

libHackRF API

Edit

New Page

Matt Thomas edited this page 23 days ago · 6 revisions

This document describes the functions, data structures and constants that libHackRF provides. It should be used as a reference for using libHackRF and the HackRF hardware.

If you are writing a generic SDR application, i.e. not tied to the HackRF hardware, we strongly recommend that you use either gr-osmosdr or SoapySDR to provide support for the broadest possible range of software defined radio hardware.

For example usage of many of these functions, see the [hackrf_transfer](#) tool.

Setup, Initialization and Shutdown

HackRF Init

Initialize libHackRF, including global libUSB context to support multiple HackRF hardware devices.

Syntax: `int hackrf_init()`

Returns: A value from the `hackrf_error` constants listed below.

HackRF Open

Syntax: `int hackrf_open(hackrf_device** device)`

Returns: A value from the `hackrf_error` constants listed below.

HackRF Device List

Retrieve a list of HackRF devices attached to the system. This function finds all devices, regardless of permissions or availability of the hardware.

Syntax: `hackrf_device_list_t* hackrf_device_list()`

Returns: A pointer to a `hackrf_device_list_t` struct, a list of HackRF devices attached to the system. The contents of the `hackrf_device_list_t` struct are described in the data structures section below.

HackRF Device List Open

Open and acquire a handle on a device from the

Syntax: `int hackrf_device_list_open(hackrf_device_list_t* list, int idx, hackrf_device** device)`

▼ Pages 25

Find a Page...

[Home](#)

[Clocking](#)

[Design Goals](#)

[FAQ](#)

[Firmware Development Setup](#)

[Future Hardware Modifications](#)

[Getting Help](#)

[Getting Started with HackRF and GNU Radio](#)

[HackRF Beta Program](#)

[HackRF Hacks](#)

[HackRF One](#)

[Hardware Components](#)


[Jawbreaker](#)


[Lemondrop Bring Up](#)

[libHackRF API](#)

Show 10 more pages...

Clone this wiki locally

<https://github.com/mossmann> 

 Clone in Desktop

Params:

`list` - A pointer to a `hackrf_device_list_t` returned by `hackrf_device_list()`

`idx` - The list index of the HackRF device to open

`device` - Output location for `hackrf_device` pointer. Only valid when return value is `HACKRF_SUCCESS`.

Returns: A value from the `hackrf_error` constants listed below.

HackRF Device List Free

Syntax: `void hackrf_device_list_free(hackrf_device_list_t* list)`

Params:

`list` - A pointer to a `hackrf_device_list_t` returned by `hackrf_device_list()`

HackRF Open By Serial

Syntax: `int hackrf_open_by_serial(const char* const desired_serial_number, hackrf_device** device)`

Returns:

HackRF Close

Syntax: `int hackrf_close(hackrf_device* device)`

Returns: A value from the `hackrf_error` constants listed below.

HackRF Exit

Cleanly shutdown libHackRF and the underlying USB context. This does not stop in progress transfers or close the HackRF hardware. `hackrf_close()` should be called before this to cleanly close the connection to the hardware.

Syntax: `int hackrf_exit()`

Returns: A value from the `hackrf_error` constants listed below.

Using the Radio

HackRF Start Rx

Syntax: `int hackrf_start_rx(hackrf_device*, hackrf_sample_block_cb_fn, void* rx_ctx)`

Params:

Returns: A value from the `hackrf_error` constants listed below.

HackRF Stop Rx

Syntax: `int hackrf_stop_rx(hackrf_device*)`

Params:

Returns: A value from the `hackrf_error` constants listed below.

HackRF Start Tx

Syntax: `int hackrf_start_tx(hackrf_device*, hackrf_sample_block_cb_fn, void* tx_ctx)`

Params:

Returns: A value from the `hackrf_error` constants listed below.

HackRF Stop Tx

Syntax: `int hackrf_stop_tx(hackrf_device*)`

Params:

Returns: A value from the `hackrf_error` constants listed below.

HackRF Set Baseband Filter Bandwidth

Syntax: `int hackrf_set_baseband_filter_bandwidth(hackrf_device*, const uint32_t bandwidth_hz)`

Params:

Returns: A value from the `hackrf_error` constants listed below.

HackRF Compute Baseband Filter BW

Compute best default value depending on sample rate (auto filter)

Syntax: `uint32_t hackrf_compute_baseband_filter_bw(const uint32_t bandwidth_hz)`

Params:

Returns: A valid baseband filter width available from the Maxim max2837 frontend used by the radio.

HackRF Compute Baseband Filter BW Round Down LT

Compute nearest freq for bw filter (manual filter)

Syntax: `uint32_t hackrf_compute_baseband_filter_bw_round_down_lt(const uint32_t bandwidth_hz)`

Params:

Returns: A valid baseband filter width available from the Maxim max2837 frontend used by the radio.

HackRF Set LNA Gain

range 0-40 step 8d, IF gain in osmosdr

Syntax: `int hackrf_set_lna_gain(hackrf_device* device, uint32_t value)`

Params:

Returns:

HackRF Set VGA Gain

range 0-62 step 2db, BB gain in osmosdr

Syntax: int hackrf_set_vga_gain(hackrf_device* device, uint32_t value)

Params:

Returns:

HackRF Set Tx VGA Gain

range 0-47 step 1db

Syntax: int hackrf_set_txvga_gain(hackrf_device* device, uint32_t value)

Params:

Returns:

HackRF Set Antenna Enable

antenna port power control **Syntax:** int hackrf_set_antenna_enable(hackrf_device* device, const uint8_t value)

Params:

Returns:

HackRF Set Freq

Syntax: int hackrf_set_freq(hackrf_device* device, const uint64_t freq_hz)

Params:

Returns:

HackRF Set Freq Explicit

Syntax: int hackrf_set_freq_explicit(hackrf_device* device, const uint64_t if_freq_hz, const uint64_t lo_freq_hz, const enum rf_path_filter path)

Params:

Returns:

HackRF Set Sample Rate

Syntax: int hackrf_set_sample_rate(hackrf_device* device, const double freq_hz)

Params:

Returns:

HackRF Set Sample Rate Manual

currently 8-20Mhz - either as a fraction, i.e. freq 20000000hz divider 2 -> 10Mhz or as plain old 10000000hz (double) preferred rates are 8, 10, 12.5, 16, 20Mhz due to less jitter

Syntax: `int hackrf_set_sample_rate_manual(hackrf_device* device, const uint32_t freq_hz, const uint32_t divider)`

Params:

Returns:

HackRF Set Amp Enable

Toggle the antenna port power for external amplifiers.

Syntax: `int hackrf_set_amp_enable(hackrf_device* device, const uint8_t value)`

Params:

`device` - A pointer to the `hackrf_device` handle

`value` - 1 to enable port power, 0 to disable

Returns:

HackRF Is Streaming

Check whether or not the HackRF device is currently streaming samples, either to or from the host system.

Syntax: `int hackrf_is_streaming(hackrf_device* device)`

Params:

`device` - A pointer to the `hackrf_device` handle

Returns: HACKRF_TRUE if the device is currently streaming.

Reading and Writing Registers

HackRF MAX2837 Read

Read register values from the MAX2837 Baseband IC.

Syntax: `int hackrf_max2837_read(hackrf_device* device, uint8_t register_number, uint16_t* value)`

Params:

Returns:

HackRF MAX2837 Write

Write register values to the MAX2837 Baseband IC.

Syntax: `int hackrf_max2837_write(hackrf_device* device, uint8_t register_number, uint16_t`

value)

Params:

Returns:

HackRF Si5351C Read

Read register values from the Si5351C clock generator IC.

Syntax: int hackrf_si5351c_read(hackrf_device* device, uint16_t register_number, uint16_t* value)

Params:

Returns:

HackRF Si5351C Write

Write register values to the Si5351C clock generator IC.

Syntax: int hackrf_si5351c_write(hackrf_device* device, uint16_t register_number, uint16_t value)

Params:

Returns:

HackRF RFFC5071 Read

Read register values from the RFFC5071 mixer IC.

Syntax: int hackrf_rffc5071_read(hackrf_device* device, uint8_t register_number, uint16_t* value)

Params:

Returns:

HackRF RFFC5071 Write

Write register values to the RFFC5071 mixer IC.

Syntax: int hackrf_rffc5071_write(hackrf_device* device, uint8_t register_number, uint16_t value)

Params:

Returns:

Updating Firmware

HAckRF CPLD Write

device will need to be reset after hackrf_cpld_write

Syntax: int hackrf_cpld_write(hackrf_device* device, unsigned char* const data, const unsigned int total_length)

Params:

Returns:

HackRF SPI Flash Erase

Syntax: int hackrf_spiflash_erase(hackrf_device* device)

Params:

Returns:

HackRF SPI Flash Write

Syntax: int hackrf_spiflash_write(hackrf_device* device, const uint32_t address, const uint16_t length, unsigned char* const data)

Params:

Returns:

HackRF Spi Flash Read

Syntax: int hackrf_spiflash_read(hackrf_device* device, const uint32_t address, const uint16_t length, unsigned char* data)

Params:

Returns:

Board Identifiers

HackRF Board ID Read

Syntax: int hackrf_board_id_read(hackrf_device* device, uint8_t* value)

Params:

Returns:

HackRF Version String Read

Syntax: int hackrf_version_string_read(hackrf_device* device, char* version, uint8_t length)

Params:

Returns:

HackRF Board Part ID Serial Number Read

Syntax: int hackrf_board_partid_serialno_read(hackrf_device* device, read_partid_serialno_t*

```
read_partid_serialno)
```

Params:

Returns:

Miscellaneous

HackRF Error Name

Syntax: `const char* hackrf_error_name(enum hackrf_error errcode)`

Params:

Returns:

HackRF Board ID Name

Syntax: `const char* hackrf_board_id_name(enum hackrf_board_id board_id)`

Params:

Returns:

HackRF USB Board ID Name

Syntax: `const char* hackrf_usb_board_id_name(enum hackrf_usb_board_id usb_board_id)`

Params:

Returns:

HackRF Filter Path Name

Syntax: `const char* hackrf_filter_path_name(const enum rf_path_filter path)`

Params:

Returns:

Data Structures

```
typedef struct hackrf_device hackrf_device
```

```
typedef struct {
    hackrf_device* device;
    uint8_t* buffer;
    int buffer_length;
    int valid_length;
    void* rx_ctx;
    void* tx_ctx;
} hackrf_transfer;
```

```
typedef struct {
    uint32_t part_id[2];
```



```
uint32_t serial_no[4];  
} read_partid_serialno_t;
```

```
typedef struct {  
    char **serial_numbers;  
    enum hackrf_usb_board_id *usb_board_ids;  
    int *usb_device_index;  
    int devicecount;  
  
    void **usb_devices;  
    int usb_devicecount;  
} hackrf_device_list_t;
```

```
typedef int (*hackrf_sample_block_cb_fn)(hackrf_transfer* transfer)
```

Enumerations

Supported board versions

These values identify the board type of the connected hardware. This value can be used as an indicator of capabilities, such as frequency range, bandwidth or antenna port power.

Board	Frequency range	Bandwidth	Antenna port power
HackRF One	1MHz - 6Ghz	20MHz	Yes
Jawbreaker	10MHz - 6GHz	20MHz	No
Rad1o	50MHz - 4GHz	20MHz	Unknown
Jellybean	N/A	20MHz	No

Most boards will identify as HackRF One, Jawbreaker or Rad1o. Jellybean was a pre-production revision of HackRF. No hardware device should intentionally report itself with an invalid board ID.

```
enum hackrf_board_id {  
    BOARD_ID_JELLYBEAN = 0,  
    BOARD_ID_JAWBREAKER = 1,  
    BOARD_ID_HACKRF_ONE = 2,  
    BOARD_ID_RAD10 = 3,  
    BOARD_ID_INVALID = 0xFF,  
};
```

USB Product IDs

```
enum hackrf_usb_board_id {  
    USB_BOARD_ID_JAWBREAKER = 0x604B,  
    USB_BOARD_ID_HACKRF_ONE = 0x6089,  
    USB_BOARD_ID_RAD10 = 0xCC15,  
    USB_BOARD_ID_INVALID = 0xFFFF,  
};
```

Transceiver Mode

HackRF can operate in three main transceiver modes, Receive, Transmit and Signal Source.

There is also a CPLD update mode which is used to write firmware images to the CPLD.

The transceiver mode can be changed with `hackrf_set_transceiver_mode` with the value parameter set to one of the following:

```
enum transceiver_mode_t {
    TRANSCEIVER_MODE_OFF = 0,
    TRANSCEIVER_MODE_RX = 1,
    TRANSCEIVER_MODE_TX = 2,
    TRANSCEIVER_MODE_SS = 3,
    TRANSCEIVER_MODE_CPLD_UPDATE = 4
};
```

Receive mode (`TRANSCEIVER_MODE_RX`) is used to stream samples from the radio to the host system. Use `hackrf_set_freq` to set the center frequency of receiver and `hackrf_set_sample_rate` to set the sample rate (effective bandwidth).

Transmit mode (`TRANSCEIVER_MODE_TX`) is used to stream samples from the host to the radio.

See [hackrf_transfer](#) for an example of setting transmit and receive mode and transferring data over USB.

Function return values

```
enum hackrf_error {
    HACKRF_SUCCESS = 0,
    HACKRF_TRUE = 1,
    HACKRF_ERROR_INVALID_PARAM = -2,
    HACKRF_ERROR_NOT_FOUND = -5,
    HACKRF_ERROR_BUSY = -6,
    HACKRF_ERROR_NO_MEM = -11,
    HACKRF_ERROR_LIBUSB = -1000,
    HACKRF_ERROR_THREAD = -1001,
    HACKRF_ERROR_STREAMING_THREAD_ERR = -1002,
    HACKRF_ERROR_STREAMING_STOPPED = -1003,
    HACKRF_ERROR_STREAMING_EXIT_CALLED = -1004,
    HACKRF_ERROR_OTHER = -9999,
};
```

RF Filter Path

```
enum rf_path_filter {
    RF_PATH_FILTER_BYPASS = 0,
    RF_PATH_FILTER_LOW_PASS = 1,
    RF_PATH_FILTER_HIGH_PASS = 2,
};
```

