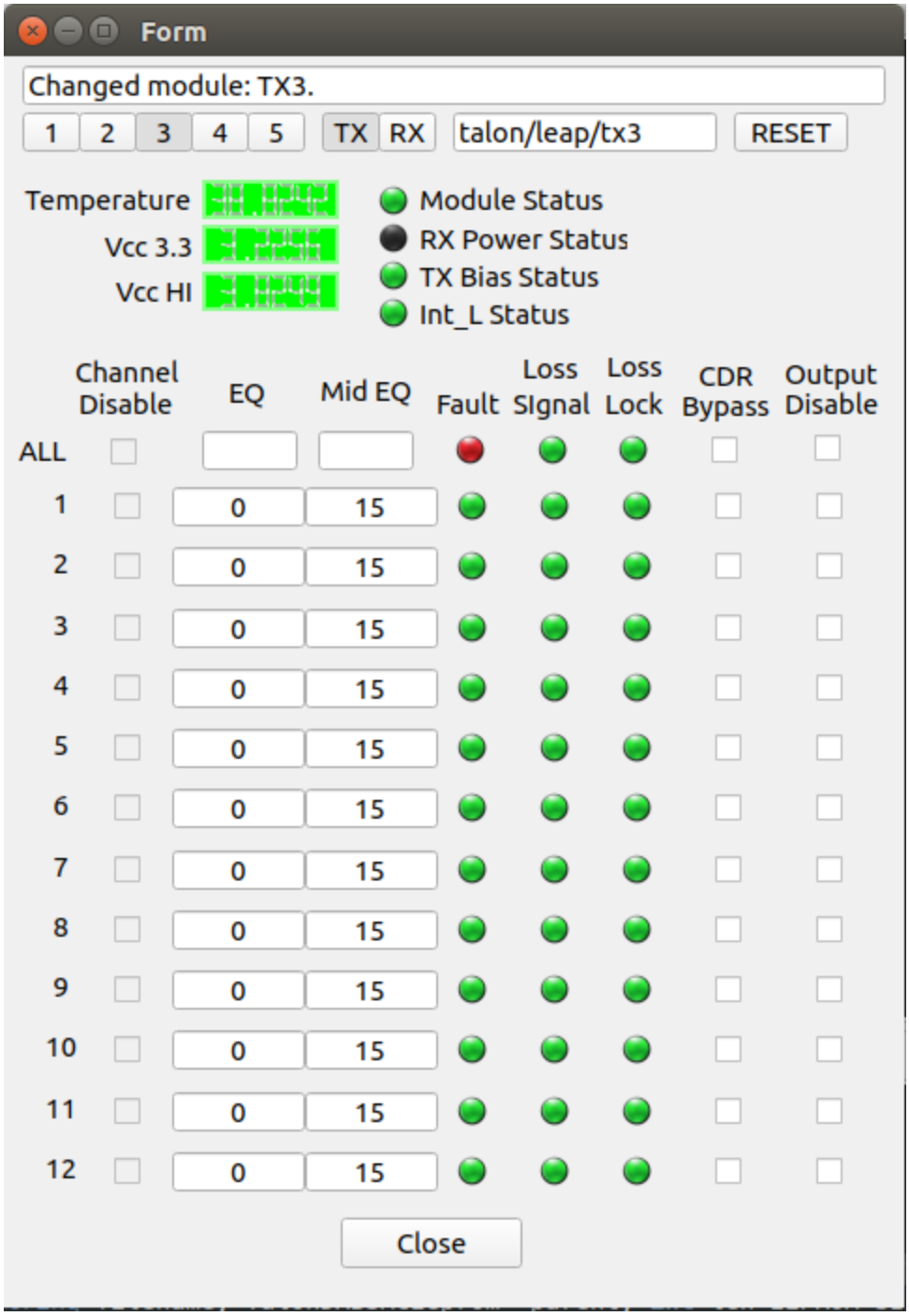
- **GetRXPowerStatusSummary** – returns the Power Status summary for the device. This is a 16 bit number (read from 2 consecutive registers from the RX module) that encodes one bit for each channel. Each bit signifies the module status for that channel. Channels are indexed from 0 through 11. Bits 12 through 15 are discarded.
- **SetGlobalCDR** – Writes a new value to the global CDR register. Writing 0 disables CDR globally to all channels. Writing 1 permits the device to follow individual channel TX/RX CDR configuration settings. The input parameter is the value to write.
- **OutputENABLE** – writes 1 to the the output enable bit in order to enable the specified TX/RX output channel. The input argument is the channel ID (0-11).
- **OutputDISABLE** – writes 0 to the output enable bit in order to disable the specified TX/RX output channel. The input argument is the channel ID (0-11).
- **outputALL** – writes the output enable bit identially for all channels simultaneously. The input argument is the new output bypass value (0=OFF, 1=ON)
- **channelENABLE** – writes 1 to the channel enable bit in order to enable the specified RX channel. The input argument is the channel ID (0-11). Not provided for TX modules.
- **channelDISABLE** – writes 0 to the channel enable bit in order to disable the specified RX channel. The input argument is the channel ID (0-11). Not provided for TX modules.
- **channelALL** – writes the channel enable bit identially for all channels simultaneously. The input argument is the new channel bypass value (0=OFF, 1=ON)
- **SetRXOutputAmplitude** – writes a new output amplitude value to an RX output channel register. The input parameter is a 32-bit DevULong value. The 0xf bits in the input parameter value mask the channel ID (0-11) and the 0x70 bits in the input parameter mask the new output amplitude value to be written.
- **SetRXOutputDeemphasis** – writes a new output de-emphasis value to an RX output channel register. The input parameter is a 32-bit DevULong value. The 0xf bits in the input parameter value mask the channel ID (0-11) and the 0x70 bits in the input parameter mask the new output deemphasis value to be written.
- **SetTXInputEQ** – writes a new input equalization value to a TX input channel register. The input parameter is a 32-bit DevULong value. The 0xf bits in the input parameter value mask the channel ID (0-11) and the 0xf0 bits in the input parameter mask the new input equalization value to be written.

- **SetTXInputMidEQ** – writes a new input mid equalization value to a TX input channel register. The input parameter is a 32-bit DevULong value. The 0xf bits in the input parameter value mask the channel ID (0-11) and the 0xf0 bits in the input parameter mask the new input mid equalization value to be written.
- **SetRXOutputAmplitudeALL** – permits the user to apply one output amplitude value to all channels simultaneously. The input parameter is the new value for output amplitude. The accepted range is 0x0-0x7, and the input value is masked internally as such.
- **SetRXOutputDeemphasisALL** – permits the user to apply one output deemphasis value to all channels simultaneously. The input parameter is the new value for output deemphasis. The accepted range is 0x0-0x7, and the input value is masked internally as such.
- **SetTXInputEQALL** – permits the user to apply one input equalization value to all channels simultaneously. The input parameter is the new value for input equalization. The accepted range is 0x0-0xf, and the input value is masked internally as such.
- **SetTXInputMidEQALL** – permits the user to apply one input mid equalization value to all channels simultaneously. The input parameter is the new value for input mid equalization. The accepted range is 0x0-0xf, and the input value is masked internally as such.

All register addresses specified in command descriptions are register addresses on the Leap MBO device. Please refer to the product specification for register descriptions.

GUI

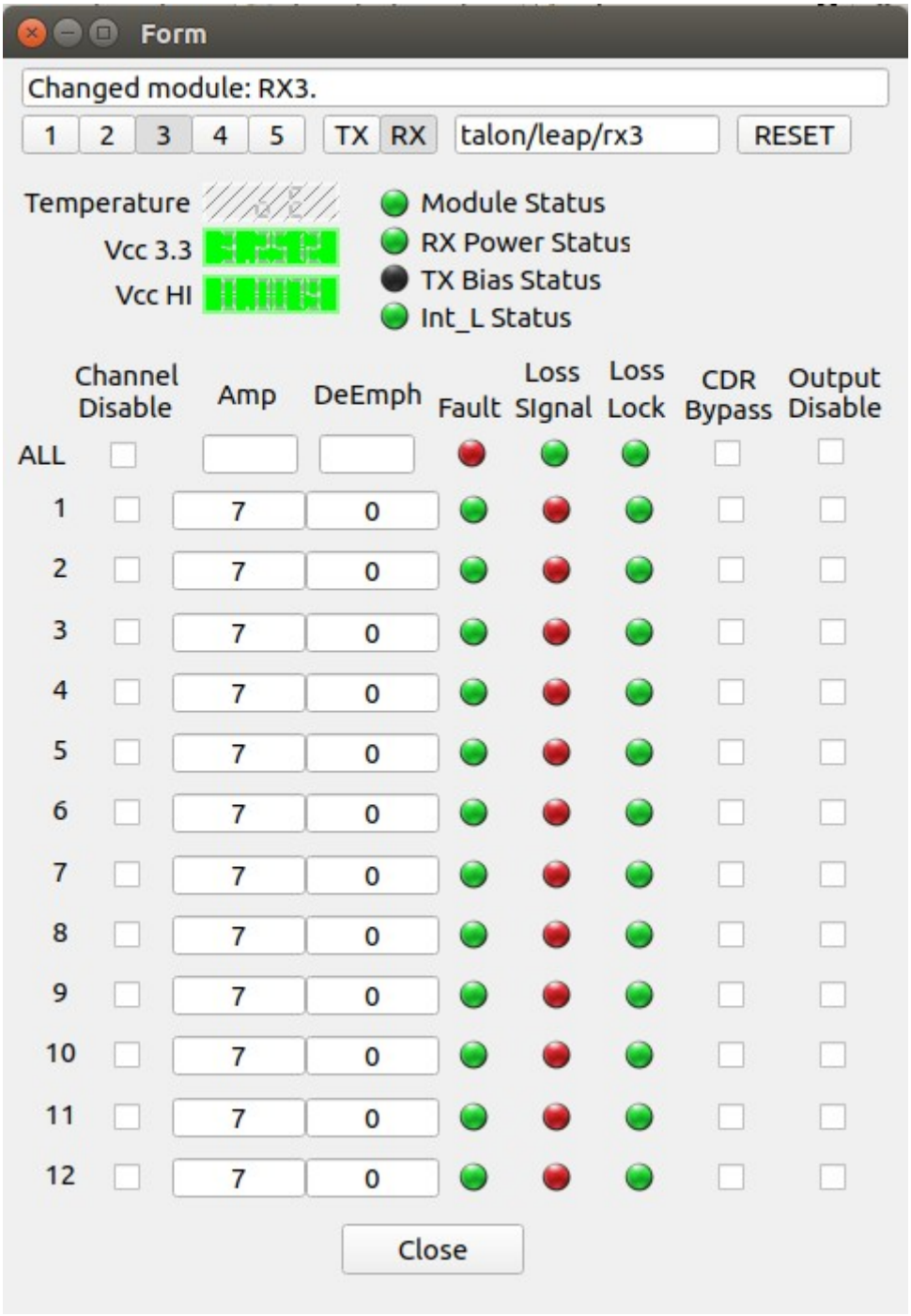
Two separate pages of GUI are required to adequately represent both the RX and TX Leap MBO modules.



The user may select specific modules via the selector at the top of each screen in the standalone version of the GUI. If selected from the main window, the control for the selected module only will appear, and the user will not be able to navigate to another module from this display.

The user is able to modify any input parameter within the channel table. Parameters providing global access are at the top of the display list. Alarms are indicated by LED display. Aggregate alarms are included at the top of the channel list.

Module-level monitor points and alarm indicators are provided at the top of the page, just below the selector. These are visible only as required, based on whether the user has selected a TX or RX module, as these modules do not necessarily share all monitor points.



The internal GUI working prevent out of range values from being written to the device by validating all input prior to action. If an input value is out of range, the previous value is replaced, and the user is informed that the value is out of range via the status display at the top of the window.