STUSB4500 Library Documentation

Release 1.0

Jessica Stokes

Oct 01, 2020

CONTENTS

CircuitPython driver for STUSB4500 USB Power Delivery board.

CHAPTER

ONE

IMPLEMENTATION NOTES

Based on SparkFun's Arduino library. Further information and notes from the ST reference implementation. Python library and project structure inspired by Adafruit's VEML7700 library, and Adafruit's other such libraries.

CHAPTER TWO

HARDWARE

- SparkFun STUSB4500

While this was developed for and tested with the SparkFun breakout, it likely works with any STUSB4500-based board which exposes I2C, including:

- USB C PD Sink by ketszim
- STMicroelectronics EVAL-SCS001V1

CHAPTER THREE

DEPENDENCIES

This driver depends on:

- Adafruit CircuitPython
- Bus Device

Please ensure all dependencies are available on the CircuitPython filesystem. This is easily achieved by downloading the Adafruit library and driver bundle.

CHAPTER

FOUR

USAGE EXAMPLE

```python
import board
import busio
import stusb4500

i2c = busio.I2C(board.SCL, board.SDA)
pd_board = stusb4500.STUSB4500(i2c)

print("Current PD Configuration:")
print("PDO Number: {}".format(pd_board.pdo_number))

for i in range(1, 4):
    print(
        "PDO{}: {}V (+{}%/-{}%), {}A".format(
            i,
            pd_board.get_voltage(i),
            pd_board.get_upper_voltage_limit(i),
            pd_board.get_lower_voltage_limit(i),
            pd_board.get_current(i)
        )
    )

print("Flex Current: {}".format(pd_board.flex_current))
print("External Power: {}".format(pd_board.external_power))
print("USB Communication Capable: {}".format(pd_board.usb_comm_capable))
print("Configuration OK GPIO: {}".format(pd_board.config_ok_gpio))
print("GPIO Control: {}".format(pd_board.gpio_ctrl))
print("Enable Power Only Above 5V: {}".format(pd_board.power_above_5v_only))
print("Request Source Current: {}".format(pd_board.req_src_current))
print("Factory Default: {}".format(pd_board.is_factory_defaults))
```

CHAPTER

FIVE

CONTRIBUTING

Contributions are welcome! Please read our Code of Conduct before contributing to help this project stay welcoming.

CHAPTER SIX

BUILDING

This library, along with dependencies and user code, can be too large to be useful on M0 boards like the Trinket M0. Built versions should show up once CI is configured and I've tagged a version, but for both your and my own development purposes I'm documenting how to just get an mpy file here.

You will need to install Adafruit's mpy-cross compiler, either Adafruit's built version or on macOS you can install via Homebrew by running `brew install ticky/utilities/circuitpython`.

CHAPTER SEVEN

DOCUMENTATION

For information on building library documentation, please check out this guide.

CHAPTER

EIGHT

TABLE OF CONTENTS

8.1 Simple test

Ensure your device works with this simple test.

Listing 1: examples/stusb4500_simpletest.py

```python
# SPDX-FileCopyrightText: Copyright (c) 2020 Jessica Stokes
#
# SPDX-License-Identifier: MIT
import board
import busio
import stusb4500

i2c = busio.I2C(board.SCL, board.SDA)
pd_board = stusb4500.STUSB4500(i2c)

print("Current PD Configuration:")
print("PDO Number: {}".format(pd_board.pdo_number))

for i in range(1, 4):
    print(
        "PDO{}: {}V (+{}%/-{}%), {}A".format(
            i,
            pd_board.get_voltage(i),
            pd_board.get_upper_voltage_limit(i),
            pd_board.get_lower_voltage_limit(i),
            pd_board.get_current(i)
        )
    )

print("Flex Current: {}".format(pd_board.flex_current))
print("External Power: {}".format(pd_board.external_power))
print("USB Communication Capable: {}".format(pd_board.usb_comm_capable))
print("Configuration OK GPIO: {}".format(pd_board.config_ok_gpio))
print("GPIO Control: {}".format(pd_board.gpio_ctrl))
print("Enable Power Only Above 5V: {}".format(pd_board.power_above_5v_only))
print("Request Source Current: {}".format(pd_board.req_src_current))
print("Factory Default: {}".format(pd_board.is_factory_defaults))
```

8.2 `stusb4500` - CircuitPython driver for STUSB4500 USB Power Delivery board

class `stusb4500.`**STUSB4500**(*i2c_bus*, *address=40*)

Represents a STUSB4500 I2C device and manages its communication, locking and configuration.

Properties and `set_*` methods do not directly write to the device, instead the device's configuration is read when this class is initialised (or by explicitly calling *`read_parameters()`* if changes should be abandoned), and parameters are read and written to that buffer.

To change settings on the device, you must set the desired configuration using the supplied methods and properties, and once satisfied may call the *`write_parameters()`* method to commit the parameters to the device's non-volatile memory.

Parameters

- **i2c_bus** (*busio.I2C*) – The I2C bus the STUSB4500 is connected to.
- **address** (*int*) – The I2C device address. If omitted, the default of `0x28` is used.

property config_ok_gpio

Controls the behaviour of the `VBUS_EN_SNK`, `POWER_OK2`, and `POWER_OK3` pins based on source capabilities.

Must be one of the following:

- **0: Configuration 1** `VBUS_EN_SNK` will be pulled low when a source is attached. `POWER_OK2` and `POWER_OK3` will always stay high.
- **2: Configuration 2** `VBUS_EN_SNK` will be pulled low when a source is attached. `POWER_OK2` will be pulled low when PDO2 is agreed upon. `POWER_OK3` will be pulled low when PDO3 is agreed upon.
- **3: Configuration 3** `VBUS_EN_SNK` will be pulled low when a source is attached. `POWER_OK2` will be pulled low when the attached source indicates availability of 3.0A at 5V. `POWER_OK3` will be pulled low when the attached source indicates availability of 1.5A at 5V.

`2` is the default. Values below `2` are treated as `0`, and values above `3` are treated as `3`.

Return type int

property external_power

The value for the `SNK_UNCONS_POWER` parameter.

`SNK_UNCONS_POWER` is the unconstrained power bit setting in the capabilities message sent by the sink. `True` means an external source of power is available and is sufficient to adequately power the system while charging external devices.

Return type bool

property flex_current

A float value to set the current common to all PDOs. This value is only used in the power negotiation if the current value for that PDO is set to `0`.

Must be between 0A and 5A.

Return type float

get_current(*pdo=None*)

Returns the current requested for the PDO number

Parameters **pdo** (*int*) – PDO number (either `1`, `2`, or `3`) to retrieve the parameter for. If omitted, or another value is passed, defaults to `3`.

Returns The current requested for the PDO, in amps.

Return type decimal

`get_lower_voltage_limit`(*pdo=None*)
Returns the under voltage lockout parameter for the PDO number

Parameters **`pdo`** (*`int`*) – PDO number (either `1`, `2`, or `3`) to retrieve the parameter for. If omitted, or another value is passed, defaults to `3`.

Returns The under voltage limit requested for the PDO, in percent.

Return type decimal

`get_upper_voltage_limit`(*pdo=None*)
Returns the over voltage lockout parameter for the PDO number

Parameters **`pdo`** (*`int`*) – PDO number (either `1`, `2`, or `3`) to retrieve the parameter for. If omitted, or another value is passed, defaults to `3`.

Returns The over voltage limit requested for the PDO, in percent.

Return type decimal

`get_voltage`(*pdo=None*)
Returns the voltage requested for the PDO number

Parameters **`pdo`** (*`int`*) – PDO number (either `1`, `2`, or `3`) to retrieve the parameter for. If omitted, or another value is passed, defaults to `3`.

Returns The voltage requested for the PDO, in volts.

Return type decimal

`property gpio_ctrl`
Controls the behaviour setting for the GPIO pin

Must be one of the following:

- **`0`: Software Controlled GPIO** The output state is controlled by the value stored in bit 0 of register `0x2D`. Low when the bit is 1.
- **`1`: Error Recovery** Hardware fault detection such as over temperature, over voltage on the CC pins, or after a hard reset. Low when a hardware fault is detected.
- **`2`: Debug** Debug accessory detection. Low when a debug accessory was detected.
- **`3`: Sink Power** Indicates USB Type-C current capability advertised by the source. Low when source indicates availability of 3.0A at 5V.

Return type int

`property is_factory_defaults`
Check whether the current configuration is the same as the factory defaults.

The factory defaults can be applied using the *`write_parameter_defaults()`* method.

Return type bool

`property pdo_number`
The value saved in memory for the highest priority PDO number.

Must be `1`, `2`, or `3`. `3` is the default.

If set to `2`, PDO3 will be ignored and PDO2 will be negotiated, followed by PDO1. If set to `1`, only PDO1 will be negotiated.

Return type int

property power_above_5v_only

When `True`, output power is only enabled when the source is attached, and the voltage is negotiated for either PDO2 or PDO3.

`False` means output power will be enabled whenever a source is connected.

Return type bool

read_parameters()

Read all the current NVM parameters from the device.

The *STUSB4500* object will automatically read the current NVM parameters when initialised, however, you may use this function to fetch the currently stored parameters if otherwise needed, for example, to reset any pending changes which have not yet been written to the device.

property req_src_current

`False` requests the sink current as operating current in the RDO message. `True` requests the source current as operating current in the RDO message.

Return type bool

set_current(*current*, *pdo=None*)

Sets the desired current for the PDO number

Current is stored as a 4-bit value, and has different precision depending on the magnitude. Values between 0.5 and 3.0A are set in 0.25A steps, while values between 3.0 and 5.0A are in 0.5A steps.

Input values are automatically rounded to the nearest possible value.

Parameters

- **current** (*decimal*) – The current to request for the PDO, in amps. Must be between 0.0A and 5.0A.
- **pdo** (*int*) – PDO number (either `1`, `2`, or `3`) to set the parameter for. If omitted, or another value is passed, defaults to `3`.

set_lower_voltage_limit(*lower_limit*, *pdo=None*)

Sets the desired under voltage lockout parameter for the PDO number

Parameters

- **lower_limit** (*decimal*) – The under voltage limit to request for the PDO, in percent. Must be between 5% and 20%.
- **pdo** (*int*) – PDO number (either `2`, or `3`) to set the parameter for. If omitted, or another value is passed, defaults to `3`. PDO1 is always set to 5%.

set_upper_voltage_limit(*upper_limit*, *pdo=None*)

Sets the desired over voltage lockout parameter for the PDO number

Parameters

- **upper_limit** (*decimal*) – The over voltage limit to request for the PDO, in percent. Must be between 5% and 20%.
- **pdo** (*int*) – PDO number (either `1`, `2`, or `3`) to set the parameter for. If omitted, or another value is passed, defaults to `3`.

set_voltage(*voltage*, *pdo=None*)

Sets the desired voltage for the PDO number

Parameters

- **voltage** (*decimal*) – The voltage to request for the PDO, in volts. Must be between 5.0V and 20.0V.
- **pdo** (*int*) – PDO number (either `2`, or `3`) to set the parameter for. If omitted, or another value is passed, defaults to `3`. PDO1 is always set to 5.0V.

property usb_comm_capable

`USB_COMM_CAPABLE` refers to USB2.0 or 3.x data communication capability by the sink system. `True` means that the sink supports data communication.

Return type bool

write_parameter_defaults()

Writes the factory-default parameters to the device.

Note that once written, the factory default configuration will not automatically be read into the configuration buffer. *read_parameters()* must be called after writing the defaults in order to do so.

write_parameters()

Writes all current parameters to the device.

CHAPTER NINE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX