Subrack Management API

Release 1.0

Simone Chiarucci (INAF-OAA)

Jan 08, 2024

CONTENTS:

CHAPTER

ONE

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
str

`get_register_name_by_address(add)`

Get register name by address.

Parameters
add (*int*) – Register address.

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

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

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='slots') 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*]) → 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.

CHAPTER

TWO

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 inthe 'value' field

do(*params=None*)

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

Returns
Dictionary of returned response

Return type
dict

class HardwareBaseClass.**HardwareAttribute**(*name*, *init_value*, *hardware=None*, *read_write=0*, *num_params=1*)

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

Returns
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
dict

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

class HardwareThreadedClass.**ThreadedHardwareCommand**(*name*, *hardware=None*, *num_params=0*, *blocking=False*)

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
float

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

Returns
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: - emulation (bool): Indicates whether the Subrack is in emulation mode.

Returns: None

class subrack_hardware.**Subrack_PLL_Locked**(*name*, *init_value*, *hardware=None*, *read_write=0*, *num_params=1*)

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

class subrack_hardware.**Subrack_Timestamp**(*name*, *init_value*, *hardware=None*, *read_write=0*, *num_params=1*)

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

class subrack_hardware.**TPM_Temperature_Alarms**(*name*, *init_value*, *hardware=None*, *read_write=0*, *num_params=1*)

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

class subrack_hardware.**TPM_Temperatures**(*name*, *init_value*, *hardware=None*, *read_write=0*, *num_params=1*)

TPMs Temperature vectors Returns 8 TPMBoard temepratures , 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

class subrack_hardware.**TPM_Voltage_Alarms**(*name*, *init_value*, *hardware=None*, *read_write=0*, *num_params=1*)

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()

Returns
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

Returns
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()

Returns
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

CHAPTER

THREE

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

str

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).

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

reg.**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_fifo_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.

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
str

get_register_name_by_address(*add*)

Get register name by address.

Parameters
add (*int*) – Register address.

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

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.

CHAPTER

FOUR

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

INDEX

E

F

G

H

I

L

M

N

P

R

S

T

U

W