PulsarAPI

Generated by Doxygen 1.9.2

Chapter 1

PulsarAPI

1.1 Introduction

This is a Dynamic Link Library (DLL) / Shared Object (SO) to control the Advanced illumination Pulsar 320E Controller from native code such as C/C++.

See the changelog for revision history.

1.2 Usage

The only header you need to include in your own code is `PulsarAPI.h`. The functions and data structures inside are documented in the included PDF.

1.2.1 Example: USB

To connect to a controller over USB (Port 0 in this case) and setting Output 1 to 5A, 350μs pulsewidth.

```
API_CHAN_CONFIG_STRUCT_T info;
int PORT = 0;
ERRCODE_T err;
const char* PRODUCT_NAME = "Ai Pulsar 320 Controller";
err = api_init(PORT, PRODUCT_NAME, NULL);
if(err) {
    reportError(err);
}
else {
    info.mode = API_MODE_PULSED;
    info.trigger = API_TRIG_1;
    info.current = 5.0; //Amps
    info.width = 350; //microseconds
    err = api_set_channel_config(PORT, API_CHAN_1, API_CHAN_CONFIG_ACTIVE, &info);
    if (err) {
        reportError(err);
    }
}
api_shutdown(port);
```

1.2.2 Example: TCP

To connect to a controller over TCP (default static IP) and setting Output 1 to 5A, 350μs pulsewidth.

```
API_CHAN_CONFIG_STRUCT_T info;
int PORT = 25; //Use port 25+ for TCP
ERRCODE_T err;
const char* PRODUCT_NAME = "Ai Pulsar 320 Controller";
err = api_connect(PORT, PRODUCT_NAME, "192.168.1.80");
if(err) {
    reportError(err);
}
else {
    info.mode = API_MODE_PULSED;
    info.trigger = API_TRIG_1;
    info.current = 5.0; //Amps
    info.width = 350; //microseconds
    err = api_set_channel_config(PORT, API_CHAN_1, API_CHAN_CONFIG_ACTIVE, &info);
    if (err) {
        reportError(err);
    }
}
api_shutdown(port);
```

Chapter 2

Changelog

2.1 Version 2.2.0

- Added function api_get_dll_full_version to get full version number programmatically

2.2 Version 2.1.3

- Pulsar 320 compatibility fixes:
  - TCP connection
  - Trigger map wasn't set correctly

2.3 Version 2.1.2

- Improved compatibility with Pulsar 320

2.4 Version 2.1.1

- Fix Repeat mode in Pulsar 320

2.5 Version 2.1.0

- Improved USB connection, fixed conflicts with other STM-based devices
- Added backward compatibility with Pulsar 320
  - Uses modified MPDH8USB_TCP with public functions renamed to avoid conflicting with AiUSBCmd
  - Modified MPDH8USB_TCP is now also cross-platform and available on Linux as well

2.6 Version 2.0.1

- Added API documentation generated by Doxygen
- Miscelleaneus bug fixes

2.7 Version 2.0

- Initial release for Pulsar 320E, cross-platform and available on Windows and Linux x86 and x86-64.

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 5

Class Documentation

5.1 API_CHAN_CONFIG_STRUCT_T Struct Reference

Channel/output configuration struct.

`#include <PulsarAPI.h>`

Public Attributes

- UINT8_T **trigger**
  Trigger input.
- UINT8_T **rising**
  Rising edge (1 = rising, 0 = falling)
- UINT8_T mode
  Operating mode.
- UINT8_T **reserved_byte**
- UINT32_T **repeat_cnt**
  Number of repeat pulses per trigger (0 = 1 pulse)
- UINT16_T **delay**
  Pulse delay (microseconds)
- FLOAT32_T **current**
  Pulse current (Amps)
- UINT32_T **width**
  Pulse width (microseconds)
- UINT32_T **offtime**
  Off time between repeat pulses (microseconds)

5.1.1 Detailed Description

Channel/output configuration struct.

5.1.2 Member Data Documentation

5.1.2.1 mode

```
UINT8_T API_CHAN_CONFIG_STRUCT_T::mode
```

Operating mode.

See also

API_MODE_T

The documentation for this struct was generated from the following file:

- PulsarAPI.h

Chapter 6

File Documentation

6.1 PulsarAPI.h File Reference

This module contains external definitions for the High Level User Application Programming Interface DLL functions in the Pulsar320 and Pulsar320E projects.

Classes

- struct API_CHAN_CONFIG_STRUCT_T

  Channel/output configuration struct.

Macros

- #define **_PULSARAPI_H**
- #define **CALLSPEC**
- #define **PULSAR_API** extern "C"
- #define **API_PULSAR_PRODUCT_NAME** "Ai Pulsar 320 Controller"
- #define **MPD_TYPES**
- #define **API_NUM_CHAN** 2
- #define **API_NUM_TRIG** 2
- #define **API_CHAN_CONFIG_DEFAULT** 1
- #define **API_CHAN_CONFIG_ACTIVE** 2
- #define **API_NUM_OPTIONAL_TESTS** 1
- #define **API_MAX_ROTARY** 999
- #define **API_DIP_MODE0** 0x1
- #define **API_DIP_MODE1** 0x2
- #define **API_DIP_MODE2** 0x4
- #define **API_DIP_MODE3** 0x8
- #define **API_TRIG_IN1** 0x1
- #define **API_TRIG_IN2** 0x2
- #define **API_CONN_LH_DETECT** 0x1
- #define **API_CONN_USBDETECT** 0x2

- #define **API_CONN_PCPRESENT** 0x4
- #define **API_CONN_PC_READY** 0x8
- #define **API_SUPPLY_VIN_ON** 0x1
- #define **API_SUPPLY_LV_ON** 0x2
- #define **API_SUPPLY_HV_ON** 0x4
- #define **API_EXP_GPIN1** 0x1
- #define **API_EXP_GPIN2** 0x2
- #define **API_EXP_GPINOUT1** 0x4
- #define **API_EXP_GPINOUT2** 0x8
- #define **API_SIGNATECH_I** 0x01
- #define **API_SIGNATECH_II** 0x02
- #define **API_AI_VIN_SWMON** 0x6F
- #define **API_AI_V_HV_STRINGMON** 0x8C
- #define **API_AI_V_LV_STRINGMON** 0xFF
- #define **API_AI_VCH1MON** 0xFF
- #define **API_AI_VCH2MON** 0xFF
- #define **API_AI_VCH3MON** 0xFF
- #define **API_AI_VCH4MON** 0xFF
- #define **API_AI_GND1** 0x74
- #define **API_AI_SIG1_ID1_MON** 0xFF
- #define **API_AI_SIG1_ID2_MON** 0xFF
- #define **API_AI_CH1_DRV_LVL** 0xFF
- #define **API_AI_CH2_DRV_LVL** 0xFF
- #define **API_AI_CH3_DRV_LVL** 0xFF
- #define **API_AI_CH4_DRV_LVL** 0xFF
- #define **API_AI_GND2** 0x75
- #define **API_AI_10VMON** 0xFF
- #define **API_AI_CH1CURRSENS** 0xA0
- #define **API_AI_CH2CURRSENS** 0xA1
- #define **API_AI_CH3CURRSENS** 0xFF
- #define **API_AI_CH4CURRSENS** 0xFF
- #define **API_AI_INT_TEMP** 0xFF
- #define **API_AI_GND3** 0xFF
- #define **API_AI_33VMON** 0xFF
- #define **API_AI_5VMON** 0x71
- #define **API_AI_15VMON** 0x70
- #define **API_AI_15VSWMON** 0x72
- #define **API_AI_USB5VMON** 0x73
- #define **API_AI_CH1INTTEMP** 0x79
- #define **API_AI_CH2INTTEMP** 0x65

Typedefs

- typedef unsigned char **UINT8_T**
- typedef unsigned short **UINT16_T**
- typedef unsigned int **UINT32_T**
- typedef char **INT8_T**
- typedef short **INT16_T**
- typedef int **INT32_T**
- typedef unsigned char **BOOLEAN_T**
- typedef float **FLOAT32_T**
- typedef INT16_T **ERRCODE_T**

  API Error Code type – error codes documented in separate "ecodes.ini" file.

Enumerations

- enum API_MODE_T {
  API_MODE_PULSED , API_MODE_REPEAT , API_MODE_TIMING_BYPASS , API_MODE_INTERNAL_TRIGGER
  ,
  API_NUM_MODE }

  Define possible operating modes.
- enum API_CHAN_T { **API_CHAN_1** = 1 , **API_CHAN_2** }

  Define the drive channel identifiers used with the API interface.
- enum { API_TRIG_NONE , API_TRIG_1 , API_TRIG_2 }

  Define the trigger input identifiers used with the API interface.
- enum {
  API_POWER_TEST , **API_RAM_TEST** , **API_FLASH_TEST** , **API_EXTIO_TEST** ,
  API_EEPROM_TEST , **API_EXT_DAC_TEST** , **API_SCI_TEST** , **API_NUM_TESTS** }

  Define identifiers for diagnostics tests.

Functions

- UINT16_T api_get_dll_version (void)

  Returns the current version number of this software.
- UINT32_T api_get_dll_full_version (void)

  Returns the full version number of this software.
- ERRCODE_T api_get_error_name (ERRCODE_T ecode, char ∗nameptr, UINT16_T namelen)

  Returns the name of the error associated with the code.
- ERRCODE_T api_get_error_text (ERRCODE_T ecode, char ∗textptr, UINT16_T textlen)

  Returns the text description of the error associated with the code.
- ERRCODE_T api_init (UINT8_T port, const INT8_T ∗product_name, const INT8_T ∗serial_num)

  Initializes the connection to a specific Pulsar.
- ERRCODE_T api_connect (UINT8_T port, const INT8_T ∗product_name, const INT8_T ∗ip_addr)

  Initializes the connection to a specific Pulsar.
- ERRCODE_T api_get_pulsar_version (UINT8_T port, INT8_T ∗fw_version, INT8_T ∗hw_version, INT8_↩
  T ∗serial_num)

  Reads firmware version, hardware version, and serial number Each string can be up to 32 characters long.
- ERRCODE_T api_get_pulsar_ipaddr (UINT8_T port, INT8_T ∗ip_addr, INT8_T ∗subnet, INT8_T ∗gateway, INT8_T ∗mac_addr)

  Reads iip configuration settings from the device Each string can be up to 32 characters long.
- ERRCODE_T api_get_status (UINT8_T port)

  Reads status information from the device.
- ERRCODE_T api_run_diags (UINT8_T port, UINT8_T test_num)

  Runs one or all of the built in diagnostics.
- ERRCODE_T api_get_channel_config (UINT8_T port, UINT8_T chan, UINT8_T type, API_CHAN_CONFIG_STRUCT_T ∗config_data)

  Gets the configuration data for one drive channel.
- ERRCODE_T api_set_channel_config (UINT8_T port, UINT8_T chan, UINT8_T type, API_CHAN_CONFIG_STRUCT_T ∗config_data)

  Sets the configuration data for one drive channel.
- ERRCODE_T api_set_channel_config_nochecks (UINT8_T port, UINT8_T chan, UINT8_T type, API_CHAN_CONFIG_STRUCT_T ∗config_data)

Sets the configuration data for one drive channel.

- ERRCODE_T api_trigger (UINT8_T port, UINT8_T trig)

  Generates a software trigger on the selected trigger channel.
- ERRCODE_T api_calc_led_temp (UINT8_T port, UINT8_T ch, UINT32_T width, FLOAT32_T current, FLOAT32_T *temp)

  Calculates the expected LED junction temperature based on the information stored in the lighthead data structure and the width and current information passed.
- ERRCODE_T api_calc_max_width (UINT8_T port, UINT8_T ch, FLOAT32_T current, UINT32_T *width)

  Calculates the maximum pulse width for the selected channel based on the information stored in the lighthead data structure and the current information passed.
- ERRCODE_T api_calc_max_current (UINT8_T port, UINT8_T ch, UINT32_T width, FLOAT32_T *current)

  Calculates the maximum pulse current for the selected channel based on the information stored in the lighthead data structure and the width information passed.
- ERRCODE_T api_calc_supply_voltage (UINT8_T port, UINT8_T ch, UINT32_T width, FLOAT32_T current, FLOAT32_T *voltage)

  Calculates the required supply voltage for a given width and current.
- ERRCODE_T api_calc_max_freq (UINT8_T port, UINT8_T ch, UINT32_T width, FLOAT32_T current, UINT16_T *freq)

  Calculates the maximum frequency allowed for a given width and current.
- ERRCODE_T api_measure_drive_current (UINT8_T port, UINT8_T ch, FLOAT32_T *current)

  Measures the drive current of the selected drive channel.
- ERRCODE_T api_measure_voltage (UINT8_T port, UINT8_T id, FLOAT32_T *voltage)

  Measures the voltage of one of the signals monitored by the Pulsar.
- ERRCODE_T api_get_user_data (UINT8_T port, UINT16_T *rotary, UINT8_T *dip, UINT8_T *trigger, UINT8_T *connect, UINT8_T *supply, UINT8_T *expansion)

  Gathers various information about the operating state of the Pulsar.
- ERRCODE_T api_get_light_data (UINT8_T port, UINT8_T *sig_model, UINT8_T *sig1_id, char *sig2_pn, char *sig2_sn)

  Gathers information regarding the lighthead type.
- ERRCODE_T api_get_light_data2 (UINT8_T port, UINT8_T channel, char *sig2_pn)

  Get Signatech II lighthead part and serial number.
- ERRCODE_T api_set_stored_data (UINT8_T port, UINT8_T *data, UINT8_T len)

  Saves a block of Ai defined data to EEPROM.
- ERRCODE_T api_get_stored_data (UINT8_T port, UINT8_T *data, UINT8_T len)

  Reads a block of Ai defined data from EEPROM.
- ERRCODE_T api_shutdown (UINT8_T port)

  Shuts down the connection to a specific Pulsar.
- ERRCODE_T api_set_configuration (UINT8_T port, UINT8_T *slew_rate, UINT8_T *diff_trigger, UINT8_T limit48v, UINT8_T disable_prot)

  Set hardware configuration.
- ERRCODE_T api_get_configuration (UINT8_T port, UINT8_T *pslew_rate, UINT8_T *pdiff_trigger, UINT8_T *limit48v, UINT8_T *disable_prot)

  Get hardware configuration.
- ERRCODE_T api_set_pulsar_ipaddr (UINT8_T port, const char *ip_addr, const char *subnet, const char *gateway)

  Set Pulsar IP address.
- ERRCODE_T api_get_errors (UINT8_T port, INT16_T *err, UINT8_T *totalErrors)

  Reads the oldest posted error.
- int api_connected_device (UINT8_T port)

Gets the type of device on the given port.

- ERRCODE_T **api_get_mac** (UINT8_T port, UINT8_T *mac)
- ERRCODE_T **api_set_mac** (UINT8_T port, UINT8_T *mac)
- ERRCODE_T **api_get_manufacturing_mac** (UINT8_T port, UINT8_T *mac)
- ERRCODE_T api_get_pulsar_addrs (UINT32_T *addrs, UINT16_T max_addrs, UINT16_T *num_found)

  Find pulsar devices on all attached network adapters.

6.1.1 Detailed Description

This module contains external definitions for the High Level User Application Programming Interface DLL functions in the Pulsar320 and Pulsar320E projects.

Copyright

COPYRIGHT © 2022, Advanced Illumination, Inc.

Contains confidential and proprietary information which may not be copied, disclosed or used by others except as expressly authorized in writing by Advanced Illumination.

External definitions for the High Level User Application Programming Interface

Written for:
Advanced Illumination, Inc.
440 State Garage Road
Rochester, VT 05767
http://advancedillumination.com

Contacts:
John Thrailkill
Phone: (802) 767-3830
FAX: (802) 767-3831
jthrailkill@advancedillumination.com

Written by:
Rob Macklin
Microprocessor Designs, Inc.
65 Longmeadow Drive
PO Box 160
Shelburne, VT 05482
http://www.updesigns.com

Revision

4

6.1.2 Enumeration Type Documentation

6.1.2.1 anonymous enum

```
anonymous enum
```

Define the trigger input identifiers used with the API interface.

Enumerator

API_TRIG_NONE	No trigger input.
API_TRIG_1	Trigger input 1.
API_TRIG_2	Trigger input 2.

6.1.2.2 API_MODE_T

```
enum API_MODE_T
```

Define possible operating modes.

Enumerator

API_MODE_PULSED	Pulse mode.
API_MODE_REPEAT	Repeat mode, same as pulsed in 320E.
API_MODE_TIMING_BYPASS	Used for Continuous mode (with 1,000,000us pulsewidth)
API_MODE_INTERNAL_TRIGGER	Used for test mode (10Hz internal trigger)
API_NUM_MODE	Number of enum entries (do not use as mode)

6.1.3 Function Documentation

6.1.3.1 api_calc_led_temp()

```
ERRCODE_T api_calc_led_temp (
            UINT8_T port,
            UINT8_T ch,
            UINT32_T width,
            FLOAT32_T current,
            FLOAT32_T * temp )
```

Calculates the expected LED junction temperature based on the information stored in the lighthead data structure and the width and current information passed.

Parameters

	port	Identifier for device
	ch	Channel to compute LED temperature
	width	Pulse width in microseconds
	current	Pulse current in amps
out	*temp*	Location to store calculated temperature

Returns

Zero if successful, error code otherwise

6.1.3.2 api_calc_max_current()

```
ERRCODE_T api_calc_max_current (
            UINT8_T port,
            UINT8_T ch,
            UINT32_T width,
            FLOAT32_T * current )
```

Calculates the maximum pulse current for the selected channel based on the information stored in the lighthead data structure and the width information passed.

Parameters

	port	Identifier for device
	ch	Channel to compute LED temperature
	width	Pulse width in microseconds
out	*current*	Maximum pulse current in amps

Returns

Zero if successful, error code otherwise

6.1.3.3 api_calc_max_freq()

```
ERRCODE_T api_calc_max_freq (
            UINT8_T port,
            UINT8_T ch,
            UINT32_T width,
            FLOAT32_T current,
            UINT16_T * freq )
```

Calculates the maximum frequency allowed for a given width and current.

This function computes maximum frequency based on the maximum power of the power supply and also from the LED cooling equations. The lower value is returned.

Parameters

	port	Identifier for device
	ch	Channel to compute maximum frequency
	width	Pulse width in microseconds
	current	Pulse current in amps
out	*freq*	Location to store calculated frequency

Returns

Zero if successful, error code otherwise

6.1.3.4 api_calc_max_width()

```
ERRCODE_T api_calc_max_width (
            UINT8_T port,
            UINT8_T ch,
            FLOAT32_T current,
            UINT32_T * width )
```

Calculates the maximum pulse width for the selected channel based on the information stored in the lighthead data structure and the current information passed.

Parameters

	port	Identifier for device
	ch	Channel to compute LED temperature
	current	Pulse current in amps
out	*width*	Maximum pulse width in microseconds

Returns

Zero if successful, error code otherwise

6.1.3.5 api_calc_supply_voltage()

```
ERRCODE_T api_calc_supply_voltage (
            UINT8_T port,
            UINT8_T ch,
            UINT32_T width,
            FLOAT32_T current,
            FLOAT32_T * voltage )
```

Calculates the required supply voltage for a given width and current.

Parameters

	port	Identifier for device
	ch	Channel to compute required supply voltage
	width	Pulse width in microseconds
	current	Pulse current in amps
out	*voltage*	Location to store calculated voltage

Returns

Zero if successful, error code otherwise

6.1.3.6 api_connect()

```
ERRCODE_T api_connect (
            UINT8_T port,
            const INT8_T * product_name,
            const INT8_T * ip_addr )
```

Initializes the connection to a specific Pulsar.

See also

api_init for USB connections

Parameters

port	Identifier for device
product_name	Name of USB device to open
ip_addr	IP address of device to open

Returns

Zero if successful, error code otherwise

6.1.3.7 api_connected_device()

```
int api_connected_device (
            UINT8_T port )
```

Gets the type of device on the given port.

Parameters

port	Communication port of the device

Returns

Zero if none connected, 1 for Pulsar 320, 2 for Pulsar 320E

6.1.3.8 api_get_channel_config()

```
ERRCODE_T api_get_channel_config (
            UINT8_T port,
            UINT8_T chan,
            UINT8_T type,
            API_CHAN_CONFIG_STRUCT_T * config_data )
```

Gets the configuration data for one drive channel.

Parameters

	port	Identifier for device
	chan	Identifier for drive channel
	type	Identifier for which data to get
out	*config_data*	Active channel configuration data

Returns

Zero if successful, error code otherwise

6.1.3.9 api_get_configuration()

```
ERRCODE_T api_get_configuration (
            UINT8_T port,
            UINT8_T * pslew_rate,
            UINT8_T * pdiff_trigger,
            UINT8_T * limit48v,
            UINT8_T * disable_prot )
```

Get hardware configuration.

Parameters

	port	Communication port to use
out	*slew_rate*	Slew rate (array, one for each channel)
out	*diff_trigger*	Differential trigger (0 or 1) (array, one for each channel)
out	*limit48v*	Limit output to 48v
out	*disable_prot*	Disable protection

Returns

Zero if successful, error code otherwise

6.1.3.10 api_get_dll_full_version()

```
UINT32_T api_get_dll_full_version (
            void  )
```

Returns the full version number of this software.

Each byte in the integer encodes a different part of the version number

- Byte 0 (highest): MAJOR
- Byte 1: MINOR
- Byte 2: PATCH
- Byte 3: Build (usually 0)

Returns

Big-endian unsigned integer encoding the version number

6.1.3.11 api_get_dll_version()

```
UINT16_T api_get_dll_version (
            void  )
```

Returns the current version number of this software.

Returns

High byte is major version number, Low byte is minor version number

6.1.3.12 api_get_error_name()

```
ERRCODE_T api_get_error_name (
            ERRCODE_T ecode,
            char * nameptr,
            UINT16_T namelen )
```

Returns the name of the error associated with the code.

Parameters

in	*ecode*	Code to be looked up
out	*nameptr*	Location to place name string
in	*namelen*	Maximum length of name string

Returns

Zero if successful, error code otherwise

6.1.3.13 api_get_error_text()

```
ERRCODE_T api_get_error_text (
            ERRCODE_T ecode,
            char * textptr,
            UINT16_T textlen )
```

Returns the text description of the error associated with the code.

Parameters

	ecode	Code to be looked up
out	*textptr*	Location to place text description string
	textlen	Maximum length of text description string

Returns

Zero if successful, error code otherwise

6.1.3.14 api_get_errors()

```
ERRCODE_T api_get_errors (
            UINT8_T port,
            INT16_T * err,
            UINT8_T * totalErrors )
```

Reads the oldest posted error.

Parameters

	port	Communication port to use
out	*err*	The oldest posted error in the device
out	*totalErrors*	The number of errors remaining to read

Returns

PULSAR_API

6.1.3.15 api_get_light_data()

```
ERRCODE_T api_get_light_data (
            UINT8_T port,
            UINT8_T * sig_model,
            UINT8_T * sig1_id,
            char * sig2_pn,
            char * sig2_sn )
```

Gathers information regarding the lighthead type.

Parameters

	port	Identifier for device
out	*sig_model*	1 - Signatech I, 2 - Signatech II, 0 - Unknown
out	*sig1_id*	Set for Signatech I lightheads
out	*sig2_pn*	Set to part number string for Signatech II
out	*sig2_sn*	Set to serial number of Signatech 2 lighthead

Returns

Zero if successful, error code otherwise

6.1.3.16 api_get_light_data2()

```
ERRCODE_T api_get_light_data2 (
            UINT8_T port,
            UINT8_T channel,
            char * sig2_pn )
```

Get Signatech II lighthead part and serial number.

Parameters

	port	Identifier for device
	channel	Output channel to read (only valid for Pulsar 320E)
out	*sig2_pn*	Part number of Signatech 2 lighthead

Returns

Zero if successful, error code otherwise.

6.1.3.17 api_get_pulsar_addrs()

```
ERRCODE_T api_get_pulsar_addrs (
            UINT32_T * addrs,
            UINT16_T max_addrs,
            UINT16_T * num_found )
```

Find pulsar devices on all attached network adapters.

Parameters

addrs	Array of UINT32_Ts to fill with found IP addresses (in network byte order).
max_addrs	Maximum number of addresses to find (size of the array)
num_found	Actual number of addresses found

Returns

Zero if successful, error code otherwise

6.1.3.18 api_get_pulsar_ipaddr()

```
ERRCODE_T api_get_pulsar_ipaddr (
            UINT8_T port,
            INT8_T * ip_addr,
            INT8_T * subnet,
            INT8_T * gateway,
            INT8_T * mac_addr )
```

Reads iip configuration settings from the device Each string can be up to 32 characters long.

Get Pulsar IP address.

Parameters

in	*port*	Identifier for device
out	*ip_addr*	IP Address
out	*subnet*	Subnet Mask
out	*gateway*	Gateway
out	*mac_addr*	Hardware MAC address

Returns

Zero if successful, error code otherwise

Parameters

	port	Communication port to use
out	*ip_addr*	IP address
out	*subnet*	Subnet mask
out	*gateway*	Gateway address
out	*mac_addr*	MAC address

Returns

Zero if successful, error code otherwise

6.1.3.19 api_get_pulsar_version()

```
ERRCODE_T api_get_pulsar_version (
            UINT8_T port,
            INT8_T * fw_version,
            INT8_T * hw_version,
            INT8_T * serial_num )
```

Reads firmware version, hardware version, and serial number Each string can be up to 32 characters long.

Parameters

in	*port*	Identifier for device
out	*fw_version*	Firmware version
out	*hw_version*	Hardware version
out	*serial_num*	Serial number

Returns

Zero if successful, error code otherwise

6.1.3.20 api_get_status()

```
ERRCODE_T api_get_status (
            UINT8_T port )
```

Reads status information from the device.

Parameters

port	Identifier for device

Returns

Zero if successful, error code otherwise

6.1.3.21 api_get_stored_data()

```
ERRCODE_T api_get_stored_data (
            UINT8_T port,
            UINT8_T * data,
            UINT8_T len )
```

Reads a block of Ai defined data from EEPROM.

Parameters

	port	Identifier for device
	len	Length of data block to read
out	*data*	Pointer to data block to be written

Returns

Zero if successful, error code otherwise

6.1.3.22 api_get_user_data()

```
ERRCODE_T api_get_user_data (
            UINT8_T port,
            UINT16_T * rotary,
            UINT8_T * dip,
            UINT8_T * trigger,
            UINT8_T * connect,
            UINT8_T * supply,
            UINT8_T * expansion )
```

Gathers various information about the operating state of the Pulsar.

Parameters

	port	Identifier for device
out	*rotary*	Value on rotary switches
out	*dip*	Value on DIP switches
out	*trigger*	State of trigger inputs
out	*connect*	Communication connection status
out	*supply*	Power supply status
out	*expansion*	Expansion status

Returns

Zero if successful, error code otherwise

Note

A NULL can be passed for any data values not needed

6.1.3.23 api_init()

```
ERRCODE_T api_init (
            UINT8_T port,
            const INT8_T * product_name,
            const INT8_T * serial_num )
```

Initializes the connection to a specific Pulsar.

Warning

FOR USB CONNECTIONS ONLY - SEE api_connect FOR TCP/IP CONNECTIONS

See also

api_connect

Parameters

port	Identifier for device
product_name	Name of USB device to open
serial_num	Serial number of device to open (or NULL)

Returns

Zero if successful, error code otherwise

6.1.3.24 api_measure_drive_current()

```
ERRCODE_T api_measure_drive_current (
            UINT8_T port,
            UINT8_T ch,
            FLOAT32_T * current )
```

Measures the drive current of the selected drive channel.

This function can be used in either pulse or continuous modes.

Parameters

	port	Identifier for device
	ch	Channel to measure pulse current
out	*current*	Location to store measured pulse current

Returns

Zero if successful, error code otherwise

6.1.3.25 api_measure_voltage()

```
ERRCODE_T api_measure_voltage (
            UINT8_T port,
            UINT8_T id,
            FLOAT32_T * voltage )
```

Measures the voltage of one of the signals monitored by the Pulsar.

Parameters

	port	Identifier for device
	id	Identifier for desired voltage to measure
out	*voltage*	Location to store measured voltage

Returns

Zero if successful, error code otherwise

6.1.3.26 api_run_diags()

```
ERRCODE_T api_run_diags (
            UINT8_T port,
            UINT8_T test_num )
```

Runs one or all of the built in diagnostics.

Parameters

port	Identifier for device
test_num	Desired test number or FF for all

Returns

Zero if successful, error code otherwise

6.1.3.27 api_set_channel_config()

```
ERRCODE_T api_set_channel_config (
            UINT8_T port,
            UINT8_T chan,
            UINT8_T type,
            API_CHAN_CONFIG_STRUCT_T * config_data )
```

Sets the configuration data for one drive channel.

Parameters

port	Identifier for device
chan	Identifier for drive channel
type	Identifier for which data to get
API_CHAN_CONFIG_STRUCT_T	* config_data New channel configuration data

Returns

Zero if successful, error code otherwise

6.1.3.28 api_set_channel_config_nochecks()

```
ERRCODE_T api_set_channel_config_nochecks (
            UINT8_T port,
            UINT8_T chan,
            UINT8_T type,
            API_CHAN_CONFIG_STRUCT_T * config_data )
```

Sets the configuration data for one drive channel.

Warning

Unlike api_set_channel_config API call, the "nochecks" function does not verify the settings. This could result it improper channel settings or configuration that could break hardware or blow flash bulbs.

Parameters

port	Identifier for device
chan	Identifier for drive channel
type	Identifier for which data to get
API_CHAN_CONFIG_STRUCT_T	* config_data New channel configuration data

Returns

Zero if successful, error code otherwise

6.1.3.29 api_set_configuration()

```
ERRCODE_T api_set_configuration (
            UINT8_T port,
            UINT8_T * slew_rate,
            UINT8_T * diff_trigger,
            UINT8_T limit48v,
            UINT8_T disable_prot )
```

Set hardware configuration.

Parameters

	port	Communication port to use
in	*slew_rate*	Slew rate (array, one for each channel)
in	*diff_trigger*	Differential trigger (0 or 1) (array, one for each channel)
in	*limit48v*	Limit output to 48v
in	*disable_prot*	Disable protection

Returns

Zero if successful, error code otherwise

6.1.3.30 api_set_pulsar_ipaddr()

```
ERRCODE_T api_set_pulsar_ipaddr (
            UINT8_T port,
            const char * ip_addr,
            const char * subnet,
            const char * gateway )
```

Set Pulsar IP address.

Parameters

	port	Communication port to use
in	*ip_addr*	IP address
in	*subnet*	Subnet mask
in	*gateway*	Gateway address

Returns

Zero if successful, error code otherwise

6.1.3.31 api_set_stored_data()

```
ERRCODE_T api_set_stored_data (
            UINT8_T port,
            UINT8_T * data,
            UINT8_T len )
```

Saves a block of Ai defined data to EEPROM.

Parameters

port	Identifier for device
data	Pointer to data block to be written
len	Length of data block to be written

Returns

Zero if successful, error code otherwise

6.1.3.32 api_shutdown()

```
ERRCODE_T api_shutdown (
            UINT8_T port )
```

Shuts down the connection to a specific Pulsar.

Parameters

port	Communication port to use

Returns

Zero if successful, error code otherwise

6.1.3.33 api_trigger()

```
ERRCODE_T api_trigger (
            UINT8_T port,
            UINT8_T trig )
```

Generates a software trigger on the selected trigger channel.

Parameters

port	Identifier for device
trig	Trigger to activate

Returns

Zero if successful, error code otherwise

6.2 PulsarAPI.h

Go to the documentation of this file.

```
1
```

```
36 #ifdef OLD_LOG
81 #endif
82
83
84 #ifndef _PULSARAPI_H
85 #define _PULSARAPI_H
86 #ifdef _WIN32
87 #include <wtypes.h>
88 #include <oleauto.h>
89 #endif
90
91 // The following ifdef block is the standard way of creating macros which make exporting
92 // from a DLL simpler. All files within this DLL are compiled with the PULSARAPI_EXPORTS
93 // symbol defined on the command line. this symbol should not be defined on any project
94 // that uses this DLL. This way any other project whose source files include this file see
95 // PULSAR_API functions CALLSPEC as being imported from a DLL, wheras this DLL sees symbols
96 // defined with this macro as being exported.  If a non-Microsoft compiler is used the
97 // PULSAR_API CALLSPEC is defined as a blank and no modifications are made to the function interfaces.
98 #ifdef _MSC_VER
99    #define CALLSPEC __stdcall
100    #ifdef PULSARAPI_EXPORTS
101    #define PULSAR_API extern "C" __declspec(dllexport)
102    #else
103    #define PULSAR_API extern "C" __declspec(dllimport)
104    #endif
105 #else
106    #define CALLSPEC
107    #define PULSAR_API extern "C"
108 #endif
109
110 /* Define name of USB device */
111 #define API_PULSAR_PRODUCT_NAME            "Ai Pulsar 320 Controller"
112
113 /* MPD Standard data type definitions */
114 #ifndef MPD_TYPES
115 #define MPD_TYPES
116 typedef unsigned char  UINT8_T;
117 typedef unsigned short UINT16_T;
118 typedef unsigned int   UINT32_T;
119 typedef char           INT8_T;
120 typedef short          INT16_T;
121 typedef int            INT32_T;
122 typedef unsigned char  BOOLEAN_T;
123 typedef float          FLOAT32_T;
128 typedef INT16_T        ERRCODE_T;
129 #endif
130
135 typedef struct
136 {
138    UINT8_T   trigger;
140    UINT8_T   rising;
143    UINT8_T   mode;
144    UINT8_T   reserved_byte;
146    UINT32_T  repeat_cnt;
148    UINT16_T  delay;
150    FLOAT32_T current;
152    UINT32_T  width;
154    UINT32_T  offtime;
155 }  API_CHAN_CONFIG_STRUCT_T;              /* 20 bytes */
156
157
161 enum API_MODE_T
162 {
164    API_MODE_PULSED,
166    API_MODE_REPEAT,
168    API_MODE_TIMING_BYPASS,
170    API_MODE_INTERNAL_TRIGGER,
172    API_NUM_MODE
173 };
174
176 enum API_CHAN_T
177 {
178    //Channel 1
179    API_CHAN_1 = 1,
180    //Channel 2
181    API_CHAN_2,
182 };
183
184 #ifdef KKP
185 #define API_NUM_CHAN  2
186 #else
```

```
187 #define API_NUM_CHAN  2
188 #endif
189
193 enum
194 {
196    API_TRIG_NONE,
198    API_TRIG_1,
200    API_TRIG_2,
201 };
202
203 #ifdef KKP
204 #define API_NUM_TRIG  2
205 #else
206 #define API_NUM_TRIG  2
207 #endif
208
209 /* Define identifiers used to handle how channel configuration data is used */
210 #define API_CHAN_CONFIG_DEFAULT        1
211 #define API_CHAN_CONFIG_ACTIVE         2
212
214 enum
215 {
216    API_POWER_TEST,
217    API_RAM_TEST,
218    API_FLASH_TEST,
219    API_EXTIO_TEST,
220    API_EEPROM_TEST,
221    API_EXT_DAC_TEST,
222    API_SCI_TEST,
223    API_NUM_TESTS
224 };
225
226 /* Optional diagnostics tests do not normally run on startup of the Pulsar */
227 /* They require additional hardware beyond the PCB Assembly itself to complete */
228 /* The optional SCI test requires a serial loopback connector between the external */
229 /* modular connector and the internal 10-pin header */
230 #define API_NUM_OPTIONAL_TESTS      1
231
232 /* Define bit identifiers for the information returned by the api_get_user_data command */
233
234 /* The rotary switch value is returned within a range of 0 to API_MAX_ROTARY */
235 #define API_MAX_ROTARY     999
236
237 /* Each bit is associated with one DIP switch position */
238 #define API_DIP_MODE0      0x1
239 #define API_DIP_MODE1      0x2
240 #define API_DIP_MODE2      0x4
241 #define API_DIP_MODE3      0x8
242
243 /* Each bit is associated with one TRIG input */
244 #define API_TRIG_IN1       0x1
245 #define API_TRIG_IN2       0x2
246
247 /* Connection state of lighthead, USB, and RS-232 */
248 #define API_CONN_LH_DETECT 0x1
249 #define API_CONN_USBDETECT 0x2
250 #define API_CONN_PCPRESENT 0x4
251 #define API_CONN_PC_READY  0x8
252
253 /* Operating state of power supplies */
254 #define API_SUPPLY_VIN_ON  0x1
255 #define API_SUPPLY_LV_ON   0x2
256 #define API_SUPPLY_HV_ON   0x4
257
258 /* Additional expansion general purpose inputs */
259 #define API_EXP_GPIN1      0x1
260 #define API_EXP_GPIN2      0x2
261 #define API_EXP_GPINOUT1   0x4
262 #define API_EXP_GPINOUT2   0x8
263
264 /* Define possible lighthead types returned by api_get_light_data command */
265 #define API_SIGNATECH_I    0x01
266 #define API_SIGNATECH_II   0x02
267
268 /* Define analog input identifiers used with api_measure_voltage command */
269 #define API_AI_VIN_SWMON      0x6F
270 #define API_AI_V_HV_STRINGMON 0x8C
271 #define API_AI_V_LV_STRINGMON 0xFF
272 #define API_AI_VCH1MON        0xFF
273 #define API_AI_VCH2MON        0xFF
274 #define API_AI_VCH3MON        0xFF
```

```
275 #define API_AI_VCH4MON        0xFF
276 #define API_AI_GND1           0x74
277 #define API_AI_SIG1_ID1_MON   0xFF
278 #define API_AI_SIG1_ID2_MON   0xFF
279 #define API_AI_CH1_DRV_LVL    0xFF
280 #define API_AI_CH2_DRV_LVL    0xFF
281 #define API_AI_CH3_DRV_LVL    0xFF
282 #define API_AI_CH4_DRV_LVL    0xFF
283 #define API_AI_GND2           0x75
284 #define API_AI_10VMON         0xFF
285 #define API_AI_CH1CURRSENS    0xA0
286 #define API_AI_CH2CURRSENS    0xA1
287 #define API_AI_CH3CURRSENS    0xFF
288 #define API_AI_CH4CURRSENS    0xFF
289 #define API_AI_INT_TEMP       0xFF
290 #define API_AI_GND3           0xFF
291 #define API_AI_33VMON         0xFF
292 #define API_AI_5VMON          0x71
293 //New values
294 #define API_AI_15VMON         0x70
295 #define API_AI_15VSWMON       0x72
296 #define API_AI_USB5VMON       0x73
297 #define API_AI_CH1INTTEMP     0x79
298 #define API_AI_CH2INTTEMP     0x65
299
300
307 PULSAR_API UINT16_T CALLSPEC api_get_dll_version(void);
308
322 PULSAR_API UINT32_T CALLSPEC api_get_dll_full_version(void);
323
334 PULSAR_API ERRCODE_T CALLSPEC api_get_error_name(ERRCODE_T ecode,char * nameptr,UINT16_T namelen);
335
345 PULSAR_API ERRCODE_T CALLSPEC api_get_error_text(ERRCODE_T ecode,char * textptr,UINT16_T textlen);
346
359 PULSAR_API ERRCODE_T CALLSPEC api_init(UINT8_T port, const INT8_T * product_name, const INT8_T *
    serial_num);
360
372 PULSAR_API ERRCODE_T CALLSPEC api_connect(UINT8_T port, const INT8_T * product_name, const INT8_T *
    ip_addr);
373
386 PULSAR_API ERRCODE_T CALLSPEC api_get_pulsar_version(UINT8_T port, INT8_T * fw_version, INT8_T * hw_version,
    INT8_T * serial_num);
387
401 PULSAR_API ERRCODE_T CALLSPEC api_get_pulsar_ipaddr(UINT8_T port, INT8_T * ip_addr, INT8_T * subnet, INT8_T
    * gateway, INT8_T * mac_addr);
402
410 PULSAR_API ERRCODE_T CALLSPEC api_get_status(UINT8_T port);
411
420 PULSAR_API ERRCODE_T CALLSPEC api_run_diags(UINT8_T port, UINT8_T test_num);
421
422
434 PULSAR_API ERRCODE_T CALLSPEC api_get_channel_config(UINT8_T port, UINT8_T chan, UINT8_T type,
435                                                      API_CHAN_CONFIG_STRUCT_T *  config_data);
436
448 PULSAR_API ERRCODE_T CALLSPEC api_set_channel_config(UINT8_T port, UINT8_T chan, UINT8_T type,
449                                                      API_CHAN_CONFIG_STRUCT_T *  config_data);
450
451
468 PULSAR_API ERRCODE_T CALLSPEC api_set_channel_config_nochecks(UINT8_T port, UINT8_T chan, UINT8_T type,
469                                                      API_CHAN_CONFIG_STRUCT_T *  config_data);
470
471
480 PULSAR_API ERRCODE_T CALLSPEC api_trigger(UINT8_T port, UINT8_T trig);
481
482
497 PULSAR_API ERRCODE_T CALLSPEC api_calc_led_temp(UINT8_T port, UINT8_T ch,
498                                         UINT32_T width, FLOAT32_T current, FLOAT32_T *temp);
499
513 PULSAR_API ERRCODE_T CALLSPEC api_calc_max_width(UINT8_T port, UINT8_T ch,
514                                          FLOAT32_T current, UINT32_T * width);
515
529 PULSAR_API ERRCODE_T CALLSPEC api_calc_max_current(UINT8_T port, UINT8_T ch,
530                                            UINT32_T width, FLOAT32_T *current);
531
544 PULSAR_API ERRCODE_T CALLSPEC api_calc_supply_voltage(UINT8_T port, UINT8_T ch,
545                                                       UINT32_T width, FLOAT32_T current, FLOAT32_T *voltage);
546
562 PULSAR_API ERRCODE_T CALLSPEC api_calc_max_freq(UINT8_T port, UINT8_T ch,
563                                         UINT32_T width, FLOAT32_T current, UINT16_T *freq);
564
576 PULSAR_API ERRCODE_T CALLSPEC api_measure_drive_current(UINT8_T port, UINT8_T ch, FLOAT32_T *current);
```

```
577
588 PULSAR_API ERRCODE_T CALLSPEC api_measure_voltage(UINT8_T port, UINT8_T id, FLOAT32_T *voltage);
589
606 PULSAR_API ERRCODE_T CALLSPEC api_get_user_data(UINT8_T port,UINT16_T *rotary,UINT8_T *dip,UINT8_T *trigger,
607                                     UINT8_T *connect,UINT8_T *supply,UINT8_T *expansion);
608
621 PULSAR_API ERRCODE_T CALLSPEC api_get_light_data(UINT8_T port, UINT8_T *sig_model, UINT8_T *sig1_id, char
       *sig2_pn, char *sig2_sn);
622
631 PULSAR_API ERRCODE_T CALLSPEC api_get_light_data2(UINT8_T port, UINT8_T channel, char *sig2_pn);
632
642 PULSAR_API ERRCODE_T CALLSPEC api_set_stored_data(UINT8_T port, UINT8_T *data, UINT8_T len);
643
654 PULSAR_API ERRCODE_T CALLSPEC api_get_stored_data(UINT8_T port, UINT8_T *data, UINT8_T len);
655
656
657
665 PULSAR_API ERRCODE_T CALLSPEC api_shutdown(UINT8_T port);
666
677 PULSAR_API ERRCODE_T CALLSPEC api_set_configuration(UINT8_T port, UINT8_T* slew_rate, UINT8_T* diff_trigger,
       UINT8_T limit48v, UINT8_T disable_prot);
678
689 PULSAR_API ERRCODE_T CALLSPEC api_get_configuration(UINT8_T port, UINT8_T* pslew_rate, UINT8_T*
       pdiff_trigger, UINT8_T* limit48v, UINT8_T* disable_prot);
690
700 PULSAR_API ERRCODE_T CALLSPEC  api_set_pulsar_ipaddr(UINT8_T port, const char* ip_addr, const char* subnet,
       const char* gateway);
701
712 PULSAR_API ERRCODE_T CALLSPEC  api_get_pulsar_ipaddr(UINT8_T port, char* ip_addr, char* subnet, char*
       gateway, char* mac_addr);
713
722 PULSAR_API ERRCODE_T CALLSPEC api_get_errors(UINT8_T port, INT16_T* err, UINT8_T* totalErrors);
723
730 PULSAR_API int api_connected_device(UINT8_T port);
731
732 // @private
733 PULSAR_API ERRCODE_T CALLSPEC api_get_mac(UINT8_T port, UINT8_T* mac);
734 // @private
735 PULSAR_API ERRCODE_T CALLSPEC api_set_mac(UINT8_T port, UINT8_T* mac);
736 // @private
737 PULSAR_API ERRCODE_T CALLSPEC api_get_manufacturing_mac(UINT8_T port, UINT8_T* mac);
738
747 PULSAR_API ERRCODE_T CALLSPEC api_get_pulsar_addrs(UINT32_T* addrs, UINT16_T max_addrs, UINT16_T*
       num_found);
748
749 #if defined(_WIN32) && defined(USE_SAFEARRAY)
750 long __declspec (dllexport) __stdcall AddLongs_SafeArray(
751       SAFEARRAY **psaArray, // the array to use
752       long *plSum); // returns the sum of all elements of the array
753 #endif
754
755 #endif /* _PULARAPI_H */
```

Index