# **Subrack Management API**

Release 1.0

Simone Chiarucci (INAF-OAA)

## **CONTENTS:**

1	Subrack management API Documentation	1
2	Web Server Documentation	13
3	CPLD Management API Documentation	27
4	<b>Subrack Tools Documentation</b>	43
5	Indices and tables	45
Ру	thon Module Index	47
Index		49

## SUBRACK MANAGEMENT API DOCUMENTATION

## exception backplane.BackplaneInvalidParameter

Exception class for invalid parameters provided to a function or class method.

## backplane.twos\_comp(val, bits)

Compute the two's complement of an integer value.

#### **Parameters**

- **val** (*int*) The integer value.
- **bits** (*int*) The number of bits representing the integer.

#### **Returns**

The two's complement of the input value.

#### **Return type**

int

## class management.MANAGEMENT(\*\*kwargs)

Class representing a MANAGEMENT instance.

## checkLoad()

Check if the registers are loaded.

#### Raises

**NameError** – If registers are not loaded or the board is not connected.

## disconnect()

Disconnect from the network.

This function closes the network connection and sets the state to "Unconnected".

#### Raises

None -

## find\_register(register\_name, display=False)

Find register information for provided search string. :param register\_name: Regular expression to search against :param display: True to output result to console :return: List of found registers

## find\_register\_names(register\_name, display=False)

Find register names for the provided search string.

## **Parameters**

- **register\_name** (*str*) Regular expression to search against.
- **display** (*bool*, *optional*) True to output result to console. Defaults to False.

```
Returns
             List of found register names.
         Return type
             list
get_bios()
     Generate a BIOS string based on specific register values.
         Returns
             BIOS information string.
         Return type
             str
get_board(ext_info_offset=16)
     Get the board name from the extended info string.
         Parameters
             ext_info_offset (int, optional) – Offset for extended info. Defaults to 0x10.
         Returns
             The extracted board name.
         Return type
             str
         Raises
             NameError – If the field BOARD doesn't exist in the extended info.
get_board_info()
     Retrieve information about the board.
         Returns
             Board information as a dictionary.
         Return type
             dict
get_extended_info(ext_info_offset=16)
     Get the extended info string from the board.
get_mac()
     Retrieve the MAC address and format it as a string.
         Returns
             MAC address string.
         Return type
get_register_name_by_address(add)
     Get register name by address.
         Parameters
             add (int) – Register address.
```

Register name or "Unknown register address" if not found.

**Returns** 

Return type str

```
get_xml_from_board(xml_map_offset)
     Get XML data from the board.
list_register_names()
     List register names.
         Raises
             None -
load_firmware_blocking(Device, path_to_xml_file=", xml_map_offset=8)
     Load firmware blocking.
pll_calib()
     Calibrate PLL.
     This function performs calibration by writing specific values to SPI addresses.
         Raises
             None -
pll_dumpcfg(cfg filename)
     Dump PLL configuration to a file.
         Parameters
             cfg_filename (str) – The path to the configuration file.
pll_ioupdate()
     Update PLL IO.
     This function updates PLL IO by writing a specific value to the SPI address.
         Raises
             None -
pll_ldcfg(cfg_filename)
     Load PLL configuration from a file.
         Parameters
             cfg_filename (str) – The path to the configuration file.
pll_read_with_update(address)
     Read from SPI with PLL update.
         Parameters
             address (int) – The address to read from.
         Returns
             The read value.
         Return type
             int
pll_write_with_update(address, value)
     Write to SPI with PLL update.
         Parameters
             • address (int) – The address to write to.
```

• **value** (*int*) – The value to write.

```
read_register(register, n=1, offset=0, device=None)
```

Get register value :param register: Register name :param n: Number of words to read :param offset: Memory address offset to read from :param device: Device/node can be explicitly specified :return: Values

### read\_spi(address)

Read SPI data from the specified address.

#### **Parameters**

**address** (*int*) – The address to read from.

#### Returns

The read value.

#### Return type

int

## write\_register(register, values, offset=0, device=None)

Set register value :param register: Register name :param values: Values to write :param offset: Memory address offset to write to :param device: Device/node can be explicitly specified

## write\_spi(address, value)

Write SPI data to the specified address.

#### **Parameters**

- address (int) The address to write to.
- **value** (*int*) The value to write.

```
management.filter_list_by_level(reg_name_list, reg_name)
```

Filter a list of register names by level.

```
management.format_num(num)
```

Convert a number to a string.

```
management.get_max_width(table1, index1)
```

Get the maximum width of the given column index

```
management.get_shift_from_mask(mask)
```

Get the shift value from a mask.

## management.hexstring2ascii(hexstring, xor=0)

Convert a hexstring to an ASCII-String.

## **Parameters**

- **hexstring** (*str*) The input hex string.
- **xor** (*int*) XOR value as an integer (default is 0).

## Returns

The converted ASCII string.

## **Return type**

str

## management.pprint\_table(table)

Prints out a table of data, padded for alignment.

## **Parameters**

**table** (*list*) – The table to print. A list of lists. Each row must have the same number of columns.

### subrack\_management\_board.Adu\_Eth\_Ping(ip, count=1, interval='0.2', size=8, wait='1')

Perform a ping on a specified IP address.

## exception subrack\_management\_board.SubrackExecFault

Define an exception which occurs when an error occur when a function or class method fails

## exception subrack\_management\_board.SubrackInvalidCmd

Define an exception which occurs when an invalid command is provided to a function or class method

## exception subrack\_management\_board.SubrackInvalidParameter

Define an exception which occurs when an invalid parameter is provided to a function or class method

## class subrack\_management\_board.SubrackMngBoard(\*\*kwargs)

This class implements methods to manage and to monitor the subrack management board

## GetCPLDLockedPLL()

This method get the status of the CPLD internal PLL Lock :return locked: value of locked status, True PLL is locked. False PLL not locked

#### GetFanAlarm()

method to get Fan Status Alarm Register of subrack :return alarms: OK, WARN, ALARM, WARN-ALARM, of each Fan

## GetFanMode(fan id)

This method get the\_bkpln\_fan\_mode :param fan\_id: id of the selected fan accepted value: 1-4 :return auto\_mode: functional fan mode: auto or manual :return status: status of operation

## GetFanPwm(fan\_id)

Retrieves the PWM (Pulse Width Modulation) percentage of a specified fan.

This method queries the fan speed and returns the PWM percentage.

#### **Parameters**

**fan\_id** – The fan identifier.

## Returns

The PWM (Pulse Width Modulation) percentage of the specified fan.

#### GetFanRpm(fan id)

Retrieves the RPM (Revolutions Per Minute) of a specified fan.

This method queries the fan speed and returns the RPM value.

#### **Parameters**

**fan\_id** – The fan identifier.

#### Returns

The RPM (Revolutions Per Minute) of the specified fan.

## GetFanSpeed(fan\_id)

This method get the get\_bkpln\_fan\_speed :param fan\_id: id of the selected fan accepted value: 1-4 :return fanrpm: fan rpm value :return fan\_bank\_pwm: pwm value of selected fan

#### GetLockedPLL()

This method get the status of the PLL Lock :return locked: value of locked status, True PLL is locked, False PLL not locked

## GetPSFanSpeed(ps\_id)

This method get the fan speed of selected Power Supply of subrack :param ps\_id: id of the selected power supply, accepted value: 1,2 :return fanspeed: speed of the fan

### **GetPSIout**(*ps\_id*)

This method get the Iout current value of selected Power Supply of subrack :param ps\_id: id of the selected power supply, accepted value: 1,2 :return vout: value of Iout in Ampere

### GetPSPower(ps\_id)

This method get the Power consumption value of selected Power Supply of subrack :param ps\_id: id of the selected power supply, accepted value: 1,2 :return power: value of power in W

### GetPSVout(ps\_id)

This method get the Vout voltage value of selected Power Supply of subrack :param ps\_id: id of the selected power supply, accepted value: 1,2 :return vout: value of Vout in Volt

## GetPingCpld()

Checks the connectivity to the CPLD using a ping operation.

This method checks if the CPLD is reachable via ping.

## Returns

True if the CPLD is reachable via ping, False otherwise.

### GetPingTPM(tpm slot id)

Checks the connectivity to a TPM using a ping operation.

This method checks if the TPM in the specified slot is present, powered on, and responds to a ping operation.

#### **Parameters**

**tpm\_slot\_id** – The TPM slot identifier.

#### Returns

True if the TPM is reachable via ping, False if unreachable, None if TPM is not present or not powered on.

#### GetPllSource()

Retrieves the PLL (Phase-Locked Loop) source.

This method reads the PLL source from the management interface.

#### Returns

"internal" if the PLL source is internal, "external" otherwise.

#### GetPowerAlarm()

method to get TPM Power consumption Alarm Register of subrack :return alarms: status vector, OK, WARN, ALM of each TPM Voltages Alarm, for each board

## GetSubrackTemperatures()

method to get temperatures from sensors placed on backplane and subrack-management boards :return temp\_mng1: temperature value of management sensor 1 :return temp\_mng2: temperature value of management sensor 2 :return temp\_bck1: temperature value of backplane sensor 1 :return temp\_bck2: temperature value of backplane sensor 2

## GetTPMCurrent(tpm\_slot\_id, force=True)

method to get current consuptin of selected tpm (providing subrack index slot of tpm) :param tpm\_slot\_id: subrack slot index for selected TPM, accepted value 1-8 :param force: force the operation even if no TPM is present in selected slot

## ${\tt GetTPMGlobalStatusAlarm}(tpm\_slot\_id, forceread = False)$

Deprecated method to retrieve TPM global status alarms.

#### **Parameters**

• tpm\_slot\_id - The TPM slot identifier.

• **forceread** – (Optional) If True, forces a read even if deprecated.

#### Raises

**SubrackExecFault** – Indicates that the method is deprecated, and subrack no longer accesses TPM.

## GetTPMIP(tpm\_slot\_id)

method to manually set volatile local ip address of a TPM board present on subrack :param tpm\_slot\_id:subrack slot index for selected TPM, accepted value 1-8 :return tpm\_ip\_str: tpm ip address

## GetTPMInfo(tpm\_slot\_id, forceread=False)

Deprecated method to retrieve TPM information.

#### **Parameters**

- tpm\_slot\_id The TPM slot identifier.
- **forceread** (Optional) If True, forces a read even if deprecated.

#### Raises

**SubrackExecFault** – Indicates that the method is deprecated, and subrack no longer accesses TPM.

## GetTPMMCUTemperature(tpm\_slot\_id, forceread=False)

Deprecated method to retrieve TPM MCU temperature.

#### **Parameters**

- tpm\_slot\_id The TPM slot identifier.
- **forceread** (Optional) If True, forces a read even if deprecated.

## Raises

**SubrackExecFault** – Indicates that the method is deprecated, and subrack no longer accesses TPM.

#### GetTPMOnOffVect()

method to get Power On status of inserted tpm, 0 off or not present, 1 power on :return vector of poweron status for slots, bits 7:0, bit 0 slot 1, bit 7 slot 8, 1 TPM power on, 0 no TPM inserted or power off

#### GetTPMPower(tpm\_slot\_id, force=True)

method to get power consumption of selected tpm (providing subrack index slot of tpm) :param tpm\_slot\_id: subrack slot index for selected TPM, accepted value 1-8 :param force force the operation even if no TPM is present in selected slot

## GetTPMPresent(tpm\_slot\_id=None)

brief method to get info about TPM board present on subrack :return TpmDetected: vector of tpm positional,1 TPM detected,0 no TPM inserted,bit 7:0,bit 0 slot 1,bit 7 slot 8

## GetTPMSupplyFault()

Method to get info about TPM supply fault status, 1 for each TPM in backplane slot :return tpmsupplyfault: vector of tpm supply fault status, 1 fault, 0 no fault,bit 7:0,bit 0 slot 1,bit 7 slot 8

## GetTPMTemperatures(tpm\_slot\_id, forceread=False)

Deprecated method to retrieve TPM temperatures.

## **Parameters**

- tpm\_slot\_id The TPM slot identifier.
- **forceread** (Optional) If True, forces a read even if deprecated.

#### Raises

**SubrackExecFault** – Indicates that the method is deprecated, and subrack no longer accesses TPM.

## GetTPMVoltage(tpm\_slot\_id, force=True)

brief method to get power consuptin of selected tpm (providing subrack index slot of tpm) :param tpm\_slot\_id:subrack slot index for selected TPM, accepted value 1-8 :param force: force the operation even if no TPM is present in selected slot

#### GetTPM\_Add\_List()

method to get the IP address will be assigned to each TPM board present on subrack :return list of IP address will be assigned assigned

#### GetUPSStatus()

Retrieves the status of the uninterruptible power supply (UPS).

This method reads data from the serial port to update UPS charge registers and ADC (Analog-to-Digital Converter) values. It also checks for warning and alarm conditions based on voltage levels.

#### Returns

A dictionary containing the UPS status information, including: - 'warning': True if there is a warning condition, False otherwise. - 'alarm': True if there is an alarm condition, False otherwise. - 'charging': True if the UPS is currently charging, False otherwise.

## GetVoltageAlarm()

method to get TPM Voltages Power supply Alarm Register of subrack :return alarms: status vector, OK, WARN, ALM of each TPM Voltages Alarm, for each board

#### Get\_API\_version()

method to get the Version of the API :return string with API version

## Get\_Subrack\_TimeTS()

method to get the subrack Time in timestamp format :return time in timestamp format

#### Get\_TPM\_temperature\_vector()

Deprecated method to retrieve TPM temperature vector.

#### Raises

**SubrackExecFault** – Indicates that the method is deprecated, and subrack no longer accesses TPM.

## Get\_tpm\_alarms\_vector()

Deprecated method to retrieve TPM alarms vector.

## Raises

**SubrackExecFault** – Indicates that the method is deprecated, and subrack no longer accesses TPM.

## Initialize(pll\_source\_internal=False)

Initialize the Subrack.

Parameters: - pll\_source\_internal (bool): Set to True to use internal PLL source.

## **PllInitialize**(source\_internal=False, pll\_cfg\_file=None)

This method initialize the PLL

## PowerOffTPM(tpm\_slot\_id, force=False)

method to power off selected tpm:param tpm\_slot\_id: subrack slot index for selected TPM, accepted value 1-8:param force: force the operation even if no TPM is present in selected slot

## **SetFanMode**(fan id blk, auto mode)

This method set the fan mode :param fan\_id\_blk: id of the fan cuople accepted value: 1-4, for fan 1,2; 3 for fan 3,4 :param auto\_mode: fan mode configuration, 1 auto(controlled by MCU), 0 manual(use SetFanSpeed method) :note fan are coupled, passing fan\_blk\_id=1 both fan 1 and 2 will be configured at same mode,

## SetFanSpeed(fan\_id, speed\_pwm\_perc)

This method set the bkpln\_fan\_speed :param fan\_id: id of the selected fan accepted value: 1-4 :param speed\_pwm\_perc: percentage value of fan RPM (MIN 0=0% - MAX 100=100%) :note settings of fan speed is possible only if fan mode is manual

## SetPSFanSpeed(ps\_id, speed\_percent)

This method set the fan speed of selected Power Supply of subrack :param ps\_id: id of the selected power supply, accepted value: 1,2 :param speed\_percent: speed in percentual value from 0 to 100

## **SetTPMIP**(tpm\_slot\_id, ip, netmask, gateway=None, bypass\_check=False, timeout=100)

method to manually set volatile local ip address of a TPM board present on subrack :param tpm\_slot\_id: subrack slot index for selected TPM, accepted value 1-8 :param ip: ip address will be assigned to selected TPM :param netmask: netmask value will be assigned to selected TPM :return status

## SetUPSVoltageAlarmThresholds(alarm\_level)

Sets the UPS voltage alarm thresholds.

This method sets the alarm level for UPS voltage, considering the current UPS presence status.

#### **Parameters**

alarm\_level - The alarm level for UPS voltage.

#### Returns

The number of errors encountered during the operation.

## SetUPSVoltageWarningThresholds(warning\_level)

Sets the UPS voltage warning thresholds.

This method sets the warning level for UPS voltage, considering the current UPS presence status.

#### **Parameters**

warning\_level - The warning level for UPS voltage.

#### Returns

The number of errors encountered during the operation.

## SubrackInitialConfiguration()

@brief method Initizlize the Subrack power control configuration for TPM current limit

#### all\_monitoring\_categories()

Returns a list of all monitoring point 'categories'. Here categories is a super-set of monitoring points and is the full list of accepted strings to set\_monitoring\_point\_attr.

#### Returns

list of categories

## Return type

list of strings

#### all\_monitoring\_points()

Returns a list of all monitoring points by finding all leaf nodes in the lookup dict that have a corresponding method field.

The monitoring points returned are strings produced from '.' delimited keys.

#### Returns

list of monitoring points

## Return type

list of strings

## bkpln\_get\_field(key)

Retrieves the value of a field from the backup location (Bkpln).

This method delegates the task to the Bkpln instance to get the value of the specified field.

#### **Parameters**

**key** – The key identifying the field to be retrieved.

#### **Returns**

The value of the specified field.

## bkpln\_set\_field(key, value, override\_protected=False)

Sets a field in the backup location (Bkpln).

This method delegates the task to the Bkpln instance to set the specified field with the given value. Optionally, it allows overriding protected fields.

#### **Parameters**

- **key** The key identifying the field to be set.
- **value** The value to be set for the specified field.
- override\_protected (Optional) If True, allows overriding protected fields.

#### Returns

The result of setting the field.

## get\_board\_info()

Get information about the Subrack board.

Returns: dict: Board information.

#### get\_health\_dict(\*\*kwargs)

Returns the dictionary of SUBRACK monitoring points with the static key only, no value

## get\_health\_status(\*\*kwargs)

Returns the current value of SUBRACK monitoring points If no group argument given, current value of all monitoring points is returned.

For example: subrack.get\_health\_status(group='temperatures') would return only the health status for:

A group attribute is provided by default, see subrack\_monitoring\_point\_lookup.py. This can be used like the below example: subrack.get\_health\_status(group='temperatures') subrack.get\_health\_status(group='voltages')

## get\_health\_status\_w\_elapsed(\*\*kwargs)

Returns the current value of SUBRACK monitoring points If no group argument given, current value of all monitoring points is returned.

For example: subrack.get\_health\_status(group='temperatures') would return only the health status for:

A group attribute is provided by default, see subrack\_monitoring\_point\_lookup.py. This can be used like the below example: subrack.get\_health\_status(group='temperatures') subrack.get\_health\_status(group='slots') subrack.get\_health\_status(group='voltages')

## get\_subrack\_cpu\_cpld\_ip()

SubrackInitialConfiguration @brief method Initizlize the Subrack power control configuration for TPM current limit :return cpu\_ip, cpld\_ip: Management CPU IP, Management CPLD IP

## read\_tpm\_singlewire(tpm\_id, address)

Reads a value from the single-wire interface of a TPM.

This method selects a TPM by psnt\_mux and reads a register value from the specified address in the single-wire interface.

#### **Parameters**

- tpm\_id The TPM identifier.
- **address** The register address in the single-wire interface.

#### Returns

The value read from the specified register.

## write\_tpm\_singlewire(tpm\_id, address, value)

Writes a value to the single-wire interface of a TPM.

This method selects a TPM by psnt\_mux and writes a value to the specified address in the single-wire interface.

#### **Parameters**

- **tpm\_id** The TPM identifier.
- **address** The register address in the single-wire interface.
- **value** The value to be written to the register.

```
subrack_management_board.detect_ip(tpm_slot_id)
```

Detect the IP address of a TPM board based on its slot ID.

```
subrack_management_board.dt_to_timestamp(d)
```

Convert a datetime object to a Unix timestamp.

```
subrack_management_board.exec_cmd(cmd, dir=None, verbose=True, exclude_line=")
```

Execute a shell command and capture its output.

```
subrack_management_board.flatten_dict(d, parent_key=", sep='_')
```

Flatten a nested dictionary.

```
subrack_management_board.int2ip(value)
```

Convert an integer value to an IP address.

```
subrack_management_board.ipstr2hex(ip)
```

Convert an IP address string to a hexadecimal representation.

```
subrack_management\_board.reduce(function, iterable[, initial]) \rightarrow value
```

Apply a function of two arguments cumulatively to the items of a sequence or iterable, from left to right, so as to reduce the iterable to a single value. For example, reduce(lambda x, y: x+y, [1, 2, 3, 4, 5]) calculates ((((1+2)+3)+4)+5). If initial is present, it is placed before the items of the iterable in the calculation, and serves as a default when the iterable is empty.

```
class subrack_monitoring_point_lookup.partial(func,/, *args, **keywords)
```

New function with partial application of the given arguments and keywords.

## WEB SERVER DOCUMENTATION

Package to implement simple device drivers. Each hardware driver is an instance of a HardwareBaseClass It contains a set of HardwareCommand objects, which implement actions (with optional parameters) and Hardware Attribute objects, which implement physical quantities. The HardwareBaseClass is controlled using commands, or by setting attributes (if they are RW). It is monitored by reading attributes. Hardware objects, commands and attributes can be subclassed for specific behaviors.

class HardwareBaseClass.GetAllAttributesCommand(name, hardware=None, num\_params=0)

command which returns a dictionary of all attributes and their current values in the 'value' field

do(params=None)

## **Parameters**

params (list) – Optional list of parameters

#### Returns

Dictionary of returned response

## Return type

dict

Attribute for a HardwareBaseClass Attributes can be scalar or vector of fixed dimension, of arbitrary types Can be read/write or read only

## name()

Returns the command name

#### Returns

Command name

## Return type

str

## read()

Read the attribute, formatting the response dictionary

#### Returns

Dictionary of returned response

## Return type

dict

## read\_value()

Reads the actual value To be overrided in the subclass

#### Returns

Attribute true value, from real device

## write(params)

write value(s) in params to the attribute

#### **Parameters**

**params** – Parameters to be written

#### **Returns**

dictionary for json answer

## Return type

dict

## write\_value(values)

Writes the actual value To be overrided in the subclass

#### **Parameters**

value - Attribute true value, to real device

## class HardwareBaseClass.HardwareBaseDevice

Server side of the hardware device.

#### add\_attribute(attribute)

Add an attribute to the command list

#### **Parameters**

attribute (HardwareBase.HardwareAttribute) - Attribute object

## Returns

True if command canbe added, False otherwise

## Return type

Bool

## add\_command(command)

Add a command to the command list

## **Parameters**

command - Command object

## Returns

True if command canbe added, False otherwise

## **Return type**

**Bool** 

## execute\_command(command, params=None)

Execute a command with optional parameters

#### **Parameters**

- **command** (str) Command name
- params Optional parameter, simple list or scalar

## Returns

dictionary for json answer

```
get_attribute(attribute)
          Get attribute values
              Parameters
                  attribute - Attribute name
              Returns
                  dictionary for json answer
     set_attribute(attribute, values)
          Set attribute values
              Parameters
                   • attribute - Attribute name
                   • values – Attribute values, simple list or scalar
              Returns
                  dictionary for json answer
class HardwareBaseClass.HardwareCommand(name, hardware=None, num params=0)
     Command for a Hardware base class. Command has a name, has access to its base hardware (if useful), and can
     check the number of parameters
     do(params=None)
          Execution method. Subclasses must override this to do real work
              Parameters
                  params (list) – Optional list of parameters
                  Dictionary of returned response
              Return type
                  dict
     name()
          Returns the command name
              Returns
                  Command name
              Return type
                  str
class HardwareBaseClass.ListAttributeCommand(name, hardware=None, num_params=0)
     Command to list names of the Hardware class commands
     do(params=None)
              Parameters
                  params (list) - Optional list of parameters
              Returns
                  Dictionary of returned response
              Return type
class HardwareBaseClass.ListCommandCommand(name, hardware=None, num params=0)
     Command to list names of the Hardware class attributes
```

do(params=None)

#### **Parameters**

**params** (list) – Optional list of parameters

#### Returns

Dictionary of returned response

## Return type

dict

## class HardwareThreadedClass.AbortCommand(name, hardware=None, num\_params=0)

Abort a thread (if specfied in the parameter) or the blocking thread

do(param=None)

#### **Param**

Command to abort or None

### **Type**

class.ThreadedHardwareCommand

## class HardwareThreadedClass.IsCompletedCommand(name, hardware=None, num\_params=0)

check if a specific command thread (if specified in the parameter) or the blocking thread is currently running

do(param=")

#### **Param**

Command to check or None

## **Type**

class.ThreadedHardwareCommand

## 

An hardware command which requires long time to complete. A thread is started to execute the command. The thread must contain frequent checkpoints, where the \_abort flag is checked and the action stopped if this is set. The completed() method checks for completion of the thread.

#### abort()

abort the current thread. Just sets the abort flag, and waits for the thread to complete

## completed()

checks whether the current thread has completed If it has, joins the thread (to free resources) :return: Whether the thread has completed :rtype: Bool

## **do**(params)

Command execution. Starts a thread and sets internal atributes to :param params: Command parameters :return: Dictionary of returned response :rtype: dict

## is\_blocking()

#### Returns

Whether the command requires blocking of the device

## Return type

Bool

## thread(params)

Execution thread. Must periodically check self.\_abort with maximum execution time of approx 1 second between checkpoints Terminates setting self.\_completed True.

```
LFAA SPS Subrack control board hardware driver.
class subrack_hardware.API_Version(name, init_value, hardware=None, read_write=0, num_params=1)
     API version Returns version of API
     read_value()
          Reads the actual value To be overrided in the subclass
              Returns
                  Attribute true value, from real device
class subrack_hardware.AreTpmsOnCommand(name, hardware=None, num_params=0)
     do(params)
          Execution method. Subclasses must override this to do real work
              Parameters
                  params (list) – Optional list of parameters
              Returns
                  Dictionary of returned response
              Return type
                  dict
class subrack_hardware.BackplaneTemperature(name, init_value, hardware=None, read_write=0,
                                                    num \ params=1)
     Backplane temperature, in celsius
     read_value()
              Returns
                  backplane temperature, in celsius, for the two backplane halves
              Return type
                  list[float]
class subrack_hardware.BoardCurrent(name, init_value, hardware=None, read_write=0, num_params=1)
     read_value()
              Returns
                  Total subrack current (A)
              Return type
class subrack_hardware.BoardTemperature(name, init_value, hardware=None, read_write=0,
                                               num_params=1)
     read_value()
              Returns
                  Subrack control board temperature, in celsius, 2 values
              Return type
                  list[float]
class subrack_hardware.CPLD_PLL_Locked(name, init_value, hardware=None, read_write=0,
                                              num \ params=1)
     Subrack CPLD PLL Lock status Returns status of CPLD internal PLL lock
```

```
read_value()
          Reads the actual value To be overrided in the subclass
               Returns
                  Attribute true value, from real device
class subrack_hardware.FanMode(name, init_value, hardware=None, read_write=0, num_params=1)
     read_value()
               Returns
                   Subrack fan mode
               Return type
                   list[float]
     write_value(mode)
          Writes the actual value To be overrided in the subclass
               Parameters
                  value - Attribute true value, to real device
class subrack_hardware.FanSpeed(name, init_value, hardware=None, read_write=0, num_params=1)
     read_value()
               Returns
                   Subrack fan speed, in RPM, for the 4 fans
               Return type
                   list[float]
class subrack_hardware.FanSpeedPercent(name, init_value, hardware=None, read_write=0,
                                               num_params=1)
     read_value()
               Returns
                   Subrack fan speed, in percent of the maximum values, for the 4 fans
               Return type
                   list[float]
class subrack_hardware.GetHealthDict(name, hardware=None, num_params=0)
     Return subrack health status dictionary.
     do(params)
          Info about subrack health status :param params: group of monitor points to report :type params: str
          :return:dictionary of monitor points :rtype: dict
class subrack_hardware.GetHealthStatus(name, hardware=None, num_params=0)
     Return info about subrack health status.
     do(params)
          Info about subrack health status :param params: group of monitor points to report :type params: str
                   dictionary of monitor point values
               Return type
                   dict
```

## **class** subrack\_hardware.**IsTpmOnCommand**(name, hardware=None, num\_params=0) Check TPM power status do(params) Check power status of TPMs **Parameters params** (str) – index of TPM to check (1-8)Returns dictionary with HardwareCommand response. Integer retvalue Return type dict **class** subrack\_hardware.**PSCurrent**(name, init\_value, hardware=None, read\_write=0, num\_params=1) Hardware attribute class for reading power supply currents. Reads current values for power supplies 1 and 2 and returns a list. Attributes: - \_hardware: Reference to the SubrackMngBoard hardware object. Methods: - read value(): Reads power supply currents and returns a list. read\_value() Reads power supply currents. Returns: list: List of current values for power supplies 1 and 2. class subrack\_hardware.PSFanSpeed(name, init\_value, hardware=None, read\_write=0, num\_params=1) Hardware attribute class for reading power supply fan speeds. Reads fan speeds for power supplies 1 and 2 and returns a list of values. Attributes: - \_hardware: Reference to the SubrackMngBoard hardware object. Methods: - read\_value(): Reads power supply fan speeds and returns a list. read\_value() Reads power supply fan speeds. Returns: list: List of fan speeds for power supplies 1 and 2. class subrack\_hardware.PSPower(name, init\_value, hardware=None, read\_write=0, num\_params=1) Hardware attribute class for reading power supply powers. Reads power values for power supplies 1 and 2 and returns a list. Attributes: - \_hardware: Reference to the SubrackMngBoard hardware object. Methods: - read\_value(): Reads power supply powers and returns a list. read\_value() Reads power supply powers. Returns: list: List of power values for power supplies 1 and 2. **class** subrack\_hardware.**PSVoltage**(name, init\_value, hardware=None, read\_write=0, num\_params=1) Hardware attribute class for reading power supply voltages. Reads voltage values for power supplies 1 and 2 and returns a list. Attributes: - \_hardware: Reference to the SubrackMngBoard hardware object.

Methods: - read\_value(): Reads power supply voltages and returns a list.

```
read_value()
          Reads power supply voltages.
          Returns: list: List of voltage values for power supplies 1 and 2.
class subrack_hardware.PowerDownCommand(name, hardware=None, num_params=0, blocking=False)
     Power off all TPMs
     thread(params)
          Power off all TPMs
              Parameters
                  params - unused
class subrack_hardware.PowerOffTpmCommand(name, hardware=None, num_params=0, blocking=False)
     Power Off TPM command. Switches off a single or multiple TPM
     thread(tpm_id)
          Power off TPMs
              Parameters
                  tpm\_id(str, list(str)) – index of TPM to power on (1-8) or list of indexes
              Returns
                  dictionary with HardwareCommand response. List of TPM On status
              Return type
                  dict
class subrack_hardware.PowerOnTpmCommand(name, hardware=None, num params=0, blocking=False)
     Power On TPM command. Switches on a single or multiple TPM
     thread(tpm_id)
          Power on TPMs
              Parameters
                  tpm_id (str, list(str)) – index of TPM to power on (1-8) or list of indexes
              Returns
                  dictionary with HardwareCommand response. List of TPM On status
              Return type
                  dict
class subrack_hardware.PowerUpCommand(name, hardware=None, num params=0, blocking=False)
     Power on all TPMs
     thread(params)
          Power on all TPMs
              Parameters
                  params - unused
class subrack_hardware.SetFanMode(name, hardware=None, num_params=0)
     Set fan mode (manual, auto)
     do(params)
              Parameters
                  params – [0]: Fan ID (in range 1-4), [1]: mode [MANUAL|AUTO]
```

### **Returns**

dictionary with HardwareCommand response.

## Return type

dict

## class subrack\_hardware.SetFanSpeed(name, hardware=None, num params=0)

Set cabinet fan speed (0-100)

do(params)

#### **Parameters**

params – [0]: Fan ID (in range 1-4), [1]: speed (percentage)

#### Returns

dictionary with HardwareCommand response. Retval is actual speed

## Return type

dict

## **class** subrack\_hardware.**SetPSFanSpeed**(name, hardware=None, num params=0)

Set Power Supply fan speed (0-100)

do(params)

#### **Parameters**

**params** – [0]: Fan ID (in range 1-2), [1]: speed (percentage)

#### Returns

dictionary with HardwareCommand response. Retval is actual speed

## Return type

dict

## class subrack\_hardware.SubrackHardware

Hardware device class representing the Subrack.

Initializes the SubrackMngBoard object and adds commands and attributes related to the Subrack.

Attributes: - subrack: Reference to the SubrackMngBoard hardware object.

Methods: - initialize(emulation=False): Initializes the SubrackMngBoard object. - execute\_command(command, params=None): Executes the specified command.

## execute\_command(command, params=None)

Executes the specified command.

Args: - command (str): The command to be executed. - params (dict): Optional parameters for the command.

Returns: dict: Dictionary containing the command execution result.

## initialize(emulation=False)

Initializes the SubrackMngBoard object.

 $Args: \hbox{ -emulation (bool): } Indicates \hbox{ whether the Subrack is in emulation mode.}$ 

Returns: None

## 

Subrack PLL Lock status Returns status of Subrack PLL lock

#### read\_value()

Reads the actual value To be overrided in the subclass

#### Returns

Attribute true value, from real device

Subrack Time in timestamp Returns Time of Subrack in timrstamp format

#### read\_value()

Reads the actual value To be overrided in the subclass

#### Returns

Attribute true value, from real device

**class** subrack\_hardware.**TPM\_Add\_List**(name, init\_value, hardware=None, read\_write=0, num\_params=1)

TPM IP Address Will Be Assigned. Returns 8 IP address, address will be assigned to each TPM in subrack slots

#### read\_value()

Reads the actual value To be overrided in the subclass

#### Returns

Attribute true value, from real device

TPMs Temperature alarm status Returns 8 temeprature alarms

#### read\_value()

Reads the actual value To be overrided in the subclass

## Returns

Attribute true value, from real device

TPMs Temperature vectors Returns 8 TPMBoard temperatures , 8 TPM FPGA1 temperatures, 8 TPM FPGA2 temperatures

## read\_value()

Reads the actual value To be overrided in the subclass

#### Returns

Attribute true value, from real device

TPMs Voltage alarm status Returns 8 voltage alarms

#### read\_value()

Reads the actual value To be overrided in the subclass

#### Returns

Attribute true value, from real device

class subrack\_hardware.TpmCurrents(name, init\_value, hardware=None, read\_write=0, num\_params=1)
TPM board current, in A. Returns 8 values, 0.0 for boards not present

```
read_value()
              Returns
                  8 values, 0.0 for boards not present
              Rvalue
                  list(float)
class subrack_hardware.TpmIPs(name, init_value, hardware=None, read_write=0, num_params=1)
     IP address of TPMs present on subrack. Returns 8 IP address, "0" for TPMs not powered off and "-1" for TPMs
     not present
     read_value()
              Returns
                  TPM IP
              Return type
                  list[str]
class subrack_hardware.TpmInfo(name, hardware=None, num_params=0)
     Return info about TPM board present on subrack.
     do(params)
          Info about TPM.
              Parameters
                  params (str) – index of TPM to check (1-8)
              Returns
                  TPM info
              Return type
                  dict
class subrack_hardware.TpmMCUTemperatures(name, init_value, hardware=None, read_write=0,
                                                 num\_params=1)
     TPM MCU temperatures, in celsius. Returns 8 values, 0.0 for boards not present or powered off
     read_value()
                  TPM MCU temperature, in Celsius
              Return type
                  list[float]
class subrack_hardware.TpmOnOffVect(name, init_value, hardware=None, read_write=0, num_params=1)
     TPM board power status
     read_value()
          Reads the actual value To be overrided in the subclass
              Returns
                  Attribute true value, from real device
class subrack_hardware.TpmPowers(name, init_value, hardware=None, read_write=0, num_params=1)
     TPM board power usage (W)
```

```
read_value()
               Returns
                   8 values, 0.0 for boards not present
               Rvalue
                   list(float)
class subrack_hardware.TpmPresent(name, init_value, hardware=None, read_write=0, num_params=1)
     TPM board presence Returns 8 values, True if board present and powered on
     read_value()
           Reads the actual value To be overrided in the subclass
                   Attribute true value, from real device
class subrack_hardware.TpmSupplyFault(name, init_value, hardware=None, read_write=0,
                                              num\_params=1)
     TPM supply fault status Returns 8 bool values, True if board supply fault has been triggered
     read_value()
           Reads the actual value To be overrided in the subclass
               Returns
                   Attribute true value, from real device
class subrack_hardware.TpmVoltages(name, init_value, hardware=None, read_write=0, num_params=1)
     TPM board power supply voltages (V)
     read_value()
                   8 values, 0.0 for boards not present
               Rvalue
                   list(float)
subrack_hardware.byte_to_bool_array(byte_in)
     Convert a byte to a list of 8 bool. Bit 0 corresponds to list[0]
           Parameters
               byte_in (int) – Byte with 8 LS bits representing 8 logical values
          Returns
               list of 8 bool values
class subrack_hardware.ups_status(name, init_value, hardware=None, read_write=0, num_params=1)
     UPS board status Returns UPS Board Status: ups_status = {"alarm":False,"warning":False,"charging":False}
     read value()
           Reads the actual value To be overrided in the subclass
               Returns
                   Attribute true value, from real device
class web_server.MyServer(request, client_address, server)
     Request handler, subclassed from http package
```

## do\_GET()

Callback for GET request Retrieves query, and splits it into a dictionary Use mangle\_dict to convert 1-lists to scalars, and numeric strings to numbers Calls appropriate methods of the hardware class. Formats the return dictionary to json and send it as response

## web\_server.mangle\_dict(input\_dict)

Takes a query dictionary from the http.getquerydict() FOr each element keeps only the first element (list of values) Converts it to scalar if it is a list of 1 element If some values are numeric convert them to numbers

## **CPLD MANAGEMENT API DOCUMENTATION**

```
config_ip.get_mac_from_eep(inst, phy_addr=160)
```

Read MAC address from EEPROM.

## **Parameters**

- **inst** (*instance*) Instance of a class with 'bsp' attribute.
- **phy\_addr** (*int*) Physical address in EEPROM (default is 0xA0).

#### Returns

List representing the MAC address.

## Return type

list

## config\_ip.int2ip(value)

Convert an integer to an IP address string.

#### **Parameters**

**value** (*int*) – Integer representation of the IP address.

## Returns

IP address string.

## **Return type**

str

## config\_ip.nuple2mac(mac)

Convert a MAC address tuple to a string.

## **Parameters**

**mac** (tuple) – Tuple representing the MAC address.

#### Returns

MAC address string.

## Return type

stı

## config\_ip.read\_string(inst, offset, max\_len=32)

Read a string from EEPROM.

#### **Parameters**

- inst (instance) Instance of a class with 'bsp' attribute.
- **offset** (*int*) Offset in EEPROM to start reading the string.
- max\_len (int) Maximum length of the string to read (default is 32).

#### Returns

Read string from EEPROM.

## Return type

str

## config\_ip.write\_string(inst, offset, string)

Write a string to EEPROM.

#### **Parameters**

- inst (instance) Instance of a class with 'bsp' attribute.
- **offset** (*int*) Offset in EEPROM to start writing the string.
- **string** (*str*) String to be written to EEPROM.

## class cpld2mcu\_serial\_ctrl\_2.FlashCmd

Class containing flash commands and related constants.

## cpld2mcu\_serial\_ctrl\_2.load\_bitstream(filename, pagesize)

Load a bitstream from a file.

## **Parameters**

- **filename** The name of the file to load.
- **pagesize** The size of the pages.

#### Returns

A tuple containing the loaded bitstream, bitstream size, and total size.

## mcu\_update.loadBitstream(filename, pagesize)

Load the bitstream from a file.

## **Parameters**

- **filename** (str) The path to the bitstream file.
- **pagesize** (*int*) The size of the pages.

#### Returns

The formatted bitstream. int: The bitstream size. int: The total size.

## Return type

bytearray

Test TPM script.

```
author = "Bubs"
```

```
phy_marvell_88X2222_init.cfg_10g(port=0, mdio_mux=3)
```

Configure a 10G port.

## **Parameters**

- **port** The port number (default is 0).
- **mdio\_mux** The MDIO multiplexer value (default is 3).

phy\_marvell\_88X2222\_init.decode\_register(port, reg\_def, reg\_value, field=None)

Decode and print the value of a register.

#### **Parameters**

• port – The port number.

- **reg\_def** The register definition.
- **reg\_value** The value to decode.
- **field** The specific field to decode (default is None).

## phy\_marvell\_88X2222\_init.get\_port\_cfg(port, mdio\_mux=2)

Get and decode the configuration of a port.

#### **Parameters**

- **port** The port number.
- mdio\_mux The MDIO multiplexer value (default is 2).

## phy\_marvell\_88X2222\_init.get\_switch\_status()

Get the status of various switch ports.

### **Returns**

Dictionary containing the status of each switch port.

```
phy_marvell_88X2222_init.rd(address)
```

Read a value from a given address.

#### **Parameters**

**address** – The address to read from.

#### **Returns**

The value read from the address.

phy\_marvell\_88X2222\_init.read22(mux, phy adr, register)

Read a value from a 22-bit register.

#### **Parameters**

- **mux** The multiplexer value.
- **phy\_adr** The physical address.
- **register** The register to read from.

## Returns

The value read from the register.

phy\_marvell\_88X2222\_init.read45(mux, phy\_adr, device, register)

Read a value from a 45-bit register.

#### **Parameters**

- **mux** The multiplexer value.
- **phy\_adr** The physical address.
- **device** The device.
- **register** The register to read from.

#### **Returns**

The value read from the register.

phy\_marvell\_88X2222\_init.read\_and\_decode(port, reg\_def, mdio\_mux=2, field=None)

Read and decode a register.

#### **Parameters**

• **port** – The port number.

- **reg\_def** The register definition.
- **mdio\_mux** The MDIO multiplexer value (default is 2).
- **field** The specific field to read (default is None).

## phy\_marvell\_88X2222\_init.read\_scratch(mux, offset)

Read a value from the scratch register.

#### **Parameters**

- **mux** The MDIO multiplexer value.
- **offset** The offset within the scratch register.

#### **Returns**

The value read from the scratch register.

```
phy_marvell_88X2222_init.read_wis(mdio_mux=3)
```

Read and print the WIS device identifier.

#### **Parameters**

**mdio\_mux** – The MDIO multiplexer value (default is 3).

phy\_marvell\_88X2222\_init.readmodifywrite(mux, phy\_adr, device, register, value, select)

Perform read-modify-write operation on a 45-bit register.

#### **Parameters**

- **mux** The multiplexer value.
- **phy\_adr** The physical address.
- **device** The device.
- **register** The register to read from.
- **value** The value to write.
- **select** The selection mask.

```
phy_marvell_88X2222_init.set_SFP(mdio_mux=2)
```

Configure the SFP module.

## **Parameters**

**mdio\_mux** – The MDIO multiplexer value (default is 2).

 $\verb"phy_marvell_88X2222_init.set_field" (port, reg\_def, field\_name, field\_value)"$ 

Set a field in a register.

## **Parameters**

- **port** The port number.
- **reg\_def** The register definition.
- **field\_name** The name of the field to set.
- **field\_value** The value to set in the field.

phy\_marvell\_88X2222\_init.wr(address, value)

Write a value to a given address.

#### **Parameters**

• **address** – The address to write to.

• **value** – The value to write.

phy\_marvell\_88X2222\_init.write22(mux, phy\_adr, register, value)

Write a value to a 22-bit register.

#### **Parameters**

- **mux** The multiplexer value.
- **phy\_adr** The physical address.
- **register** The register to write to.
- value The value to write.

phy\_marvell\_88X2222\_init.write22\_reg(port, reg\_def, reg\_value)

Write a value to a 22-bit register.

#### **Parameters**

- port The port number.
- **reg\_def** The register definition.
- reg\_value The value to write.

phy\_marvell\_88X2222\_init.write45(mux, phy\_adr, device, register, value)

Write a value to a 45-bit register.

#### **Parameters**

- **mux** The multiplexer value.
- **phy\_adr** The physical address.
- **device** The device.
- **register** The register to write to.
- value The value to write.

phy\_marvell\_88X2222\_init.write\_scratch(mux, offset, value)

Write a value to the scratch register.

## **Parameters**

- **mux** The MDIO multiplexer value.
- **offset** The offset within the scratch register.
- value The value to write.

## reg.get\_max\_width(table1, index1)

Get the maximum width of the given column index

## req.pprint\_table(table)

Prints out a table of data. @param table: The table to print. A list of lists. Each row must have the same number of columns.

@package rmp UDP socket management and RMP packet encoding/decoding

This package provides functions for network initializing and basic 32-bit read/write operations on the network-attached device using RMP protocol. This is rough and minimal code not exploiting all the RMP protocol features.

## class management\_flash.MngProgFlash(board, rmp)

Management Class for CPLD and FPGA SPI Flash bitfile storage/access class.

Attributes: - rmp: Pointer to RMP instance. - board: Pointer to board instance. - add4bytemode (bool): Flag indicating whether 4-byte addressing mode is enabled.

## **DeviceErase**(flashdeviceindedx, address, size)

Erase a specified range on the flash device.

#### **Parameters**

- **self** The object instance.
- **flashdeviceindedx** (*int*) Index of the flash device.
- address (int) Address in the flash to start erasing.
- **size** (*int*) Size of the range to erase.

## DeviceEraseChip(flashdeviceindedx)

Erase the entire flash chip.

#### **Parameters**

- **self** The object instance.
- **flashdeviceindedx** (*int*) Index of the flash device.

## DeviceGetID(flashdeviceindedx)

Get the identification of the flash device.

#### **Parameters**

- **self** The object instance.
- **flashdeviceindedx** (*int*) Index of the flash device.

#### Returns

Identification of the flash device.

## Return type

int

## DeviceGetInfo(flashdeviceindedx)

Get information about the flash device.

#### **Parameters**

- **self** The object instance.
- **flashdeviceindedx** (*int*) Index of the flash device.

## Returns

Information about the flash device.

## Return type

FlashDevice

## DeviceWrite(flashdeviceindedx, address, txbuff, size)

Write data to the flash device.

## **Parameters**

- **self** The object instance.
- **flashdeviceindedx** (*int*) Index of the flash device.

- **address** (*int*) Address in the flash where to start writing.
- **txbuff** (*bytes*) Data to be written.
- **size** (*int*) Size of the data to write.

### FlashDevice\_Enter4byteAddMode(device)

Enter 4-byte addressing mode for Flash device.

#### **Parameters**

**device** – Flash device.

### FlashDevice\_Exit4byteAddMode(device)

Exit 4-byte addressing mode for Flash device.

#### **Parameters**

**device** – Flash device.

### FlashDevice\_chiperase(device)

Erase the entire Flash chip.

#### **Parameters**

**device** – Flash device.

### FlashDevice\_erase(device, address, size)

Erase a range of memory in the Flash device.

#### **Parameters**

- **device** Flash device.
- **address** Starting address of the memory range to erase.
- **size** Size of the memory range to erase.

### FlashDevice\_eraseSector(device, address)

Erase a sector in the Flash device.

### **Parameters**

- **device** Flash device.
- address Memory address of the sector to erase.

### FlashDevice\_prepareCommand(command, address, device)

Prepare a Flash device command.

### **Parameters**

- **command** Command code.
- address Flash memory address.
- **device** Flash device.

#### Returns

Prepared command buffer.

### FlashDevice\_readIdentification(device)

Read identification from the Flash device.

### **Parameters**

**device** – Flash device.

#### **Returns**

Identification value.

### FlashDevice\_readPage(device, address, size)

Read a page from the Flash device.

#### **Parameters**

- **device** Flash device.
- address Memory address to read from.
- **size** Size of the page to read.

#### **Returns**

Read buffer.

### FlashDevice\_readReg(device, reg)

Read from Flash device register.

### **Parameters**

- **device** Flash device.
- **reg** Register to read.

#### **Returns**

Register value.

### FlashDevice\_readsector(device, address)

Read a sector from the Flash device.

#### **Parameters**

- **device** Flash device.
- address Memory address to read from.

#### **Returns**

Read buffer.

### FlashDevice\_waitTillReady(device)

Wait until the Flash device is ready.

### **Parameters**

**device** – Flash device.

### FlashDevice\_writeDisable(device)

Disable write for the Flash device.

### **Parameters**

**device** – Flash device.

### FlashDevice\_writeEnable(device)

Enable write for the Flash device.

### **Parameters**

**device** – Flash device.

### FlashDevice\_writePage(device, address, size, buffer)

Write a page to the Flash device.

#### **Parameters**

• **device** – Flash device.

- address Memory address to write to.
- **size** Size of the data to write.
- **buffer** Data buffer to write.

### **Returns**

Read buffer.

### FlashDevice\_writeReg(device, reg, value=None)

Write to Flash device register.

#### **Parameters**

- **device** Flash device.
- **reg** Register to write.
- **value** Value to write.

### FlashDevice\_writesector(device, address, buffer)

Write a sector to the Flash device.

#### **Parameters**

- **device** Flash device.
- address Memory address to write to.
- **buffer** Data buffer to write.

### **SPITransaction**(*device*, *TxBuffer*, *cmd*, *size*)

Perform an SPI transaction.

### **Parameters**

- **device** Flash device.
- TxBuffer Transmit buffer.
- cmd Command to be sent.
- **size** Size of the SPI transaction.

### Returns

Received buffer.

**firmwareProgram**(flashdeviceindedx, bitstreamFilename, address, dumpFilename=None, erase\_all=False, erase\_size=None, add\_len=False)

Program the firmware onto the flash device.

#### **Parameters**

- **self** The object instance.
- **flashdeviceindedx** (*int*) Index of the flash device.
- bitstreamFilename (str) Filename of the bitstream.
- address (int) Address in the flash where the bitstream will be written.
- dumpFilename (str, optional) Filename for dumping flash content. Defaults to None.
- **erase\_all** (*bool*, *optional*) Flag to indicate whether to erase the entire flash. Defaults to False.

- erase\_size (int, optional) Size to erase, if specified. Defaults to None.
- add\_len (bool, optional) Flag to prepend bitstream size during writing. Defaults to False.

### firmwareRead(flashdeviceindedx, address, size, dumpFilename)

Read firmware from the flash device.

#### **Parameters**

- **self** The object instance.
- **flashdeviceindedx** (*int*) Index of the flash device.
- address (int) Address in the flash from where to start reading.
- **size** (*int*) Size of the data to read.
- **dumpFilename** (*str*) Filename for dumping the read data.

### loadBitstream(filename, sectorSize)

Load a bitstream file into memory.

#### **Parameters**

- **filename** Path to the bitstream file.
- **sectorSize** Size of Flash sectors.

#### Returns

Tuple containing the loaded bitstream, bitstream size, and the size of the allocated memory block.

### saveBitstream(filename, memblock, bitstreamSize)

Save a bitstream from memory to a file.

#### **Parameters**

- **filename** Path to the output bitstream file.
- memblock Data to be saved.
- bitstreamSize Size of the bitstream data.

### spi\_chipselect(isactive)

Set or clear the SPI Chip Select.

#### **Parameters**

**isactive** – True to activate the Chip Select, False to deactivate.

### spi\_config(spi\_cs\_ow)

Configure SPI.

### **Parameters**

**spi\_cs\_ow** – 1 to enable, 0 to disable the SPI Chip Select One-Wire.

### spi\_mux\_selection(slaveid)

Select the SPI MUX.

### **Parameters**

slaveid - Slave ID for SPI MUX selection.

### spi\_rx\_available()

Get the number of available SPI receive bytes.

#### Returns

Number of available SPI receive bytes.

### spi\_sync(slaveid, txBuffer, cmd, length)

Perform a synchronous SPI transaction.

#### **Parameters**

- slaveid Slave ID for SPI communication.
- txBuffer Transmit buffer.
- cmd Command to be sent.
- **length** Length of SPI transaction.

#### **Returns**

Received buffer.

### spi\_trigger(length)

Trigger SPI transmission with the specified length.

#### **Parameters**

**length** – Length of SPI transmission.

### spi\_tx\_remaining()

Get the number of remaining SPI transmit bytes.

#### Returns

Number of remaining SPI transmit bytes.

### class management\_flash.spiregisters

Class representing SPI registers.

Attributes: - spi\_cs\_ow (int): SPI Chip Select One-Wire register address. - spi\_cs0 (int): SPI Chip Select 0 register address. - spi\_tx\_byte (int): SPI Transmit Byte register address. - spi\_rx\_byte (int): SPI Receive Byte register address. - spi\_tx\_buf\_len (int): SPI Transmit Buffer Length register address. - spi\_rx\_buf\_len (int): SPI Receive Buffer Length register address. - spi\_fio\_addr (int): SPI FIFO Address register address. - spi\_mux (int): SPI Multiplexer register address. - spi\_rxtxbuffer (int): SPI Receive/Transmit Buffer register address.

### class management\_mcu\_uart.MngMcuUart(board, rmp)

Management Class for CPLD2MCU Uart control for MCU update

#### reset\_mcu()

Resets the MCU.

### start\_mcu\_sam\_ba\_monitor()

Initiates the MCU SAM-BA monitor.

### Returns

The operation status (0 for success, 1 for timeout).

### uart\_receive\_byte()

Receives a single byte over UART from the MCU.

### Returns

A tuple containing the received byte and the operation status (0 for success, 1 for timeout).

### uart\_send\_buffer(databuff)

Sends a buffer of data over UART to the MCU.

### **Parameters**

**databuff** – The data buffer to be sent.

#### Returns

The operation status (0 for success, 1 for timeout).

### uart\_send\_buffer\_wrx(databuff)

Sends a buffer of data over UART to the MCU and receives a response.

#### **Parameters**

databuff - The data buffer to be sent.

#### Returns

A tuple containing the operation status (0 for success, 1 for timeout) and the received data buffer.

### uart\_send\_byte(dataw)

Sends a single byte over UART to the MCU.

#### **Parameters**

**dataw** – The byte of data to be sent.

#### Returns

The operation status (0 for success, 1 for timeout).

### class management.MANAGEMENT(\*\*kwargs)

Class representing a MANAGEMENT instance.

### checkLoad()

Check if the registers are loaded.

### **Raises**

**NameError** – If registers are not loaded or the board is not connected.

### disconnect()

Disconnect from the network.

This function closes the network connection and sets the state to "Unconnected".

### Raises

None -

### find\_register(register\_name, display=False)

Find register information for provided search string. :param register\_name: Regular expression to search against :param display: True to output result to console :return: List of found registers

### find\_register\_names(register\_name, display=False)

Find register names for the provided search string.

### **Parameters**

- $register\_name(str)$  Regular expression to search against.
- **display** (bool, optional) True to output result to console. Defaults to False.

#### Returns

List of found register names.

### Return type

list

## get\_bios() Generate a BIOS string based on specific register values. Returns BIOS information string. **Return type** str get\_board(ext\_info\_offset=16) Get the board name from the extended info string. **Parameters ext\_info\_offset** (*int*, *optional*) – Offset for extended info. Defaults to 0x10. **Returns** The extracted board name. Return type str Raises **NameError** – If the field BOARD doesn't exist in the extended info. get\_board\_info() Retrieve information about the board. Board information as a dictionary. Return type dict get\_extended\_info(ext\_info\_offset=16) Get the extended info string from the board. get\_mac() Retrieve the MAC address and format it as a string. Returns MAC address string. **Return type** str get\_register\_name\_by\_address(add)

Get register name by address.

### **Parameters**

**add** (*int*) – Register address.

Register name or "Unknown register address" if not found.

### Return type

### get\_xml\_from\_board(xml\_map\_offset)

Get XML data from the board.

```
list_register_names()
     List register names.
         Raises
             None -
load_firmware_blocking(Device, path_to_xml_file=", xml_map_offset=8)
     Load firmware blocking.
pll_calib()
     Calibrate PLL.
     This function performs calibration by writing specific values to SPI addresses.
             None -
pll_dumpcfg(cfg_filename)
     Dump PLL configuration to a file.
         Parameters
             cfg_filename (str) – The path to the configuration file.
pll_ioupdate()
     Update PLL IO.
     This function updates PLL IO by writing a specific value to the SPI address.
         Raises
             None -
pll_ldcfg(cfg_filename)
     Load PLL configuration from a file.
         Parameters
             cfg_filename (str) – The path to the configuration file.
pll_read_with_update(address)
     Read from SPI with PLL update.
         Parameters
             address (int) – The address to read from.
         Returns
             The read value.
         Return type
             int
pll_write_with_update(address, value)
     Write to SPI with PLL update.
         Parameters
             • address (int) – The address to write to.
```

• value (int) – The value to write.

```
read_register(register, n=1, offset=0, device=None)
```

Get register value :param register: Register name :param n: Number of words to read :param offset: Memory address offset to read from :param device: Device/node can be explicitly specified :return: Values

### read\_spi(address)

Read SPI data from the specified address.

### **Parameters**

**address** (*int*) – The address to read from.

#### Returns

The read value.

### Return type

int

### write\_register(register, values, offset=0, device=None)

Set register value :param register: Register name :param values: Values to write :param offset: Memory address offset to write to :param device: Device/node can be explicitly specified

### write\_spi(address, value)

Write SPI data to the specified address.

#### **Parameters**

- address (int) The address to write to.
- **value** (*int*) The value to write.

### management.filter\_list\_by\_level(reg\_name\_list, reg\_name)

Filter a list of register names by level.

### management.format\_num(num)

Convert a number to a string.

### management.get\_max\_width(table1, index1)

Get the maximum width of the given column index

### management.get\_shift\_from\_mask(mask)

Get the shift value from a mask.

### management.hexstring2ascii(hexstring, xor=0)

Convert a hexstring to an ASCII-String.

### **Parameters**

- **hexstring** (*str*) The input hex string.
- **xor** (*int*) XOR value as an integer (default is 0).

#### Returns

The converted ASCII string.

### **Return type**

str

### management.pprint\_table(table)

Prints out a table of data, padded for alignment.

#### **Parameters**

**table** (*list*) – The table to print. A list of lists. Each row must have the same number of columns.

### SUBRACK TOOLS DOCUMENTATION

power\_on\_tpm.usage\_hexample = '(es power on TPM1 and TPM3). %prog --t1 --t3\n(es power on all TPM). %prog --all\n'

TPM Power Control Script

This script provides a command-line interface for controlling the power state of TPMs (Trusted Platform Modules) in a SKALAB Subrack. It uses the OptionParser module to parse command-line options for selecting TPMs to power on.

### **Command-Line Options:**

-t1, -t2, ..., -t8: Select individual TPMs (1 to 8) to power on. -all: Select all TPMs to power on.

### **Usage Example:**

\$ python script\_name.py -t1 -t3 -all

Note: Replace 'script\_name.py' with the actual name of the script file.

power\_off\_tpm.usage\_hexample = '(es power off TPM1 and TPM3). %prog --t1 --t3\n(es power off all TPM). %prog --all\n'

TPM Power Control Script

This script provides a command-line interface for controlling the power state of TPMs (Trusted Platform Modules) in a SKALAB Subrack. It uses the OptionParser module to parse command-line options for selecting TPMs to power off.

### **Command-Line Options:**

-t1, -t2, ..., -t8: Select individual TPMs (1 to 8) to power off. -all: Select all TPMs to power off.

### **Usage Example:**

\$ python script\_name.py -t1 -t3 -all

Note: Replace 'script\_name.py' with the actual name of the script file.

#### i2c\_reg.get\_dev\_add(devname)

Get the I2C device address based on the device name.

### **Parameters**

**devname** (str) – The name of the I2C device.

### Returns

The I2C device address.

### **Return type**

int

### Raises

**SystemExit** – If the provided device name is incorrect.

## CHAPTER

## **FIVE**

# **INDICES AND TABLES**

- genindex
- modindex
- search

## **PYTHON MODULE INDEX**

```
b
                                                  W
                                                  web_server, 24
backplane, 1
С
config_ip, 27
cpld2mcu_serial_ctrl_2, 28
fpga_i2c_reg, 43
fpga_reg, 43
h
HardwareBaseClass, 13
HardwareThreadedClass, 16
i2c_reg, 43
m
management, 1
management_bsp, 31
management_flash, 31
management_mcu_uart, 37
management_pll, 28
management_spi, 38
mcu_update, 28
mng_update, 28
\verb"phy_marvell_88X2222_init", 28"
power_off_tpm, 43
power_on_tpm, 43
r
reg, 31
rmp, 31
S
subrack_hardware, 17
subrack_management_board, 4
{\tt subrack\_monitoring\_point\_lookup}, 11
```

48 Python Module Index

# **INDEX**

A	16
abort() (HardwareThreaded-	config_ip
Class.ThreadedHardwareCommand method),	module, 27
16	cpld2mcu_serial_ctrl_2
AbortCommand (class in HardwareThreadedClass), 16	module, 28
add_attribute() (HardwareBase-	CPLD_PLL_Locked (class in subrack_hardware), 17
Class.HardwareBaseDevice method), 14	D
add_command() (HardwareBase-	D
Class.HardwareBaseDevice method), 14	decode_register() (in module
Adu_Eth_Ping() (in module sub-	phy_marvell_88X2222_init), 28
$rack\_management\_board), 4$	<pre>detect_ip() (in module subrack_management_board),</pre>
all_monitoring_categories() (sub-	11
rack_management_board.SubrackMngBoard method), 9	DeviceErase() (management_flash.MngProgFlash method), 32
all_monitoring_points() (sub-	DeviceEraseChip() (manage-
rack_management_board.SubrackMngBoard	$ment\_flash.MngProgFlash\ method),\ 32$
method), 9	DeviceGetID() (management_flash.MngProgFlash
API_Version (class in subrack_hardware), 17	method), 32
AreTpmsOnCommand (class in subrack_hardware), 17	<pre>DeviceGetInfo() (management_flash.MngProgFlash     method), 32</pre>
В	DeviceWrite() (management_flash.MngProgFlash
backplane	method), 32
module, 1	disconnect() (management.MANAGEMENT method),
BackplaneInvalidParameter, 1	(Handran Dan Class Catallate it as Comment
BackplaneTemperature (class in subrack_hardware), 17	do() (HardwareBaseClass.GetAllAttributesCommand method), 13
<pre>bkpln_get_field()</pre>	do() (HardwareBaseClass.HardwareCommand method),
rack_management_board.SubrackMngBoard	15
method), 10	do() (HardwareBaseClass.ListAttributeCommand
bkpln_set_field() (sub-	method), 15
rack_management_board.SubrackMngBoard method), 10	do() (HardwareBaseClass.ListCommandCommand method), 15
BoardCurrent (class in subrack_hardware), 17	do() (HardwareThreadedClass.AbortCommand
BoardTemperature (class in subrack_hardware), 17	method), 16
byte_to_bool_array() (in module sub-rack_hardware), 24	do() (HardwareThreadedClass.IsCompletedCommand method), 16
,, = .	do() (HardwareThreaded-
C	Class.ThreadedHardwareCommand method), 16
cfg_10g() (in module phy_marvell_88X2222_init), 28	do() (subrack_hardware.AreTpmsOnCommand method),
checkLoad() (management.MANAGEMENT method), 1	17
completed() (HardwareThreaded-	do() (subrack_hardware.GetHealthDict method), 18

do() (subrack_hardware.GetHealthStatus method), 18	FlashDevice_waitTillReady() (manage-
do() (subrack_hardware.IsTpmOnCommand method),	ment_flash.MngProgFlash method), 34
19	FlashDevice_writeDisable() (manage-
do() (subrack_hardware.SetFanMode method), 20	$ment\_flash.MngProgFlash\ method),\ 34$
do() (subrack_hardware.SetFanSpeed method), 21	FlashDevice_writeEnable() (manage-
do() (subrack_hardware.SetPSFanSpeed method), 21	ment_flash.MngProgFlash method), 34
do() (subrack_hardware.TpmInfo method), 23	FlashDevice_writePage() (manage-
do_GET() (web_server.MyServer method), 24	ment_flash.MngProgFlash method), 34
<pre>dt_to_timestamp() (in module sub- rack_management_board), 11</pre>	FlashDevice_writeReg() (manage- ment_flash.MngProgFlash method), 35
rack_management_board), 11	FlashDevice_writesector() (manage-
E	ment_flash.MngProgFlash method), 35
<pre>exec_cmd() (in module subrack_management_board),</pre>	flatten_dict() (in module sub-
11	rack_management_board), 11
execute_command() (HardwareBase-	<pre>format_num() (in module management), 4</pre>
Class.HardwareBaseDevice method), 14	fpga_i2c_reg
execute_command() (sub-	module, 43
rack_hardware.SubrackHardware method),	fpga_reg
21	module, 43
F	G
	Get_API_version() (sub-
FanMode (class in subrack_hardware), 18 FanSpeed (class in subrack_hardware), 18	rack_management_board.SubrackMngBoard
FanSpeedPercent (class in subrack_hardware), 18	method), 8
filter_list_by_level() (in module management), 4	<pre>get_attribute()</pre>
find_register() (management.MANAGEMENT	Class.HardwareBaseDevice method), 14
method), 1	<pre>get_bios() (management.MANAGEMENT method), 2</pre>
<pre>find_register_names() (manage-</pre>	<pre>get_board() (management.MANAGEMENT method), 2</pre>
ment.MANAGEMENT method), 1	<pre>get_board_info() (management.MANAGEMENT</pre>
firmwareProgram() (manage-	method), 2
ment_flash.MngProgFlash method), 35	<pre>get_board_info()</pre>
firmwareRead() (management_flash.MngProgFlash	rack_management_board.SubrackMngBoard
method), 36	method), 10 get_dev_add() (in module i2c_reg), 43
FlashCmd (class in cpld2mcu_serial_ctrl_2), 28 FlashDevice_chiperase() (manage-	get_extended_info() (management.MANAGEMENT
FlashDevice_chiperase() (manage- ment_flash.MngProgFlash method), 33	method), 2
FlashDevice_Enter4byteAddMode() (manage-	get_health_dict() (sub-
ment_flash.MngProgFlash method), 33	rack_management_board.SubrackMngBoard
FlashDevice_erase() (manage-	method), 10
ment_flash.MngProgFlash method), 33	<pre>get_health_status() (sub-</pre>
FlashDevice_eraseSector() (manage-	rack_management_board.SubrackMngBoard
ment_flash.MngProgFlash method), 33	method), 10
FlashDevice_Exit4byteAddMode() (manage-	<pre>get_health_status_w_elapsed() (sub-</pre>
ment_flash.MngProgFlash method), 33	rack_management_board.SubrackMngBoard
FlashDevice_prepareCommand() (manage-	method), 10
ment_flash.MngProgFlash method), 33	<pre>get_mac() (management.MANAGEMENT method), 2 get_mac_from_eep() (in module config_ip), 27</pre>
FlashDevice_readIdentification() (manage-	get_max_width() (in module management), 4
<pre>ment_flash.MngProgFlash method), 33 FlashDevice_readPage() (manage-</pre>	get_max_width() (in module reg), 31
ment_flash.MngProgFlash method), 34	get_port_cfg() (in module module
FlashDevice_readReg() (manage-	phy_marvell_88X2222_init), 29
ment_flash.MngProgFlash method), 34	<pre>get_register_name_by_address() (manage-</pre>
FlashDevice_readsector() (manage-	ment.MANAGEMENT method), 2
ment_flash.MngProgFlash method), 34	<pre>get_shift_from_mask() (in module management), 4</pre>

<u> </u>	<pre>method), 6 GetSubrackTemperatures()</pre>	(sub-
<pre>method), 10 Get_Subrack_TimeTS() (sub-</pre>	rack_management_board.SubrackMngB method), 6	oara
	GetTPM_Add_List()	(sub-
method), 8	rack_management_board.SubrackMngB	`
<pre>get_switch_status()</pre>	method), 8	
phy_marvell_88X2222_init), 29	<pre>GetTPMCurrent()</pre>	(sub-
<pre>Get_tpm_alarms_vector() (sub-</pre>	rack_management_board.SubrackMngB	oard
rack_management_board.SubrackMngBoard	method), 6	
method), 8	<pre>GetTPMGlobalStatusAlarm()</pre>	(sub-
Get_TPM_temperature_vector() (sub-	rack_management_board.SubrackMngB	oard
rack_management_board.SubrackMngBoard	method), 6	
method), 8	GetTPMInfo() (subrack_management_board.Sub	rackMngBoard
get_xml_from_board() (management.MANAGEMENT	method), 7	-l-M Dl
method), 2	GetTPMIP() (subrack_management_board.Subrack_m	скипдвоага
GetAllAttributesCommand (class in HardwareBase- Class), 13	<pre>method), 7 GetTPMMCUTemperature()</pre>	(sub-
GetCPLDLockedPLL() (sub-	rack_management_board.SubrackMngB	`
rack_management_board.SubrackMngBoard	method), 7	oara
	GetTPMOnOffVect()	(sub-
GetFanAlarm() (subrack_management_board.SubrackMn,		`
method), 5	method), 7	
<pre>GetFanMode() (subrack_management_board.SubrackMng.</pre>	· ·	brackMngBoard
method), 5	method), 7	
<pre>GetFanPwm() (subrack_management_board.SubrackMngB</pre>	<b>@at</b> dTPMPresent()	(sub-
method), 5	rack_management_board.SubrackMngB	oard
${\tt GetFanRpm()} \ (subrack\_management\_board.SubrackMngBarter) \ (s$		
	<pre>GetTPMSupplyFault()</pre>	(sub-
<pre>GetFanSpeed() (subrack_management_board.SubrackMn,</pre>	gBoard rack_management_board.SubrackMngB method), 7	oard
	<pre>GetTPMTemperatures()</pre>	(sub-
GetHealthStatus (class in subrack_hardware), 18	rack_management_board.SubrackMngB	oard
GetLockedPLL() (sub-	method), 7	
rack_management_board.SubrackMngBoard		(sub-
method), 5	rack_management_board.SubrackMngB	oard
GetPingCpld() (subrack_management_board.SubrackMn		(aul
<pre>method), 6 GetPingTPM() (subrack_management_board.SubrackMng.</pre>	GetUPSStatus() Board rack_management_board.SubrackMngB	(sub-
method), 6	method), 8	оата
GetPllSource() (sub-	GetVoltageAlarm()	(sub-
rack_management_board.SubrackMngBoard	rack_management_board.SubrackMngB	`
method), 6	method), 8	
GetPowerAlarm() (sub-	,, -	
rack_management_board.SubrackMngBoard	Н	
method), 6	HardwareAttribute (class in HardwareBaseCla	uss), 13
GetPSFanSpeed() (sub-	HardwareBaseClass	,,
rack_management_board.SubrackMngBoard	module, 13	
method), 5	HardwareBaseDevice (class in HardwareBase	eClass),
${\tt GetPSIout()} \ (subrack\_management\_board.SubrackMngB$	oard 14	
method), 5	${\tt Hardware Command} \ (class \ in \ Hardware Base Class$	), 15
GetPSPower() (subrack_management_board.SubrackMng.		
method), 6	module, 16	
${\tt GetPSVout()} \ (subrack\_management\_board.SubrackMngB$	Mexstring2ascii() (in module management), 4	

I	MngProgFlash (class in management_flash), 31
i2c_reg	module
module, 43	backplane, 1
<pre>initialize() (subrack_hardware.SubrackHardware     method), 21</pre>	<pre>config_ip, 27 cpld2mcu_serial_ctrl_2, 28</pre>
<pre>Initialize() (subrack_management_board.SubrackMng</pre>	Board <sup>f</sup> pga_i2c_reg, 43 fpga_reg, 43
method), 8	HardwareBaseClass, 13
int2ip() (in module config_ip), 27	HardwareThreadedClass, 16
int2ip() (in module subrack_management_board), 11	i2c_reg, 43
<pre>ipstr2hex() (in module subrack_management_board),</pre>	management, 1
11	management_bsp, 31
is_blocking() (HardwareThreaded- Class.ThreadedHardwareCommand method),	management_flash, 31
16	management_mcu_uart, 37
IsCompletedCommand (class in HardwareThreaded-	management_p11,28
Class), 16	management_spi,38
IsTpmOnCommand (class in subrack_hardware), 18	mcu_update, 28
151pmoncommand (class in subrack_naraware), 16	mng_update, 28
L	phy_marvell_88X2222_init, 28
	power_off_tpm, 43
list_register_names() (manage-	power_on_tpm, 43
ment.MANAGEMENT method), 3	reg, 31
ListAttributeCommand (class in HardwareBaseClass),	rmp, 31
15	subrack_hardware, 17
ListCommandCommand (class in HardwareBaseClass),	<pre>subrack_management_board, 4</pre>
15	<pre>subrack_monitoring_point_lookup, 11</pre>
load_bitstream() (in module	web_server, 24
<pre>cpld2mcu_serial_ctrl_2), 28 load_firmware_blocking() (manage-</pre>	MyServer (class in web_server), 24
ment.MANAGEMENT method), 3	N
loadBitstream() (in module mcu_update), 28	name() (HardwareBaseClass.HardwareAttribute
loadBitstream() (management_flash.MngProgFlash	method), 13
method), 36	name() (HardwareBaseClass.HardwareCommand
M	method), 15 nuple2mac() (in module config_ip), 27
management	nuprezmac() (in module conjig_ip), 27
module, 1	P
MANAGEMENT (class in management), 1	
management_bsp	partial (class in subrack_monitoring_point_lookup), 11
module, 31	phy_marvell_88X2222_init
management_flash	module, 28
module, 31	pll_calib() (management.MANAGEMENT method), 3 pll_dumpcfg() (management.MANAGEMENT
management_mcu_uart	method), 3
module, 37	pll_ioupdate() (management.MANAGEMENT
management_pl1	method), 3
module, 28	pll_ldcfg() (management.MANAGEMENT method), 3
management_spi	pll_read_with_update() (manage-
module, 38	ment.MANAGEMENT method), 3
mangle_dict() (in module web_server), 25	pll_write_with_update() (manage-
mcu_update	ment.MANAGEMENT method), 3
module, 28	PllInitialize() (sub-
mng_update	rack_management_board.SubrackMngBoard
module, 28	method), 8
MngMcuUart (class in management_mcu_uart), 37	power_off_tpm

module, 43	read_value() (subrack_hardware.PSFanSpeed
power_on_tpm	method), 19
module, 43	read_value() (subrack_hardware.PSPower method), 19
PowerDownCommand (class in subrack_hardware), 20	<pre>read_value() (subrack_hardware.PSVoltage method),</pre>
PowerOffTPM() (subrack_management_board.SubrackMn	
method), 8	<pre>read_value() (subrack_hardware.Subrack_PLL_Locked</pre>
${\tt PowerOffTpmCommand}~(class~in~subrack\_hardware),~20$	method), 21
PowerOnTpmCommand (class in subrack_hardware), 20	read_value() (subrack_hardware.Subrack_Timestamp
PowerUpCommand (class in subrack_hardware), 20	method), 22
<pre>pprint_table() (in module management), 4</pre>	read_value() (subrack_hardware.TPM_Add_List
pprint_table() (in module reg), 31	method), 22
PSCurrent (class in subrack_hardware), 19	$\verb"read_value()" (subrack\_hardware.TPM\_Temperature\_Alarms")$
PSFanSpeed (class in subrack_hardware), 19	method), 22
PSPower (class in subrack_hardware), 19	read_value() (subrack_hardware.TPM_Temperatures
PSVoltage (class in subrack_hardware), 19	method), 22
R	read_value() (subrack_hardware.TPM_Voltage_Alarms
	method), 22
rd() (in module phy_marvell_88X2222_init), 29	read_value() (subrack_hardware.TpmCurrents
read() (HardwareBaseClass.HardwareAttribute	method), 22
method), 13	read_value() (subrack_hardware.TpmIPs method), 23
read22() (in module phy_marvell_88X2222_init), 29	read_value() (subrack_hardware.TpmMCUTemperatures
read45() (in module phy_marvell_88X2222_init), 29	method), 23
read_and_decode() (in module	read_value() (subrack_hardware.TpmOnOffVect
phy_marvell_88X2222_init), 29	method), 23
read_register() (management.MANAGEMENT	read_value() (subrack_hardware.TpmPowers method),
method), 3	23
read_scratch() (in module	read_value() (subrack_hardware.TpmPresent
phy_marvell_88X2222_init), 30	method), 24
read_spi() (management.MANAGEMENT method), 4	read_value() (subrack_hardware.TpmSupplyFault
read_string() (in module config_ip), 27	method), 24
read_tpm_singlewire() (sub-	read_value() (subrack_hardware.TpmVoltages
rack_management_board.SubrackMngBoard	method), 24
method), 11	read_value() (subrack_hardware.ups_status method), 24
read_value() (HardwareBaseClass.HardwareAttribute	
method), 13	<pre>read_wis() (in module phy_marvell_88X2222_init), 30 readmodifywrite() (in module</pre>
read_value() (subrack_hardware.API_Version	
method), 17	phy_marvell_88X2222_init), 30
read_value() (subrack_hardware.BackplaneTemperature	
method), 17	reg module, 31
read_value() (subrack_hardware.BoardCurrent	reset_mcu() (management_mcu_uart.MngMcuUart
method), 17	method), 37
read_value() (subrack_hardware.BoardTemperature	
method), 17	rmp module, 31
read_value() (subrack_hardware.CPLD_PLL_Locked	module, 51
method), 17	S
read_value() (subrack_hardware.FanMode method),	
18	
read_value() (subrack_hardware.FanSpeed method),	method), 36
18 road value() (subrack hardware Fan Speed Percent	set_attribute() (HardwareBase- Class.HardwareBaseDevice method), 15
read_value() (subrack_hardware.FanSpeedPercent	
method), 18	<pre>set_field() (in module phy_marvell_88X2222_init), 30</pre>
read_value() (subrack_hardware.PSCurrent method), 19	set_SFP() (in module phy_marvell_88X2222_init), 30
17	SetFanMode (class in subrack_hardware), 20

SetFanMode() (subrack_management_board.SubrackMng	Board
method), 8	thread() (HardwareThreaded-
SetFanSpeed (class in subrack_hardware), 21	Class.ThreadedHardwareCommand method),
SetFanSpeed() (subrack_management_board.SubrackMn	gBoard 16
method), 9	thread() (subrack_hardware.PowerDownCommand
SetPSFanSpeed (class in subrack_hardware), 21	method), 20
SetPSFanSpeed() (sub-	<pre>thread() (subrack_hardware.PowerOffTpmCommand</pre>
rack_management_board.SubrackMngBoard	method), 20
method), 9	<pre>thread() (subrack_hardware.PowerOnTpmCommand</pre>
SetTPMIP() (subrack_management_board.SubrackMngBo	ard method), 20
method), 9	thread() (subrack_hardware.PowerUpCommand
SetUPSVoltageAlarmThresholds() (sub-	method), 20
rack_management_board.SubrackMngBoard	${\tt Threaded Hardware Command} \ ({\it class in Hardware Thread-}$
method), 9	edClass), 16
SetUPSVoltageWarningThresholds() (sub-	<pre>TPM_Add_List (class in subrack_hardware), 22</pre>
rack_management_board.SubrackMngBoard	TPM_Temperature_Alarms (class in sub-
method), 9	rack_hardware), 22
<pre>spi_chipselect() (management_flash.MngProgFlash</pre>	TPM_Temperatures (class in subrack_hardware), 22
method), 36	<pre>TPM_Voltage_Alarms (class in subrack_hardware), 22</pre>
spi_config() (management_flash.MngProgFlash	TpmCurrents (class in subrack_hardware), 22
method), 36	TpmInfo (class in subrack_hardware), 23
<pre>spi_mux_selection()</pre>	TpmIPs (class in subrack_hardware), 23
ment_flash.MngProgFlash method), 36	<pre>TpmMCUTemperatures (class in subrack_hardware), 23</pre>
spi_rx_available() (manage-	<pre>TpmOnOffVect (class in subrack_hardware), 23</pre>
ment_flash.MngProgFlash method), 36	TpmPowers (class in subrack_hardware), 23
spi_sync() (management_flash.MngProgFlash	<pre>TpmPresent (class in subrack_hardware), 24</pre>
method), 37	<pre>TpmSupplyFault (class in subrack_hardware), 24</pre>
spi_trigger() (management_flash.MngProgFlash	<pre>TpmVoltages (class in subrack_hardware), 24</pre>
method), 37	twos_comp() (in module backplane), 1
spi_tx_remaining() (manage-	
ment_flash.MngProgFlash method), 37	U
spiregisters (class in management_flash), 37	<pre>uart_receive_byte()</pre>
SPITransaction() (management_flash.MngProgFlash	ment_mcu_uart.MngMcuUart method), 37
method), 35	<pre>uart_send_buffer()</pre>
<pre>start_mcu_sam_ba_monitor() (manage- ment_mcu_uart.MngMcuUart method), 37</pre>	ment_mcu_uart.MngMcuUart method), 37
subrack_hardware	<pre>uart_send_buffer_wrx()</pre>
module, 17	ment_mcu_uart.MngMcuUart method), 38
subrack_management_board	<pre>uart_send_byte()</pre>
module, 4	ment_mcu_uart.MngMcuUart method), 38
subrack_monitoring_point_lookup	ups_status (class in subrack_hardware), 24
module, 11	<pre>usage_hexample (in module power_off_tpm), 43</pre>
Subrack_PLL_Locked (class in subrack_hardware), 21	<pre>usage_hexample (in module power_on_tpm), 43</pre>
Subrack_Timestamp (class in subrack_hardware), 22	147
SubrackExecFault, 5	W
SubrackHardware (class in subrack_hardware), 21	web_server
SubrackInitialConfiguration() (sub-	module, 24
rack_management_board.SubrackMngBoard	wr() (in module phy_marvell_88X2222_init), 30
method), 9	<pre>write() (HardwareBaseClass.HardwareAttribute</pre>
SubrackInvalidCmd, 5	method), 14
SubrackInvalidParameter, 5	<pre>write22() (in module phy_marvell_88X2222_init), 31</pre>
SubrackMngBoard (class in sub-	<pre>write22_reg() (in module phy_marvell_88X2222_init),</pre>
rack_management_board), 5	31
_ 0	<pre>write45() (in module phy_marvell_88X2222_init), 31</pre>

```
write_register()
                      (management.MANAGEMENT
        method), 4
write_scratch()
                                          module
        phy_marvell_88X2222_init), 31
write_spi() (management.MANAGEMENT method), 4
write_string() (in module config_ip), 28
write_tpm_singlewire()
                                           (sub-
        rack_management_board.SubrackMngBoard
        method), 11
write_value()
                                  (HardwareBase-
        Class.HardwareAttribute method), 14
write_value() (subrack_hardware.FanMode method),
```