Scorpio Hub Design

Introduction / Scope

This is intended to define all software objects which in one way or another provide a Scorpio specific interface to functionality present on the FPGA on the HUB PCB present in the Scorpio motion platform / Equinox product. Note that in some cases this will just link to other pages which will then go into more detail about the technical aspects of the functionality or design.

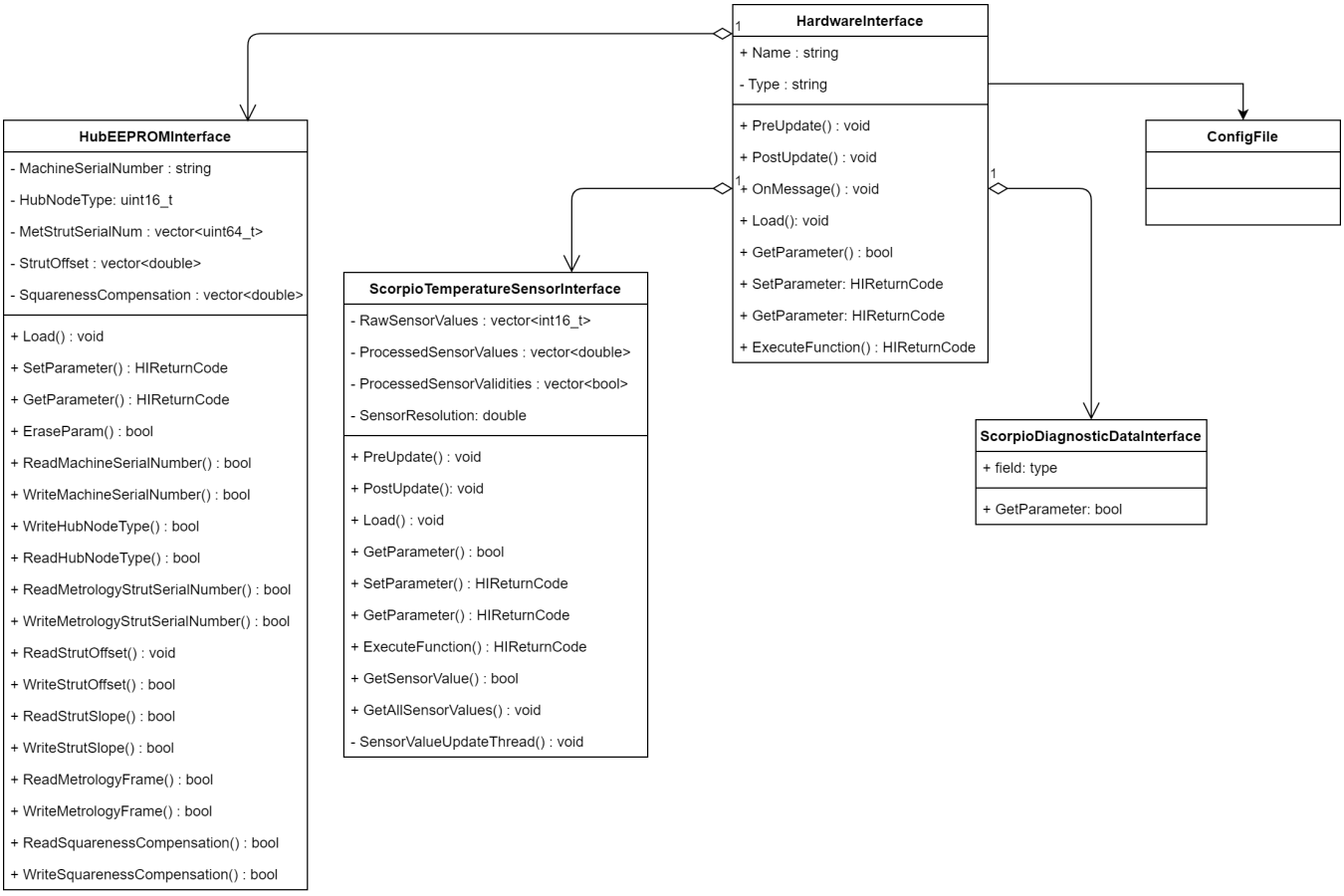
Functionality Definition

Fundamentally, what the electronics exposes can be split into the following high level aspects

- General Diagnostic Data - primarily accessed via the ScorpioDiagnosticDataInterface although some historic data will be recorded in the EEPROM and accessed via the ScorpioHUBEEPROMInterface.
- Temperature Sensor Interface - technical aspects defined in Scorpio Temperature Sensor Retrieval although functionally exposed via the ScorpioTemperatureSensorInterface object
- Timer & Watchdog Functionality - implemented in the ScorpioTimer and ScorpioWatchdog objects

Object Design & Inheritance

The diagram below shows the objects which provide Scorpio specific functionality along with any objects they directly inherit from or use.



ScorpioDiagnosticDataInterface

The ScorpioDiagnosticDataInterface object is really a catch all object for diagnostic data that is disparate such that there isn't an obvious wider bit of functionality it fits under.

Constructor()

Definition: This is the first in a two stage initialisation process, with the second stage using the information from the configuration file. Here we predominantly configure the parent class and make sure the object is in a fail "safe" mode.

Parameter Definition:

- in Name - An alphanumeric string indicating the (unique) identifier for an instance of the object, which will be used by the rest of the Generic Controller codebase to access it

- in ErrorHandlerMsgQ - A pointer to the message queue wrapper object owned by the Error Handler through which all errors are sent.

Return: No return defined.

PreUpdate()

Definition: This would normally be used to perform feedback based servo clock operations but as this object is for accessing diagnostic data it doesn't need to do anything.

Parameter Definition:

- in LoopIndex - The identifier for the servo clock making this call.

Return: None.

PostUpdate()

Definition: This would normally be used to perform demand based servo clock operations but but as this object is for accessing diagnostic data it doesn't need to do anything.

Parameter Definition:

- in LoopIndex - The identifier for the servo clock making this call.

Return: None.

Load()

Definition: This is used to configure the object from entries defined in the configuration file. The following entries in the config file are expected to be present, with no additional optional parameters:

- HumidityResolution - The value in degrees C of a single count in the raw sensor data.

Parameter Definition:

- in Params - A map of alphanumeric string to alphanumeric string defining the entries in the config file section for this object.

Return: None.

GetParameter

Definition: This is the entry point for the Data Logger object to access values within the object, routed through from the HardwareInterface base class. For information on the supported parameters, including potential values, please refer to the Scorpio Temperature Sensor Interface section in Scorpio Data Logger Parameters.

Parameter Definition:

- in ParameterID - A signed 16-bit integer defining the identity of the value to be retrieved. *Not sure why the type is 16-bit as it gets sent over comms as a string*
- out Value - The double precision representation of the requested parameter
- out ScalingFactor - The resolution of a single count when the value is converted into a 32-bit integer for transmission across the communications channel
- out IsUnsignedData - An indication of whether the value is signed or unsigned so as to be able to use the maximum possible range when converting to a 32-bit integer for transmission across the communications channel.

Return: A Boolean indicating whether the value has been successfully retrieved or not.

SetParameter()

Definition: This is the user request to set a parameter routed through the HardwareInterface base class and exposed to the user in the COM dll defined in HardwareInterface.dll Interface. Currently we do not support the setting of any parameters.

Parameter Definition:

- in ParamNum - A signed 32-bit integer defining the identity of the value to be retrieved
- in Values - An array of HIParam objects (which are essentially an individual discriminated union) containing the new value or values of the defined parameter.

Return: A value in the enumerated type EHIReturnCode, defined in HardwareInterface.dll Interface although always the value HI_PARAM_NOT_RECOGNISED.

GetParameter()

Definition: This is the user request to get a parameter value routed through the HardwareInterface base class and exposed to the user in the HardwareInterfaceInterface COM interface defined in HardwareInterface.dll Interface. Currently we support the same parameters as the Data Logger GetParameter function and as such use the same parameter identifier mapping - the exception to this is the parameter identifier 1000 which retrieves all 13 processed temperature sensor values. Note that if the processed sensor value is invalid then this function will override the error value of 0, which could be a valid value, and report a value of -273.0.

Parameter Definition:

- in ParamNum - A signed 32-bit integer defining the identity of the value to be retrieved
- out Values - An array of HIParam objects (which are essentially an individual discriminated union) containing the value or values of the requested parameter

Return: A value in the enumerated type EHIReturnCode, defined in HardwareInterface.dll Interface. When the parameter identity is recognised and a value is set the value of HI_SUCCESS, will be returned although if the parameter identity is not recognised then a value of HI_PARAM_NOT_RECOGNISED will be returned.

ExecuteFunction()

Definition: This is the user request to perform a functional task, routed through the HardwareInterface base class and exposed to the user in the COM dll defined in HardwareInterface.dll Interface. Currently we do not support the execution of any functional tasks.

Parameter Definition:

- in FunctionNum - A signed 32-bit integer defining the identity of the functional task to be executed.
- in Values - An array of HIParam objects (which are essentially an individual discriminated union) containing a value or values required by the functional task.

Return: A value in the enumerated type EHIReturnCode, defined in HardwareInterface.dll Interface although always the value HI_FUNCTION_NOT_RECOGNISED.

ScorpioTemperatureSensorInterface

The ScorpioTemperatureSensorInterface brings together the 13 different temperature sensors exposed on the Scorpio HUB FPGA and deals with the fact we want to be able to both treat them independently and as a group. Note that in order to interface to the rest of the Generic Controller codebase it will inherit from the HardwareInterface object.

Constructor()

Definition: This is the first in a two stage initialisation process, with the second stage using the information from the configuration file. Here we predominantly configure the parent class and make sure the object is in a fail "safe" mode.

Parameter Definition:

- in Name - An alphanumeric string indicating the (unique) identifier for an instance of the object, which will be used by the rest of the Generic Controller codebase to access it
- in ErrorHandlerMsgQ - A pointer to the message queue wrapper object owned by the Error Handler through which all errors are sent.

Return: No return defined.

PreUpdate()

Definition: This would normally be used to perform feedback based servo clock operations but due to the frequency of updating it doesn't need to do anything.

Parameter Definition:

- in LoopIndex - The identifier for the servo clock making this call.

Return: None.

PostUpdate()

Definition: This would normally be used to perform demand based servo clock operations but due to the frequency of updating and the fact the object is just feedback based means it doesn't need to do anything.

Parameter Definition:

- in LoopIndex - The identifier for the servo clock making this call.

Return: None.

Load()

Definition: This is used to configure the object from entries defined in the configuration file. The following entries in the config file are expected to be present, with no additional optional parameters:

- SensorResolution - The value in degrees C of a single count in the raw sensor data.

Parameter Definition:

- in Params - A map of alphanumeric string to alphanumeric string defining the entries in the config file section for this object.

Return: None.

GetParameter

Definition: This is the entry point for the Data Logger object to access values within the object, routed through from the HardwareInterface base class. For information on the supported parameters, including potential values, please refer to the Scorpio Temperature Sensor Interface section in Scorpio Data Logger Parameters.

Parameter Definition:

- in ParameterID - A signed 16-bit integer defining the identity of the value to be retrieved. *Not sure why the type is 16-bit as it gets sent over comms as a string*
- out Value - The double precision representation of the requested parameter
- out ScalingFactor - The resolution of a single count when the value is converted into a 32-bit integer for transmission across the communications channel
- out IsUnsignedData - An indication of whether the value is signed or unsigned so as to be able to use the maximum possible range when converting to a 32-bit integer for transmission across the communications channel.

Return: A Boolean indicating whether the value has been successfully retrieved or not.

SetParameter()

Definition: This is the user request to set a parameter routed through the HardwareInterface base class and exposed to the user in the COM dll defined in HardwareInterface.dll Interface. Currently we do not support the setting of any parameters.

Parameter Definition:

- in ParamNum - A signed 32-bit integer defining the identity of the value to be retrieved
- in Values - An array of HIParam objects (which are essentially an individual discriminated union) containing the new value or values of the defined parameter.

Return: A value in the enumerated type EHIReturnCode, defined in HardwareInterface.dll Interface although always the value HI_PARAM_NOT_RECOGNISED.

GetParameter()

Definition: This is the user request to get a parameter value routed through the HardwareInterface base class and exposed to the user in the HardwareInterfaceInterface COM interface defined in HardwareInterface.dll Interface. Currently we support the same parameters as the Data Logger GetParameter function and as such use the same parameter identifier mapping - the exception to this is the parameter identifier 1000 which retrieves all 13 processed temperature sensor values. Note that if the processed sensor value is invalid then this function will override the error value of 0, which could be a valid value, and report a value of -273.0.

Parameter Definition:

- in ParamNum - A signed 32-bit integer defining the identity of the value to be retrieved
- out Values - An array of HIParam objects (which are essentially an individual discriminated union) containing the value or values of the requested parameter

Return: A value in the enumerated type EHIReturnCode, defined in HardwareInterface.dll Interface. When the parameter identity is recognised and a value is set the value of HI_SUCCESS, will be returned although if the parameter identity is not recognised then a value of HI_PARAM_NOT_RECOGNISED will be returned.

ExecuteFunction()

Definition: This is the user request to perform a functional task, routed through the HardwareInterface base class and exposed to the user in the COM dll defined in HardwareInterface.dll Interface. Currently we do not support the execution of any functional tasks.

Parameter Definition:

- in FunctionNum - A signed 32-bit integer defining the identity of the functional task to be executed.
- in Values - An array of HIParam objects (which are essentially an individual discriminated union) containing a value or values required by the functional task.

Return: A value in the enumerated type EHIReturnCode, defined in HardwareInterface.dll Interface although always the value HI_FUNCTION_NOT_RECOGNISED.

GetSensorValue()

Definition: This will allow for the retrieval of a single processed temperature sensor value and it's validity. No data retrieval or processing will occur during the execution of this function

Parameter Definition:

- in SensorIdentity - A 32-bit signed integer defining the identity of the sensor to be retrieved
- out ProcessedSensorValue - The last processed value for the defined sensor
- out SensorValueValidity - The validity of the last processed sensor value.

Return: A Boolean value, where a value of true is returned if the sensor identity is recognised, and a value of false is returned if the sensor identity is not recognised.

GetAllSensorValues()

Definition: This will allow for the retrieval of all processed temperature sensor values and their corresponding validity. No data retrieval or processing will occur during the execution of this function

Parameter Definition:

- out ProcessedSensorValues - An array of the last processed value for each of the sensors with the identities between 0 and 12 inclusive
- out SensorValueValidities - An array of the validity of the last processed value for each of the sensors with the identities between 0 and 12 inclusive.

Return: None.

SensorValueUpdateThread()

Definition: This is the thread which will go through and process each of the 13 sensors so that we have fully processed sensor values and validities in line with the process defined in Scorpio Temperature Sensor Retrieval Note that, implementationally, it is expected that this function will be of private non static scope within the ScorpioTemperatureSensorInterface and that their will be a corresponding SensorValueUpdateThreadEntry private static function to cope with the fact the thread itself must take in a pointer to the object it is to operate on. The thread should use the priority "task band 4" and run at a frequency suitable to the update rate of the sensors. Note that the Data Moderation needs to happen outside of this thread as it needs the consistent tick of the servo clock as it is uses the time delta in it's calculations.

Parameter Definition:

- in This - a void pointer which actually contains the pointer to the ScorpioTemperatureSensorInterface object the thread is to operate on.

Return: None.

RawSensorValues

Definition: These are the raw 16-bit values read directly from the electronics. Primarily, these values are stored for diagnostics purposes.

Type: vector<int16_t>

ProcessedSensorValues

Definition: These are the fully processed temperature sensor values following the process defined in Scorpio Temperature Sensor Retrieval Note that, if the sensor value is not valid for any reason then internally we are going to set the processed value to 0.0 although any request for the value will return a value of -273.0, but only if the validity is not defined as part of the same request.

Type: vector<double>

ProcessedSensorValidities

Definition: These are the validity indications of the fully processed sensor values following the process defined in Scorpio Temperature Sensor Retrieval

Type: vector<bool>

SensorResolution

Definition: This is the resolution in degrees C per count. While it is expected that this value will remain constant at 0.0078 degrees C, the resolution will be exposed in the configuration file to future proof the software in case, at some point in the future, a different sensor will be used in a newer variant of the Scorpio electro-mechanical motion platform.

Type: double.

HubEEPROMInterface

The HubEEPROMInterface class will provide read/write access to the EEPROM data present on the scorpio Hub. Note that in order to interface to the rest of the Generic Controller codebase it will inherit from the HardwareInterface object.

Constructor()

Definition: This is the first in a two stage initialisation process, with the second stage using the information from the configuration file.

Parameter Definition:

- in Name - An alphanumeric string indicating the (unique) identifier for an instance of the object, which will be used by the rest of the Generic Controller codebase to access it
- in ErrorHandlerMsgQ - A pointer to the message queue wrapper object owned by the Error Handler through which all errors are sent.

Return: No return defined.

Load()

Definition: This is used to configure the object from entries defined in the configuration file. The following entries in the config file are expected to be present, with no additional optional parameters:

- EEPROMMemoryAddress - Memory address to access EEPROM present on the FPGA on the equinox hub.
- EEPROMOffsetAddress - Memory offset to point at the start of "Calibration data" section on the EEPROM.

Parameter Definition:

- in Params - A map of alphanumeric string to alphanumeric string defining the entries in the config file section for this object.

Return: None.

SetParameter()

Definition: This is the user request to set a parameter routed through the HardwareInterface base class and exposed to the user in the COM dll defined in HardwareInterface.dll Interface.

Parameter Definition:

- in ParamNum - A signed 32-bit integer defining the identity of the value to be retrieved
- in Values - An array of HIParam objects (which are essentially an individual discriminated union) containing the new value or values of the defined parameter.

Return: A value in the enumerated type EHIReturnCode, defined in HardwareInterface.dll Interface. HI_SUCCESS returned if the request was successful, HI_HARDWARE_FAILURE is returned if the request failed, HI_INVALID_VALUE is returned if any of the input parameters are invalid.

GetParameter()

Definition: This is the user request to get a parameter value routed through the HardwareInterface base class and exposed to the user in the HardwareInterfaceInterface COM interface defined in HardwareInterface.dll Interface.

Parameter Definition:

- in ParamNum - A signed 32-bit integer defining the identity of the value to be retrieved
- out Values - An array of HIParam objects (which are essentially an individual discriminated union) containing the value or values of the requested parameter

Return: A value in the enumerated type EHIReturnCode, defined in HardwareInterface.dll Interface. When the parameter identity is recognised and a value is set the value of HI_SUCCESS, will be returned although if the parameter identity is not recognised then a value of HI_PARAM_NOT_RECOGNISED will be returned.

EraseParam()

Definition: This is used to set the relevant data bytes of the selected parameter to 0xFF on the EEPROM.

Parameter Definition:

- in ParamNum - A signed 32-bit integer defining the identity of the value to be retrieved

Return: A bool value depending upon the outcome of the function. True if the selected parameter is erased successfully and the relevant data bytes is set to 0xFF, otherwise false will be returned.

ReadMachineSerialNumber()

Definition: This is used to read the "Machine Serial Number" parameter defined in Equinox Hub Memory Map from the EEPROM present on the scorpio hub.

Parameter Definition:

- out MachineSerialNumber - A string pointer that will contain the read machine's serial number.

Return: A bool value depending upon the outcome of the function. True if the machine serial number is read successfully, otherwise false will be returned.

WriteMachineSerialNumber()

Definition: This is used to write the "Machine Serial Number" parameter defined in Equinox Hub Memory Map to the EEPROM present on the scorpio hub.

Parameter Definition:

- in MachineSerialNumber - A string pointer containing the new machine serial number to be written.

Return: A bool value depending upon the outcome of the function. True if the machine serial number is written successfully, otherwise false will be returned.

ReadHubNodeType()

Definition: This is used to read the "Hub Node Type" parameter defined in Equinox Hub Memory Map from the EEPROM present on the scorpio hub.

Parameter Definition:

- out HubNodeType - An unsigned 16-bit integer pointer that will contain the read hub node type value.

Return: A bool value depending upon the outcome of the function. True if the hub node type is read successfully, otherwise false will be returned.

WriteHubNodeType()

Definition: This is used to write the "Hub Node Type" parameter defined in Equinox Hub Memory Map to the EEPROM present on the scorpio hub.

Parameter Definition:

- in HubNodeType - An unsigned 16-bit integer pointer containing the new hub node type value to be written.

Return: A bool value depending upon the outcome of the function. True if the hub node type is written successfully, otherwise false will be returned.

ReadMetrologyStrutSerialNumber()

Definition: This is used to read the "Metrology Strut Serial Numbers" parameter defined in Equinox Hub Memory Map from the EEPROM present on the scorpio hub.

Parameter Definition:

- in StrutNum- A size_t integer containing the strut number used to select a specific strut.
- out MetStrutSerialNumber - A string pointer that will contain the read metrology strut serial number of the selected strut.

Return: A bool value depending upon the outcome of the function. True if the metrology strut serial number is read successfully, otherwise false will be returned.

WriteMetrologyStrutSerialNumber()

Definition: This is used to write the "Metrology Strut Serial Numbers" parameter defined in Equinox Hub Memory Map to the EEPROM present on the scorpio hub.

Parameter Definition:

- in StrutNum- A size_t integer containing the strut number used to select a specific strut.
- in MetStrutSerialNumber - A string pointer containing the new value of metrology strut serial number to be written.

Return: A bool value depending upon the outcome of the function. True if the metrology strut serial number is written successfully, otherwise false will be returned.

ReadStrutOffset()

Definition: This is used to read the "Strut Offset" parameter defined in Equinox Hub Memory Map from the EEPROM present on the scorpio hub.

Parameter Definition:

- in StrutNum- A size_t integer containing the strut number used to select a specific strut.
- out StrutOffset - An unsigned 32-bit integer pointer that will contain the read strut offset of the selected strut.

Return: A bool value depending upon the outcome of the function. True if the strut offset is read successfully, otherwise false will be returned.

WriteStrutOffset()

Definition: This is used to write the "Strut Offset" parameter defined in Equinox Hub Memory Map to the EEPROM present on the scorpio hub.

Parameter Definition:

- in StrutNum- A size_t integer containing the strut number used to select a specific strut.
- in StrutOffset - An unsigned 32-bit integer pointer containing the new value of strut offset to be written.

Return: A bool value depending upon the outcome of the function. True if the strut offset is written successfully, otherwise false will be returned.

ReadStrutSlope()

Definition: This is used to read the "Strut Slope" parameter defined in Equinox Hub Memory Map from the EEPROM present on the scorpio hub.

Parameter Definition:

- in StrutNum- A size_t integer containing the strut number used to select a specific strut.
- out StrutSlope - An unsigned 32-bit integer pointer that will contain the read strut slope of the selected strut.

Return: A bool value depending upon the outcome of the function. True if the strut slope is read successfully, otherwise false will be returned.

WriteStrutSlope()

Definition: This is used to write the "Strut Slope" parameter defined in Equinox Hub Memory Map to the EEPROM present on the scorpio hub.

Parameter Definition:

- in StrutNum- A size_t integer containing the strut number used to select a specific strut.
- in StrutSlope - An unsigned 32-bit integer pointer containing the new value of strut slope to be written.

Return: A bool value depending upon the outcome of the function. True if the strut slope is written successfully, otherwise false will be returned.

ReadMetrologyFrame()

Definition: This is used to read the "Metrology Frame" with sub-parameter defined in Equinox Hub Memory Map from the EEPROM present on the scorpio hub.

Parameter Definition:

- out MetrologyFrame - A structure pointer that will contain the read metrology frame.

Return: A bool value depending upon the outcome of the function. True if the metrology frame is read successfully, otherwise false will be returned.

WriteMetrologyFrame()

Definition: This is used to write the "Metrology Frame" with sub-parameter defined in Equinox Hub Memory Map to the EEPROM present on the scorpio hub.

Parameter Definition:

- in MetrologyFrame - An structure pointer containing the new metrology frame to be written.

Return: A bool value depending upon the outcome of the function. True if the metrology frame is written successfully, otherwise false will be returned.

ReadSquarenessCompensation()

Definition: This is used to read the "Squareness Compensation" parameter defined in Equinox Hub Memory Map from the EEPROM present on the scorpio hub.

Parameter Definition:

- out MetrologyFrame - A structure pointer that will contain the read squareness compensation value.

Return: A bool value depending upon the outcome of the function. True if the squareness compensation value is read successfully, otherwise false will be returned.

WriteSquarenessCompensation()

Definition: This is used to write the "Squareness Compensation" parameter defined in Equinox Hub Memory Map to the EEPROM present on the scorpio hub.

Parameter Definition:

- in MetrologyFrame - An structure pointer containing the new squareness compensation value to be written.

Return: A bool value depending upon the outcome of the function. True if the squareness compensation value is written successfully, otherwise false will be returned.