Talaria TWO™ Host API Reference Manual

Release: 04-04-2020

InnoPhase, Inc.
6815 Flanders Drive
San Diego, CA 92121
www.innophaseinc.com

Contents

1 Figures

2 Terms & Definitions

AES	Advanced Encryption Standard
A-MPDU	Aggregate MAC Protocol Data Unit
AP	Access Point
API	Application Programming Interface
BLE	Bluetooth Low Energy
BSD	Berkeley Software Distribution
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
EAP	Extensible Authentication Protocol
FAST	Flexible Authentication via Secure Tunneling
GCM	Galois/Counter Mode
GTC	Generic Token Card
HAPI	Host Application Processor Interface
HIO	Host Interface Operation
HTTP	Hypertext Transfer Protocol
ICMP	Internet Control Message Protocol
IoT	Internet of Things
IP	Internet Protocol
LEAP	Lightweight Extensible Authentication Protocol
MAC	Media Access Control
MQTT	Message Queuing Telemetry Transport
MS-CHAP	Microsoft version of the Challenge-Handshake Authentication Protocol

OS	Operating System
PEAP	Protected Extensible Authentication Protocol
PHY	Physical Layer
PSK	Pre Shared Key
PUF	Physically Unclonable Function
RC4	Rivest Cipher 4
RF	Radio Frequency
RTOS	Real Time Operating System
Rx	Receive
SHA1/2	Secure Hash Algorithm 1/2
SPI	Serial Peripheral Interface
SSID	Service Set Identifier
SSL	Secure Sockets Layer
T2	Talaria TWO
TCP	Transmission Control Protocol
TDES	Triple Data Encryption Algorithm
TLS	Transport Layer Security
TTLS	Tunneled Transport Layer Security
UART	Universal Asynchronous Receiver-Transmitter
UDP	User Datagram Protocol
WLAN	Wireless Local Area Network
WPA	Wireless Access Point
XEX	Ciphertext Stealing

3 Introduction

The InnoPhase Talaria TWO Multi-Protocol Platform is a highly integrated, single-chip wireless solution offering ultimate size, power, and cost advantages for a wide range of low-power IoT designs. The Talaria TWO system was designed for power efficiency and intelligent integration from the beginning for the unique demands of IoT applications.

4 Architecture

4.1 Overview

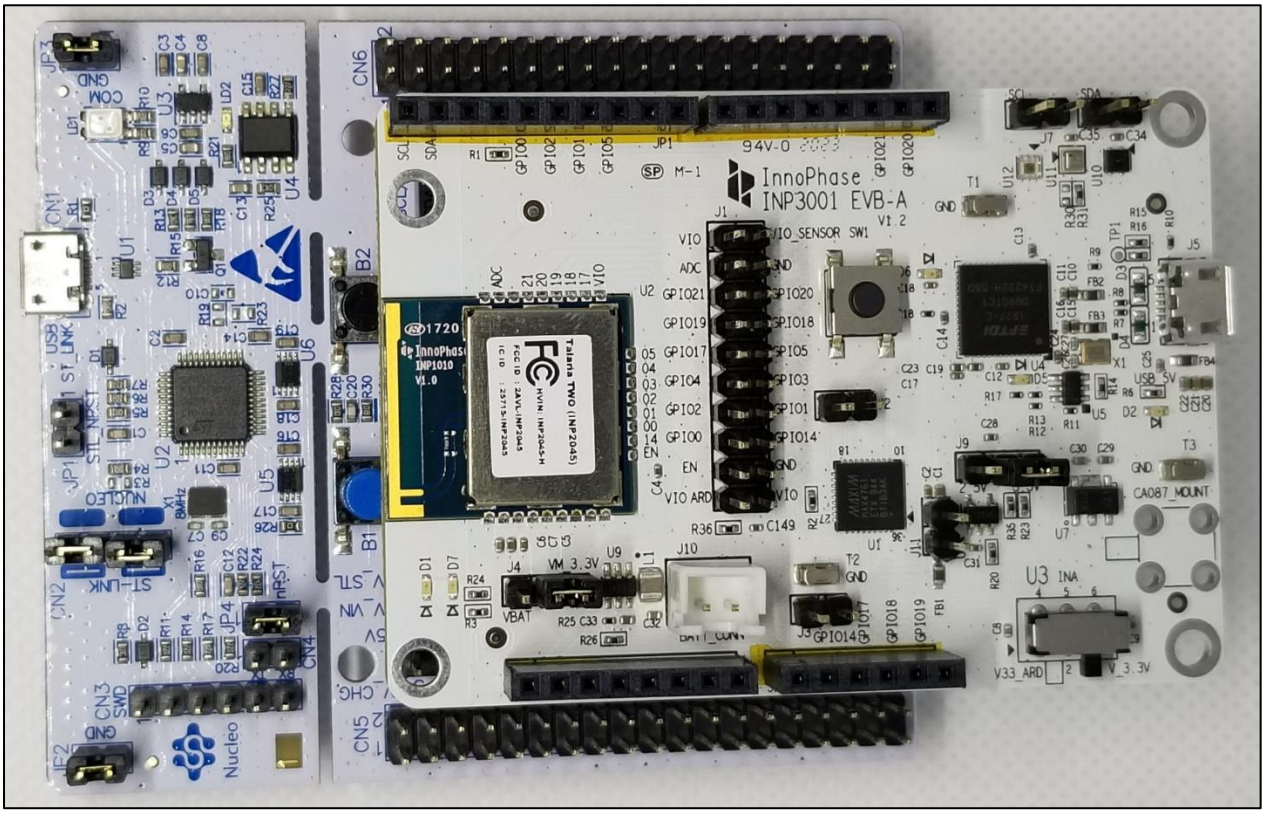

Figure 1: Talaria TWO Multi-Protocol Platform shown as a shield to STM32L433RC board

5 Talaria TWO System on Chip (SoC)

Talaria TWO performs the following based on commands from the Host processor.

1. Provides wireless (802.11b/g/n) link between the Host processor and AP or Hotspot
2. Scan and Connect to the AP specified by the Host
3. Performs WPA2 security handshake
4. Enables IP supports like TCP, UDP and DHCP
5. Adds network protocols like MQTT and HTTP
6. Supports transport protocols like SSL and TLS
7. Enables Serial encryption protocols
8. Provides BLE connectivity for provisioning

The major components in Talaria TWO are shown in Figure 2.

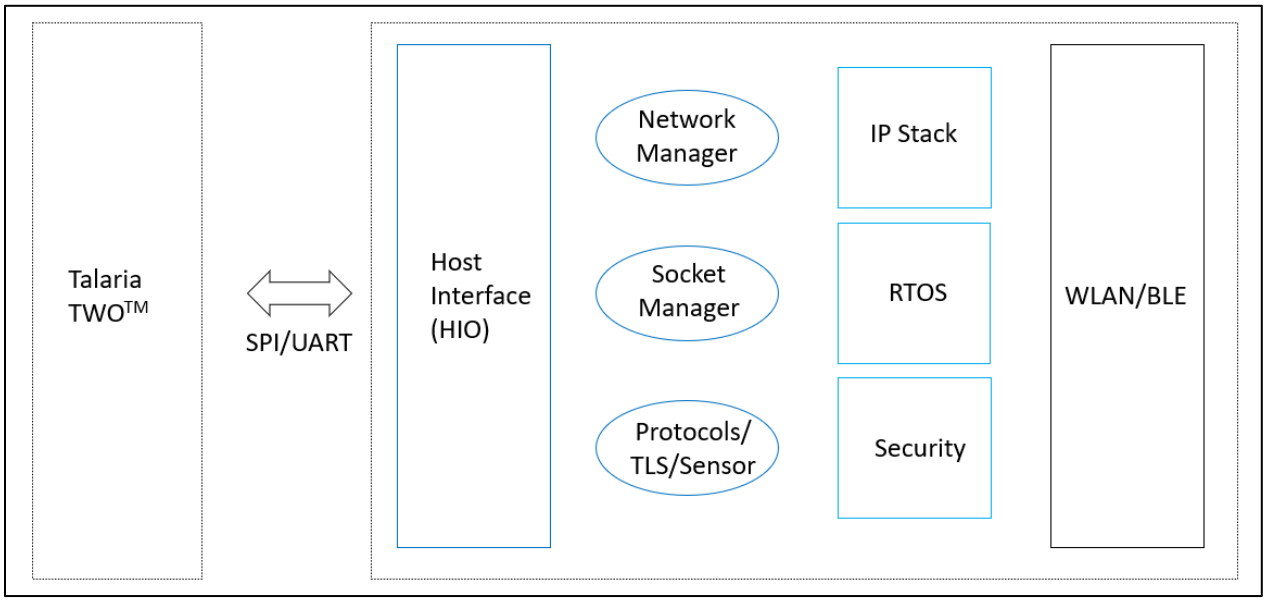

Figure 2: Major components in Talaria TWO

5.1 Wi-Fi Connection Manager

This is the network connection manager which handles all the Wi-Fi connection/disconnection.

5.2 Socket Manager

HIO handles all socket operations. It supports TCP, UDP, and raw sockets.

5.3 RTOS

Highly efficient, low footprint, real-time OS for low power applications.

5.4 IPSTACK

1. IPv4
2. ICMP
3. UDP
4. TCP
5. DHCP
6. DNS Resolver
7. BSD Sockets Interface
8. TLS
9. MQTT
10. IPv6

6 STM32 Host Processor

Host processor consists of the Host Application Processor (HAPI) Interface Layer and Host Applications. Host Applications may vary and will interact with Talaria TWO via APIs in the interface layer. HAPI provides APIs for Host Application to facilitate communication with the Talaria TWO.

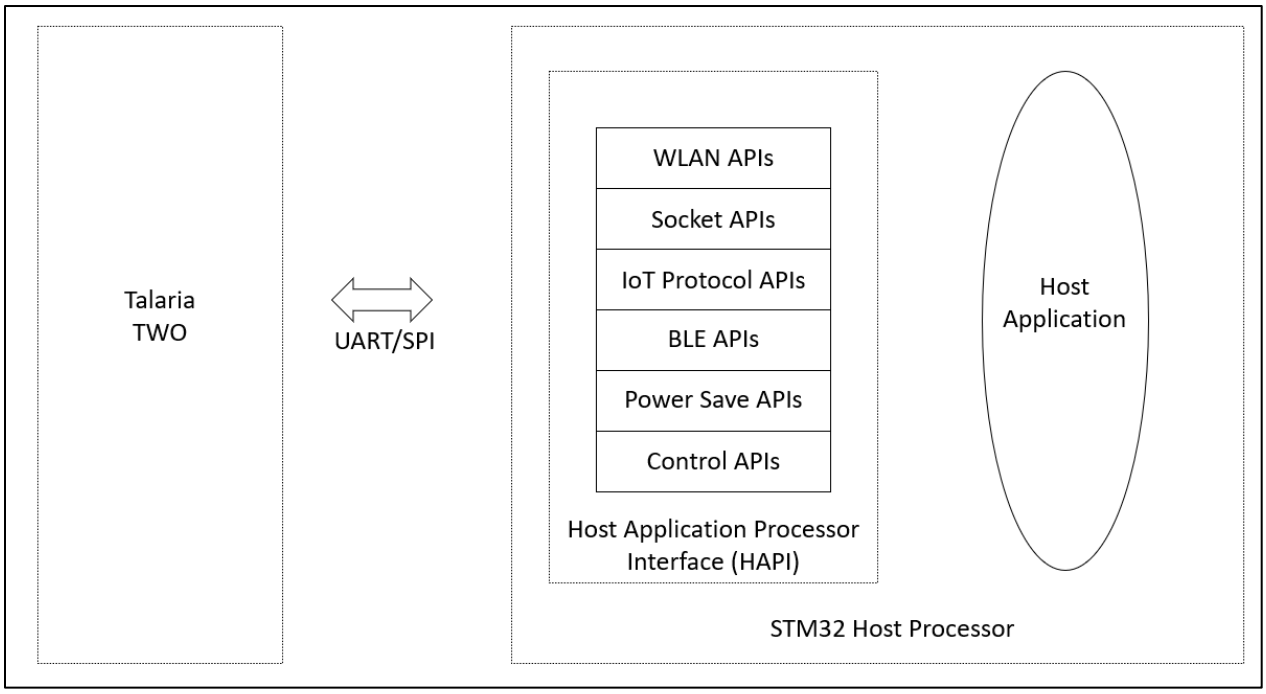

Figure 3: Communication between Host and Talaria TWO via UART/SPI

7 Talaria TWO - Host Processor Interface

Host processor communicates with Talaria TWO via a SPI or UART and follows a protocol to exchange command and data. This protocol is implemented on the host side and are provided as APIs. The host application can then use these APIs to access and control Talaria TWO.

7.1 Talaria TWO - Host APIs (HAPI)

APIs are grouped into:

1. WLAN APIs
2. Socket APIs
3. BLE APIs
4. IoT Protocols
5. Interface Port APIs

Host applications use HAPI WLAN and Socket APIs, which internally use interface port APIs to transfer data between the wireless network and host processor.

7.1.1 Port APIs

These APIs provides basic read/write over the hardware interface (SPI/UART) between the host and Talaria TWO where each API must be defined for each port.

7.1.1.1 hapi_uart_init

API to initialize HAPI UART interface.

```
void hapi_uart_init(struct hapi * hapi, void* hapi_uart_ptr, int
baudrate)
```

Arguments:

1. `hapi`: HAPI instance pointer
2. `hapi_uart_ptr`: pointer to the HAPI UART instance
3. `baudrate`: Baud rate to be configured

Return: None

7.1.1.2 hapi_uart_write

API to write data to Talaria TWO over UART HAPI.

```
ssize_t hapi_uart_write(struct hapi *hapi, void *data, size_t
length)
```

Arguments:

1. `hapi`: HAPI instance pointer
2. `data`: source buffer address
3. `length`: number of bytes to be written

Return: number of bytes >0 on success, -1 on error.

7.1.1.3 hapi_uart_read

API to read data from Talaria TWO over UART HAPI.

```
ssize_t hapi_uart_read(struct hapi *hapi, void *data, size_t
length)
```


Arguments:

1. `hapi`: HAPI instance pointer
2. `data`: source buffer address
3. `length`: number of bytes to be read

Return: >=0 on success. -1 on error.

7.1.1.4 hapi_spi_init

API to initialize HAPI SPI interface.

```
void hapi_spi_init(struct hapi * hapi, void* hapi_spi_ptr)
```

Arguments:

1. `hapi`: HAPI instance pointer
2. `hapi_spi_ptr`: pointer to the HAPI SPI instance

Return: None

7.1.1.5 hapi_spi_write

API to write data to Talaria TWO over SPI HAPI.

```
ssize_t hapi_spi_write(struct hapi *hapi, void *data, size_t
length)
```

Arguments:

1. `hapi`: HAPI instance pointer
2. `data`: source buffer address
3. `length`: number of bytes to be written

Return: number of bytes >0 on success, -1 on error.

7.1.1.6 hapi_spi_read

API to read data from Talaria TWO over SPI HAPI.

```
ssize_t hapi_spi_read(struct hapi *hapi, void *data, size_t
length)
```


Arguments:

1. `hapi`: HAPI instance pointer
2. `data`: source buffer address
3. `length`: number of bytes to be read

Return: >=0 on success. -1 on error.

7.1.2 WLAN APIs

7.1.2.1 hapi_wcm_create

This API creates the HAPI WLAN manager interface and should be called before any WLAN APIs.

```
struct hapi_wcm * hapi_wcm_create(struct hapi *hapi)
```

Arguments:

1. `hapi`: HAPI instance pointer.

Return: a valid pointer points to the HAPI WLAN instance on success. -1 on error.

7.1.2.2 hapi_wcm_connect

This API triggers the scan and connects to the AP specified by the SSID and uses the passphrase for the WPA2 security. This is a blocking API.

```
bool hapi_wcm_connect(struct hapi_wcm *hapi_wcm, const char *ssid,
const char *passphrase)
```

Arguments:

1. `hapi_wcm`: HAPI WLAN instance pointer
2. `SSID`: is the SSID of the AP to scan and connect
3. `Passphrase`: is the password for the WPA2-PSK security

Return: 0(OK) On success. -1(FAIL) on error.

7.1.2.3 hapi_wcm_disconnect

This API triggers the disconnect operation of the AP being associated.

```
hapi_wcm_disconnect(struct hapi_wcm *hapi_wcm)
```

Arguments:

1. `hapi_wcm`: HAPI WLAN instance pointer

Return: 0(OK) on success. -1(FAIL) on error.

7.1.2.4 hapi_wcm_set_link_cb

This API registers the callback function to the HAPI WLAN interface for the asynchronous WLAN link change notification.

```
void hapi_wcm_set_link_cb(struct hapi_wcm *hapi_wcm,

hapi_wcm_link_cb cb, void *context)
```

Arguments:

1. `hapi_wcm`: HAPI WLAN instance pointer
2. `cb`: The call back function to be registered for link change notification
3. `context`: context pointer to be passed when the call back is getting called

Return: none.

7.1.2.5 hapi_wcm_destroy

This API removes the HAPI WLAN manager interface created.

```
bool hapi_wcm_destroy(struct hapi_wcm *hapi_wcm)
```

Arguments:

1. `hapi`: HAPI instance pointer

Return: a valid pointer points to the HAPI WLAN instance on success. -1 on error.

7.1.3 BLE APIs

7.1.3.1 hapi_ble_create

This API creates the HAPI BLE interface and should be called before any BLE APIs.

```
struct hapi_ble *hapi_ble_create(struct hapi *hapi)
```

Arguments:

1. `hapi`: HAPI instance pointer.

Return: a valid pointer points to the HAPI BLE instance on success. -1 on error.

7.1.3.2 hapi_ble_server_create

This API creates the HAPI BLE interface Gatt server instance.

```
bool hapi_ble_server_create(struct hapi_ble *hapi_ble, const char
*name, const char *manufacture_name)
```

Arguments:

1. `hapi_ble`: BLE HAPI instance pointer.
2. `Name`: Name of the BLE device to be created.
3. `manufacture_name`: Manufacture name to be registered when the BLE device get created.

Return: true (0) on success. -1 on error.

7.1.3.3 hapi_ble_service_add

This API registers a std/custom service to the Gatt server created.

```
bool      hapi_ble_service_add(struct      hapi_ble      *hapi_ble,
ble_custom_service_type type, const      char*      type_value,int
permission,int property)
```


Arguments:

1. `hapi_ble`: BLE HAPI instance pointer.
2. `Type`: Type of the ble service to be added, it is of type BLE_custom_service_type
3. `type_value`: The value string of the type to be added.
4. `Permission`: The permission feature of the service type.
5. `Property`: The property of the type.

Return: true (0) on success. -1 on error.

7.1.3.4 hapi_ble_service_remove

This API removes the service registered on the GATT Server.

```
bool hapi_ble_service_remove(struct hapi_ble *hapi_ble,
ble_custom_service_type type)
```

Arguments:

1. `hapi_ble`: BLE HAPI instance pointer.
2. `Type`: the type of the service registered and needs to be removed.

Return: true (0) on success. -1 on error.

7.1.3.5 hapi_ble_start_server

This API starts the GATT server. It internally creates the BLE device as discoverable/connectable and any central device can connect to this.

```
bool hapi_ble_start_server(struct hapi_ble *hapi_ble,bt_address_t
addr)
```

Arguments:

1. `hapi_ble`: BLE HAPI instance pointer.

2. Addr: The hardware (MAC) address of the BLE device to be created.

Return: true (0) on success. -1 on error.

7.1.3.6 hapi_ble_stop_server

This API stops the GATT server started and BLE device become inactive.

```
bool hapi_ble_stop_server(struct hapi_ble *hapi_ble)
```

Arguments:

1. `hapi_ble`: BLE HAPI instance pointer.

Return: true (0) on success. -1 on error.

7.1.3.7 hapi_ble_destroy

This API removes the HAPI BLE interface and deallocates all memory .

```
void hapi_ble_destroy(struct hapi_ble *hapi_ble)
```

Arguments:

1. `hapi_ble`: BLE HAPI instance pointer.

Return: true (0) on success. -1 on error.

7.1.3.8 hapi_ble_prov_info

This API check and return the BLE provision status and data.

```
bool hapi_ble_prov_info(struct hapi_ble *hapi_ble, void *userdata)
```

Arguments:

1. `hapi_ble`: BLE HAPI instance pointer.
2. `Userdata`: The provisioned data to be copied.

Return (0) on success. -1 on error.

7.1.4 Power Save APIs

7.1.4.1 hapi_send_sleep

This API request to enable the sleep in T2.

```
Void hapi_send_sleep(struct hapi *hapi)
```

Arguments:

1. `hapi`: HAPI instance pointer.

Return: None.

7.1.5 Socket APIs

7.1.5.1 socket_create

This API creates a socket according to the parameter passed.

```
int socket_create(struct hapi *hapi, int proto, char *server, char
*port)
```

Arguments:

1. `hapi`: HAPI instance pointer
2. `proto`: specifies the protocol used for the socket to create. The valid combinations are TCP client, UDP client, TCP server and UDP server
3. `server`: The server URL for the TCP or UDP client connection
4. `port`: the port number to connect. If the proto is TCP/UDP server this is the port on which the Talaria TWO waits for connection

Return: integer value > = 0 on success or -1 on failure.

7.1.5.2 socket_send

This API is used to send data on a socket.

```
bool socket_send (struct hapi *hapi, uint32_t socket, const void
*data, size_t len)
```

Arguments:

1. `hapi`: HAPI instance pointer
2. `socket`: The socket ID which has been created
3. `data`: The data to be sent on the socket
4. `len`: The length of the data to be sent

Return: TRUE (0) on success and FALSE (-1) on failure.

7.1.5.3 socket_receive

This API is used to receive data from a socket.


```
size_t socket_receive(struct hapi *hapi, uint32_t socket, void
*data, size_t len)
```

Arguments:

1. `hapi`: HAPI instance pointer
2. `socket`: The socket ID which has been created
3. `data`: The data pointer on which the data is to be received from the socket
4. `len`: The length of the data to be received

Return: the length of the actual data received.

7.1.5.4 socket_getavailable

This API is used to check received data available on a socket.

```
int socket_getavailable(struct hapi *hapi, uint32_t socket)
```

Arguments:

1. `hapi`: HAPI instance pointer
2. `socket`: The socket ID which has been created

Return: The length of the data available on the socket which can be read.

7.1.5.5 socket_close

This API is used to close a socket which has been opened.

```
void socket_close(struct hapi *hapi, uint32_t socket)
```

Arguments:

1. `hapi`: HAPI instance pointer
2. `socket`: The socket ID which has been created

Return: none.

7.1.6 Common APIs

7.1.6.1 hapi_init

This API initializes and starts HAPI on the host micro controller. This should be called before accessing any HAPI API.

```
void* hapi_init(void *console_uart_ptr,  void* hapi_uart_ptr, int
hapi_baudrate, void* hapi_spi_ptr)
```

Arguments:

1. `console_uart_ptr` : debug UART pointer
2. `hapi_uart_ptr` : HAPI interface UART pointer
3. `hapi_baudrate`: UART baudrate
4. `hapi_spi_ptr`: HAPI interface SPI pointer

Return: HAPI instance pointer on success, NULL on failure.

7.1.6.2 hapi_get_error_code

This API returns the currently set error code in HAPI layer.

```
int hapi_get_error_code(struct hapi *hapi)
```

Arguments:

1. `hapi`: HAPI instance pointer

Return: integer value corresponding to the error code.

7.1.6.3 hapi_get_error_message

This API returns the currently set error message in HAPI layer.

```
const char*hapi_get_error_message(struct hapi *hapi)
```

Arguments:

1. `hapi`: HAPI instance pointer

Return: error message in string format corresponding to the error code.

7.1.6.4 set_hapi_encryption_mode

This API sets the encryption enable/disable in serial communication.

```
void set_hapi_encryption_mode(struct hapi *hapi, int enable, void*
enc_ctx, void* key, encryption_fn enc_fn, decryption_fn dec)
```

Arguments:

1. `hapi`: HAPI instance pointer
2. `enable`: 1 to enable the pass or 0 to disable
3. `enc_ctx`: context pointer passed along with encryption/decryption callback function
4. `key`: encryption/decryption key
5. `enc_fn`: encryption callback function
6. `dec_fn`: decryption callback function

Return: none.

7.1.6.5 hapi_add_ind_handler

This API request to add an indication handler for a message in a group.

```
struct hapi_ind_handler * hapi_add_ind_handler(
        struct hapi *hapi,
        uint8_t group_id,
        uint8_t msg_id,
```



```
        hapi_ind_callback ind_cb,
        void * context);
```

Arguments:

1. `hapi`: HAPI instance pointer
2. `group_id`: The group id to which it the handler registered.
3. `msg_id`: The message id to which it the handler registered.
4. `ind_cb`: The callback function to be called.
5. `context`: The context to be passed when the call back is getting called.

Return: The valid pointer on success or -1 on failure.

7.1.6.6 hapi_config

This API Configure the HAPI interface.

```
Void hapi_config(struct hapi *hapi, bool suspend_enable, uint8_t
wakeup_pin, uint8_t wakeup_level, uint8_t irq_pin, uint8_t
irq_mode)
```

Arguments:

1. `hapi`: HAPI instance pointer
2. `suspend_enable`: suspend enabled or not.
3. `wakeup_pin`: The pin used to wake up from suspend
4. `wakeup_level`: The level of the wake pin state.
5. `irq_pin`: The interrupt request pin.
6. `irq_mode: The irq mode to be configured.`

Return: None.

7.1.6.7 set_hapi_default_interface

This API sets the default interface as specified.

```
void set_hapi_default_interface(struct hapi *hapi, int
def_interface)
```

Arguments:

1. `hapi`: HAPI instance pointer
2. `def_interface`: default interface index, 0->uart,1->spi

Return: none.

7.1.6.8 hapi_hio_query

This API checks if the Talaria-2 is ready to accept the HIO commands from the host.

```
hapi_hio_query(struct hapi *hapi)
```

Arguments:

1. `hapi`: HAPI instance pointer

Return: none.

8 Support

1. Sales Support: Contact an InnoPhase sales representative via email – sales@innophaseinc.com
2. Technical Support:
    a. Visit: https://innophaseinc.com/support/
    b. Contact: support@innophaseinc.com

InnoPhase is working diligently to provided outstanding support to all customers.

9 Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, InnoPhase Incorporated does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and assumes no liability associated with the use of such information. InnoPhase Incorporated takes no responsibility for the content in this document if provided by an information source outside of InnoPhase Incorporated.

InnoPhase Incorporated disclaims liability for any indirect, incidental, punitive, special or consequential damages associated with the use of this document, applications and any products associated with information in this document, whether or not such damages are based on tort (including negligence), warranty, including warranty of merchantability, warranty of fitness for a particular purpose, breach of contract or any other legal theory. Further, InnoPhase Incorporated accepts no liability and makes no warranty, express or implied, for any assistance given with respect to any applications described herein or customer product design, or the application or use by any customer's third-party customer(s).

Notwithstanding any damages that a customer might incur for any reason whatsoever, InnoPhase Incorporated' aggregate and cumulative liability for the products described herein shall be limited in accordance with the Terms and Conditions of identified in the commercial sale documentation for such InnoPhase Incorporated products.

Right to make changes — InnoPhase Incorporated reserves the right to make changes to information published in this document, including, without limitation, changes to any specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — InnoPhase Incorporated products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an InnoPhase Incorporated product can reasonably be expected to result in personal injury, death or severe property or environmental damage. InnoPhase Incorporated and its suppliers accept no liability for inclusion and/or use of InnoPhase Incorporated products in such equipment or applications and such inclusion and/or use is at the customer's own risk.

